

A FULL TURN OF THE SCREW

The Complete Collection of
"Turn of The Screw" articles from
Rainbow Magazine, Jan '83 to Jul '89

By

Tony DiStefano

A FULL TURN OF THE SCREW

The Complete Collection of
"Turn of The Screw" articles from
Rainbow Magazine, Jan '83 to Jul '89

By

Tony DiStefano

PUBLISHED BY

Tony DiStefano
3489 Marian
Laval, Quebec
Canada H7P 5K8

Copyright© 1983,1984,1985
1986,1987,1988,1989
by Falsoft, Inc.

All rights reserved. No part of the contents of this book
may be reproduced or transmitted in any form
or by any means without permission.

This book includes material Copyright© 1983-1989 by
Falsoft, Inc., The Falsoft Building, P.O. Box 385,
Prospect, KY USA, 40059 (502) 228-4492,
publishers of "The Rainbow"® and used with permission.

Disclaimer

The author has exercised due care in the preparation of
this book and the programs contained in it. Neither the
author nor the publisher make any warranties either
expressed or implied with regard to the information and
programs contained in this book. In no event shall the
author or publisher be liable for any damages caused by
this book.

The Color Computer is a registered trademark of the Tandy
Corp. The Rainbow is a resigtered trademark of Falsoft Inc.

First edition published in September 1989

Table of Contents

Article	Date	Page
Poke Speedup	01/83	01
Finger Saving Circuit	02/83	01
Green On black Video	03/83	03
Reduce RFI on Your Monitor	04/83	04
Close Look at Memory	06/83	05
Make ROM Port "Y" Adapter	07/83	07
Parallel Printer Interface	08/83	08
Interfacing the GI Sound Generator	09/83	10
Project Odds & Ends	10/83	14
Build a Speaker/Amplifier	11/83	15
Installing a ROM Switcher	12/83	16
Upgrading Guide for the COCO 2	01/84	18
Program Pak Connection	02/84	19
Add Function Keys to Keyboard	03/84	21
A 12 Volt Power Supply for COCO 2	04/84	23
Design A Video Monitor Output	05/84	24
Phoneme Speech Synthesizer	06/84	25
Duelling Cassettes	07/84	27
Popular Misconceptions & Common Problems	08/84	29
The Halt Pin and it's Function	09/84	30
The Modem to Printer Connection	10/84	32
Force a Cold Start from Reset Project	11/84	33
Lights! Camera! COCO!	12/84	35
Intro to the inside of the COCO 2	01/85	37
How the Multi-Pak Interface Works	02/85	39
Construct 16K of EPROM for Controller	03/85	42
Add Numeric Keypad to your COCO	04/85	45
Hookup a Voice Synthesizer from RS	05/85	48
Follow a Memory Map	06/85	50
Look Ma, No switch	07/85	53
Switching Double Sided Drives	08/85	55
COCO! with more LEDs	09/85	58
Analog to Digital Converter Pt1	10/85	60
Analog to Digital Converter Pt2	11/85	62
What is a VDG?	12/85	64
Beginners Hardware Course Pt1	01/86	67
Beginners Hardware Course Pt2	02/86	69
Introduction to Timing	03/86	72
Memory, How it works	04/86	75
Exploring Memory Cells	05/86	79
The CPU	06/86	82
The PIA	07/86	84
Timing & the SAM Chip	08/86	87
The New Video Display Generator	09/86	89

Table of Contents

Article	Date	Page
The COCO 2B	10/86	91
Hardware Fixes, Video Display Generator	11/86	93
The No Switch VDG	12/86	95
How Monitors Work	01/87	97
COCO! Music to your Ears	02/87	99
Buffers	04/87	101
Hardware Projects Review	05/87	103
Expandable Relay Project	06/87	105
Cache of the Day	07/87	107
Uses for Memory	08/87	110
Build an EPROM emulator	09/87	112
Disk Controllers	10/87	113
Improved Printer Adapter	11/87	116
Finishing the Printer Adapter	12/87	118
Add a LED to your Controller	01/88	120
Build an Electronic EPROM Emulator Eraser	02/88	122
Bigger & Better EPROMs	03/88	124
Build a Megabyte ROM Disk	04/88	126
Multi-Pak LED Update	05/88	128
Increase Character Display	06/88	130
Project Adapter "2 for 1"	07/88	132
Serial Paks	08/88	134
Summer Cleanup	09/88	136
A Simple Expandable LED Project	11/88	138
Project Expansion	12/88	141
Do You Read Me?	01/89	143
Lights Out!	04/89	145
The ABCs of a Disk Drive	05/89	147
The DEFs of a Disk Drive	06/89	148
Dynamic RAM Explained	07/89	150

Introduction

I remember, on a hot summer night, back in 1981, I was browsing around on CompuServe. I came upon a notice on the Tandy forum for a free copy of a newsletter. I ordered it. It was just about the only information I could find back then, for my newly acquired COLOR COMPUTER. A few days later I received it. A two page thing, stapled together, and printed on an EPSON printer. It was called THE RAINBOW. I subscribed to it. It grew and I absorbed every bit of it. As time passed, I learned more about my COLOR COMPUTER. I did things to it that no one else thought of doing. My friends wanted the same. The more I did, the more they wanted. By then, Rainbow had grown to a full blown magazine. I felt I had gained enough knowledge about the COCO to be able to write for Rainbow. I called them up and asked. I spoke to a gentleman called Lonnie. He gave me the go ahead. It was that simple. A lot of people ask me how I got started. That was it, I called!

Shortly after I started writing, I met Christian Rochon. He bought and sold COCOs and add-ons. He needed a controller to add to his drive kits. This was how DISTO was born. He said if I was able to design a controller, he was able to market it. As you know, it worked out quite well. Disto now has many products for your COCO, and CRC is still marketing them.

Through the years, many people have asked me if I had a copy of this article or that article. That gave me an idea. Why not make a collection of all my articles into a book. This way, anyone who wanted, would have access to any article for research or reference purposes. This is what this book is all about.

From the first day I brought home my COCO until today, I have made a lot of friends. Too many to mention them all, but here are the most important to me. First, I would like to thank Larry Callahan for getting me interested in computers, and always being there for me when I needed someone. Next, I want to thank Christian and his wife Johanne for the endless patience they had dealing with me and DISTO products. Then comes Kevin Darling, who has given me his full support whenever OS9 was involved. Without him, the SC-II would not have succeeded. Finally, I want to thank Lonnie Falk for making all this possible.

I hope you enjoy this book as much as I did writing it.

High Speed *POKE* Has Effect On CoCo Hardware

By Tony DiStefano
Rainbow Hardware Columnist

This is the first of a series of articles that will deal with the 'insides' of your Color Computer. Every month I will explore and explain different parts of the Radio Shack Color Computer hardware; its limitations, what it can do, what it cannot do, and how to improve it. In general, just digging into your computer and learning about the hardware that all that great software runs on.

In my first article I would like to clear up a controversy that has cropped up concerning the so-called *high speed* computer. As most people know, *POKE 65495,0 speeds up Basic programs* by about 65%. But why does it work on some computers and not on others? Also, why does it not work with most disk systems? There is also the *POKE 65497,0*. That seems to do some strange things on the screen. What does that do and why? Well, Here's the story!

All timings in the computer are derived from a 14.31818 mhz crystal. This frequency is the clock input to the 6883 (SAM) chip. When you power up, the Basic power up routine sets the SAM to divide the crystal frequency by 16, making the 6809E (MPU) frequency of .894 mhz clock rate. A write to \$FFD7 ("S" denotes hex number) or *POKE 65495,0* sets the SAM into what is known as the A.D. (Address Dependent) rate. This means that the MPU will work at one of two speeds .894 mhz or 1.788 mhz clock rates. This rate depends upon where the MPU is addressing. That's right! The SAM will switch between fast and slow clock rates depending where it is addressing memory. If the MPU is addressing memory between locations 0 and \$7FFF (reading or writing) it will run at the slow clock rate. This area is usually RAM. That is 32K of RAM. When it addresses memory between \$8000 and \$FEFF it will run at the fast clock rate. This area of memory is usually occupied by Extended Basic, Basic, and DOS ROMS. I say usually because in another SAM mode this area could be RAM also. In the I/O (input/output) area, any addressing done between \$FF00 and \$FFF1F is at slow clock speed. The rest of the I/O area between \$FF20 and \$FFFF is at fast speed. This means that only one of the PIAs go to high speed, and not both, like many people think. The PIA that does go to high speed is the one that does the D to A conversions and the VDG controls.

What does all this mean to you? Well, you can use this information to find out why your computer doesn't work at the dual or high speed. We'll start with the easiest and least expensive ways. First, if you have a disk drive, disconnect it and try to get the computer to work without it. If the high speed doesn't work without the drive plugged in you will have to open the computer. Turn the computer off before opening it. (P.S. Refer to your service manual for instructions before you attempt to open your computer. Oh! By the way, you may void your warranty by opening up the computer.) Now, remove the RF shield and locate the two capacitors labeled C73 and C75. These two capacitors along with resistors R73 and R74 make up a RF suppression circuit in the main clock circuit. This, unfortunately, distorts the square wave shape of the E and Q clock signals. This may prevent the system from working at the higher speed. OK, now make sure the computer is off, and remember to make sure it's off before you do any modification.

Cut one side of both of these capacitors. Why only one side? Because you may want to resolder it if it has no effect on the high speed after you cut it. After all, it is a part of the

RF suppression circuit. Turn the computer on and try the high speed. If it works, great; if not, you will have to continue. The next step is to check the PIAs. Since only one of the PIAs goes to high speed, the D to A and the VDG control one, try changing the PIAs around. The chance that both will not work at high speed is rare. If the other one works then you are on your way. If not, well, you will have to go one step further. At this point you may have to change some ICs. If you can, borrow rather than buy one two-mhz PIA (MC 68B21) and one two-mhz MPU (MC 68B09E) IC, because if after you have changed these two parts you may still be out of luck. Replace the MPU and the PIA with faster ones. Now it should work at the higher speed. If not, the only other components that you can change then are the Basic and Extended Basic ICs themselves.

With your computer working at high speeds it's time to try it with your Disk drive. What! It still doesn't work? Don't despair; I have another trick up my sleeve. There is one more capacitor to cut, it is labeled C85. This capacitor has the same purpose as the other two—RF suppression. Try the high speed with the Disk controller in. WOW! It works. But, if it should happen that your computer still doesn't work, the DOS ROM may not be fast enough.

Chances are your system will now work at the higher speed. If you still have problems after cutting these three capacitors and changing the PIA and MPU (which is very unlikely), there is not much more that you can do. Now that *POKE 65495,0* works, what about *POKE 65497,0*? This is a mode in which the SAM will run at the high speed throughout the whole 64K of memory. Everything is in high speed, ROM, RAM, and all I/O. The reason that the screen goes haywire is that at that speed the SAM chip does not have time to latch in the video, therefore the garbage on the screen. But if your RAM is fast enough, the computer can still work, even though the display makes you think otherwise. Here is a short test program to see if yours will work:

```
10 POKE 65497,0
20 FOR I = 1 to 500: NEXT I
30 POKE 65496,0
```

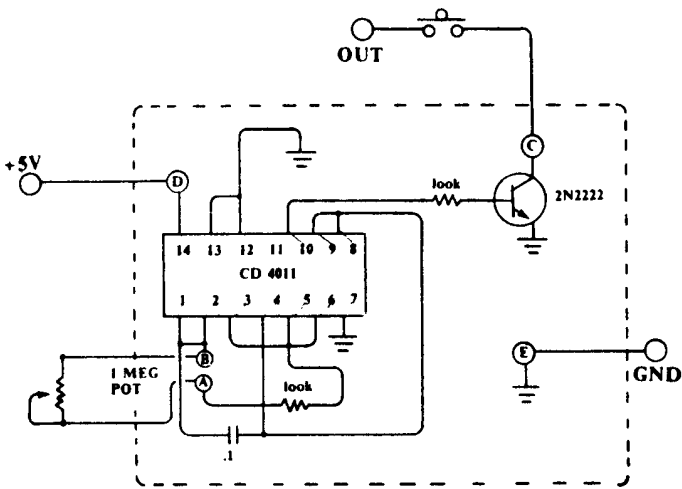
If this program comes back with OK on the screen, then your computer works at the DOUBLE SPEED. You can use this mode whenever there are a lot of calculations to do and there is no need for the video screen. Sound and keyboard functions should work OK, but do not try to do I/O in this mode. If it does not work, but works at the High Speed, then all you need is faster RAM. If you are using 4116 or 4164 chips make sure they are 150 nano seconds or faster. Well, that is all for now. Good luck with high speed. I'll see you next month. ☺

Construct A Finger-Saving Circuit For Your Joystick

By Tony DiStefano

Have you ever played a game on your computer so long that your 'fire-button' finger got sore? Well I did, and that's what prompted me to do something about it. I was at a friend's place the other day and he showed me his new game. It was a great game, but when you pressed the fire button only one shot came out of the "space cannon." It didn't fire rapidly like a machine gun. Every time you wanted to fire again you had to let the button go and press it again. After an hour of playing, you can bet my finger was numb. Then I

thought, if I could make an auto-fire button on my joystick, things would go a lot easier on my poor ol' finger. So, I set out to do just that. After a little drilling, and cutting and soldering, I came out with a circuit that I call my "Finger Saving Rapid Fire Circuit!" It also has speed adjust. Here's the circuit.



This isn't a very complicated project, but it does require a little experience in project building. The first thing you will need is a joystick. Any joystick will do, but since this is a Color Computer I modified a Radio Shack joystick. The next item on the agenda is a parts list. Again, I used Radio Shack parts in this project because there's a Radio Shack store right around the corner from where I live. It's a lot easier than going all the way downtown. If you're a hardware hacker like me, you'll probably have all the parts in your junk box and won't have to buy any of these parts. Here is the list.

PARTS LIST

Quantity	Description	RS Part #
1	IC CD4011	# 276-2411
1	Button	# 275-8080
1	Potentiometer	# 271-1722
1	Transistor 2N2222	# 276-1617
2	Resistors 100k 1/4w	# 271-1347
1	Capacitor .1uf 50v	# 272-0135
1	14 Pin Socket	# 276-1999
1	Small Perfboard	# 276-1392

Now that you have all the parts, it's time to put it together. First, you must mount the pot (potentiometer) and the button. Open the joystick by removing the big screws on the bottom of the joystick. In the case of the more recent sticks there will be only two screws. Remove the lid. You will need a drill and two bits, a 1/2 inch bit and a 5/16 inch bit. Now you must drill two holes in the front of the joystick; that is, one on each side of the fire button. Don't drill into the lid, but rather in the same part the joystick assembly is mounted on. If you look at the front (looking at the button) with the stick pointing upwards, the pot mounts on the right side and the button mounts on the left. I did it that way because the button doesn't fit on the other side. The button hole size is 1/2 inch, while the pot hole size is 5/16 inch. Use the 1/2 inch bit to make the button hole and the 5/16 inch bit to make the pot hole. A pilot hole, using a 1/8 inch bit, is

better, but not necessary. Be careful when you drill into the plastic, there are wires on the other side and you don't want to break them. Mount the pot and the button with the hardware supplied. Tighten them well so that they won't come loose in the middle of a fierce battle. You may want to seal each nut with a little dab of nail polish. Ok, let's put that aside for a while and start on the circuit board.

Cut the perfboard into a piece about 1 1/4 inches by 3/4 inches. This should be just big enough to mount all the parts, yet be small enough to fit inside the joystick. Insert the socket in the center of the board and to one side. The long side of the socket should align with the long side of the board. Note that pin #1 on the socket should match with pin #1 of the chip. Pin #1 is the bottom left hand corner of the socket—the side with the notch. Also note that the pin numbers go counter clockwise around the chip. All pin numbers are looking down on top of the chip and are reversed when soldering underneath the board. Solder in the rest of the components (except the button and the pot) according to the schematic drawing. Do not solder anything to the points marked with letters just now, I will get to that later. Use the long leads of the components as connecting wires to the socket. Do not solder onto the chip itself; use the socket and make sure that the chip is not in the socket when you solder. In fact, you should not insert the chip until all the wiring is done and you are ready to test the circuit. This is a CMOS chip and is very sensitive to static electricity.

Now that all the components are in, it's time to solder wires to connect to the rest of the circuit. There are five wires coming off the board labeled A to E. Each has its special place, and I will describe them one by one...

- A) Wire A goes to the center terminal and one side of the pot that is mounted on the joystick.
- B) Wire B goes to the other side of the pot.
- C) Wire C goes to one side of the button which we mounted earlier.
- D) Wire D goes to the 5 volt supply. On my joystick it's the white wire that comes from the main cable. This may not be the same on all joysticks, so it is best that you trace it from the connector. This is pin #5 on the connector.
- E) Wire E goes to the ground of the joystick. This wire is black on my joystick, but again it may be different on yours. This is pin #3 on the connector.

There is only one more wire to add. This is the point marked "out" on the schematic. One end of the wire comes from the free end of the button you mounted. The other end of this wire goes to the already existing button. There are two wires on the existing button. The one you want is the one that comes from the connector side, not the side that goes to the joystick pot—that's ground. Solder your wire to the same spot, but make sure that the original wire does not come loose when you do. This completes the wiring.

Carefully insert the chip into its socket. Make sure that pin #1 is in the right place. You are now ready to try out the circuit. With the board not touching anything (off to one side) plug in the joystick and turn on the computer. Type in this program and run it...

10 CLS
20 PRINT@0, PEEK (55280) : GOTO 20

A number should appear on the screen. Press the regular fire button. The number should change. It doesn't matter what the number is, just that it changes. Now press the rapid fire button. The value you see on your screen should change rapidly. Turn the pot on the joystick from one end to the other slowly. This is your speed control. You should adjust it according to your needs. The number should change from slow to fast. If it does, then all is well and it's time to close up the joystick. If it doesn't, then check your work carefully and correct the errors. The most common is the pinout of the transistor. Make sure that the base and collector are in their right places. Before you close up the joystick, put a little dab of rubber cement or silicon glue to hold down the board to the inside of the joystick. Anywhere that fits will do. Make sure that it doesn't touch anything. Close it all up and have fun.

That's about it for this month. I hope my "Finger Saving Rapid Fire Circuit" can save a few fingers.



Green On Black Video: 'Eye Friendly' Conversion

By Tony DiStefano

It's two o'clock in the morning and you are typing away on your TRS-80 Color Computer. Your eyes are burning because you've been staring at that bright green screen trying to create your "Do everything program" for hours. So you turn down the color, contrast and brightness of the display but that doesn't help too much. It's still a big square of light. Well...what can you do? Follow these simple instructions and when you are finished you will have a reversed screen like mine.

Though these instructions are simple, only those with soldering experience should attempt this project. You will need a Phillips screwdriver, a grounded soldering iron, solder, an IC extractor, two pieces of thin wire, flux cleaner, and a little patience. And, if you haven't received the warning before, opening your computer may void your warranty.

Before you start tearing into your computer, a bit of background on the VDG (Video Display Generator) is in order. The VDG is a large scale integrated circuit (LSI) chip that takes care of all the video you see on the screen, be it Alphanumeric or full graphic. The VDG continually scans memory (Via the SAM) and displays what it sees. In the Alphanumeric mode it converts the ASCII code of a byte of memory into a graphic block that looks like the letter it represents. Normally an upper case letter or number is black with a green background. Lower case letters are the opposite, green with a black background. What my circuit modification does is reverse the order so that upper case letters are green with black background and lower case letters are black with green background. This does not change anything in memory nor does it interfere with BASIC. It also does not change any graphic modes or color. Everything stays the same except the letters, numbers and

symbols. The diagrams in this article pertain to most versions of the computer. Version 1.1, 1.0, 4K, 16K, 32K, 64K, BASIC, Extended BASIC, and even Disk BASIC are OK. The only version of which I cannot say "it works" is the newest version, the one with the power supply in the bottom left hand corner. It should be the version "F" but it is not written anywhere on the board.

Before you start into this, make sure that you have a large clean work space. Make sure the computer is not plugged in.

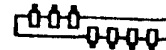


FIG-1

Put the computer upside down on a soft surface. Unscrew the seven screws that hold the cover on. If you haven't opened it before, the seventh screw is under the black sticker that warns you not to open this thing. Turn the unit over again (top side up) and pick up all the screws that fall out. Put them aside in a safe place. Remove the top cover and put that away, too. Lean forward slightly and gently pull up on the keyboard. Unplug the connector that ties the keyboard to the main board. Put the keyboard aside. Now cut the two tie wraps that hold the RF shield in place. That's the big square piece of metal with holes in it. Remove the RF shield and put it with the other parts. You are now ready for part two.

Before you start part two, let me tell you that the board is very sensitive to static electricity. Try to avoid dry areas and avoid touching the contacts on the board whenever possible. OK, let's get going. Locate and pull out the 74LS02 IC marked U29 on the PC board. Carefully bend pins 1, 2 and 3 so they stand straight up in the air upside down. The dot denotes pin #1. If you are facing the computer it is the top left hand corner. See Figure 1. Now solder one end of a four-inch piece of wire to pin #1 of the IC. The best wire to use is a #28 or #30 Wire Wrap wire. Solder another piece of six-inch wire to pins 2 and 3. Yes, both pins together. Now put the IC back in the socket. Make sure it is in the right orientation. The dot should be in the upper left hand corner. Also make sure that the wires and the pins do not touch the side of the RF shield. Now carefully remove the MC6847 IC marked U7 on the PC board. Bend pin #32 outwards just enough so that when you replace it, it does not go into the socket. Replace the MC6847. Again, make sure of the orientation. The dot should be in the upper right hand corner. Take the other end of wire that connects to pin 1 of the 74LS02 and solder it to pin 32 of the MC6847. Take the other end of wire that connects to pins 2 and 3 of the 74LS02 and solder that to pin 2 on the MC6847. Be careful not to solder the pin to the socket. You won't be able to get the IC out if you do. Check the wiring and make sure that there are no shorts. Your wiring should look like the wiring in Figure 2. Now turn the power on. You should see the normal SIGN ON and copyright notice. Adjust the contrast, brightness and color on your TV so that you get crisp green letters with no background shading.

The closing up of the unit is the same as the opening, but in reverse. Turn the power off and replace the RF shield.

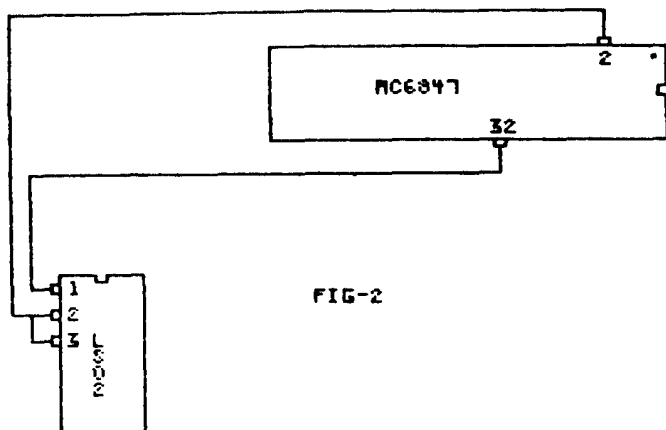


FIG-2

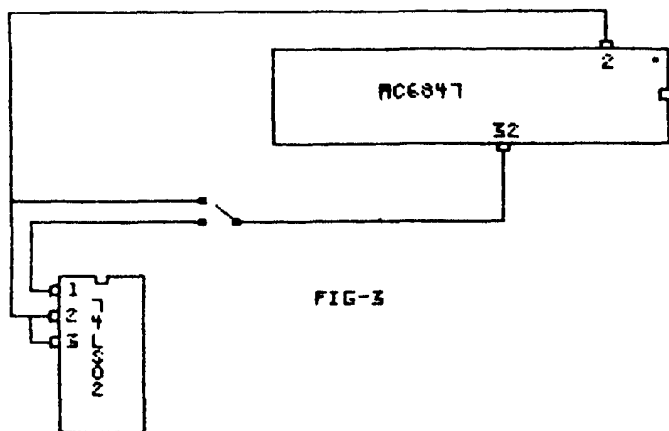


FIG-3

Again, make sure that there are no wires hanging out, and that there are no small pieces of wire or solder left in the closed area. Replace the keyboard and connector and put the lid back on. One thing to note when you are putting the bottom screws on is that there are two short screws. They go under the keyboard. If you put the long ones there you'll pierce a hole in the top cover. Do not overtighten them. After the computer is all back together again check all the functions just to make sure that all is running properly.

The whole operation should go off without a hitch, but if you do have problems retrace all of the above steps. The circuit does work, and if *yours* doesn't work you may have made a mistake somewhere. Some of you might want to add a switch to be able to change back and forth between normal and reversed screen. To do so, follow the wiring diagram in Figure 3. Make the wires long enough to be able to mount the switch on the outside cover or somewhere accessible when the cover is on. Warning! Wires that run outside of the RF shield can cause interference with your TV. Using a shielded wire will help. Ground the switch cover with a separate wire. This should take care of most of the RF problems.

Good luck.

How To Reduce That RFI On Your Monitor

By
Tony Distefano

First of all I would like to clear up a problem with the "Finger Saving Rapid Fire Circuit." The potentiometer in the Parts list has the wrong part number. It is not *the Rainbow's* fault, it is mine. You see, when I first made this circuit, it was indeed a 100k ohms pot, like the part number stipulates. But I thought the firing speed was not variable enough, so I changed the pot to 1 Mega ohms. If you have already bought the 100k ohms pot, do not despair, it will still work. The only difference is that the firing speed will not go as slow as the 1 Mega ohms one will. Radio Shack does not have a 1 Mega ohms pot in that package, so I cannot give you another part number for it. Almost any other electronic hobby shop should have it, though.

Okay, let's get on with this month's project. One of the problems with some of the older Color Computers is that when you plug in a disk drive, you get a lot of noise on the screen. The type of noise I am talking about is not a buzz from the speaker, but a type of wavy, herringbone pattern that seems to swim across the screen at a regular rate. Yes, that is "RFI." That stands for Radio Frequency Interference. I talked a little about RFI in the January '83 issue of *Rainbow*. It is very annoying to see this noise going back and forth on the screen all the time. Fortunately, there are a few things you can do to eliminate it.

One of the things you can do is this. Open the door and look inside the cartridge port. On both sides of the connector there should be metal clips. If there aren't, your local Radio Shack Repair Center can put them in for you. Apparently they will do this at no charge. I guess you will have to find this out for yourself. What this does is, when you have a disk controller plugged in, the clips act as extra ground connections. This prevents the controller from acting like an antenna.


Another way to reduce the RFI in the Color Computer is to get the aluminum shield from Radio Shack (again!). This shield fits under the keyboard. It snaps into the main board between the plastic standoffs and the board. The rest goes under the keyboard without any other connections. This extends the ground plain that is under the main board to the keyboard, too. The third way, and the main topic of this month's article, is to modify the TV that you are using with the Color Computer.

Before you start digging into your TV set, I'll give you a little background on how the signal gets from the Color Computer to the tuner. It starts from the connector in the back of the computer. It then goes down a shielded piece of wire to the connector box supplied by RS. This is a switch box which allows you to connect your antenna to it and

switch back and forth between regular TV signals and the computer without disconnecting anything. **THIS BOX IS A BIG SOURCE OF NOISE!** Get rid of it immediately! RFI can seep through that box like water through a screen door on a submarine. It is best to get rid of the wire that RS supplies too. You must make your own wire. This is not hard. Buy the four-foot white coax cable from RS part #15-1529. On one side, push on one of the F-56 connectors (supplied with the kit). On the other side install a Shielded Phono Plug, RS part #274-321. That is the end that goes into the computer. If your TV set has only the two screw type terminals you will need a F-61 connector as well, RS part #278-212. (more on that later).

So far, what you have done should reduce the RFI by quite a bit, but if there is still RFI coming in you must modify the insides of your TV. The next step requires that you remove the back of your TV. Only experienced hackers should take off the back of a TV. There are high voltages present in there. If your TV is like mine and most TV sets, the antenna connections are done via a small circuit that isolates the ground of the TV to the antenna. This is done to prevent electric shocks, because since there is no power transformer, one side of the AC line is directly connected to the internal ground. Touching the ground of the TV is like touching one side of a plug. Nothing will happen until you touch a ground point like the third pin of a three-prong plug or a water pipe. The Color Computer is grounded with a three-prong plug. If you try to connect them, watch out. Then you will see all the sparks fly. This is why the manufacturer of the set put a high impedance circuit to isolate the line from the antenna input. A small circuit is a lot less expensive (and a lot lighter) than a power transformer. Unfortunately this circuit is very sensitive to RFI. You have to remove this circuit and connect the antenna terminals directly to the tuner.

The first thing you must do before you take out the circuit is to add in a transformer to isolate the line. The transformer must be a *line isolation transformer*. Your local electronics-store should have one. The power rating of the transformer must match the power rating of your TV. It is usually written on the back of the set, or in the operating manual. Now, remove the back from the TV and mount the transformer somewhere inside, with the proper mounting hardware. Cut the AC cord that runs inside the set. If your set has a removable cord, cut the wires from the internal side of the connector. Re-route the AC side of the two wires to the input of the transformer and route the output of the transformer back to the TV input. This will isolate the ground from the AC line. I cannot emphasize enough the need for this transformer; if you don't put it in and you remove the circuit, you stand the chance of burning out your TV and your computer. Then you won't have to worry about RFI, only fire. Enough of this, now it's time to remove the circuit.

Unsolder the circuit from the antenna terminals. If the terminal is not the cable TV type, drill a hole and mount the F-61 connector. The other side of the circuit is usually a shielded wire that leads to the tuner. Cut the wire as close to the circuit as possible. Strip off the insulator and solder the inside wire to the tip of the F-61 connector. Solder the shielded part to the outer part of the connector. This will connect the antenna terminal directly to the tuner input. Before you plug in the TV, a little check is in order. With an ohm meter, and the TV on (but not plugged in) measure the resistance between the AC cord and the antenna terminal. Test both wires. If they read high impedance you are in business, if not, then check the wiring again. There should be no shorts between the AC cord and the antenna terminals. Replace the cover and try it. There you are, a clean picture. 

Memories Of The PROM

By Tony DiStefano
Rainbow Contributing Editor

This month I would like to take a close look at *memory*. What is a ROM? What is RAM? Or PROM? Or EPROM? Or EEPROM? They are all forms of memory chips, I think that before I go on, I'd better cough up a little background on memory chips. For those of you who know all about memory chips. I think that before I go on, I'd better cough up a little background on memory chips. For those of you who know all about memory chips, bear with me while I explain the concept of *memory* to those who are not quite up on the subject.

The first thing I'll look at is memory chips in general. A memory chip is a device which holds a certain amount of information. How much information it holds depends on the chip itself. It can be anywhere from 1K by 1 to 6K by 8 and more. (1K=1024) More on this later. A memory chip is

much like a telephone book. You look up a name and it gives you a telephone number. The name (in the phone book) is equivalent to the address lines of a memory chip. The telephone number (in the book) is equivalent to the data lines of a memory chip. Your fingers are equivalent to the CPU (Central Processing Unit), in this case the MC6809.

Let's take a look at the address lines first. A typical memory chip has between 10 and 14 address lines. This depends on how much memory the chip has. Address lines on a chip form a binary number (quick, look up binary numbers in your nearest math book). Each number is one memory location. One memory location is one byte. If the chip has 10 address lines then it has 2 to the power of 10 different combinations. That is $2*2*2*2*2*2*2*2*2*2$ and that is equal to 1024. (Is my math right?) In this chip (or phone book) there are 1024 bytes (or names). The CPU (or

phone book) there are 1024 bytes (or names). The CPU (or figures) can ask to look at any one of these bytes by giving the memory chip a binary number. This number, in the form of address lines then, tells the memory chip, what byte of information the CPU wants. This is the function of address lines.

The CPU gives the memory chip a binary number that corresponds to the address of where the byte is to be found. The memory chip then reacts by giving the CPU the information that is stored at that location, with the data lines. Data lines (like address lines) form a binary number. Memory chips can have from 1 to 16 data lines. Each line is known as one bit. Four bits make one nibble. Two nibbles or eight bits make one byte. Two bytes or 16 bits make one word. Most microprocessors work with 8 bits or 1 byte. Some work with 16 bits or one word. The Color Computer works with 8 bits. That means the CPU in the computer has 8 data lines or an 8-bit data bus. A bus is no more than wires that connect all of the chips together.

The last set of lines that are associated with the memory chip are control lines. Two of these lines include power and ground to the chip. The rest of the control lines are quite invisible to the user. The only one that is of interest is the chip select. This line tells the memory chip when to activate. Since there are usually more than one memory chip in a computer system, there must be a way of controlling which chip is to be giving or taking data from the CPU. This is where the chip select line comes in. A memory chip will not give or take data unless this line is activated. Well, that's enough on memory chips in general.

ROM stands for Read Only Memory. In this type of memory, the information that is in it cannot be changed, erased or lost. ROM memory is non-volatile. As soon as power is applied to a ROM, the data is available. The data in these chips was entered into it when the chip was made at the factory. Anyone can have a ROM made with their own data in it, but there is usually a minimum order of about 1000 pieces. It also takes a long time for delivery. Not practical for a home user. A ROM is said to be masked with the data when produced. All computers need at least some ROM memory in order to function. The Color Computer has Color Basic in ROM. Without ROM the computer would not be able to do anything.

RAM stands for Random Access Memory. This is quite different from ROM. RAM memory can be changed, erased and lost. When power is applied to a RAM chip, there is nothing in it. The computer can put any data it wants in it and change the data that is in it whenever it wants. One thing about RAM is that as soon as the power is removed from the chip, the data that was there is lost forever. RAM memory is volatile.

PROM stands for Programmable Read Only Memory. This chip is much like the ROM. The difference is that a PROM is blank. It has no data in it. All of the bits in a PROM are HI. With the proper accessories a user can put any data into a PROM. Once the data is entered or programmed into the chip, it becomes just like a ROM. It has all the properties of a ROM. It cannot be changed, erased, or lost. The only exception to that is, if a PROM is programmed more than once, the data can become very scrambled and totally useless.

EPROM stands for Erasable Programmable Read Only Memory. This chip is very much like a PROM. The major difference is that (like the name says) it can be erased. An EPROM is like a PROM but has a little window in the chip that exposes the internal circuits. When an EPROM is exposed to ultraviolet light it is erased. To protect an EPROM from being erased, a small sticker is placed over the window. All the bits return to their original state of HI. An EPROM can then be re-programmed with different data. It can be re-used over and over again.

EEPROM stands for Electrically Erasable Programmable Read Only Memory. This chip is much like the EPROM. The difference is that, instead of using a window and ultraviolet light to erase the memory, an electrical pulse is used. There is no need for a window or an ultraviolet light to erase an EEPROM.

How are memory chips used in the Color Computer? The CPU in the Color Computer is a MC6809E. It has 16 address lines. That means it is capable of addressing (or looking at) 65535 different bytes of memory. Normally it is said that this CPU can access 64K of memory. That is like having a phone book with 65535 names in it. A 32K Disk Color BASIC computer has many memory chips. First, it has 32K or RAM. Then it has 8K BASIC ROM, 8K Extended BASIC ROM and 8K Disk ROM. There is also 8K memory not being used. That totals up to 64K of memory. That is our full 65535 telephone book. But what if you had another phone book? What if you could switch between two phone books? That could give you much more memory. Or could it? In the Color Computer there is a chip called the SAM chip. SAM stands for Synchronous Address Multiplexer. This chip has the ability to switch between two phone books. EHH!? I mean between different memory chips. This gives the computer the capability to access a total of 96K bytes of memory. In a full blown Color Computer there is 96K of memory. Not all of this memory can be accessed at one time (especially with Radio Shack BASIC), but with the SAM chip in action and the right software, all of the 96K of memory can be used.

This brings me to the most asked question about the Color Computer. "How come, when I put 64K memory chips in my computer, I do not get any more free memory when I type in *PRINT MEM*, than with 32K memory?" The answer is that the BASIC INTERPRETER was not written to handle more than 32K of RAM. It is possible however, to use all the available RAM by using the right software. As soon as more companies realize that the extra memory is there, more and more programs will be written to take advantage of the full 64K memory.

Build A 'Y' Adapter For Your Disk Controller

By Tony DiStefano
Rainbow Contributing Editor

A lot of people have been asking me to explain how to expand their computer without having to spend a lot of money on expansion interfaces, power supplies, and the like. Well, here goes. This is the first of a series of expansion projects for the Color Computer. The emphasis on these projects will be *low cost*. They will be geared toward the experimentalist or the "hacker." They will satisfy the person who is tired of playing games and wants to expand his or her knowledge about hardware by experimenting. All of these projects will be done via the Program Pak connector. A problem arises in trying to experiment when you have a disk drive. Those of you that have disk drives really don't like to constantly remove the controller and plug in some experimental board and then replace the controller. And when it comes to using software, having first to save the program on cassette (yuk), unplug the controller, try the software out on the project and then replace the controller is not a very interesting proposition.

Did you ever try to plug two pair of headphones into one headphone jack? You can't. What you have to do is, go to your nearest Radio Shack store and buy a "Y" adapter for your headphones. That is what you are going to do; go to your nearest Radio Shack store and get a "Y" adapter for your disk controller. Well, not quite! You see they don't make a "Y" adapter for a disk controller. What a shame! I guess you'll have to make one. This brings me to the first project for the Color Computer. I call it "The Color Computer Y-er," or is that "wire?" In any case, it will solve the problem of having access to the bus with the disk controller plugged in. Putting this together is not that hard, and not expensive, but you have to remember that this just gives you access to the bus, it is not a buffered expansion interface. You cannot plug in a ROM Pak and expect it to work. To do that will require some circuitry. That may come later.

The Y-er requires four parts: one project board, Radio Shack No. 276-163; two 40-pin Card Edge Connectors, Radio Shack No. 276-1558, and a 12" piece of 40-wide flat ribbon cable. You can use Radio Shack No. 276-1542. This, however, has a connector on one end. You don't need it and have to cut it off.

If you can get ribbon wire from another source (like I did), do so; why pay more for a connector when you don't have to? As for tools, all you need is the regular set of tools for electronic projects. The only other tool you will need is a four inch vice. You need that to crimp the connector to the ribbon cable. And that's it—one hour later, you'll have your very own Y-er.

Okay, let's start. Take the project board and cut it in half, at about the "20" mark. You will need the half with the lower numbers. The other half may be used in a later project, but for now, put it aside. With a sharp knife, separate one end of the ribbon wire into individual wires about one inch long. Strip about 3/16 inch of insulation off of each wire. Tin each wire with solder. This is where the tricky part starts. This has

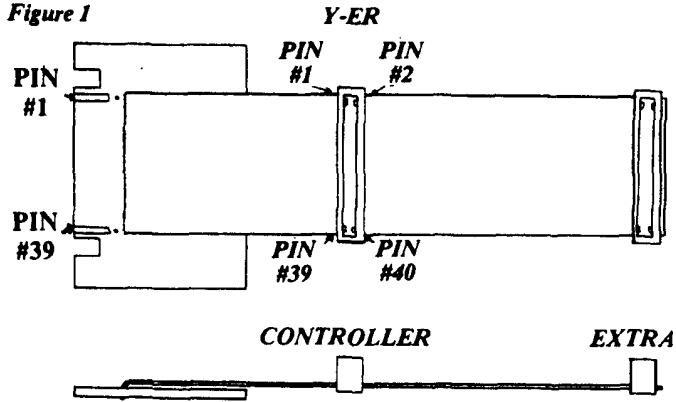


Here is the "Y-er" in use with next months project.

to be done just right. Hold up the ribbon wire by the stripped end and let the rest of the wire hang down. Starting from the right hand side, bend the ends of the wire alternately forward and backward. The first one on the right side goes away from you. This divides the ribbon into two sections. Counting from right to left, the odd numbers are away from you and the even numbers are close to you. The top section and the bottom section. The top section will solder to the top (component side) of the project board and the bottom will solder to the bottom (copper side). You do this by soldering the bottom side first. The first wire on the right goes into the hole just below the first finger on the right. That means that it will solder to the copper side. The second wire goes on the first finger on the component side directly above the first wire. Then the third wire goes under the second finger to the finger on the copper side. The fourth wire goes on top of the second finger and so forth until all of the wires are done. The last wire on the left goes on the top (component) side of the last finger. From now on this is known as the top side. The first finger on the right side is pin #1, the pin directly underneath is pin #2, the last finger on the top side is pin #39 and the pin under that is pin #40.

Now, it's time to put the connectors on. Slip one connector into the other end of the ribbon wire. The connector should be pointing upwards, in the same direction as the top of the project board. Place the connector about two inches away from the edge of the project board. Examine the connector and wire carefully and make sure that all the wires line up with the teeth of the connector. You might have to stretch and tug the wire into place. Gently pinch the connector together between two fingers. The teeth should start to press against the wire. Again check that all the teeth align with the wires. When they do, sandwich the connector in between two small pieces of wood. Put the wood and the

connector into a vise. Turn the vise until the connector is completely closed. Examine the connector to be sure that it is properly closed. If not, then give it another shot on the vise. It is important that the connector be fully closed. Now, slip in the second connector. It should stay close to the end of the wire. Crimp it like you did the first. If you think that you cannot properly crimp the connector, local electronics shop personnel might be able to help you.



Your "Y-er" should now look like the one in Figure 1. Before you go plugging this thing in, you should run a few tests. The first test is to determine if all the wires have continuity. This is where the other half of the project board comes in. Plug the board into one of the connectors. With an OHM meter, check that all the wires show continuity

between the two ends. Make sure that they all line up! Pin #1 on one should be pin #1 on the other. That is important: reversed wires can cause a disaster. Next check the continuity of the other connector. If all is well there is one more thing to check before you can use the "Y-er." You must check for shorts between the pins. Put one lead of the OHM meter on pin #1. Place the other lead on each of the surrounding pins one at a time. All of the readings should show high. There should not be any resistance between any pins. After all this checks out, remove the flux left behind when you soldered the wires to the project board. This can be done with flux cleaner. If you don't have any, an old toothbrush and lighter fluid will work. You might have to get down to a little bit of scrubbing. If you bought the Radio Shack connectors you will have to do a little trimming in order for the disk controller to fit in correctly. A small knife will do the trick. Cut deep enough that the controller fits in all the way.

After you feel sure beyond the shadow of a doubt that there are no shorts and no opens it's time to try it out. With the power off, plug the "Y-er" into the Color Computer's cartridge slot. Make sure it is in tight. Turn the computer on. If all is well, then turn it off again and plug the controller into the first connector. Turn it on and there you are, you have access to the bus with the controller plugged in. Right now you don't have anything to try it out with, but next month my project is a parallel printer port. For now try plugging the controller into the other connector to make sure that it works.

Build This Parallel Printer Interface

The *Parallel Printer Interface* is the first project that will adapt to my Y-ER expansion card. After you build this circuit, you will be able to use any parallel printer that is Centronics compatible. This circuit uses one MC6821 PIA. The other two chips used are for decoding the address bus to memory map the PIA from \$FF70 to \$FF73. The PIA has two functions: 1) to check if the printer is busy and 2) to transfer data to the printer. Bit 0 of port A is used to monitor if the printer is busy. All 8 bits of port B are used to transfer the data to the printer. The Control line CB2 is used to strobe the data into the printer. The PIA is initialized in such a way that CA2 auto strobes when a write to port B is done. Refer to the Motorola MC6821 PIA data sheet for more details on how a PIA works.

To put the circuit together is not too hard, but, like all electronic projects, care should be taken in the process. The circuit is shown in Figure 1. It consists of only three chips. The shopping list below includes everything you need to build the project. The first thing you must do is trim one side of one of the connectors of the project board. I'll explain why later. Look at Figure 2 to get the location of where to cut the board. You have to remove three pads. Well, it is actually six pads because there are three pads on each side. Use a hacksaw to cut the board. Be careful not to cut or

scratch any of the other pads. Next, position the IC sockets as shown in Figure 2. Note the position of pin one on each socket. They all go on the bottom and to the left. Position them the same way. Solder all the pins on all of the sockets. The next thing to do is to get the B-Plus and the B-Minus buses in. Turn the card upside down and locate the bus that is parallel to position 5 written on the sides. That will be the ground bus. I traced all the legs of the ground bus with a black grease pencil. This makes finding a ground point easier. The other bus, at location 33 on the sides, will be the B-plus line. That's the 5-volt line. Use a red grease pencil to mark it.

The rest of the soldering on this card will be made following the schematic. Solder the wires one by one, and after each connection is done, mark it on the schematic. This prevents you from trying to connect a wire twice or forgetting others. The small pads that point to the left on the schematic mean that it goes to a pin on the Color Computer cartridge side. The small pads that point to the right mean that it goes on the printer connector side. Refer to Figure 2 to get the proper location of the pinout for both the computer side and the printer side. Note that pin 1 for the computer side is the top of the upper pad and that pin 1 for the printer side is the bottom lower pad. I did it that way because the

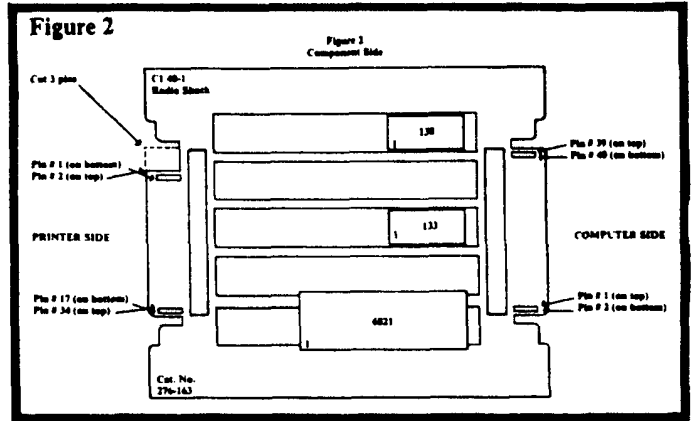
ribbon cable will sit properly in the connector. It will drop downwards. After all the wiring is done, clean the board of the flux residue. This can be done with flux remover, or anything else that will remove the flux. Check your work carefully and make sure there are no shorts or cold solder joints. It would be wise to check the wiring once again.

Put the board aside for now, it's time to concentrate on the printer ribbon cable. You have two choices: 1) buy one from Radio Shack, or 2) make one yourself. The first choice is simple; go to your nearest Radio Shack store and buy printer cable #26-1401. That is a 34-pin edge card to 36-pin plug. It's for a Model I/III to standard parallel printer cable. It will work perfectly. The second choice is a bit more work but will cost you much less. You will need three parts.

- 1) 34-pin edge card to ribbon connector, RS # 276-1564.
- 2) 36-pin Centronics type connector, RS # N/A.
- 3) 6 feet of 34-conductor ribbon wire, RS # N/A.

Take one end of the ribbon wire and connect it to the 34-pin edge card connector. Procedures on how to connect a ribbon wire to a connector are explained in last month's issue in my Y'ER article. Now, the other end is a bit tricky. There are 36 pins and only 34 wires. The last two are not used. When you put the ribbon and the connector together, make sure that the first wire (pin 1 on the edge card connector) meets with pin 1 on the Centronics connector. The last two pins will be left empty. On the connector the empty pin numbers are # 36 and # 18. Then, press the connector in the usual manner. This will give you a printer cable for about half the price of one you would buy.

Okay, now you have the board and the connector. After you are sure that both are constructed right, it's time to plug it in. Now, the computer gives you the familiar logo, but what do you do with it? It doesn't work, does it? You are missing some software to hook it into BASIC. The machine language program listed below will re-route the *PRINT#-2* command to the parallel port. All you have to do to hook it in is *EXEC*. When you type *EXEC* again, it will unhook itself and *PRINT#-2* will again go to the RS-232 port. Be careful that you give it an *ORG* in the right place, and make sure that you reserve enough memory, so you don't crash the program. That's all there is to it!



Shopping List For The Parallel Printer Port

Quantity	Description	RS # (if any)
1	PROJECT BOARD	276-163
1	40 pin IC socket	276-1996
2	16 pin IC socket	276-1998
1	74LS133	N/A
1	74LS138	N/A
1	MC6821	N/A
2	.01 uf CAP	272-1265

The listing:

* PARALLEL PRINTER ROUTINE
* BY TONY DISTEFANO

```

0001 0E00          NAM PPRINT
0002 0E00          ORG $7F00

0003 006F          PRNO EQU $6F
0004 009C          CRHOOK EQU $9C
0005 016B          PRHOOK EQU $16B
0006 FF70          PIA EQU $FF70
0007 7F00 BEFF70   INIT LDX #PIA          FIA LOCATION
0008 7F03 4F          CLRA
0009 7F04 A701        STA 1,X          DDR ACCESS A
0010 7F06 A703        STA 3,X          DDR ACCESS B
0011 7F08 A784        STA 0,X          ALL INPUT A
0012 7F0A 4C          INCA
0013 7F0B 979C        STA CRHOOK      BASIC IDEO
0014 7F0D B6FF        LDA #FF
0015 7F0F A702        STA 2,X          ALL OUTPUTS B
0016 7F11 A701        STA 1,X          CONTROL ACCESS
0017 7F13 B62C        LDA #2C         B+STROBE
0018 7F15 A703        STA 3,X          CONTROL ACCESS
0019 7F17 BE016B      LDX PRHOOK
0020 7F1A 10BE7F40    LDY RETURN+1
0021 7F1E BF7F40      STX RETURN+1
0022 7F21 10BF016B    STY PRHOOK
0023 7F25 39          RTS

0024 7F26 3402        PRINT PSHS A          PRINT DEVICE #
0025 7F28 966F        LDA PRNO
0026 7F2A B1FE        CMPA #FE            TO PRINTER?
0027 7F2C 260F        BNE NOGO            NOT PRINTER
0028 7F2E B6FF70      P1 LDA FIA
0029 7F31 B401        ANDA #1             IS PRINTER
0030 7F33 26F9        BNE P1              READY?
0031 7F35 3502        PULS A              TO PRINTER
0032 7F37 B7FF72      STA PIA+2           & STROBE
0033 7F3A 7262        LEAS 2,S            GO BACK TO
0034 7F3C 39          RTS                  CALLER

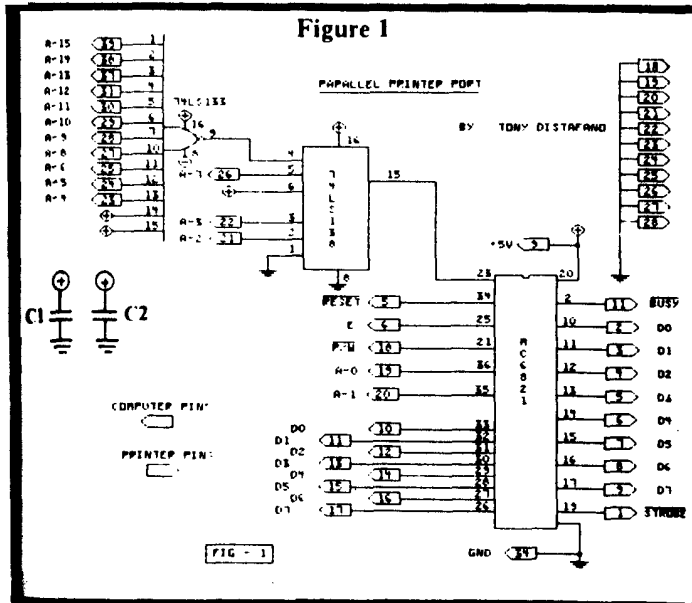
0035 7F3D 3502        NOGO PULS A          NOT FOR PRINTER
0036 7F3F 7E7F26      RETURN JMP PRINT    ADDRESS

0037 7F42          END

NO ERRORS FOUND

CRHOOK 009C 0013
INIT 7F00
NOGO 7F3D 0027
P1 7F2E 0030
PIA FF70 0007 0028 0032
PRHOOK 016B 0019 0022
PRINT 7F26 0036
PRNO 006F 0025
RETURN 7F3F 0020 0021

```



GI Sound Generator: Software Control For Complex Sounds

Last month's project was a practical one. This month we'll have some fun. How about making some interesting sound effects? Well, I'll show you how to interface the General Instrument's programmable sound generator number AY-3-8910 to the Color Computer. The features of this chip according to GI are:

- Full software control of sound generation.
- Interfaces to most 8-bit and 16-bit microprocessors.
- Three independently programmed analog outputs.
- Two 8-bit general purpose I/O ports (AY-3-8910).
- Single +5 Volt supply.

This Programmable Sound Generator (from now on, known as the PSG) is a LSI Circuit which can produce a wide variety of complex sounds under software control. Its flexibility makes it useful in applications such as music synthesis, sound effects generation tone signalling and even FSK modems (with a little extra circuitry). All of these sounds can be produced with just a few simple *POKES*, leaving the processor free to do other tasks like calculating more sounds, updating the screen or doing graphic animation (in the case of arcade type games). One or two pokes can produce sounds that carry on for several seconds, or even continuously.

This PSG is a register-oriented device. This means that communication between the processor and the PSG is based on the concept of memory-mapped I/O. The control commands are issued to the PSG by writing (*POKES* or *STAs*) to two memory locations. The first location (memory mapped at hex address FF65) is to select which internal register you wish to access. The second memory location is for the data you wish to enter or retrieve and is at hex address FF64. All functions of the PSG are controlled through 16 registers which once programmed, generate and sustain sounds on its own. More on how to program it later.

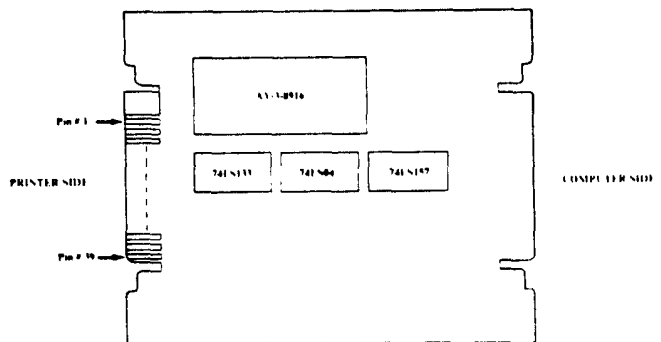
Refer to List #1 to get all the parts needed for this project. It is not hard to put together, but like any electronic project, care should be taken when working with static sensitive IC's.

Quantity	Part #	Desc
1	AY-3-8910	PSG
1	74LS133	TTL
1	74LS04	TTL
1	74LS157	TTL
2	16 PIN	SOCKET
1	14 PIN	SOCKET
1	40 PIN	SOCKET
2	.01 UF	CAPACITORS
1	10K OHMS	RESISTOR
1	PROJECT BOARD	RS # 276-163

LIST # 1

With the schematic (Diagram 1) in one hand and a soldering iron in the other, it's time to put the board together. Start by getting the ground and B-Plus buses wired in. It is best to wire all the connections to the connector on the side of the board that has the lowest number. That's the side with the

number one on the edge. The two buses run close to the edge connector, it will be easier to connect to. Refer to Diagram 2 for the proper layout of the sockets. The rest of the soldering is quite straightforward. Follow the schematic and cross off each line after it's done. This will eliminate any missed wires. When you are finished, clean the board in the usual manner. Check again the wiring with the schematic, remember that the Y'er is not buffered and is not forgiving of wiring errors. A short can cause many headaches. When you're finished, insert the chips (remember pin 1's) and plug it in. Turn the



computer on, when you get the familiar sign on, turn up the sound, type in and *RUN* this program.

```
10 AUDIO ON
20 POKE $HFF01,$HB4
30 POKE &HFF03,&H3F
40 SR = &HFF65 : WD = &HFF64
50 POKE SR,RND(15)-1 : POKE WD,RND(256)-1 :
GOTO 50
```

This short program will generate random sounds, beeps, pops, and whistles in the speaker of your TV. This is more or less just a test to make sure that the circuit is working. (You will have to use your imagination to come up with better software.) If you do not get any sound, check the wiring again; this circuit does work. I have a working model right here in front of me. Here, just put your ear a little closer and listen. Can you hear it? I told you it works. Okay, enough foolin' around, the following descriptions of the PSG are excerpts taken from the GI product description manual.

Sound Generating Blocks

The basic blocks in the PSG which produce the programmed sounds include:

- | | |
|--------------------------|---|
| Tone Generators | Produce the basic square wave tone frequencies for each channel (A, B, C) |
| Noise Generator | Produces a frequency modulated pseudo random pulse width square wave output. |
| Mixers | Combine the outputs of the Tone Generators and the Noise Generator. One for each channel (A, B, C). |
| Amplitude Control | Provides the D/A Converters with either a fixed or variable amplitude pattern. The fixed amplitude is under direct CPU control; |

the variable amplitude is accomplished by using the output of the Envelope Generator.

Envelope Generator Produces an envelope pattern which can be used to amplitude modulate the output of each Mixer.

D/A Converters The three D/A Converters each produce up to a 16 level output signal as determined by the Amplitude Control.

Operation

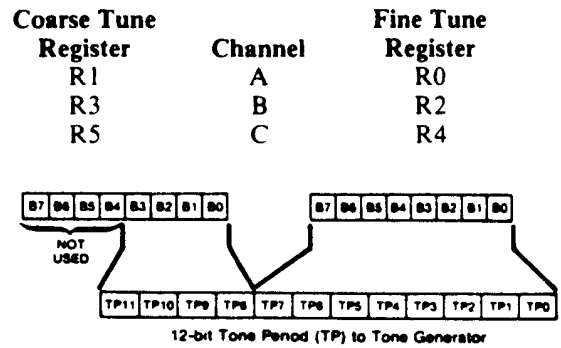
Since all functions of the PSG are controlled by the processor via a series of register loads, a detailed description of the PSG operation can best be accomplished by relating each PSG function to the control of its corresponding register. The function of creating or programming a specific sound or sound effect logically follows the control sequence listed:

Operation	Registers	Function
Tone Generator Control	R0-R5	Program tone periods.
Noise Generator Control	R6	Program noise period.
Mixer Control	R7	Enable tone and/or noise on selected channels.
Amplitude Control	R10-R12	Select "fixed" or "envelope-variable" amplitudes.
Envelope Generator Control	R13-R15	Program envelope period and select envelope pattern

Tone Generator Control (Registers R10, R1, R2, R3, R4, R5)

The frequency of each square wave generated by the three Tone Generators (one each for Channels A, B, and C) is

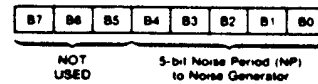
obtained in the PSG by first counting down the input clock by 16, then by further counting down the result by the programmed 12-bit Tone Period value. Each 12-bit value is obtained in the PSG by combining the contents of the relative Coarse and Fine Tune registers, as illustrated in the following:



Noise Generator Control (Register R6)

The frequency of the noise source is obtained in the PSG by first cutting down the input clock by 16, then by further counting down the result by the programmed 5-bit Noise Period value. This 5-bit value consists of the lower 5 bits (B4-B0) of register R6, as illustrated in the following:

Noise Period Register R6



Mixer Control—I/O Enable (Register R7)

Register R7 is a multi-function Enable register which controls the three Noise Tone Mixers and the two general purpose I/O Ports.

The Mixers, as previously described, combine the noise and tone frequencies for each of the three channels. The determination of combining neither/either/both noise and tone frequencies on each channel is made by the state of bits B5-B0 or R7.

The direction (input or output) of the two general purpose I/O Ports (IOA and IOB) is determined by the state of bits B7 and B6 of R7.

These functions are illustrated in the following:

Mixer Control—I/O Enable Register R7

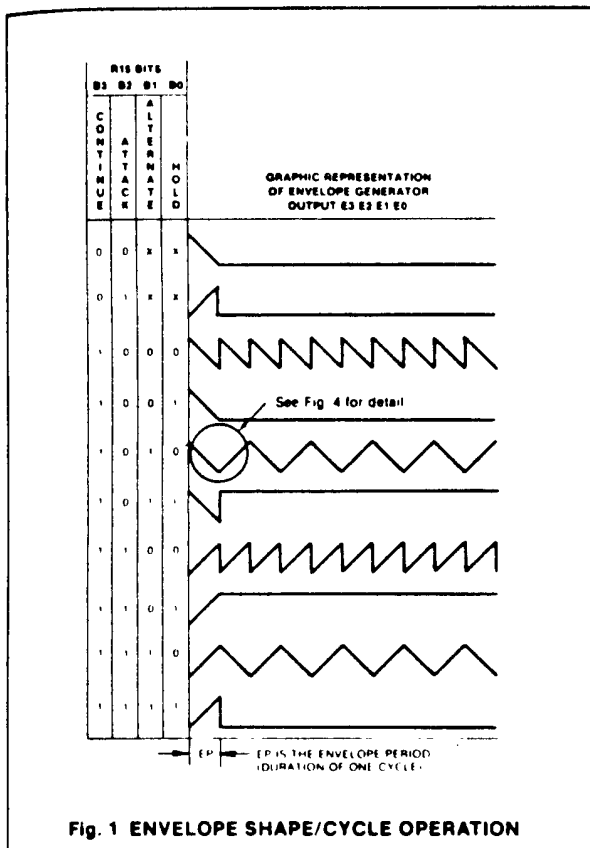
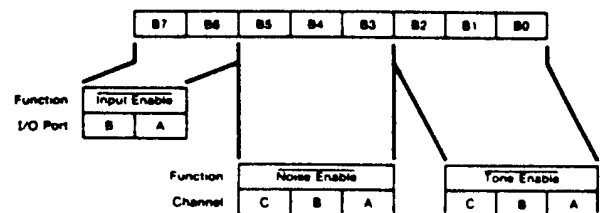
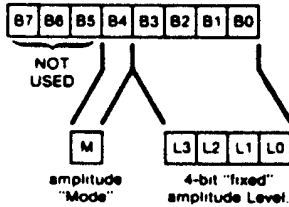


Fig. 1 ENVELOPE SHAPE/CYCLE OPERATION

Amplitude Control (Registers R10, R11, R12)

The amplitudes of the signals generated by each of the three D/A Converters (one each for Channels A, B, and C) is determined by the contents of the lower 5 bits (B4-B0) of registers R10, R11, and R12 as illustrated in the following:

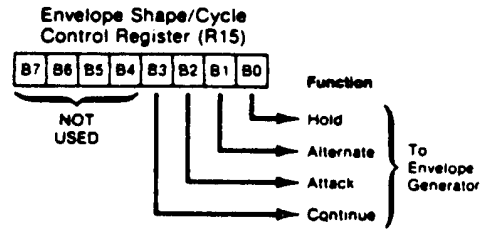
Register	Channel
R10	A
R11	B
R12	C



Envelope Shape/Cycle Control (Register R15)

The Envelope Generator further counts down the envelope frequency by 16, producing a 16-state per cycle envelope pattern as defined by its 4-bit counter output, E3 E2 E1 E0. The particular shape and cycle pattern of any desired envelope is accomplished by controlling the count pattern (count up/count down) of the 4-bit counter and by defining a single-cycle or repeat-cycle pattern.

This envelope shape/cycle control is contained in the lower 4 bit (B3-B0) of register R15. Each of these 4 bits controls a function on the envelope generator, as illustrated in the following:

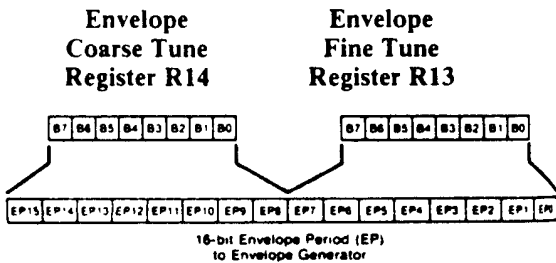
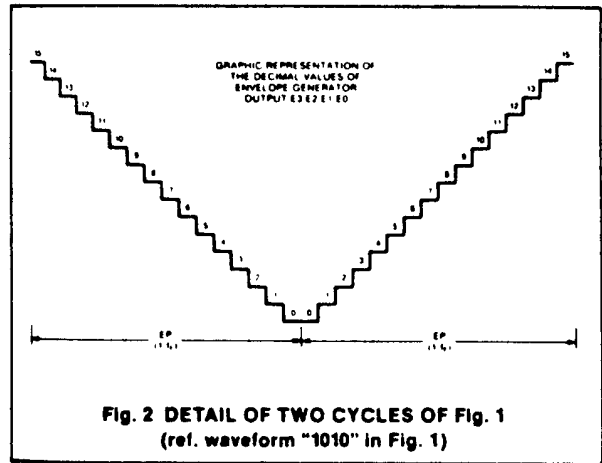


Envelope Generator Control (Registers R13, R14, R15)

To accomplish the generation of fairly complex envelope patterns, two independent methods of control are provided in the PSG: first, it is possible to vary the frequency of the envelope using registers R13 and R14; and second, the relative shape and cycle pattern of the envelope can be varied using register R15. The following paragraphs explain the details of the envelope control functions, describing first the envelope period control and then the envelope shape/cycle control.

Envelope Period Control (Registers R13, R14)

The frequency of the envelope is obtained in the PSG by first counting down the input clock by 256, then by further counting down the result by the programmed 16-bit Envelope Period value. This 16-bit value is obtained in the PSG by combining the contents of the Envelope Coarse and Fine Tune registers, as illustrated in the following:

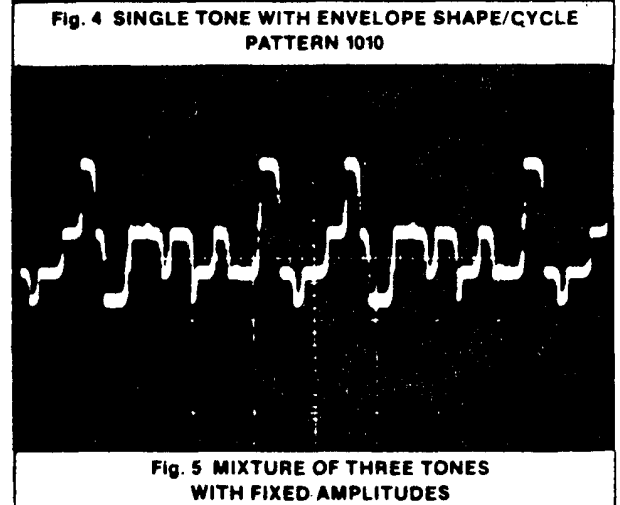
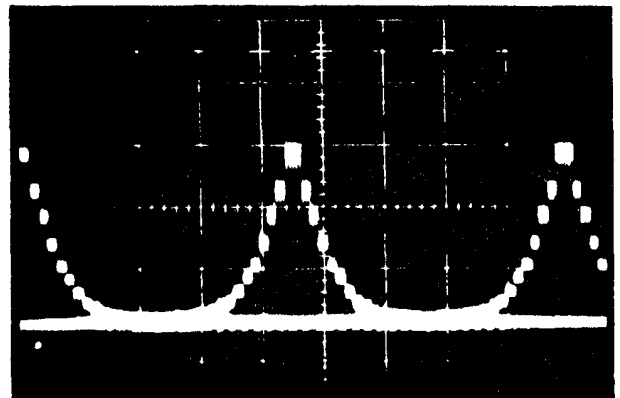
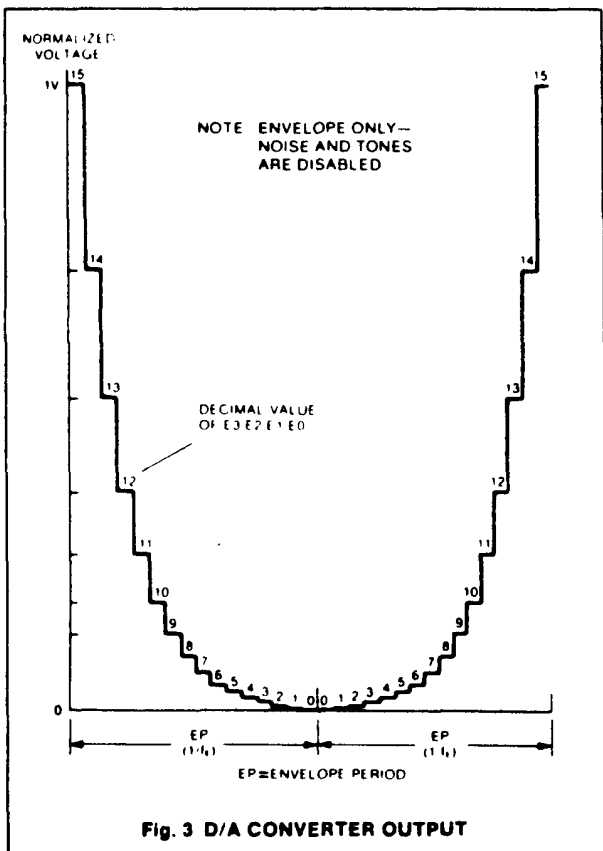
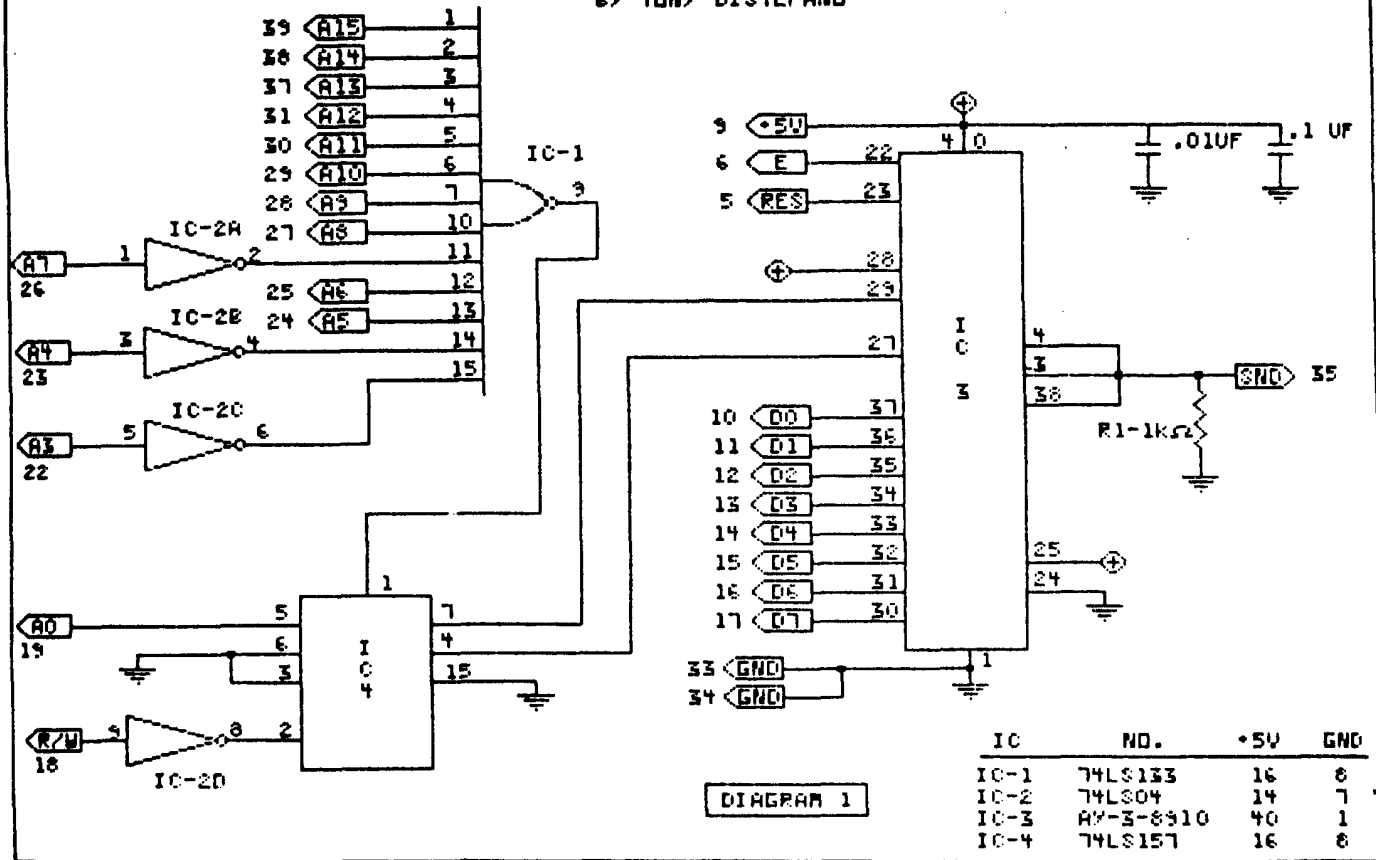


D/A Converter Operation

Since the primary use of the PSG is to produce sound for the highly imperfect amplitude detection mechanism of the human ear, the D/A conversion is performed in logarithmic steps with a normalized voltage range of from 1 to 1 Volt. The specific amplitude control of each of the three D/A Converters is accomplished by the three sets of 4-bit outputs of the Amplitude Control block, while the Mixer outputs provide the base signal frequency (Noise and/or Tone).

(Reprinted by permission, courtesy General Instruments/Microelectronics)

**GI PROGRAMMABLE SOUND GENERATOR
FOR THE RADIO SHACK 80C
BY TONY DISTEFANO**



A Hardware Hacker Cleans House

Trying to come up with one project a month is sometimes just too much. First, I must think of a good project, then, there is the research. Making diagrams and designing circuits. After that, there is the proto-typing. (that is the hardest part and the most time consuming)—buying the parts, soldering it together, and then trying to find out why it doesn't work. Sometimes that requires a whole change of circuit. After the hardware works fine, it's time to write the article. All this must be leading up to something; it is, this month is cleanup month. It's time to answer a few questions and clear up a few problems. That is to say I didn't have time to complete a project. But, I'll tell you this, there will be some *hot* projects coming this fall.

Okay, the first thing on the agenda is a correction: Radio Shack does have a 1 mega-ohm potentiometer. The part number is 271-211. This correction comes from the April 1983 issue of *the Rainbow*. It stated that Radio Shack did not carry this part, but as someone pointed out to me, they do. This was in my finger-saving rapid-fire project.

The next thing is a little longer. A reader sent me a letter and asked me if it was possible to do my Reverse Screen on a "F" board, or the latest version, the one which has the smaller RF shield. Well, it is possible to do it. There are just a few differences. The first change is the "U," or chip numbers. Since Radio Shack decided to change the complete layout of the Color Computer, they changed the chip ID numbers. U29 on the old board now becomes U8 on the new board, that's the 74LS02. U7 on the old board becomes U6 on the new board, that's the MC6847. I stated in my article you have to remove the 74LS02 and bend the pins upwards and replace the chip in the socket with the pins sticking out. Well you can't do that on the new board. Radio Shack decided to save a few cents by not putting this chip in a socket. Fortunately they had the insight not to solder the input pins to ground. When making the modification you don't have to remove the chip, just solder your wire straight to the pin, there is nothing connected on the other side. Use the same pin numbers as the other chip. Remember though, you still have to bend pin number 32 on the MC6847. Apart from these changes, the reverse screen will work fine.

The next problem is with my Y'er. You cannot plug in a Radio Shack program pack or any other pack for that matter, into one of the slots when the disk controller is plugged in the other slot. It will not work and might even cause damage to the computer and or to the disk controller. This is because the bus is not buffered nor does it have the switches to select between different slots. It will only work with my projects or other projects that are independently memory mapped. That is to say it does not use the CTS (pin number 32 on the cartridge connector) or SCS (pin number 32 on the cartridge connector) for selecting the device. These signals are being used by the disk controller software and

hardware. If another device were to use these signals, there would be a bus contention and the CPU would get very confused. Maybe later on I could work on an adapter that would let you use these signals without any problems.

Another point of interest to you goes back to my article on memory chips. If you can recall, I talked about ROMs and EPROMs. Here is a little more. The socket that is available for Extended BASIC inside your computer has 24 pins. It usually holds an 8K ROM supplied by Radio Shack. This is where Extended BASIC resides. It is necessary though, to put an Extended BASIC ROM there. You can put different software there. All you need is to insert a chip. What chip? That depends on how long your program is. It is possible to put software that takes from 1K to 8K of memory. Most of the time an average user puts in an EPROM, because they are so easy to program, and are relatively inexpensive. All you need is some software and an EPROM Programmer, and of course an EPROM. Most of the common EPROM chips available today are 24 pins, that means that they are pin compatible with the socket (in the Color Computer) and will plug into the socket directly. There is however, one chip that is not. This is the 2764 8K EPROM. They why use it? You might ask. Well it's the least expensive 8K EPROM chip on

This adaptation works for the 2764 EPROM only. After the adaptation, it will fit in any of the Color Computer's ROM sockets: BASIC, Extended BASIC, or even the Disk BASIC socket.

This adaptation can be done directly to the chip or in-between two extra sockets. If done with the sockets, one 24 and one 28-pin socket is needed. The 28-pin to seat the EPROM and the 24 to go into the other socket.

- 1) Directly on the Chip.
 - a) Solder pins 1, 28, 27 and 26 together.
 - b) Pry up pin 20 so that it does not go back in the socket when the chip is replaced.
 - c) Solder pins 20 and 22 together
 - d) Solder a wire to pin 2 and insert the other side of the wire into the hold left by pin 20.
 - e) Insert the chip so that pin 3 on the chip goes into pin 1 in the socket.
- 2) Using two sockets.
 - a) Align pin 3 of the 28 pin socket on top of pin 1 of the 24 pin.
 - b) Solder all the pins but pin 20 of the 28 pin socket to the 24 pin socket.
 - c) Solder pins 1, 28, 27, 26 of the 28 pin together.
 - d) Solder pin 20 to pin 22 on the 28 pin socket.
 - e) Insert chip into the top socket. Pin 1 of the chip goes into pin 1 of the socket.
 - f) Insert the bottom socket into computer socket. Pin 1 goes into pin 1 on both sockets.

The only other consideration left is when programming the 2764. The above modification reverses the address lines A11 and A12 as seen by the Color Computer. This means that, at programming time, these lines must be again reversed. This can be done in software or in hardware. Hardware requires that the two traces that lead to the EPROM programmer socket be reversed. In software all you have to do is transfer the second 2K block of memory with the third 2K block of memory.

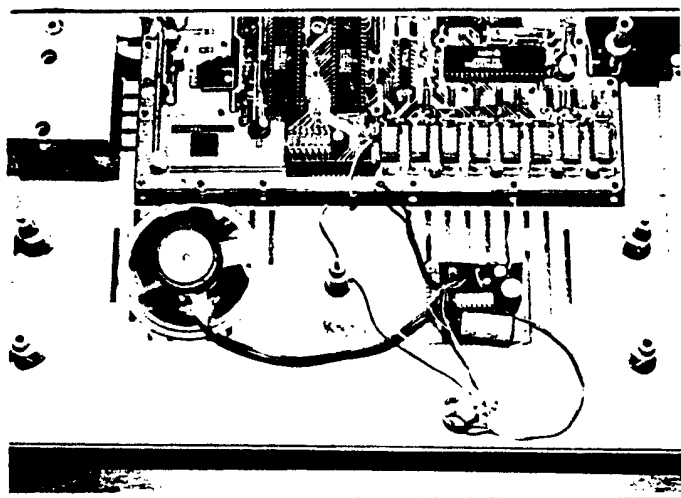
Some of you who have had problems with my projects have written me asking for help. I must confess. I have a hard time answering letters. If you do write me, be patient, I will answer in time. Tell you what, I'll set aside one night a week, let's say Monday night, when you can call me at home, and talk to me about your problems. My telephone number is (514) 473-4910. But please, don't call before 7 p.m. or after 11 p.m. The cost of your long distance call might be worth not having to wait for a response in a letter.

Build A Speaker/Amplifier For Your Computer

January 9, 1981; that was a great day. I bought my first Color Computer. Today, two and a half years later I bought a video monitor. It is a standard composite-video monitor. It is a 9" green phosphorus screen Electro-home. I know what you are thinking, "Oh no, not another video monitor adapter!" Well, I'm not about to bore you with another version of this adapter. I used one of them myself rather than design my own. When I connected my monitor, I was delighted with the clear, crisp quality of the picture. I found that it had one thing missing—a speaker. I could not make any sounds with this monitor because it did not have a built-in speaker. At first, I would keep my color TV set next to it with the volume up. That was quite an inconvenience. Well, you guessed it, this month's project is a low cost, built-in speaker and amplifier for the Color Computer. The whole thing fits under the keyboard. It even has a volume control with an on/off switch.

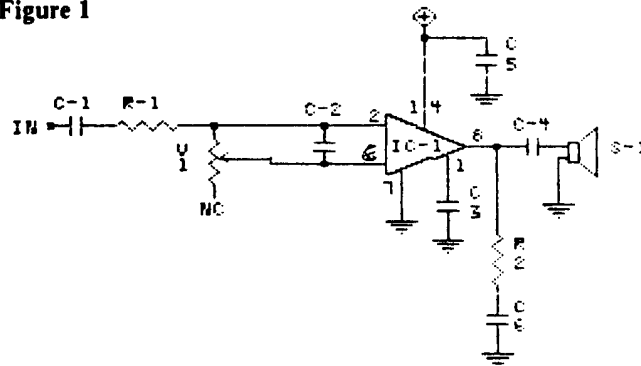
The heart of the amplifier is the power IC # LM80. This is a complete preamp and amp in one. It requires a minimum of support parts and is quite rugged. It also has a high input impedance, about 150k ohms. A high impedance is needed in our case because the sound circuit on the Color Computer is high impedance. If you look in the '83 Radio Shack catalog, you will see that they say the power supply has a maximum of 10 volts. That is not true, it must be a mistake in printing. It can, in fact, take up to 22 volts for B+. We will be using 12 volts.

To construct this project you will need the standard project tools—things like screwdrivers and pliers and cutters and soldering iron and solder and a drill to mount the volume control. You will also need everything on the list of parts. See Figure 2. All of these parts are quite common and need not be bought at Radio Shack. As a matter of fact, I had all of the parts in my parts bin. I have included the Radio Shack numbers, where possible, just as a matter of convenience. Mount all of the components except the speaker and volume control on the Proto board. Following the schematic in Figure 1, solder all the components together. All the ground points indicated on the schematic should be soldered together at one point. This is to prevent what is known as ground loops. A ground loop is when an electrical signal has two or more paths to get to the same point. This path or loop can act like an antenna, in which it



can radiate RF noise or act like an RC circuit and cause feedback. Though it is not indicated in the schematic, pins 3, 4, 5, 10, 11, 12 are also grounded. This acts like a heatsink for the IC, and should be enough for most applications, but if you think that you'll be using this amplifier very loud, it

Figure 1



would be wise to add a small heatsink to the IC. Also, make sure that the ground wire that goes from this board to the main board is at least 22 gauge. Make this wire about 4" long. Now, the B+ line (12 volts) should also be 22 gauge.

This wire will go to one side of the switch on the pot. Make this wire 5" long. The other side of the switch will go to the 12 volt supply. You might think this to be heavy wire, but this chip can deliver up to 8 watts. (That is a lot of power.) The switch-to-power wire should be about 10" long. That will go to the B+ on the main board. The connections for the speaker should be 24 gauge. Make these wires about 5" long. That should be long enough to reach the holes on the other side of the computer just underneath the keyboard. Solder the other ends of the two wires to the speaker. There are two more wires from the board, and they go to the volume control—one wire to the center and the other to the left side. Make them about 5" long, too. That will be long enough to reach anywhere in the front of the computer.

Open the computer in the usual way and remove the keyboard. Place the components in accordance with Photo #1. You may want to tape them down temporarily so that they don't move around too much. Drill a hole in the computer to mount the volume control. Personal taste will judge exactly where to drill it. The hole should be $\frac{5}{16}$ ". Mount the volume control in the hole. Be careful not to break the attached wires. Make sure that the position of the volume control will not get in the way of the keyboard. The next step is to connect the B+ (12 volts) and ground. If you have the "F" (or 285) board, find the power by looking at the photo. It is the jumper for 16/64K memory. Use the one marked 16K. For the ground connection, scratch off a bit of the green coating on the PCB just to the right of the keyboard connector, under C59, and solder to that. If you have another version, use Test point #9 for the 12 volts and Test point #4 for ground. The last connection to make is the input. That connection goes all the way to the top. It connects to pin #3 on the RF adapter. This is all that has to be done; but before you close the computer, check your work.

Replace the keyboard and turn the computer on. To test your amplifier, any sound command will work. This one line

program works fine:

```
10 SOUND RND(255),1 : GOTO 10
```

Turn the volume control on. You should hear a click. Turning the volume control up should result in some random sounds coming out of the speaker. 100 IF SOUND = NONE THEN TROUBLESHOOT ELSE CONTINUE. Only kidding folks, but that is the next step. If you don't get sound, check your wiring and check for cold solder joints. Make sure that the chip is plugged in the right way. If the sound is loud at first and drops as you turn the volume control up, you have the outside wire on the pot on the wrong side. Unsolder, and reverse it. Other than that, you should have no problems.

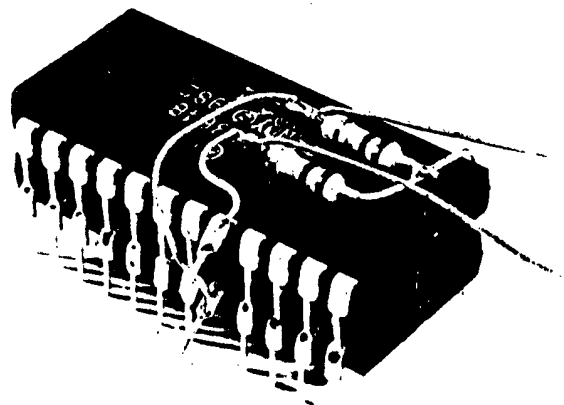
Now, you could leave the speaker and amplifier just taped down, or you could use some rubber cement or screws. Don't use a permanent glue though, it could make a mess if ever you have to remove this thing from the computer or change the speaker. When all checks out, replace the cover and enjoy your new speaker and amplifier.

Figure 2

#	Part	Description	RS #
R-1	Resistor	150K ohms	271-047
R-2	Resistor	2.7 ohms	n/a
C-1	Capacitor	10 uf @ 16v	272-1423
C-2	Capacitor	.022 uf @ 16v	272-1066
C-3	Capacitor	10 uf @ 16v	272-1423
C-4	Capacitor	470 uf @ 16v	272-957
C-5	Capacitor	220 uf @ 16v	272-1006
C-6	Capacitor	.1 uf @ 16v	272-1069
S-1	Speaker	2 to 5 inch	40-248
IC-1	Amplifier	LM 380	276-076
PC	Proto-Board	.1 inch spacing	276-1392
V-1	Potentiometer	100K ohms	271-216

Install Your Own ROM 'Switcher'

Ready 1.0, 1.1, 1.2, 7.5, 9.4 hike! No, this is not a football lesson. It's a problem that Radio Shack has presented Color Computer users. There are presently three versions of the BASIC ROM. These are 1.0, the very first version to come out; 1.1, the second one to come out and probably the most common; and the latest one, 1.2. There are many differences between them, (I am not about to describe all of their differences in this article), but when a friend of mine bought the latest version of the Color Computer, he was very happy to find out that a lot of the basic software ran a little faster on the 1.2 version. Of course, I wanted the newer version, too. But I also wanted to be compatible with the older software. I wanted the best of the



wo worlds. So I got out my old soldering iron and proceeded to do just that. I'll show you how to modify your computer to have and be able to select between two of them.

The first thing you'll need to know is what version you have in your computer. The way to do that is simple. You just have to type in: EXEC 41174 [ENTER]. This will tell you what version you have. If you don't have the 1.2 version then read on! You, too, may want to be able to select between the two. If you have the 1.2 version already, you may want to have the 1.1 version also. If you have 1.0, you may want to have 1.1 also, or 1.2 — in fact you may have any two of the ROMs.

The next thing you have to do is to acquire the newer 1.2 BASIC ROM (or whichever one you want to add). I went to my local Radio Shack Computer Center and tried to order one. They said that it would take weeks to arrive. I am much too impatient to wait that long, so I called up my old friend Bob Rosen from Spectrum Projects. He had some in stock and sent me one right away.

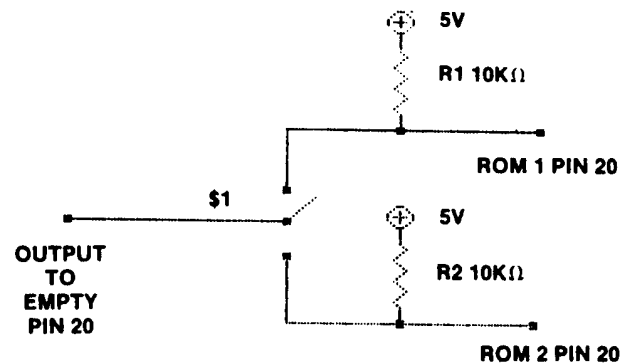
Now it's time to get down to the nitty-gritty of this month's project. All you will need for this project is the 1.2 ROM (or any other), two 10K ohm resistors and a single pole double throw switch. A good switch to use is a RS #275-625. Any SPDT switch will work, but I like this one because it is very small and will fit almost anywhere. Open your computer in the usual way. Remove the BASIC ROM with an IC remover. If you don't have an IC remover, a small flat screwdriver will work. Stick the blade under one side of the IC. Push in very slowly. The IC should start to lift. Don't push too hard or too far in. Push in just enough so that the IC begins to lift. Now remove the screwdriver and insert it in the other end. Again push in until it starts to lift. Keep doing this procedure back and forth until the IC comes out.

Next, mount the new IC on top of the old one. Make sure that pin 1 on the old pin matches pin 1 on the new one. It's time to solder them together, but leave a space in between the two chips, so that the air can circulate between the chips and keep them cool. You don't want a heat problem to develop. I used a popsicle stick as a spacer. Solder all the pins except pin 20. Leave pin 20 of the two ROMs unsoldered. In fact, pull them apart a little just to be sure that you don't. Next, cut the two 10K ohm resistors so as to make them fit. Examine Photo 1 to get the proper positioning of the two resistors. Solder them together according to the photo. Solder the resistors to pin 1 of the IC pack (or better known as piggy-back — remember 32K). Next, take two small pieces of light gauge wire (wirewrap wire 278-501 from RS is a good wire to use) and wire the resistors and the protruding pins together using Figure 1 as schematic. Remember to clean any residue left from soldering. Solder the other ends of these wires to each side of switch 1. The center pin of the switch goes to the empty pin 20 on the

socket, the pin that we lifted earlier. The best way to do that is to take a $\frac{3}{16}$ " piece of stiff wire (a snapped off piece of resistor lead will do just great) and solder it to the end of the wire. The end of this wire will go into the empty pin of the socket.

Find a good spot to mount the switch. You will need a $\frac{3}{16}$ " drill bit and drill if you used the RS one. I mounted mine on the back cover just left of the reset button. Okay, now insert the IC pack into the socket. Make sure that pin 20 on the bottom IC does not enter the socket. It must be bent enough so that it rests on the outside of the socket and another wire can be inserted into the hole. Insert the wire that comes from the center pin of the switch into the hole. Check your work carefully. When you think all is right, turn on the computer. Make a note which side the switch is on and type in EXEC 41175. This will tell you which ROM is active. Turn the computer off, flip the switch and turn it on again. Type in EXEC 41175 again. Now you should have the other ROM active. Note the setting on the switch and mark it above the switch. Flipping the switch when the computer is on will not harm the computer but it is not recommended because the BASIC interpreter might get lost. That is to say, the software expects certain routines to be in certain places. If you switch the routines around (by switching the ROMs) without telling BASIC about it, BASIC will jump to the wrong place and get lost. Close up the computer and tidy up your room. I'll see you next month.

Figure 1

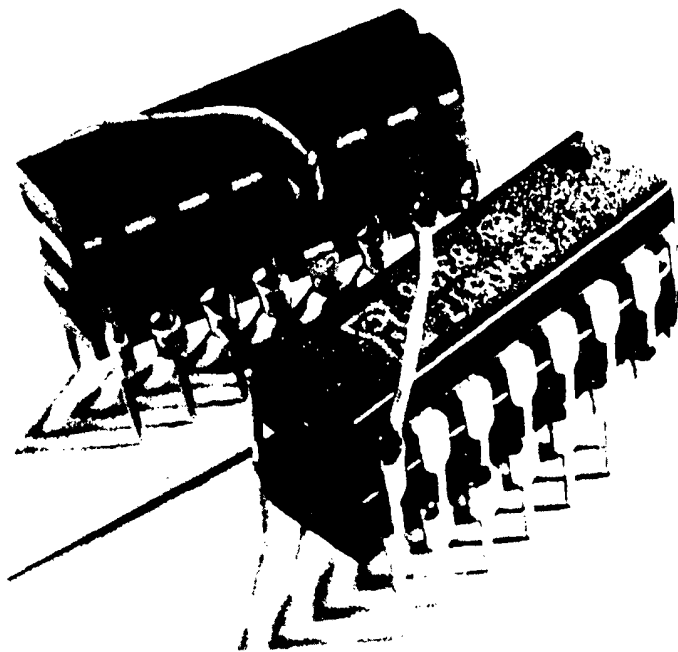


UPGRADING YOUR COLOR COMPUTER 2

A long, long, long, long, long time ago Radio Shack introduced a little gem called the Color Computer. It had a whopping 4K RAM memory. The first thing you knew, the hobbyists were able to expand this computer's memory to 16K, probably even before Radio Shack introduced its 16K. Then the hobbyists boosted the memory to 32K by a method known as piggyback, again before Radio Shack. The hobbyists didn't stop there, 64K memory was next. But the 64K memory did not consist of piggybacking four sets of 16K chips, it was a different chip. All of the 32K piggyback chips were put on the shelf or put in computers whose owners did not care for 64K. Where am I leading with all this, you may ask? Well, a few months ago Radio Shack released another little gem, the Color Computer 2. Only this time they started with 16K memory and after looking inside one, I saw that it was very easy to upgrade to 64K. Most of the owners will be upgrading to 64K. This is part one of this month's article: How to upgrade from 16K to 64K memory in the Color Computer 2. To upgrade your Color Computer 2 to 64K, you must first get a Color Computer 2. Ha ha, only kidding, but you will need some 64K memory chips. The chips to get are #4164, with an access time of 300ns or faster. There are many different suppliers for this chip, with many different numbers, but as long as they are 4164 compatible they will do.

Open the CC-2 by removing all of the screws on the bottom. Remove the top cover. Carefully remove the keyboard by pulling the ribbon wire out by the dark colored base. The eight identical chips along the bottom of the PCB are the old 16K chips. Using an IC extractor or a small flat screwdriver, remove the eight chips. Put them aside for now. Insert the 4164 chips. Make sure that pin one on the chip goes in pin one in the socket. The only other thing you have to do is to make one solder joint. Look for a small "W1" in between the 6822 PIA and the SN74LS244 chip. Right above this W1 mark are two solder points. Solder these two points together and *voila*, 64K memory. That's all there is to it! Now, if a whole lot of people convert to 64K, that will leave a lot of 16K chips sitting around doing nothing.

This will be the second part of this month's article. Those 16K chips that are removed from the Color Computer 2 to make 64K can be used to give you 32K. Yes, it will be in the piggyback fashion. It is a little harder to do than a 64K upgrade, but nevertheless can be done in less than one hour. To upgrade a 16K Color Computer 2 to 32K you will need either a set of 16K chips removed from another Color Computer 2 or buy a set of 16K chips. One important note to remember is these chips are not ordinary 16K chips. They are not the same chips that come from the regular Color Computer. The chips that come from the first CoCo are 4116 chips. The 16K chips that go into the Color Computer 2



are 2118 chips. The main difference between a 4116 and a 2118 chip is that the older type 4116 needs three power supplies to run. It needs +5 volts, +12 volts and -5 volts. The newer 2118 needs only +5 volts to run. It is also more power efficient.

If you piggyback 4116 chips in the CC-2, it will not work, and might even do some damage, so don't put 4116 memory chips in the CC-2. Now that the warning has been said, it's time to continue. Remove the eight memory chips from the board. You should now have 16 memory chips, eight from your computer and eight from another source. Put half of them aside for now. Examine one of the chips carefully, notice the pins. When a pin comes out of the chip it is wide, then it becomes narrow. The narrow part of the pin is the part that goes into the socket. With a narrow pair of long-nose pliers (or a finger, if you have narrow fingers) grab the narrow part of pin 4. Bend the pin back and forth until it falls off. Be careful that you don't bend the wide part of the pin. That part of the pin should stay intact. Do this to seven more chips to give you a total of eight chips with the narrow part of pin 4 removed. Next take out the untouched chips. Mount the chips with the short pin on top of the chips with all the pins. The photo will help you determine how to position the chip. This photo was taken with a mirror, so that you can see both sides of the chip (there is only one chip in the photo). Before you start soldering, make sure that pin

I on the top chip is on top of pin one on the bottom chip. Leave a small gap in between the top and bottom chip. This is needed for ventilation. Next, solder all the pins together. All but pin 4 of course, it is now too short to reach anyway. Okay, now get a small piece of wire. Any thin wire will do, I used some Radio Shack wirewrap wire. Cut eight pieces about 1/2" long. Strip off about 1/32" of insulation off each end of each wire. Solder one end of this wire to pin 4 (the one with the short leg) of the chip-pak and the other end to pin 9 (still on the top) on the chip. Do this to all eight chip-paks. After you are finished clean the chip-paks carefully with a resin remover. Radio Shack now carries resin remover. The part number is 64-2322. It is not of the best quality, but is good for small jobs like the chip-paks. After the chip-paks are clean, check them over for shorts or cold solder joints and repair them. When you are sure that they are all okay,

plug the paks into main board. Again, make sure that pin one on the chip goes into pin one in the socket. When you are finished, turn the computer on and type in:

PRINT MEM [ENTER]

Without a disk drive plugged in, the amount of memory displayed should be 24871. With a drive plugged in, the value should be 22823. And finally without Extended BASIC it should be 31015. If you get these values, all is okay and you can close up your CC-2. That's all there is to do to upgrade a CC-2 from 16K to 32K. If you have problems, chances are that you soldered one of the chips in backwards. In that case you might as well throw the two chips away and start again. Well, that's all for this month.

I hope you have lots of good memories.



Trouble Shooter Makes Program Pak Connection

I would like to get right into business this month. The first thing I want to discuss is about telephone calls. I was good enough to give out my number to those people who had problems with my projects or want to express an idea or opinion and I think that it is great that I got a lot of response; but please limit your calls to Monday nights only! For those of you who do not have my number and those of you that just started getting *the Rainbow*, my number again is (514) 473-4910. Call only after 7 p.m. EST and not too late. I am an early riser!

Okay, now back to the order of the day (month?). One of the best things to come out of these phone calls is that people can point out errors in my articles. (Yes, I do make mistakes. You should see my replacement-parts bills.) The faster I know about the mistakes, the faster I can write a fix for them. The main reason for the mistakes or errors is the transfer of information from my proto-board to you, *the Rainbow* reader. All of my projects are tried and tested before I write them in here. If a project that you put together does not work, check your work carefully. If it still doesn't work, call me and I'll give you a fix. If I can't give you a fix on the spot, I'll write one up in the following article. Speaking about fixes, here is one.

There is a problem with my internal speaker/amplifier project. The capacitor marked C-2 in the parts list is wrong. It is not a .022 uf capacitor. It should be a .002 uf capacitor. Also, the part number for the LM-380 (IC-1) is not 276-076 like it says in the article, but 276-706. Sometimes my fingers get carried away. The last thing to mention is a misprint in Figure 1. The little scribble to the right of C-2 should read "6." That is pin #6 of the IC. I would like to thank Hilton Wasserman for pointing this out to me. For your interest, the schematics in "Turn of the Screw" are drawn with the help of my Color Computer and an EPSON printer. I use

the program *Schematic Drafting Processor*, currently being distributed by Spectrum Projects. See the ad in this magazine.

I received a letter from Kyle Rogers this month, this is a part of it; ". . . I enjoy reading 'Turn of the Screw,' and I would like to build many, if not all, of the projects presented. But I find that I have neither the tools, skills, nor the time to construct the devices. Many hardware columns in other magazines have alleviated this problem by making an agreement with an outside company for that company to manufacture and market pre-assembled versions of the projects presented in that magazine. . . ." Can anyone help? Please contact me through *the Rainbow*.

The remainder of this article will be in answer to Tewfick Chidiac's question, "What do all the pins in the Program Pak connector, on the side of the computer, connect to, anyway?" Okay, Tewf, here is a detailed description of the Program Pak connector.

First of all, the main use for this connector is to plug in (you guessed it), Program Paks. These are little plastic cases that contain a small PCB (Printed Circuit Board). On this PCB there is usually one or more ROMs (Read Only Memory). This is where the game or utility software is stored. Other examples of different types of Paks are; disk controllers, RAM (Random Access Memory) boards, printer ports, I/O (Input/Output) boards, serial communications boards and so on. They all have one thing in common. They access the "bus." A bus is a term used to represent common wiring that connect to many components. Having access to the bus lets you expand the capabilities of your computer. The bus in the Color Computer fall into three main categories; data lines, address lines and control lines. Our computer has eight data lines, it is known as an 8-bit data bus. It also has 16 address lines and several control lines. The

following is a list of all the lines (or pins) that come out of the connector.

Color Computer Bus Descriptions

PIN#	Function	Description	Direction
1	-12v	-12 Volts	Output
2	+12v	+12 Volts	Output
3	HALT	Halt line to CPU	Input
4	NMI	Non Maskable Interrupt	Input
5	RESET	Resets the computer	Input
6	E	Main clock signal	Output
7	Q	Secondary clock signal	Output
8	CART	Cartridge detect signal	Input
9	+5v	+5 Volts	Output
10	D0	CPU Data line #0	I/O
11	D1	CPU Data line #1	I/O
12	D2	CPU Data line #2	I/O
13	D3	CPU Data line #3	I/O
14	D4	CPU Data line #4	I/O
15	D5	CPU Data line #5	I/O
16	D6	CPU Data line #6	I/O
17	D7	CPU Data line #7	I/O
18	R/W	Read/Write signal	Output
19	A0	CPU Address line #0	Output
20	A1	CPU Address line #1	Output
21	A2	CPU Address line #2	Output
22	A3	CPU Address line #3	Output
23	A4	CPU Address line #4	Output
24	A5	CPU Address line #5	Output
25	A6	CPU Address line #6	Output
26	A7	CPU Address line #7	Output
27	A8	CPU Address line #8	Output
28	A9	CPU Address line #9	Output
29	A10	CPU Address line #10	Output
30	A11	CPU Address line #11	Output
31	A12	CPU Address line #12	Output
32	CTS	Cartridge Select signal	Input
33	GND	Ground Return	Input
34	GND	Ground Return	Input
35	SND	Sound Input	Input
36	SCS	Spare Select signal	Output
37	A13	CPU Address line #13	Output
38	A14	CPU Address line #14	Output
39	A15	CPU Address line #15	Output
40	SLENB	Device Disable	Input

I shall describe each pin in detail and where it connects to inside the computer. 1) This output pin comes from the power supply. It supplies -12 Volts to any component, maximum drain is 100 ma (miliamps). 2) This output pin also comes from the power supply. It supplies +12 Volts and has a maximum of 300 ma. 3) The Halt line is an input line that goes directly to the CPU. It is tied to normally HI (+5v), by a resistor of 4.7k ohms. When this pin goes low, the CPU completes its last instruction and goes into the tri-state mode. Tri-state means that all of the CPU bus lines are high impedance, They are neither HI nor LOW. It is as if nothing was connected to it. 4) The NMI input line goes directly to the CPU. It is also tied HI. When this line goes low, the CPU performs a non-maskable interrupt. That means that the CPU will jump to a predetermined address and continue to execute this code until it reaches an RTI (Return from

Interrupt), in which case it will continue doing what it was doing before the NMI line went low. 5) The RESET line connects to the CPU and all the man chips that have reset lines. All except the VDG chip. That is only controlled by the external [RESET] switch in the back of the computer. The function of the RESET line is to initialize all the components to powerup conditions. Under software control, if the value in byte # \$71 (113) is not equal to \$55 (\$ denotes Hex), the computer will do a cold start. If it is, it will attempt to do a arm start. This line is also tied HI, but ith a 100k ohm resistor. 6) The E clock is the main timing for the CPU. The E clock is generated by the SAM (Synchronous Address Multiplexer) and goes into the CPU and nto the bus. 7) The Q clock is the secondary clock. It is also generated by the SAM. The Q clock leads the E clock by 90 degrees. 8) This input goes into one of the PIAs (Peripheral Interface Adapter). It is tied HI with a 10k ohm resistor. The function of this line is to detect the presence of a Program-Pak and to jump to it. 9) This output pin comes from the power supply. It supplies +5 volts to any component with a maximum of 300ma. 10-17) These eight DATA pins provide bi-directional communications between the CPU and the system. They connect directy to the CPU and all other data related chips. 18) The Read/ Write line is an output which tells all data related chips which direction the data lines of the CPU are in. 19-31 and 37-39) These 16 pins address lines come from the CPU and tell all other data related chips, where in memory the CPU wants to Read or Write. 32) This output is a chip select. It comes from pin #12 of the 74LS138. It is memory mapped to select memory between \$C000 (49152) and \$FEFF (65279). This is a 16K block of memory known as the cartridge memory or the Color Disk BASIC ROM area if you have a disk drive plugged in. The pin is active LOW, which mans that the meory chips associated with this pin will respond when it is low. 33-34) These two pins are ground returns. All signals are returned to the system through them. 35) This input is connected directy to the sound multiplexor (MC14529b) pin #12. With this pin, sounds in the audio range can be output to the TV speaker. 36) This output is another chip select. It comes from pin #9 of the 75LS138. It is memory mapped to select memory between \$FF40 (65344) and \$FF5F (65375). This is a 32 byte long block of memory mainly used for external I/O for such devices as a disk controller or PIAs. The pin is active LOW, which means that the I/O devices associated with this pin will respond when it is low. 40) This input is connected to pin 6 of the 74LS138. This active LOW pin disables the internal device selection. This allows decoded but unused sections of memory to be used by the cartridge hardware.

Now that you know all about the cartridge connector, go out and experiment but be careful, CPUs and SAMs are quite expensive.

References:

Radio Shack Color Computer Technical Reference Manual
 Motorola Microprocessors Data Manual.
 Artwick Microcomputer Interfacing



Adding Function Keys To Your Keyboard

In the last few months, there have been many new (and better) keyboards introduced for the Color Computer by companies other than Radio Shack. Some of them are functionally the same as the original Color Computer keyboard, meaning that all the keys are all in the same place and do all the same things. There are, however, some keyboards that are different. They have extra keys. Some have one extra key, some have more. Why are these keys there? What do these keys do? How can I get these keys to work with my computer without having to buy? These and many more questions will be answered in this article.

The first thing I'll give you is a background on how the Color Computer keyboard works. The keyboard itself is nothing more than a bunch of switches. Fifty-two to be exact. The computer monitors these switches and when you press one, the computer responds in some predetermined way, most of the time by putting an ASCII character on the screen. The computer must be able to read or scan all of the 52 keys. One way to do this would be to have 52 inputs to the computer via many PIAs (peripheral interface adapters). A better method of reading these keys is to matrix the switches. This is where the switches (or keys) are arranged in rows and columns. That is how the Color Computer reads the keyboard.

Figure 1 shows us how the keyboard is wired. The PIA marked number U8 (U18 on the "F" board and U7 on the CoCo-2) is the only digital circuit used. The PIA chip is a programmable interface device which functions as both an input and an output register. The eight keyboard columns are attached to the B side of the PIA. These eight lines are programmed to be outputs. The seven keyboard rows are attached to the A side of the PIA. These seven PIA lines are programmed to be inputs. To read the keyboard, only one column is enabled by writing a zero in the bit that corresponds to that column and by writing ones in all the other bits. If a key has been pressed in that column, one of the input lines will be a zero and the key location will correspond to the bit that is low. By scanning each column in the keyboard, all of the keys may be checked. Eight columns by seven rows should give you access to 56 keys. The color computer only offers you 52. There is a difference of four locations (or keys) that are not accessible from the keyboard. There are simply no switches for those locations. Okay, if you look carefully at Figure 1 again, the row with the [ENTER], [CLEAR], [BREAK] and [SHIFT] keys has the four empty spots. That means that all we have to do is add four switches to these empty spots and then we can access all 56 locations.

Figure 2 shows how to wire up four switches. These switches can be any single pole, single throw, normally open, or momentary on switches. The Radio Shack switch #275-1547 will do fine. There are five to a package, and they're not

very expensive. They are small enough that they fit almost anywhere. In Figure 2, the numbers that go to the four switches and the common are the pin numbers to the keyboard connector. That is where the keyboard connects to the main board. The connector is marked 1 and 16 on each end. It is very easy to solder to the back of the connector on all the Color Computer models. If you have a CoCo-2, then it is easier to solder the wires to the PIA itself. You cannot get to the back of the CoCo-2 keyboard connector, it is soldered straight up. The pin numbers that correspond to the PIA are marked in brackets.

Before you solder in the wires and switches, you must decide where to put the four switches. There are many possibilities. I drilled four holes on the top cover of the Color Computer, just above the TRS-80 decal. I used a five pin connector to the wires, so that I could remove the top cover when I go in to do some experiments, which is almost every second day. Anyway, I thought of putting the four switches right into the keyboard. While this is possible, it is very tricky to solder to thin-film PCB. That is what the newer keyboards are made of. I don't recommend that anyone do it unless they have a lot of experience in soldering. As soon as you touch this stuff with a soldering iron, it melts. Maybe you can mount the buttons inside the keyboard and run separate wires out of the keyboard and to the connector. This is possible, though I haven't tried it myself. I will leave this part up to you. You're on your own. Put the switches wherever they best suit your needs.

The next part is the software. This short program will show you which key is which and what ASCII value it has. Type it in and *RUN* it. Then press all the keys one at a time. Try them with the shift key, too. Then you can label the four keys accordingly. Some of the ASCII values are regular ASCII characters and can be gotten from the keyboard. Also, there is one combination that does not even produce a character. That is [SHIFT] F1. In order to use these keys in your program, you must use *CHR\$* or program it in machine language. The ASCII values I got were using Color BASIC 1.1.

If everyone could agree on some kind of standard for these keys, then the software companies would be able to include them in their software, i.e., control codes or special functions like delete and insert in such programs as a word processor or spreadsheet. I would like to mount a campaign to standardize these function keys. I hope to hear from all of the software writers and the keyboard manufacturers so we can get started. If we generate enough interest, maybe Radio Shack will add these keys to future Color Computers. Are you listening, Radio Shack?

Figure 1

Keyboard Wiring Diagram. The eight keyboard columns are attached to the B side of the PIA. These eight lines are programmed to be outputs. The seven keyboard rows are attached to the A side of the PIA. These seven PIA lines are programmed to be inputs.

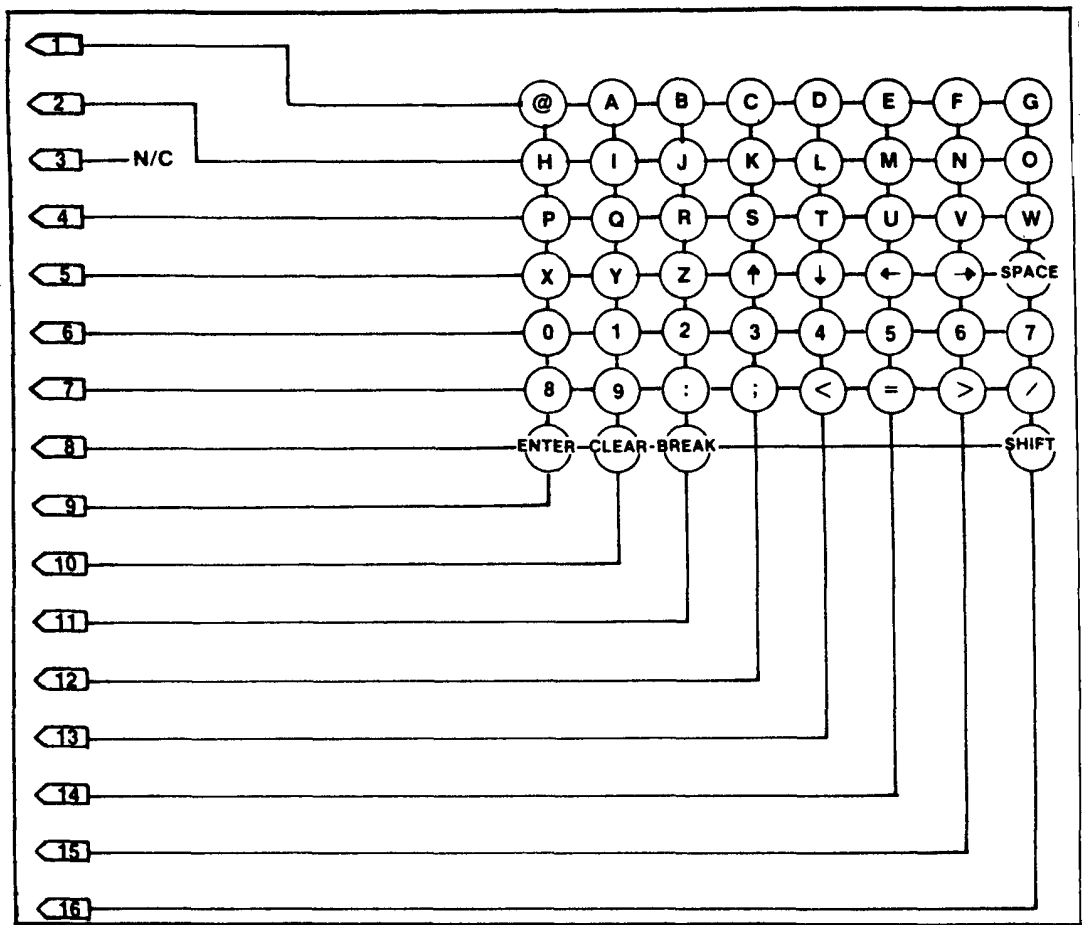
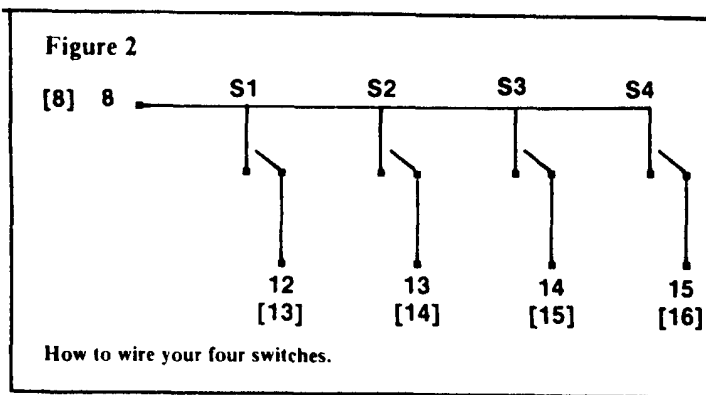


Figure 2



How to wire your four switches.

The Listing:

```

10 ' A PROGRAM TO CHECK WHAT
20 ' FUNCTION KEY CORRESPONDS
30 ' WITH WHAT ASCII CODE
40 '
50 CLS
60 A$ = INKEY$
70 IF A$ = "" THEN 60
80 IF A$ = CHR$(64) THEN PRINT "
FUNCTION KEY #1 (UNSHIFTED)" : G
OTO 150

```

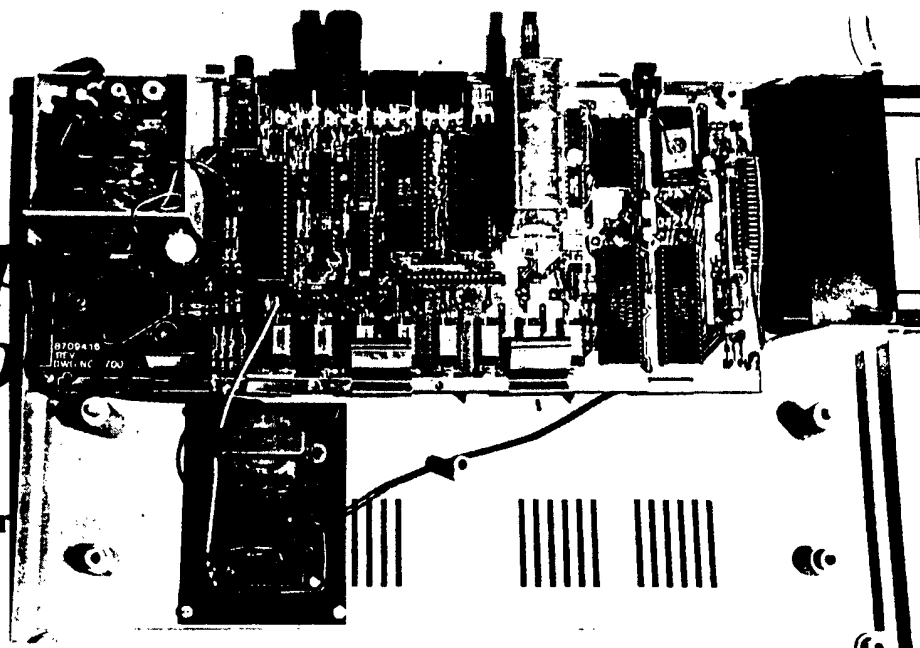
```

90 IF A$ = CHR$(189) THEN PRINT "
FUNCTION KEY #2 (UNSHIFTED)" :
GOTO 150
100 IF A$ = CHR$(103) THEN PRINT "
FUNCTION KEY #3 (UNSHIFTED)" :
GOTO 150
110 IF A$ = CHR$(4) THEN PRINT "
FUNCTION KEY #4 (UNSHIFTED)" : G
OTO 150
120 IF A$ = CHR$(1) THEN PRINT "
FUNCTION KEY #2 (SHIFTED)" : GOT
O 150
130 IF A$ = CHR$(52) THEN PRINT "
FUNCTION KEY #3 (SHIFTED)" : G
TO 150
140 IF A$ = CHR$(214) THEN PRINT "
FUNCTION KEY #4 (SHIFTED)" : G
OTO 150
150 PRINT : PRINT : PRINT A$
160 PRINT : PRINT : PRINT : PRIN
T "HIT ANY KEY TO CONTINUE"
170 A$ = INKEY$
180 IF A$ = "" THEN 170
190 GOTO 50

```

A 12-Volt Power Supply For The CoCo

By Tony DiStefano
Rainbow Contributing Editor



When Radio Shack came out with the CoCo 2, they made it as close as possible to the old Color Computer as they could. Nevertheless, there are some minor differences. First of all, the physical size of the case is different. It is a lot smaller. All the other differences are not very obvious. You cannot see them from the outside and most are invisible to the user. That means that even though they are different, it will function the same. For example: The RS-232 circuit is completely different, different ICs are used, they are placed in a different part of the computer. But, when you use the RS-232, it will work with all the old software. Another difference is the RF modulator. It is a completely different modulator. The circuit is all changed, yet it works. So what is all this coming to? Why am I telling you all this if it is all the same? There is one change Radio Shack did that will affect the user. It is in the power supply.

In the Color Computer, there are four voltages coming from the power supply. Five volts, 12 volts, -5 volts, and -12 volts. In the CoCo 2 there is just one voltage. That is 5 volts. A small negative voltage is produced on board for the RS-232, which requires negative voltage to work. It does not have 12 volts. Most people would say, "So what!" Well, if you don't have a disk system or a graphics tablet, you wouldn't know the difference. But, if you have the old Radio Shack controller (the ones sold with the gray drives) or a graphics tablet, you will find that neither of them work with your CoCo 2. Why? This is where the 12 volts come in. Both of these accessories (and probably many more) need 12 volts to function. The CoCo 2 does not have 12 volts. One way to solve this problem is to get the Multi-Pak Interface from Radio Shack. Not a bad idea, it has the 12 volts and is quite handy if you have many things to plug into it. On the other hand, it is expensive if all you have is a disk drive. Well, there is another solution, build a small 12 volt power supply. I'll show you how.

This power supply is small enough that it will fit under the keyboard of the CoCo 2. The IC that I used can supply up to

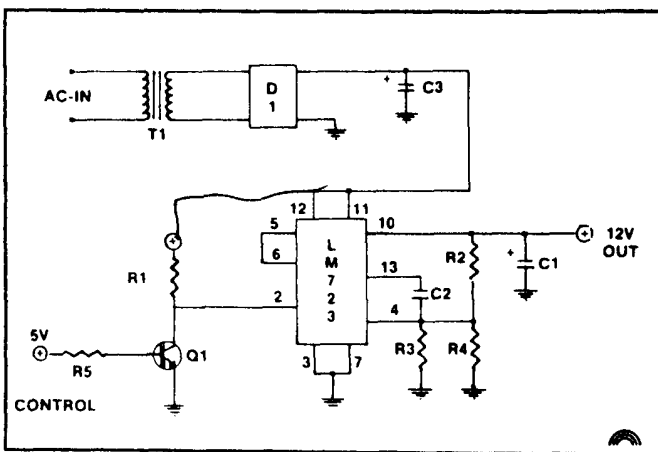
150 milliamps or .15 amps. The reference manual for the regular Color Computer says that the 12 volts can supply up to 300 milliamps. I don't think that you need that much current, seeing that the Radio Shack controller needs only about 25 milliamps. Another reason that I used this chip is that it has the capability of shutdown. This means that under certain circumstances the IC will stop to output voltage. This is very important when you want to turn the CoCo 2 off. At first I thought of just adding a switch. But then that would mean that every time you wanted to turn the computer on, you would have to first turn on the 12 volt switch and then the five. In the case of the WD-1793 (which, by the way, is the FDC or Floppy Disk Controller used in the Radio Shack Disk Color System), the 12 volt supply must go on at the same time or before the 5 volt. The 12 volt must also be shut off before the 5 volt. That is a lot to ask for, just to turn the computer on and off. Next, I tried a relay to switch the 12 volt on and off, but that was just not fast enough. Well, that is why the LM-723 chip suited this case so well. It can be switched on and off by an external source and was fast enough to boot!

The first thing you must do (like always) is to get parts. There are not a lot of parts and are all available at your local Radio Shack store. Here is a list of parts you will need:

Quantity	ID	Description	RS#
1	IC-1	LM-723 (voltage regular)	276-1740
1	T1	12 volt transformer	273-1385
1	D1	50v bridge rectifier	276-1151
1		Perf board	276-158
1	C3	1000uf @ capacitor @ 35v	272-1019
1	C1	10uf @ capacitor @ 35v	272-1013
1	C2	100pf capacitor @ 25v	272-123
1	Q1	2N2222 transistor	276-1617
3	R1,2,5	4.7 ohm 1/2w resistor	271-8019
2	R3,4	15k ohm 1/2w resistor	271-8036
1		14 pin socket	276-1999

There is also the regular paraphernalia like wire, screws, and solder, etc., that you must get. I'll leave that for you to figure out. Next, mount all the components (except the IC) on the perf board according to the photo of my prototype. The component layout is not too important since you are doing point-to-point wiring. Wire the components according to the schematic in Figure 1. The two 15k resistors in parallel are there because I needed a 7.5k resistor and Radio Shack did not have one. The only problem you will have is with the power transformer. The pins do not quite fit in the holes. Make a mark on the board where the pins sit. Use a small drill and widen the holes so that the transformer will fit in. Check the wiring carefully. Now it is time to test it. Please do not install this power supply before you test it. Putting more than 12 volts on the controller will cause many dollars of repair. Plug the IC in the socket. Make sure that pin 1 of the IC is pin 1 of the socket.

Figure 1



transformer to the AC line of the CoCo 2. The polarity is not important in this case. The two points to solder are the center one and the right side one. That puts our circuit on the side of the fuse. It saves us from putting in another one. Plug the CoCo 2 in and measure the voltage at the output. It should be zero volts. Now take the control wire and touch it to the plus side of the 1000µf capacitor. The output voltage

should jump up to 12 volts. If it does, it is okay and time to install it into your CoCo 2. If not, back to the drawing board and check the wiring again. The output voltage should not vary more than five percent. If it does, try changing the voltage divider resistors. The three resistors that control the output voltage are R-2, R-3 and R-4. Do not change these values by much, just try another of the same value; it might have enough difference.

The final thing to do is to mount the board properly and make the rest of the connections. Again, make sure that the CoCo 2 is unplugged when soldering to the computer. The transformer just fits under the keyboard. Use four screws to secure it to the base. Solder the ground wire of the power supply to the ground on the main board. The base of two diodes that are on the bottom left is just fine. The control input can go to any 5 volt location on the main board. I put it on the top side of C-28. The only wire left to connect is the output. That connects to pin 2 on the cartridge connector. When you solder to it, make sure that you don't short out any other pin. The last test to do is to check the 12 volt pin. Plug the CoCo 2 in and leave it off. Measure the voltage at pin 2 and ground. It should be zero volts. If not, check your work again. Now turn the computer on and the voltage should jump to 12. If so, turn it off and plug the controller (or other) into the computer. Turn it back on and measure the voltage. If it is 12 volts, turn everything off and close it up. That is all there is to it.

A lot of people have been calling me about the CoCo 2 64K article. It seems that there is a revision "B" on the CoCo 2 and that they could not get it to recognize the 64K. I have not seen this revision myself, but from what I hear through the grapevine, it should work anyway. If anyone can tell me for sure, send me a line.

The article "ROM Switcher" has a bug in it. The two resistors that solder to pin 1 of the chip do not go to pin 1. I took a photo of the wrong chip and therefore made an error on the pinout. They should go to pin 24. If you tried this out and found that it didn't work, that is the problem. Just do the modification and all will work okay. It should not have caused any damage to the chips or the computer. Till next time.

Designing A Video Monitor Output

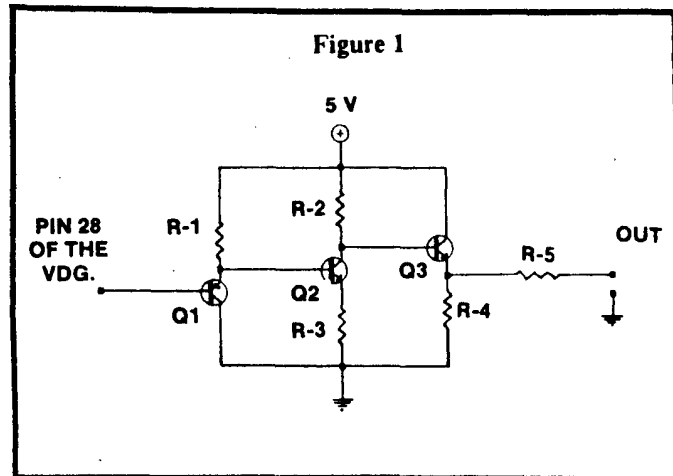
A few months ago, when I wrote the article on how to add a speaker to your CoCo, I mentioned that I was not going to do an article on a video monitor output for the CoCo. Well, I got a lot of letters and phone calls asking me to do one. The major complaint is that most of the monochrome (black and white) video outputs do not have enough gain to drive certain monitors. I thought this was quite strange because I had made one from a schematic in *the Rainbow* and did not have any trouble with gain. I always had plenty of brightness and good contrast with my Electrohome monitor. Well, just this week, I bought an Amdek 300A amber monitor, and guess what? My video monitor adapter did not have enough gain to drive this monitor properly. I thought there was something wrong with the monitor. I brought it back to the place I bought it and aired my complaint. They checked it out and told me that the monitor was okay and that my computer was not strong enough to drive this type monitor. It didn't take long

before I took my video monitor adapter and threw it out the window. Now what was I to do? Humm! I guess I'll have to design my own.

What follows is what I designed as a video monitor output for the Color Computer. Following the schematic in Figure 1, you see a three stage amplifier. The first transistor is used as an impedance amplifier. The second transistor is an inverting voltage gain amplifier. The last transistor is used as an emitter follower. This adds the current gain necessary to drive monitors that are terminated with a 75 ohm load, just like the Amdek. It is not hard to construct this circuit. You will need all of the usual project tools like a soldering iron, pliers, cutters, screwdrivers and the like. Get all the parts in the parts list, though I think that most of you will have all of these parts in your junk bin. There is nothing hard to get, but do get all the right resistor values, close is not good enough. You can mount it on a piece of perf board like in the list, or you can mount it on just about anything. The output connection can be made in many ways. You can drill a hole in the back of your CoCo and install a chassis mount RCA connector — Radio Shack #274-346. If you don't want to drill a hole in your CoCo, just use a long wire with an RCA jack on the end, or whatever type terminator your monitor has. Most monitors have RCA terminators. You can mount the board inside the computer with double-sided tape on top of the RF adapter.

The .1 uf capacitor in the parts list does not show up on the schematic. This is a decoupling capacitor and goes from the +5 volt line to ground. This is only to eliminate noise generated from the power supply. This video monitor output will work on any CoCo version, it will even work on the CoCo 2.

Like usual, if you have some problems with my projects or modification, or if you have a good idea you would like to share with me, give me a call on any Monday night after 7 p.m. My telephone number is (514) 473-4910. If you want to write to me, do so. If you need a reply to a question, include a SASE. Till next time.



Parts List		
Number	Description	RS#
Q1	MPS2907 PNP	276-2023
Q2,3	MPS3904 NPN	276-2016
R1	470 OHMS ¼W	271-1317
R2	100 OHMS ¼W	271-1311
R3	27 OHMS ¼W	N/A
R4	220 OHMS	271-1313
R5	10 OHMS	271-1301
C1	.1 UF 25V	272-1069
P1	PERF BOARD	276-162

Equip Your Computer With A Phoneme Speech Synthesizer

This month I have an interesting project. It is using the Votrax SC-01 and the cartridge connector. This is an LSI (large scale integration) chip. With the right interface, this chip will translate certain predetermined data into voice sounds. In other words, it talks. It is a phoneme speech synthesizer. I will show you how to connect it to the Color Computer and how to use it.

The first thing we must do is get all the parts. A parts list appears at the end of this article. You will need, what I call, the standard "kit building tools." This includes soldering iron, solder, pliers, cutters, screwdrivers, knife, drill and bits, hacksaw and your favorite beverage.

There is nothing hard about this project. The regular care in project building will suffice. The Votrax chip is a CMOS

Table 1

Phoneme Code	Phoneme Symbol	Duration (ms)	Example Word
00	EH3	59	<u>ja</u> cket
01	EH2	71	<u>en</u> list
02	EH1	121	<u>heav</u> y
03	PA0	47	no sound
04	DT	47	<u>but</u> ter
05	A2	71	<u>ma</u> de
06	A1	103	<u>ma</u> de
07	ZH	90	<u>az</u> ure
08	AH2	71	<u>honest</u>
09	I3	55	<u>inhibit</u>
10	I2	80	<u>inhibit</u>
11	I1	121	<u>inhibit</u>
12	M	103	<u>mat</u>
13	N	80	<u>sun</u>
14	B	71	<u>bag</u>
15	V	71	<u>yan</u>
16	CH*	71	<u>chip</u>
17	SH	121	<u>shop</u>
18	Z	71	<u>zoo</u>
19	AW1	146	<u>lawful</u>
20	NG	121	<u>thing</u>
21	AH1	146	<u>father</u>
22	OO1	103	<u>look</u> ing
23	OO	185	<u>book</u>
24	L	103	<u>land</u>
25	K	80	<u>trick</u>
26	J*	47	<u>judge</u>
27	H	71	<u>hello</u>
28	G	71	<u>get</u>
29	F	103	<u>fast</u>
30	D	55	<u>paid</u>

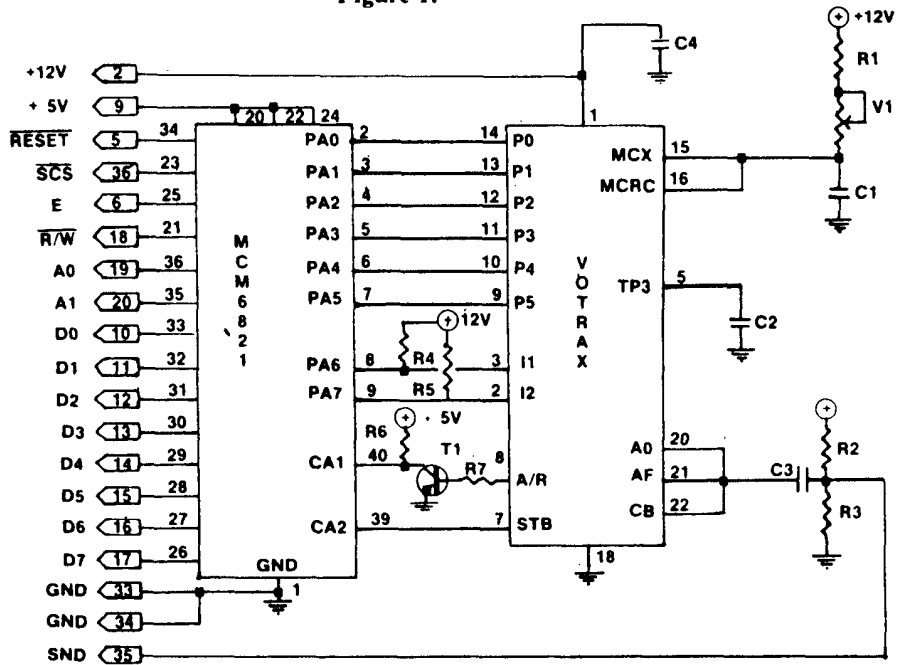
31	S	90	<u>pass</u>
32	A	185	<u>day</u>
33	AY	65	<u>day</u>
34	Y1	80	<u>yard</u>
35	UH3	47	<u>mission</u>
36	AH	250	<u>mop</u>
37	P	103	<u>past</u>
38	O	185	<u>cold</u>
39	I	185	<u>pin</u>
40	U	185	<u>move</u>
41	Y	103	<u>any</u>
42	T	71	<u>tap</u>
43	R	90	<u>red</u>
44	E	185	<u>meet</u>
45	W	80	<u>win</u>
46	AE	185	<u>dad</u>
47	AE1	103	<u>after</u>
48	AW2	90	<u>salty</u>
49	UH2	71	<u>about</u>
50	UH1	103	<u>uncle</u>
51	UH	185	<u>cup</u>
52	O2	80	<u>for</u>
53	O1	121	<u>aboard</u>
54	IU	59	<u>you</u>
55	U1	90	<u>you</u>
56	THV	80	<u>the</u>
57	TH	71	<u>thin</u>
58	ER	146	<u>bird</u>
59	EH	185	<u>get</u>
60	E1	121	<u>be</u>
61	AW	250	<u>call</u>
62	PA1	185	no sound
63	STOP	47	no sound

* T must precede /CH/ to produce CH sound.
D must precede /J/ to produce J sound.

Parts List

ID	Description
IC1	MC621
IC2	VOTRAX SC-01
R1	2K OHMS
R2,R3,R6	4.7K OHMS
R4,R5	10K OHMS
R7	100K OHMS
C1	220 pf
C2	.01 Mf
C3,C4	.1 Mf
T1	2N2222
S1	40-pin socket
S2	22-pin socket
PCB	40-pin edge card

Figure 1:



Schematic of Spectrum Voice Pak, courtesy of Spectrum Projects

chip, so be careful not to zap it with a static charge. Using the schematic in Figure 1, mount and solder all the components in the parts list. Use the sockets for the two IC's. The triangle boxes in the schematic refer to the Color Computer connector. Remember that pin #1 on the Color Computer is the top right-hand side looking into the cartridge slot. This chip needs 12V to operate so that means it will not work with the CoCo 2, unless you have built my 12V supply for the CoCo 2. It also uses the SCS select line, so if you have a disk drive you must use one of the many expansion boxes available. If you have the Radio Shack Multi-Pak Interface, put the voice box in slot 3, the controller in slot 4, and type in this extra line in the BASIC program.

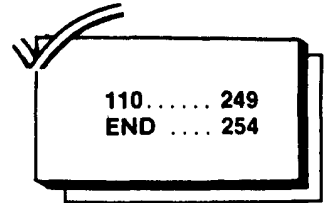
1 POKE 65407,254

Once you are finished mounting the components, type in the short program (listed later), and try it out. Turn the volume of your TV up, because the sound is routed to the sound output of the Color Computer, and it will come out of your TV. Now that your voice box works, here are some details you will need to work on the Votrax chip. This chip phonetically synthesizes continuous speech of unlimited vocabulary. A phoneme is a building block for speech. It is like a single lip movement, like "ohhh" or "ahhh." It is a part of speech. For example, the word hello is made of several phonemes. The first is 'H', next would be 'E', followed by an 'L' and a long 'O'. Together, these phonemes pronounce the word "hello." In order to make a complete sentence, you must break each word down phoneme by phoneme. The SC-01 is capable of reproducing 64 phonemes. Each phoneme is a part of everyday speech. Using all of the 64 phonemes, you can produce almost any speech pattern you wish. Table 1 describes each phoneme, the numeric value, the duration in milliseconds, and an example of the sound it makes.

One more feature the SC-01 has is that it has built in inflection. This is the ability to speed up or slow down the speech in order to add accent to the voice. An example would be when you ask a question. There are four inflections (or speeds). They are invoked by adding one of four values to the phoneme code. The four values are 0, 64, 128, 196. The default is 0, or the slowest speed. The next three speeds are each a little faster than the last.

Okay, once your voice project works, you may want to put it in a small case. An old game pak from Radio Shack will do fine. Trim the PCB so that it will fit in the case. If the posts are in the way, cut them off and glue the pak shut. If you don't have an old pak you can get one from Bob Rosen at Spectrum Projects for \$6. The Votrax SC-01 chip is also available from the same company for \$35. If you don't want to put it together yourself, you can get a complete Votrax package for \$69.95 from him too. To order the Votrax chip or the case from Spectrum Projects, dial (212) 441-2807. In Canada call MICRO R.G.S. at (800) 361-5155.

Til next time, *au revoir*.



The listing:

```

10 CLS : A = 65344 ' VOTRAX LOCA
TION
20 POKEA+1,0:POKEA,255:POKEA+1,5
2 ' INIT PIA
30 POKE65281,180:POKE65283,61:PO
KE65315,60 ' INIT COCO SOUND OUT
PUT
40 X=63:GOSUB200
50 PRINT@200,"VOTRAX SC-01"
60 DATA 27,47,24,52,53,55,62,62,
21,0,9,47,0,12,12,56,60,60
70 DATA 25,25,21,24,58
80 DATA 25,25,50,49,12,37,34,54,
55,42,58
90 DATA 62,62,62,62
100 DATA 6,33,41,14,60,41,31,60,
41,30,60,41,60,41,2,1,29,30,26,6
0,41
110 DATA 6,33,41,42,16,21,0,9,41
,30,26,0,6,33,41,25,0,6,33,41
120 DATA 2,0,35,24,2,1,12,2,1,13
,52,53,55,3
130 DATA 37,60,41,62,25,34,54,55
,55,21,49,58,2,1,31,42,60
140 DATA 33,41,34,54,55,55,15,60
,33,41,30,50,14,35,24,34
150 DATA 54,54,2,1,25,31,31,45,2
1,0,9,41,18,60,41
160 DATA 63
170 FOR I=1 TO 142:READX:GOSUB20
0:NEXT I
180 X=63:GOSUB200
190 END
200 POKEA,X ' STORE DATA
210 POKEA+1,52 ' STROBE ON
220 POKEA+1,60 ' STROBE OFF
230 V=PEEK(A)
240 IF (PEEK(A+1) AND 128) THEN
RETURN ELSE 240

```

Dualing Cassettes

I got the idea for this month's article from someone who gave me a call on a Monday night. He was working on a project that would control the motors of two cassette players and was having some problems with it. We spoke for a while, but I could not figure out what his problem was over the phone. I told him that I would put together one and present it in one of my articles. There is one thing — I cannot for the life of me remember his name. You know who you are, so give me a call and I'll give you credit for this idea.

First we must describe what this project is and what it does. It is what I call a Dual Cassette Controller, which fits in a small ROM pack, and plugs into the CoCo or CoCo 2 expansion port. It has three DIN connectors. One plug fits into your cassette connector in the back of the computer. The other two connectors connect to two tape recorders. This Dual Cassette Controller will enable the user to transfer files from one cassette to another. This could be useful in making backup copies of your software a lot easier than with one cassette. With the proper software, it could allow you to make complete backups of everything on one cassette to another. It could also be useful when sorting or changing ASCII text files. An example would be if you have a telephone list, and someone changed his or her address or telephone number, it would be easier with two cassette recorders to update the file. The next few paragraphs will show you how to build and operate the Dual Cassette Controller.

The first thing to do in this project is to get the parts and

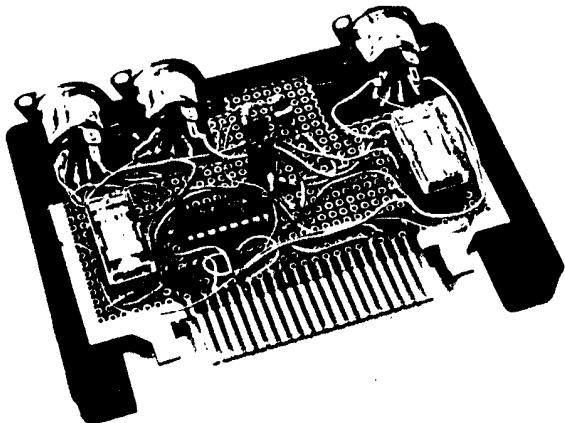
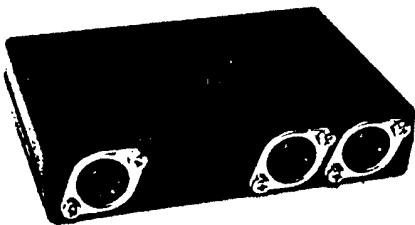
(Tony DiStefano is well known as an early specialist in Color Computer hardware projects. He is one of the acknowledged experts on the "insides" of CoCo.)

tools necessary to construct the Controller. You will find a parts list later on in this article. The tools you will need this time are the "standard tool kit," drill, round file and a sharp knife.

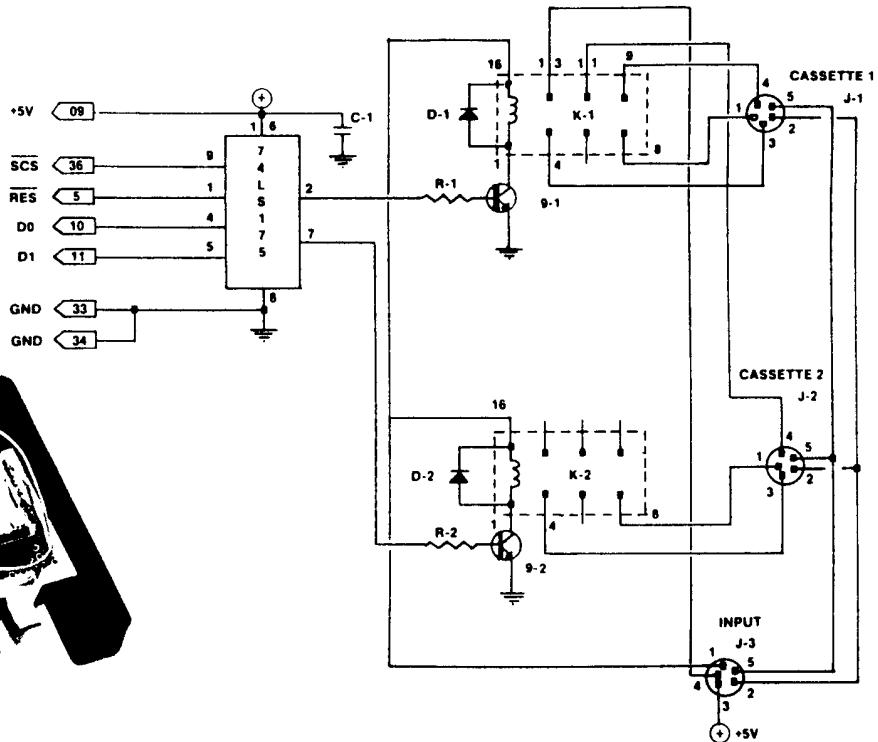
This project is just as much electronic as it is mechanic. It involves cutting, drilling and filing things into shape. It is up to you to make it as nice as you can or want. Halfway into building it I thought of mounting the whole thing inside the computer. Then I thought there are always many ways of modifying your computer to suit your needs. Do it the way you please. I included a few photos to show you how I built my proto-type. You can do it the same way or come up with your own design. However the mechanics are done, the electronics are the same.

Following the schematic, solder all the components together. If you want the thing to fit in a ROM pack case, place the components as shown in the photos. Also, do not use sockets for the relays, it won't fit in the case. From past experiences, there seems to be a difference in Radio Shack part numbers in Canada and the U.S. Some numbers do not always match, so be careful. When you are not sure, use the description to get the part. Use at least a 24-gauge wire for the connections to and from the relays that connect to the motor connections on all the connectors. There are no surprises in the circuit, it is quite simple, only the regular care for static sensitive IC's will do. Remember to clean the PCB when you are finished.

In the "Turn Of The Screw" column by Tony DiStefano in our June 1984 issue, we stated that the schematic of the Spectrum Voice Pak was supplied courtesy of Spectrum Projects. We should add that the schematic is copyrighted by John Kelty of Kelty Engineering.



DUAL CASSETTE CONTROLLER



Mount the three connectors on the end of the case. Drill, cut and file the plastic case until they fit. Then cut the PCB until it fits in the case. Make sure that no wires touch together and all solder joints are solid. Use my photos as a guide.

To try out the controller, follow these simple steps. Turn off the computer. Plug the controller into the computer slot. Plug one end of the DIN to DIN wire into the computer's cassette port. Connect the other end into the controller's input and connect the two cassette recorders into the proper connectors on the controller. Next, turn on the computer. In order to test the relays, type this in:

MOTOR ON ENTER

The internal relay should click on.

POKE 65344,1 ENTER

Relay number 1 should click on.

POKE 65344,0 ENTER

Relay number 1 should click off.

POKE 65344,2 ENTER

Relay number 2 should click on.

POKE 65344,3 ENTER

Both relays should be on. If all this works then the relays work okay. Now try to *CSAVE* and *CLOAD* to each cassette. To access the first cassette you must first:

POKE 65344,1 ENTER

Then all I/O will be through cassette number one. If you want to access cassette number two you must first:

POKE 65344,2 ENTER

That will give you access to the second cassette. *CSAVEs* and *CLOADs* will be done through this cassette. There is one more interesting thing with this controller. If you *POKE 65344,3* and ENTER, you will be able to *CSAVE* to both cassettes. Since both motors are on and the output goes to both recorders, you will get two copies of whatever you *CSAVEd* or *CSAVEMd*. This will not, however work with *CLOADs* because the inputs are switched. With some good machine language code, a user could open two cassette files say, *OPEN "O", #-3, "FILENAME"*. If you want to know where I got that proto-board and case, it was from Micro R.G.S. It is a great proto-board and suits CoCo projects quite well.



Parts List

ID #	Description	RS Part #
U1	74LS175	N/A
R1,R2	470 ohm 1/4w	271-1317
J1,J2,J3	5-Pin DIN Female	274-005
Q1,Q2	2N3904	276-2016
K1,K2	5V Relay DPDT	271-215
D1,D2	1N4004	276-1103
C1	.1uf 10V	272-111
MISC	Proto-board	N/A
	Case	N/A
	16-Pin Socket	276-1998
	5-Pin to 5-Pin wire	42-2151

Popular Misconceptions And Common Problems

In the past three and a half years I have learned much about the Color Computer. Playing and poking around inside I compiled lots of information about how this computer works. I listen to everyone that has something to say about it, in case I learn something new. If I do, I immediately race home and try it out. To see if what I heard or what I saw really works or is true. However, not everything I hear is right. This brings me to this month's topic. I will try to clear up the "hearsay" and "did you know" about the Color Computer. Some of them are started by good ol' Radio Shack and others are started by well known people in the Color Computer circles, but most are started by people who misunderstand something and repeat it to someone else. Nevertheless, where ever they come from, I would like to clear up the ones I am familiar with.

The first one pertains to disk drives and disk controllers. Some believe that the new I.1 disk controller needs and gets

its 12 volts from the power inside the disk drive. That means that the I.1 controller can only work with the newer white drives. This is simply not true. The new I.1 controller does not get 12 volts from the disk drive. The fact is the engineers at Radio Shack redesigned the I.1 disk controller so that it *does not* use 12 volts. They used a different controller chip and data separator in the I.1 controller. They did this so one could use this controller in the newer CoCo 2. You see, the CoCo 2 has no 12 volts inside, so the older controllers would not work with it. This is the way it is. The older I.0 controller will work with the regular CoCo only. The newer I.1 controller will work with both the regular CoCo and the CoCo 2. The older gray disk drives will work with either controller without any modification. The newer white disk drives will work with either controller without any modification.

The next misconception is that some software can damage your hardware. This, in most cases, is not true. The software cannot hurt the hardware. If the software crashes (does not work right), then at most, you could erase a disk if it was not write protected and the door to the disk drive was closed. You will lose what you have in memory, or turn the cassette player on and if it is in the record mode, you could write on top of something important. If you see garbage on the video screen or see the sync break up and the picture tear all across the screen, just turn the computer off, wait for 15 seconds, then turn it on again and all is well. This will not hurt the computer. The only case where I can see a problem is if the software turns the cassette relay on and off repeatedly at a high speed. If you were to leave this condition for an extended length of time, it could burn out the relay. I have never seen this happen to my computer. Another highly unlikely problem could exist with a disk drive. If the software were to bang the read/write head repeatedly to track 0, the head could get out of alignment. But again, you could stop it before any damage could result.

The third misconception involves memory. So many people call me and say, "I just had a 64K upgrade put in my computer. How come when I type *PRINT MEM* I get less than 32K? Did I only get 32K? Where are the other 32K?" I covered this topic in an article last year but the amount of times I hear this question warrants me to explain it again. The CPU inside the CoCo and the CoCo 2 is an MCM6809. This CPU can only access or work with 64K memory total, ROM and RAM total. When you turn on your computer, a total of 32K memory is reserved for BASIC, Extended BASIC, and Disk Extended BASIC. This right away leaves only 32K left for *PRINT MEM*. The rest of the memory difference is being reserved for such things like video area, graphics pages, I/O buffers, and variables. The other 32K of RAM is sleeping. BASIC cannot get to it because it does not know how to wake it up. It takes programs that are written with 64K in mind. Programs that know how to wake up the sleeping 32K are usually advertised as being able to make use of the full 64K. They will perform a test to see how much memory is available and make use of all of it.

The second part of this month's article is about common problems. There are a lot of little quirks that bother the average user about the CoCo. The biggest one I can think about is with disk drives. The ever popular I/O Error. What a nightmare when the project you were working on for hours is lost to an I/O Error. There are a lot of so called "fixes" for I/O Errors, like hiding the directory on track 35 or backups of backups of backups. Then there are those programs that

try to recover your lost files. Don't get me wrong, they are good programs and I did have to recover files myself, but if you have a lot of I/O Errors, it might be wise to take a look at your hardware. I get a lot of letters from people who have these problems.

Here are some good tips on how to prevent disk I/O Errors. The most common cause of errors is the connection between the controller and the computer. The Radio Shack controllers have lead-coated contacts and they get dirty. They oxidize and prevent the signal from going through. Some say to clean the contacts with a soft pink eraser. Others say that it is no good and say to use alcohol and a Q-Tip. I say use both. First the eraser to clean the big dirt and then the alcohol to mop up. It works great!

Some of the older disk drives have problems with speed. The speed drifts and causes I/O Errors because the drive belt slips due to excess oils present in that area. What you have to do is remove the cover and clean the belt. Clean your heads regularly. Finally, remember to always open the drive door whenever you are not doing I/O to disk. If your program crashes, there is no chance that it will garble your disk. Always have the door open when turning your computer on.

The next common problem is the Radio Shack keyboard. Sometimes the older keyboard keys can stick or give double characters. The best way to clean this would be to take it all apart and clean each key one by one. But if you take it apart you will be greeted with a springy surprise. Yes, many little springs are inside the keyboard. A much easier way to clean a key is to squirt a little shot of lighter fluid into the space around the key and quickly press the key several times. Do this again if the problem persists.

The last problem that is common to the CoCo user (especially the old "D" and "E" boards) is in the power supply. The symptoms are strange. At first, it might look like the software has crashed. Then, the screen might go blank . . . all white, no control. Hitting the Reset is no help, but turning it on and off fast sometimes fixes the problem. It sounds like the power switch is defective, but that is not the problem. The current sensing resistor is likely out of tolerance. It is supposed to be a .33 ohm two-watt resistor. The resistance in one case was up to .47. That gave a false reading to the current sensing amplifier. Then the five-volt section of the power supply shut down, thinking that there was a short, causing the computer to fail. The 12 volts to the RF adapter was still on. That gave the blank screen effect. To solve this problem change the resistor, with the same value, of course. On the "D" and "E" board it is R66 and on the "F" board it is R24.

The Halt Pin And Its Function

A while ago I wrote about the pins' functions on the cartridge connector of your computer. One of the pins was the "HALT" pin, which is the center of discussion for this month.

The HALT pin is not one of the most popular pins. Certainly not as popular as, let's say, an address line or a data line. Address and data lines are used continuously while the HALT line can sit idly forever. In fact, if you don't have a disk controller or anything else plugged in the cartridge slot, the HALT line will not be used. The disk drive controller always uses the HALT line to do its I/O.

What does the HALT line do? It does what it says it does — halt. When this line is logically high (five volts), it is inactive. But once the HALT line goes low, at zero volts, many things start happening. The CPU will stop. First of all, the CPU will finish its current instruction, which takes between two to 15 clock cycles, depending on what instruction the CPU was executing. Then the CPU will tristate the address bus and the data bus, which means the CPU will neither input nor output — it

is inactive. Everything stops, however, nothing is lost. When halted, the BA (Bus Available) and the BS (Bus Status) lines will go high. This indicates that the CPU is in the halt state. You don't have to worry about these lines; Radio Shack chose not to use them by not bringing them to the cartridge connector. The CPU registers are all preserved and the RAM (random access memory) is still refreshed. That's the SAM chip's job.

Everything will stay halted until the HALT line returns to a high state. Then

"The HALT line has a multitude of uses. The most useful and practical is to slow down a BASIC listing."

the CPU will continue just as before. While halted, the CPU will not respond to external real-time requests such as the Interrupt Request or the Fast Interrupt Request. The Non-Maskable Inter-

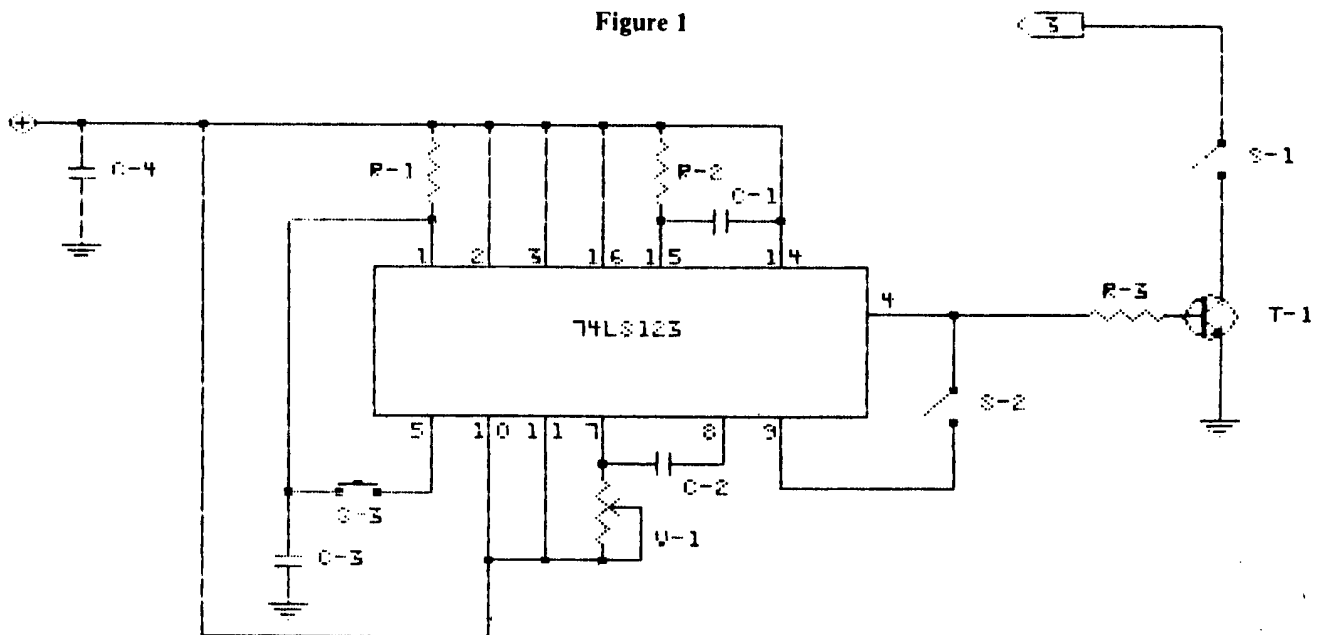
rupt and the Reset will be latched for later request. Stopping the CPU will usually not cause any problems, but under certain conditions, problems can occur. This is when the CPU is involved in critical timing. Examples are cassette or disk I/O; timekeeping or serial I/O like printer; or modem I/O. If the CPU is HALT'ed during these and other timing conditions, loss of data or complete scrambling of data is eminent. Timing loops can be thrown off, so stay away from the halt line when doing I/O or timing.

What could one use the HALT line for if one had control of it? Well, there are a multitude of uses. The most useful and practical is to slow down a BASIC listing. You know, when you do a *LIST* and a long flash of text just streams by? Well, you could slow that down to a reasonable speed using the halt line. Another use is to study, step by step, how the CPU draws graphics. You study the different techniques programmers use to draw and move objects on the Hi-Res graphics screen. A third use is to study how BASIC commands function such as *PRINT* and *SET* and *RESET*.

Now that you know all about the HALT line and what useful things you can do with it, let me show you how to put together a small circuit that will let

(Tony DiStefano is well known as an early specialist in Color Computer hardware projects. He is one of the acknowledged experts on the "insides" of CoCo.)

Figure 1



you control the HALT line. First you will need parts, which are listed in Table 1. You will also need the standard "tool kit" for assembling. It is not a very difficult circuit to put together, just the usual parts. Follow the diagram in Figure 1, and put the circuit together. Use the socket for the IC. When you are finished plug the board into the ROM Pak of the CoCo or any one of the slots of the Multi-Pak Interface. It will also work with the CoCo 2. You might want to put the switches and the pot on another small board with remote wires so that it would be more accessible when using it. Also if the rapid fire mode is too fast or slow, try changing the value of C2. The lower the value, the faster it will go and vice versa. Try a .001 to a .01

capacitor.

With switches 1 and 2 off, turn on the computer. Everything should work normally. Now turn switch 1 on. The cursor should stop. Press the push-button several times. The cursor will flash occasionally. Turn on switch 2. When you push the button, the cursor should start to flash slowly. Turn the potentiometer from one end to the other. The cursor should speed up and slow down. That is your speed control when switch 2 is on. When switch 2 is off, the push button acts like a single stepper. When it is on, it is rapid fire. When switch 1 is off, the whole thing is disabled. The task is complete. I'm sure that you will find many uses for the HALT line.

Table 1

PARTS LIST	
ID	DESCRIPTION
R1,2,3	1K OHMS ½ WATT
V1	500K OHMS POTENTIOMETER
C1	150 PF 10 VOLTS
C2	.005 MF 10 VOLTS
C3,4	.1 MF 10 VOLTS
IC1	74LS123
T1	2N3904
S1,S2	SPST SWITCH
S3	MOMENTARY PUSH ON SWITCH
PCB	PROTO-BOARD (RGS MICRO)
—	16 PIN SOCKET

The Modem To Printer Connection

Of all my projects, the short and fast ones seem to be the most popular. The ones that seem to better the computer and help the user on his quest for good computing are the ones that people call me to thank me for. I also get ideas from these people. For instance, the "Dual Cassette" project was an idea I got from a reader. When I presented this, I had forgotten his name, and wanted him to call me. Well, he did; his name is Lennie James. Thank you, Lennie, for the idea. The basis of this month's article actually came from several people. It is based on the RS-232 port of the Color Computer. The original question was this: Is there a way to connect a printer and a modem together so that everything that comes from the modem can also go to the printer at the same time? The answer is "yes." There are many ways of doing this. Some are very easy and fast, others require a bit more work and money. I'll tell you the theory on how to do it and let you decide on what method to use.

What is RS-232 anyway? The full

name for this is EIA RS-232C. EIA stands for Electronic Industries Association. The EIA RS-232C standard defines the interfacing between data terminal equipment and data communications equipment employing serial binary data interchange. Electrical signal and mechanical aspects of the interface are well specified. The complete RS-232C interface consists of 25 data lines. This would seem to be enough signals for a complex parallel communication line, but many of the 25 lines are very specialized and a few are undefined. Most computer terminals only require from three to five of these lines to be operational. Table 1 briefly describes all 25 of the defined lines.

Table 1

PIN	DESCRIPTION
1	Protective Ground
2	Transmitted Data
3	Received Data
4	Request to Send
5	Clear to Send
6	Data Set Ready
7	Signal Ground
8	Received Line Signal Detector
9	Unassigned
10	Unassigned
11	Unassigned

12	Sec. Rec'd Line Sig. Detector
13	Sec. Clear to Send
14	Sec. Transmitted Data
15	Transmission Signal Element Timing
16	Sec. Received Data
17	Receiver Signal Element Timing
18	Unassigned
19	Sec. Request to Send
20	Data Terminal Ready
21	Signal Quality Detector
22	Ring Indicator
23	Data Signal Rate Selector
24	Transmit Signal Element Timing
25	Unassigned

Table 2

PIN	DESCRIPTION
1	CD — Status Input Line
2	RS232IN — Serial Data Input
3	GROUND — Zero Voltage Reference
4	RS232OUT — Serial Data Out

(Tony DiStefano is well known as an early specialist in Color Computer hardware projects. He is one of the acknowledged experts on the "insides" of CoCo.)

The Color Computer uses only four of these lines. They are the four most used in small computers. Table 2 shows the pin and description for the Color Computer version of the RS-232. Pin 1 on the computer is equal to pin 5 or pin 8 on the EIA RS-232C; pin 2 on the computer is equal to pin 3; pin 3 on the computer is equal to pin 7; and pin 4 on the computer is equal to pin 2.

So much for the theory, now for the good part. The secret to this is to connect the Transmit (Serial Output) of the modem to the Receive (Serial Input) of the printer. Now there are many ways to do this. It all depends on what kind of equipment you have. If you are one who just unplugs your printer cable to plug in your modem, you will have the most to do. If you have one of the several switchers available for your modem and printer, all you need is a switch and a piece of wire.

Step 1

Follow these instructions if you have a switcher. If you don't have an SPST switch, RS #275-624 is good and small. First you have to take the switcher apart. You will need the right screwdriver. After the switcher is apart, locate the connector that the modem connects to. Solder one end of a piece of wire to pin 2 of that connector. Solder the other end of this wire to one end of a SPST switch. Solder one end of another piece of wire to the other end of the switch. Now locate the connector that the printer connects to. Solder the last end of wire to pin 4 of that connector. Mount the new switch somewhere in the switcher. Close up the switcher. I'll show you how to use it later.

Step 2

Follow these instructions if you do not have a switcher. Undo the modem

connector that plugs into the computer. Solder a wire to pin 3 in the connector. Using a piece of tape, label this wire "G" for ground. Solder another wire to pin 2 of the connector. Reassemble the connector. Undo the printer connector that plugs into the computer. Solder a wire to pin 3 in the connector. Label this wire "G" for ground. Solder another wire to pin 4 of the connector. Reassemble the connector. Solder the two wires labeled G together. Solder the other two wires to each side of an SPST switch. Mount the switch any way you want.

"Is there a way to connect a printer and a modem together so that everything that comes from the modem can also go to the printer at the same time? The answer is 'yes.'"

In order that the printer prints all that comes in on the modem, the printer parameters must be set correctly. Most modem communications use 300 Baud. That means your printer must be set to 300 Baud. Other parameters, like seven or eight bits, even, odd or no parity, must also be set right. That will depend on what parameters the host computer is using. The fact is that all these parameters must be looked into before the printer will function right. Another thing I should mention is that the printer may or may not print what you type. That depends if you are working in full or half duplex, you will not see on paper what you type; with full duplex you will see it. At certain times you may not want to see what you type in, so just change to

half duplex if the host computer will allow you.

The next thing you must do is set up the wiring correctly. If you are using Step 1, then you must set the switch you installed to the "on" position and the switcher to the modem side. When you want to use the printer alone, make sure that the switch is in the "off" position and the switcher is set to the printer side. If you followed Step 2, then plug in the modem connector and turn the switch on. When you want to use the printer, turn the switch off and plug the printer connector on.

During normal printing, there is handshaking going on between the printer and the computer. That is, before the printer sends out a character to the printer, the computer checks if the printer is busy. If it is, the computer will wait until the printer is ready. In modem communication, there is no such handshaking. That means if the printer is busy and the modem transmits a character, the printer will miss that character and not print it. This is especially true when the printer is doing a carriage return or line feed. If your printer has an input buffer and can print faster than about 30 characters per second (300 Baud) or 120 characters per second (1200 Baud) you will not miss any characters. Another way to avoid missing characters is if the host computer can be programmed to wait after every carriage return; the printer would have time to catch up.

If you have problems with one of my projects or you want to discuss one of your own projects, I have reserved Monday nights for this. I'll be happy to talk with you if you call me then. The number to call is (514) 473-4910. But limit the calls to Monday nights, any other time is forbidden fruit.

Well, that is it for this time, good modem printing. ☺

Force A Cold Start From Reset With This Simple Project

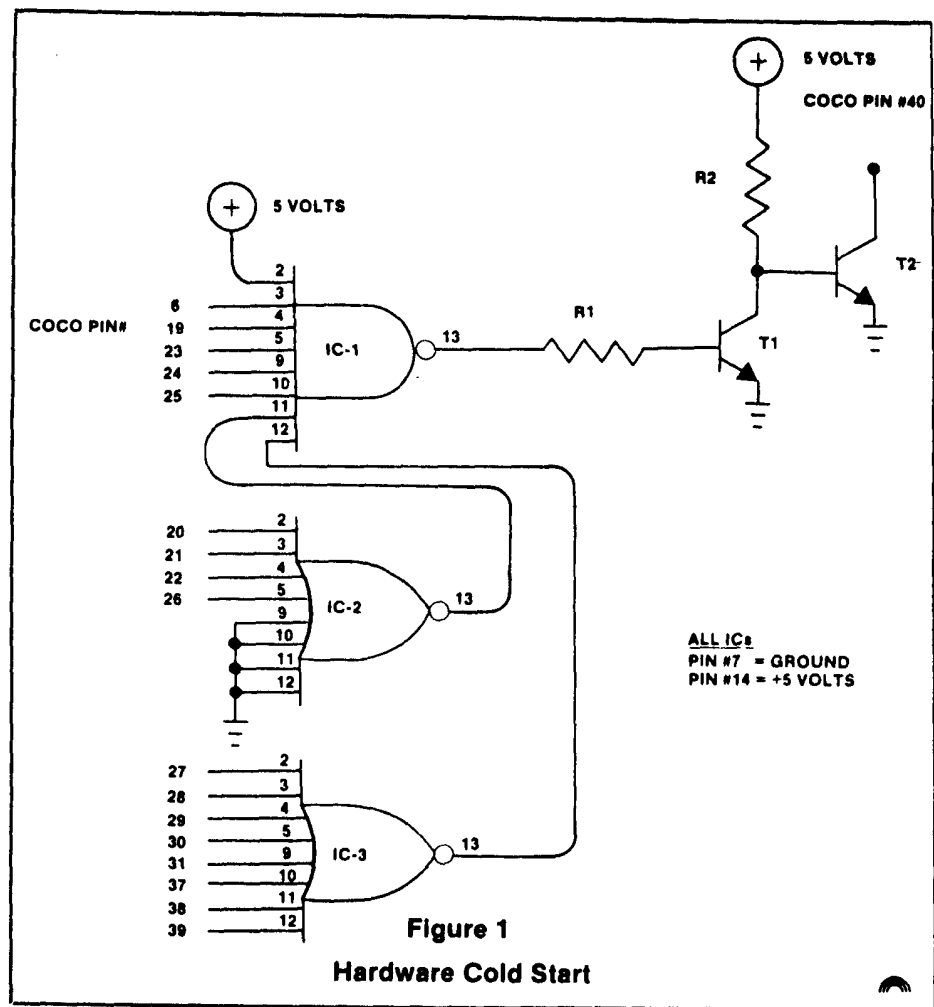
Does this ever happen to you? You are playing a nice game, or heavy into some database. Then, you get tired and want to quit. So, you hit the Reset button in hopes of clearing what is in memory, and the software appears on the screen again. So you hit the Reset button again in disbelief but to no avail, it comes right back. There is no way of getting out of it. You then proceed to a power down routine. First you open the doors to all

your drives, to avoid destroying a disk. Then you turn the computer off. Count to 15 and then turn the computer back on. Next you close the doors to the drives in use. It happens to me all the time, especially when I use protected software. Well, I decided to do something about it.

Before I get into the construction part of this article, a little theory on what is happening. When someone first turns on the computer, it does what I call "a cold start routine." It does things like check how much memory is present and initializes the PIA and SAM chips. It then initializes all the necessary pointers, etc. Before it turns control over to the user by putting the OK prompt on the screen, it puts the value \$55 (\$ denotes a Hex number) or 85 in decimal in location \$71, 113 in decimal. But first it checks to see if it has been on before (if it has done this initializing routine before). It does this by seeing if memory location \$71 or 113 in decimal contains \$55 or 85 in decimal. If it does, it means the computer has already been on before the Reset button was pressed and that it does not have to do a cold start. Instead, it does a warm start. This warm start first initializes the PIAs and SAM chips only and then jumps to the warm start vector. The warm start vector is located in memory locations \$72 and \$73, 114 and 115 in decimal.

You can see that if you were to change the reset vector to your own program, and made sure that \$71 contained \$55, then, if someone were to press the Reset button, control of the computer would not return to the user's program, but rather the program pointed to by the reset vector. This is how a program can come back after you press the Reset. NOP is the first byte to which the reset vector must point. That is \$12, 18 in decimal. That is another condition of a warm start. The BASIC ROM checks for that.

Now that we know what the computer does when we hit the Reset button, how do we change these conditions to suit our own needs? Well, it's simple, in theory anyway. What if we were to deny the CPU access to that particular byte (\$71)? If the computer could not read or write to that byte, then when it made its test, it would never see \$55 and always do a cold start. So much for theory, this is the real world. The makers of the Color Computer were kind (or smart) enough to put a "MEMORY DISABLE" or better known as the SLEND



pin, on the 40-pin bus connector. This pin is normally high (five volts), and when some device or other pulls it low (0 volts), all forms of memory chips (ROM, RAM and PIAs) are disabled. I will be using this pin in conjunction with my circuit to deny access to memory location \$71 to the CPU.

The actual circuit is in Figure 1 and the parts list is in Table I. Some of these parts are not available at your local Radio Shack. You will have to go to a more specialized electronic store or to a mail order store like Active Electronics or JDR Electronics. You can get a complete parts kit from RGS Micro Inc. Just ask for the "Turn of the Screw" hardware kit # 1. The USA order line is 800-361-4970 and the Canadian line is 800-361-5338. Also look in this magazine for their ad. The chips used in this circuit are called CMOS (Complementary Metal Oxide Semi-conductor) chips and they are quite delicate. The slightest static charge can permanently damage

the chip. The shock you receive from rubbing your feet on a carpet is enough to kill a CMOS chip if you were to come in contact with it. Make sure you and your work are grounded before you plug the chips into their sockets. Leave the chips in their original package until you are ready to plug the computer in.

The construction is simple. The regular Tool Kit will do. Just connect the wires to the right points. The Proto-Board I like to use is made by RGS Micro. There are three capacitors in this circuit, used for power supply decoupling. Place them close to each chip on the board. As usual, clean the board after all is done. Place the switch where it is easily accessed. If you have a Multi-Pak Interface like I do, it is better to mount the switch upside down. This circuit will work for any board version (CoCo 2 also) except the "F" board; a small modification to this computer version is needed. If you have this board, open the computer and cut a

capacitor. It is labeled C77. This capacitor is tied to the SLEND line and ground. Cutting this capacitor should not interfere with the normal operation of the computer.

Forcing a cold start is now quite easy. Hold down the switch with one hand. Hit and release the Reset button with the other. When the computer returns to power on condition, release the switch, it's as easy as that. Any time you don't

want a cold start (a normal reset), just don't hold down the switch and you will get a normal reset condition.

NOTE: There is an error in last month's "Halt Pin And Its Functions" schematic. Pin #8 should read Pin #6 and a Pin #8 go ground should be added.

**Table 1
Parts List**

Quantity	ID #	Description	RS Part#
1	IC-1	CD4068	N/A
2	IC-2,3	CD4078	N/A
1	R-1	1000 OHMS 1/4W	271-1321
1	R-2	100 OHMS 1/4W	271-1311
2	T-1,2	MPS3904 or MPS222A	276-2016 276-2009
3	C-1,2,3	.1 uF CAPACITOR	272-1053
3	-	14 PIN SOCKETS	276-1999
1	-	PROTO-BOARD	N/A

Lights! Camera! CoCo!

This is an enlightening project which involves lights. That's right, a computer controlled light show. This could be used to light up your Christmas tree, brighten up your house or porch, or even change your den into a disco. You know those strings of lights you can buy at Christmas time that come in sets of 20 or 30? They are perfect to use.

Normally I would now start to describe how to put the project together, get the parts and run the thing, but one of my friends, Mike Schmidt, told me that I would do well to explain the theory of how my projects work. Well here goes, a little explanation goes a long way into understanding how the things work.

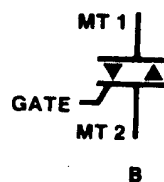
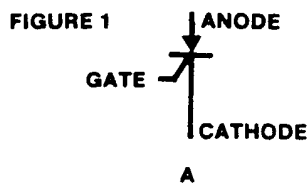
The heart of the project is based on a nifty little chip called a "thyristor." It is better known as a triac. To fully understand a triac, one must first look at an SCR (Silicon-Controlled-Rectifier). Figure 1 displays the schematic diagram of an SCR.

There are three parts to an SCR. The anode, the cathode and the gate. As you can see by the diagram, it doesn't look like more than a diode with another wire going to it. Well, that's basically what it is. The main part of it is a diode, but this diode does not conduct in any direction. It is an open circuit capable of withstanding rated voltage until triggered. That is where the gate comes in. When a small current is applied to the gate, the current path of the diode part of the SRC becomes low-impedance in one direction and remains so, even after the trigger source current is removed. It will remain so until current through the path stops or is reduced below a minimum "holding" level. An SCR is useful for DC and half-wave AC applications.

Figure 2 shows the diagram of a triac. It looks just like two SCR's back to back. In fact, a triac is nothing more than a bidirectional thyristor. A single trigger source turns the device on for load current in either direction. Since

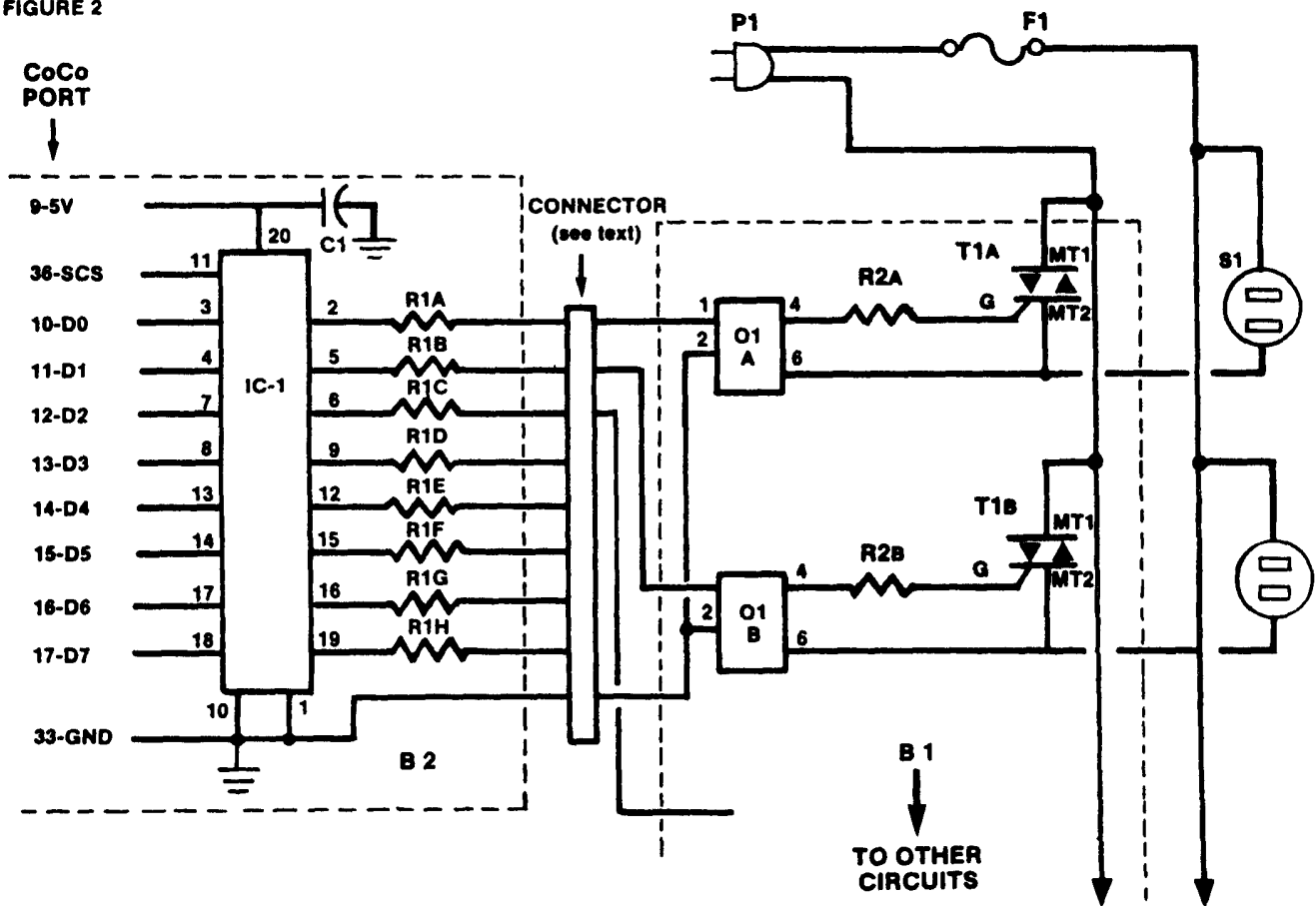
they conduct in both directions, triacs are useful in AC power applications that require full source power control capability to be applied to the load. This capability is what we need in this project. In short, a triac can be described as an electronic switch. It can also be used as a variable control switch, but that capability will not be used in this project.

The Radio Shack Optocoupler is a special type of triac device. Instead of the normal gate controlled trigger, it has an optoisolator device connected to the gate. This is important to us because high voltage like the AC coming from the wall is very dangerous to a low voltage computer. Even the slightest spike of noise can destroy a computer. The optoisolator part of this device will protect the high voltage from coming close to your computer. Only one problem, the current handling capabilities of this device is too limited to be useful. So we'll use it to trigger the gate of a more powerful triac. The triac, in series with a load (our lights) and the AC from the wall, will complete a circuit. Before, I told you that a triac is an electronic switch. With the right signal to the Optocoupler, we can control the load



(Tony DiStefano is well known as an early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.)

FIGURE 2



(our lights) on and off.

We now know that the right signal to the Optocoupler can turn our lights on and off. What is the right signal? A small current, say, from a computer, is quite enough. The rest is quite simple. One eight-bit latch to control the Optocoupler and eight resistors and we are off. The rest is just construction. There is a parts list in Table 1.

All the parts that have a quantity of "*" need explanation. You do not need to make all eight of the triac circuit. It all depends on your use. If all you want is a light chaser, then you need only three of each part. If you want to do something more elaborate, you may want to construct all eight of the triac circuits. The maximum is, of course, eight. More is possible but requires more circuitry. I don't think there is the need for more, but if there is, write to me for details. As it is, the eight-bit latch is memory-mapped at \$FF40 (65433 in decimal), so the use of this with a disk drive is impossible unless you have one of the expansion interfaces available on the market.

Table 1
Parts List

ID	Quantity	Description	Available At
IC-1	1	74LS374	Electronics Store
C1	1	.1 UF 25WVDC	RS
F1	1	10 amp Fuse & holder	RS
R1	*	220 ohms 1/4w	RS
R2	*	150 ohms 1/4w	RS
O1	*	Optocoupler	RS #276-134
T1	*	Triac	RS #276-1001
H1	*	Heatsink	RS #276-1363
S1	*	AC socket	Hardware Store
P1	1	AC Plug	Hardware Store
B1	1	Proto-Board Main-Board	RS #276-161
B2	1	Proto-Board Computer-Side	R.G.S Micro

Misc.: Wire, connectors, sockets, solder, mounting hardware, plastic project box.

This project is basically in two parts. The first part is the computer side. The only parts that go on the proto-board (B2) are the latch, resistors and the capacitor. What will leave this board is a ground wire and one wire for every triac circuit you need. You may connect the two boards together directly or use a connector. What connector you use depends on how many wires you use. Refer to the Radio Shack catalog for the right connector.

The second part of the project is the main board (B1). It consists of all the remaining parts. There is enough room on the board to fit all eight triac circuits. There is not much to this part, just examine photo 1 for placement of all the parts and follow the circuit.

Before trying this, you should run a few tests. Plug in all the ICs except the 74LS374. Plug in the control box and the lights. None of the lights should be on. If some or all of the lights are on, turn everything off and check your work. Next, take a little piece of wire and jumper pin 1 to pin 20, 3, 4, 7, 8, 11, 12, 15, 16 — one at a time. As you do

this each light should go on. If this is OK, turn everything off and plug in the last chip. To see if all is OK, turn everything on. All the lights should be off. Type *POKE &HFF40,255*, or *POKE 65433,255*. The *65433 (&HFF40* in Hex) is the control byte. The lights should go on. *POKEing* a zero into the same location should turn the lights off. The short listings provided will give you an example of what you can do with the lights.

The last step is how to control each light separately. *POKEing* a zero into the control byte will turn off all the lights. Each of the eight lights is controlled by one bit. The first bit controls the first light, the second bit controls the second light, and so on.

Table 2 shows the decimal value of each light. To have any light on, just poke the decimal value of the light

number into the control byte. If you want more than one light on, you must add the decimal values of each light. Example, if you want light 2 and light 6 on, you must do $2 + 32 = 34$. *POKE 34* into the control byte. I wrote a little program in BASIC to give you an example of what you can do with these lights.

Table 2

POKE value	Light to turn on
1	1
2	2
4	3
8	4
16	5
32	6
64	7
128	8

There are a few things to remember, though. Each individual triac circuit load (light or set of lights) must not exceed 400 watts and the total power must not exceed 1200 watts. To get the chaser effect, you need just three triac circuits and three sets of lights. Arrange the lights in parallel and tie them together so that the sequence of lights goes 1, 2, 3 . . . 1, 2, 3 . . . 1, 2, 3. *RUN* the chaser program and, there you have it.

It has been brought to my attention that there seems to be a problem with my parallel printer adapter. The problem is with the grounding of pin 18. While on my Epson printer, I have no problems, on most printers there is a positive voltage on this pin. Connecting this in to ground can cause damage to the printer. To solve this do not ground pin 18 in the output connector. ☺

An Introduction To The Inside Of The CoCo 2

My, doesn't time go fast? I can't believe I've been writing for RAINBOW for two years now.

January being THE RAINBOW's Beginners issue, I decided to introduce the novice to the inside world of the Color Computer. The latest CoCo 2 is the newest Color Computer to be introduced by Tandy. It is different inside from the old CoCo 2. You can tell the difference by the shield covering the power transformer. Though it functions the same, the insides of this CoCo are very different (again!). More on that later.

Before we get on our way, let me mention that I just came back from my second RAINBOWfest. I must say that these shows are great. I found THE RAINBOW staff to be very friendly and helpful. It is amazing to see that much enthusiasm generated about the Color Computer. Chances are I'll see some of you at the next RAINBOWfest, too, in California. Stop in and say hello. Look for me at the R.G.S. Micro booth.



Now, let's look into this little thing, but remember, opening your computer might void your warranty. Radio Shack only warranties the computer for three months, so after that you are on your own, anyway. First of all, *never* open the computer with the power on. Now that that's said, let us continue.

To open your CoCo, use the following

procedure. Place the computer upside down on a towel (or other soft surface) on a clean work table. Remove the four screws (one in each corner) with a medium-sized Phillips screwdriver. There is one more screw to remove; it is behind the little sticker that says "Opening case will void warranty. See owner's manual for warranty informa-

tion." You must break this seal to remove the last screw. That is how Radio Shack can tell if you have opened it. Just push the screwdriver through the center of the sticker; it will give way to a hole. Some of the CoCo 2s may have a sixth screw on the other side. Remove the last screw. Turn the

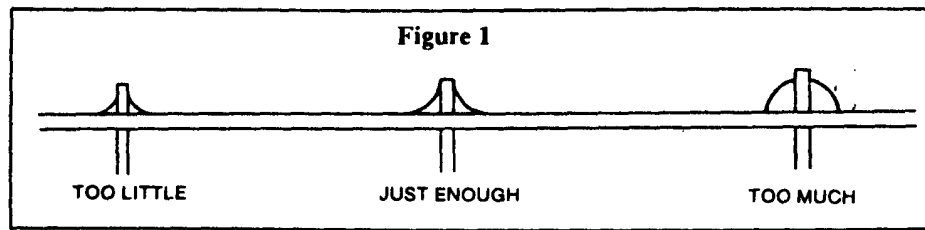


TABLE 1

IC #	Name	Description
1	SC77527	SALT Power supply and RS-232 CHIP
2	MC6821P	PIA Peripheral interface adapter
3	SC77526	DAC Digital to analog converter
4	NE555D	Timer for color burst in PMODE 4
5	74LS273	Octal D-Type Flip-flop
6	74LS244	Octal Buffer Driver
7	SC67331P	IIA Industrial interface adapter
8	MC6847P	VDG Video Display Generator
9	MC6809EP	CPU Central Processing Unit.
10	74LS02	Quad 2-input Nor Gate
11	74LS138	3 to 8 Decoder Chip
12	8040364B	ROM BASIC 1.2
13	8042364A	ROM EX BASIC 1.1
14-21	8040517	16K DRAM Dynamic Random Access Memory
22	MC6883P	SAM Synchronous Address Multiplexer

Beginners Project Parts List

Quantity	Description	Radio Shack #
1	LED	276-068 or 276-069 or 276-073
1	RESISTOR 1k ohms	271-8023

computer back right side up, and gather up the screws that drop out. Grab the top cover of the computer and pull it off. Wow! Look at all those things. The components marked with the letter 'U' (or 'IC' in the case of the newest CoCo) are known as ICs (Integrated Circuits). Table 1 labels all the ICs used in the computer and gives a short description of each.

Some of the components that make up the CoCo are very sensitive to static electricity. You must be careful not to

zap (permanent damage caused by static discharge) a chip by touching the pins with your fingers. If you must touch a chip, always touch a ground point with your fingers first. This will discharge any static your body might be carrying to ground. A good ground point to touch is the RF adapter. That is the big metal can sitting to the left, where you plug in the TV wire. Another point is one of the metal clips that hold the bottom shield to the main PCB (Printed Circuit Board). You will find these clips all around the edge of the PCB.

Now that we have seen the insides of the CoCo and are a bit more familiar with its parts, let's do something to it. About the simplest thing we can do is add a pilot light. It is not hard, and if you take it one step at a time, anyone will be able to do it, and the good thing about it is that it costs less than \$1. By the way, this pilot light will work on any version, not just the CoCo 2. Before you plunge into this though, if you do not have any soldering experience, practice on something else first. To do this, you will need a soldering iron. A low power, medium or fine tip soldering iron will do. The solder to use must be a rosin core and not too thick. Radio Shack sells both at a reasonable price. If you have never handled a soldering iron before, get Radio Shack's proto-board and practice on it first.

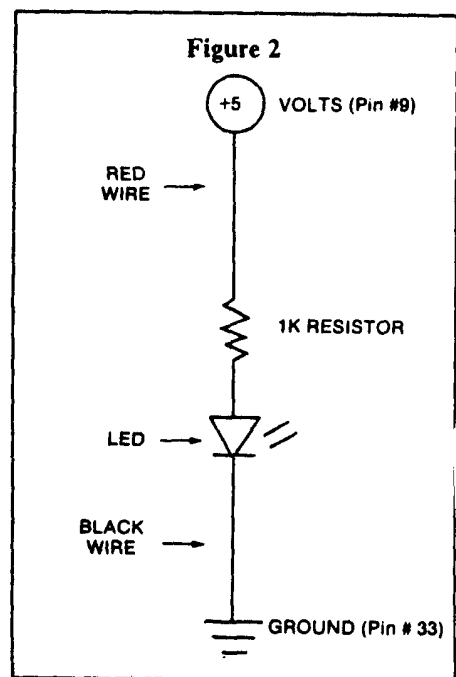
Here are the step-by-step instructions on how to solder:

- 1) Make sure that your soldering tip is clean and hot. A wet sponge is great to clean the tip.
- 2) Secure the component to the PCB.
- 3) Heat the component and the PCB with the iron.
- 4) Touch the end of the solder to the component. My personal habit is to position the solder so that it will touch the iron, component and PCB at the same time.
- 5) When enough solder flows, remove the solder.

- 6) Remove the iron from the joint.
- 7) Wait until it cools before moving the component or the PCB.

To make a good joint takes practice. To put the right amount of solder also takes practice. Too much or too little could result in a bad connection. Examine Figure 1, and notice the difference between too little and too much solder. Sometimes a bad connection can be turned into a good connection just by heating up the joint again. After it cools, the joint should be shiny and smooth. Practice several times until you get the hang of it. There is one more thing to remember; after all the soldering is done, clean the PCB with Radio Shack rosin cleaner-remover.

Now that you feel more at ease with soldering, it is time to put your newly acquired talent to work. Yes, the pilot light. There are only four parts to this project. The LED, a 1K (K=1000) ohm 1/2 watt resistor, and two short lengths of colored wire (preferably red and black). That is it. Examine the schematic



in Figure 2. This is a diagram on how the components connect together and to the computer. The first thing to do is mount the LED. You must decide where to put it. After that, you must check that when mounted, it does not interfere with the normal operation of the computer, i.e., short out or lean on other components and does not prevent the cover from fitting properly.

Mount the LED by drilling a 1/4-inch hole where the LED is to be mounted. Cut both sides of the resistor leads to about 1/4 inch. Solder one side of the resistor to the long end of the LED. Solder one end of the red wire to the other end of the resistor. Solder one end of the black wire to the other (short)

end of the LED. Twist the two wires together lightly and cut them about 18 inches long. This should be long enough to have the cover out of your way if ever you want to open the computer again.

Now, solder the other end of the red wire to inside of pin 9 of the edge connector. That is the five volts side. How do you get to pin 9? Simple, just start counting from the end closest to the back of the computer. All the top pins are odd numbered, so count 1, 3, 5, 7, 9. Make sure that you don't short out two pins with the solder. Finally, solder the black wire to pin 33, count that one the same way. Pin 33 is the ground return pin.

Place the cover on top of the computer (without the screws for now) and turn the computer on. The LED should turn on. If not, chances are that you got the wires to the LED reversed. In that case, unsolder the resistor and the black wire to the LED and resolder them the other way. Otherwise, you should not have any problems. Tuck the wire in the cover and place the cover back on. Make sure that the wire does not stick out and that the keyboard is sitting on the pegs properly. Turn the computer over and replace the screws. There you are, your first modification to your computer. Now doesn't that make your day?



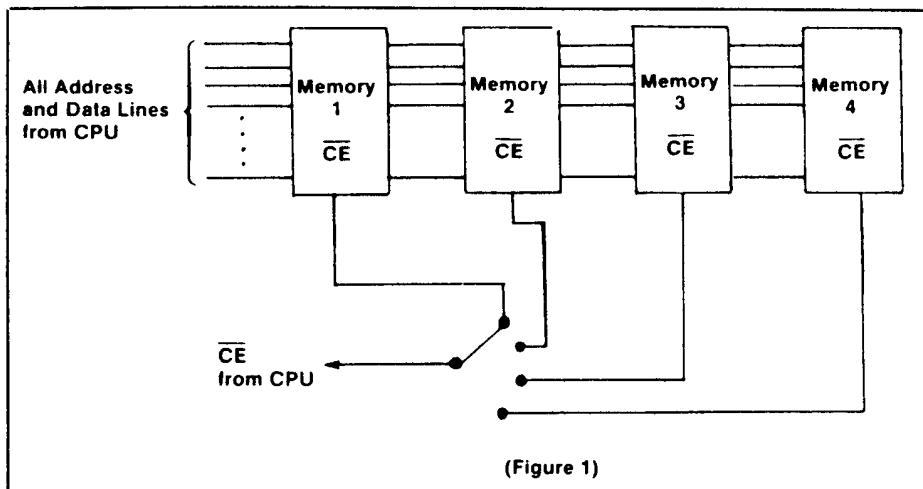
A Look At How The Multi-Pak Interface Works

This month we'll be looking at what makes Radio Shack's Multi-Pak Interface (MPI for short) tick, and finish off by adding a little LED numeric display to tell you what slot is active.

First off, a little background on the memory map of the Color Computer is necessary. Judging by the amount of questions I get, the concept of a "memory map" is very confusing to many. Hopefully, after reading this article, the memory map for the Color Computer will be better understood by all.

The CPU in this computer is the MC6809. It has 16 address lines. In binary numbers, 16 bits can have 65,536 different combinations, or 2 to the power of 16. That means the CPU can directly access 65,536 (better known as 64K) bytes of memory. The key word here is "directly." At any one time, the CPU will read or write within this boundary, but there is no rule that says we can't fool the CPU into accessing

(Tony DiStefano is well known as an early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.)



more. To the CPU, it looks like only 64K; to the user the amount of memory the CPU can access can be almost limitless. The secret (not a very big one) is bank switching.

A memory chip, be it RAM, ROM, EPROM or whatever, has what is known as a chip enable (CE for short) pin. This pin activates the chip for a read or a write. When this pin is not activated, the chip becomes invisible to the CPU; it is as if it was not there.

Now, think of several chips all in parallel, except for the CE pin. Put all

the CE chips on a switch so you can select one at a time (see Figure 1). Changing the switch would mean whatever memory chip was connected by the chip would be activated. This technique allows the user to have access to more than 64K of memory — how much more depends on how many switches you have.

Let's take this one step further. Instead of the manual switch, as in Figure 1, an electronic switch is put in (see Figure 2) and if this electronic switch could be controlled by the

computer, it could switch to different chips all by itself. That way, the CPU could actually access more than 64K. All the CPU would have to do is change the electronic select switch.

This is done, of course, in software. The software must know there is more than 64K online. It must also know how to access this memory in reference to where the switches are. This is basically what the Multi-Pak Interface is — an extension of the CPU's memory capacity. It comes complete with mechanical and electrical switches, along with everything else you need to make it work, like a power supply, buffers, wires and connectors, etc.

Now that we know what it can do, let's look at how it does it. In order to understand how the Multi-Pak works, an understanding of the Color Computer memory map is necessary. Note that all versions of the CoCo and CoCo 2 have the same memory map. (Figure 3 shows the memory map.) This is a hardware memory map rather than a software map. The hardware map shows what chips are where and what areas are reserved for them. A software map would show what variables are where, i.e., printer Baud rate, input hook, cassette buffer and so on. Right now we are interested in the hardware map.

The following is a point by point description of the memory map as it is when you turn on the computer. The map can deviate from this with certain commands to the SAM (Synchronous Address Multiplexer) chip, but these are the default settings (on power up). The "\$" denotes a Hex number.

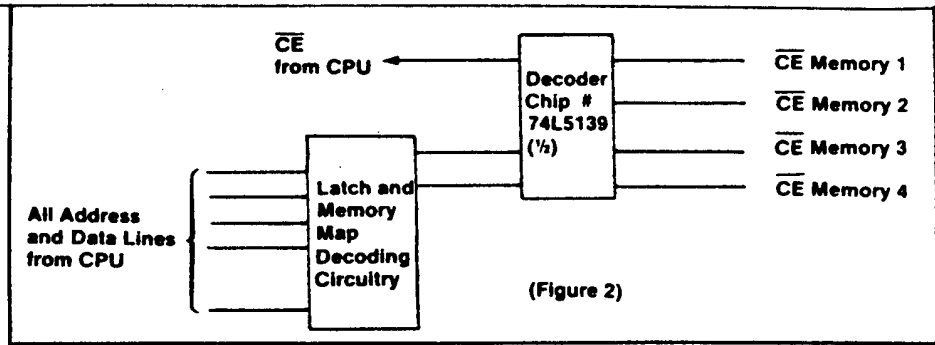
1) 0 to 32767 (\$0-\$7FFF) — This area uses the internal RAM chips. They can be one to two banks of 4K, or 16K DRAM (Dynamic Random Access Memory), or 1/2 of 64K DRAM.

2) 32768 to 40959 (\$8000-\$9FFF) — This area uses an internal 8K * 8 ROM chip. This space is usually taken up by Extended BASIC.

3) 40960 to 49151 (\$A000-\$BFFF) — This area uses another internal 8K * 8 ROM chip. This space is occupied by Color BASIC.

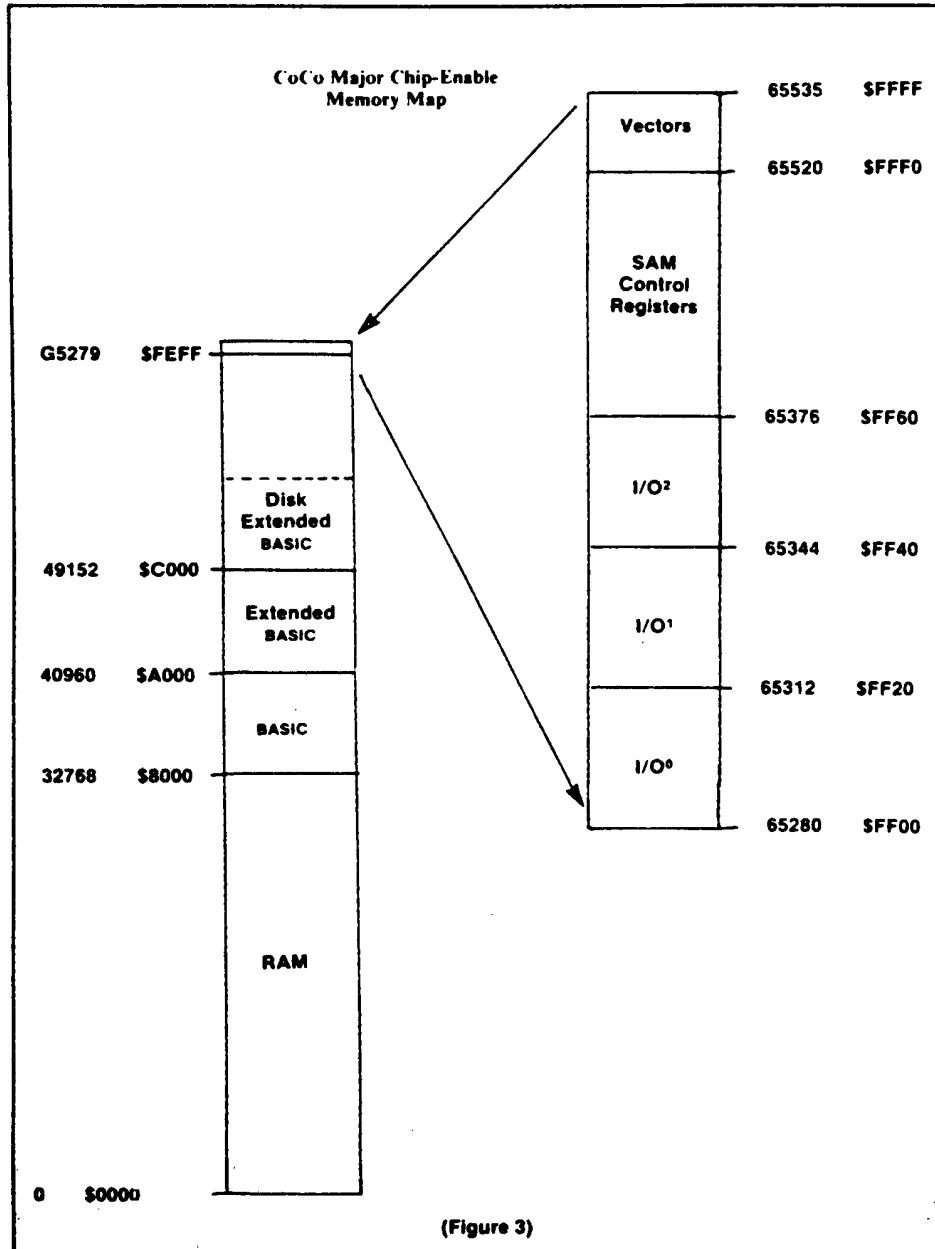
4) 49152 to 65279 (\$C000-\$FEFF) — This area is 16128 (\$3F00) long. It is one page (page = 256 or \$100) less than 16K. This area is reserved for external memory. It is accessible via the cartridge connector on the side of the computer. More on this later.

5) 65280 to 65311 (\$FF00-\$FF1F)



— This area is normally used as an I/O port. It is used to control a PIA (Peripheral Interface Adapter). This PIA is connected to the keyboard, analog MUX select lines, horizontal and vertical sync interrupt, joysticks and buttons.

6) 65312 to 65343 (\$FF20-\$FF3F) — This area is another internal I/O port. The second PIA in this computer, it controls the 6-bit D/A, cassette I/O, RS-232 I/O, RAM size, motor control, sound enable, single bit sound output, graphics mode control and



cartridge interrupt input.

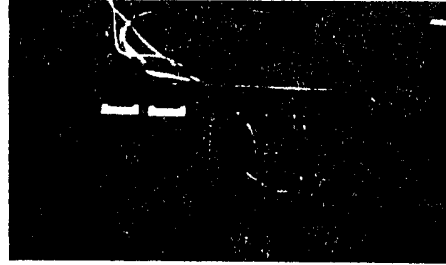
8) 65344 to 65375 (\$FF40-\$FF5F)
 — This area is the third I/O port and is reserved for external use. It is accessible via the cartridge connector on the side of the computer. More on this later.

9) 65376 to 65519 (\$FF60-\$FFEF?)
 — This area controls the SAM chip. The SAM chip generates all the system timing and all of the device selection.

10) 65520 to 65535 (\$FFF0-\$FFFF)
 — Finally, this area is the indirect pointers to the CPU interrupt vectors. Each pointer is two bytes long. Starting from the top, they are: Reset, NMI, SWI, IRQ, FIRQ, SWI2, SWI3 and the last one is Reserved. This area is controlled by the SAM chip and whenever it is accessed, the SAM chip will re-route (re-map) it to 49151 (\$BFFF), the top of the Color BASIC area. The reason for this is the CPU must use these vectors, and the only ROM that definitely comes with the computer is this one.

As you can see from the map, the areas that will concern the MPI are #4 and #8. They are accessible through the cartridge port.

Let's start with #4. The most common use for this area is the ROM-Pak. All of Radio Shack ROM-Paks use this area, however, not all of them use the whole 16K area available. Some use 2K or 4K, but most use 8K. In the case of the disk drive system, the software

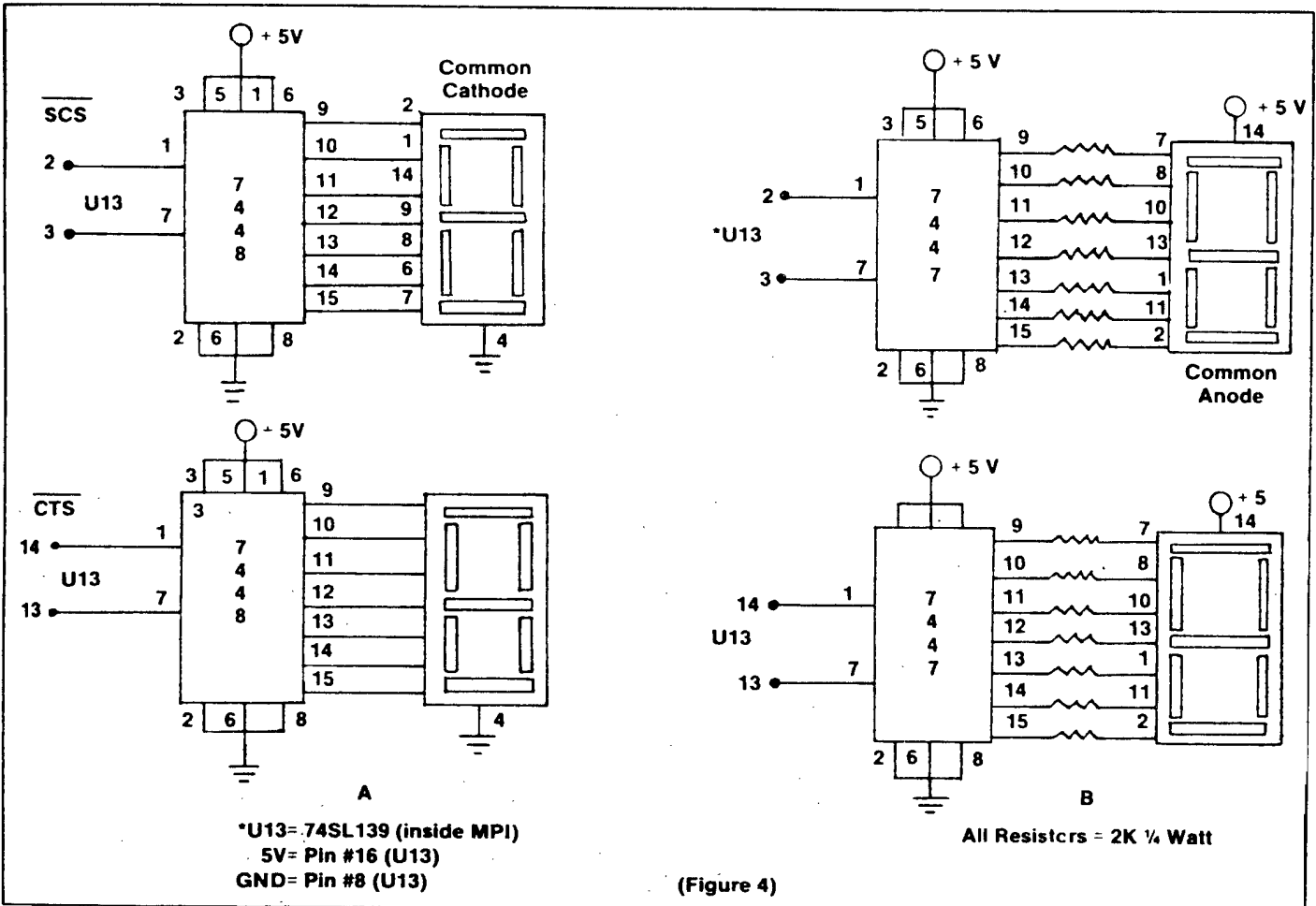


known as Disk Extended Color BASIC resides in this area. As a matter of interest, this software is kept on an 8K ROM chip, but only uses a little more than 6K of it. The rest of it is blank. The pin that controls (chip enable) this area on the cartridge connector is #32. It is called the Cartridge Select Signal (CTS) and is active LOW.

The second area available to the cartridge port is #8. It is generally used as an I/O port, but can be used for just about anything. The 32 byte length limits it to mostly I/O. Radio Shack game ROM-Paks do not use this area; the disk system does. It uses this area to communicate to the disk controller. Some of my projects also use this area. The pin that controls (chip enable) this area on the cartridge connector is #36. It is called the Spare Select Signal (SCS) and is also active LOW.

There are four slots in the MPI. This means you could put up to four ROM-Paks in there. They don't have to all be ROM-Paks; you could put in a ROM-Pak, a disk controller, a voicepak, an RS-232 adapter, an x-pad and your own "gizmo," just to name a few. They are all different, but fall into two categories: ones that use the SCS and/or CTS, and ones that use their own memory map decoding.

Let's look at the ones that do use these signals. The MPI has two ways of selecting which slot will be active: 1) The switch in front of the MPI. This is used as a "power up" default switch. When you turn the system on, the slot



that will be active will correspond to the switch's position. If you want the game in slot #2 to run, place the switch to #2 and turn the computer on. 2) The second way to select the active slot is by the built-in electronic switch. The electronic switch is nothing more than a memory-mapped byte. At this location, there is a latch so the associated circuitry can remember what slot is active. This latch is at 65407 (\$FF7F). Writing to this byte will change the active slot so it is equal to the value stored in that byte. To change the active slot, a poke or a store will do. You can also read the latch. The value returned will correspond to the active slot.

To make matters more complicated, the SCS and the CTS can be switched separately. Yes, the SCS can be in slot 1 and the CTS in slot 3. The electronic switch is divided into two parts, or nibbles. Each is four bits, making it eight bits, which is equal to one byte. The lower four bits controls the SCS and the upper four bits the CTS. A four-bit binary number can have 16 different combinations, but only the first four are used in the MPI. That makes four ports. The value needed to select a given port must start with zero. This is the first slot, even though the numbers start from one.

To select a slot, a little calculation is necessary. It is, of course, easier in Hex numbers. Here is a table that references the slots.

Slot #	CTS	SCS
1	0 (\$0)	0 (\$0)
2	16 (\$10)	1 (\$1)
3	32 (\$20)	2 (\$2)
4	48 (\$30)	3 (\$3)

To select a CTS and an SCS is simple: take the value from the CTS column

that corresponds to the slot number you want active, and add it to the value of the SCS that corresponds to the slot of that one. For example, if you want the CTS to be in slot 3 and the SCS in slot 2, the sequence would be as follows:

$$32 (\$20) + 1 (\$1) = 33 (\$21)$$

You would then *POKE 65407,33* but you must remember when you change

“There are four slots in the MPI . . . you could put up to four ROM-Paks in there. They don't have to all be ROM-Paks; you could put in a ROM-Pak, a disk controller, a voice pak, an RS-232 adapter, an x-pad and your own ‘gizmo,’ just to name a few.”

slot numbers, the computer might crash. It all depends on what software is running at the time. If, for instance, you were running Disk Extended BASIC and changed the CTS to another slot, a crash would occur and the disk software would no longer be there. If the slot that received control was auto-starting, it may start properly, depending on the status of the interrupts.

Now for the project. This is a simple 2-IC circuit. The IC I used in this project is the 7448. It is a BCD (Binary Coded Decimal) to seven-Segment decoder driver. This chip takes a four-bit binary number from zero to nine, and turns on the proper LED display segments to make them look like numbers. This IC can drive the display directly without resistors. It also uses the less expensive common cathode display (RS #276-075).

Unfortunately, the 7448 is not available at Radio Shack. The one available is the 7447 (RS #276-1805). There are two differences between the two: 1) it needs resistors to drive the display, and 2) it drives a common anode (more expensive) display. The choice is yours. If you can find the 7448, then use the common cathode display. If not, then use the 7447 with the common anode display (RS #276-053) and the resistors. Both schematics are shown in Figure 4.

I mounted the ICs and the displays on the same protoboard, as you can see from the photo. I will leave it up to you to mount the display where you want it. The display and the ICs do not have to be on the same board. You could always cut a square hole in the cover and mount the displays there.

To see if the display is working right, with all slots empty, place the front switch to slot #1 and turn the computer and MPI on. The display should read 00. Turn the switch to each position — #2, #3 and #4 — the display should read 11, 22 and 33, respectively. Try *POKEing* different values according to the Slot Table, and verify that the numbers change accordingly. From now on you will be able to see at a glance which slot is active.

Constructing 16K Of EPROM For Your Disk Controller

A lot of people call or write to me with suggestions about doing this and trying that, and I plan to start doing some of them soon. Some of the most popular ones are quite good, but I'll not mention them right now. I wouldn't want to say something and not live up to it later.

I would like to apologize to my readers for the errors that sometimes appear in "Turn Of The Screw." You see, all of the projects that appear in this article, I have built, tested and debugged. The biggest problem is when it is time to write the article, I have to take my prototype and transfer all the hardware information into type. That means diagrams, parts lists, text and schematics. This is where I am most vulnerable to errors. Once I have finished the rough draft, I read it over again, then when all is completed, I read the whole thing once more. Errors, however, do creep in; please bear with me, I do my best.

If, when constructing one of my projects, you do come across something that does not seem right, don't continue. Stop and study the situation. If you don't come to a solution, contact me either by letter (include a SASE) to THE RAINBOW, or by calling me on any Monday night at (514) 473-4910. Never try to do something unless you are sure of what you are doing. Be forewarned, the computer is not very forgiving. One error can cause a lot of damage. I know, I have burnt out a few chips in my time and occasionally still do.

Now to get to this month's topic. One of the memory mapped areas I described in last month's article is the area reserved for the cartridge ROM pack. I also said that when you plugged in the disk controller, the Disk Operating System (Disk BASIC) used this area. This month, we will look into expanding Disk BASIC hardware.

To recap this area, the *CTS pin on the controller controls the ROM chip that contains the disk software. The *CTS select line can access a total of 16,128 bytes. (Better known as 16K.) It is memory mapped from 49,152 (\$C000) to 65,279 (\$FEFF).

The ROM that Radio Shack uses in Disk BASIC is only 8K long, the lower 8K, from 49,152 (\$C000) to 57,343 (\$DFFF). All references to the "lower 8K" will be at this address. That leaves the upper 8K, from 57,344 (\$E000) to 65,279 (\$FEFF), of unused memory.

All references to the "upper 8K" will be this area.

Actually, this memory is not unused. It is memory mirrored to the lower 8K. This means it is not properly decoded and when the upper 8K accesses, the lower 8K chip responds. For example, type in:

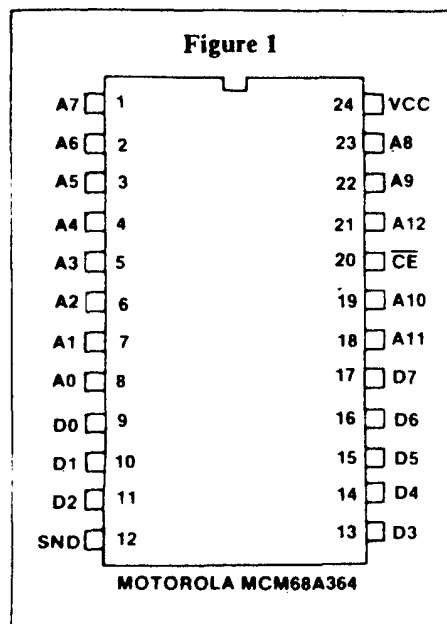
```
PRINT PEEK(49152) ENTER
```

and

```
PRINT PEEK(57344) ENTER
```

Both responses will be the same. Now, if we were able to properly decode this area, we could use the free space to add another chip, usually an EPROM. This chip could be used as an extension of Disk BASIC or often-used utilities.

For example, the Spectrum DOS, by Spectrum Projects, could be burned into EPROMs, and whenever you turned the computer on, it would be



right there. (I will not go into how to work with or use EPROM programmers. There are several on the market and all seem to be good; it all depends on price and ease of use. Usually the more you pay, the easier it is to use. I will leave the software programming up to you.)

What I intend to do in this article is describe the chip that is in the Radio Shack controller when you buy it, and the way you can interface two 8K EPROMs or one 16K EPROM.

The 8K EPROM I will use is the Intel 2764; it is the most economical one I have found. The 16K EPROM is the Intel 27128 (a little more expensive, but a little less trouble). Other manufacturers make the same chip, but make sure it is the Intel pinout as opposed to the TI pinout. You can use the TI pinout chip, but you'll have to figure out the pinout changes for yourself. Another note: If you like to use the high speed poke, for POKE 65495,0 you must use a 300 ns. access time chip, or faster, in order for it to work. The slower 450 ns. chip works in the regular mode, but not at the faster rate.

Now, the chip that contains the Disk BASIC software is made by Motorola. This chip is a masked ROM — ROM means Read Only Memory. That means the data contained in this chip can never be changed, erased or lost (unless you burn out the chip). The data is permanently printed directly on the chip itself at the time of production. It costs less to produce a ROM as long as the quantity is high.

The chip used here is an MCM68A364. It is an 8K by 8 ROM. Figure 1 shows the pinout of this chip. By the way, the BASIC and Extended BASIC chips are also the same chip, just different masks.

The first way of using all of the 16K memory in the cartridge area is to use a 16K EPROM. Figure 2 shows the pinout of an Intel 27128 EPROM. Examine the diagram and compare it to Figure 1.

What is wrong with this picture? There are 28 pins on this chip. The 8K ROM has only 24. This is a bit of a problem, but certainly not unsurmountable. It's time to get the ol' soldering iron and wire out. The following is a step-by-step procedure to modify and solder up a 27128 EPROM to fit (kind of) into a 24 pin socket. I recommend only those experienced in soldering attempt this.

The first thing we must do is study the pinout for this chip. Examine Figure 2, the Intel 27128 chip. The first thing we notice is that it has 28 pins, four more than the socket. Pin numbers 1, 2, 27 and 28 are the odd ones. If you line up pin #3 of the EPROM and pin #1 of the ROM, the rest of the pins are almost the same as the ROM. The different pins between an Intel 27128 and an MCM68A364 are as follows:

Pin #	EPROM	ROM
1 (-)	Vpp	N/C
2 (-)	A12	N/C
20 (18)	CE	A11
22 (20)	OE	CE
23 (21)	A11	A12
26 (24)	A13	Vcc
27 (-)	PGM	N/C
28 (-)	Vcc	N/C

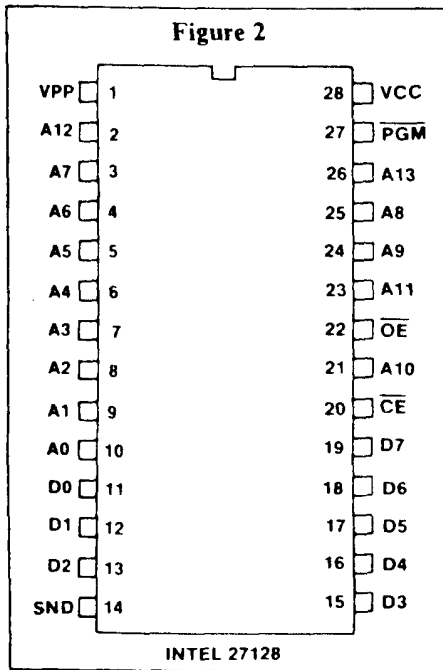
When it is time to insert this chip into the 24 pin socket, let the odd pins hang out. Pin #3 on the IC will plug into pin #1 on the socket. Make sure you get pin #1 right. It is usually marked with a small hole or a notch.

Step 1 — Bend pins #20, #23 and #26 (on the IC) out far enough so when you insert the chip these pins will not enter the socket. Make sure it does not touch anything.

Step 2 — Solder a short piece of #30 wire from pin #20 to pin #22 on the IC.

Step 3 — Solder another piece of #30 wire from pin #1 to pin #28 and pin #27 on the IC.

Step 4 — Solder one end of a one-inch piece of #22 wire to pin #28 on



the IC. Strip 1/8 inch of insulation from the other end. This end will insert into the empty pin #24 of the socket.

Step 5 — Solder one end of a two-inch piece of #22 wire to pin #2 on the IC. Strip 1/8 inch of insulation from the other end. This end will insert into the empty pin #21 of the socket.

Step 6 — Solder one end of a one-inch piece of #22 wire to pin #23 of the IC. Strip 1/8 inch of insulation from the other end. This end will insert into the empty pin #18 of the socket.

Step 7 — Solder one end of a four-inch piece of #30 wire to pin #26 on the IC. Solder the other end of this wire to pin #37 on the edge connector, the side that plugs into the computer. That is the second to last pin closest to you on top, on the right-hand side if you are looking at the front of the controller.

That's it! Carefully insert the chip into the socket making sure there are no shorts. You now have a 16K EPROM in your controller. If you want to erase this EPROM, just remove all of the solder spots and start over again. If you do a good job in soldering and de-soldering, the EPROM could stand about 10 or so recyclings.

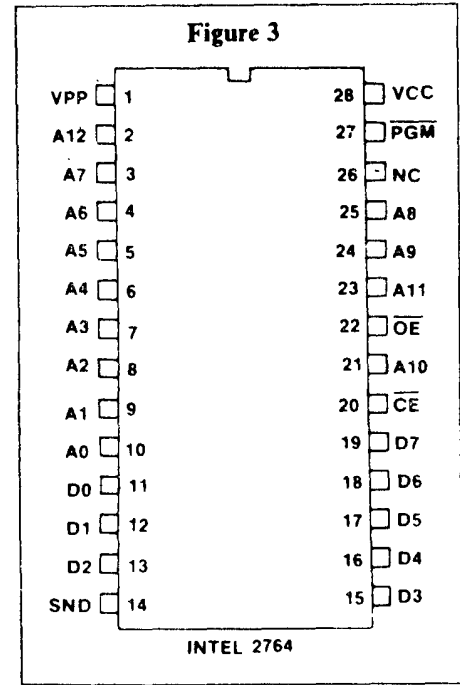
If you don't want to use a 16K EPROM, or your present programmer cannot handle 16K chips, then using two 8K EPROMs is the answer. There are two problems with using two 8K EPROMs. The first problem is how to decode the two separate chip select lines and the last address line. Figure 3 shows the pinout of an Intel 2764.

Notice that pin #27 is the pin used in the programming of this chip. However, if this pin is low during a read cycle, the chip "deselects" — the chip does not respond to a read. It stays deselect all the time this pin is low. If we were to attach the last address line to it (A13) when this line was low, the chip would not activate. The fact that A13 is low means you are accessing the lower 8K block. Since the chip deactivates when it is low, it meets the decoding needs of the upper block.

On the other hand, pin #20 of the is made to activate the chip when low. So, if we tied A13 to this line, the chip would behave opposite to the first. It would be deactivated when A13 is high. This would properly decode for the lower 8K block and deactivate for the upper.

Using this technique would solve our first problem, but we still have one more problem: Where to put the second chip? I have used this technique before and most likely I'll use it again — it's great. It is called the "piggyback" technique. We will solder the two chips on top of each other, except A13 and a few more, to get it to fit in a 24 pin socket.

Before we go any further here, there is a difference between an Intel 2764



and an Intel 27128: The 2764 has an N/C on pin #26, whereas the 27128 has A13.

The following is a step-by-step instruction on how to solder up two 2764s to fit in a 24 pin socket and be accessed as a 16K chip.

Step 1 — Program the first chip with the data that goes into the lower 8K and mark it as the lower chip. Program the second with the data that goes into the upper 8K and mark that one as the upper chip. It is important not to get the two mixed up, they are not wired up the same way.

Step 2 — Take the lower chip and bend pins #20 and #23. Take the upper chip and bend pins #20 and #27.

Step 3 — Mount the upper chip on top of the lower chip so pin #1 is on pin #1, 2 on 2, and so forth, leaving a small gap for air circulation. Solder all the unbent tips of the upper chip to the bases of the lower chip. Even if the lower pin is bent, in the case of pin #23.

Step 4 — Solder a one-inch piece of #30 wire from pin #20 to pin #22 on the upper IC.

Step 5 — Solder another piece of #30 wire from pin #1 of the lower IC to pin #28, #27 and #26 of the lower IC.

Step 6 — Solder one end of a two-inch piece of #22 wire to pin #2 on the upper IC. Strip 1/8 inch of insulation from the other end. This end will insert into the empty pin #21 of the socket.


Step 7 — Solder one end of a two-inch piece of #22 wire to pin #23 on the upper IC. Strip $\frac{1}{8}$ inch of insulation from the other end. This end will insert into the empty pin #18 of the socket.

Step 8 — Solder yet another two-inch piece of #30 wire from pin #20 of the lower IC to pin #27 of the upper IC. Solder one end of another four-inch piece of wire to pin #27 of the upper

IC and solder the other end of that to pin #37 of the edge connector. See the first Step 7 for proper location of this pin.

You are now ready to plug the "spider" (as I call it) into the socket. Remember pin #3 in the spider goes into pin #1 of the socket. The other four pins sticking out over the socket are #1, #2, #27 and #28.

By popular request, for those of you who do not want to build this project, there is a board adapter available, built and tested, that you can buy from R.G.S. Micro which does the same thing. See their ad in this magazine. It fits inside only the J&M controller and is made for two Intel 2764 EPROMs.

That is it for now, enjoy your 16K Disk BASIC. 

Adding A Numeric Keypad To Your CoCo

Just the other day, I walked into my local electronics store and saw they had recently opened a warehouse bargain section in the rear of the store. I immediately went in and started to browse. This place is a gold mine of old parts and nifty gadgets. Some items I found were individual keyboard switches. They were surplus from who knows where, were of good quality and very inexpensive.

If you recall, a while back I did an article on adding function keys to your keyboard. I explained that in the eight by seven matrix that makes up the CoCo keyboard, there are four free areas and how to add switches. Ever since then, I get requests to write an article on how to add a numeric keypad to the Color Computer.

I looked into it and found that it would be quite easy to wire one up. Very few components would be needed and it would not cost too much. About the only thing that was keeping me from doing such an article was the actual



keypad switches — there were none to be had around here. I could have used regular switches; after all, that is all that makes up the keyboard part of a CoCo, but it would not look like a nice keypad. Therefore, I put the numeric keypad article on the back burner.

Back at the electronics store, I picked up about 20 keyboard switches along with an assortment of keycaps. In no

time at all, I had myself a nice numeric keypad. It was then that I decided I should submit "Adding a Numeric Keypad" to THE RAINBOW.

It is still up to you to find your own keyboard switches and keycaps. You will also have to build your own keypad case since the size and shape of your case will depend on what kind of keyboard switches you get and how many you decide to add (I'll explain later). In other words, all of the cosmetic side of this project will be left up to you. I will supply the schematic, parts list and method of putting together a numeric keypad.

For the benefit of those of you who do not know how the CoCo keyboard works, a little background information may help you with this project.

The keyboard has 53 keys. A PIA (Peripheral Interface Adapter) is used to scan these switches (keys). The eight keyboard columns are attached to the 'B' side of the PIA. These eight lines are programmed to be outputs.

The seven keyboard rows are attached to the 'A' side of the PIA. These seven PIA lines are programmed to be inputs. To read the keyboard, only one column is enabled by writing a zero in the bit that corresponds to that column and

(Tony DiStefano is well-known as an early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.)

by writing ones in all the other bits. If a key has been pressed in that column, one of the input lines will be a zero and the key location will correspond to the bit that is low. By scanning each column in the keyboard, all of the keys may be checked.

My idea is (if I run some wires in parallel to that of the keyboard lines) to take those wires and run them to a connector, and finally to some keyboard-like switches. Since the switches would be in parallel, this would give you the choice of pressing, for example, the number '1' on the main keyboard or on the numeric keypad. You could enter all your numeric data from the keypad.

But also, I included a few more keys that would be handy: the plus key (+), the minus key (-), the multiplication key (*) and the division key (/). Then there is the decimal point (.) and the ENTER key (CR).

The schematic in Figure 1 shows how to wire the above keys to the main keyboard connector. I chose those keys because it suited my needs. There is no reason why you could not change them to fit your needs, or for that matter,

you can add a complete second keyboard. All you have to do is get the right wiring.

Figure 2 shows the complete wiring diagram of the CoCo keyboard. All versions of the CoCo or CoCo 2 keyboards are the same, even though the keyboards look different. That is one of the few things that did not change in the ever-changing CoCo.

Now, the next thing I didn't like was that if you wanted to enter a multiplication sign or a plus sign, you had to press the SHIFT key. I had two choices: 1) include the SHIFT key and press it every time you wanted these functions, or 2) make a small electronic circuit to automatically press the SHIFT key when you hit these keys. I elected to do the latter of the two.

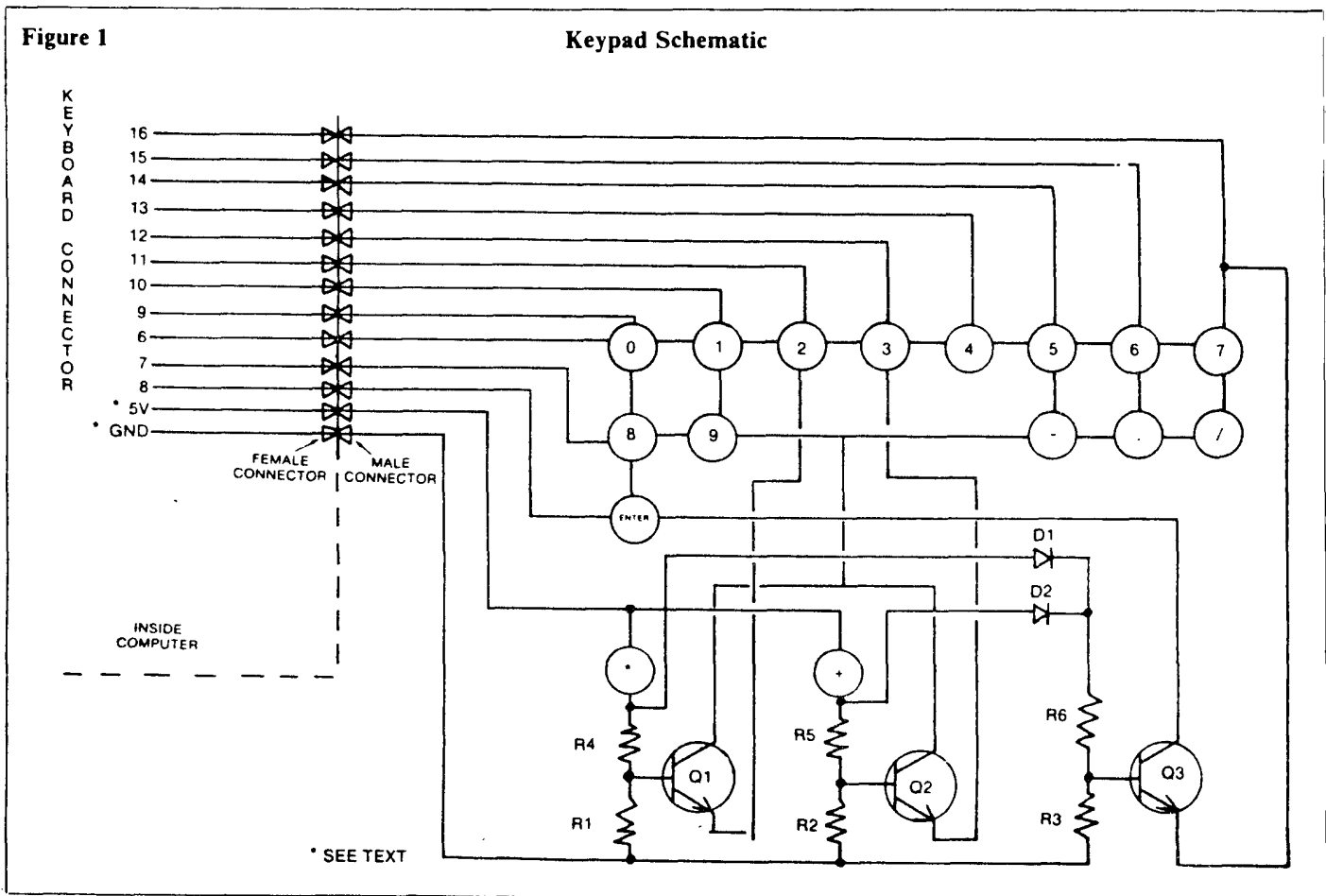
The switch that is normally used for the keyboard is an SPST (Single Pole Single Throw) momentary on. To automatically hit the SHIFT key and the key you want shifted would require a DPST (Double Pole Single Throw) momentary on. That way, both circuits would connect and we would get the shifted function, if any, of that key. That is a good way to do things, but

I could not find a DPST switch in the shape of a keyboard key. So, I decided to make an equivalent transistorized circuit.

Look at the transistor part of the circuit diagram in Figure 1. Each transistor acts like a switch. The 10K bias resistor makes sure the transistor stays off when not being accessed. That is the equivalent of the switch being off or open (no key pressed).

The emitter of the transistor is connected to the output side (Port B) of the PIA. If you recall, all the 'B' lines are programmed to be outputs and are all high or five volts. Only one line at a time goes low, so when the line that has the emitter of the transistor connected to it goes low, the transistor's emitter is effectively connected to ground.

The 1K base resistor is used to limit the base current, but enough to turn the transistor on. The switch in this circuit is connected to five volts. When the switch is on (key pressed), current flows through the resistor, therefore, turning the transistor on. That makes the collector of the transistor ground potential. In turn, the ground potential



on the collector grounds one of the corresponding input pins on the 'A' Port of the PIA. To the computer, this translates into a pressed key.

Now, take the plus key for example. The Port 'B' output that connects to this key is PB3 (keyboard connector #12). The input pin is PA5 (keyboard connector #7). I placed my transistor circuit on these two points as described above. Now, when I press the switch connected to the transistor, I get the semicolon (;), the unshifted plus.

I then made another transistor circuit for the SHIFT key and connected the base resistor to the same switch as the plus key. Now when I press the switch connected to the two transistors, I get the shifted plus in one key press. Nice, but this would require two transistors for every shifted key I needed. Use a simple diode to isolate the two transistors and now you only need one diode per shifted key. (I'm sure that someone will write me saying, "I found a way to do it with fewer parts," but this one works, so I'll use it.)

As a point of interest, this circuit can make an easy pause key. When you want to stop a listing, you press the SHIFT @ key. Well, this would make a one-key pause button.

You can really get carried away and make all of the shifted keys "one-key only." For example, "SHIFT backspace" means backspace the complete line. You can now have one key to "delete line." Another good one is the question mark (?). It is used as a short form for the PRINT statement.

The construction of this project requires a bit of doing and cutting. I'll leave that part up to you. As you can see in the photo, I used a proto-board and glued the keys onto it. You can see the transistors and resistors at the bottom. The important parts, like the theory of operation, schematic diagrams and keyboard layout, are here. There should be enough information here to get you going.

Since there are many board revisions to the CoCo, there might be a problem as to where to find the right connections to the keyboard. The best way to cover all versions is to connect directly to the keyboard connector itself. It is a 16-pin connector and all of them are wired the same way, even though the connector might be different.

I suggest you solder your wires to the connector. If the connector is too close to the board and you cannot reach

its soldered pins, you could always remove the board and solder to the pins from the bottom side.

Remember, when soldering from the bottom, the pin numbers are backwards. The pin numbers go from 1 to 16; it is marked which side is 1 (left) and which side is 16 (right).

Find a good spot to mount the 15-pin connector — on the side of the computer directly under the keyboard is not bad. On the left or right depends if you are left- or right-handed.

Mount the female connector to the computer. Using a short length of ribbon wire, solder all the pins needed from the keyboard connector to the 15-pin connector.

There are two more connections that go to the 15-pin connector that do not go to the keyboard connector: the ground wire and the five volt wire. There is always a question of where is the best place to connect the five volts and ground. I always look for a 1. uf decoupling capacitor. They usually connect to the right points. If you are not sure, use pin #8 on the 74LS138 for ground and pin #16 on the same chip for the five volts. That is all the wiring you have to do on the computer side; the rest is all in the keypad adapter.

Solder another short length (the length is up to you) of ribbon wire to the male 15-pin connector. Make sure all the wires match the pinout of the female side. The rest of the wiring is done on the proto-board with the keyboard switches and other parts.

The parts list matches the needs of the schematic in Figure 1. If you are adding more keys, you will have to add more parts. The connector I used has 15 pins. There are a few free ones, but if you decide to do a complete remote keyboard (or somewhere in between) you will need to move up to the 25-pin connector since the 15-pin connector is not enough.

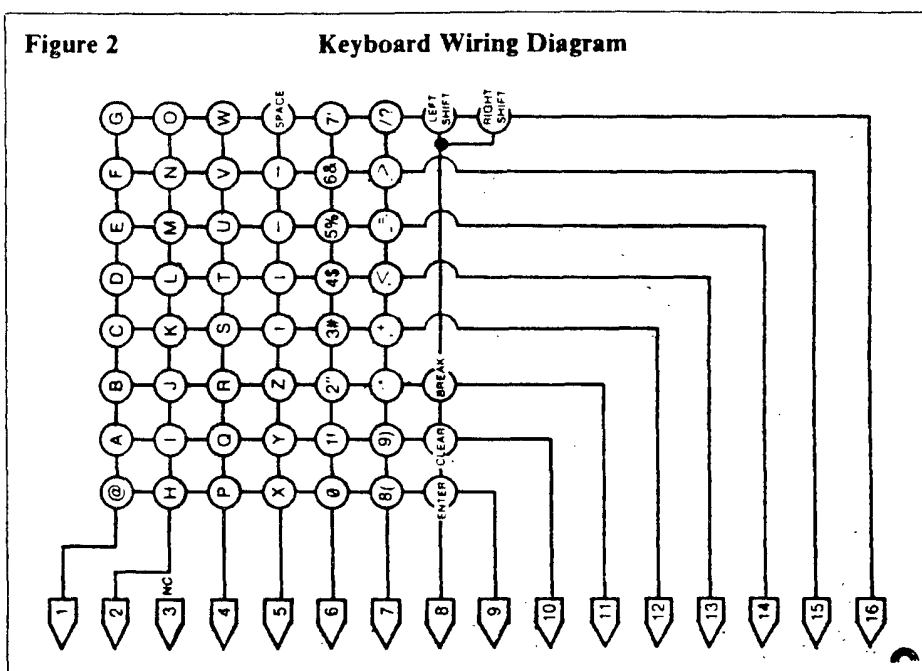
One last thing, if you write me and expect an answer, please include a SASE or, for faster responses, you can call me on Monday nights *only* (please). My number is (514) 474-4910.

Parts List

ID	Description
Q1, 2, 3	2N3904 transistor
R1, 2, 3	10K 1/4W resistor
R4, 5, 6	1K 1/4W resistor
D1, 2	1N914 diode
C1	15 pin sub-D male
C2	15 pin sub-D female
Miscellaneous	16 (or more) conductor-ribbon wire
	12 key-switches
	12 key-caps
	proto-board
Hardware	plastic or metal case screws and mounting lugs, etc.

Reference

TRS-80 Color Computer Technical Reference Manual



How To Hook Up The Radio Shack Voice Synthesizer

A little while back, I did a project using the Votrax SC-02 voice synthesizer chip to make CoCo talk. It was an interesting project and I got a lot of correspondence about it. However, not all of it was good. People found that the chip was hard to find, and when they found it, it was very expensive. Ever since that time, I have been getting letters inquiring about how to hook up Radio Shack's own speech synthesizer to the CoCo.

I just came back from the Irvine RAINBOWfest in California, and believe it or not, more than one person asked me about this synthesizer. I know I am slow at times, but I think I finally got the message. So, this month we are presenting "How to Hook up the Radio Shack Voice Synthesizer to Your Color Computer."

But, first things first! I got a good piece of information while I was at the Irvine RAINBOWfest. I was talking to a gentleman about the "ins and outs" of the CoCo and we came upon the subject of repairing. If you have the 'F' board (also known as the 285 board) this is for you.

On some of these boards, there is a

(Tony DiStefano is well-known as an early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.)

problem (an intermittent one at that). The symptoms are as follows: the computer works fine for a while, then all of a sudden random characters start to appear in columns one and nine of the screen. Just about this time, the computer freezes up and all work that is there gets lost. According to the gentleman I spoke with, the problem stems from the SAM; it is some sort of heat problem.

He says that Radio Shack is aware of this and is offering some help. Go to your Radio Shack dealer and order the "Final Fix" for the old CoCo.

Let's get back to the synthesizer. When I went to buy the chip, I saw there were two different sets: the older 276-1783 chip set and the newer 276-1784. Only the 1784 is listed in the new catalog, so I decided to go with it. This is just one chip while the 1783 is two. The package of this chip says that it comes 1) complete with specifications, applications data and programming information, and 2) requires additional components and skill in project assembly.

That's fine, but they don't tell you how to hook it up to your Color Computer.

Usually, in this next section I describe the functions of my project. This time, the project I am doing comes with a 20-page manual. I must say it is not

a bad manual; the only thing left out was the circuit to connect it to the CoCo. But once the circuit is up and running, all you need to start writing programs that talk is in the book. It has all of the "allophone" set (as they call it) and even has a dictionary of words. It also includes a set of rules for using these allophones.

From the diagram and the description of this chip, I think you can add more chips to it so it can speak more sounds, possibly whole sentences and phrases. There is, however, no reference to part numbers or where and when these chips will be available. There will probably be more on this later. Anyway, I included a BASIC program listing you can use to try out your project.

This chip, as is, is quite easy to implement to the CoCo. It is basically divided into two parts. The first part is to get the data to the chip. The second is to poll the chip until it is not busy. Then you can give it the next piece of data.

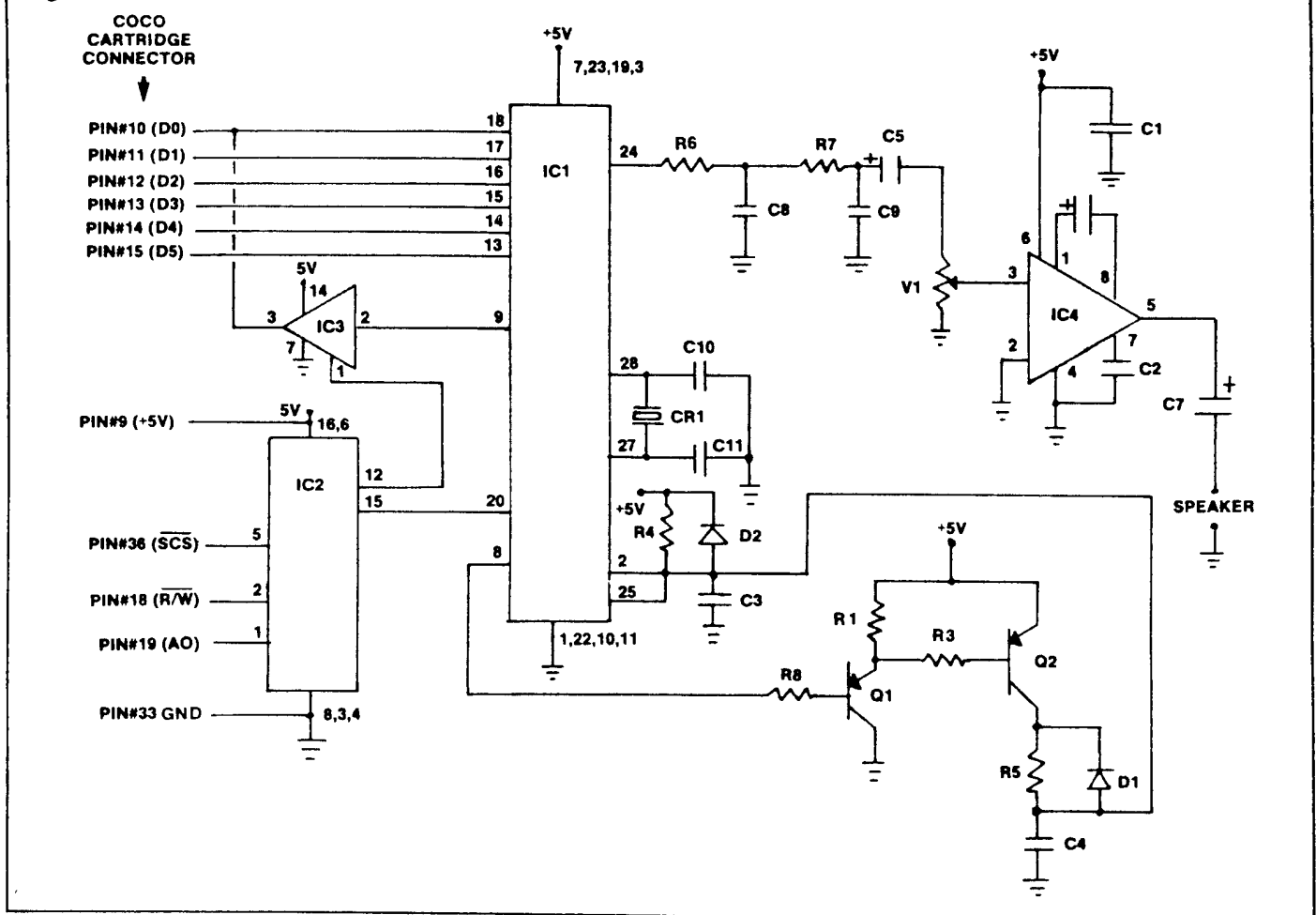
It is just like a parallel printer; in fact, if the CoCo had a parallel printer port, I would have used it without any other extra parts. But that is not the case, so I used the old cartridge method which means it is connected to the CoCo via the cartridge connector, so if you have a disk drive, you will need a multipack or some kind of a switch.

The listing: *TURNSCRW*

```
1 ' THIS IS FOR THE RADIO SHACK
2 ' VOICE SYNTHESIZER IC
3 ' FROM RAINBOW'S
4 ' TURN OF THE SCREW
5 ' BY TONY DISTEFANO
6 '
10 FOR I= 1 TO 55
20 READ A
30 POKE&HFF40,A
40 IF (PEEK(&HFF41) AND 1) = 1 T
HEN 40
```

```
50 NEXT I
60 END
100 DATA 27,7,45,15,53,4
110 DATA 24,06,04
120 DATA 26,26,16,4
130 DATA 20,04
140 DATA 13,23,23,2,42,12,44,4
150 DATA 42,15,16,9,49,22,13,51,
4
160 DATA 4
170 DATA 63,24,06,4
180 DATA 13,53,11,19,4
190 DATA 33,12,55,0,13,7,0,40,26
,26,56,53,1
```

Figure 1



It will not work with just a Y-cable because I used the *SCS output of the computer and it is used by the disk controller (see my previous article on the Multi-Pak Interface). With a few more chips, you *can* make it work with just a Y-cable. I will be doing an article soon on the technique of memory mapping and how to memory map something anywhere in memory.

To get data to the chip is easy. All you have to do is strobe the *ALD pin with data valid on the data lines and the data is entered. You can use the *SCS pin and you would not need any other parts, but there is another location to monitor. That is the pin that says when the chip is no longer busy with the last command you gave it, which is the *LRQ pin.

This is where the two TTL chips I added come in. The first chip is a 74LS138; this is a decoder chip. It is capable of decoding a three-bit binary number into its eight different outputs. It also has three other select lines. Examine the 74LS138 in Figure 1 and notice that all I used is three lines. That

is all that's necessary for this project.

We need two locations, one to write the data to the chip, and the other to read the busy pin of the chip. The *SCS pin of the CoCo selects the 74LS138 chip. A0 selects which location and the *R/W lines select a read or a write. Since we are using the *SCS pin on the CoCo, location \$FF40 (65344 in decimal) is the base address. We are using only A0 so the two locations are \$FF40 and \$FF41.

In this case, \$FF40 (65344) is the write location which is used to transfer data to the chip. Location number \$FF41 (65345 in decimal) is used to monitor if the chip is busy. Reading (or PEEKing in BASIC) this location reveals whether it is busy or not.

The *LRQ line is connected to the input of a tri-state buffer. This is the 74LS125 chip. Only one of the four gates is connected. The output of this gate is connected to D0 on the CoCo bus. When you read the location, all other bits in the byte are irrelevant. If bit 0 is a logic 1, the buffer is full and the chip is busy. When this bit is logical

0, the chip is free and waiting for another command.

The rest of the circuit is the same as the recommended circuit by Radio Shack. There is one thing that confuses me about the Radio Shack circuit and I don't have the solution. It is the reset circuit: the two transistors, diodes, capacitor and resistors that connect to the reset and SBT reset. This circuit, as is, does nothing. I think it has something to do with the little arrow and the "NOTE" sign. What does that note refer to? Where is that note? What does that do?

I constructed the whole circuit, along with my circuit, and it worked fine. I monitored the SBY pin on the synthesizer chip and found it did nothing. It was always a logical 1. I disconnected the pin from the rest of the circuit; it still made no difference, so I cut out all the components except for the 100K resistor.

If you feel you must leave this circuit in, fine. Better yet, if you have an answer as to why it is there, please write me; I would like to know.

If you have a Multi-Pak from Radio Shack, a simple poke will give you access to the chip by changing the soft switch inside the Multi-Pak Interface. Remember that the Multi-Pak can change access to the *CTS pin and the *SCS pin. The *CTS pin controls 16K of software and the *SCS controls 32 bytes of I/O. The control byte is \$FF7F (65407 in decimal).

To change the selector, you must poke a number into this byte. The most common configuration is to have the controller in slot #4 and the voice in slot #1. In that case, the value you must poke in the control location is a value of \$30 (48 in decimal). Refer to your MPI manual for more details.

Table 1 lists all the parts necessary to build this voice synthesizer, including the parts in that reset circuit. At the end of this article, there is a list of mail-order stores that you can get parts from. There is no guarantee that any or all of these stores will have these parts. Except for the TTL chips and the proto-board, Radio Shack will have all of these parts.

There is one more thing to note. The diagram requires a 3.12 MHz crystal. The manual says you can order this

crystal from Radio Shack, but you will have to wait. I didn't want to wait, so I bought Radio Shack's 3.579545 MHz crystal instead, which they had in stock. It works just as well, except the voice will be about 14 percent faster.

As usual, if you have problems with this or any of my projects, write to me (through RAINBOW) and I'll try to help you. If you have an emergency, you can call me on *Monday night only* after supper. My number is (514) 473-4910.

Electronics Parts Mail-Order Houses

JDR Microdevices
1224 S. Bascom Ave.
San Jose, CA 95128
(800) 538-5000 or (800) 662-6279 (CA)

Jameco Electronics
1355 Shoreway Road
Belmont, CA 94002
(415) 592-8097

Dokay Computer Products, Inc.
2100 De La Cruz Blvd.
Santa Clara, CA 95050
(800) 538-8800

Table 1

Number	Description
IC1	SPO256-AL2 (Radio Shack #276-1784)
IC2	74LS138
IC3	74LS125
IC4	LM386
C1,2,3,4	.1uf 15V
C5	1uf 15V
C6	10uf 15V
C7	100uf 15V
C8,9	.022uf 15V
C10,11	22pf 15V
R1	1K ¼W
R2	10K ¼W
R3	200K ¼W
R4,5	100K ¼W
R6,7	33K ¼W
R8	10 ¼W
V1	10K POT
CR1	3.12 MHz (see text)
DI,2	1N914
Q1,2	MPS 2907 or 2N2907
Misc.	Proto-board, speaker, solder, wire, case

How To Follow A Memory Map

I feel like a broken record, but I still get a lot of questions and calls about memory mapping. Don't feel bad — it took me quite a while to get it right myself.

Let's go over it step by step. This time, I'll go into some hardware on how to memory map something to the CoCo *SCS area, which is the area mapped at 65344 (\$FF40) to 65375 (\$FF5F). This memory mapping technique will work on any version of the CoCo or CoCo 2 since the theory is the same. In fact, most of this theory will work on just about any computer.

A basic understanding of a CPU is a must when trying to understand

mapping. By now everyone understands the importance of binary and Hex numbers; it has everything to do with mapping.

Let's start with binary: zero and one. That's it. A binary digit has only two values, zero and one. Two binary digits have four combinations: 00, 01, 10, 11. Three digits have eight and so on. Table 1 shows a four-bit number and the relation between decimal numbers, Hex numbers and binary.

As you can see, a number from zero to 15 in decimal can be represented by one character from '0' to 'F' which is four binary bits. This is called a nibble. Now, a number from zero to 255 in

decimal can be represented in Hex from '0' to "FF". This is called a byte. In binary, a byte takes up eight bits or two nibbles. The 6809 CPU (the CPU in the CoCo) has a data bus of eight bits, better known as an eight-bit CPU. (The internal structure is 16-bit, but I'll get into that story another day.)

Back to our nibble. This nibble represents 16 different combinations or discrete locations. Each different location becomes one memory location and each memory location has its own discrete address.

Address 0 (0000 in binary) is the first memory location (zero is a valid number). Address 1 (0001 in binary) is

Table 1

Decimal	Hex	Binary
0	0	1111
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

the second. But, that is only four bits; the 6809 has 16 bits used for memory locations which is a 16-bit address bus. Sixteen address lines means the CPU can access 65536 different locations. The first location is "0000000000000000" and the last location is "11111111 11111111," with 65534 combinations in between. For example, "10101000 01101010" is a valid location.

Writing out 16 zeros and ones every time we want to mention an address is silly. If we go back to our nibble, it can be represented by a single character. Sixteen bits would be four nibbles. Each nibble represents one-fourth of the 16-bit address. So, going back to our first location, we can now write it as a four-digit number, \$0000.

The '\$' in front of the number means the number to follow is in Hex; it can also be represented by the letter 'h' at the end of the number. The last location would now be \$FFFF, and a number somewhere in between would be \$CD8A.

That is the basic memory map of a CPU. Let's go back to our nibble for now — it is a little easier to work with. If we were to spread out each of the 16 locations into individual outputs, there would be 16 of them.

Most computer peripheral devices such as PIAs and VDGs require that a logical zero be used to select that particular device. That means if you have several devices connected to the same computer and want to select one at a time, all the select lines would be at logical one, except the peripheral that

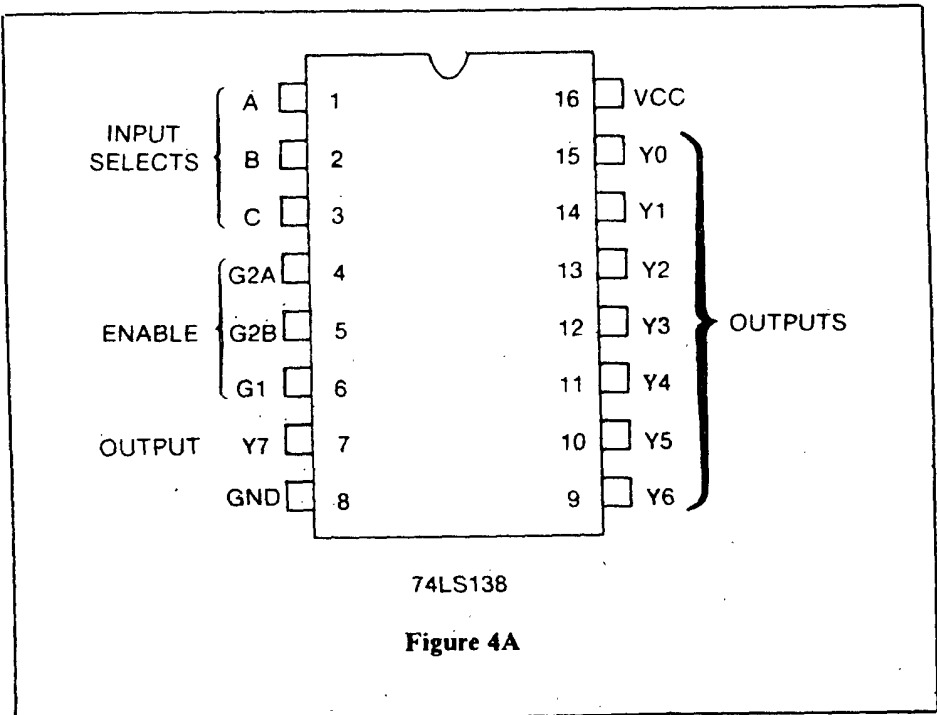


Figure 4A

is to be selected. If we were to map out our four-bit address to one of 16, the result would look like the results in Table 2.

Table 2

Binary Number	One of 16 Select Lines
0000	1111111111111110
0001	1111111111111101
0010	1111111111111011
0011	1111111111110111
0100	1111111111101111
0101	1111111111011111
0110	1111111110111111
0111	1111111101111111
1000	1111111011111111
1001	1111110111111111
1010	1111101111111111
1011	1111011111111111
1100	1110111111111111
1101	1101111111111111
1110	1011111111111111
1111	0111111111111111

In each of the 16 examples, only one of the 16 lines is low, therefore only one of the possible 16 devices is selected. This is known as decoding. Decoding means separating a binary input to its individual outputs.

That is only four bits. If we were to look at 16 bits (the amount of address lines the 6809 CPU has), the decoded output would be one of 65536. Listing a table of the outputs would require

several hundred pages (I think I'll pass on that one).

You can see the amount of components that goes into a chip. The amount of individual outputs doubles with every addition of one bit. Table 3 shows the relation between the amount of binary bits to the amount of individual select lines possible.

Table 3

Number of Bits	Number of Select Lines
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536

Do the numbers in the right-hand column look a bit familiar? They should: 1K, 2K, 4K, 8K, 16K, 32K and 64K. These are the real values people talk about when they say "K's." When you say your computer has "16K," it really has 16384 bytes of memory; 16K is just a rounded off number for the

real thing.

OK, we now understand how a CPU can access all those bytes of memory. "How come I can't see thousands of wires and chip selects in my computer?" would be the next question. Well, there are thousands of wires and chip selects in your computer, but most of them occur inside the major chips of the computer.

Take, for instance, the Color BASIC chip. It is 8K, or 8192 bytes long. This is a good place to start. If you look back to Table 3, it takes 13 address lines (lines A0 through A12) to make up 8K of memory. The chip used for Color BASIC has 13 address lines. They connect to the first 13 address lines of the computer. That leaves us with a balance of three lines.

A typical Chip Enable line on a memory chip activates the whole chip. When *CE is activated, it works in conjunction with the other 13 lines. It is sort of a *master* select. The computer tells the chip that I want a byte of data. The other 13 lines tell the chip which of the 8192 bytes of data it wants.

Now, look back at Table 2. For the sake of theory, take the Color BASIC chip. Connect the first 13 lines (least significant) to the CPU. You are left with three unused lines (most significant). Look at the first three bits in Table 2. If you apply that theory to this situation, three bits can select eight devices.

Consider the Color BASIC chip as a device and connect one of the output lines of the three to eight decoders. A decoder such as this does exist; it has three inputs and has eight output lines. It also has other control lines, but we'll look into that a little later in this article. Connect the three binary input lines to the last three free address lines of the CPU. Depending on which output line we use, the CPU will select the Color BASIC chip on one of eight 8K borders.

If we put the chip on the first line, the CPU will activate the chip from memory location 0000 to 8191. If the chip was hooked up to the second, it then would see the chip as being from 8192 to 16383, the third would be from 16384 to 24576 and so on and so on, increasing by 8K every time, until we reach 64K. This is known as memory mapping. What we have done is memory mapped an 8K chip to the CPU. Again, where this 8K is depends on what output line of our decoder we use.

We have used all of the address lines in this situation. There are times when not all of the lines need to be used.

INPUTS				OUTPUTS								
ENABLE		SELECT										
G1	G2*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	L	H	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	H	L	H	H	H	H	H	L	H	H
H	L	H	H	H	H	H	H	H	H	H	L	H

*G2=G2A+G2B

H=LOGIC 1, L=LOGIC 0, X=IRREVELANT

Figure 4B

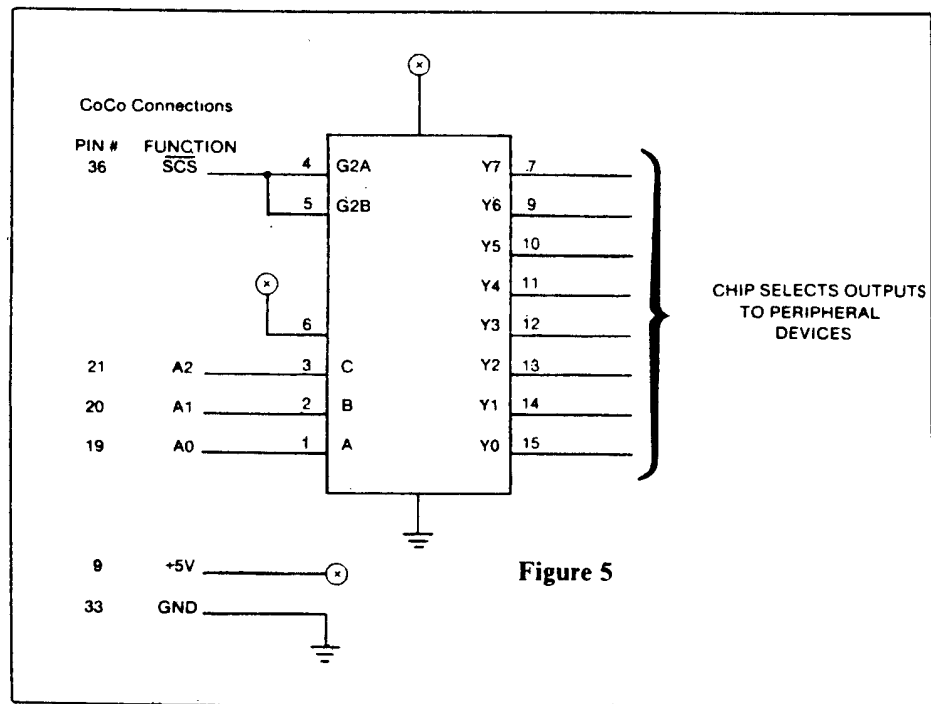


Figure 5

When memory mapping a device to a CPU and not all the address lines are used, a condition called memory "ghosting" or "mirroring" is formed.

Memory mirroring is produced when the same chip is activated in two or more areas of memory. The best way

to explain this is to use an example. Take the previous example of the 8K Color BASIC chip. The chip itself has 13 address lines connected to the CPU and the remaining three (most significant) lines of the CPU are decoded to one of eight. That leaves no address line

free or unused. If we were to use a 4K chip instead of the 8K, there would be one less address line. Table 3 reveals that.

Now, leave this address line free and not connected to anything. When the CPU reads the first 4K of the chip (the only 4K in this case) all is fine, but when the CPU reads the next 4K, the 13th address line will change state. Since it is not connected to anything, the CPU will read the same thing as the first 4K. That is because the only address line that changed for the second 4K of memory is that free address line.

Let's take this one further and use a 2K chip. Now we have two free address lines. The CPU will see the same repeated data every 2K for the duration of the 8K bank. Bank is a word used to describe an area of memory. It is not any particular size, but referred to as an 8K bank or a 2K bank, whatever the size in question is.

It is not wrong to leave free lines when memory mapping, but it does make for inefficient use of memory. Take for instance the Disk Extended Color BASIC from Radio Shack. The chip itself is only 8K long, but is mirrored twice into 16K. It still works but renders the other 8K of memory unusable without more hardware to decode the

free lines. This, however, does make for a less expensive parts count.

Now to get down to hardware. The area most frequently used by CoCo hardware hackers is *SCS: Spare Chip Select. It is already partially decoded by the SAM chip. It is sort of a mini "Master Select." The SAM chip decodes this area to be from 65344 (\$FF40) to 65375 (\$FF5F). It is only 32 bytes long, therefore also requires five address lines. These are A0 to A4. So the *SCS (Master Select), along with five address lines, makes up the 32 bytes of the memory map.

This area is great for I/O purposes such as the projects I presented in this column. Take, for example, my article "Lights! Camera! CoCo!" (December 1984, Page 24). It uses the *SCS pin. This is just the sort of thing I am talking about. I used just the *SCS pin and none of the other address lines. That means the chip I used is memory mirrored throughout the 32 bytes (five address lines) and is only one byte wide. I saved adding some chips, but in this case, I didn't need the rest of the area.

Now, if we take the three to eight decoder I mentioned earlier, and integrate it into the *SCS circuit, we could access more chips. Figure 4A shows the functions of a chip called the

74LS128. This is a computer compatible chip that works well with the CoCo. In fact, there is already one of these chips inside the CoCo. If you have a schematic for the CoCo, look it up.

Figure 4B shows the Truth Table for this chip. When you examine this table, you will notice the similarity between this and Table 1, only it is only three bits wide. There is a four to 16 decoder chip available, called a 74LS154, but you'll have to look that one up yourself.

Now, the diagram in Figure 5 shows how this chip can be hooked up to the CoCo and the *SCS pin. This is hooked up as such: You have eight separate chip enables from 65344 (\$FF40) to 65351 (\$FF47) and it is memory mirrored four times to make a total of 32 bytes. If we were to replace A0 with A1, A1 with A2, A3 with A4 and left A0 not connected, we would have every second byte memory mirrored. If we moved the address lines up one more, it would be every four bytes memory mirrored.

If we added more 74LS138s, we could even have 32 bytes not mirrored at all. It all depends on the decoding technique and how many free address lines we want.

I hope all this decoding has helped you understand more on how the CoCo works. See you next time, and we'll say hi to LEDS. ☺

Look, Ma No Switch!

Before I get into this month's project, I want to thank my readers for being so good about calling me only on Monday nights. I have had many interesting calls, and not all on problems relating to my projects, either. Some just call me to discuss theory and hardware.

For all new readers to this column, let me explain what I am talking about. I have set aside Monday nights for people to call me about problems they might encounter in putting my projects together. My number [in Canada] is (514) 473-4910. If by chance you want to write me a letter, by all means do, but if you want a written answer please

include a self-addressed, stamped envelope, otherwise I'll take it as just a point of interest. But please keep the calls to Monday nights.

The next thing on the agenda is a piece of software/hardware I had the pleasure of trying — the *CoCo Max* drawing package. The software part is great. The hardware part is used to read the joystick (or in this case, a mouse) more precisely than the CoCo's internal joystick connection.

In my case, it was no problem to connect this ROM pack-looking adapter to CoCo; I have a Radio Shack Multi-Pak Interface. I just plug it in and away we go. This, however, might be a

problem to users who have a disk drive and no Multi-Pak Interface: they both plug into the Expansion Slot on the side of the CoCo.

In order to have both the disk drive controller and the joystick connected together without a Multi-Pak, you need an adapter. These adapters (better known as 'Y' adapters), don't come cheap. I did an article on how to make a 'Y' adapter for your CoCo in the July 1983 issue ("Build A 'Y' Adapter For Your Disk Controller," Page 176). If you don't have it, call up RAINBOW to get a back issue. This adapter will work with the *CoCo Max* and isn't nearly as expensive.

On with the Project

Once upon a time, long, long ago in a place far, far away I did a project on how to get inverse video on the CoCo. For those of you who are not familiar with the term "inverse video," let me explain: When you turn on the CoCo, you are greeted with a green square on the screen with black letters inside — the "normal" screen. When you enter in lowercase letters, you get this black square with a green letter in it. That is how you can tell it is lowercase.

Anyway, the project was a modification so that the computer would show a totally black screen with green uppercase letters. The lowercase letters were black with a green square. Whatever was green before was now black and whatever was black before was now green. Consequently, the term "inverse video."

Included with the circuit was a switch so you could switch between inverse video and normal video. I also said if there was enough interest I would come up with a circuit that would let you do the switching (the part of the physical switch) in software. There was; I did; here it is.

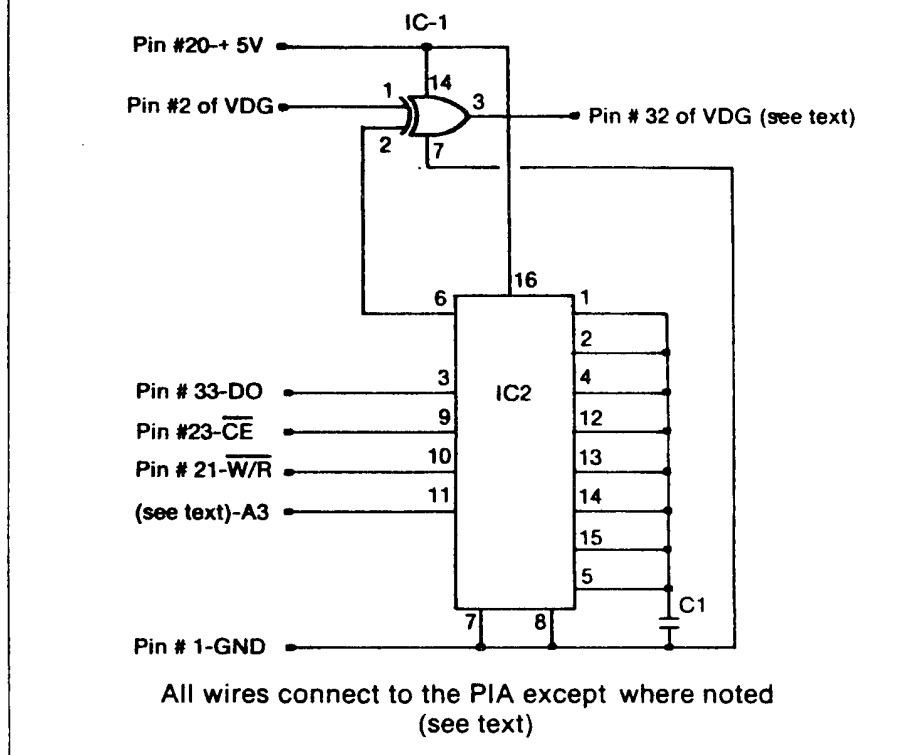
This month, I will show you the circuit and how to connect it so a simple POKE command will switch you between normal and inverse video. If you have an EPROM burner and know how to modify the DOS, you can have it built into the DOS. In order to build this circuit, you will need the standard kit builder's paraphernalia. You must also be unafraid of digging into your CoCo. As far as I know, this circuit will work on any version of the CoCo family.

The first part, as usual, is the circuit. Find in Figure 1 the circuit for this electronic switch. It consists of two chips and one capacitor. In order to understand how this circuit works, you must first know a little about the chips I used. The first chip is the 74LS86. This chip is a quadruple 2-input exclusive-or. Exclusive means one or the other but not both. This short truth table will help explain.

Input A	Input B	Output
L	L	L
L	H	H
H	L	H
H	H	L

H = High level (1) L = Low level (0)

Figure 1



In my circuit I used the 'B' input of one of the four gates as the inverter control. When 'B' is high the output is inverted with respect to the input 'A'. When 'B' is low, the output is the same with respect to the input 'A'. This is nothing new and it has been done before. A 74LS86 is almost always used for this purpose. If you were to put a switch here, you would have an inverted controller, switchable by hand. My circuit goes one further.

The next chip is a little more tricky. What is required from this part of the circuit is a type of switch action with memory, which means memory mapping (remember last month?).

This puts theory to good use. Inside the CoCo there are two possible areas we can use: either the keyboard PIA (MC6821 or MC6822) or the DAC (Digital to Analog Converter) PIA. They are mapped at \$FF00 (65280) and \$FF20 (65312), respectively. (I'll go into the differences between the two later.) Both of these I/O areas are 32 bytes long. The functions these I/O areas control (the PIA) require only four bytes, and the other 28 bytes are memory mirrored. What this chip does is decode this area into two halves. This becomes either \$FF10 (65296) or \$FF30

(65328), depending on what PIA you hook it up to.

In order to do this, we need four signals. The first is the chip enable from the PIA that selects the 32 bytes. The next is the Read/Write line, to make the memory mapped byte a write-only byte. The next signal needed is address line #3, which is the address line needed to get us the two halves of the area. The final signal needed is a data line so we can write a signal to control the 74LS86 and latch it. All this adds up to a one-bit latch that can be changed by software; our one output needed to feed the 'B' input (control) of the 74LS86.

The chip used is a 74LS151. This chip is a one-of-eight Data Selector/Multiplexer. Although the chip is not known for its latching capabilities, my good friend and co-worker Larry Callahan and I worked out a way to latch the output with the input by using a small capacitor. By feeding the output to all of the inputs but one with a capacitor, it acted as a latch with the last input as a control. We then used the inverting output as a latched bit to control the input 'B' of the 74LS86.

When a zero is written to chip it is remembered by the capacitor and other

inputs. When a 1 is written to the same location, it is remembered as well. That is all there is to it. For more information on these and all the "74LS" family of chips, refer to *The TTL Data Book* by Texas Instruments. By the way, the value of the capacitor is a mere 100 PF at 50 volts.

To construct this circuit, I used a little piece of perf board one-half inch by two inches and stuck it on top of the PIA using double-sided tape. I ran the wires along the sides to the proper pins of the PIA and the VDG (MC6845). I soldered directly to the pins of the PIA and VDG, though, some like to use a socket so as not to damage the components themselves.

All of the wires just need to be connected but, there is one pin that has to be lifted out of its socket; that is pin #32 of the VDG. If your VDG is not soldered in, just remove it, bend the pin up on an angle, replace the chip back in its socket, and then solder your wire to the lifted pin. If your VDG is soldered in, you will have to cut the pin using a sharp, pointed knife. Be careful not to damage the adjacent pins. If you use a socket, bend the pin of the socket and solder your wire to the bent socket pin. Solder all the wires to the perf board according to the schematic diagram. Solder all the wires to their respective pins on the PIA.

All the pins except A3 solder to the PIA. There are several places to get A3. It is pin #22 on the cartridge connector, pin #12 on the CPU (MC6809E), pin #5 on either ROMs or pin #19 on the

SAM (MC6883). This brings us to the PIA.

Which PIA should you use? Well, that all depends on whether you want the screen to go back to the inverse mode when you hit Reset. If you put the circuit on the DAC PIA, the software switch will stay in the same position until you turn off the computer or change it. If you put the circuit on the keyboard PIA, the screen will default to the inverse mode every time you hit the Reset key. The choice is up to you.

The poke will be the same, though the addresses will be different depending on which PIA you use. If you have it on the DAC PIA the address to POKE into is \$FF30 (65328) and the keyboard PIA has the address set to \$FF10 (65296). From now on, if you want to change the state of your screen from one to the other, and the circuit is connected to the DAC PIA, all you have to do is:

POKE 65328,0 for Inverted Video
POKE 65328,1 for Normal Video

for the keyboard PIA, the command is:

POKE 65296,0 for Inverted Video
POKE 65296,1 for Normal Video

This modification should not interfere with the normal operation of the computer. It will only invert text in the text mode. None of the graphics mode will be affected and none of the colors of the PSET or PRESET will be changed.

One more thing! The chips for this project are not available from Radio Shack, but most electronics parts mail order houses have them. There is nothing special about these chips. If you can't find them, try JDR Microdevices, 1224 S. Bascom Avenue, San Jose, CA 95128. The telephone number is (408) 995-5430.

In closing, I would like to say I have been getting a few letters from readers who are interested in making a computer storage scope using the CoCo. The average price for such a scope starts at about \$10,000. The hardware that goes into one also costs big bucks.

To try to make a project of one in this column is not quite possible. First of all, the CoCo is not a fast enough computer to make it worthwhile. Second, the hardware required would run up a bill I don't have the means of paying. And last of all, the time it would take to develop a schematic, you would be lucky to get it by 1999. But keep your ideas coming.

Parts List

Part	Description
IC-1	74LS86
IC-2	74LS151
C1	100pf 50V
Miscellaneous	Perf board, Wire, 14-pin socket, 16-pin socket

Switching Double-Sided Disks

It was great seeing the whole RAINBOW gang at the Chicago RAINBOWfest in May. That made my first RAINBOWfest anniversary. There were a lot of new products to be seen. Fancy software, new and improved hardware, and a lot of new faces.

These get-togethers are quite warm and friendly. I have gone to many computer shows, some for different kinds of computers and some that host just one brand. But, I have never seen one that came close to the atmosphere

at a RAINBOWfest. I tip my hat to the CoCo Community.

Speaking of new products, look forward to seeing my new line of products, starting with the DISTO disk controller.

Clearing up Confusion

The topic of this month's project involves disk drives and disk controllers. There seems to be some confusion about disk drives being double-sided, double-density, single-sided, single-

density, 96 or 48 tpi (tracks per inch) and the compatibility between them. Especially when you talk about OS-9.

"When the Color Computer first came out, the only mass storage available was a cassette recorder. Though the cassette

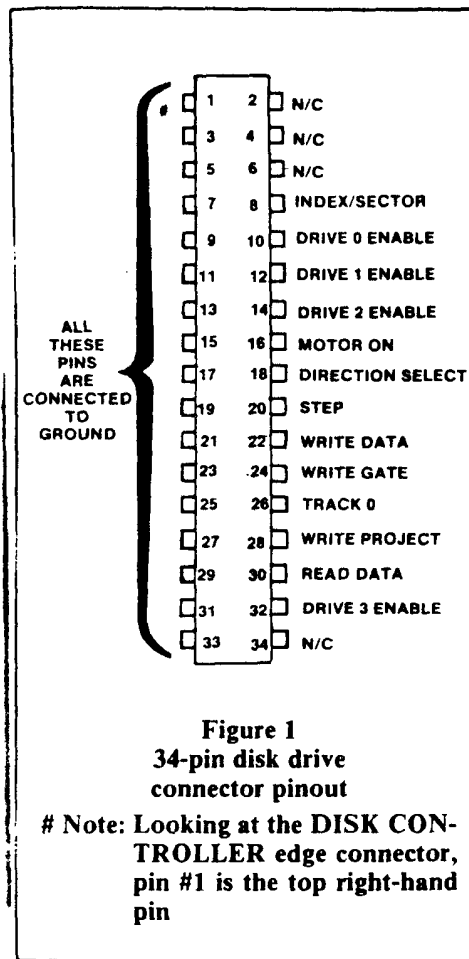
“When the Color Computer first came out, the only mass storage available was a cassette recorder. Though the cassette recorder works well for music and speech, it was slow and not well-suited for computer work. A new form of mass storage had to be invented: The diskette was introduced.”

I hope to clear all that up right here and now and follow it up with a small project to let you see just what side of the fence, uh . . . I mean disk, you are on.

I will start off by describing a diskette and a disk drive. First, a diskette is a form of media. It holds information — what information is up to you. A telephone numbers file, a game or two, your favorite word processor. All of these are files that make your computer function the way it does; this data has to be stored somewhere.

When the Color Computer first came out, the only mass storage available was a cassette recorder. Though the cassette recorder works well for music and speech, it was slow and not well-suited for computer work. A new form of mass storage had to be invented: The diskette was introduced. There are many kinds of diskettes on the market today, but I will limit this discussion to those that are compatible with our lovable CoCo.

Without going into too much detail, the Radio Shack standard diskette used with the CoCo is a 5½-inch, single-sided, double-density, 35 tracks at 48 tpi, soft-sectored diskette. The Radio Shack Disk BASIC, disk operating system, drive and controller are made to comply with these standards. You can get more details on the DOS in the Disk BASIC manual. The Radio Shack controller is made to handle two or four drives, depending on what cable you have.



The disk drive itself connects to the controller via a 34-pin ribbon connector. Figure 1 shows the pin configuration of the “disk side” of the controller. As you can see from the diagram, four pins are used for selecting or activating up to four drives. Radio Shack drives differ from standard drives by the way they are selected. You see, all four pins on Radio Shack drives are connected together and the selecting is done by missing pins in the cable connector.

For example, to select Drive 2, the cable connector that is configured to be number 2 has the pins that correspond to drive numbers 0, 1 and 3 missing. That way, when another drive is selected, it won't affect that drive because that pin is missing.

There is one more interesting thing about the Radio Shack cable configuration. Drive 3 pin on the controller is not in the normal position for a standard drive. The normal position for a standard Drive 3 is pin #6, where Radio Shack chose to keep this pin empty.

Interestingly enough, though, the place they did put it is where the standard disk drive has its side select,

pin #32 (for double-sided drives only). Since this pin is connected to the controller, it gives us access to the second side of a disk drive. All the hardware is there to use the second side, providing you have double-sided drives.

Today, the price of double-sided drives is so low that in some cases it is cheaper to buy a double-sided drive from another company than it is to buy a single-sided drive from Radio Shack. More and more people already have them and are not using the second side because Disk BASIC does not allow them to do so. I will show you a couple of ways to access the second side. One is software and the other is hardware. Use the method that suits you best. Either way, you will want to build the project if you have double-sided drives.

“There are two ways to change the mask byte in software. One is to burn the new mask byte into an EPROM. The second is to use the 64K mode of the computer and make the changes in RAM.”

The first thing to do to use the double-sided drive is make sure you have one! You must connect it to the Radio Shack controller. Remember, I said there were pins missing in the Radio Shack cable and that will give us problems.

The side select pin is only present on a four-drive cable, and then only on the fourth drive. You must add another connector for every double-sided drive you are adding to your system. (They are available at your nearest Radio Shack Computer Center.) The connector is a 34-pin edge card connector. If you don't know how to install it on your cable, ask your dealer to do it for you. Have him press the new connector about an inch and a half away from the old connector.

The disk drive now has to be configured to which drive number you want. There are jumpers inside the drive you must set. In the owner's manual of the drive there will be instructions on how to do that.

Now you have a double-sided drive on line, but you will still need a way

to access it. The first way is in software. The way Disk BASIC selects the drive is by using four "mask" bytes. Each byte contains the necessary data in order to activate that drive number. There are four bits that control each of the output pins as seen in Figure 1.

In the controller, there is a memory-mapped byte that controls the output of these pins. It is at \$FF40 or 65344. Try this:

POKE 65344,1

The select light on Drive 0 turned on.

Now try the values two, four and 64 instead of one. This will turn on drive numbers 1, 2 and 3, respectively. The last value of 64 activates Drive 3 (if you have four drives), but remember on our double-sided drive that is the side select. By changing the values on the four mask bytes we can access the second side of the drive. By changing the mask data, you can access the second side of the drive as another drive.

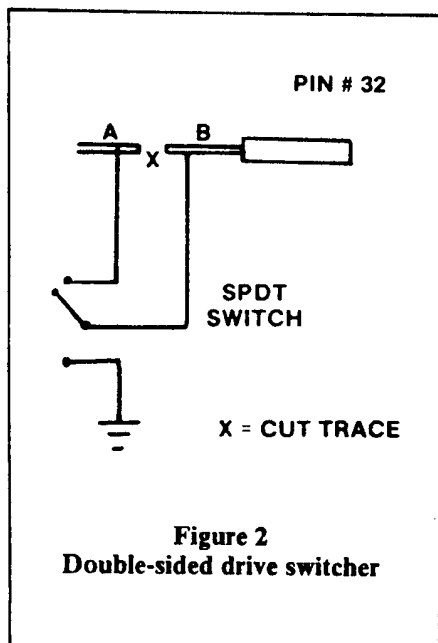


Figure 2
Double-sided drive switcher

Example: If you have one single-sided Radio Shack drive and one double-sided drive with the right changes to the mask byte, you will have three drives on line. The Radio Shack drive is the first, the first side of the double-sided drive is the second and the second side of the double-sided drive is third. If you had two double-sided drives, it would be as if you had four separate drives. Two double-sided drives is the maximum you can have with Disk BASIC because there are only four mask bytes.

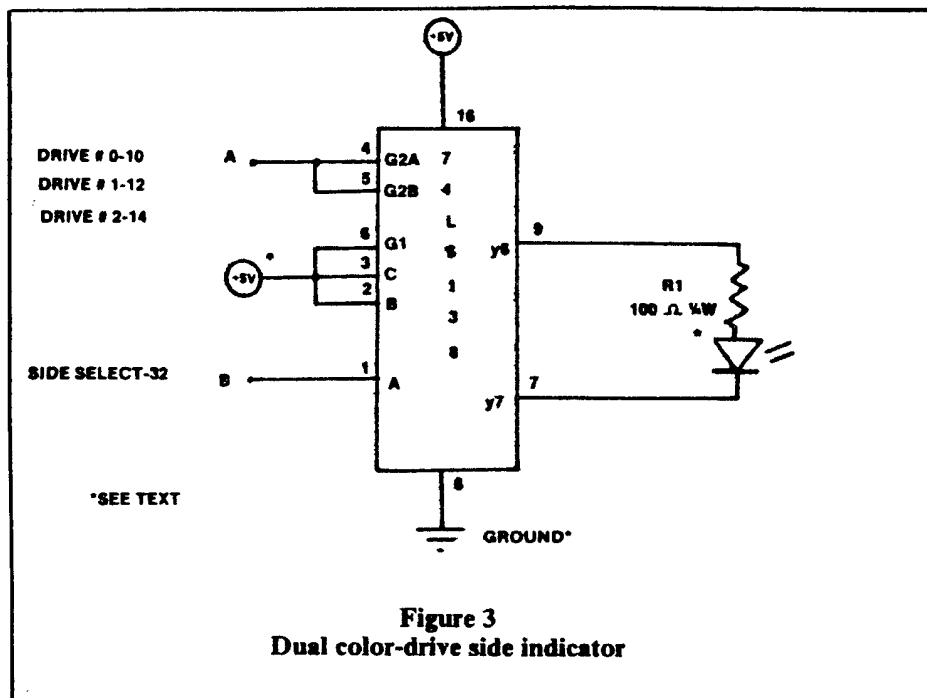


Figure 3
Dual color-drive side indicator

There are two ways to change the mask byte in software. One is to burn the new mask byte into an EPROM. The second is to use the 64K mode of the computer and make the changes in RAM. I'll leave that part up to you, but what I will do is tell you how to change the mask byte.

The four mask bytes correspond to four drives. Since we are using the last drive number as a side select, we can no longer use it as a drive but only as a side select. That leaves us with three other mask byte values. The values are 1, 2 and 4. The side select value is 64. Any combination of this will work (maximum of four).

First example: Your first drive is a Radio Shack single-sided drive. You want it to be Drive 0, so the value of the first mask byte is '1.' Your second drive is a double-sided drive; they will be Drive 1 and Drive 2. The second mask byte will be '2' and the third byte will be 65. The fourth byte will be untouched.

Second example: You have two double-sided drives. Drive 0 will be the normal side of the first drive; Drive 1 will be the normal side of the second drive. Drive 2 will be the second side of the first drive and Drive 3 will be the second side of the second drive. The four mask bytes are 1, 2, 65, 66.

Radio Shack has two versions of DOS: 1.0 and 1.1. The memory address of the four mask bytes for DOS 1.0 is \$D7AA (55210); the address mask

bytes for DOS 1.1 is \$D89D (55453), plus the next three bytes for the other three values.

If all that doesn't thrill you, you can select the other side by adding a small switch to your disk controller. Figure 2 shows how to hook up the switch to your controller. You must cut the foil between points 'A' and 'B.' Drill a suitable hole in the cover of the controller to mount the switch. When the switch is in the up position, the normal sides of all double-sided drives are accessed. When the switch is in the down position, the second side is accessed. Never change the switch when doing I/O to disk since it will ruin both sides. Again, remember, you must not use the fourth drive on a four-drive connector.

To some, it is easier to install the switch than to do it in software, but it is a little more difficult to manually flip the switch. In any case, visual cue as to what side of the disk you are really on is almost a necessary option.

Figure 3 is a schematic for a circuit that will tell you what side of the drive you are using by lighting a different color LED for each side. This circuit goes inside the disk drive and replaces the "active drive" select LED. The heart of the circuit is the Radio Shack Tri-Color LED (part #276-035). This LED glows one of three colors. We will be using only two of these colors, red and green. The circuit uses a 74LS138 decoder.

When no drive is selected, the two outputs used are logical level one and the LED is off. When the drive in question is selected, the 'A' (drive select) input goes low, therefore activating the chip. If the 'B' (side select) is high (first side of the drive), the Y7 output goes low. This will cause a positive voltage to appear across the Tri-Color LED which makes the LED glow red. If the 'B' input is low, the Y6 output goes low, in which case there will be a negative voltage across the LED. Then the LED will glow green. When the 'A' input is high (drive not selected) the chip is disabled and both Y7 and Y6 are high, the LED will be off. I put red as the first side because it is the color of a single-sided drive. That way when I see green, I automatically know I'm on the

other side.

There are just a few things to consider when hooking up this circuit inside the drive. The first is where to get the five volts and ground needed to run the circuit. The easiest place to get a ground is pin #1 of the drive cable connector. Pin #1 is on the side of the connector that has all the pins connected together. They are all the odd-numbered pins. The drive connector pins are numbered on each end.

Five volts can be taken from the last pin of any 74LSXX chip. Use a volt meter to check the voltage. This is either pin #14 or #16 depending on how many pins there are on that chip.

The second thing to watch for is to make sure the 'A' input matches that of the drive selected. This means if the

'A' wire goes on Drive 0, make sure the drive configuration block is set to Drive 0, otherwise the LED will never light.

The actual construction of the circuit can be done on a small perf board. Tape or glue down the board in an unused area of the disk drive. Make sure it doesn't get in the way of the diskette that enters the drive. Remove the old LED. Replace it with the new one. Use tape or glue to hold it down.

Now, try the drive and access the first side of the drive. The LED should be red. If it is green, reverse the wires that go to the LED. When all is OK, the LED will glow red for the first side and green for the second side. This way you will always know which side of the drive the software is accessing.

Making CoCo Shine With More LEDs

It seems to me a lot of people like to do projects that light up or make noise. The projects I get the most response from are the ones that involve LEDs (Light Emitting Diodes). Well, who am I to argue with my readers? (I'll let you in on a little secret — I like them, too!) In order to keep my readers happy, here is another one.

This month's project is a two-fold project, and maybe a little more. The first part involves three LEDs. These LEDs will be connected to the RS-232 port. The second part is a Reset button mounted up front. A Reset button up front may not be a new idea, but the way I do it the wires will not get in anyone's way. The part about "maybe a little more" means that if ever you want to add more things to your "cover," there are leftover wires. If you are confused, read on; it is all explained in this month's article.

The Color Computer's RS-232 port has four wires, three lines and one ground. There are two inputs and one

output. As so labeled by Radio Shack, the two inputs are RS-232 "IN" and Carrier Detect, "CD." The third line is an output known as the RS-232 "OUT."

These three lines can take the standard RS-232C levels. The "level" in this case means at what voltage level the computer considers a logic level of one (logic level HI) or logic level of zero (logic level LO). The levels for standard RS-232C

"If you find the LEDs never light, try soldering them in backwards."

are plus 12 volts (+12V) and minus 12 volts (-12V). These levels are maximum levels. The Color Computer also has one RS-232 output. The standard for RS-232C output is also plus/minus 12 volts.

Now, here is where the interesting part begins. According to the EIA (Electronic Industries Association), a voltage above plus three volts shall be considered a logical one and a voltage

below minus three volts shall be considered a logical zero. Any voltage in between these two limits will be considered undefined. That means if a voltage is 12 volts and on its way down, the logic level will not change from a one to a zero until it reaches minus three volts. The same is true for a voltage on its way up.

As far as the RS-232 is concerned, there are essentially two Color Computers. All of the "big" Color Computers (the older gray or white models) are the same. All of the "small" Color Computer 2s are also the same. They differ only in the output voltage levels. The CoCo outputs the full plus/minus 12 volts, while the CoCo 2 only outputs plus/minus five volts. On the input side, the CoCo can safely handle plus/minus 12 volts while the CoCo 2 can take up to plus/minus 25 volts.

I mentioned that the voltage levels less than three volts and greater than minus three volts are undefined. This is done to improve the reliability of RS-232 communications. It improves the noise margin level. For example, a signal that fluctuates a volt or two will not pass threshold level; therefore will not produce false data.

(Tony DiStefano is well-known as an early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.)

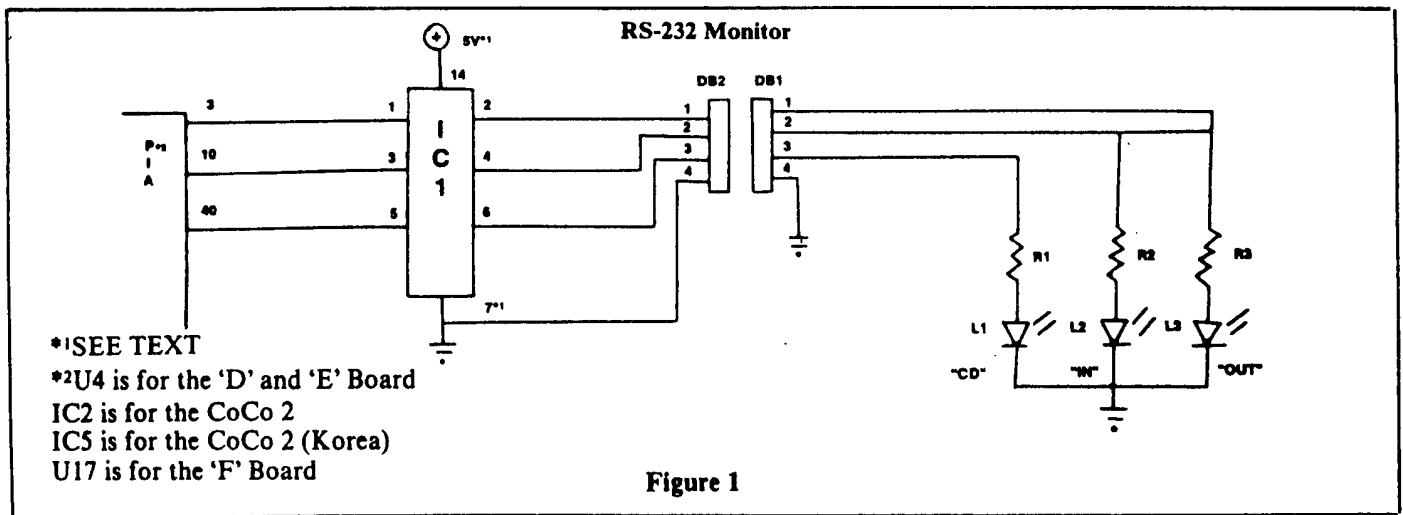


Figure 1

This method of transmitting data is more reliable, but the electronic circuitry needed is also more expensive to produce. Radio Shack did not spend the extra money. Instead, they made a circuit that has no margin of safety. In the case of the CoCo, any voltage greater than 2.6 volts is considered to be a logic level of one, and any voltage less than 2.6 is considered to be a logic level of zero. The CoCo 2's voltage level is also 2.6, but I think on some CoCo 2s it is set to 2.0 volts.

That takes care of the "ins" and "outs" of RS-232 in the Color Computer; now let's get down to the hardware part. I have had several letters and phone calls about RS-232 compatibility between the CoCo 2 and the CoCo, or the CoCo 2 and some other peripheral such as a modem. This may not solve your problem, but it will focus on whether or not the computer is the problem because this month's project is an RS-232 monitor. Since there are only three lines on the CoCo's RS-232, you will need three LEDs.

The schematic in Figure 1 shows the simple circuit involved in building the RS-232 monitor. The chip I used is a TTL (Transistor-Transistor-Logic) chip. I used a buffer/inverter to drive the LEDs. Only three of the six buffer/inverters are used; the other three are unused and are free to be used in another project.

The inputs to these buffers come from the PIA (Peripheral Interface Adapter) that controls the RS-232. I took the signals from these points because the voltage levels are compatible with the TTL chip used. Also, these points are the points the computer sees and not what is coming in on the RS-232 lines.

The chip I used inverts the signal. I did it that way because on power-up, all three signals on my computer are

ones. Normally that would mean all the LEDs would be on. To me, that is a bit distracting. The inverter turned all the LEDs off when I powered up.

The logic here is that everything is off until you use something. For instance, when the printer is online, the "IN" LED would light up. Another example is when my modem is on, the "CD" LED is on when there is a carrier — not the other way around. If the LED says there is a zero (by being on) the computer sees a one. This way you can visually see exactly what the computer reads and writes. You will be able to see, at a glance, whether your modem is online, your printer is busy or when the computer is transmitting something. All in all, it monitors all RS-232 functions.

If you don't like the fact that the LED

information is inverted, there is a simple way to change it. First, reverse all the LEDs' polarity by plugging them in backwards and, instead of connecting the other (common) side to ground, connect it to the five-volt side. This will inverse the inverter, giving you a non-inversed signal. When you see the LED on, there is a one on the RS-232 line in question. Do it whichever way you choose.

This is the major part of the project, but equally important are how and where the LEDs are mounted. I mounted the three LEDs on the top cover of my CoCo. At first thought, there is nothing unusual about this, but if you are like me, the cover is always unscrewed. I must pull the cover off my computer at least a dozen times a week. If I have a lot of wires going to the cover, chances

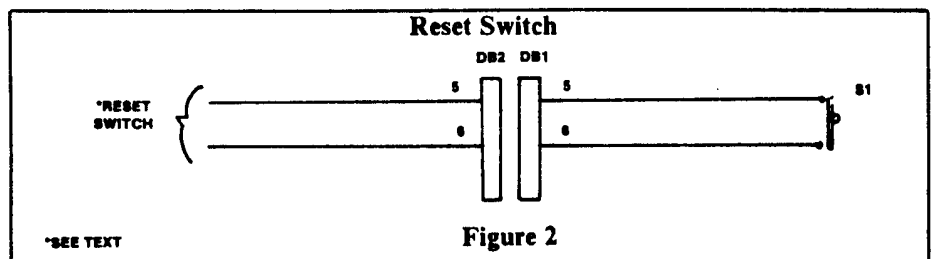


Figure 2

Parts List

ID	Description	Radio Shack Number
R1,2,3	100 ohms ¼ watt	271-1311
IC1	74LS04 or 7404	276-1802
L1,2,3	LED (with mount)	276-018
DB1	Sub-D Male	276-1537
DB2	Sub-D Female	276-1538
S1	SPST Momentary	275-1547
Miscellaneous	Wire, solder, proto-board, etc.	

are a few of them will break off before the week is out, so I put in a connector. That way, when I remove the cover to dive into my CoCo, all I have to do is disconnect it and the cover is completely removable. It is a good idea that saves resoldering the wires every time one breaks off.

Though any connector can be used, I used a DB-9 male and a DB-9 female for two reasons: 1) They are both available from Radio Shack, and 2) It is impossible to plug them in backwards. Leave eight to 10 inches of wire on each side of the connector to give plenty of slack, but be careful that the connector does not short out when you stuff it inside the CoCo. Some kind of sleeve would be good. The chip can be mounted on an optional piece of proto-

board or glued upside-down on top of the PIA. Again, this is your choice.

The rest of the circuit is quite straightforward. If you find the LEDs never light, try soldering them in backwards. I always have trouble finding the anode to those things. Some people have expressed difficulty in finding where to connect to get plus five volts and ground. A good place to find plus five volts is on Pin 9 of the edge connector. Finding Pin 9 is simple. Start from the back of the connector (the part closest to the rear of the computer) and count the top pins 1, 3, 5, 7 and 9, and there you are. Ground is on Pin 33; count the same way.

As an added bonus to the RS-232 monitor, I added a Reset switch in the front of my computer. The circuit in

Figure 2 shows how to add it in. Mount the switch on the front cover on whichever side that suits your needs. Use two of the unused pins on the DB-9 connector to do the wiring, so that way you can disconnect it from the computer at the same time you disconnect the LEDs.

There are about as many different reset switches as there are different CoCos. Some of them have six pins, and some have only two. The one that has only two is easy to wire; put one on each and there you go. The ones that have six pins are a bit different. The easiest way to figure out which one is which is use a short piece of wire. With the computer on, touch any two pins on the reset switch with each end of the wire. When a reset occurs, those are the two pins to use.

The Analog-To-Digital Converter, Part 1

The world inside your computer consists of zeros and ones — all that goes on inside your computer hinges on two values. Memory, PIAs, CPUs, VDGs and SAM chips all transfer information between each other using only two different states. These states are called *logic states*.

The first logic state is zero, also known as "low" or "lo." In the Color Computer (and most computers) a logic state low is zero volts, also known as *ground level*. The second logic state is one, also known as "high" or "hi." Again, in the Color Computer, a logic state high is five volts. Except for specified tolerances, all other voltages in between are undefined and if encountered can give the computer some unpredictable results. This is the digital universe of computers. Figure 1 shows a typical digital wave form.

The real world, however, deals in ever changing states. Digital ones and zeros are just two of millions of different states that exist. The real world is an analog world. A good example of the

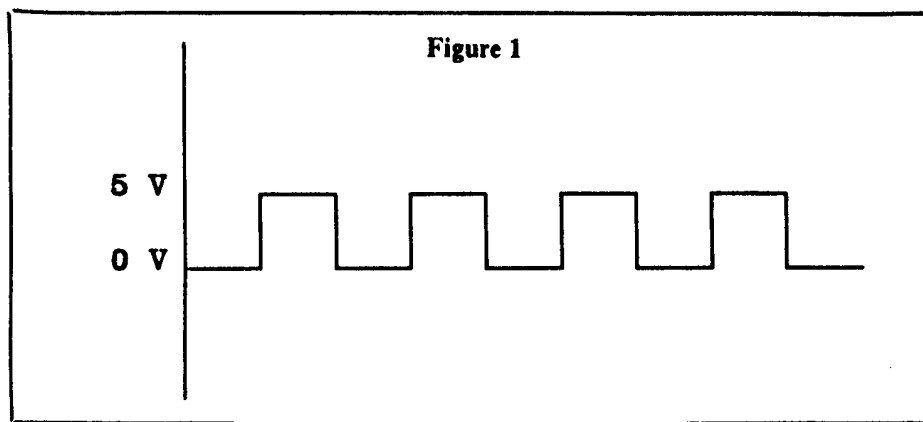
analog world is speech. You can speak loud, you can speak low or many levels in between.

Sound waves are ever changing. For example, if you take a microphone and amplifier and hum into it, the speaker will vibrate, reproducing the sound you are making. That vibration is a back and forth motion. The frequency of the back and forth motion depends on the frequency of your hum. Frequency is measured by how many times a wave form goes back and forth in one second. Every time the speaker moves back and forth is one cycle.

From 1886 to 1888, the work of Heinrich Rudolph Hertz led to his

discovery of electromagnetic waves. The German physicist's revelation opened the way for the development of radio, television and radar. As a tribute to him, the frequency of any wave, be it digital or analog, is measured in hertz (or Hz, for short). In the audible range, the frequency is from about 20 Hz to 20,000 Hz or 20 kHz. The 'k' stands for "kilo" meaning thousand. Our CoCos, for instance, run at 894,000 Hz or .9 MHz. The 'M' stands for "mega" meaning million.

Figure 2 shows a graphic representation of the output of a sound wave. Compare it to the wave form in Figure 1. There are some obvious differences;



(Tony DiStefano is well-known as an early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.)

it is these differences that make it impossible for a computer to directly and accurately read and duplicate an analog wave form.

Don't despair, there are ways around it. This is the first of a two-part project on how you can use a computer to measure analog signals. This project stems from several letters received from my readers requesting that I build a computerized oscilloscope adapter for the joystick port. I looked into the joystick port as an input, but found it to be inaccurate or not fast enough. By the time you finish reading this, you'll know why.

Anyway, this month we'll cover the theory on how a computer (and a little hardware) can convert an analog signal to a digital value. Next month we'll cover how to build and calibrate the analog to digital converter.

Now to the task of explaining how a computer can convert an analog signal to a digital value. The first thing the computer needs is some hardware, a comparator. A comparator is an IC that has two inputs (the "positive" input and the "negative" input) and one output.

The output has two states; on or off, good for a digital computer. The inputs, however, have analog inputs.

Here is how a comparator works. When the positive input voltage is higher (more voltage) than the negative input, the output is high. When the positive input voltage is lower (less voltage) than the negative input, the output is low. Figure 3 shows a block diagram of a computer-controlled comparator.

The way it works is simple. If we had a known voltage at the negative input, by reading the output (high or low) we could tell if the test voltage at the positive input is higher or lower than our reference voltage. Furthermore, if we change our reference voltage and zero into the unknown voltage, we will then know what the unknown voltage is. This technique is known as successive approximation.

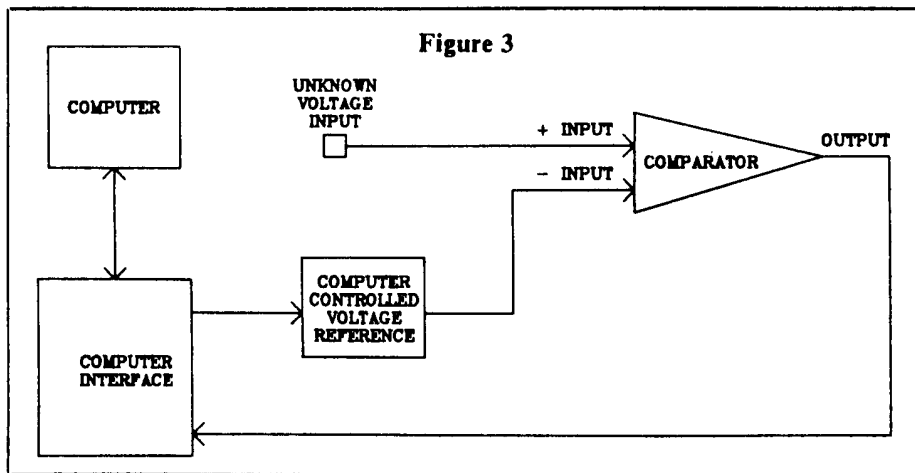
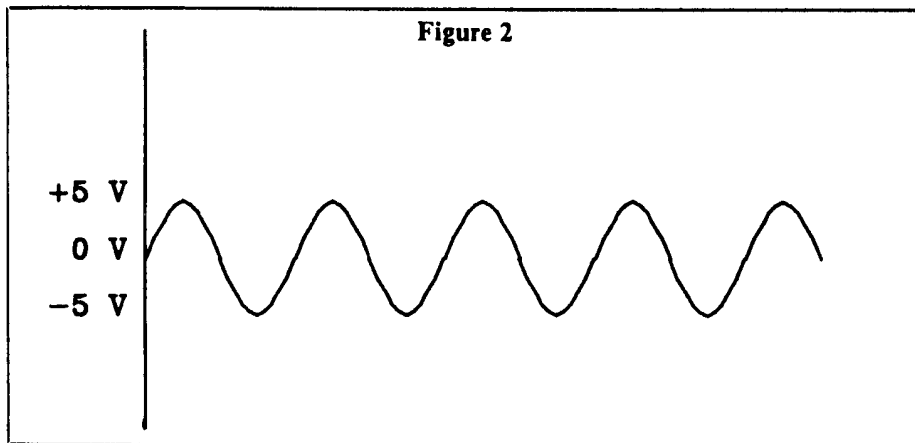
The procedure for successive approximation is as follows: Start by putting half of the maximum voltage your device can measure to the reference voltage. If the output of the comparator is high, that means the unknown voltage

is higher than the reference voltage. We then increase the reference voltage by half the difference of the present value and the last value and test again. If the output of the comparator is low, that means the unknown voltage is lower than the reference voltage. We would then decrease the reference voltage by half the difference and test again. Continue to do this until we have reached the unknown voltage.

Let's take an example. In this example, I round off the reference voltage to the nearest whole number for ease of calculation. The maximum voltage is 100 and the unknown voltage 47. The first reference value is 50 — too high, so we subtract from the present value using the successive approximation method. New reference is now $(100-50)/2$ or 25; the new reference is 25 — too low, so we add. The new reference is now $(25+50)/2$ or 37.5; the new reference is 38, still too low. Add again, $(38+50)/2$ or 44. The new reference is 45, again too low. Add $(45+50)/2$ or 47.5 and the new reference is 49. That's too high, so subtract $(49-47)/2$ or 1. The new reference is now 46, which is too low, so add $(46+47)/2$ or 46.5. The new reference voltage is now 48. Too high, so subtract $(48-47)/2$ or 0.5. We have now reached the point where our reference voltage matches the unknown voltage.

Actually, the rounding off is not limited to integer calculation, but rather to the resolution of the reference voltage. When zeroing into the unknown voltage, you divide until the unit change in voltage is one. You cannot divide and get a more accurate fix on the unknown value. No matter how close you get, the comparator will always give a higher or lower value. The more accurate the reference, the closer you get to the real value of the unknown voltage.

This reference accuracy is one of the reasons why I chose not to use the joystick input. You see, inside the Color Computer there is all of the previously mentioned circuitry: a voltage comparator, a variable voltage reference, an unknown voltage input (joystick) and the interfacing circuit to control it all. A more common name for a variable voltage reference is "Digital-to-Analog Converter" or DAC for short. The DAC inside the CoCo is limited. It has a fixed output of .25 to 4.75 volts and the resolution of about 0.0715 volts.



The range is not very good for an analog-to-digital project.

Another reason for not using the joystick input is speed. You see, the successive approximation method talked about earlier is time-consuming. The CPU has to calculate the next reference voltage value, set up the DAC, read the comparator output and make the proper decision.

The speed at which an unknown voltage can be found is very important. When the unknown voltage is stable and not changing, the computer can take all the time in the world to figure out what the voltage is. But, if the unknown voltage is changing, like the humming mentioned earlier, speed is important.

The amount of time it takes the CPU or other device to find an unknown voltage value is called the "conversion time." The faster the conversion time, the more samples can be taken and the more accurate the wave shape reproduction can be. For example, if you have a loop to read the A to D converter which takes 10 ms (ms = milliseconds = 1/1000 seconds), that means you have

100 samples per second. If you are sampling a wave form that is 1,000 Hz, you will miss a lot of information. It is safe to say you need at least 10 times the sample rate to reproduce a particular sine wave with reasonable accuracy.

A BASIC loop using the JOYSTK command will limit you to about 3 Hz — not very fast. In machine language, you can get a lot faster, but it is still slow due to the overhead created by the CPU having to do the conversion. In the case that the CPU has an external A to D converter, a great increase in speed and accuracy can be achieved. With the right software the effective conversion rate for an external A to D could be as high as 800 hertz.

The last thing I must mention this month is that the A to D circuit requires negative voltage. This is no problem with the first CoCos, but it is with the CoCo 2. The CoCo 2 has no negative voltage available at the cartridge connector. There is, however, negative voltage available inside the CoCo 2.

To bring this voltage to the cartridge connector is simple; you just need one

piece of wire and a soldering iron. First, unplug the computer, then open it and locate the chip with the number SC77527; this is the SALT chip. You will find -12 volts on pin 15 of this chip (just what the doctor ordered). Solder one end of a piece of wire to that pin. Locate pin #1 of the cartridge connector (it is the top pin closest to the back of the computer) and solder the other end of wire to this pin. Before you plug anything into the computer, measure the voltage to that pin. It should be about -12 volts, give or take two volts.

On the CoCo 2 this pin is normally not connected to anything. On the regular CoCo, this is the regulated -12 volt pin. The -12 volts you just added to that pin are not regulated, but in this and most cases, it will not matter. There will be a negative voltage regulator on the A to D convertor. Of all the peripherals I have seen for the CoCo, only one uses the negative voltage and it doesn't matter that it is not regulated.

If all is well, close your computer and I'll see you next month with Part 2 of the A to D converter. ☺

Putting The Finishing Touches On The Analog-To-Digital Converter

This month we'll finish the analog-to-digital project we started last month. The most important part of this project is the chip that does all the work. There are many chips on the market today, ranging from very cheap and slow to extremely fast and expensive. My budget (and I am sure I'm not alone) is very tight. I found this chip in a local electronics surplus store and paid a little less than two dollars for it. The chip is the Teledyne Semiconductor number 8700CJ. It is an eight-bit analog-to-digital converter.

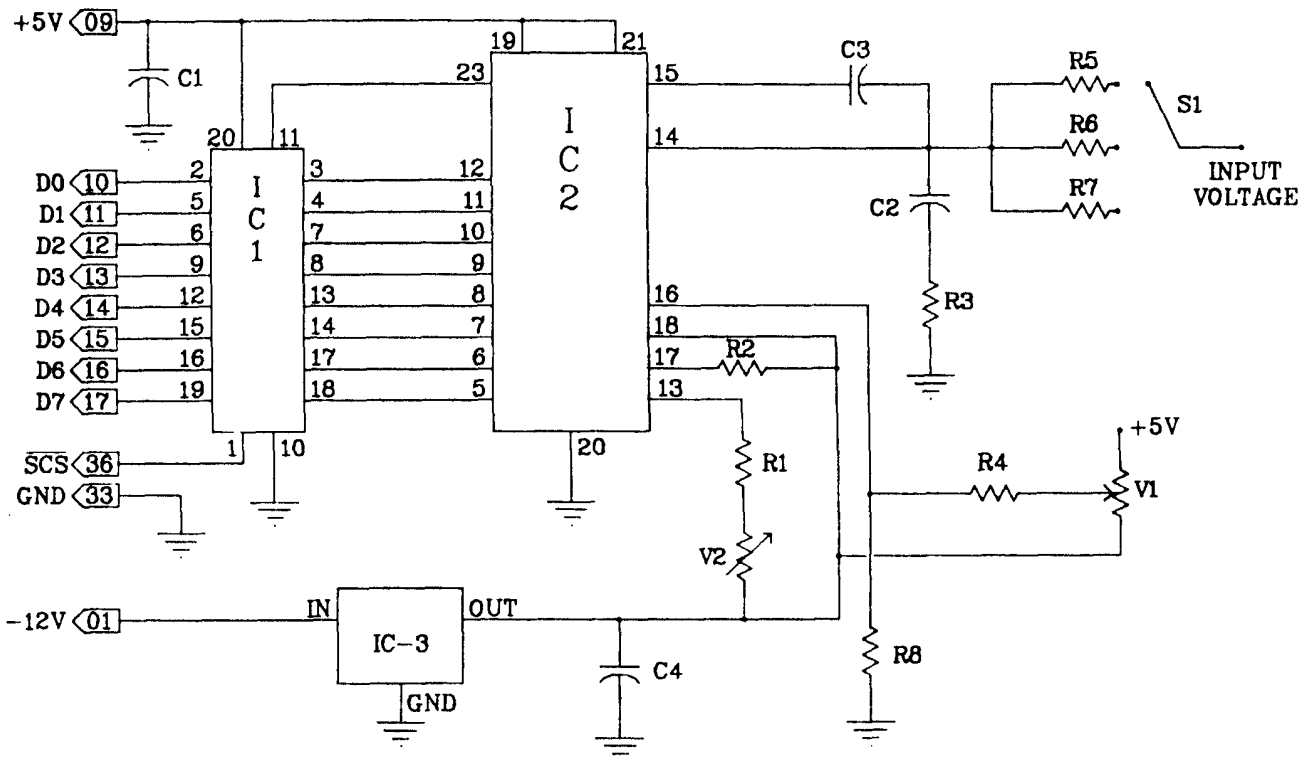
This converter is a fully self-contained, single 24-pin, dual in-line package. The circuit requires only passive support components. The con-

version technique used in this chip is a bit different than the one I talked about last month, but the net results are quite the same. Conversion is performed by an incremental charge balancing technique that has inherently high accuracy, linearity and noise immunity.

An amplifier integrates the sum of the unknown analog current and pulses of reference current. The number of pulses (charge increments) needed to maintain the amplifier summing junction near zero is counted. At the end of conversion the total count is transferred into the eight digital outputs. Figure 1 shows the pinout of the 8700CJ analog-to-digital converter. The following is a pin-by-pin description of this converter.

Pin #	Description
1 to 4	No connection
5 to 12	Eight data lines — These output-only data lines represent the eight-bit value as a result of the conversion. Pin #5 is the most significant bit, Bit 7. Pin #12 is the least significant bit, Bit 0.
13	Iref — This the reference input current used to compare to the unknown current.
14	Iin — This is the unknown input current to be measured.
15	AMPout — The output of the first comparator. Used to limit high frequency oscillation.

Figure 2



Analog-to-Digital Converter

circuit is finished and ready to be tested, insert the converter and power up. Like most of my projects, this one is made to work with the cartridge connector on the side of the computer. It will not work with a disk drive controller plugged in because it uses the SCS line and is memory mapped at \$FF40 or 65344. If you want to change where it is mapped, read my article, "How to Follow a Memory Map," in the June 1985 RAINBOW. It will, however, work with a Multi-Pak Interface.

Follow the procedures with the MPI to set it up. In order to see if all is working well, a simple program is necessary.

```
10 CLS
20 PRINT @0, PEEK(65344) : GOTO 20
```

Run the program. Touch and let go the junction of C3 and C2 with your finger. The number on the screen should change value. If it does, all is well and you are ready for the adjustment procedures. If it doesn't, check over the circuit, repair the problem and try again.

The adjustment procedure is simple. The first adjustment is the *zero adjust*. Ground the input, that is, add a jumper from the input pin to ground. Adjust V1 until the value on the screen reads zero.

Increase V1 until it just changes to one and then back off until it changes back to zero. Now remove the ground clip and enter a reference voltage. This reference voltage should be the full-scale voltage of the resistor selected above. This is the full-scale adjustment. For instance, if R6 is selected and the value is one megohm, the full-scale voltage is 10 volts. Put a known 10-volt source to the input.

Different resistor values require different full-scale voltages. Adjust V2 until you read 254. Increase V2 until it just turns to 255. Go back to the zero adjust and check it again. Do this until both adjustments are right. If your values for R5, R6 and R7 are accurate all the other scales will follow. The accuracy depends on the accuracy of these resistors. If you are a real stickler, you can add a trim pot on every resistor and adjust each full-scale separately. That is all the adjustments you have to do.

That covers the hardware end of an analog-to-digital converter. There are a few things to remember about the circuit. First of all, it is only good for positive voltages. Negative voltages will register only as zero. It will not, however, hurt the converter. There is a way of biasing the converter to except neg-

ative voltages. If enough readers are interested, I'll do another article on how to expand on this converter.

The input impedance depends on the full-scale resistor. It will typically range from 100K ohms to about 10 megohms. The possible uses for this type of circuit are endless. First, it is a voltage meter, used for measuring voltages of batteries, transformers, adapters, other circuits and many more. But, for most of these items it is simpler to use a \$5 Radio Shack volt meter.

So why the fancy-pants converter? Well, there are many purposes. With the proper input device, one could make a long term study of outside temperature patterns. Another would be the slow changes of alpha waves in meditation. With the right software you could use your computer as an oscilloscope or even control the temperature of your house. I can think of many things, just use your imagination.

As always, if you have a question or a problem and absolutely can't wait for the post office, call me on Monday nights *only*, and after supper, at (514) 473-4910. If you write and want a response, include a self-addressed, stamped envelope; my address is 4680 18th Street, Laval Quest, Quebec H7R 2P9. Sorry, I don't do windows.

Parts List

IC1	— 74LS374 octal flip-flop	R5	— 10K ohms ¼ watt one percent metal film	C4	— .1 uf 25 volts
IC2	— 8700CJ eight-bit A-to-D (teledyne semiconductor)	R6	— 1 meg ohms ¼ watt one percent metal film	V1	— 20K trim pot
IC3	— 7905 -5 volt regulator	R7	— 10 meg ohms ¼ watt one percent metal film	V2	— 50K trim pot
R1	— 320K ohms ¼ watt	R8	— 1K ohms ¼ watt	S1	— SPTT rotary switch
R2	— 100K ohms ¼ watt	C1	— .1 uf 25 volts	Misc.	— 24-pin socket, 20-pin socket, CoCo proto board, wire, solder, case, etc.
R3	— 100 ohms ¼ watt	C2	— 270 pf 25 volts		
R4	— 100K ohms ¼ watt	C3	— 68 pf 25 volts		

A helpful list of some computer acronyms and abbreviations

What Is A VDG, Anyway?

I wrote an article on how to add a video output to the CoCo in the May 1984 issue of RAINBOW. To this very day, I still get letters about it; I decided to take a moment and answer the most common questions. The single most-asked question is "Where (or what) is a VDG?"

All the chips in the Color Computer have part numbers that identify them. While numbers are good for ordering and sorting, they say very little about what the chip does. The makers of these chips have given names to them that describe their respective functions. For instance, the heart of the CoCo is a microprocessing unit designed and manufactured by Motorola. Motorola gave this part the code number of MC6809. For technicians who are very familiar with chip numbers, it is no problem to remember that a MC6809 is a microprocessing unit.

There are so many different chips made by different companies (which do basically the same thing) that numbers no longer have a clear meaning. Especially when talking about computers in

general and not about one specific model. People started calling chips with a specific function by nicknames. A MicroProcessing Unit soon came to be known as an "MPU." This is known as an *acronym*. An acronym is a word formed from the first letter or group of letters of a series of words. There are a lot of abbreviations and acronyms in computer jargon. Some of them are directly related to the CoCo and some are not.

I have compiled a list of all the

Acronyms and Abbreviations

ACIA (*used in the Deluxe RS-232 Pak) — Asynchronous Communications Interface Adapter. Used for serial data.

A/D — Analog-to-Digital. A chip that converts an analog voltage to a digital value.

ALU — Arithmetic Logic Unit. Used to perform binary arithmetic functions.

ANSI — American National Standards Institute.

"There are so many different chips made by different companies (which do basically the same thing) that numbers no longer have a clear meaning."

abbreviations and acronyms for computer parts I could think of. The ones marked with an asterisk (*) mean that the CoCo has one of them inside or uses it in one of its add-ons, such as a disk drive. Along with the acronyms is a full name and short description. Not all of the acronyms represent one chip — some may represent a group of chips and some represent a type of standard. I am open to letters for the ones I may have missed and will write an update as soon as I can.

ASCII — American Standard Code for Information Interchange. Better known as ASCII characters. The format is such that all alphanumeric and special characters on a typical computer keyboard are given a specific numeric value. Anyone using the ASCII standard will use the same values.

BASIC (The language, as in Color BASIC 1.1) — Beginners' All-purpose Symbolic Instruction Code. (Bet you didn't know that one, ay?)

BCD — Binary Coded Decimal.

(Tony DiStefano is well-known as an early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.)

- 16 ZEROadjust — This input is used to adjust so a zero voltage will be accurate.
- 17 I_{bias} — This input current adjusts between the speed of the conversion and the supply current. The faster the conversion, the more current it requires.
- 18 VSS — This pin requires 5 volts power supply.
- 19 VDD — This pin requires +5 volts power supply.
- 20 GND — This pin is the system ground.
- 21 INIT — This pin is a TTL level input used to start the conversion process. Can also be connected to be free-running.
- 22 BUSY — This pin is a TTL level output. When it is high (logic 1), the converter is busy calculating the next value.
- 23 VALID — This pin is a TTL level output that is high when the data at the eight data pins is latched with valid data.
- 24 Another pin with no connection.

Figure 2 shows the circuit I designed for this project. IC #2 is the converter chip — it is the heart of the project. All the pins described need not be repeated, however, there are a couple of other support chips that could use a little explanation.

The first (IC #1) is a 74LS374. This

is an eight-bit, D-Type flip-flop with tri-state outputs. It is used to store the data produced by the converter and to act as a buffer to the computer. The converter is wired in a free-running mode. That means as soon as it is finished doing a conversion, it immediately starts again as opposed to waiting for a signal from the computer to do another conversion. The *data valid* pin of the converter is connected to the *clock* pin of the 374, therefore transferring valid data from the converter to the flip-flops. Data is transferred from the converter to the flip-flop on the rising edge of the signal only, therefore no data is lost when the converter is busy doing the next conversion.

IC #3 is the other chip needed to make this work. It is a voltage regulator; a negative voltage regulator at that! It can take any negative input voltage from about -8 volts to -30 volts. The output will be a regulated -5 volts.

Why all this negative voltage? Well, the converter is kind of fussy that way. It needs -5 volts to work (something to do with the linearity I am told). If you are using a regular CoCo or a Multi-Pak Interface, there is no problem, but if you have a CoCo 2, you will have to fish out some negative voltage. (See last month's issue on how to do that.)

The 7905 is a three-pin chip that looks more like a power transistor than an IC. The pin numbers and description of this chip are simple. Looking at the chip and legs pointing downward, the left-most pin is ground. The center pin is the input and the right-most pin is the output.

The IC does not need to be mounted on a heatsink; there is not enough power demanded of it. It also does not need a socket.

The rest of the parts are just to make the converter work properly. There are only two adjustments to make; I'll get to that later, but now I would like to focus your attention on the three resistors, R5, R6, R7, and switch S1. You may or may not want to include these in your final circuit. You may want even more than three resistors. It all depends on what you want to use this circuit for.

The input resistor, R5, R6 or R7, depending on which one is in circuit at the time, is a scaling resistor. The value of this resistor will determine what the full-scale voltage value will be. To determine the full scale voltage, you must follow this simple formula: $R_{in} = V_{in \text{ full scale}} / 10\mu A$.

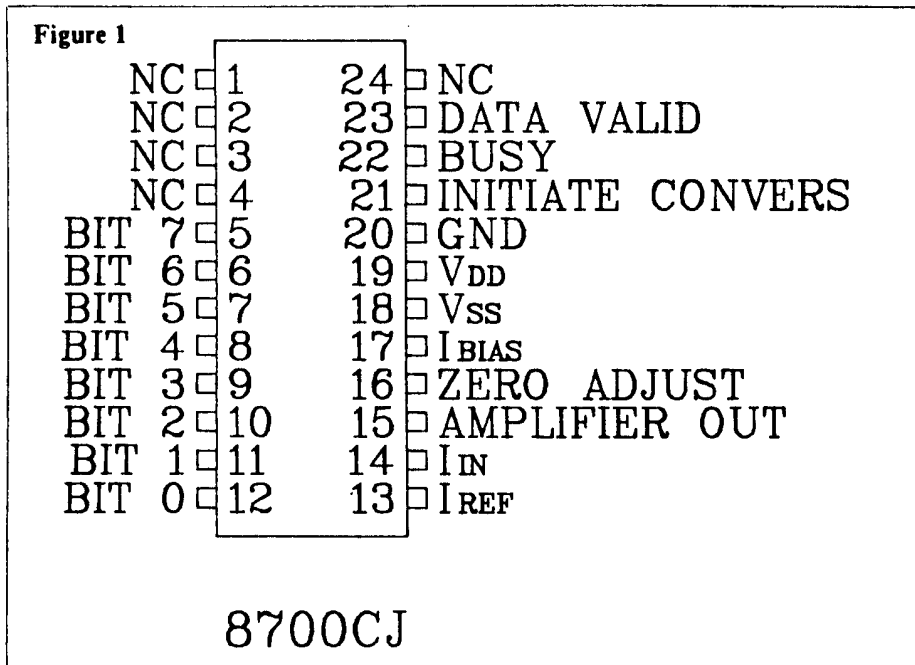
Where R_{in} is the resistor value in question, V_{in} is the *full-scale* voltage wanted and 10uA (micro amps) is the current required for full scale. This current can be changed but will remain constant for now.

For example: You want a 10-volt full scale. Using the above formula, 10 volts divided by 10 micro amps equals 1,000,000 or one megaohm. I put R5 at 100K ohms to give a 1-volt full-scale reading, R6 is one megaohm for a 10-volt reading and R7 at 10 megaohm to give a 100-volt full-scale reading. These should be metal film resistors with a one percent tolerance or better to ensure high accuracy. I used these because of their thermal stability and low noise generation.

These three scales can be whatever you choose. Just follow the formula and you won't go wrong. You can even add more resistors for more scales, but of course you will have to change the switch S1. If you need only one scale, only one resistor is needed and you won't need the switch either.

The construction is not too complicated, but care has to be taken because the 8700CJ converter is a CMOS chip and it is very sensitive to static electricity. Use proper anti-static handling procedures. Do not insert the chip into its socket until everything is finished, checked and cleaned. It is important to clean the board properly. Leftover solder flux on the board can affect the accuracy of the circuit. It may even cause it to fail altogether.

Use the Parts List and the diagram in Figure 2 to build the circuit. After the



CPU (*the CoCo uses an MC6809) — Central Processing Unit. Basically the same as MPU described earlier.

CRT — Cathode-Ray Tube.

CRTC — Cathode-Ray Tube Controller. A chip used when an 80 by 24 character display is needed. Sometimes referred to as a CRT.

CTM (*the CoCo uses an MC1372) — Color Television Modulator. It takes the signals from the VDG and converts them into a signal that is suitable for a color TV.

D/A — Digital to Analog. A chip that converts a digital value to an analog voltage.

DMA — Direct Memory Access. A process of moving data from one device or memory area to another device or memory area without the use of the CPU.

EIA — Electronic Industries Association. An agency that sets standards.

FDC (*the older Radio Shack disk controller uses the WD 1793 by Western Digital and the newer Radio Shack uses the WD 1773 by the same company) — Floppy Disk Controller. This is the main chip used when a computer talks to a floppy disk.

EEPROM — Electrically Erasable, Programmable, Read-Only Memory. It is the same as an EPROM except that electricity rather than ultraviolet light is used to erase it.

EPROM — Erasable, Programmable, Read-Only Memory. More permanent than RAM but less than ROM.

IC — Integrated Circuits. It means all chip-like components.

IIA — (*the CoCo 'F' board and the CoCo 2 use an MC6822) — Industrial Interface Adapter. It is much like a PIA but has slightly different input capabilities. Used in conjunction with the newer keyboards.

LCD — Liquid-Crystal Display. Usually seen on a digital watch.

LED — Light Emitting Diode. The indicator on almost any disk drive.

LSI — Large Scale Integration. Many transistors in one package.

MMU — Memory Management Unit. Something that is lacking in our CoCo, this chip lets a CPU handle more memory than it could without it.

MSI — Medium-Scale Integration. Smaller than the LSI.

MSO — Montreal Symphony Orchestra. Has nothing to do with computers, but it is something we are proud of in Canada.

OP-AMP (the CoCo uses the old standards LM741 and LM339) — OPera-

tional Amplifier. Used in audio circuits to amplify a given signal.

PIA (*there are two of these in the CoCo, they are both MC6821s) — Peripheral Interface Adapter. Lets the computer "talk" to things like a keyboard or joysticks.

PIC — Priority Interrupt Controller. This chip is useful when a computer has many levels of interrupt.

PLL — Phase Locked Loop. This is a device that compares the phase of one signal with another.

PROM — Programmable, Read-Only Memory. Like an EPROM but not erasable.

RAM — Random-Access Memory. Usually pertains to any kind of memory but mostly refers to static memory, as opposed to dynamic memory.

ROM (*the regular CoCo has one of these. If you have Extended BASIC, it is another one. If you have Disk Extended BASIC, there are a total of three ROMs in your CoCo. The newest CoCo 2 has BASIC and Extended BASIC ROMs bundled together) — Read-Only Memory. It is called "read-only" because the information is inserted into the chip at the factory and cannot ever be changed. This process is called a "masked" ROM.

SALT (*only the CoCo 2s have this custom chip) — Supply And Level Translator. In the CoCo this chip is responsible for main voltage supply regulation, RS-232 interface level conversion, cassette read operations and driving the cassette relay.

SASE — Self-Addressed, Stamped Envelope (see the last paragraph).

SAM (*the CoCo has one of these — an MC6883) — Synchronous Address Multiplexer. This chip takes care of the DRAM ROM and I/O in the CoCo.

SSI — Small-Scale Integration. Even smaller than MSI.

TTL (*there are a few of these in the CoCo) — Transistor-Transistor Logic. Actually TTL refers to a whole family of chips that do everything from simple buffering to AND and NOR gates to full memory refreshing. There are many levels of TTL, ranging from the regular to the 'S' (Schottky) series and the LS (Low-power Schottky) series. Today there are even more. There is the ALS (Advanced Low-power Schottky) series, the AS (Advanced Schottky) series and even the 'F' for fast series. There is even a HC (High-speed CMOS) series. All have different specifications for speed, power dissipation and price.

UART — Universal Asynchronous Receiver/Transmitter.

VDG (*the CoCo and CoCo 2 use the MC6847) — Finally, the one we have been waiting for! It stands for Video Display Generator. It is the chip that translates memory data into the visual display with which we are most familiar.

VLSI — Very Large Scale Integration. Refers to chips that have thousands and thousands of transistors, something like the Motorola MC68000, a 16-bit CPU that is used in the . . . sorry I just can't say it, red-fruit like computer.

There are also a lot more chips and components that go into making up the CoCo, but the rest do not have fancy abbreviated names. The following is a list of the active components in the CoCo that are not mentioned in the list of acronyms.

The MC14050B is a latch used in the D/A circuit.

The UM1285-8 is a Modulator. It takes the signal provided by the CTM and converts it into a signal that can be used by a regular home TV.

The MC14529 is a data separator used to select the analog inputs for the A/D circuit. These are the joysticks, sound, cassette and exterior sound. In fact, all of the analog-type signals that are in the CoCo go through this chip.

The CoCo also has four voltage regulators. The regulated voltages are +12 volts, -12 volts, +5 volts and -5 volts. The CoCo 2, on the other hand, has only two regulated voltages: +5 volts for all the circuits and an internal (to the SALT chip) -5 volts.

Through the years the CoCo has evolved from the first board (which I believe to be the 'B' board) to the latest CoCo 2. As a point of interest the next list is the amount of components it takes to build a CoCo 'F' board and a CoCo 2 'A' suffix board:

	CoCo 'F'	CoCo 2 'A'
Capacitors	85	49
Connectors	7	6
Crystals	1	1
Diodes	17	13
ICs	29	16
Fuses	1	1
Inductors	10	7
Relays	1	1
Resistors	83	34
Switches	3	3
Transformers	1	1
Transistors	4	2
Misc. components	89	43

Though I do not have the exact numbers, the CoCo 'B' board has even more parts than the CoCo 'F' board and the CoCo 2 'B' board is supposed to have even less parts than the CoCo 2 'A' board. How is that for progress? And don't forget the price difference, too.

Next time someone talks to you

about his VDGs and DMAs, you will be able to understand what he is talking about, and tell him what we have in our own CoCo.

As always, if you have a problem with something in this column and absolutely can't wait for the mail, give me a call on Monday nights *only*, at (514)

473-4910. My address is 4680 18th Street, Laval Ouest, Quebec H7R 2P9. If you write to me and expect an answer, include an SASE; you won't get an answer without one. I am sure you know what SASE means, right? □

Understanding how a computer works

A Beginner's Hardware Course

Part 1

This being the Beginners issue, I will start a multi-part article on how a computer works, starting from "simple theory" to "how to build one of my projects." This month, we will begin with basic concepts: what is a bit, what does digital mean, what is analog, how does it differ from digital, and a look at a different numbering system.

The dictionary meaning of analog is "proportionate." When speaking, you can speak loud or low. Light can be dark or bright, or any shade in-between. Radio waves and TV pictures are all said to be analog signals. These are examples of analog wave shapes — continuously changing. When we talk about a digital system, there are no shades or continuous motion. There are only two states in a digital signal: ON or OFF. There is no in-between. This is the core of computing. Everything your computer does is accomplished using these two states. OK, let's expand on these states.

First, there is ON. It is also known as "high" (Hi or 'H'), "plus," "one" (or '1'), "mark," "voltage" and many others. The two terms I use most often are Hi and '1'; these are the terms I will use throughout these articles. In most microcomputers, the operating voltage for the hardware is five volts. Virtually all the microcomputer and support chips work with five volts. It is pretty much a norm. Given this, a Hi measures about five volts on a voltage meter, but

4.5 volts is also considered Hi. There are limits to how low the voltage can be before it is considered invalid. In fact, any voltage greater than two volts is considered to be a logic level Hi or '1'.

Next is the OFF state. It, too, has many names: "low" (Lo or 'L'), "minus," "zero" (or '0'), "space" and "ground," just to name a few. To keep consistent, I will use Lo and '0' to mean OFF. A low state is considered to have zero volts and when measured with a voltage meter, nothing registers. Under certain conditions, a small voltage can be present. Any voltage below .8 volts is considered to be a logic level '0'. Any voltage greater than .8 volts or less than two volts is not a valid logic state and results are, at least, unpredictable.

Now we know about the highs and lows of digital operation. The next step is a "bit." A bit is one piece of logic information. It has, as we now know, two states, either Lo or Hi. It's also known as a binary digit, binary meaning two. The two states are:

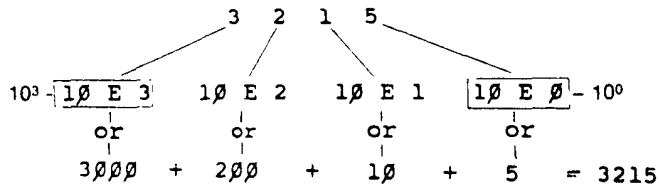
State 0 = 0 (Low)
State 1 = 1 (High)

But, just two pieces of information is not very much to work with. If we use two bits side by side, and considered every combination of 0's and 1's, there are four separate combinations.

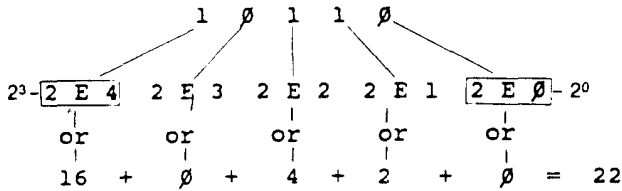
State 0 = 00
State 1 = 01
State 2 = 10
State 3 = 11

(Tony DiStefano is well-known as an early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.)

I am sure you recognize these numbers. Once the top of the number ladder is reached, you add another digit to the left of it. Each number added raises the value of that digit in the number by a factor of 10.



When large numbers are to be represented, there are more digits. Each new digit added means adding another power of 10. Numbers ranging in the millions require only seven digits in Base 10 numbers, but require many digits in Base 2 since every added digit is only to the power of 2.



You can see that a Base 2 number adds up to a lot less than Base 10. There is yet a better-suited numbering system for computers, but first let's look at a bit more (ha, ha).

The Color Computer (all versions) has an eight-bit CPU. That means all data, program code and characters are stored in eight-bit values. These groups are better known as bytes. A byte can hold any value from 00000000 (Base 2) to 11111111, or in decimal, from zero to 255. If you convert 11111111 to decimal, it works out to 255. Each byte in the CoCo is one memory location. A byte can hold one ASCII character, one piece of data or one machine language code. We'll look more at memory later on.

In the computer environment there is another numbering system. It is most used and is called the hexadecimal numbering system, or Hex for short. The Hex system, as the name implies, is a Base 16 number. This means there must be 16 symbols before the carry over to the next digit. In Hex, the symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. Just as the next digit after '3' is '4' (3+1), the next digit after '9' (9+1) is 'A'. Remember that A, B, C, D, E and F are digits, not letters, in the hexadecimal system. The following table exemplifies the different numbering systems described.

Decimal	Hex	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

If you have three bits side by side, there are eight different combinations.

- State 0 = 000
- State 1 = 001
- State 2 = 010
- State 3 = 011
- State 4 = 100
- State 5 = 101
- State 6 = 110
- State 7 = 111

Can you see a pattern start to develop? Every time one more bit is added, you double the amount of different combinations possible. This is known as Base 2 or binary numbering system. Most of us are more familiar with Base 10 or decimal numbering system. In short, Base 10 numbers, unlike Base 2 numbers, have 10 different combinations per digit.

- State 0 = 0
- State 1 = 1
- State 2 = 2
- State 3 = 3
- State 4 = 4
- State 5 = 5
- State 6 = 6
- State 7 = 7
- State 8 = 8
- State 9 = 9

As you can see from the table, the Hex numbering system is the most efficient because of its highest base number. The decimal system takes two characters to the one character needed by Hex; binary takes four characters. Since the CoCo has an eight-bit bus (a memory byte), you can represent a memory location with eight bits (11111111) or three decimal digits (255) or a two-digit Hex number (\$FF). From now on we will use all three numbering systems, whichever ever happens to be the best for the occasion. When using Hex, however, I will put the character '\$' in front of it. Some like to put an 'h' at the rear of the number — both are correct, I just prefer the dollar sign.

Understanding the Hex and binary numbering systems and what they stand for in a computer is the base from where your knowledge of the CoCo will grow. I will not cover adding and subtracting or conversion from one base to another in this article, but if you want to learn more on numbering systems, your local library should have numbering systems in the math section.

One of the command functions built into Extended BASIC is HEX\$, pronounced "Hex string." This command transfers a normal decimal value into a string variable in hexadecimal format. The syntax for this command is HEX\$(X) where 'X' can be a direct value or any numeric expression. As an example, to get the Hex equivalent of the decimal value 207, type PRINT HEX\$(207) and ENTER. This prints CF and is the Hex equivalent of 207. A very handy command to have.

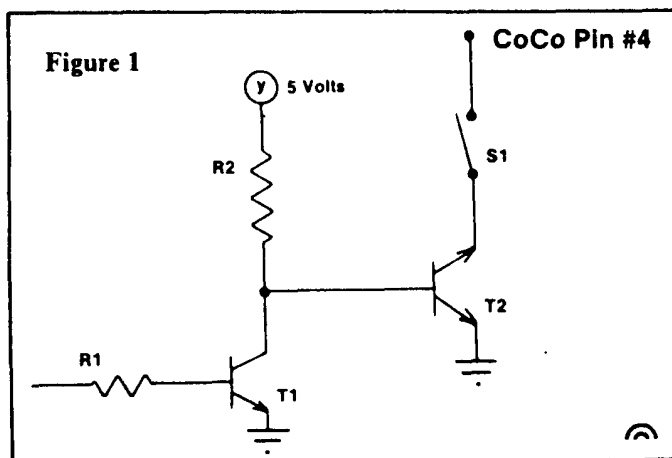
On the other hand, how would we change a Hex value into a decimal value? Extended BASIC comes to the rescue again, for it has another function that allows entry of Hex values, the &H sign. Anytime you need to enter a value in Hex, use the &H in front of the value. For instance, if you have a line that sets the value of 'X' to the Hex value FF, you can calculate \$FF to a decimal value or you can enter

it as $100 \times = \&HFF$. Another use of the function $\&H$ is to convert a Hex number to decimal. Since all numbers printed are done in decimal, to convert a Hex number to decimal all you have to do is `PRINT &HX` and `ENTER`, where 'X' is any Hex number and the result is printed in decimal on the screen. If you are to substitute the letter 'O' instead of 'H', all values will be in octal, or Base 8.

* * *

I got a letter from a reader just this week. He pointed out a problem with "Turn of the Screw" in the November 1984 issue. There is mention of a switch in the text, but no such switch existed in the diagram. Figure 1 shows where this switch goes.

Next month, we'll look into digital logic gates, truth tables and their use in computers.



A Beginner's Hardware Course

Part 2

Last month we took a look at binary bits and different numbering systems. So far, there doesn't seem to be any relation between these and computers. All we did is express numbers in different forms. But, we are a little closer to computers than you think. We know the computer is made up of a lot of chips that use bits of zeros and ones. In order to understand the ins and outs of these chips, I will go into detail of how chips use zeros and ones.

The heart of all digital computers is the logic circuit elements. They perform binary arithmetic operations, make logical decisions and perform operations such as counting and temporary storage. The basic type of logic element is called a "gate." Gates are circuits that look at two or more binary signals and produce a binary output, which depends upon the conditions of the input signals.

In order to comprehend this better, let's look at an equivalent circuit that is easier to understand, using conven-

(Tony DiStefano is well-known as an early specialist in computer hardware projects. He lives in Laval Ovest, Quebec.)

tional components you are likely to find around the house. If you want to build and test these circuits yourself, Radio Shack has all the parts. The switches are single pole, single throw. Any battery and bulb combination will do, just be sure the battery and the light bulb are the same voltage rating, otherwise you may end up burning out the bulb or get no results at all. Such a circuit is shown in Figure 2.

This circuit contains three components: a battery, a switch and a bulb. Here, the switch is considered the input and the bulb is considered the output. When the switch is on (a logical 1) the bulb is lit (this is also considered a logical 1). When the switch is off (logical 0) the bulb is off, also giving us a logical 0. In a logical element such as this, the input (the switch) and the output (the bulb) follow each other, one to one or zero to zero.

The symbol used to represent this circuit or logical element in a logic (or computer) schematic is shown in Figure 1a. This gate is called a "buffer." The input is exactly the same as the output. Not very useful in a logical sense, in that it does nothing, but it is needed under certain circumstances. For instance, when the output of a gate (logic ele-

ment) is connected to many other gates, it may not have enough power to drive all the gates properly. In this case a buffer is used. Whenever a gate is used there is always a small delay between when the input changes and the output changes; a buffer is sometimes used just for that delay.

To continue our understanding of gates, let's introduce another factor in our battery circuit. Now study the circuit in Figure 3. It has two switches. The two switches are in a series, that is, one after the other. Therefore, they must both be on before the bulb will turn on. This circuit or logical element is known as an AND gate. The definition of an AND gate is: "The AND gate is a logical element with two or more inputs and a single output. Both (or all in the case of more than two) inputs must be binary '1' to produce an output of binary '1'."

The symbol for an AND gate is shown in Figure 1b. The main value of the AND gate is its ability to detect when all inputs are binary '1'. For example, in a control system when all the motors are on, turn on the extra generator. A quick way to remember this gate is, when 'A' AND 'B' are '1', then 'Y' is '1'. Hence the term AND.

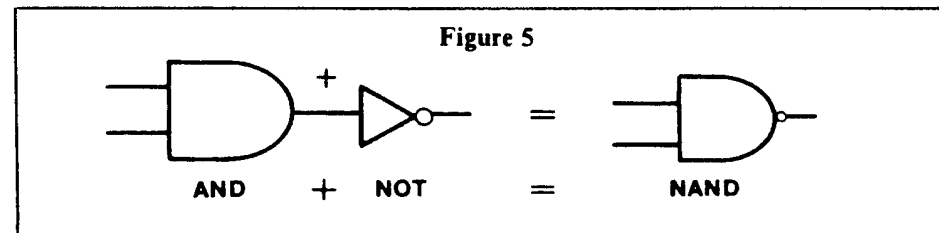
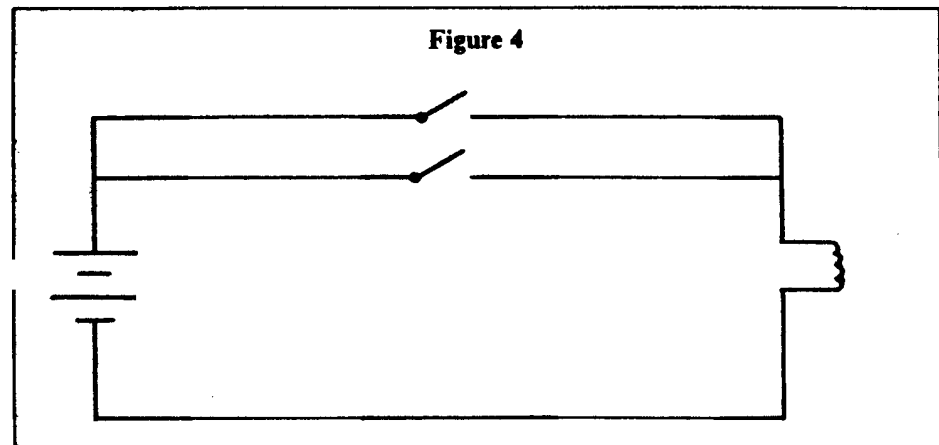
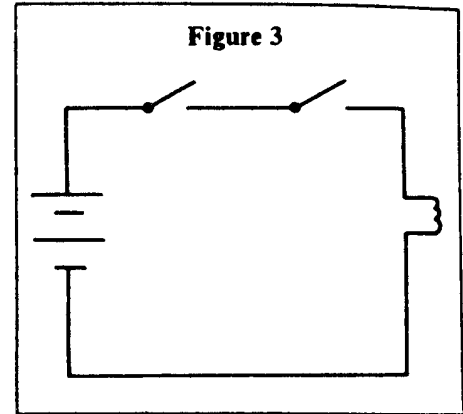
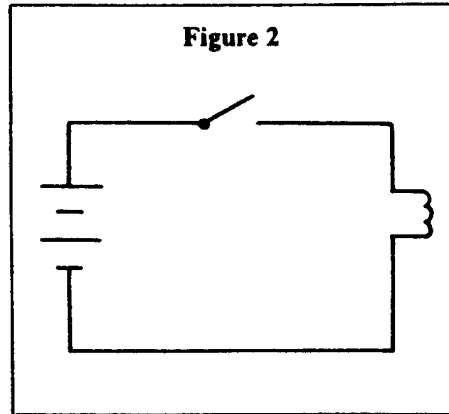
two inputs is a binary '1'. The other input being a binary '0'."

A quick way to remember the function of this gate is when the inputs are different, the output is '1'. Like the other gates, it, too, has the inverted version. It is called the EXCLUSIVE-NOR or XNOR for short. The definition of an XNOR gate is the same as the XOR, but has its output inverted to a binary '0' when either of the two inputs is a binary '1'.

The gates described so far are quite simple in structure. They have one or two inputs and one output. They are the fundamental elements in creating more complex chips, and even the basis of complete computer CPUs. In the case of the simple two-input AND gate, there are four discrete combinations of inputs. The two inputs are represented by a two-digit binary number. Remember last month? They are 00, 01, 10 and 11, and the output for each given condition is 0, 0, 0 and 1, respectively. Not so hard to remember or display. But, in other chips, where there might be five or six inputs and eight or 10 outputs, it can be too much to remember. Now is when the "truth table comes in. The definition of a truth table is: "A truth table is a graphic representation of all possible combinations of inputs versus outputs of a particular logic element."

The second column of Figure 1 represents the truth tables for the given gates. Notice that all possible combinations of inputs are given. Columns A and B are the inputs, as you can see from the gates in Column 1. Column Y is the output. Read the truth table as you read text, one line at a time. Each line is one condition. The condition is given for 'A' and 'B'. The output, 'Y', is the result for a given gate. Every line is different, and continues until all possible combinations for that gate are shown. This way, at a glance, you can tell what the output is for a given input of any gate. In these cases, it is not too difficult to follow or remember. Later on, when I show you the truth tables for some of the chips that make up our good ol' CoCo, you will be glad I introduced you to these tables.

Though I will not be getting into great detail in this series of articles, I feel it is necessary to talk a little about Boolean algebra. The definition of Boolean algebra is: "A system of mathematical logic used to represent digital logic signals and express the logic operations



performed by digital signals."

To put it into simple terms, Boolean algebra is an equation that represents the function of a logical element. Take, for instance, the buffer in Figure 1. The output is equal to the input. A Boolean equation would be:

$$Y = A$$

Now an inverter would look like this:

$$Y = \text{NOT } A \text{ or } Y = *A$$

The AND symbol in a Boolean expression is a dot in the middle of the line, like the multiplication sign in regular math. Notice its occurrences in Figure 1. The OR symbol in a Boolean expression is a plus sign (+). Again, the Boolean OR symbol can be seen in Figure 1. The next Boolean symbol is the EXCLUSIVE-OR. This is no more than the plus symbol with a circle

around it. Figure 1 also shows the XOR symbol. Any of the inverting symbols in Boolean algebra are represented by a small horizontal bar above the equation in question. You can see the inverting gates in Figure 1.

That is it for this month. If you are going to the Palo Alto RAINBOWfest, Feb. 14-16, come and see me at the DISTO booth.

References

- 1) *Contemporary Electronics*, McGraw-Hill Continuing Education Center.
- 2) *Digital Computer Logic and Electronics*, The Algorithms Press.
- 3) *Model 100 Service Manual*, Radio Shack, Tandy Corporation.
- 4) *The TTL Data Book*, Texas Instruments, Incorporated.
- 5) *Microcomputer Interfacing*, Prentice-Hall, Inc.

The next gate we will study is the OR gate. Again, we have two switches in our next diagram, Figure 3. The difference is that now they are wired in parallel, one on top of the other. If either switch is on, then the bulb will be on. If both are on, the light is, of course, still on. This circuit or gate is known as an OR gate. The definition of an OR gate is as follows: "The OR gate is a logical element with two or more inputs and a single output. If any one input is a binary '1' then the output is binary '1'."

The symbol for an OR gate is shown in Figure 1c. The main value of the OR gate is its ability to detect when any input is binary '1'. An example of this use is when any door or window opens, an alarm sounds. A quick way to remember this gate is when 'A' OR 'B' is '1', then 'Y' is '1'. Hence the term OR.

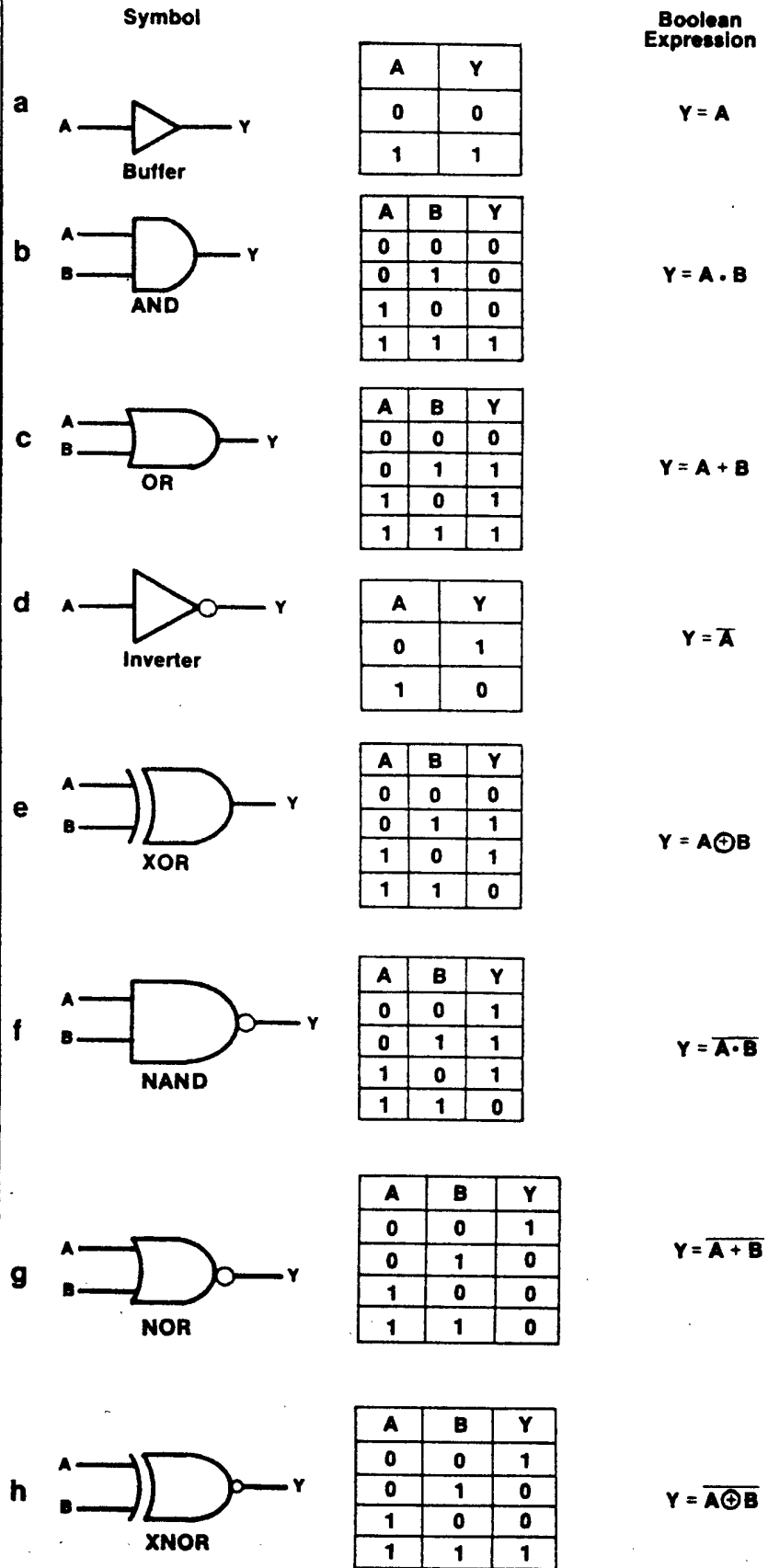
If we look back to our first gate, the buffer, we notice the input matches the output. Since the input and the output are the same, it is called a "non-inverting" output. This gate, and most other gates, can also come in an "inverting" output. In the case of our buffer, it becomes an inverter, or better known as a NOT gate. Figure 1d shows the symbol of an inverter. The definition of an inverter is: "An inverter is a logic element whose output is always the complement (the opposite) of its input."

Notice the difference between a buffer symbol and an inverter symbol. The inverter symbol has a small circle on the output side. Any inverting output gate has a small circle on the output. This is true for the AND and the OR gate, too. If you take the output of an AND gate and tie it to the input of a NOT gate, the result (the output of the NOT gate) is an inverted AND gate (see Figure 5). This requires two gates and some wiring. It is so often used that the IC designers decided to put it all in one chip. This is called a NAND gate. The same thing goes with an OR gate — it becomes a NOR gate. These two gates are defined as follows: "NAND and NOR gates are the complements of AND and OR gates, respectively."

The last gate we will look at is the EXCLUSIVE-OR gate. The symbol for the EXCLUSIVE-OR gate is shown in Figure 1e. For short, this gate is called XOR. It is a little different than the OR gate and is used mostly when a signal needs to be inverted in some cases and not in others. The definition of an XOR gate is: "The logical XOR is defined as a binary '1' output when either of the

Figure 1

Truth Table



An Introduction to Timing

Continuing our journey into the CoCo, this month I will look into the heart of this and any computer — timing. All the hardware of the computer is controlled by timing. The most important part of the timing is to keep the CPU in step. What is a CPU, anyway? Well, the letters CPU stand for Central Processing Unit. The CPU inside the CoCo is the MC6809. The CPU, in a way, does all the work. It can move data from one part of memory to another, compare two values and act according to the result, add and subtract values and so forth. In fact, without the CPU, the rest of the hardware that makes up a computer would be worthless. The CPU is a very complex chip. It has data lines, address lines, interrupt lines, status lines and more. The timing that goes with the CPU is also important. OK, let's get into it. It is a prerequisite to understanding how a CPU works.

Up till now, when I talked about zeros and ones and the change from one

to the other, it was considered to be instantaneous. There was no mention of how long it took to change from one state to another. In fact, we are dealing with real life, not just theory. Situations in theory rarely work in real life the way

"The first fact of the real world is propagational delay."

you want or expect them to. Welcome to the real world of delays. Ever caught an on-time airline flight? Ha!

The first fact of the real world is propagational delay. Take, for instance, a simple inverter. Figure 1a shows an inverter. When there is a '1' at the input there is a '0' at the output. A '0' input will give a '1' output. But when the input changes from one state to another, there is a short delay before the output

changes. This delay is called the propagational delay, which means the amount of time it takes an electrical signal to go through a logic element or wire.

Figure 1b shows a graph of the input and the output of an inverter. The X-axis (from left to right) shows the passing of time. This can be in seconds, hundredths of seconds, thousandths of seconds and even millionths of seconds. When no time base is given, then time factor is not relevant. Typical delay times for the TTL family (more on chip families in later articles) is from five to 30 ns (ns = nanoseconds). The Y-axis usually shows the binary level of '0' and '1'. When two or more signals are shown that are related to each other, they are shown on top of each other with the left-to-right passing of time common to each.

Getting back to Figure 1b, we see the passing of time and the relation of the input to the output. There is no delay shown in this diagram. To show the delays of each signal for a given complex gate would confuse the diagram. Instead, an overall delay is given for the gate. But, in order to get used to the idea of delays, Figure 1c shows the time delays of a typical inverter. Along with

Figure 1a

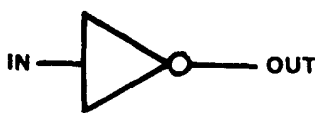


Figure 1b

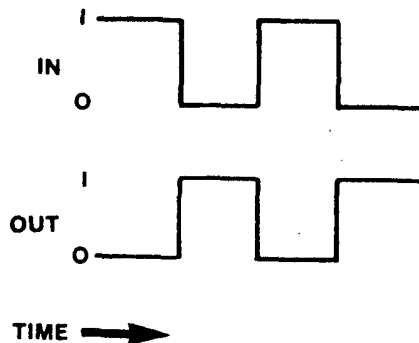


Figure 1c

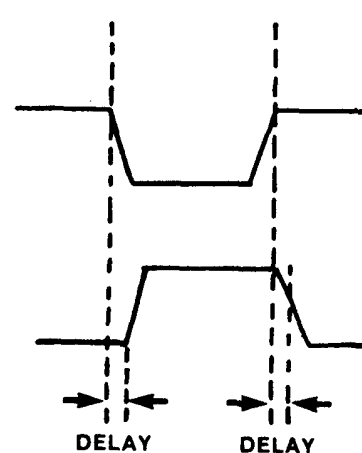


Figure 2a

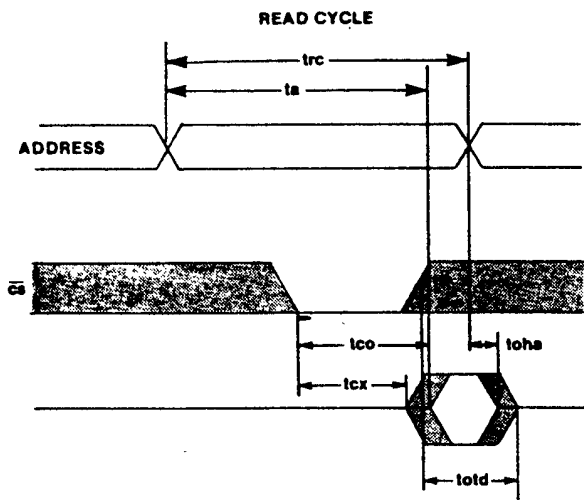
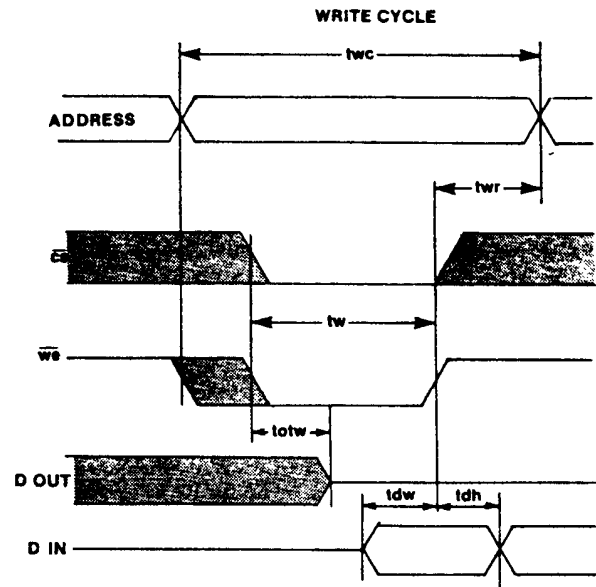


Figure 2b



the delay of the signal there is also the rise and fall time. The rise time of a signal is described as the time it takes for a given signal to reach 90 percent of maximum voltage from the 10 percent voltage level. The fall time of a signal is described as the time it takes for a given signal to drop to 10 percent voltage from the 90 percent voltage level. In the case of the CoCo, the voltage considered a logical level '1' (or HI) is five volts. The logical level '0' (or LO) is, of course, zero volts. The actual working voltages may be slightly different.

Delay, rise and fall times are important mainly to the designer of the system. When an engineer designs a computer he must know these timings and make sure that all operations are within the given limits. For example, two signals go to one gate, but one goes through several gates first. Each time the signal travels from one gate to another there is more delay. If the signal is delayed enough, an improper signal output results.

It sounds like I'm making a big deal of delays. While it is important, it is not a major concern to computer hackers (or should I use the term hobbyist?) and even less to end users. More important to us is another kind of delay. It is known as "access time," which means the mean time between the request for memory and the actual valid data.

Let us look at a typical memory chip. There are thousands of gates and transistors inside this chip. All of these gates

inside the chip cause a significant delay between the time when the address to the chip is valid and the time when the data output appears on the data bus. This is known as access time. When talking about memory, an important parameter is access time. These access times can range from super-fast static memory at about 10 ns to very slow dynamic memory at 450 ns and slower. It is this limitation that controls and

"More important to us is another kind of delay. It is known as 'access time,' which means the mean time between the request for memory and the actual valid data."

limits the speed of CPUs. Figure 2a shows the read cycle timing diagram of a memory chip. Figure 2b shows the write cycle for the same chip. What follows is a description of what each line on the diagram means.

Address — These are the address lines that select what byte is to be accessed. It is shown with two lines, one high and one low. It is shown this way because there are usually several lines and since the timing is the same no matter what

byte you access, it is not relevant which address line is high or which line is low. The two lines (one on top and one on the bottom) represent any given address within the chip. Where the lines criss-cross means a change of address. That is when the CPU is finished with that byte and requests another by putting another address on the bus. Access times are always measured with respect to the address change from the CPU. Actually, it starts when the address is stable, better known as a "valid address."

Chip Select (CS) — Remember the *CS line on memory chips in past articles? It is used to select or activate the chip. From the diagram of the read cycle, we can now see the relation between when the address is valid, the *CS line and when the data is valid.

Data out — This, of course, is the data that the CPU requested. Notice the data valid area. That is the time when the data that appears on the bus is the data that is held in that memory location. Notice the top and bottom dual line display. It has the same description as address lines, some are ones and some are zeros. The line in front of the data valid section is halfway between zero and one. That means the data lines are tri-state and no valid data is input or output. The shaded area on both sides of the data valid window is the transition time between tri-state and data valid. In this area, data lines are changing to their proper values. A read in this area will not yield valid data.

Read/Write — The *R/W line is used to select a read cycle or a write cycle. Straightforward, no problems there. In the CoCo this line is logical '1' to read and '0' to write.

The following is a description of all the relevant parameters used in Figures 2a and 2b.

t(rc) — Read Cycle Time: the time it takes for a complete read cycle given in ns.

t(a) — Access Time: the delay between a valid address and data valid.

t(co) — Chip Select to Output Valid: the delay between when the *CS is active and the data is valid. This is only true with a valid and stable address.

t(cx) — Chip Select to Output Active: same as t(co) but not to data valid; to when the data lines start changing from tri-state to output. Usually of minor importance.

t(otd) — Output Tri-state from Deselection: the time that the data stays valid after the *CE goes inactive or deselected.

t(oha) — Output Hold from Address Change: the time that the data stays valid after an address change is detected.

t(wc) — Write Cycle Time: same as the t(rc) except for a write cycle.

t(w) — Write Time: the minimum time the write line has to remain low.

t(wr) — Write Release Time: time between the *WE line deselected and a change of address.

t(otw) — Output Tri-state from Write: the time it takes the data lines to go to tri-state from a write request.

t(dw) — Data to Write Time Overlap: the time data must be stable before the *WE line deselected.

t(dh) — Data Hold from Write Time: the time data must be stable after the *WE line deselected.

Figures 2a and 2b show the read and write cycle parameters for a typical memory chip. Though these are not the memory chips inside the CoCo, the timing and parameters are quite similar.

Now with no further *delays*, it is time to look into the CPU . . . well, sort of! There is one more thing we must look into; it is CPU related, though. We are getting closer. It is the master clock, which is a master reference wave form used to synchronize all of the logic in a system.

The master clock is usually the highest frequency in the computer. All other timings are derived (divided) from this clock. The CPU clock is the speed or frequency at which all instructions and data are retrieved and stored to memory. Depending on the system design, the CPU clock can be equal to the master clock, or any division thereof. In the case of the CoCo, the master clock frequency is 14.31818 MHz (mega-hertz or million hertz) and the CPU clock frequency is 1/16 that of the master clock at 0.8948 MHz. Well, there are two clock speeds in the CoCo. Under special conditions, the CPU can work at 1.8 MHz.

Now you might say, "Wow, my CoCo has a clock rate of only .894 MHz!" Compared to that of the 4 MHz of other computers, that may or may not be slower. You see, it gets more complicated. The CPU clock does not always mean the net speed of the computer. There are some other factors involved, such as synchronous I/O, as opposed to asynchronous I/O.

Let's look at synchronous I/O first. As the word implies, synchronous I/O means that any memory, read or write, is synchronized. Synchronized to what? The CPU clock, of course. On any given clock cycle, the CPU can do one I/O. You know exactly when the CPU will need the bus. It corresponds to the clock cycle. In an asynchronous situation, the CPU requires more than one clock cycle to do a read or write. Asynchronous I/O requires either three or four cycles depending on what kind of I/O it is doing. On this type of CPU, signals are required to tell memory or other devices that an I/O has started.

Just about now, a little bit of math is required. Given that the clock fre-

Figure 3

READ CYCLE

SYMBOL	PARAMETER	MIN	MAX	UNIT
tpc	READ CYCLE TIME	250		NS
ta	ACCESS TIME		250	NS
tco	CHIP SELECT TO OUTPUT		85	NS
tcx	CHIP SELECT TO OUTPUT	10		NS
totd	OUTPUT TRI-STATE FROM DESELECTION	15		NS
toha	OUTPUT HOLD FROM ADDRESS CHANGE		20	NS

WRITE CYCLE

SYMBOL	PARAMETER	MIN	MAX	UNIT
twc	WRITE CYCLE TIME	250		NS
tw	WRITE TIME	135		NS
twr	WRITE RELEASE TIME	0		NS
totw	OUTPUT 3-STATE FROM WRITE		60	NS
tdw	DATA TO WRITE TIME OVERLAP	135		NS
tdh	DATA HOLD FROM WRITE TIME	0		NS

quency of the CoCo is 894886 hertz or 0.894 MHz, one clock cycle is 1117 nanoseconds. The way I did this is to transfer from frequency to time period. The equation used is:

$$T = 1/F$$

where 'T' is in seconds and 'F' (frequency) is in hertz. So the frequency of

0.894 MHz is a time period of .000001117 seconds, or 1117 nanoseconds, or 1.117 microseconds. Now, when we talk about speed, we can say that the CoCo can do about one I/O per microsecond — a much more accurate way to measure the effective speed of a CPU.

I hope these articles about the hardware of the CoCo are informative to

you. Also, I hope I am not going too fast; it is hard for me to judge what audience I am writing for. If you have some comments to make, a direction to take or something you don't understand, write to me through RAINBOW and I'll try to answer the interesting and common ones here in this column. Next time, we'll look deeper into the heart of the CoCo. ☺

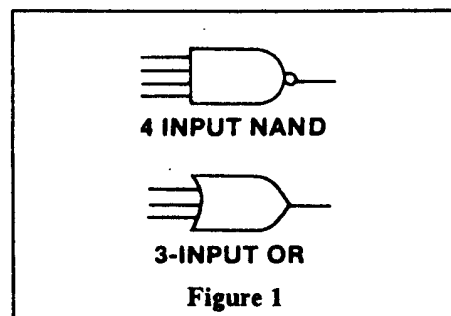
The Makings of Memory and How it Works

Remember AND, OR and XOR gates? Along with these gates came simple truth tables and Boolean expressions. All the gate examples that were given were two inputs and one output. Most of these gates also come in multiple inputs and a variety of outputs. For example, Figure 1 shows a four-input NAND and a three-input OR. If you also examine the accompanying truth tables, you will see that the rule of thumb for these gates still applies. (See my column in the February 1986 RAINBOW, Page 154, for the rules to logic gates.) Some gates are made, for instance, to have up to 13 inputs, but they are used mostly for memory mapping. I'll be going into more detail about memory mapping in a future article. No matter how many inputs you have, though, all the same rules apply.

Another property of logic gates (which have more to do with hardware than logistics) that we haven't touched on yet is the type of output. So far, all

the outputs we have talked about are either ones or zeros; a one being a positive voltage (+5 volts in the case of the CoCo) and a zero being no voltage or ground. There are two other types of outputs to consider. The *open-collector* output and the *tri-state* output.

Let us look at the tri-state output first. "Tri," meaning three, tells us there are three possible output conditions.



How can that be with a binary output? The word binary implies two conditions. What is the third state? The third state is called high-impedance. That is when the output is neither one nor zero. It is as if the output was not connected. The physical connection to the chip is still there (in the chip), but the internal connection is broken as if a switch was inserted.

Examine Figure 2a. It shows an example of how a tri-state gate works. It is not practical to show a switch every time there is an output that has tri-state capabilities. Figure 2b shows us how a tri-state output is symbolized. The extra line shown is for the tri-state output control. It is an input. Depending on the chip, this input can be active high or active low. By active, I mean that the switch (Figure 2a) is closed. Active high means the switch is closed when a one is present at the tri-state control input. Active low is when a zero is present.

This type of output is needed when there are two outputs connected together. Look at Figure 2c and try to think what logic level Point B is if Point A=0 and Point C=1. This could lead to some problems. One gate wants to be five volts and the other wants to be ground. A short circuit exists and one, if not both gates, can suffer damage. A condition like this cannot exist. It is up to the system designer to make sure there is no possibility for output conflicts such as the one in Figure 2c.

However, in a computer, there are times when two outputs must meet and go into one (or more) inputs. It is then necessary to use tri-state outputs. The main use of tri-state outputs is when

Tony DiStefano is well-known as an early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

there must exist, on a single connection, more than one output. An example of this is right on the CoCo. When you add a ROM pack or a disk drive controller to your computer, the pack and the computer share common connections, therefore both must have tri-state outputs.

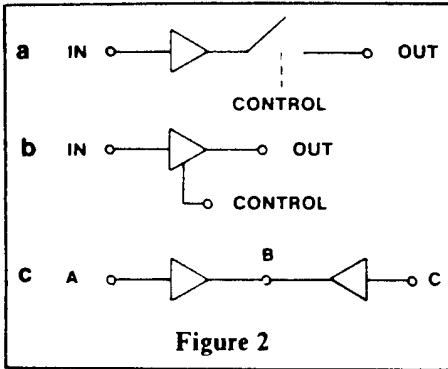


Figure 2

The second type of output is open-collector. In electronic terms, the output circuitry means that the last transistor connection to the output pin is the collector. The emitter is connected to ground and the base connects to the previous transistor. Now, to speak English. Figure 3a shows a typical open collector output. If you are not up on your electronics, and that is a bit too much to swallow, let's look at it in another way.

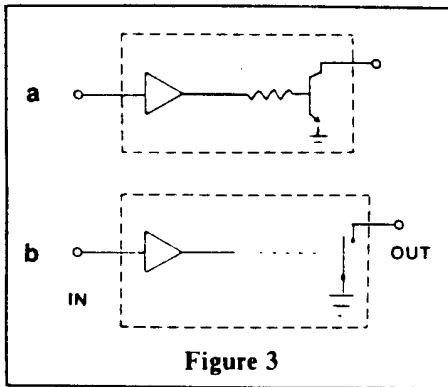


Figure 3

The output of an open collector gate can be seen as a switch with one end connected to ground. It has two states: 1) if the output is high (one), it is high impedance, not a logical one, as if it wasn't connected; 2) if the output is low (zero), it is logical zero or ground. Figure 3b shows the equivalent circuit to an open-collector output.

There is no special symbol for an open-collector gate — only the data sheet of the gate in question will tell. Usually, open-collector outputs have a resistor connected to plus voltage that gives it away. There are specific uses for this type of output. I will not go into too

much detail here, but an example of this is in the disk controller. The controller uses open-collector outputs to control the disk drives.

Now it's time to move on to new material. So far, all the gates we have looked at have a given output for given inputs. If the inputs are removed, the output is no longer valid. In a computer, there is a need to remember previous events. For example, when you use a calculator to add two numbers, the first number must be remembered or stored to be used later. The ability to remember a previous event in a computer is called, yes, you guessed it, *memory*. The simplest form of memory is one bit. A one or a zero is one bit of memory. A *flip-flop* is a logic gate with memory. The simplest form of flip-flop is called the R-S (Reset, Set) flip-flop. It is made by using two gates we have already looked at.

Examine the diagram in Figure 4a. It uses two NAND gates. A NOR gate

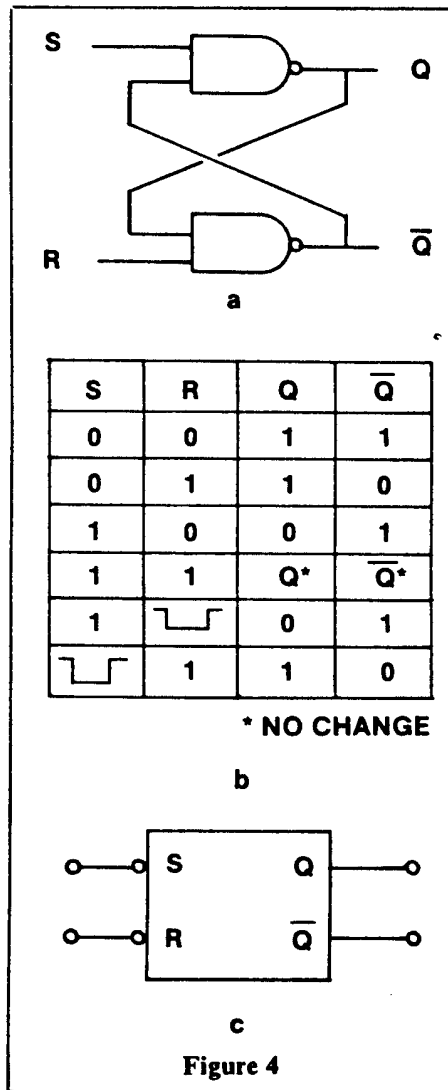


Figure 4

could also be used; the only difference is that the polarity required to activate the device is inverted. Given that the 'S' and 'R' inputs are both ones, the outputs 'Q' and " \bar{Q} " (the use of the symbol ' \bar{Q} ' simply means *not* or active low; it is usually shown using this symbol or as a small black bar above the character) would be one and the other zero. The outputs are always the complement of each other.

Due to the nature of this circuit, it is impossible to tell which output is which when power is first applied. It is an *indeterminate* state. If we were to change the 'S' input to zero and then back to one, we would have what is known as a *pulse*. A pulse is a change of logic state for a predetermined amount of time, then it returns to its original state. That means if a signal is normally one, a pulse is a negative-going pulse. If the signal is normally zero, a pulse is a positive-going pulse.

This comes right in line with what is called the active state. Let's say we have a signal that is high (one) when it is idle (doing nothing) and when we want this line to do something, it goes low (zero). This is called active low. The same is true in reverse: A signal that is normally low and pulses high to activate is called active high.

To get back to our flip-flop, the result of a low pulse on the 'S' line "flips" the outputs to a known state. The 'Q' output is one and " \bar{Q} " is zero. If we were to pulse the 'R' line, the outputs "flop" to just the opposite. If both 'R' and 'S' were to be pulsed, the output is again indeterminate. The truth table for an R-S flip-flop is shown in Figure 4b. The symbol for a NAND R-S flip-flop is shown in Figure 4c.

The next diagram, Figure 5a, is called a *clocked* R-S flip-flop. This is used when it is necessary to set up the input conditions, but delay the actual setting or resetting action until a pulse is given from another source. The CK (clock) line is used to inhibit the 'S' and 'R' lines from entering the flip-flop stage. Follow the logic using the truth table in Figure 5b. Figure 5c shows the symbol for this.

To continue our quest to understanding memory, let's go one step further. If we were to add an inverter to the 'R' side of our R/S flip-flop and tie its input to the 'S' side (Figure 6), we now have a D-type flip-flop. The D-type flip-flop is one step closer to making a memory chip. The 'D' stands for data. The logical state of 'D' is transferred to the 'Q' output on the leading edge of the

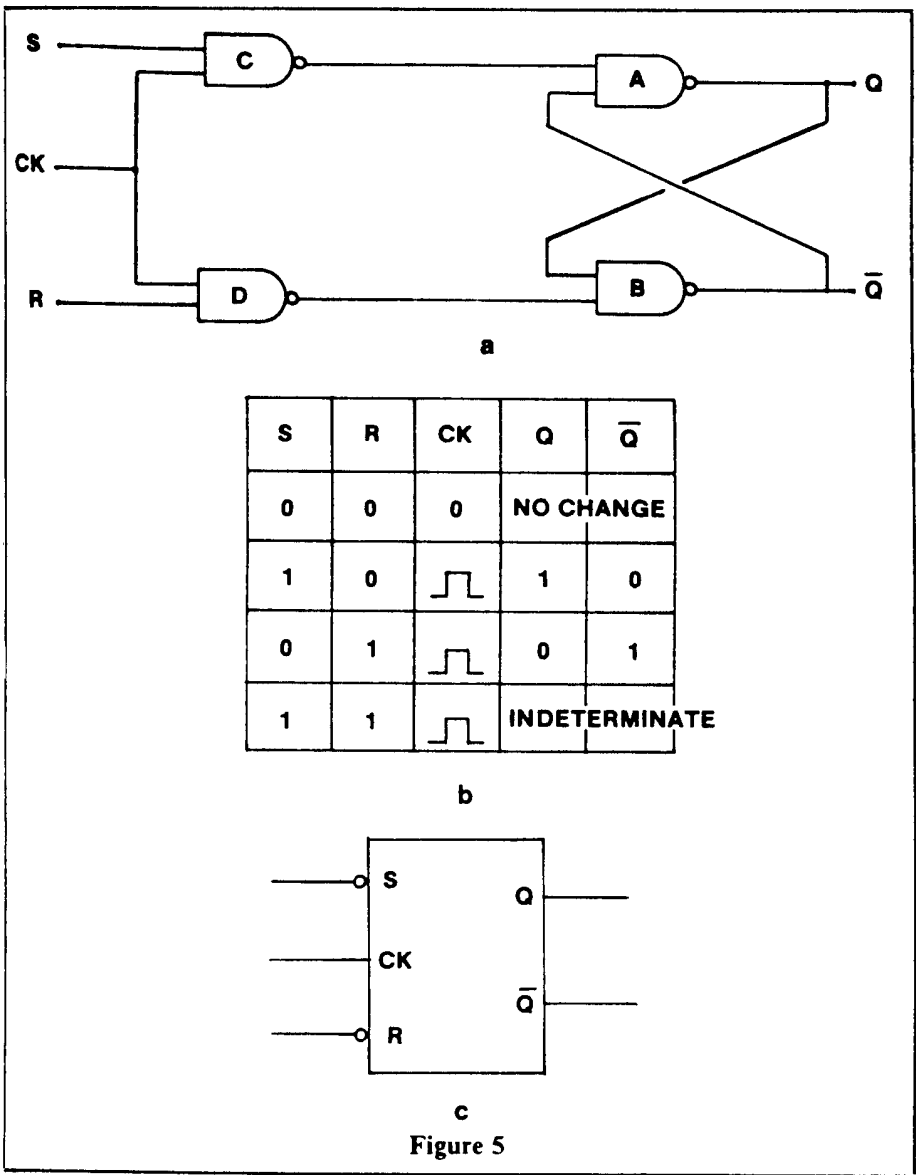


Figure 5

clock pulse. The word "edge" in this context means the precise moment the pulse changes state. This means the instant the CK input goes high, the gate ('A' or 'B') that has the one transfers to

the R/S section of the D-type flip-flop. When the CK line returns to its inactive zero state, the data is locked into the flip-flop.

You can say that this is a one-by-one

memory chip. It is a far cry from the 65,536-by-eight memory capabilities of the CPU inside the CoCo. Can you imagine how big the computer would be if it had 524,280 chips in it? We will work up to that next month. In the meantime, back to the flip-flop.

There are many limitations to the simple D-type flip-flop. The main one being that since there is a single input (apart from the clock), the 'D' input must remain stable for the duration of the clock pulse. This is to ensure that the data is accurately transmitted to the output. There are many types of flip-flops. For right now, I will go into detailed explanations of only the ones that will help us understand the makings of a memory chip.

The next diagram, Figure 7a, shows a more sophisticated flip-flop. It is labeled a "positive-edge triggered D-type flip-flop" (whew, what a mouthful). This gate is one step closer to resembling the memory chips inside today's computers. The 'S' and 'R' inputs are normally one or active low. The CK line for now should be zero. When the CK goes high, the output of Gate B goes low, causing the R-S flip-flop formed by 'E' and 'F' to be set. If, while the CK is still one, the 'D' input changes, the output of Gate D changes, although this has no effect on the output since Gate C is inhibited by the output Gate B.

When CK returns low, the output of 'B' goes back to one, but 'C' is now inhibited by the zero state of the CK. The output now reflects the 'D' input. This circuit is very similar to one bit in a RAM chip. Figure 7b shows the symbol for this gate and Figure 7c shows the truth table.

The CoCo's CPU reads and writes data eight bits (one byte) at a time. This

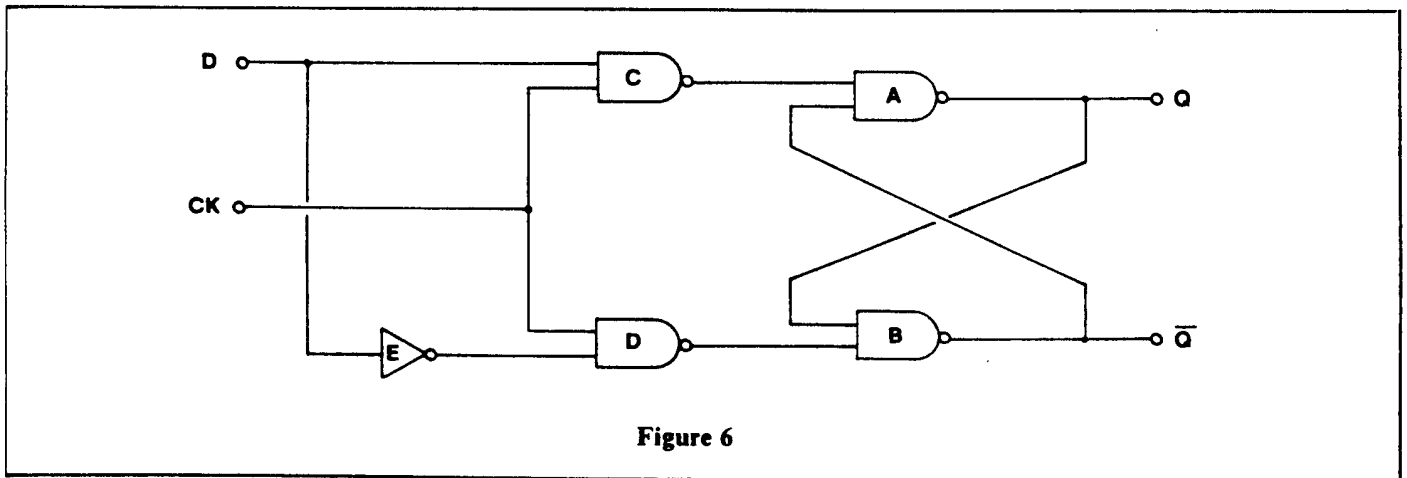


Figure 6

is not a big problem for us; all we have to do is make eight flip-flop circuits for every byte we need. There is, however, another problem we have not yet seen. This and most CPUs do not have separate input and output pins. That would make 16 pins. Instead, it has only eight pins, commonly known as the *data bus*, and one direction pin. This direction pin is known as the *read/write* line, or **R/W* for short. The **R/W* pin on this CPU is active low for writing. That means when this output is high, the CPU is reading or entering data (the action of transferring data from memory to the CPU). Likewise, when it is low, it is writing or producing data (the action of

transferring data from the CPU to memory).

With just a few more gates, the famous positive-edge triggered D-Type flip-flop will concede to the CPU's demands. Figure 8 shows one way of making this happen. Remember the tri-state output described earlier in this article? Well, it is finally put to good use. The 'R' and 'S' lines are the same as before. In most memory circuits, they are never used. The 'Q' line, however, is tied to the input of a tri-state buffer. The **Q* in this case is not used. The output of the buffer becomes the new 'D' line. This is also the input, but a new line has been added — it is the **R/W*

input. When this input line is high and the CK line is high, the action is a read. The tri-state Buffer A is activated, therefore the output of 'Q' appears at the 'D' line.

When the **W/R* line is low and the CK line is high, the action is a write. The 'Q' output is blocked by the tri-state Gate A, but Gate B allows the 'D' input to be transferred to the R/S flip-flop and, therefore, memorized into this bit. This is the basis of how memory storage works in a computer.

Next time, we'll look at how many bits of memory form bytes and how many bytes of memory form a memory map. □

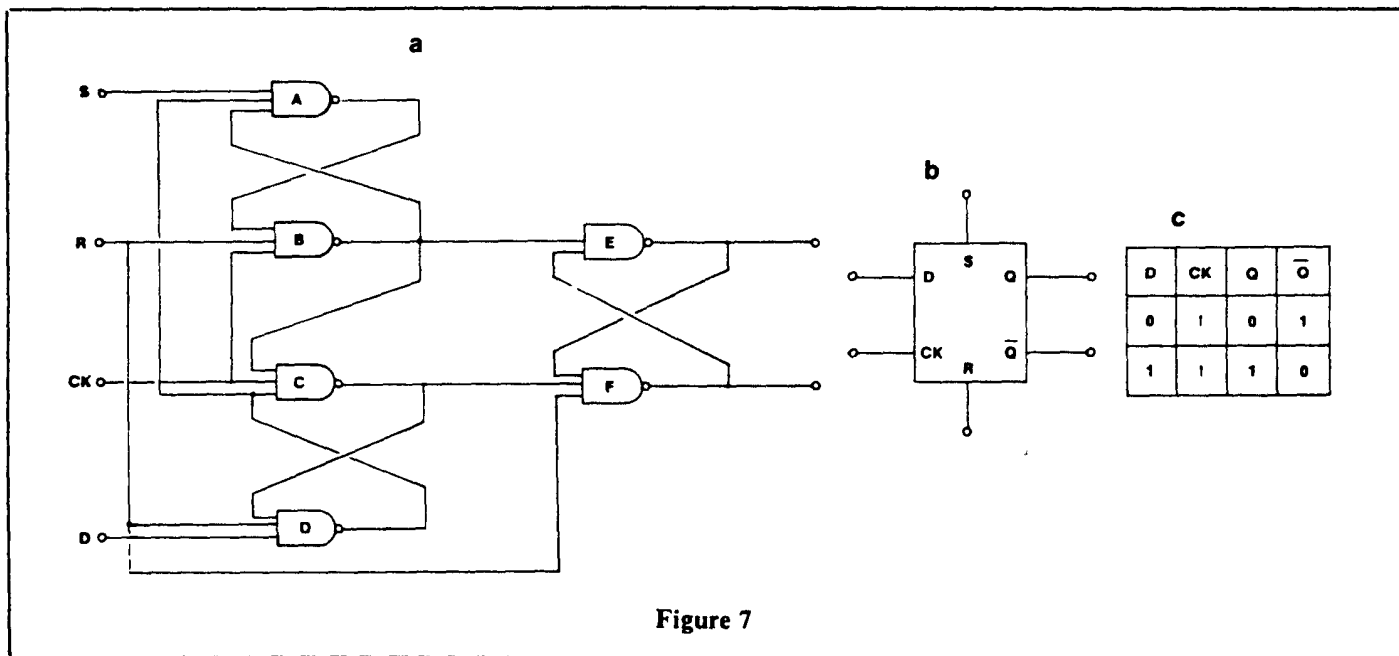


Figure 7

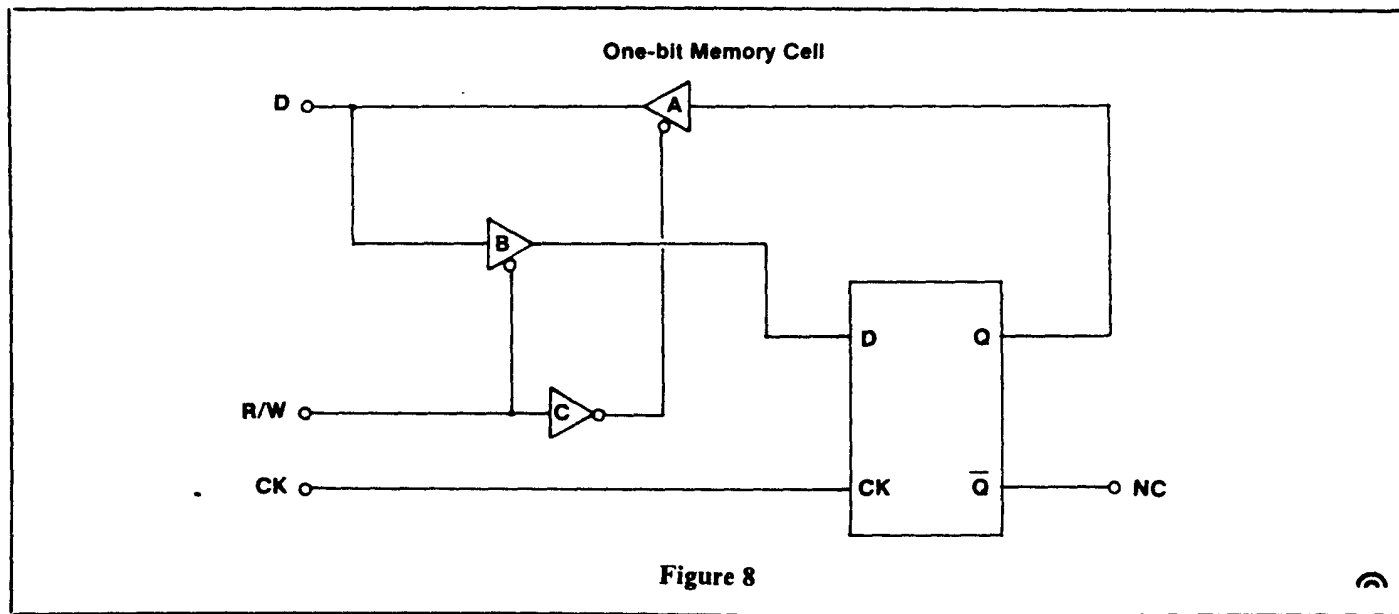


Figure 8

Exploring Memory Cells

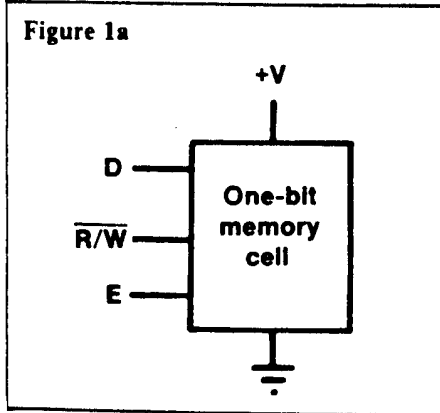
By Tony DiStefano
Rainbow Contributing Editor

Last time we looked at how a few flip-flops and gates added up to make a memory cell. A memory cell can also be part of a bigger block of memory cells. This time, I'll expand on the theory of memory cells and describe in detail the concept of memory mapping, chip select, data and address buses.

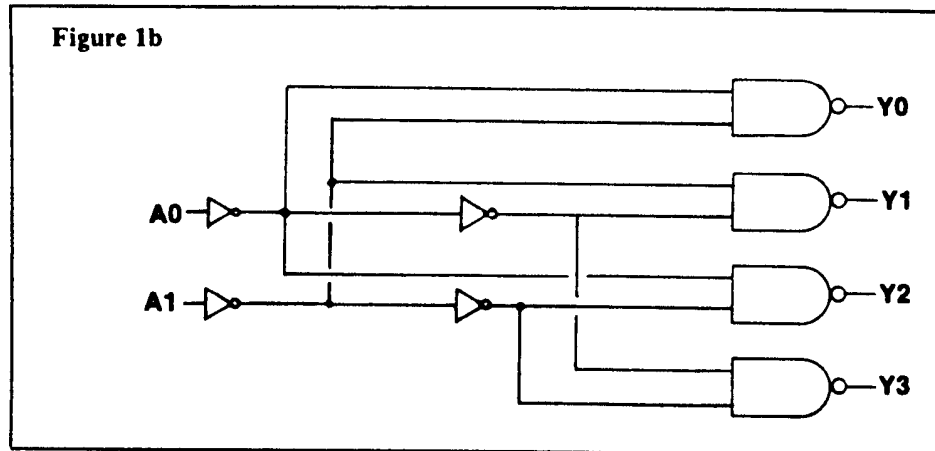
Let's start back at the one-cell memory bit. Figure 1a shows the block diagram of a one-bit by one-bit memory chip. This chip does not exist on the market; it is too simple. It would take thousands of these chips along with thousands of wires to make a decent amount of memory. Today there are

static memory chips that have 8K by 8 bits wide in one 28-pin DIP (Dual Inline Package) and dynamic memory chips pushing one megabit (that's one million bits).

are high. When A0 is low and A1 is high, Y1 is low. When A0 is high and A1 is low, Y2 is low. And finally, a 1 on both A0 and A1 produces a low on Y3. If you look at the truth table for this circuit



Tony DiStefano is well-known as an early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.



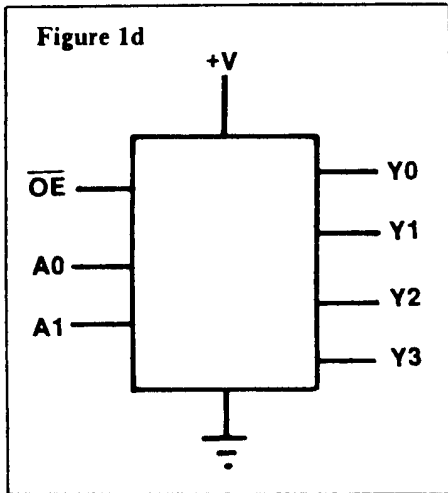
It's time for some theory. Remember when I described the binary number system? This is where it comes in handy. Let's look at two binary bits to start with. Two binary bits have four different combinations: 00, 01, 10, 11. Figure 1b shows a circuit that has two inputs and four outputs. This type of circuit is known as a decoder. There are decoders with two-, three- and four-bit inputs. More about this later. For now, two bits will prove my point. Look again at Figure 1b. When A0 and A1 (on the input side) are both low, Y0 (on the output side) is low and the other three

Figure 1c

INPUTS			OUTPUTS			
\overline{OE}	A0	A1	Y0	Y1	Y2	Y3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	X	X	1	1	1	1

(Figure 1c), notice that binary counting and individual outputs are related.

I hope by now you are starting to understand Hex and binary relations because they get more important as we go along. Figure 1d shows the block diagram of this two-to-four decoder. The other line in our decoder (Figure 1d only) is an input. The name of this line



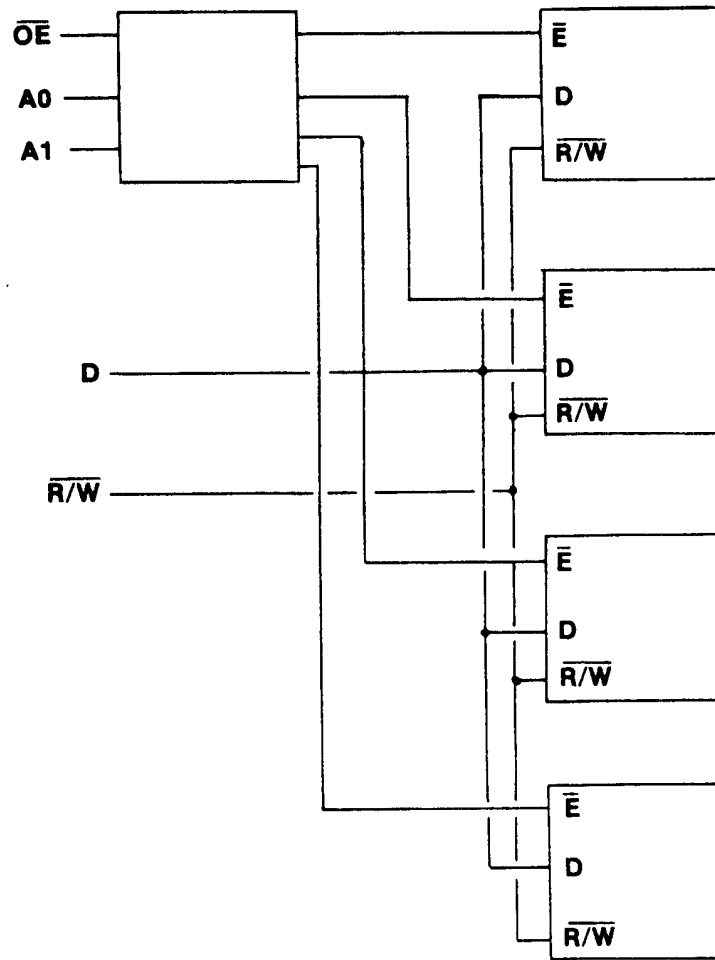
is Output Enable, \overline{OE} for short. When this line is low, all of the preceding is true, but when this is high, the outputs Y0 to Y3 never go low. It can also be known as a Chip Select if it is connected to the right gates.

Figure 2a shows how the decoder and our one-bit memory cell go hand-in-hand. One thing to notice is the decoder inputs are labeled A0 and A1. There is a good reason for using the letter A. In this case and almost all cases, the letter A, along with another number, is short for Address lines. In a computer system, there are address lines to form an address bus.

The definition of address bus is: Address lines are inputs that reflect a binary number and identify a specific position or location in a memory system. Or more plainly, when the CPU wants a specific piece of data in memory, it puts out a binary number equal to the number of the location it wants. The 6809 CPU in the CoCo can specify 65,535 different locations. If you recall, that boils down to 16 binary bits (2 to the power of 16). That is so the 6809 CPU has 16 Address lines, A0 to A15. Maybe we should get back to our two-address memory block.

To continue showing individual gates for decoding would not only be silly, it would take up enormous amounts of room in this magazine. They don't call

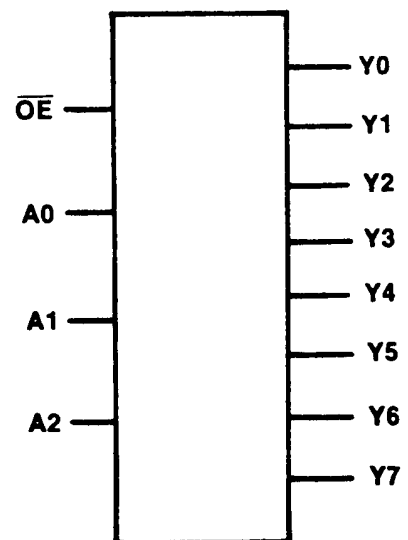
Figure 2a



today's chips LSI (Large Scale Integration) for nothing. A single chip can have the equivalent of a quarter of a million gates. Imagine trying to draw that many gates! It's much easier to draw block diagrams, and as long as you understand the theory behind these blocks, it makes the diagrams a lot easier to read. From now on, I will use block diagrams whenever it is not convenient to use discrete gates.

The block diagram in Figure 2b shows the same idea as Figure 1a, but with more address lines, therefore more output lines, and can thus select more flip-flop memory cells. Each time an address line is added, the number of gates needed to decode the input goes up exponentially and the amount of outputs doubles. If there are four address lines, you can access 16 different locations; if there are five address lines you can access 32 and six address lines

Figure 2b



gives access to 64. Here is a list relating the address lines to the amount of discrete locations possible:

Address Lines	Discrete Locations	
	Decimal	Hex
1	2	2
2	4	4
3	8	8
4	16	10
5	32	20
6	64	40
7	128	80
8	256	100
9	512	200
10	1024	400
11	2048	800
12	4096	1000
13	8192	2000
14	16384	4000
15	32768	8000
16	65536	10000

Look how neat the Hex column is. It's much easier to see the doubling effect of adding one more address line. It's also a lot cleaner.

Up to now, I have shown you only one data bit per location. The CPU can

access eight data bits at a time. We could duplicate the circuit eight times; it was done in the past and is still done in the case of dynamic RAM chips. One bit wide per chip. The CoCo also used chips eight bits wide, but that's ROM. Figure 2c again shows our two-address memory chip but with a twist. Every decoded address line (Y0 to Y3) is connected to eight memory cells. Each of these cells has its own line. Each of these lines is labeled with the letter D and a number.

As you may have guessed, the numbers represent which bit is being accessed. They begin with zero and can go up to any number, usually the amount of data bits that the CPU can handle. Most small microcomputer CPUs have eight bits. They form one byte. Model 100, the Apple II+, the Atari 800 and the Commodore 64 all have eight bits. Other CPUs have 16 bits like the Amiga, the Apple Macintosh and the Atari 520. Then there are more powerful CPUs with 32 and 64 bits. Those are the minis and full mainframe computers.

Another aspect of data bits is a little harder to explain. That is the aspect of internal and external data bus. This leads us to another definition, the Data

Bus: data lines that are bi-directional lines providing communication between discrete components in a computer system. Some CPUs have only eight data lines coming from the CPU, which is to say there are eight pins on the CPU chip, but it can handle more than eight bits internally. Usually a CPU has double the number of internal data capabilities than external. In order for the CPU to read or write double-capacity data, it must do two reads or two writes — one after the other and incrementing the address bus by one before the second. I'll get into the structure of the 6809 at a later date.

If you gather all the information and theory I have given you in the last few articles and stuff it all into one package, what do you get? Presto, you have a full-blown memory chip. Figure 3 shows the pinout of a typical 2K by 8 RAM chip. This chip contains 16,384 memory cells arranged into 2,048 locations of eight bits each. That means 11 address lines (2 to the power of 11 equals 2,048, right?) and eight data lines. It has all of the inputs and outputs that I have been describing in the last few articles. There should be no surprises. The following is a pin-by-pin description of this chip.

Figure 2c

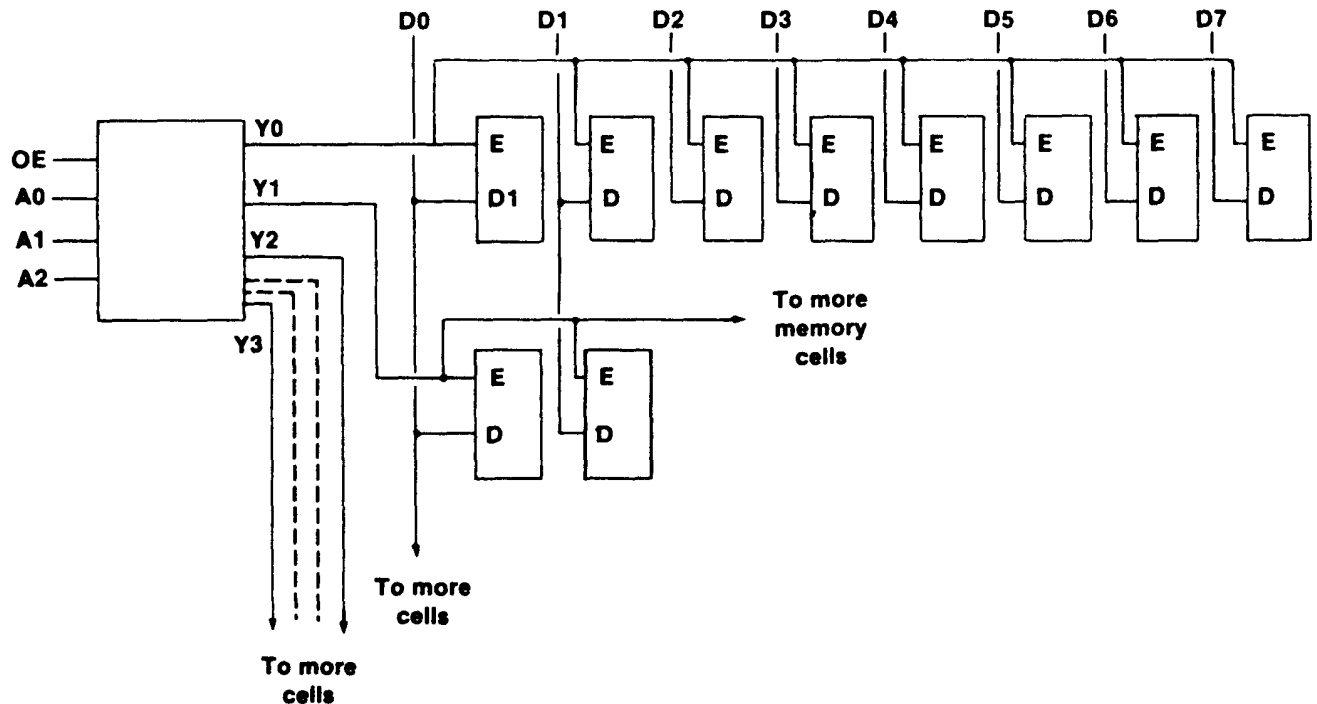
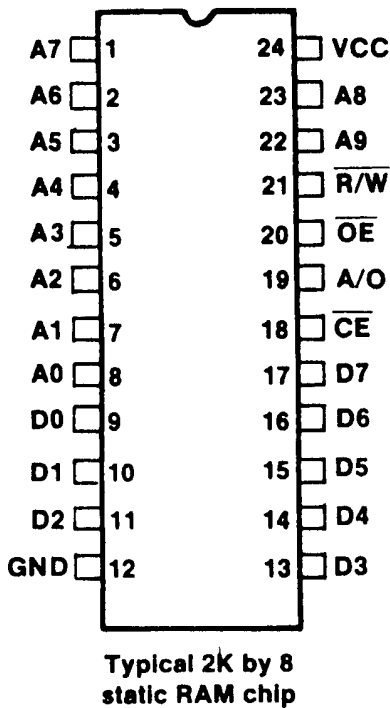


Figure 3



A0 to A10 — These inputs only are address lines that choose which memory byte is to be selected. There are a total of 11 address lines, therefore this chip has a maximum of 2,048 (800) bytes of memory.

D0 to D7 — These bi-directional lines form the data bus in which data can be transferred to or retrieved from the internal flip-flops.

GND — This is an electrical ground to the chip. All signal levels to the chip are with respect to this pin. Commonly known as V_{ss} , it has a voltage potential of zero volts.

Vcc — This input supplies power to the whole chip. The voltage requirement for a typical memory chip is from three to eight volts. The current (power) requirements for a 2K memory chip can range from 10 to 100 milliamps for a regular chip and 10 to 100 microamps for a CMOS chip.

R/W — This Read/Write input determines the direction of data flow through

the data lines. When this pin is high, the memory chip sends out data stored inside. This is a Read action. When it is low, the data lines enter data to the chip to be remembered. This is a Write action.

\overline{CE} — This input selects the chip. When this pin is high, the chip is in the tri-state mode. The chip is inactive and the data lines are not reading or writing.

\overline{OE} — This input is an Output Enable pin. If the \overline{CE} pin is low and this pin is high when reading, the data is ready, but the data lines are kept in tri-state until the \overline{OE} line goes low. When reading, this pin can be used as a second chip select or enable.

All of these lines in one package make up a memory chip. All computers need memory. There are a lot of different kinds of memory chips and what I described here is just one of them. But, whatever the kind of memory or the packaging material used, they are all basically (in theory) the same as the ones described. That's it for this time.

Investigating the CPU

If you have been following the past several articles, you are familiar with how a memory chip works. A memory chip by itself is not useful and a Printed Circuit Board (PCB) full of memory chips cannot do much. We need something that can write to and read from this memory. What we need is a CPU. What is a CPU? It's a Central Processing Unit.

This chip is the workhorse of the computer. It does just about everything. In the CoCo, the CPU is made by Motorola. The part number for this chip is MC6809EPC. The "MC" stands for the company, the "6809" is the part number, the 'E' means it's driven by an external clock and the "PC" means it is a plastic package.

It's common knowledge that the 6809 is one of the most powerful eight-bit CPUs made. In fact, some people argue it is *the* most powerful. Whatever the case, we're going to dig into it and look at it from a hardware point of view.

The most important thing to know about this chip is the pinout. The 6809

is contained in a 40-pin DIP (Dual Inline Package) the same size and shape as the VDG, PIA and SAM chips also in the CoCo. Figure 1 shows this 40-pin chip and the pin names. The following is a pin-by-pin description of the 6809.

Pin 1 — V_{ss} . This is the ground pin to which all signals are referenced. It has a potential of zero volts.

Pin 2 — \overline{NMI} . This normally high (five volts or logic state of one) input triggers on the negative edge of a pulse. This in turn requests that a non-maskable interrupt sequence be generated. A non-maskable interrupt (as the word indicates) cannot be inhibited by the program. It also has a higher priority than \overline{FIRQ} , \overline{IRQ} or software interrupts.

During recognition of this \overline{NMI} , the entire machine state is saved on the hardware stack. After a reset, an \overline{NMI} is not recognized until the first program load of the hardware stack pointer. The pulse width of the \overline{NMI} low must be at least one E-cycle long before it is recognized.

Pin 3 — \overline{IRQ} . This input triggers in the

same way as the \overline{NMI} except it initiates an interrupt request, providing the \overline{IRQ} bit in the CC (Condition Code register) is clear. This also saves the entire machine state on the stack. The \overline{IRQ} has a lower priority than the \overline{FIRQ} . It is up to the service routine to clear the source of the interrupt before doing an RTI (Return from Interrupt).

Pin 4 — \overline{FIRQ} . This input, like the \overline{IRQ} , initiates a fast interrupt request, providing the \overline{FIRQ} bit in the CC is clear. This has higher priority than the \overline{IRQ} , but only saves the CC register and the program counter on the stack. The interrupt service routine should clear the source of the interrupt before doing an RTI.

Pins 5 and 6 — **BS (Bus Status)** and **BA (Bus Available)**. Two outputs that work together to generate the condition of the CPU. When BS and BA are both low, a normal or running condition exists. When BS and BA are both high, it indicates the CPU is in the halt mode. When BS is high and BA is low, an interrupt or reset is acknowledged. And

finally, when BS is low and BA is high, the CPU is in a sync acknowledge mode.

Pin 7 — Vcc. This input powers the CPU with five volts.

Pin 8 to 23 — A0 to A15. These 16 pins are used to generate one of 65,535 different address locations the 6809 CPU can access for data transfer. When the processor does not require the bus for a data transfer, it sends out all that is on the address bus. The R/W line equals one, but BS equals zero. This is known as a dummy access or VMA cycle. All addressed bus lines go into a high-impedance state when BA is high or when TSC is driven high.

Pin 24 to 31 — D7 to D0. These eight bi-directional pins are used to transfer data to and from the CPU and other devices connected on the data bus.

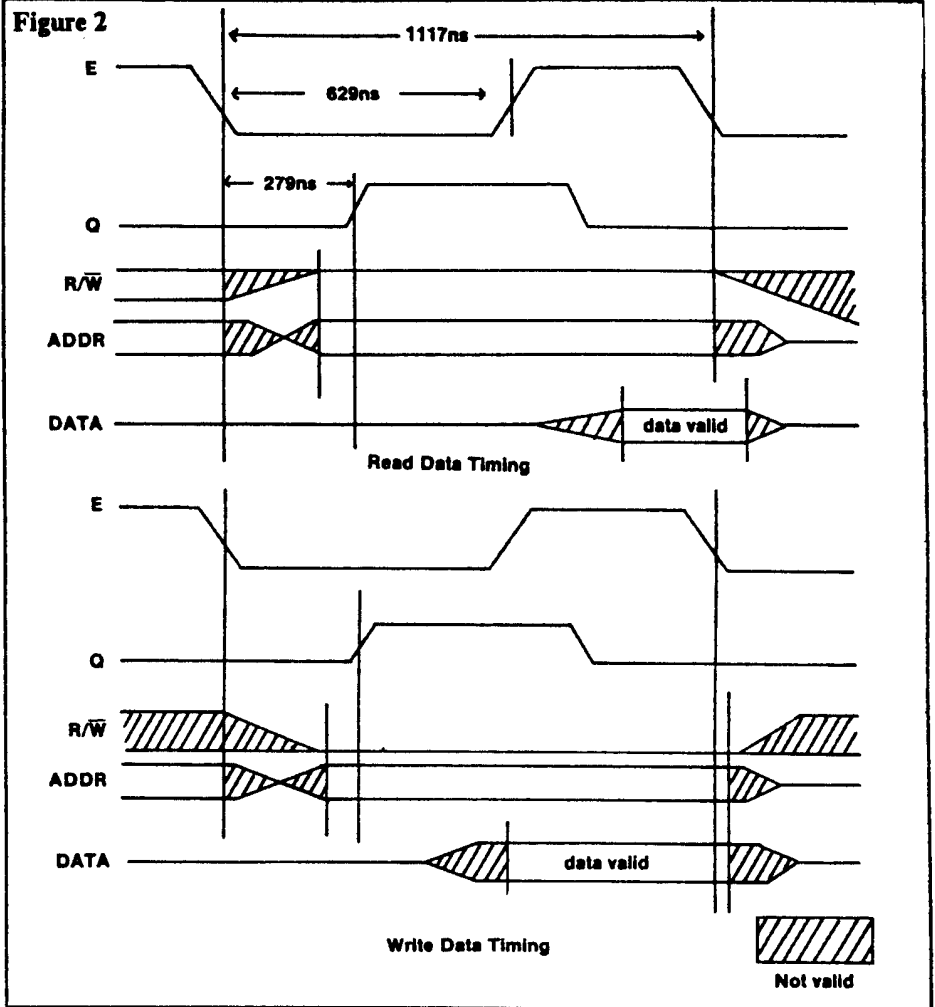
Pin 32 — R/W. This output signal indicates the direction of the data transfer on the data bus. A low indicates the CPU is writing data to the data bus. A high means the CPU is reading. When BA is high or when TSC is high, the output is made high-impedance.

Pin 33 — BUSY. This output-pin signal indicates that bus re-arbitration should be deferred. Wow, what a mouthful! This means BUSY is high for the first two cycles of any instruction that first reads, then writes new data, high during the first byte of a double-byte access, and during the first byte of any indirect access.

Pin 34 and 35 — E and Q. These clock signals are required only by a 6809 that has an 'E' prefix. In the CoCo, these signals are generated by the SAM (MC6883) chip. These signals bring the CPU to life. The 'Q' clock must lead the 'E' clock. Addresses are valid after the falling edge of the 'E' clock, and data is latched from the bus by the falling edge of the 'E' clock. More on 'E' and 'Q' clocks later.

Pin 36 — AVMA. This output is the advanced VMA signal and indicates the CPU will use the bus in the following bus cycle. The predictive nature of the AVMA signal allows efficient shared-bus multiprocessor systems. When the CPU is in either a halt or sync state, the AVMA is low. The CoCo does not support this feature.

Pin 37 — RESET. A low on this normally high input forces the CPU into a reset condition. The reset vectors are loaded into the program counter from locations \$FFFE and \$FFFF, then the CPU begins to execute the instructions it finds. Because the reset threshold



voltage is higher than that of standard peripherals, it ensures all peripherals are out of reset state before the CPU goes to work.

Pin 38 — LIC. The last instruction cycle is high during the last cycle of every instruction, and its transition from high to low indicates the first byte of an opcode will be latched at the end of the present bus cycle. LIC is high when the CPU is halted at the end of an instruction.

Pin 39 — TSC. This three-state control causes the address, data and R/W lines to go into a high-impedance state. The control signals BA, BS, BUSY, AVMA and LIC do not go into the High-impedance state. To force the CPU into this state, TSC must be made high just before the end of the previous cycle. To regain access, TSC is brought low and the clocks for that processor restarted when the addresses become valid.

Pin 40 — HALT. A low level on this input pin causes the CPU to stop running at the end of the present instruction and remain indefinitely without loss of

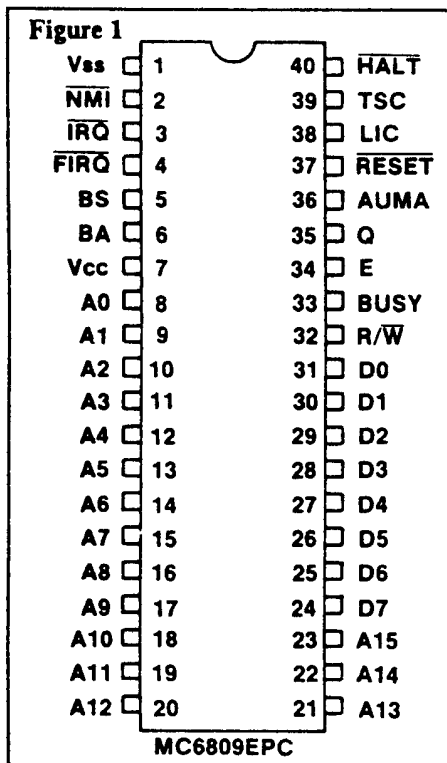
data. When halted, the BA output becomes high, indicating the buses are free. While stopped, the CPU does not respond to external requests, although NMI or RESET will be latched for later response. During the halt state the 'Q' and 'E' clocks must continue to run normally.

Figure 2 is a diagram of the timing information of a read cycle and write cycle for memory or peripherals.

The complete read cycle for the CoCo is about 1,117 ns (nanoseconds) long. The cycle starts with the falling edge of the 'E' clock. Some 200 or so nanoseconds later, the address bus is stable. That means the bus holds a valid 16-bit address. At the same time, the R/W line is stable with a logic level of one. A little later, about 80 ns or so, the 'Q' clock changes to a high condition. But for now, it is not very important.

The next change is the 'E' clock. About 629 ns after the 'E' clock falls, it rises again. This change is important but I'll get to that later. Next, the 'Q'

Figure 1



clock falls to a zero state. Now, most important, when the falling edge of the 'E' clock occurs, the data on the data bus is transferred into the CPU.

There is a small catch: the data must have been valid (stable and not changing) 80 ns before the falling edge of the 'E' clock. It is up to the memory device or peripheral to make sure the data is there on time. The CPU does not wait; if the data is not there on time, wrong data is entered into the CPU.

The second part of Figure 2 is a write cycle. The complete write cycle is the same length as the read cycle, approximately 1,117 nanoseconds. Again, everything starts with the falling edge of the 'E' clock, and again the address bus is stable with a 16-bit address. This time the R/W line is stable with a logic level

of zero. The 'Q' clock rises and the 'E' clock rises; the 'Q' clock falls and the 'E' clock falls. But this time, the CPU supplies the data.

The data is valid no later than 200 ns after the rising edge of the 'Q' clock, which occurs just before the rising of the 'E' clock. The data stays valid until about 20 ns after the falling edge of the 'E' clock. In that time, the memory of the peripheral device must take the data from the data bus. Then another cycle starts. The CPU decides whether it is a read or a write, depending on what it's doing next.

How the CPU decides depends on what it did in the previous cycle. When the computer is first turned on, the reset line keeps it from doing anything until everything stabilizes. When the reset line starts the CPU going, it always does the same thing — two reads. These two reads are always at the same place, \$FFFE and \$FFFF. This is the reset vector, which is a pointer that points to a memory location. Because the 6809 can access 65,535 bytes of data (16 bits), the pointer must be exactly 16 bits long. Since the 6809 can access only eight bits at a time, the pointer must be two bytes long.

After the CPU reads these two bytes, it places them in an internal register called the program counter. This program counter always points to the CPU's next instruction. The CPU reads the first instruction. Instructions in the 6809 can be one to four bytes long, so the CPU has to read zero to three bytes more depending on the instruction. After the complete instruction has been read, the CPU acts on it. This instruction could be read data, write data or do something internal. Whatever the case, the CPU continues to read and write until turned off.

Now we know how a CPU accesses

devices on the bus. It's time to join the CPU and the memory chips discussed in previous articles.

For example, let's use an 8K memory device. This could be a ROM or a RAM chip. It has 13 address lines, A0 to A12. Two to the power of 13 is 8K. Since the CPU can access 64K, eight of these memory chips can be used. But how? Some sort of decoding has to be set up. If we hook up the first 13 address lines to the CPU, we are left with three unused lines.

Now, we can use a three to eight decoder chip (explained in an earlier article). If we connect the three unused address lines of the CPU to this decoder, we have eight individual address locations. In turn, these eight lines can be used to control the chip enable lines of eight 8K memory devices. That brings our total to 64K of memory. Mixing the 8K devices between ROM and RAM would give us a complete computer. Well, almost.

Eight chips of memory do not a computer make. It needs a little more than that. Things like a keyboard, video, drives and joysticks are a must on a computer. These connect a computer to the real world. But these things are not as complex as you might think. They are just more devices connected to the CPU via address lines, data lines and control lines.

For instance, the keyboard is simply a bunch of switches and, through a device called a PIA (Peripheral Interface Adapter), the CPU monitors the switches and interprets them according to the software. Each switch represents a letter of the alphabet or a number. That's all. PIAs and other I/O devices take up little room in a memory map. A PIA only takes up four bytes. More decoder chips are needed in order to map it properly, but the same theory is used.

Investigating the PIA

This month I'm looking deeply into a PIA. The letters PIA stand for Peripheral Interface Adapter. The Color Computer uses two of the PIAs. The older, regular CoCo uses two MC6821 PIAs. The newer CoCos and the CoCo 2s use one MC6821 and one MC6822. The differences between the two are minor. The 6822 is called an IIA. This stands for Industrial Interface Adapter. Both have the same pinout and function in the same way. I will describe the differences between them later in this article.

It's interesting to know what PIA stands for, but what does it do? A PIA provides the means of interfacing external hardware or devices to a computer. In our case, the MC6809 CPU. Most devices do not conform to the specifications of a CPU. Take, for instance, a switch. That's right, an everyday household switch. It turns on the lights, stove, radio and so forth. It works well, but us not computer compatible. This is where a PIA comes in. It's a go-between from the CPU to the switch. With a PIA and a little circuit, the CPU can tell if the switch is on or off. Or in CPU terms, a zero or a one. This is known as an input. If the computer had to control a light or a motor, a PIA would be used to switch a transistor on or off and, in turn, the transistor would control a relay and the relay would turn the motor on or off. This is known as an output.

This particular PIA has two bidirectional eight-bit peripheral data buses for interface to external devices and four individually controlled interrupt and interrupt disable capability plus two control registers and two data direction registers.

The PIA, like many other devices, looks like memory to the CPU. Therefore, the PIA must have address lines, data lines and control lines such as chip enable and read/write. Figure 1 shows the pinout of an MC6821 PIA chip. You should, by now, recognize many of the pins and their names. The following is a pin-by-pin description of this chip.

PB0 - PB7 - The second eight peripheral data lines, which can be programmed as outputs or inputs.

CB1 - Is an input only line that sets the interrupt flags of the B control register.

CB2 - Is either an interrupt input line

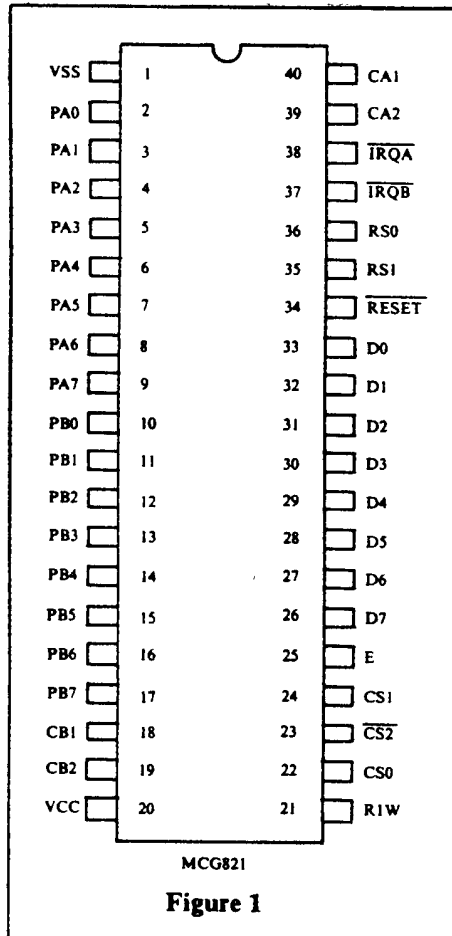


Figure 1

or a peripheral control output line.

Vcc - This is the five volt input that powers the chip.

R/W - This input controls the PIA as a read or a write to the registers.

CS0 - Chip Select 0 is an active high input. When this pin is low the chip is disabled.

CS2 - Chip Select 2 is an active low input. When this pin is high, the chip is disabled.

CS1 - Chip Select 1 is an active high input. When this pin is low, the chip is disabled.

E - This is the Enable clock or the 'E' clock. Used to enable input or output.

D7 to D0 - These are the eight data lines that the CPU uses to read and write data to the PIA.

RESET - This active low input initializes the PIA to power up conditions.

RS1 - This input is the second address line used to access one of four locations on the PIA.

RS0 - This input is the first address line used to access one of four locations on the PIA.

IRQB - This active low output is used to generate an interrupt to the CPU from port B. The method of interrupt depends on how the control register is set up.

IRQA - This active low output is used to generate an interrupt to the CPU from port A. The method of interrupt depends on how the control register is set up.

CA2 - Is an input only line that sets the interrupt flags of the B control register.

CA1 - Is either an interrupt input line or a peripheral control output line.

First let me talk about the structure of the PIA. Basically, there are two ports. Each port has two control lines. Each PIA has two address lines and takes up four memory locations in the CPU's memory map.

Table 1 shows the memory map of a PIA. Address locations 0 and 2 are ports A and B respectively. Address locations 1 and 3 are control registers A and B respectively. I hope by now you can recognize addresses by binary bits. It may be a little confusing as to what CRA2 and CRB2 have to do with the memory map. There are actually six registers to a PIA. But, if you remember your binary math, six is not an even power.

The designers could have added another address line and wasted the other two address locations. But instead, they put a software switch in the control register. Bit 2 to be exact. When the switch (bit 2) is low (zero) then address 0 or 2 becomes a data direction

register. If you write a one in any bit position in that register, that bit becomes an input. On the other hand, if you write a zero, that bit becomes an output.

After all bits have been selected as ins or outs, then turn the switch at CRA2 or CRB2 back to a one. Now the 0 and 2 addresses become input and output peripheral ports as programmed.

The next part of the PIA is a little more complex. This includes control bits and interrupts. Along with the two eight-bit ports, this PIA also has four other pins. There are two pins used for inputs or outputs and there are two pins that are inputs only. These four pins work in conjunction with the bits in the control register of the PIA. Table 2 explains the bit names of control register A (CRA) and control register B (CRB).

Let's look at CA1 and CB1 first. They are inputs only. On given conditions, these inputs generate an interrupt. Bits 0 and 1 in the respective control registers have the following influence on the interrupts. If bits 0 and 1 are both low (either register), the interrupts are disabled and no interrupts go through. Only the interrupt flags are set on the falling edge of the input. If bit 1 is low and bit 0 is high, the falling edge of the CA1 or CB1 input causes an interrupt and sets the flag. Bit 1 high and bit 0 low sets the flag on the rising edge of the input but does not cause an interrupt. Bit 1 high and bit 0 high causes an interrupt and sets the flag on the rising edge of the input. The CA1 and CB1 interrupt flags are on bit 7 of the respective control byte. In other words, bit 1 enables or disables the interrupts and bit 0 controls on which edge the input causes an interrupt.

Bits 3, 4 and 5 of the control byte control the CA2 and CB2 pins. These pins are a little more flexible than the CA1 and CB1 pins. They can be outputs or inputs controlled by bit 5. If bit 5 (on either control byte) is high, then the pin is an output. If it is low, then it is an input. When bit 5 is low, bits 4 and 3 make these pins behave exactly like bits 1 and 0 make pins CA1 and CB1 behave. When CA2 or CB2 are initialized as outputs, they behave a little differently.

Let's look at CA2 first. There are four possible combinations of operation. The first is when bit 4 is low and bit 3 is low. This goes low after the first negative transition of the E clock after the CPU reads Port A. It returns high

RS1	RS0	Register Bit		Location Selected
		CRA-2	CRB-2	
0	0	1	x	Peripheral Reg. A
0	0	0	x	Data Direction Reg. A
0	1	x	x	Control Reg. A
1	0	x	1	Peripheral Reg. B
1	0	x	0	Data Direction Reg. B
1	1	x	x	Control Reg. B

Table 1: Internal Addressing

BITS	7	6	5	4	3	2	1	0
CRA	IRQ A1	IRQ A2	CA2 Control			DDR A	CA1 Control	
CRB	IRQ B1	IRQ B2	CB2 Control			DDR B	CA2 Control	

Table 2: Control Registers

when the interrupt flag is set (CRA-7) by the active transition of the CA1 signal. If bit 4 is low and bit 3 is high, it is the same, but goes high after the first 'E' clock cycle. This mode is used mainly as an acknowledgment of a read (handshaking) to another peripheral. If bit 4 is high and bit 3 is low, CA2 is low. If bit 4 is high and bit 3 is high, CA2 is high. This mode is used when CA2 is to be used as a latched bit to control an external device.

Next there is CB2. There are also four possible combinations of operation for this output pin. When bit 4 is low and bit 3 is low, this pin goes low on the positive transition of the first 'E' clock as a result of a write to the B port; then goes high again when the interrupt flag bit (CRB-7) is set by an active transition of the CB1 input. When bit 4 is low and bit 3 is high it is the same, but goes high again on the positive edge of the first 'E' clock following that write. This mode is used when there is a need to autostrobe or select an exterior device. If bit 4 is high and bit 3 is low, it causes CB2 to go low and stay low. If bit 4 is high and bit 3 is high, it causes CB2 to go high and stay high. This is another latched bit to control an external device.

In Conclusion

There you have it, the internal work-

ings of a PIA. As I stated before, there are two such beasts in our CoCos and CoCo 2s. If you want to add a third PIA, the most logical place to put it in the memory map would be in the Spare Chip Select area. This is at \$FF40 and is 16 bytes long. That is the same place that the controller is mapped. You could always use a Multi-Pak Interface. You should now know enough about CPUs and signals to interface this PIA to the computer, but for those of you who are still unsure, I have included some guidelines.

Using the pinout of the PIA in Figure 1 and the pinout of the CoCo expansion bus in earlier articles, connect the following signals together. Five volts to five volts. Ground to ground. All eight data lines to all eight data lines. The first two address lines of the CPU to RS0 and RS1 respectively. The R/W line to the R/W line. The 'E' clock to the 'E' clock. The RESET line to the RESET line. The SCS line to the CS2 line. And finally, CS0 and CS1 to five volts. You can connect IRQA and IRQB to the cart line of the computer, but watch out, this can (under certain conditions) cause an interrupt that makes the computer crash. Make sure you know what you're doing with the interrupt routines for the CoCo and the setting of the interrupt pins in the PIA. ☺

Timing and the SAM Chip

As we all know, the CPU in our CoCo is the MC6809E. It is the heart of the computer. It requires RAM and ROM and I/O and Video and so on to help support it. All these devices must be memory mapped. They must appear somewhere in the 64K bytes of memory the CPU can access. The proper timing and sequencing must be within the specifications of the CPU. Normally, a handful of TTL (74 series) logic chips take care of this. In the CoCo one big chip takes care of all of this and more. The chip is the MC6883, sometimes known as the 74LS783N. The name of this chip is a Synchronous Address Multiplexer or SAM for short. This is a 40-pin chip that mates the MC6809E and the MC6847 (the video chip). This chip also does all of the dynamic memory refresh timing and memory mapping of all the other major chips of the CoCo. As you can see, this thing is a real workhorse of a chip. By the time I am finished describing this chip, everyone will have as much respect for it as I do.

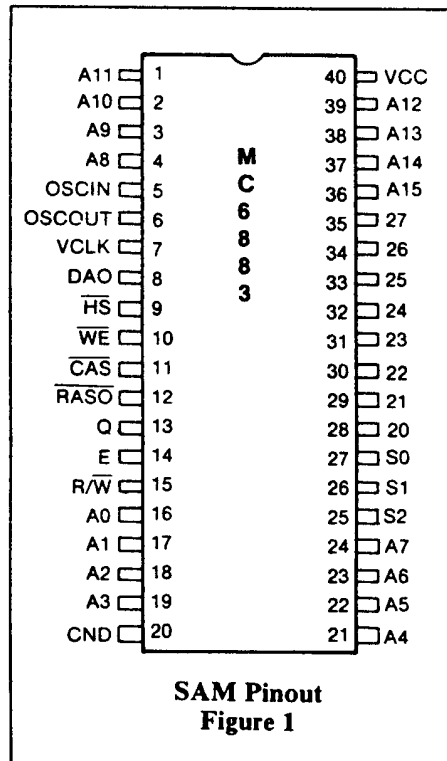
The first part of this article is a pinout of this chip. Figure 1 shows the pinout of the SAM chip. The following is a pin-by-pin description of the Motorola chip number MC6883.

Pin 1 to 4 — A11 to A8. Address lines A11 to A8 respectively from the MC6809E. These are four of the 16 address lines the SAM requires to fully control the memory mapping of the CPU.

Pins 5 and 6 — OSCin and OSCout. These are the crystal oscillator inputs. A crystal and supporting components supply the SAM a master frequency of 14.31818 MHz. This is the highest frequency available in the CoCo.

Pin 7 — VClk. The first function of this pin is to generate an output of 3.579545 MHz. This supplies the color carrier for the VDG (Video Display Generator) Clk pin. The second function resets the SAM when this pin is pulled to a logic level of 0, acting as an input. In the CoCo, this pin is part of the reset circuitry.

Pin 8 — DA0 (Display Address 0). The function of this pin as described in the Motorola manual is the least significant bit of a 16-bit video display address. The more significant 15-bits are outputs from an internal 15-bit counter which is clocked by DA0. The second function,



falling edge of the pulse in order to initiate eight dynamic RAM refresh Column Address Strobe. It strobes the most significant 6, 7 or 8 address bits cycles. It also resets four least significant bits of the internal video address counter.

Pin 10 — WE. This output is the write enable pulse that enables the CPU to write into dynamic RAM.

Pin 11 — CAS. This output is the into dynamic RAMs.

Pin 12 — RAS0. This output is the Row Address Strobe 0. It strobes the least significant 6, 7 or 8 address bits into dynamic RAMs in Bank 0.

Pin 13 — Q. This output is the Quadrature clock used by the CPU that leads the 'E' clock by about 90 degrees.

Pin 14 — E. This output 'E' clock, better known as the Enable clock, is used by the CPU. It is the main CPU timing and is also used by most peripheral devices. This clock determines the speed at which the CPU operates.

Pin 15 — R/W. This input is fed from the CPU's R/W line. It tells SAM whether the CPU is reading or writing data to memory, writing to the SAM registers or device 0.

Pins 16 to 19 — A0 to A3. Address lines A0 to A3 respectively from the

not used by the CoCo, is to indirectly enter the logic level of the VDG FS (field synchronization pulse) for vertical video address updating.

Pin 9 — HS. This input, connected to the HS output of the VDG, detects the

Table 1 SAM Control Registers

Address	Mode	Label	Name	Descriptions																											
FF DF	S	TY	map type	1 = all RAM 0 = ROM/RAM																											
FF DE	C																														
FF DD	S	M1	memory																												
FF DC	C																														
FF DB	S	M0	size	00 = 4K 01 = 16K 10 = 64K 11 = static RAM																											
FF DA	C																														
FF D9	S																														
FF D8	C	R1	CPU rate	00 = slow 01 = dual speed 11 = fast																											
FF D7	S																														
FF D6	C	R0																													
FF D5	S																														
FF D4	C	P1	page #	SET = PAGE #1 CLEAR = PAGE #0																											
FF D3	S																														
FF D2	C	F6	display offset	start of display address and 512 byte offset																											
FF D1	S																														
FF D0	C	F5																													
FF CF	S																														
FF CE	C	F4																													
FF CD	S																														
FF CC	C	F3																													
FF CB	S																														
FF CA	C	F2																													
FF C9	S																														
FF C8	C	F1																													
FF C7	S																														
FF C6	C	F0																													
FF C5	S																														
FF C4	C	V2	VDG MODE (SAM)	<table border="1"> <tr> <td></td> <td>1</td> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td></td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td></td> <td>1</td> <td>0</td> <td>1</td> <td>0</td> <td>1</td> <td>0</td> <td>1</td> <td>0</td> </tr> </table>		1	1	1	1	0	0	0	0		1	1	0	0	1	1	0	0		1	0	1	0	1	0	1	0
	1	1			1	1	0	0	0	0																					
	1	1			0	0	1	1	0	0																					
	1	0			1	0	1	0	1	0																					
FF C3	S	V1																													
FF C2	C	V0																													
FF C1	S																														
FF C0	C																														

S = set C = clear

MC6809E. These are the next four of the 16 address lines the SAM requires to fully control the memory mapping of the CPU.

Pin 20 — GND. Return ground for the five volts. Signal ground to which all signals are referenced.

Pins 21 to 24 — A4 to A7. Address lines A4 to A7 respectively from the MC6809E. These are the next four of the 16 address lines the SAM requires to fully control the memory mapping of the CPU.

Pins 25 to 27 — S2 to S0. S2 is the most significant bit of the three device select bits. The binary value of these three pins selects one of eight chunks of CPU memory map, device 0 to 7. Varying in length, these chunks provide chip selects for three ROMs, RAM, three I/O areas and boot area; the boot area is not used in the CoCo.

Pins 28 to 35 — Z0 to Z7. These are the eight multiplexed address lines needed to access 64K dynamic RAM. With 16K dynamic RAM, only Z0 to Z6 are used and Z7 is $\overline{\text{RAS1}}$ for a second bank of 16K chips. With 4K dynamic RAM, Z6 is not used. These lines are also used to generate the video address refresh on the alternate 'E' cycle.

Pins 36 to 39 — A15 to A12. Address lines A15 to A12 respectively from the MC6809E. These are the last four of the 16 address lines the SAM requires to fully control the memory mapping of the CPU.

Pin 40 — Vcc. This pin requires +5 volts. It powers all the functions in this chip.

As you can see from these descriptions, the SAM chip and VDG chip are closely linked. The SAM chip generates data from its RAM and delivers it to the VDG. That is one of the functions of the SAM. It works closely with the VDG monitoring the horizontal and vertical syncs in order to give it the proper data that the VDG later converts to a video signal. The SAM has many modes in which it delivers video data to the VDG. These modes are selected by a set of registers in the SAM's memory map. But since the SAM chip has no data lines going to it, the registers are accessed by writing to odd address locations to set the register and writing to even address locations to clear the register. The data written to these locations is irrelevant. Table 1 shows all the SAM control registers and their functions. Most of the registers shown are used with the VDG.

Pins S0 to S2 are used to decode chunks of memory. These so-called chunks of memory are what memory maps the CoCo into what we know it to be. For instance, BASIC is one chunk that is 8K long. Disk Extended BASIC is another chunk that takes up 16K. These eight chunks are decoded from the three pins by using a 74LS138. You might remember this from a past article, but if you don't, a '138 is a three-input

to eight-output decoder; just what the doctor ordered. Each one of these eight outputs controls one chunk of memory. Table 2 shows all eight chunks and describes where in the memory map they appear and what use each has in the CoCo.

In Table 2, notice that part of S7 are the SAM control registers. Table 1 describes the SAM control registers. The SAM control registers are divided into six areas. The following is a description of each of these areas.

The first area is the map type. When cleared, the SAM is in the map type 0. This is the mode that BASIC sets it up to be. The ROMs are active and a maximum of 32K RAM is accessible. When set, the SAM is in the map type 1. This mode is better known as the 64K mode or the RAM mode. In this mode none of the ROMs are active but all 64K RAM is accessible. The OS-9 operating system uses this mode.

The next mode is the memory size. The SAM can use three types of dynamic memory, 4K, 16K and 64K. When your CoCo is first turned on, a routine in the BASIC ROM checks to see what kind of RAM is installed and sets the SAM chip accordingly.

The third mode is CPU rate. The SAM has some control as to the speed at which the CPU can operate. It has three choices; the first is called slow. In this mode the CPU runs at .894 MHz. The next is the dual speed mode. De-

Chunk Name	Mapped Area	Description	Chunk Name	Mapped Area	Description
S0	\$0000 to \$7FFF 0 to 32767	This area in a 64K machine is 32K of user RAM.	S5	\$FF20 to \$FF3F 65312 to 65343	This area is also 32 bytes long and again only four bytes are used for a PIA to which the VDG controls, D/A, cassette motor, RS-232 and interrupts are connected.
S1	\$8000 to \$9FFF 32768 to 40959	This area is occupied by the 8K Extended BASIC ROM chip.	S6	\$FF40 to \$FF5F 65344 to 65375	This 32-byte area is used with a disk controller to control things like drive select, FDC control and drive motors.
S2	\$A000 to \$BFFF 40960 to 49151	This area is occupied by the 8K Color BASIC ROM chip.	S7	\$FF60 to \$FFDF 65276 to 65503	This is not used except for the SAM control registers.
S3	\$C000 to \$FEFF 49152 to 65279	Normally this area is occupied by the 8K Disk ROM chip, but this area can access up to 16K.			
S4	\$FF00 to \$FF1F 65280 to 65311	This area is 32 bytes long. Four bytes are used for a PIA to which the keyboard HS, VS and audio select are connected.			

Table 2

pending on where the CPU is accessing memory, it can access it at .894 MHz or at the faster 1.78 MHz. At the dual speed, S0 and S4 are accessed at the slower speed, all other accesses are at the higher speed. The third speed is the fast speed. This is where all accesses are done at the high speed, but at that speed, the SAM chip does not have the time to do video. The video screen displays garbage.

The fourth mode is the page mode. When the SAM is in map type 0 and is using 64K memory chips, only half, 32K, of memory is used. The other half is just sitting unaccessible. Setting this register switches in the other half of

memory and switches out the first half.

The fifth mode area is a big one in that it takes up a lot of room. This is the display offset. This offset tells the SAM chip where in memory to start the video scanning. Since the smallest memory area the SAM can scan is 512 bytes, all offsets are 512 bytes apart. The display offset is a binary address to the start of the video display.

The sixth area is the VDG mode. Since graphics pages take up more memory than text, the SAM has to scan more memory. The amount of memory scanned depends on the graphics resolution mode required. Basically there are three amounts of graphics memory.

The first is 1.5K memory, the second is 3K and the highest is 6K. These modes of graphics must match the graphics mode the VDG is set to. You will find more detail on these modes in the BASIC manual supplied by Radio Shack.

The last mode is reserved for future use. Who knows what Motorola has in store for these unused registers.

The SAM chip is a very complex chip, indeed. I have just described only the major parts of this chip. Complete details on this chip are available from your Motorola dealer. The details I have given are taken from that manual and the TRS-80 Color Computer Technical Reference Manual, available at your local Radio Shack store. ☺

More on the New Video Display Generator

Last month I described the new VDG (Video Display Generator) LMC6847T1 and the modes that are possible. I also showed you how to hook up a few switches in order to access these modes. The only problem with this is the new VDG is only available in the CoCo 2 'B' model. At home, I have the regular white CoCo. They call it the 'F' board. I wanted the new TI chip in my CoCo, too. So, with the help of Bill Warnica, I modified my 'F' board CoCo to work with this new chip.

The new VDG and the old VDG are very similar but not pin-for-pin compatible, so you can't just pull the old one out and plug the new one in. It is, however, not too difficult to modify the computer board to make it fit. The new VDG also has built-in hardware that saves two chips on the computer board.

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

The chips that are saved are no longer on the 'B' board. That is why the new board is smaller than the older boards. The two chip numbers saved are the 74LS244 and 74LS273. These chips are TTL logic gates used to isolate the CPU data bus from the video data bus.

Without getting into too much detail, these two chips are now part of the VDG and are no longer needed on the main board. At first, it was thought that both of these chips had to be removed from the old PC board and the new VDG completely rewired to fit in. Luckily, it turns out that only one of these chips has to be removed. This saves a lot of wiring.

Like most of my projects, this one requires you to open the computer and dig inside with a soldering iron and some tools. A good hardware hacker with experience is needed to do this one. To do this project, you will need a soldering iron, tools, wire, solder and, of course, a new VDG. More on the parts later.

The upgrade I did was on an 'F' board CoCo. As far as I know, these instruc-

tions work for just about every CoCo and CoCo 2, but on certain models, the VDG and other parts involved are soldered directly onto the PC Board. That means you have to unsolder the chips and insert a socket. This can be done, and I have done it many times, but it requires a solder sucker or chip remover. Soldering experience is necessary. Also, before you start, be forewarned! The jumpers I will tell you to install in the 'F' board may be different on different boards. But, not all is gloom and doom. A little trial and error and you should find the right pin numbers.

There are two parts you need. The first is the VDG, Motorola part number MC6847T1. If you cannot get this part at your local electronics store, try Radio Shack. The part number is MX-6551. The next part is just a plain and simple resistor. The resistor value is 1K or 1000 ohms quarter watt or half watt. That's it; the rest is a little bit of work.

Unplug the computer, undo the case, remove the keyboard, etc. You know, all those boring things.

Now comes the fun-part. The first thing you must do is remove the VDG. That's simple. It's the chip marked MC6847, or U9 on the 'F' board. On other boards, the U number might be different but it will always be the MC6847. On some boards the VDG is soldered in. In that case, you must unsolder the VDG and insert a 40-pin socket. Prepare the new VDG (T1) in the following manner. Cut the resistor leads so that it will just fit between pins 25 and 11. Put the resistor across the top of the VDG and solder one end of the resistor to the top part of Pin 25. Make sure the solder doesn't leak down the pin. Next, solder the other end to Pin 11 (same precaution). Now pry out Pin 31 vertically, so it does not insert into the socket when you plug the new VDG in.

Insert the new VDG into the socket. Make sure Pin 1 is in the right place. Now solder a short piece of wire-wrap wire to Pin 1 of the VDG. Don't solder the pin to the socket. You won't be able to get the chip out if you do. (If you prefer, solder all connections to these pins before inserting the chip into the

socket.) Solder the other end of this wire to Pin 31, the one that you bent up before. Solder a second wire to Pin 12 of the new VDG. Run this wire to Pin 10 of the SAM (Synchronous Address

"Never connect two outputs together, and never connect two inputs together."

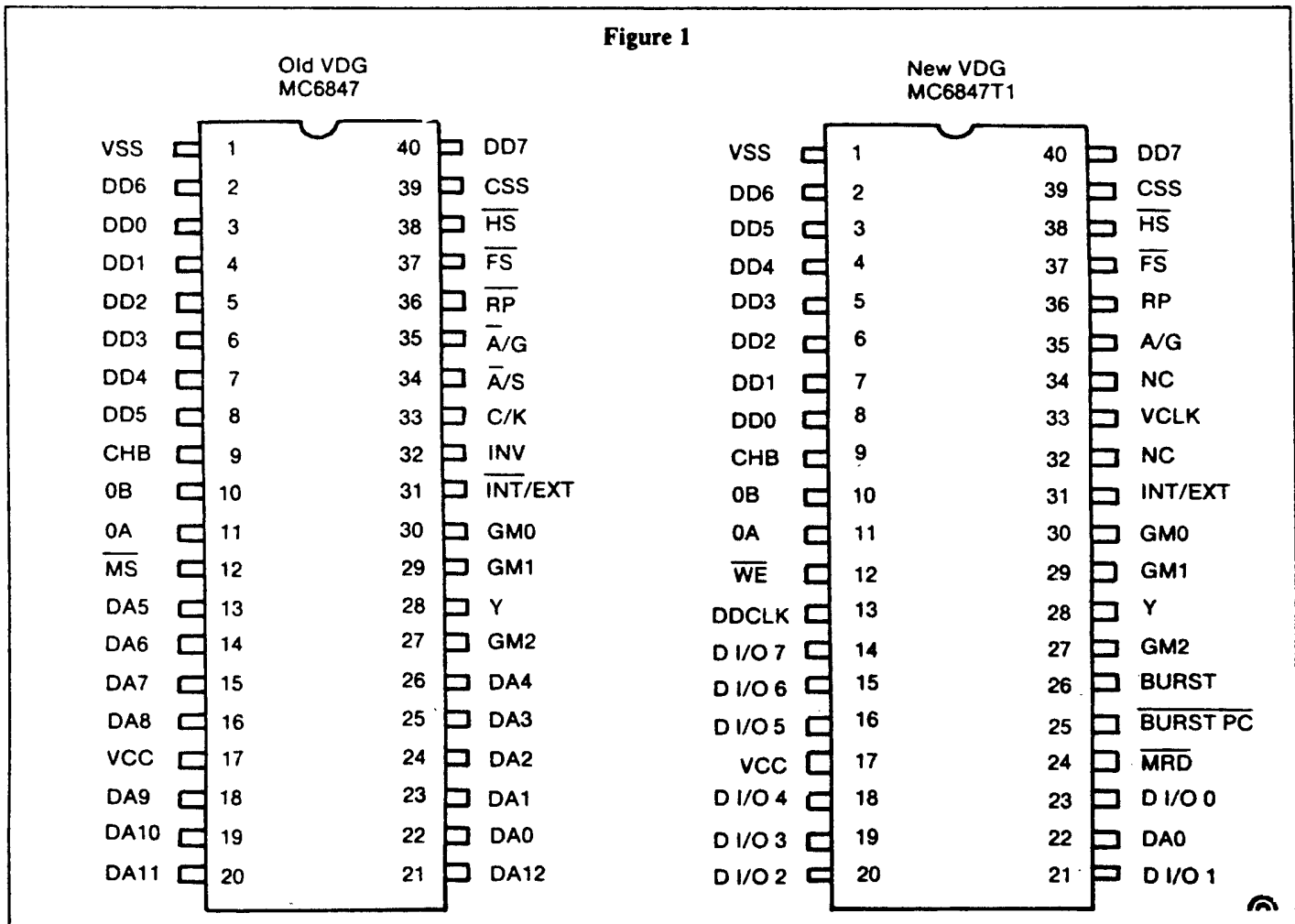
Multiplexer). You remember ol' SAM, she's the one that does all the timing in the CoCo. I did an article on her not long ago in this magazine. Her name is MC6883 or SN74LS783N.

It was said, by whom I don't know, that you needed the new MC6885 or SN74LS785N SAM in order to make this new VDG work, but this rumor turns out to be false. The old one works just fine. As a matter of fact, I have the

old SAM in my CoCo and it just purrs along. Anyway, back to work. Solder a third wire from Pin 13 of the VDG to Pin 12 of the SAM. That's about it for the VDG. But there is a little more work to do.

The next stage of this project deals with the buffer chip I mentioned earlier. Start off by removing the chip, number 74LS273, from its socket. You no longer need this chip, but keep it in your parts bin for a rainy day or in case you decide to remove the modification and replace the old VDG chip. The modification I did is on the so called 'F' board and the 74LS273 chip labeled U13 on the PC board. It also was not soldered in. It had a socket; all I had to do was pull it. If you are doing this on another board and the IC is not socketed, you must do a little more work. First, remove the old chip. Then solder in a 20-pin socket. You need the socket for this next step.

Prepare eight (about 1.5 inches) short pieces of wire by stripping 3/16 inch of insulation off each end. Use a number 22 or 24 gauge solid wire. Old Bell wire is best. Now insert each wire into the pins of the 20-pin socket as follows.



One End	Other End
3	12
4	15
7	9
8	6
13	2
14	5
17	16
18	19

Pins 1, 10, 11 and 20 are left empty. Do not connect anything to these pins. (Pin 10 is ground and Pin 20 is the 5-volt supply. You may use them if you need these power connections in other projects.)

Now, this chip is called an Octal D-type flip-flop. If you recall, many moons (monthly articles) ago, I described flip-flops; they are no more than a sort of latch. This particular chip has eight latches. One for each of the eight data bits of the CPU. Each of these bits has an input and an output. I have arranged the pin numbers in such a way that the One End column pin numbers

are all inputs and all the pin numbers in the Other End column are outputs. This is important to know. Notice that one jumper exists for every in/out pair. If you are trying to modify a board other than the 'F' board, the pin numbers may not match. Not having tried all the CoCos and CoCo 2s, I cannot print every pin diagram. Try to wire the connections as they stand above, but if the screen looks confused and you do not get the same letters on the screen you type on the keyboard, it's because the pinout is different.

In that case you will have to do a trial and error method to get the right combination. There are two rules to follow: Never connect two outputs together, and never connect two inputs together. The first may cause permanent damage to your computer. Jumper all eight wires and try it. If it is not right, make note of the combination you did and try another. If you do combinations in order, you will eventually get the right combination. When you do, if you send

me the pinout combination and which computer board you did it on, I will print them in the next article I write and give you credit for it.

That's all there is to it! Plug everything back in and turn it on. You now have the new VDG in your CoCo. If you want to access the new modes of the new VDG, you will have to do a little more work. Last month, I wrote on how to access the new modes using switches or software. It works for this modification perfectly. All you have to do is follow the instructions and use the method that suits you best. Next month, I'll show you how to use the new modes without switches. All you will need are a few electronic parts. When you change modes from text to graphics, you won't have to throw all your switches — the electronics will do it for you.

For those who are interested, Figure 1 shows the pinouts of the old and the new VDGs side by side so you can compare the differences between them. □

Let's Take a Look at the CoCo 2 B

This week I had the honor of repairing an old 'D' board Color Computer belonging to "KISSable OS-9" author Dale L. Puckett. Although there are a lot of old CoCos still out there, you can't get any more of them. Today, Radio Shack is peddling a CoCo with the letter 'B' in the catalog number. I don't know what the 'B' stands for, but there are a few changes inside. I bought one at the Palo Alto RAINBOWfest. What I want to do here is explain some of the changes Radio Shack has made.

The first thing I noticed when I opened the box is that it says Tandy on the computer and not Radio Shack. It also says Color Computer and not CoCo 2. This is the smallest PCB (Printed Circuit Board) I have seen for a CoCo. Small is good in many ways. First, it costs less to produce. It also has the least parts count of all the CoCos ever made. Not only is this good for

production costs, it's also good for users. The lower the parts count in a computer, the less likely a breakdown. Then there is the question of heat; all electronic parts, whether digital or analog, dissipate heat. How many times have you heard that the computer crashes when it is too hot? Fewer parts mean less heat.

This computer does not have any regulated 12 volts, the same as the other CoCo 2s. There is no negative voltage available except on the SALT chip, which buffers and converts the RS-232 signals. In theory, RS-232 specifies that the signal be +/-12 volts. This new CoCo 2 (and all other CoCo 2s) have only +/-5 volts. While this will work with most RS-232 devices, check the specifications to be sure. Again, as with the other CoCo 2s, there are about 12 volts unregulated at the power diodes, which can be used for devices needing the voltage. The diodes are numbered

D10 and D11. Remember, the side with the white band is the positive side.

The next interesting part in this CoCo 2 B is a PIA (Peripheral Interface Adapter). The first CoCos had two PIAs of the same kind. They were both MC6821s by Motorola. The next stage of the CoCo had one MC6821 and one MC6822. This 6822 is called an IIA (Industrial Interface Adapter). There is just a small difference between the two. Now the second PIA in the CoCo 2 B is no longer an MC6822, but an SC67331P. It is a Motorola part, and compatible with the MC6822. The difference is in the impedance matching between the keyboard and the PIA — custom made for Tandy, no doubt. If you happen to destroy this part, a regular MC6822 will work. The keyboard matrix is the same.

As with the CoCo 2 A, there are six jumpers, J1 to J6. One of the jumpers is used to detect the presence of 64K

memory RAM. The other five jumpers are labeled 64K/128K. A lot of people think that this means you can have 128K of RAM. This is not true. Look again; there is only one place for ROM. Before, there were two sockets, one for the BASIC ROM and the other for the Extended BASIC ROM, each ROM being 8K long. A ROM's capacity is usually expressed in bits. In the CoCo, the data bus is eight bits wide. Therefore an 8K ROM has 8K times eight bits, giving you 64K bits. Starting to get the picture? Since there is only one place on the PCB for BASIC and Extended BASIC, a new chip with both 8K ROMs (or 64K bits) gives you 16K or, like the label says, 128K.

If you bought the computer without Extended BASIC, you got a socket and an 8K ROM in a 28-pin package. The jumpers are set to the 64K position. If you bought an Extended BASIC machine, you got a soldered-in 16K ROM with the jumpers set to the 128K side. In both cases you got a new version of BASIC, Version 1.3. If you have Extended BASIC, then you only see the Extended BASIC Version, 1.1. To see the BASIC version type in EXEC 41175.

To take this further, the two ROMs Tandy uses, 8K and 16K, are pin-for-pin compatible with two EPROM counterparts. The 28-pin BASIC ROM is compatible with the Intel 2764 EPROM. The 28-pin Extended BASIC ROM is compatible with the Intel 27128 EPROM. Now you can see where the 64K/128K numbers come from. If you have an EPROM programmer, modify these ROMs to suit yourself and plug them right in. Of course, if the ROM is soldered in, you will have to desolder it and put in a socket. Don't forget to change the jumpers to the right place. More on this later.

The RAM portion is quite impressive. There are three ways to add 64K to this CoCo 2 B. If you have 16K of RAM on the computer, chances are the chips Tandy used are two 4416 RAM chips. These chips are 16K by four bits each. Since the CoCo needs eight bits, there are only two of these chips. These chips are in the two 18-pin sockets between the two white connectors. The first way to upgrade this 16K computer is to change these two chips for the 64K counter part. The number to this is 4464. There are a lot of different numbers that are compatible with this chip. Just ask for a 4464, a 64K by four DRAM or an equivalent.

CN3		CN4	
Pin	Function	Pin	Function
1	GND	1	GND
2	+5V	2	A7
3	A4	3	A3
4	A5	4	A2
5	A6	5	A1
6	RAS	6	A0
7	WE	7	DQ6
8	DQ1	8	DQ5
9	DQ0	9	DQ7
10	DQ3	10	DQ4
11	DQ2	11	CAS
12	GND	12	GND

Figure 1

With the computer turned off, remove the two memory chips and install the two new ones. On the left side, there is a white box marked J6, jumper 6. You must solder a jumper across the two pins inside this box. This tells the software that there are 64K memory chips installed. That's all there is to it.

The next way to upgrade is using the two white connectors. These connectors consist of all the lines necessary to connect 64K of memory. A small PCB will be necessary. The pinouts to the connectors are in Figure 1.

There are two reasons why I'm not going into details on how to construct this piggyback board. The first is that it is available, fully assembled and tested, from CRC Inc. (514) 383-5293 for a modest price, and the other reason is that there is a third method of upgrading this CoCo 2 B.

If you have some 64K chips lying around gathering dust, you'll like the third way to upgrade. See all those holes filled with solder? Do you see the eight empty IC names soldermasked on the PCB? These eight blank areas are made for 64K memory chips. The regular run of the mill 4164s. All you have to do is add eight sockets and plug them right in. There is a small catch: The holes for these ICs are filled with solder. You must first empty the holes of their solder. You can use a device such as Radio Shack's desoldering pump (less than \$20). Just heat up the hole to be cleaned with a hot soldering iron. Then bring the desoldering pump to the hole. Remove the iron, press the pump to the hole and press the pump button. Go through all the holes of each pin. It would be wise to solder in sockets, not the chips directly. On some boards, the

eight decoupling capacitors are also missing. Insert eight .1 UF capacitors. As with any upgrade to 64K, don't forget to jumper the connections at J6. That's all there is to it.

There are a few more changes in the CoCo 2 B. Until now, all CoCos used the Motorola MC6847 as a display processor. This is the chip that gives the text on the screen and all of the graphics modes. Text on the screen has been green with black letters. When typing in lowercase letters, they would appear as inverted blocks of black with green letters. The new chip that Tandy uses on this CoCo 2 B is slightly different. It is an MC6847T1. (This chip might also have the part #XC80652P.) This chip is different. It has built-in real lowercase characters and you can also get rid of that border in certain cases. This is a real nifty improvement to the CoCo's display. The only problem with this is that Extended BASIC will not let you use these added features. Next month, I'll get my soldering iron out and add a few switches to change the default values.

The last change the good people at Tandy made was in the SAM (Synchronous Address Multiplexer). With all these changes to memory, video and circuitry, a new SAM chip is needed. It is the SN74LS785. A Motorola part that is upward compatible with the old SN74LS783 or the MC6883.

Back to the 28-pin ROM. Earlier, I mentioned that the ROM Tandy used is pin-for-pin compatible with an EPROM. A long time ago, a reader asked if there was a way to add a DOS chip inside the CoCo. Now there is. There are many ways to do this. Different people like to solder things together in different ways. I like the fastest and

easiest way. Some people like to make it neat. The chip you must use is either a 2764 or a 27128. All of the address lines, data lines and power lines are the same. The only line that is different will be the chip select line. We'll get that line from another chip. The chip enable line on an EPROM is pins 20 and 22. These are the pins that must connect to the extra enable. The thing to do is connect

all the pins except the two enable pins. Here is where some people differ. I used a 28-pin socket and soldered all the pins (except 20 and 22) to the 28-pin ROM. I bent pins 20 and 22 up and soldered them together, running a wire to Pin 12 of the 74LS138. That's the easy way.

Now, plug in the new EPROM and the cartridge area socket will be inside the CoCo. Some people don't like to

solder directly to a ROM. Use a wire wrap socket and solder a second socket to the legs about halfway down. Cut pins 20 and 22 from the top socket. Solder these two pins to the 74LS138 mentioned above. Plug the ROM into the lower socket and the EPROM into the upper socket. The same results happen, but it is neater. No soldered ROM, but it is also a little more trouble. Take your pick. ☺

Some Hardware Fixes for the Video Display Generator

Last month, I described in detail the innards of the new CoCo B series computer. One difference inside this computer is a new version of the VDG (Video Display Generator). I described it as being an improved version of the old faithful VDG that has been in the CoCo since the beginning.

To make the new VDG compatible with the old one, the new functions of this VDG are not readily accessible. For instance, this VDG has a built-in lowercase character set. But press the old SHIFT/0 and nothing happens. You still get that crummy inverse video lowercase character. So what gives?

Well, in order to get it to work, you may have to add in a little hardware. This is where I come in. Get out the old soldering iron and dig in as I lead you through the modifications to get the most out of your new 'B' series computer. Note: The letter 'B' must appear on the model number of the computer and not inside on the PCB. For instance, the one I have is model number 21-3134B.

Let's start with the basics. The old VDG chip number is Motorola MC6847. The new part is another Motorola part numbered MC6847T1, though in some computers, the part number might be XC80652P.

The first and most important change is the lowercase capability. Normally it is disabled, meaning you will not see the lowercase characters when using the SHIFT/0 on the keyboard. Instead, you get the normal inversed character set. You can change it in software. The pin that controls which mode you are in is connected to the PIA, which is memory mapped at \$FF20 to \$FF23, or 65312 to 65315 in decimal. It is connected to PB4 or Bit 4 of address location \$FF22 or 65314. This bit is normally a zero. Changing this to a one gives you real lowercase characters. The only problem is the routine in Extended BASIC will change it back to a zero every time you print something. If you want to do it in BASIC, add this line every time you want to change the screen to true lowercase:

```
10 POKE &HFF22 , (PEEK (&HFF22)  
OR 16)
```

What this line does is change Bit 4 to logical level one. But remember, each time you print on the screen or change

from graphics to text, Extended BASIC changes this back. You may want to make this line into a subroutine. Better yet, why don't you do it in hardware? It's more permanent.

There are many ways of doing this change in hardware. Use the one that suits you best, but the first way I present is the simplest. Remove the chip from the socket. Bend Pin 30 (GM0) out so that it does not plug back into the socket. Solder a short piece of wire from Pin 30 to Pin 17. Pin 17 is the 5-volt supply. This action permanently changes the level of the pin to logical level one, giving lowercase all the time.

If the VDG is soldered into the board without a socket, then just cut Pin 30 at the base and pry it up. Use slim-line cutters or a razor blade. Be careful not to cut anything else.

The second way to make the hardware change requires an SPDT switch. Figure 1 shows two ways of wiring the switch to this circuit. Using Figure 1a as a guide, pull Pin 30 out as described before. Solder a wire from Pin 30 to the center of the switch. Solder another wire from one side of the switch to Pin 17 of the VDG. Solder a third wire to the other side of the switch and to Pin 1 of the VDG.

When the switch is toward Pin 17, the

Tony DiStefano is well-known as an early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

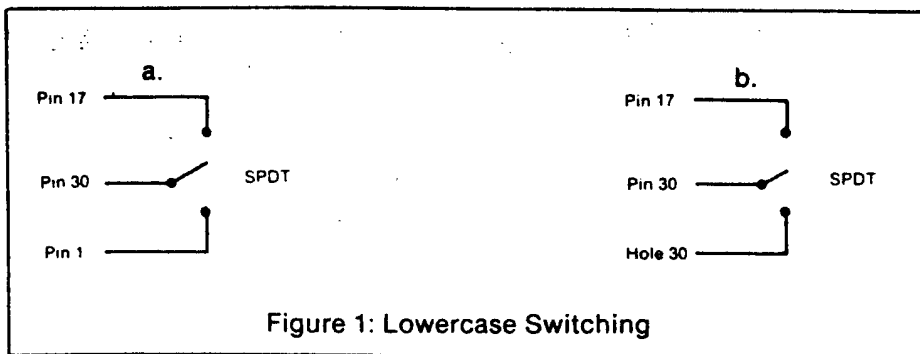


Figure 1: Lowercase Switching

display will always show lowercase characters. When it is the other way, it will always display inverse characters. Figure 1b shows basically the same way as before, but instead of connecting the third wire to Pin 1, connect it to the empty pinhole created when you pulled Pin 30. This way, when the switch is toward Pin 17, you always get lowercase characters. When the switch is the other way, you get whatever display Bit 4 of the PIA is set to. This is the most versatile way of connecting this pin.

The next change has to do with the border. In the normal text mode you see a big green square with black letters. This border is always black in the text mode. Now there is another alternative. How about a green border? There is a way of doing this in software. The pin that controls which mode you are in is connected to the PIA which is memory mapped at \$FF20 to \$FF23 or 65312 to 65315 in decimal. It is connected to PB6 or Bit 6 of address location \$FF22 or 65314. This bit is normally a zero. Changing this to a one gives a green border. The only problem is that the same routine in Extended BASIC that changes the lowercase pin every time you print something also changes this pin. If you want to do it in BASIC, add this line:

```
10 POKE &HFF22 , (PEEK (&HFF22)
OR 64)
```

What this line does is change Bit 6 to logical level one. If you want to change both the lowercase and the green border, change the last value to 80 (16 + 64). The new line to change both the lowercase and green border would look like this:

```
10 POKE &HFF22 , (PEEK (&HFF22)
OR 80)
```

But remember, every time you print on the screen or change from graphics to text, Extended BASIC changes this

back, so again, you may want to make this line into a subroutine. And again, this can be done in hardware.

One way to do this is to remove the chip from the socket. Bend Pin 27 out so that it does not plug back into the socket. Solder a short piece of wire from Pin 27 to Pin 17. This action permanently changes the pin to logical level one, giving a green screen all the time.

If the VDG is soldered into the board without a socket, cut Pin 27 at the base and pry it up.

The second way requires an SPDT switch. Figure 2 shows two ways of wiring the switch to this circuit. Pull Pin 27 out as described previously (see Figure 2a). Solder a wire from Pin 27 to the center of the switch. Solder another wire from one side of the switch to Pin 17 of the VDG. Solder a third wire to the other side of the switch and to Pin 1 of the VDG.

When the switch is toward Pin 17, the display will always have a green border; when it's the other way, it will always have a black border. Figure 2b is basically the same way as before, but instead of connecting the third wire to Pin 1, connect it to the empty pinhole created when Pin 27 was pulled. This way, when the switch is toward Pin 17, you always get a green border and when the switch is the other way, you get whatever display Bit 6 of the PIA is set to. This is also the most versatile way of connecting this pin.

The third modification is the famous inverse video screen. You no longer need to add a gate to do inverse video. The procedure is basically the same as the others, but with different values and different pin numbers. You can change it in software. The pin that controls which mode you are in is connected to PB5 or Bit 5 of address location \$FF22 or 65314. This bit is normally a zero. Changing it to a one gives you an inverse video screen. But remember, Extended BASIC will change it back. If you want to do it in BASIC, add this line every time you want to change to an inverse screen:

```
10 POKE &HFF22 , (PEEK (&HFF22)
OR 32)
```

This line changes Bit 5 to logical level one. To change both the lowercase and the inverse video, change the last value to 48 (32 + 16). The new line to change both the lowercase and inverse video looks like this:

```
10 POKE &HFF22 , (PEEK (&HFF22)
OR 48)
```

Since Extended BASIC will change this back, again you may want to make this line into a subroutine. Don't bother to add the green border value when using the inverse video — it has a lower priority and shuts off anyway. Again, you can do it in hardware.

To make the change in hardware, remove the chip from the socket and bend Pin 29 out. Solder a short piece of wire from Pin 29 to Pin 17. This permanently changes the pin to logical level one, giving inverted video all the time. (Pin 17 is the 5-volt supply.)

If the VDG is soldered into the board without a socket, then just cut Pin 29 at the base and pry it up.

The second way uses an SPDT switch. Figure 3 shows two ways of

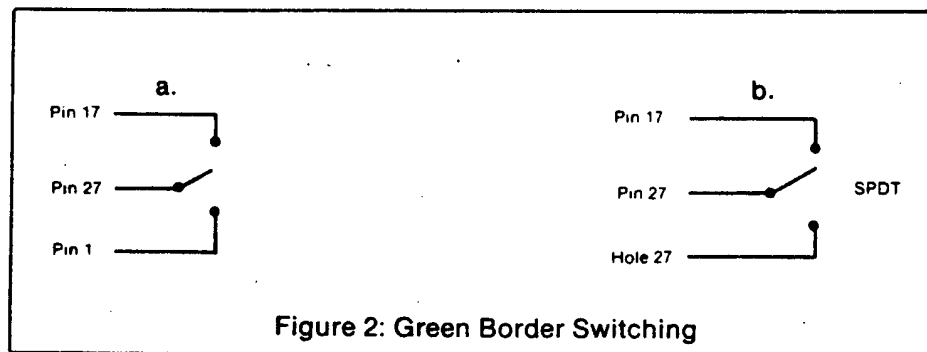


Figure 2: Green Border Switching

wiring the switch to this circuit. To use the first method (Figure 3a), pull Pin 29 out. Solder a wire from Pin 29 to the center of the switch. Solder another wire from one side of the switch to Pin 17 of the VDG. Solder a third wire to the other side of the switch and to Pin 1 of the VDG.

When the switch is toward Pin 17, the display will always have an inverse video; when it's the other way, it will always have a normal screen. The second method (Figure 3b) is much the same as the first. Instead of connecting

the third wire to Pin 1, connect it to the empty pinhole. When the switch is toward Pin 17, you always get inverted video; when the switch is the other way, you get whatever display Bit 5 of the PIA is set to.

These three changes to the new VDG add to the versatility of the CoCo's display. However, I suggest you wire the three pins using the SPDT switches and the empty hole left by each pin because, when in any graphics mode, these three pins are also used by the VDG to control which graphics mode you are in.

"You no longer need to add a gate to do inverse video."

If you hard wire the pins into a particular mode, you will lose certain graphics modes, depending on which pin you hard wired. If you use the most versatile way for each switch, all you have to do to return to the normal or default mode when you need a certain graphics mode is to throw a few switches.

Next month, I'll get into a step-by-step description of how to integrate the new MC6847T1 chip into your older non-'B' CoCos. I wonder just how many original CoCos are still out there? I would like to thank James R. Igou of Newark, Delaware for supplying me with the manual and an MC6847T1 chip to work with. I would also like to thank Bill Warnica of Barrie, Ontario, for his assistance with this and the next article on the new VDG chip. ☺

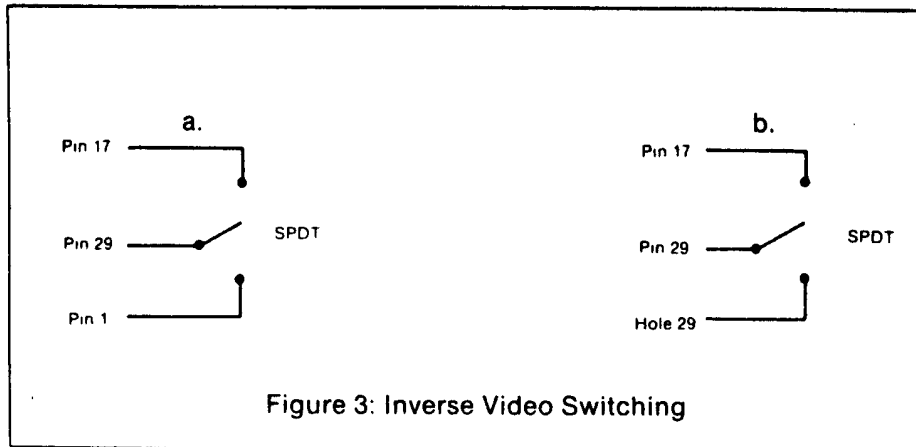


Figure 3: Inverse Video Switching

The No-Switch VDG

Two months ago I introduced you to the new VDG-T1. It came installed in the CoCo 2 B series. I guess I shouldn't say "new" any more; the CoCo 3 is out. I have ordered mine and, as you might well guess, I will turn the screw on it and see what is under the hood. I'm sure I will be able to come up with some hardware ideas on what we can do with this new CoCo 3. If any of you have a hardware idea for the new computer, jot it down and send it to "Turn of the Screw Wish List," care of RAINBOW at the Falsoft building. Meanwhile, back to the ol' CoCo 2.

The VDG-T1 has a lot of nifty changes to make it better. But these changes are not very accessible. The changes are hidden away deep inside CoCo. I wrote on how to dig these changes out so that you could make use

of them. These changes included lowercase letters, inverse screen and a colored border. I discussed a couple of ways to access these. One was in software and the other was in hardware. The software way is a pain at best. You have to insert a BASIC line every time you printed to the screen. If you have a machine language program, forget it, the software just will not work.

The hardware way is a little better. You have three switches and set them up the way you want. The only problem is that the switches interfere with the graphics modes. So when you use graphics, you have to switch the three switches back to their original position. Again, what a pain. If you have a program that switches back and forth between graphics and text, you either

have to play "flip the switches," or not bother with them at all and set them to their default settings. If that is the case, why put them in, in the first place? Don't despair, this month I'll show you how to eliminate the switches and still have the best of the VDG-T1.

First, let me review what the three pins and switches do. The three pins in question are pins 27, 29 and 30. The Motorola specifications manual for the MC6847T1 states that these pins are named GM2, GM1 and GM0 respectively. These three pins have dual purposes. There is another pin on this VDG known as the A/G Pin (Pin 35). This pin is an input. It controls whether the VDG is in Alpha/Numeric mode or in one of the many graphics modes. When this pin is logic state 0, or low, the VDG is in the Alpha/Numeric or text mode.

When the pin is high it is in the graphics mode. This is the dual mode that other three pins can go into. In the graphics mode (A/G = HI) these three pins tell the VDG what graphics mode you want. For example, you can be in the 128-by-64 pixel resolution mode or the 256-by-192 pixel resolution. Table 1 shows all the different graphics combinations available with this VDG.

In the text mode (A/G LO) the three pins in question control in which text mode the VDG will display the text characters. For example, true upper- and lowercase characters, inverse upper- and lowercase characters, green border or black border. Table 2 is a description of how the three pins affect the text display on your screen.

The three control pins and the A/G pin are all inputs. To control them, the CoCo uses four pins on a PIA (Peripheral Interface Adapter). By now we are all familiar with the idiosyncrasies of Color BASIC. It controls these pins according to the old 6847 VDG, not the T1 VDG. Two articles ago I told you how to use switches to get around this. In last month's article I showed you how to hook up the T1 VDG to an old CoCo. This time I'll use an electronic switch to do the same switching action. With this modification, you won't have to fiddle with switches. It's compatible with all software. This modification will work with any CoCo that has the new T1 chip installed. The main part you will need is a TTL logic chip, the number is SN74LS157. Unfortunately, it is not available at your local Radio Shack store, but you can get it at any good electronics shop.

This chip is a quad 2-to-1 data selector. For each of the four gates, there are two inputs (A and B) and one output (Y). It also has a control line. The way it works is that when the control line is low, the output Y is the same level as the A input and disregards what is at input B. When the control line is high, the output Y is the same level as the B input and disregards what is at input A. Can you see it coming? We can use this chip to control the three pins of the T1 VDG and use the A/G line to control the selector chip.

Look at the schematic in Figure 1. It shows the wiring to this modification. I disconnected the three output pins of the PIA (that connects to the VDG). These three pins now go to the B input on three of the four gates on the selector

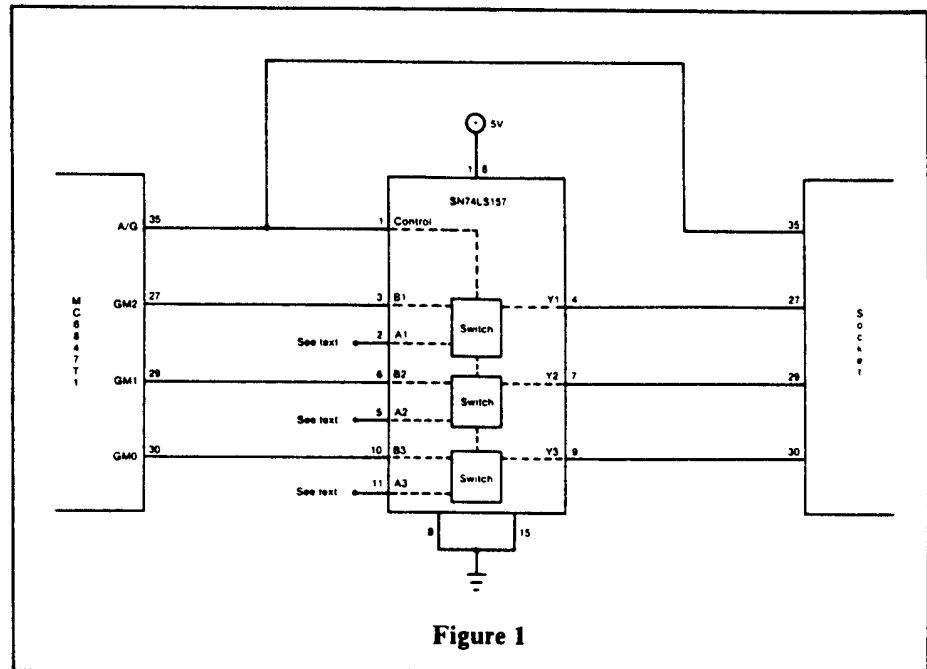


Figure 1

Table 1

GM0	GM1	GM2	Description
0	0	0	64 by 64 4 Color
0	0	1	128 by 64 2 Color
0	1	0	128 by 64 4 Color
0	1	1	128 by 96 2 Color
1	0	0	128 by 96 4 Color
1	0	1	128 by 192 2 Color
1	1	0	128 by 192 4 Color
1	1	1	256 by 192 2 Color

Table 2

Pin No.	Pin Name	Logic Level	Function
30	GM0	Low	Inverse Lowercase characters.
30	GM0	High	True Lowercase characters.
29	GM1	Low	Normal green screen.
29	GM1	High	Inverse black screen.
27	GM2	Low	Black border.
27	GM2	High	Colored border.

chip. The outputs of these three gates go to the VDG. The fourth gate is not used. I have also connected the control pin of the selector chip to the A/G pin of the VDG. When the CoCo is in the graphics mode, this pin is high. This makes the control pin on the selector chip high also. When the control pin is high, the Y output will follow the B input. Given that the PIA pins are connected to the

B inputs, when the control pin is high, it is as if the selector chip were not even there.

Now, when the CoCo is in text mode, the A/G pin is low. Since the control pin of the selector chip is connected to the A/G pin (in the text mode), the control pin is low. What happens when the control pin of our selector is low? The Y outputs follow the A inputs. What did

you connect the A inputs to? Well, that all depends on how you want the text mode to look. Each of the three pins does something different. For example, the pin that connects to Pin 30 controls true lowercase characters or inverse lowercase characters. Table 2 shows what each pin does.

When you have picked which mode you want, you have to connect the A input to match that mode. When that mode requires a low, you have to connect that A input to ground or Pin 8 on the selector chip. When that mode requires a high, you have to connect that A input to 5 volts or Pin 16 on the selector chip.

The construction of this modification

is typical of my projects. You need all the regular tools. Some people don't like to cut and solder directly on chips and PC boards. In that case you will need a 40-pin socket. I used a socket on this one. I hollowed out the center of it and used the space to put the selector chip in. If you don't want to use a socket, just piggy-back the selector chip on top of the VDG. If you use a socket, pry up pins 27, 29 and 30 so they do not go back into the socket. Use the empty pinhole as the connection to the PIA. If you don't use a socket, cut VDG pins 27, 29 and 30 and pry them up. Make sure you cut the pin high enough so you can solder to the stub that is left; it is the connection to the PIA. Either way,

make sure you don't cut A/G Pin 35. Even though we are using this pin, the VDG still needs it too. Remember, the Y outputs of the selector chip go to the VDG, and the B inputs come from the connection that used to go to the VDG.

If you made last month's mod that puts the new VDG into the older CoCo, make sure you don't break any wires. If you used a socket to do it, you can use the same socket in last month's project for this one. You save a socket.

Well, that's it. When all the connections are done, close up the computer and try it. Check to make sure that all the modes work and that the text mode is the mode you want. ☺

Taking a Look at How Monitors Work

Well, I finally got my CoCo 3. The first thing I needed to do was to plug it into a monitor. In my computer room I have various color monitors, TVs and monochrome monitors. I read through the CoCo 3 manual and found out it has three ways of connecting a display to it. The first and most common is the RF output. This is where you connect an ordinary TV to it. The second is a composite color output, sometimes known as a video output. The third is an RGB output.

Now, most people are familiar with the RF output. Many people know about video outputs, but what is this RGB stuff? It is not new to me because I use an RGB monitor for my other computers. With the right connector, a piece of ribbon wire and the right information, I connected the CoCo 3 to my Sony RGB monitor.

Ever since I wrote an article on how to connect your CoCo to a monochrome monitor, I have been getting calls about it. So, with the coming of the CoCo 3, it is time to do an "everything you ever wanted to know about monitors but were afraid to ask" article. Here it is.

I am going to start from the basics and work my way up to RGB. Let's begin with some theory on a monochrome monitor. The mono part of that word implies one color. At first, all picture tubes were white. Then green was the "in" color and then amber became popular. Whatever the actual color of the tube, it is still one color, hence monochrome. A picture tube is made of glass. Inside this tube is a vacuum. On the inside surface of the display area there is a thin coating of phosphorus. One physical property of phosphorus is that when bombarded with electrons (high voltage electricity) it glows. Inside the back end (neck) of a picture tube there are circuits that shoot electrons at the phosphorus. The construction of the tube is beyond this article, but when it is on, a stream of electrons hits the phosphorus, and where it hits, the phosphorus glows. But alone, all that does is make a glowing dot in the center of the screen. Not much good.

Since electrons are affected by a magnetic field, putting a magnet close to the tube will deflect our dot. The dot would move according to the strength

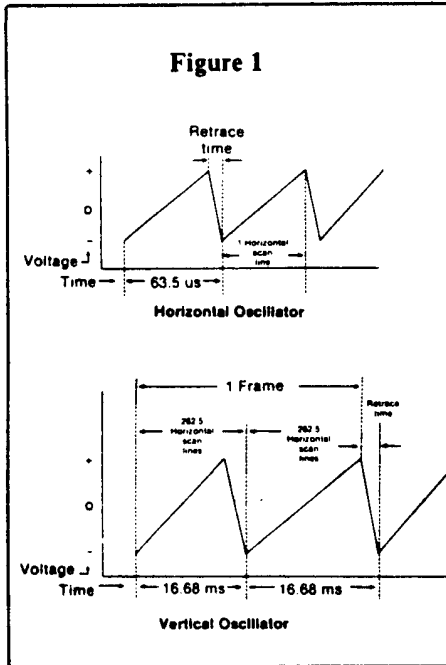
and direction of our magnetic field. An electric current through a wire creates a magnetic field. The more current, the stronger the magnetic field. A length of wire wrapped in a coil is enough to deflect our dot anywhere on the screen. In most monitors, two coils of wire wrapped around the neck of the tube are used to move our dot around. One coil is positioned so that a varying amount of current makes the dot move sideways or horizontally. The other is positioned to give up/down or vertical motion.

Given the right amount of current and in the proper sequence, our dot now moves from right to left and from top to bottom, in the same motion as reading. Make that dot move fast enough and it appears to fill the screen with light, since phosphorus continues to glow for a short time after the dot has moved. Those lines you see on your screen are made by one moving dot.

So far, we have one moving dot that fills the screen with light. If, while moving this dot, you were to increase and decrease the number of electrons hitting the phosphorus, you would get varying amounts of light. The amount of light produced is directly proportional to the number of electrons hitting

the phosphorus.

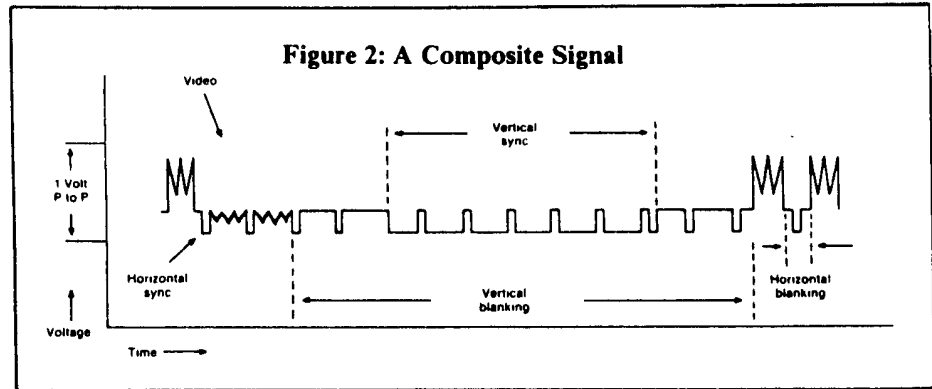
Things are shaping up to a picture. In a TV monitor, there are many signals and currents, one of which is called the horizontal oscillator. This circuit is connected to the coil that deflects the dot horizontally. Figure 1 shows the wave shape of the horizontal oscillator. It starts off negatively, deflecting the dot to the left. It increases linearly to a positive position, moving the dot smoothly across the screen. Then, it



quickly jumps back to the original position. During this time the electron flow is cut off so that it will not appear on the screen. This time period is known as the retrace time, and the circuit that cuts off the electron flow is called a blanking circuit.

Another circuit in a TV is the vertical oscillator and yes, you guessed it, it controls the dot vertically. The wave shape of the vertical oscillator is basically the same as the horizontal one, only much slower. Many horizontal cycles fit inside one vertical cycle (more on this later). The vertical oscillator also has retrace time and vertical blanking circuits. Due to its nature, one horizontal cycle is called a scan line, and one vertical cycle is called a frame.

When our dot is not doing horizontal retrace or vertical retrace it appears on the screen. This is known as active video. It is during this time that our dot gives the viewer useful information. This information can be a picture like ordinary TV, or computer generated characters. In either case, the video



signal is proportional to the brightness of the picture. A higher signal produces a brighter dot and a lower signal produces a softer dot.

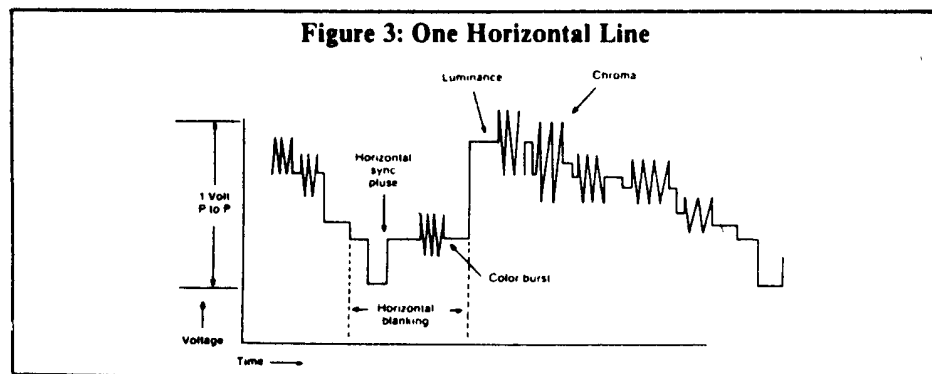
In order for a picture to appear on a video monitor, three signals are needed; horizontal, vertical and video signals. It is not efficient to run three signals and a ground return to a TV receiver or monitor. A method was developed to combine these three signals into one. Instead of supplying complete horizontal and vertical wave shapes, the source need only send a pulse signifying the start of every horizontal line and the start of every vertical frame. These pulses are known as sync pulses. The rest of the wave shape is then regenerated inside the monitor. It is then up to the monitor to make sure that the internal horizontal and vertical oscillators keep up with the sync pulses.

These sync pulses and video signals are mixed together in a specific way to form one signal called "composite video," for obvious reasons. Figure 2 shows part of a composite signal. In North America, all composite video conforms to the NTSC (National Television Systems Committee) standard; more on that later. In a monitor, circuits are made to separate the video information from the sync signals, and are then translated to drive currents that connect to the coils and the picture tube.

Up till now, I've been talking about monochrome (black and white) pictures. But, there is a good reason why Tandy calls our CoCo a Color Computer — it can display a color picture. When TV first came out, it was only in black and white. When color came out, a method had to be developed so that a color signal would be compatible with a black and white TV.

It was up to the NTSC to develop a composite signal that would carry the extra color signal and still be compatible with the older black and white signal. In 1953, the NTSC established the color television standards. In these standards, the signal is to have 525 line interlaced scan. The horizontal scan frequency is 15.734 kHz; the vertical frequency is 59.94 Hz. The color information is contained in a 3.579545 MHz subcarrier. The phase angle of the subcarrier represents the color, and the amplitude of the carrier represents the saturation. Figure 3 shows one horizontal line. Notice the color frequency burst just after the horizontal sync. The phase difference between this reference burst and the actual signal describes what color that particular part of the screen should be. The amplitude of the color signal represents how much of that color to put on the screen.

On the monitor side, color is quite complicated to reproduce. You have to



start with a completely new tube. Instead of smooth monochrome phosphorus, the tube has to be striped with alternate red, green and blue phosphorus. The smaller the stripe the better the picture quality.

When a color composite video signal enters a color monitor, it is first stripped of its sync signals, and then the monochrome (called luminance) and color signals (called chroma) are split into three signals, the red content, the green content and the blue content. Three separate electron beams are used to display the three colors on the screen. The beam carrying the red content has to hit all the red strips. The green hits the green strips and so on. If a beam that has red information hits any other color than red, a wrong color results. It requires a lot of electronic circuits to keep this from happening. That is not the worst part; the color frequency carrier is 3.58 MHz. In order to isolate the color carrier from the monochrome signal, a filter is used that removes any frequency higher than 3.58 MHz. This seriously limits the resolution of a color signal. In fact, the resolution of a color

signal, at absolute best, is about 400 lines. That is OK for the CoCo and CoCo 2 but is not good enough for the CoCo 3.

When you put a color signal in a monochrome monitor, the color information shows up as dots on the screen. Figure 3 shows that. The frequency of the color signal is 3.58 MHz. A monochrome monitor with a 20-MHz bandwidth has no filter to remove the color carrier. The monitor will have no problems displaying the color carrier — as an annoying monochrome mess of dots.

Now comes the CoCo 3. It has a resolution of 640-by-192. That is very nice but have you ever seen a 640-by-192 screen on a regular composite monitor? Believe me, it's not a pretty sight. What is Tandy to do? The only reasonable thing is to get rid of that color carrier and put out the color information separately. Now that is a great idea and for once Tandy did it right! The CoCo 3 has an output known as an RGB output. That's right, RGB stands for Red, Green and Blue. No color carrier, no filters and no sync pulses, just clean color.

Wait a minute, that won't work without sync pulses. So Tandy added some more lines and added sync pulses. In fact, the CoCo 3's RGB output is the best color picture ever for a CoCo! The clarity is limited only by the resolution of the monitor.

You don't have to have a Tandy RGB monitor. However, if you plan on going out to buy a brand X monitor at some discount mail order house, here are a few tips to help you get started with fewer headaches.

First, when you select a model you want (or can afford), make sure that it is an RGB analog monitor with negative or composite sync (like my Sony) with a horizontal frequency of 15.7 kHz and a vertical frequency of 60 Hz. Also make sure that the bare connector to the monitor is available. You will also need a connector for the CoCo 3 side of it. That requires a 10-pin female socket connector for flat ribbon cable. And don't forget to get three or four feet of 10-conductor flat ribbon cable. Use the pinout supplied in the CoCo 3 manual and match the pinout of the RGB monitor manual to it. Now plug it in and watch it go! ☺

The CoCo Is Music to the Ears

A long time ago, I did an article on an analog-to-digital converter. I explained that you can take a varying signal and convert it into a digital value from 0 to 255. This time I'll do the opposite.

This month's project is called a D-to-A converter, where a digital value from 0 to 255 is converted into an analog voltage. But that is just part of it. I'll show you how to make two of these things. With two of these and some software, we will be able to make music in stereo. Our scenario starts by making two D-to-A converters. Then, with a couple of preamps, some connectors, a stereo system and some software you'll be playing computer music. We'll start today with the D-to-A converters and finish up next month with the preamp and some music software.

You can buy a complete, two-channel D-to-A converter chip, but they are a little expensive and most require three voltages. This is a problem with the one-voltage CoCo 2 and 3. Besides, it's more fun building your own. Now, let's get into some theory on D-to-A converters.

Remember that a digital value from 0 to 255 is made up of eight binary bits. Each of these bits has a value of 0 (ground) or 1 (5 volts). If you use every combination of eight bits, you come up with 255.

Let's introduce another component: a resistor. Yes, the good ol' resistor. If you put a voltage between the two points of a resistor, you could measure the voltage across it. If you put two resistors in series (Figure 1) and measured the voltage across both resistors, you would get the voltage that you put in. For instance, in Figure 1, if you put 5 volts across both resistors, you would measure 5 volts. If you measured across just one resistor, you would get a value somewhat less than 5 volts. If you measured the voltage across the second resistor and added that value to the value of the first, you would get 5 volts. The voltage is divided between the two resistors. If you had three resistors, the sum of the voltages of the three would add up to the total voltage applied. It is a simple mathematical equation and it depends on the resistance value

of the resistor. In a resistor circuit, the higher the resistor value, the higher the voltage across it.

If we had 255 different resistors hooked up to a voltage and were able to control which resistor had the voltage on it, we would have an acceptable D-to-A converter. But I'm sure you don't want to hook 255 resistors to some circuit. Well, you don't have to. All you need is nine resistors: eight for the eight data bits and one used as a voltage reference or source. It is used as a divider. This is commonly known as a resistor ladder.

If we use that theory, plus a bit of computer theory, we can convert a digital binary value of eight 0- and 5-volt levels to an analog level. A computer's data bus is continually changing as the computer does its thing. In order to isolate an eight-bit value, a latch is needed. The easiest place to add a latch is on the cartridge port. So, get out the tools and let's get started.

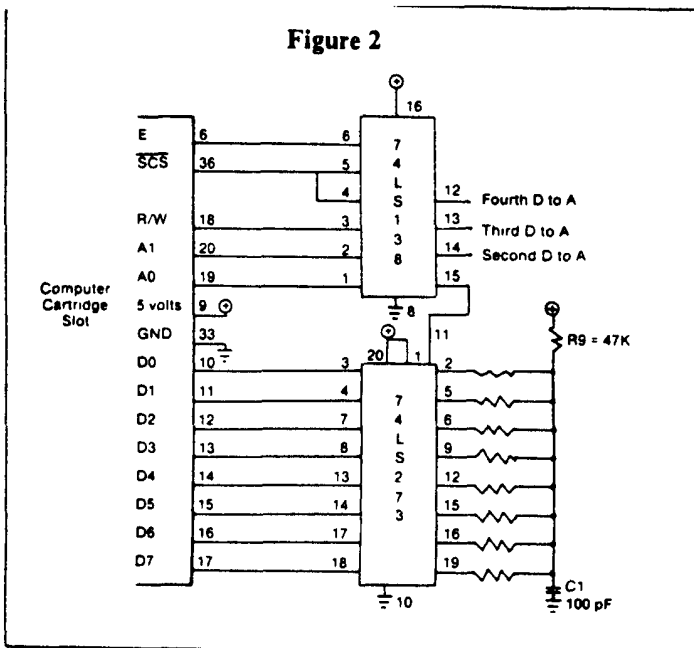
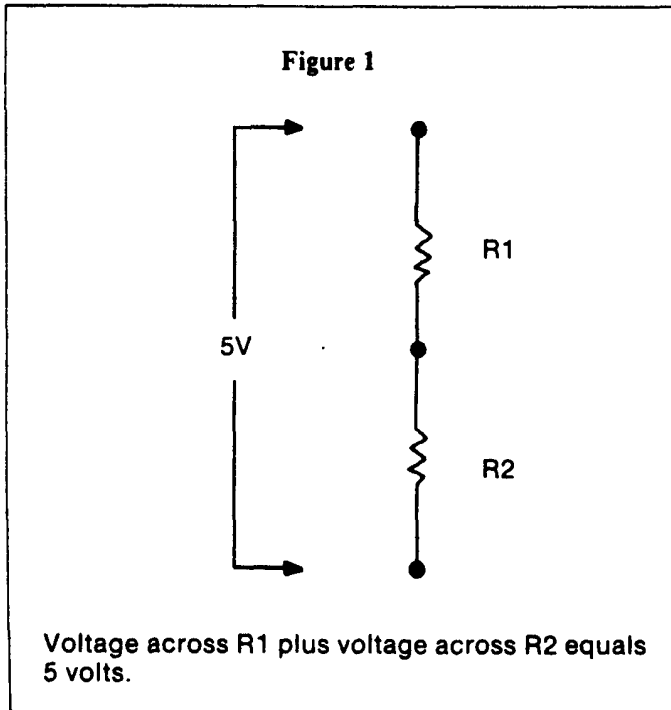
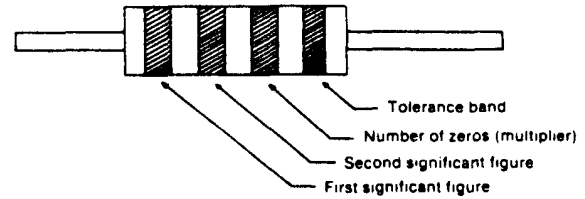


Figure 3: Standard Resistor Color Code



Color	Significant figure	Multiplying value
Black	0	1
Brown	1	10
Red	2	100
Orange	3	1,000
Yellow	4	10,000
Green	5	100,000
Blue	6	1,000,000
Violet	7	10,000,000
Gray	8	100,000,000
White	9	1,000,000,000
Gold	± 5% tolerance	
Silver	± 10% tolerance	
No color	± 20% tolerance	
Red	± 2% tolerance	

You will need all the usual things for a project. A protoboard, sockets, wire and a few parts. The first two parts are not that hard to find. A good electronics hobby shop will have them. They are a 74LS138 and a 74LS273.

You will see the circuit and how to hook it up to the CoCo bus in Figure 2. If you want stereo or two channels, you will need another 74LS273 and another nine resistors and capacitor. In fact, this circuit can have as many as four channels of D-to-A. All are identical to the one in this diagram except where Pin 11 connects to the 74LS138. Also, nine resistors are connected to each 74LS273. The diagram shows how to connect the other three circuits. The output of this D-to-A converter is about .1 volts on the low end and about 4.9 volts on the high end. The capacitor is used for high-frequency roll-off and to dampen switching noise.

So far, there haven't been any problems, but notice that I haven't given any resistor values. This is where the tricky part comes. The resistor value for R9 is simple: 47K ohms, half-watt or quarter-watt. But the other resistors are a different story. In theory, the value for each resistor is double the previous value. For example, if the first resistor value is 1K ohms, the next value must be 2K and so on. Using this method, the values are:

- | | |
|--------------|----------------|
| R1 = 1K ohms | R5 = 16K ohms |
| R2 = 2K ohms | R6 = 32K ohms |
| R3 = 4K ohms | R7 = 64K ohms |
| R4 = 8K ohms | R8 = 128K ohms |

That is fine in theory, but try to find these values in any store! It is next to impossible, but don't despair; you can get these values by using more than one resistor for each value. For instance, a 4K resistor does not exist (unless you want to custom-order it in quantities of 10,000). But, if you put two 2K resistors in series with each other, you get 4K. You

see, resistors in series add up in value. A 10K resistor in series with a 22K resistor gives you 32K. Now, the trick is to find the right combination of resistors, to match the values above. Some may require only one or two resistors, but other values will require as many as four or five resistors to add up to the right value. It all depends on what value resistors your dealer carries.

To make matters worse, the precision of the resistors has to be high. The ideal resistor must have a tolerance of .1 percent. Again, these are expensive and rare. If you are like me, you have a resistor bin. I went through the bin with an ohmmeter and measured the values and took the closest value. If you are not sure how to read the value of a resistor, Figure 3 shows a resistor color code chart and how to read it. The first and second colors are the numeric value and the third is a multiplier. For example, if you have a resistor that has a color code of red, violet and orange, its value is 27,000 ohms or 27K. Some resistor values are just not made. Here is a table of resistors that I found and used for my D-to-A circuit.

R1 = 1K	R5 = 15K + 1K
R2 = 2K	R6 = 22K + 10K
R3 = 2K + 2K	R7 = 27K + 27K + 10K
R4 = 6.8K + 1.2K	R8 = 100K + 27K + 1K

Again, it is important to have the right values. If you don't have the right values, keep adding more resistors until you do; they aren't expensive. Even after you get the right theoretical values, use a precise ohmmeter to fine-tune these resistors. Remember, the closer the values you use, the better the sound it will make. If your resistors are not perfect, at best, you will get a little harmonic distortion; at worst, you will get a bad sound.

As far as the parts are concerned, you can get the protoboard and the ICs from CRC Inc., 10802 Lajeunesse, Montreal, Quebec, Canada H3L 2E8. The resistors you will have to dig up yourself.

See you next month.



Transistor Buffers for Stereo Amplification

Last time, I showed you how to wire digital-to-analog converters. A D-to-A converter is a device that, when hooked up to a computer, converts (or changes) a digital value, or number into an analog voltage. In the case of the CoCo, the digital value is from 0 to 255, represented as an eight-bit binary value. Remember binary? Anyway, this eight-bit binary value is converted into a voltage. The voltage output is directly proportional to the input value. The lowest possible digital value (0) gives the lowest output voltage, 0 volts. The highest digital value (255) gives the highest voltage. In this case it should be about 5 volts.

This time, I'll show you how to hook up a couple of small amplifiers and get some sound out of them. If you recall, the outputs of the D-to-A converters are the sum of several resistors. This has an output of about 0 to 5 volts. If you want to connect this output to an external amplifier, such as a stereo system, then you don't need an amplifier but just a buffer. The reason you don't need an amplifier is the output voltage is high

enough to drive a stereo. In fact, it is a bit too high. The typical input voltage of a "line in" on a stereo is about 1 volt. It needs to be brought down a little. Figure 1 shows one transistor buffer. It is an emitter-follower. It has a lot of current gain but no voltage gain. This is what we need. R2 in the circuit is used to lower the voltage to a usable level for the stereo. V1 in the circuit is used as a volume control. If you only build one D-to-A converter you only need one circuit. If you build two D-to-As then you need two circuits. But instead of using two volume controls and adjusting them separately, use a stereo volume control that has two potentiometers built into one.

If your stereo is too far away or you don't have a stereo, then you may want to build a small amp to drive some speakers. Figure 2 shows a circuit that does just that. It is an amplifier module that has just a few milliwatts. In fact it has 325 milliwatts, just right for a small speaker.

All of the parts are available at your local Radio Shack store. If you used the

CRC Project board, then there should be enough room left on the board to mount all of the parts. If you want to use a socket for the IC, then use an 8-pin socket. There is no special care needed in the construction of the amp, except the usual care in dealing with parts that can be damaged by static electricity. The usual project tools will be necessary; things like a soldering iron, pliers, cutters and a drill to mount the variable resistor. Hook up the circuit as in the diagram. The capacitor C2 should be as close to the IC as possible; It's a power supply decoupling cap, so the closer the better. J1 is just a 4-pin connector so if you want to disconnect the speakers, you won't have to unsolder the thing every time.

The way the outputs are connected now, the signal coming from the D-to-A is very square. That is to say, it is very fast to change from one analog state to another. This tends to make the music very rich in harmonics, sometimes to the point that it may sound like distortion. C5 in the circuit acts as a low pass filter by shorting out high frequencies to

ground. If you like the rich sounds of harmonics, leave out C5. Otherwise a value from .1 uf to 1 uf will soften these harmonics. Try several values and use the one that you like best.

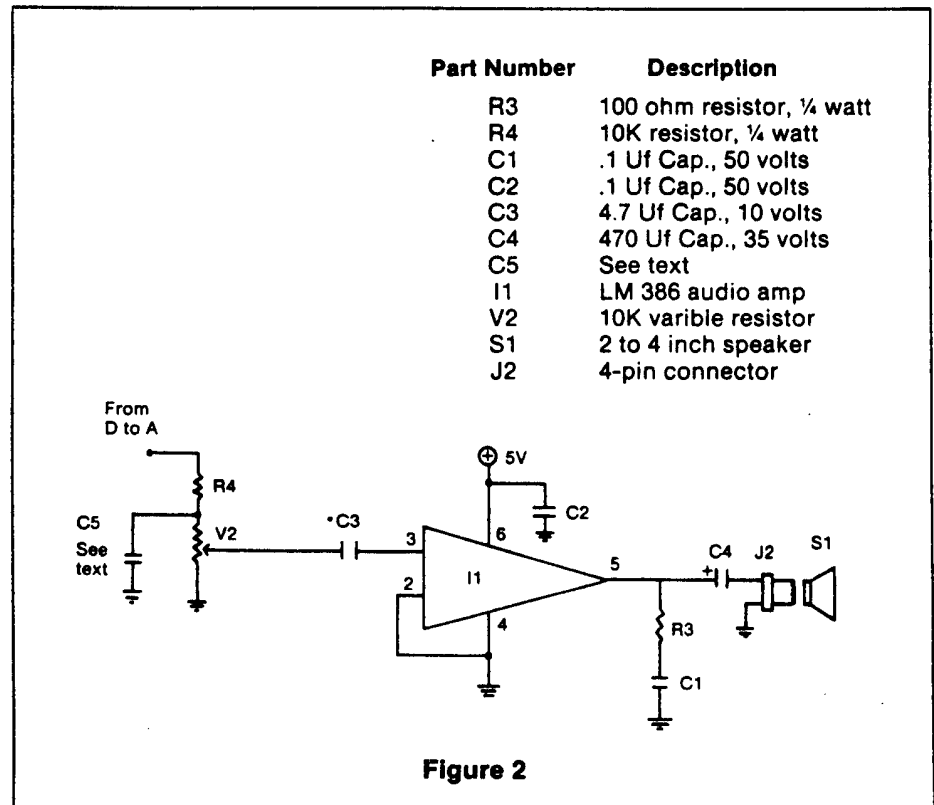
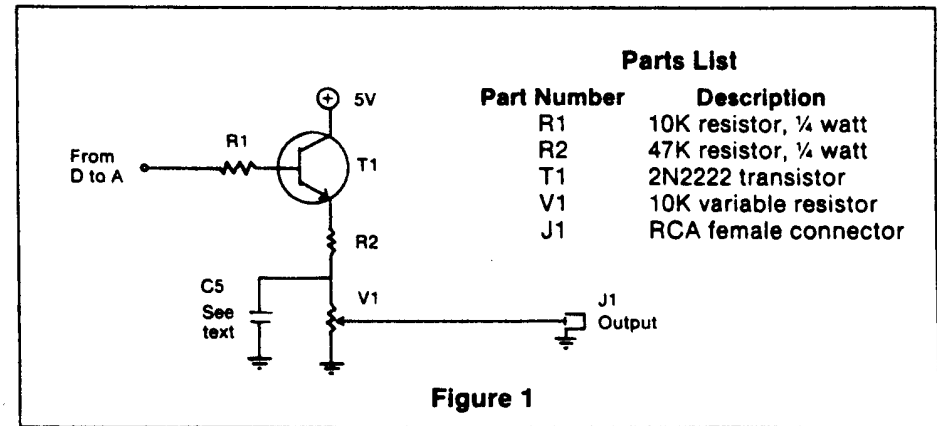
Now for the hard part. I say hard because for a hardware buff like me, software is a pain. But, hardware without software is not much good, so I have to deal with it. I looked around to see what I had in terms of musical software. After running through my old RAINBOWS, I found that the machine language routines used to generate four voices did not have listings, but only pages and pages of DATA statements. This makes it hard to find the driver routines and change them.

So I decided to give basic guidelines on how to modify them yourself. Inside the CoCo there is a built-in D-to-A converter. It is located at \$FF20 or 65312 in decimal. The D-to-A converter you have just built is at \$FF40 or 65344, and if you built two D-to-As, the second one is at \$FF41 or 65345. The idea is to find the location in memory that matches the address \$FF20 and change it to \$FF40. One thing to remember is that the address \$FF40 is divided into two bytes, since the CoCo can only work with eight bits of information. The first is \$FF or 255 in decimal and the second is \$20 or 32 in decimal. I wrote a short BASIC program to locate any presence of the address \$FF20 and change it to \$FF40. This is the program:

```
10 FOR I = &H1500 TO &H7EFF
20 IF PEEK(I)=255 AND PEEK
(I+1)=32 THEN POKE I+1,&H40
30 NEXT I
```

There are a few things to remember with this program. First, PCLEAR 1 before typing it in. Then, load in your music driver and music and run the program. The memory area covered by this program starts just after the BASIC program and runs to the top of a 32K machine. This is only a guideline on how to find the memory locations; people with good machine language skills will be able to find it with no problems.

After looking through my disks of software, I found that I had the program *Musica2*. I checked the machine language driver and found the point at which the program referenced address was \$FF20. I changed it by typing this



statement:

```
POKE &H3F79 ,&H40
```

That redirected the output to the external D-to-A I built. It was great. If you are using a multipack, you must do another poke to change the access of the slot that the controller is in, to the slot that the D-to-A is in. There is a simple way of doing that:

```
POKE &HFF7F , ((X-1) * 16)
+(Y-1)
```

Where X is the slot number that the controller is in (a number from 1 to 4), and Y is the slot number that the D-to-A converter is in.

To make sure that two D-to-A converters work, I built two of them. I took my machine language disassembler and looked at how the program worked. After a short time, I came up with a stereo version. These are the pokes I did to convert the *Musica2* Play program to use my stereo D-to-A converter:

```
POKE &H3F6F ,&HE6
POKE &H3F73 ,&HEB
POKE &H3F77 ,&HFD
POKE &H3F79 ,&H40
SAVEM"MU2ST",&H3F00,&H3FBF,
&H3F00
```

This will make the modifications necessary to run it on my D-to-As and save a copy of it to disk.

The Hardware Project Basics Review

By Tony DiStefano
Rainbow Contributing Editor

Recently it was brought to my attention that some readers of my column seem to be having a bit of trouble constructing the projects. When I bought my first CoCo, the first thing I did when I got it home was take the cover off to see what made it tick. I spent many hours inside the CoCo and many hours reading technical manuals on how the different components of a computer work. I also spent a lot of time with a soldering iron, soldering things together and taking them apart again. In short, I have a lot of experience with the CoCo and computers.

A lot of people use the CoCo as a means to an end. They are not hardware hackers. Other people are hobbyists who want to dig in and learn all about the insides of the computer. Whatever background you have with computers, or whatever electronics experience you have, I am sure you will agree that digging inside a computer requires a little skill and a lot of patience. Here are a few hints and tips on how to successfully complete a project.

The first and most important factor to consider is the project itself. Ask yourself, will it suit my needs? Is it within my budget? Will I learn anything from it? If all the answers are yes, then you can proceed. Once you have decided to tackle the project, you must learn all you can about it before you begin. It is important to read the whole article before you start. Make sure you understand the object of the project and the skills required. If there are parts you don't understand, study them over and

over. Refer to other articles of the same nature or reference books that touch on that subject. Technical data books are readily available from your local electronics distributor. Texas Instruments and Motorola have excellent books on understanding microprocessors, as well as technical data manuals. Never start a project if you are not sure you can finish it or if you do not fully understand the whole project, start to finish.

Once you have read the article and are confident you can finish it, the next step is parts. Make a list of all the parts you will need. Then do your shopping. If you are missing a part or two, don't start the project. Not only is it frustrating to have to stop because you are missing a part, but you lose the momentum of the project. Returning to a project after a while may cause you to skip a step. You may even lose interest and give up the project all together. So don't start until all the parts are in.

Another point you must consider is whether you have all the tools you need. Projects that require you to build some kind of hardware gizmo require tools. Make sure you have all the tools necessary to build your project. If mounting parts requires you to drill a hole, make sure you have the right size drill bit and that it's sharp. A sharp bit (or whatever the tool) is easier to work with; you don't have to force a sharp bit into a sheet of metal. A dull bit will drift before it cuts and may cause damage to the project and to you.

When you are about to start a project, take time to clear a good place to work. The work area must be free for the entire time it takes to do the project. If it is going to take more than one day to do, make sure that you will not need the work place for something else. It is too

easy to lose or break something if you have to put the computer and all the loose parts aside to repair the toaster. This way, you can continue exactly where you left off. If the project has a complex circuit or a lot of instructions, it may be wise to photocopy the article and staple the article together. This way you don't have to worry about getting your magazine dirty, torn or worn.

OK, you have all the parts, a clean workplace and all the tools necessary to do the project. Take things one step at a time. Build, cut, solder and do whatever is required of the project. At this time, it may be wise to consider the well-being of your computer. Some projects may require you to cut, bend, or modify the computer's PCB (Printed Circuit Board). Now, cutting a PCB may be quite permanent. Before you cut, look around to make sure you will not accidentally cut other things. Also, make your cut in a place that is accessible enough that you can rewire that cut, if you need to.

Removing ICs is a tricky task. There are more versions of the CoCo than there are versions of *PacMan*, and every one of them is different. On the older CoCo 1s, almost all the chips are socketed. The contrary is true for the newer CoCo 3s; almost all the chips are soldered directly to the PCB.

Special consideration must be given to this situation. If the project requires that you lift, bend or cut a pin on an IC that is soldered in, you have two choices. Both of them are a pain. The first choice is to cut the pin and solder it to the stub. If you don't do it right, the pin will break off. Also be careful not to short-circuit to the next pin.

The other way is a bit longer. It requires you to completely remove the

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

chip, which is a very hard procedure to follow. A lot of soldering experience is required. Most people use a soldering iron and some solder wick. Heat each pin (one at a time) until the solder flows. Remove the iron, lay the wick onto the pin and press the iron on top. The wick will draw the solder and leave the pin and hole empty. Do each pin one at a time. Then remove the chip and clean the pins with the same stuff. Now, insert a socket where the chip was and solder it into place. Remember to position the socket so that Pin 1 of the socket matches Pin 1 of the chip.

When at all possible, if the instructions of the project require you to solder to an IC or bend the pin of an IC, use a socket instead. Remove the IC in question, insert the IC into a good quality socket and treat the pins of the socket like the pins of the IC. This way, if you break a pin off, all you have to do is replace the socket and not the chip.

When you are finished, don't jump right in and try it. Take time to clean up your bench, put your tools away and check your work. Go over the project step by step again. Make sure all wires are properly soldered, components are all in their proper places, and there are no short circuits on your project. Before you plug in your project, blow the dust and shavings out of the computer. If you don't have anything to blow with, hold the computer or project upside-down and gently tap the bottom of the computer. This will dislodge any bits of drill shavings or wire that may have fallen into the computer. Now put the thing together and try it out.

Plug all connectors and wires in and connect the power to the computer. Turn the TV or monitor on first; do not connect anything that is not necessary for the operation of the project you are testing. For instance, if you don't need to have the disk controller plugged in, leave it out. Then if you do have a problem, you will at least have saved your controller from abuse. Turn the

"If you are certain your work is good, but it still doesn't function, look at the circuit diagram."

computer on and watch the monitor. The familiar CoCo screen should appear. Pay close attention to the monitor; if you suspect something is wrong, turn the computer off right away.

Don't panic! Go over your work step by step. Check all your connections. Check for chips that have been installed backward, or transistors or diodes that have been reversed. Depending on the type of solder you use, a deposit will be left behind after you solder. This can act as a conductor at high CPU clock speeds. It must be cleaned off. Use a recommended flux remover. Remember, some of the stronger flux remover can melt plastic. Don't clean it on your work table. Go outside or to the sink in the laundry room.

If you are certain your work is good,

but it still doesn't function, look at the circuit diagram. There may be a way to check only a section of the project at a time. Fall back on the theory of the circuit, and if you have some test equipment, check for proper voltages on the power supply and ground. If all else fails, restore the computer to what it was before you started and make sure it still works. Once you know the computer works, check the project carefully and try it again.

A few articles ago, I showed you how to interface the MC6847T1 chip into the older CoCos. One of the chips required rewiring. I had the diagram for the computers I had, but not all of the computers are the same. A couple of readers sent me the pinouts to this chip. I have not tried these out, but I am passing them along to you anyway. You are on your own to verify whether they work.

The first is from John A. Lind of Corona, Calif., whom I met at the Color Expo 87. The computer in question is the CoCo 2, Catalog No. 26-3127. The board markings are 8709416 Rev B DWG NO. 1700235. The jumpers for the U5 (74LS273) socket are as follows:

FROM	TO
3	15
4	12
7	6
8	9
13	6
14	2
17	19
18	16

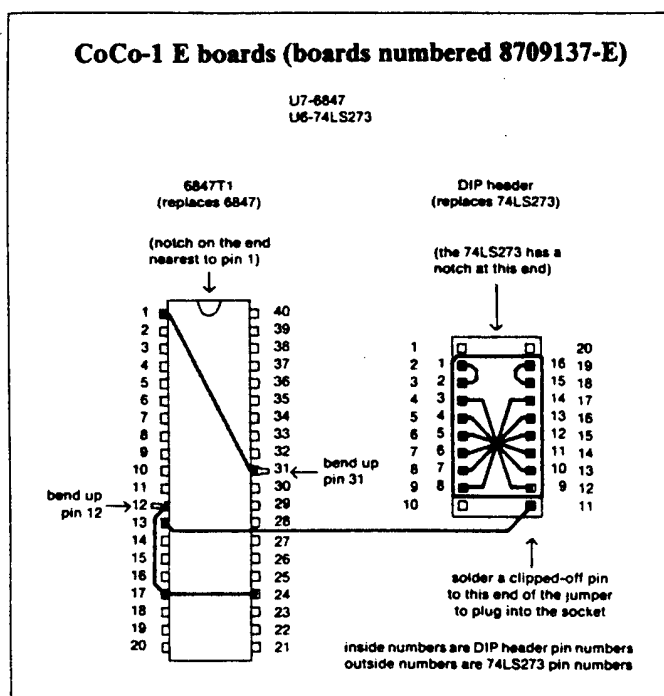
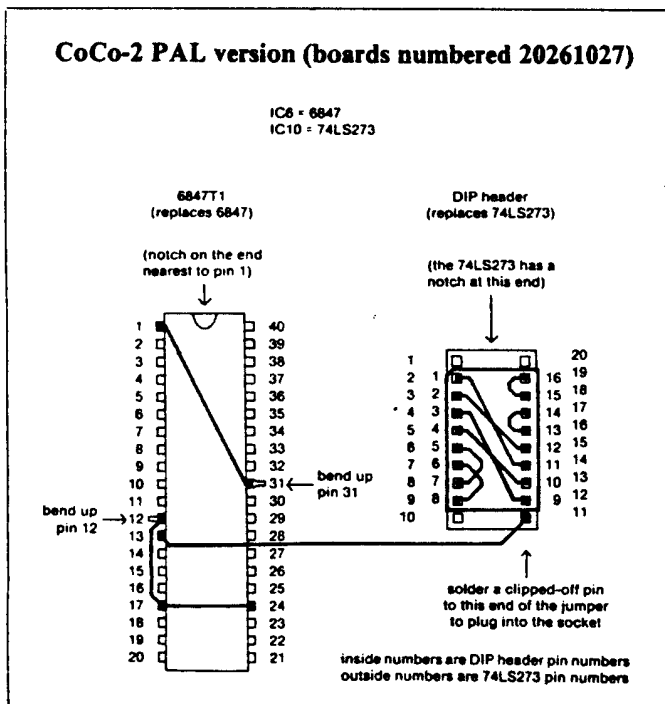


Figure 1 shows Mr. Fox's diagram on how to wire the CoCo 1 'E' board. Figure 2 shows how to wire the CoCo 2 PAL version. Thank you for these diagrams and information; if I get more, I will pass them on to RAINBOW readers.

Added notes: Pins 10, 11, and 20 are used for clock, +5V and GND. No jumper to the SAM chip is necessary.

The second diagram is from Mr. Ralph Fox of New

Zealand. Mr. Fox wrote: "Although your column did not mention it, it is also necessary to bend up Pin 12 of the 6847T1 VDG as well as Pin 31. The reason is Pin 12 is connected to Pin 37 (FS) on the CoCo's main circuit board; so if you jumper Pin 12 of the new VDG to Pin 10 (\overline{WE}) of the SAM, you will get contention between the VDG's \overline{FS} output and the SAM's \overline{WE} output."

An Expandable Relay Project

About two years ago, I wrote an article called "Lights, Camera, CoCo!" [Dec. '84]. It describes how to hook up as many as eight lights to the CoCo and have the computer control the on and off of each light. Ever since then, I have been getting letters about it. Some of the letters ask how to add more lights to the system, and other letters ask how to connect relays and other devices to the circuit.

Well, this article will answer a whole lot of letters with a project that is similar to "Lights, Camera, CoCo!," but more expandable. The idea is to be able to put many relays online to the computer and to be able to tell if the relay is on or off.

The heart of the circuit is a TTL (Transistor-Transistor-Logic) logic gate. I have talked about and used TTL logic gates ever since I began writing articles, so they should not be new to you. I have also used this particular chip many times before. The chip is a 74LS138. Ah yes, the good ol' 138. It is a decoder — a three-input to eight-output decoder, with three control lines. Remember binary counting? If we have a three-bit number, it represents eight separate digits, 0 to 7. If we connect these three input bits to the lower three address lines of the CPU, then the CPU can access eight address locations.

Study the pinout of the 74LS138 in Figure 1. Notice that the three inputs are connected to three address lines of the CPU. That determines the eight address locations to be used. The CPU in the CoCo is an MC6809 and is capable of accessing 64K locations of memory. We only want eight. We can decode the other address lines to map only the eight locations we need, or we can use the already-decoded location in the computer.

This decoding is done in the SAM chip inside the CoCo. The pin that does this decoding is labeled "SCS" and is an active low output. That means the pin is normally high and, when accessed, will go low. In this case, the pin will go low when the CPU accesses memory locations \$FF40 to \$FF5F

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

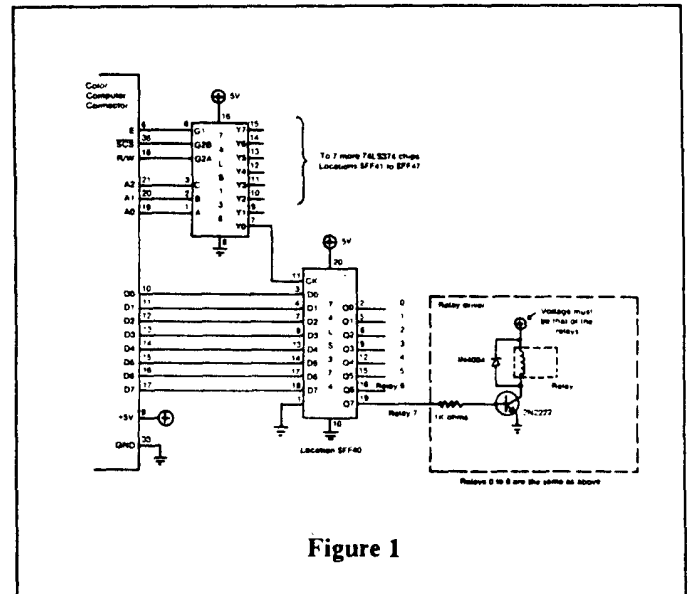


Figure 1

(65344 to 65375 in decimal). This represents a memory area of 32 bytes. If you have a disk drive system, it is reserved for I/O to the hardware of the drive. More on this later. We use this pin to activate the 74LS138. Since we only need eight of the 32 locations, the other locations will become mirror images of these eight locations and should not be used.

The next connection we make is the R/W line. This output line comes from the CPU and tells the hardware whether the CPU is reading or writing. In this case, the pin is high to read and low to write. Since our circuit controls relays, the CPU need only be able to write. The last line on the input side of the 74LS138 is connected to the E clock of the CPU. The E clock is a signal generated by the CPU to be used by hardware as a timing signal indicating when the data is valid on a read or a write. The other eight pins on this chip are outputs. Each of these output lines represents one memory location and can control one device.

Having eight locations means that you can control eight devices.

The 74LS138 chip is used to decode eight memory locations. Relax; we are getting closer to the relays. Now, the data that goes around on the data bus is always changing. The CPU is always busy. We need a component that will hold the data we write to these locations and remember it. This kind of part is called a latch. The one I will use is a 74LS374. It is an eight-bit latch.

Examine the 74LS374 in Figure 1. It has eight input bits that are connected to the CPU data bus. It also has eight output bits. These bits hold the value that is put into it when the CPU writes to that location. The location is controlled by the 74LS138. Each of these 374s has eight bits. Each of these bits can control one device. For instance, a relay is one device. However, the output of the 374 is not strong enough to turn on a relay by itself. A driver is needed. A one-stage transistor will do in most cases. In the diagram, only one circuit is shown, but it is to be repeated for every relay to be used.

Finally, we get to the relays! The relay you use depends on your needs. If you use the relay for very small current applications, then a relay such as the Radio Shack No. 275-243 will do. It will switch 2 amps and works directly off of 5 volts. If you need a higher capacity relay you must figure out the details by yourself.

The transistor used in this circuit can handle about 30 volts and can sink about 200 mA. Overdriving the transistor may damage it due to overheating. One 374 can control eight transistors and eight relays. If you need more than eight relays you must use another 74LS374. This will allow you to connect eight more relays and, for every 374 you add, another eight relays can be controlled. When I tried this circuit, I used three 74LS374s but only eight relays. Theoretically, you can connect up to 64 relays with this circuit, but I am sure you would run into power supply problems. You will have to drive the relays with a separate power supply.

So far, you can just write to the address locations that control relays. The only way the software can find out which relay is on is by keeping track of what value you stored in that location. But, with a little more hardware, you can read the memory locations and find out exactly which relay is on and which relay is off. The only drawback to this is that it limits the number of relays you can control to 32 instead of the 64 write-only relays. The choice is yours to make.

Figure 2 shows how to make a relay system that allows you to read the location as well as write to it. You will first notice the changes to the 74LS138. The A2 line is removed and replaced with the read/write line. This divides the eight output lines of the 138 to four read lines and four write lines. The four write lines connect to our transistor and relay system just like before. But now we have four read lines. We will need a different chip in order to read the output of the 374s. There are many chips you can use; I chose the 74LS244. It is an eight-bit buffer with tri-state. The inputs of the chip are connected to the outputs of the 374s. This way, the CPU can see right away, by accessing a read to the particular location, which relay is on or off by seeing which bit is high or low. The outputs of the 244s are connected to the CPU data bus. When the chip is selected (by a read), the data that is on the input appears to the CPU. It is as simple as that.

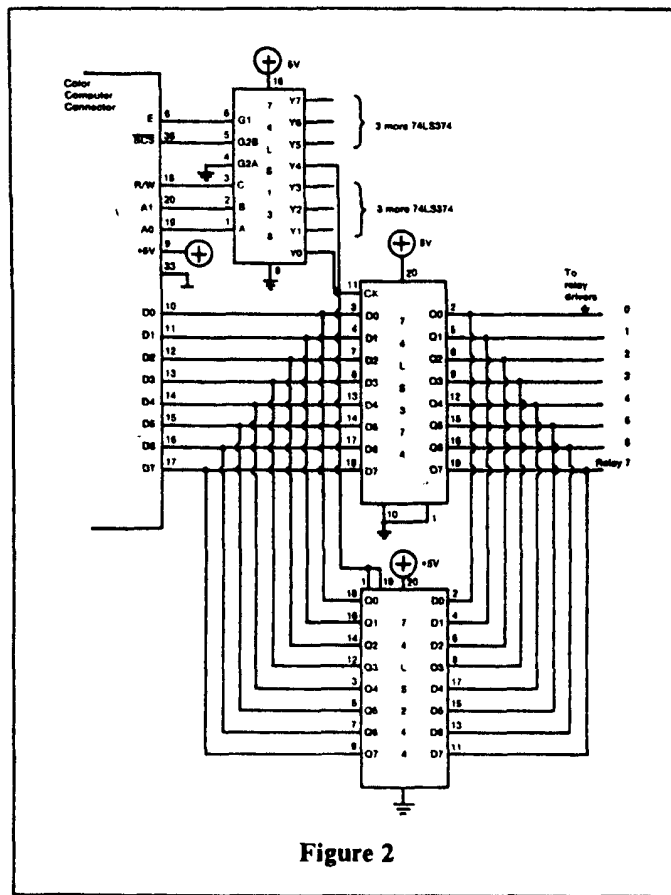


Figure 2

Now for the software. As I said before, we are using the SCS signal from the SAM chip. This signal maps our relays from \$FF40 to \$FF5F. If you are using the circuit in Figure 1, then the following structure is used:

Memory Location	Write Only to Relays
\$FF40	Relay 0 to 7
\$FF41	Relay 8 to 15
\$FF42	Relay 16 to 23
\$FF43	Relay 24 to 31
\$FF44	Relay 32 to 39
\$FF45	Relay 40 to 47
\$FF46	Relay 48 to 55
\$FF47	Relay 56 to 63

These relays are always least significant bit first. For example, relays 0 and 8 are on Data Bit D0 and relays 1 and 9 are on Data Bit D1.

If you wired up the circuit in Figure 2, then it should look like this:

Memory Location	Read/Write to Relays
\$FF40	Relay 0 to 7
\$FF41	Relay 8 to 15
\$FF42	Relay 16 to 23
\$FF43	Relay 24 to 32

The memory locations from \$FF44 to \$FF47 are the same as locations from \$FF40 to \$FF43, respectively.

Reading the locations \$FF40 to \$FF47 in Figure 1 is allowed, but the values you get will not be valid. To turn one relay on or off you must store (POKE command in BASIC) a value into one of the locations. What value you use depends on where the relay is. If you want to turn on Relay 0, then you must store a value of 1 in that location. If, for

example, you also want Relay 3 on, you must add the value of 8 to your previous value. Each bit value has a numeric value. Remember the binary counting system; I told you it would come up over and over again. I hope by now you understand what binary is all about. Anyway, the values associated with each bit go like this:

Bit Number	Decimal Value	Hex Value
D0	1	1
D1	2	2
D2	4	4
D3	8	8
D4	16	10
D5	32	20
D6	64	40
D7	128	80

The last thing I must talk about is the Multi-Pak Interface. If you are using a Radio Shack Multi-Pak Interface and a floppy disk controller, there is some

switching you must do first. The Multi-Pak has four slots. Each of these slots has two memory-mapping pins. The first is called the CTS pin. It is used to map up to 16K of memory area. The software for the disk drives called DOS usually resides there.

The second is the SCS pin we are using. The Multi-Pak has the capability of switching these signals to one of the four slots. It also has the capability of switching them separately. I mentioned earlier the hardware that controls the disk drives uses this pin. It uses the SCS in the slot the controller is in. If you want to use the relay complex with the Multi-Pak and a disk drive controller, you will have to do some switching before you use the relays. After you are finished, switch back to the original slot. Place the disk controller in Slot 4 and the relay complex in Slot 1. When you want to use the relay complex you must first do the command `PDKE &HFF7F, &H30`.

When you are finished and want to use the drive again, you must do the command `PDKE &HFF7F, &H33`. ☺

Cache of the Day

I recently asked some people who own computers, "What would you like to add to your computer?" Almost 80 percent of them said they wanted more memory. This is a universal problem. It is not limited to just the CoCo. The Apple, Commodore, Atari and IBM PC owners all said that they wanted more memory, too. They want it in whatever form they can get it: Main memory, bank-switched memory, RAM disk memory, ROM disk memory, ported memory — whatever the format, they want more! Well, the CoCo 3 has up to 512K of memory bank-switched into 8K blocks, we all know that. RAM disk adapters are available from several sources, including Disto (me). ROM disks are not that popular because they require an EPROM programmer and a knowledge of machine language programming.

What is left is ported memory. Now some people may say there is no difference between a RAM disk memory and ported memory. And as far as the hardware goes, there isn't. The difference is all in the software. A RAM disk and related software emulate a disk drive. You read and write to the RAM disk via files. To save data, you have to open a file, output to it, then close the file. When you want to retrieve data you have to open the file again, read your data and close it up when you are finished. This process takes time. It also uses the DOS (Disk Operating System). Now, ported RAM is the same, but since it doesn't use the DOS, it is not

restricted to using DOS and files.

You have to configure the use of the ported RAM yourself. The ported RAM you will see today is only 2K long. That means you will have 2,048 bytes to work with. Now, these bytes are only eight bits wide. The CPU in the CoCo can only handle eight bits at a time. So, when you want to save a numerical value, it can only be a number from 0 to 255. If you want to use numbers that are greater, you must use more than that one byte. For instance, if you want to use a number from 0 to 65,535, you will need two bytes. Or, if you want to use a signed number (i.e., a number from -32,767 to 32,767) you still need two bytes.

If you need still bigger numbers, you will have to go to a different type of format. A floating-point number takes up five bytes of memory for its mantissa and exponent. An explanation of these numbers goes beyond the scope of this article; see a math book for more details. You can also store alphanumeric characters. You need one byte for every character you have to store.

Now let's talk about memory-mapping. What, more memory-mapping? I am starting to sound like a broken record, but I still get a lot of letters about this subject. So, here we go again.

The CPU that is used in the CoCo is an MC6809E. This CPU can directly access only 64K of memory. In order to access that much memory, the CPU has

16 address lines. If you count in binary numbers, 16 lines gives you 65,535 different locations, better known as 64K memory. There are ways of fooling the CPU into accessing more memory. The technique is called page- or bank-switching. Bank-switching means you have more memory than the CPU can use at one time, but the memory is switched back and forth. An example is the CoCo 3. It comes standard with 128K memory and is upgradable to 512K memory. How is this done?

There is a chip in the CoCo 3 called the GIME. One of the functions of the GIME is called an MMU or Memory Management Unit. The MMU part of the GIME has the job of accessing 512K of memory and, at the request of the CPU, accesses all of it a bit at a time.

A good illustration of this is a radio. A radio can receive many stations, but only one at a time. The CoCo 3 has the equivalent of eight radios. Each radio can tune in one station at a time. Each radio (at the choice of the CPU) can access the same station. In the CoCo 3, each radio or page is 8K or 8192 bytes. There are 64 of these pages in a 512K CoCo 3 (8K x 64 = 512K). These eight pages of 8K bytes represent 64K to the CPU. There are eight control bytes in the MMU for pages. Each byte tells the MMU which page the CPU wants to see. Changing the data in these bytes changes which page the CPU can access. That is how the CPU can access more than 64K memory.

Now, what I am about to show you

is a mini-version of the MMU. Very mini. For users of the CoCo 3 with 512K of memory, this may not excite you, but for the memory-poor CoCo 1 or 2 user, a few extra bytes are always handy. The amount of extra memory is only 2K bytes. Not much by today's standards, but if you are working on that "does everything" program and you need "just a few more bytes" for something, this is where you are going to find them. I say mini because the "pages" are only 256 bytes long. With some special circuitry, that means this only uses up one byte of memory in the memory map. You also need one control or address byte. That is a total of two bytes in the memory map. Not too bad for 2K of memory. There is, however, a catch.

There are two ways of memory-

mapping this extra memory. The first is to have a couple of latches that hold the address of the memory. You set up the address of the memory byte you want, and then you read from, or write to, that address. That is the fastest way to get to any one byte, but you need to change the address every time you want another byte. The second way is to "auto-increment" every time you access data. For example, you read one byte, then when you read that same location again, you get Data Address 2; when you read it again, you get Address 3 and so on, until you get to the end of your memory. This is faster than setting up the address every time you need more data, but slower when the data you need is at the end of the file. If you are familiar with the structure of BASIC files, the first is

like a Random Access File and the second is like a Sequential File. Both have advantages and disadvantages.

What I have in mind is the best of both worlds. A little bit of auto-incrementing and a bit of address-latching. This way, you auto-increment by pages. I think a good auto-increment value is 256 bytes. That just happens to be the size of a disk sector. In a 2K RAM chip, there are eight 256-byte sectors, which means you can have up to eight pages of 256 bytes each. This is the basic description of the hardware project I have in mind for today. It is divided into three parts. The first is the hardware, the second is the memory-mapping of the hardware and the third is the software.

First, the hardware. In Figure 1, you will find the schematic diagram of the Memory Cache Project. The heart of the project is an HM6116 memory chip. This is a 2K-by-8-byte RAM (Random Access Memory) CMOS memory chip. It is made by virtually every memory manufacturing company. The 6116 number will always be the same, but the letters (which tell you which company the part comes from) may change.

Attached to the lower eight address lines is a binary counter. That is the auto-incrementing part. Attached to the upper address lines is a latch. That is the direct access part of the circuit. The fourth and fifth chips in the group are decoder chips, which map the thing properly.

The standard "project-building" tools are necessary, but there is one thing to remember. The memory chip I used is a CMOS part. It is easily destroyed by static electricity, so use a static-free work place. Also use sockets when trying out this circuit. It is better than soldering the chips directly — if you happen to burn one out, you won't have to desolder and resolder. All of these chips are available from your local electronics shop or from CRC Inc. The project board (to build your circuit on) can also be obtained from CRC Inc.

The second part of this project is memory-mapping. The circuit in Figure 1 is mapped at \$FF40 (65344). That, as you know by now, is where the disk drive and controller are mapped. If you have a multipack interface, it is not too bad; you can switch the multipack to the different slot and work with the extra memory from there.

If you only have a 'Y' cable, then the circuit in Figure 1 won't work. You need the second part. Figure 2 contains a circuit that decodes the address bus of

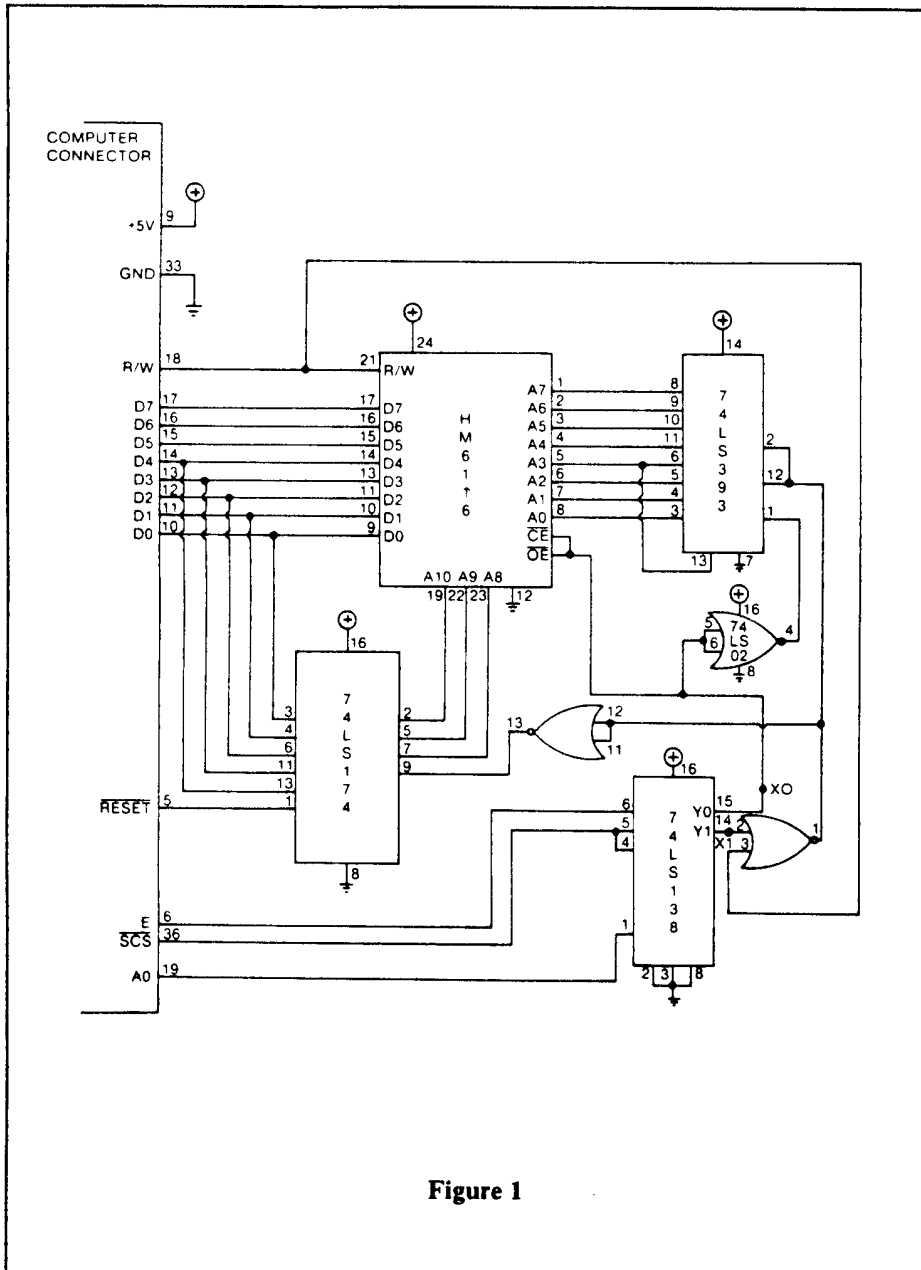


Figure 1

the CPU and maps it into a different place. It re-maps the address of \$FF40 (65344) to \$FF74 (65396), leaving \$FF40 free for the disk drive. This area of the memory map may be used by other products, so watch out for memory conflicts.

Finally, the software. It is not hard to access this memory cache. It is in two parts. The first part is to set up which of the eight pages you want to use. You do this by storing the page value at the base location. An example of this in BASIC is `POKE X, A`, where `X` is the base address. The base address is \$FF40 (65344) if you are using just the circuit in Figure 1 and \$FF74 (65396) if you are also using the circuit in Figure 2. The value `A` is the page number you want to access.

The second part is reading data or writing data into the 256-byte block. Remember, it is auto-incrementing and you have to access it 256 times to get to the last byte. An example of writing to the page in BASIC is:

```
1000 FOR I = 1 TO 256
1010 POKE Y, A(I)
1020 NEXT I
```

where `I` is your 256 auto-increment value. `Y` is your base address + 1 and `A(I)` is an array of data 256 bytes long, which must be previously defined. To read the block in BASIC, use this example:

```
2000 FOR I = 1 TO 256
2010 A(I) = PEEK(Y)
2020 NEXT I
```

where `I` is again your auto-increment value, `Y` is the base address + 1, and `A(I)` is your data array. Remember, though, this is just an example of how to read and write data to the RAM cache, just to show you how it is done.

You can use any method you choose. One point to keep in mind: Before you access a page, you must store the proper page number in the base address. This also clears your auto-increment counter to make sure you start at the right place. You wouldn't want to start in the middle.

If you have a problem with the circuit or want to make a comment on my projects, send me your letter along with a self-addressed, stamped envelope to me in care of RAINBOW, and I will get you an answer as soon as possible. (Please remember: no envelope, no answer.) □

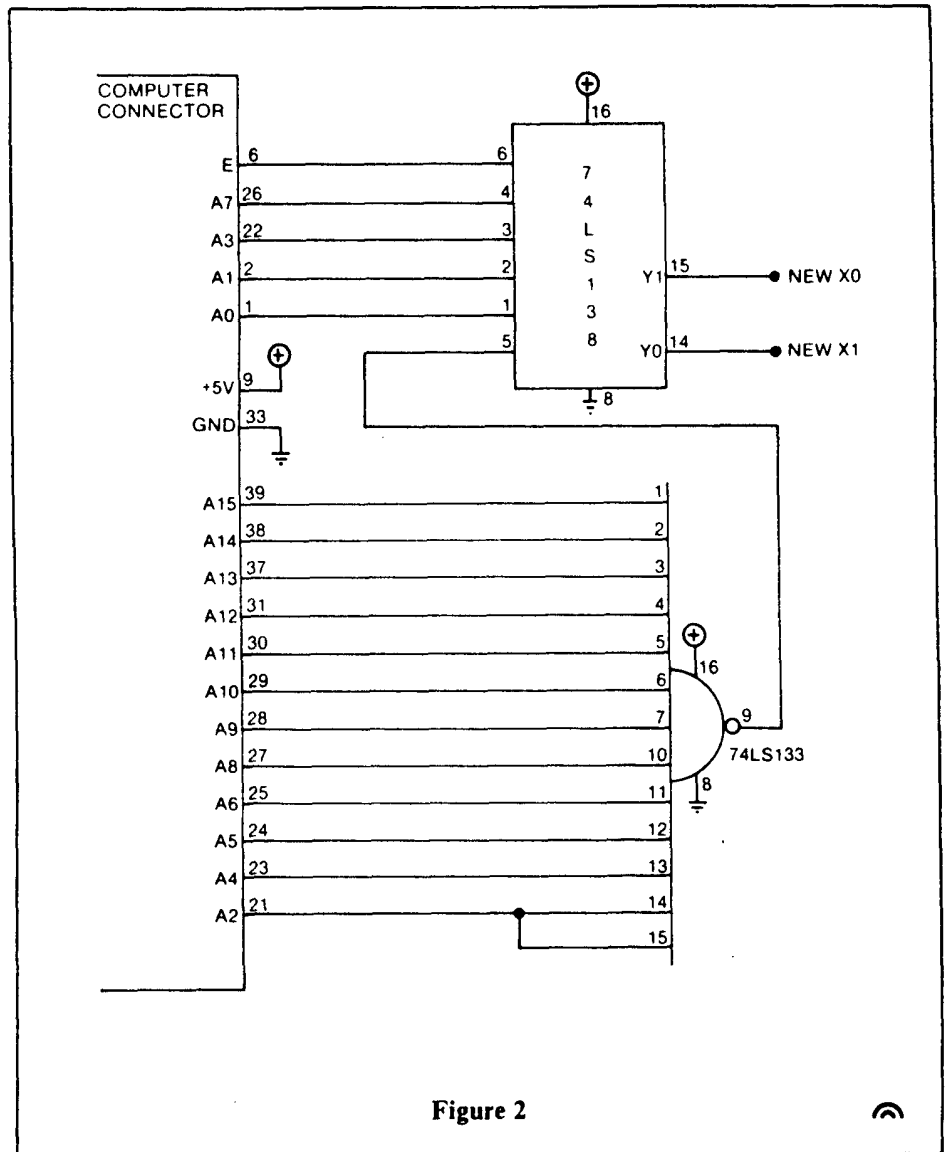


Figure 2

Clever Uses for Memory

By Tony DiStefano
Rainbow Contributing Editor

Many years ago, when the CoCo first came out, I was studying the memory map of the CoCo's CPU. I had only 4K of memory then, but soon realized that this CPU could access a lot more. In fact, everyone should know by now that the CPU in the CoCo can access 64K of memory.

I soon upgraded to 16K; that was easy. Then I read an article about upgrading the CoCo to 32K using a technique called "piggyback." That was wonderful. I now had a full 32K. Remember, this was before the time of 64K chips. I also had BASIC and Extended BASIC. That was another 16K, making a total of 48K of memory. There was 16K left, which was reserved for the cartridge slot. I started to wonder how I could put more memory in there. I now have a CoCo 3 with 512K, and I am still asking myself the same question!

I looked in what were then the latest catalogs on memory chips and came across a memory chip called a 2114. This is a 1K- by four-bit static RAM chip. Static RAM means it does not have to be refreshed as does dynamic RAM. It took two of these chips to make 1K of RAM. But I was desperate for more RAM, so I bought 16 of them, hoping to make an 8K RAM module for the CoCo cartridge slot.

After many hours of work over a hot soldering iron, I managed to make this 8K module work. It was mapped from \$C000 to \$DFFF. (For you people who still think in decimal, from 49152 to 57343.) It was great; I was the only kid

on the block to have that memory. I had many hours of fun with it.

Then came the 64K memory, and out went the 32K piggyback memory: A little bit of modification to the board and a little bit of wiring to the 74LS02, and presto — 64K of memory. That was great, but when it came time to use my 8K RAM module, it didn't work anymore. What the heck, I had 64K, so I just left it. Then I got my disk drive. It connected to the cartridge slot and there was no longer room for my 8K module. I put it on a shelf, where it gathered dust for many years.

Just the other day, I was working on something that required a little bit of memory that was protected. By protected, I mean I could not write to it when I needed. That is not the case of the CoCo in the 64K mode. You can write to anywhere in 64K when in the "All-RAM" mode. I thought of using an EPROM. It would certainly do the job, but an EPROM is a lot of trouble. You have to get out the EPROM burner, run the EPROM software, and erase it every time you have to start anew.

Well, this wouldn't do, so I went over to my long-term storage bin and pulled out my old 8K RAM module. With a bit of modification, I could make my RAM module into a ROM module, with just a switch to control it. Great idea — only one problem.

When it came time to write to the 8K module, nothing worked. I couldn't figure it out. Why wasn't I able to write to the cartridge area? After a long look at the CoCo schematic, I figured it out. When I had added the 64K memory chips, I had done a modification using the 74LS02. That modification prevented the CoCo from writing to the cartridge slot area. I was in trouble; my

little 8K module was now useless.

After some thought, I came up with a solution. It required a little bit of circuitry, but I was able to write to the cartridge area. For the circuit I am presenting here, I didn't want to use 16 chips to make 8K of memory, so I looked into my newest catalog and found one chip that replaced all 16 of the old memory chips. This chip is a 6264, which is an 8K- by eight-bit memory chip all rolled into one chip; my, how technology has advanced!

Building this circuit is a two-step process. With the proper hardware, I set up a one-byte read/write memory latch and a flip-flop, mapped at \$FF40. Remember them, way back when I was explaining about TTL gates? The first step is to store or poke a value into the one-byte memory. I used a 74LS374 for this, which is an octal latch. When you store the eight-bit value to that latch, you also preset half of a 74LS74. This is a D-type flip-flop with preset and clear. The output of this flip-flop goes to one side of a dual-input OR gate. You now have a valid byte in the latch and have flipped the flip-flop.

The second step is to read a byte from the 8K module. Remember that this read is to the non-writable area from \$C000 to \$DFFF, where the module is. The read does two things; first, it selects the 8K module. You are reading this location using a load or a peek command. But, if you look at the circuit in Figure 1, you will see that the output of the OR gate goes to the R/W (read/write) line of the memory. Normally, when you read from this location, the R/W line is high, which puts the chip in the read mode. Now that the flip-flop is flipped, however, the R/W line will go low when you read from the area. So,

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

the memory chip goes into the write mode.

But, the CPU is reading, and if the CPU is reading and the memory chip is writing, where does the data come from? Well, remember the latch? The output of the OR gate is also connected to the Output Enable of our latch. The memory chip gets its data from the latch, which is putting its data on the bus. There is no conflict because nothing else is putting anything on the bus; the CPU is reading and the memory chip wants data in the write mode. This action causes the data that we put into the latch to be put into the memory chip. That is how you write to an area of memory that is not writable. To end things, when we are finished reading, or should I say writing, the flip-flop is flopped back to the original state.

To summarize, every time you want to write to a location from \$C000 to \$DFFF, you must first store or poke that data to \$FF40. That loads up the

latch and flips the flip-flop. Then, read the location you wanted to write to, to transfer the data into it. That's all there is to it! By the way, it is automatically write-protected. You can't write to it and change the data — that is why I made this in the first place.

Now for the construction of the project. There are only four parts to it, as you can see from the schematic in Figure 1. In the case of the 74LS74 and the 74LS32, unmarked pins are unused. Here is a list of connections to the chips that connect +5V and GND:

IC #	Name	+5V	GND
U1	6264	28	14
U2	74LS374	20	10
U3	74LS32	14	7
U4	74LS74	14	7

It is recommended that you put all of these chips into sockets because if you make a mistake and burn out one of

them, it is a real pain to unsolder all the connections. You will also need a board to mount the parts on. You can get such a board from C.R.C. Computers Inc., (514) 383-5293. In fact, they have all the parts you need. The standard project building tools are necessary for this project.

A note to people who are using a Multi-Pak: In order to use this module with the Multi-Pak, you must set the switch to the slot that the module is in. If you have a disk controller and are using Disk Extended BASIC, you can switch to the modules slot by software, but you will lose Disk BASIC software, and the computer will crash. A good knowledge of machine language programming and Disk Extended BASIC is necessary to avoid crashing. The same goes with the CoCo 3. You can use it with the CoCo 3, but you must know how to switch into the ROM/RAM mode. Again, a knowledge of the machine is necessary. □

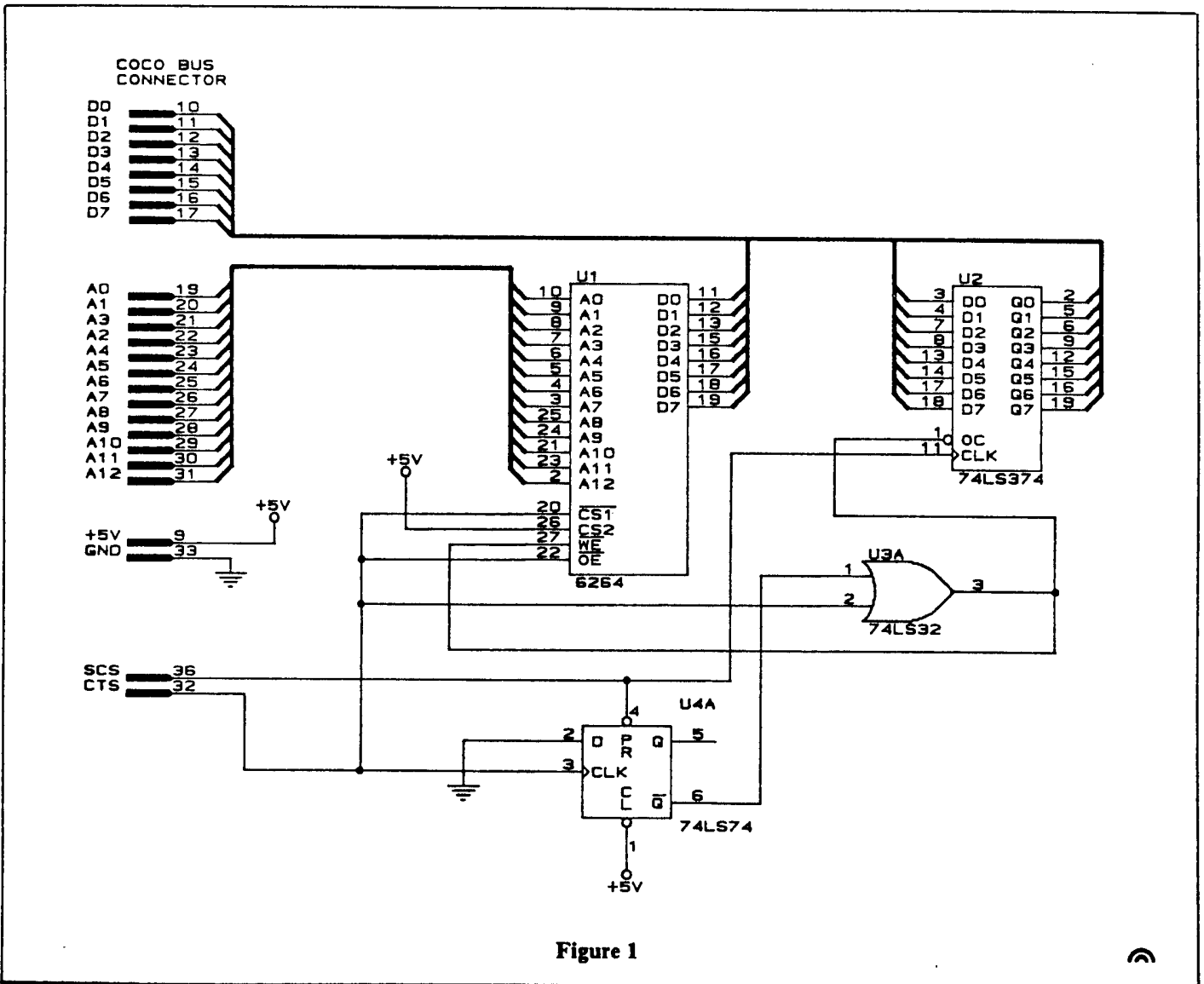


Figure 1

Building an EPROM Emulator

By Tony DiStefano
Rainbow Contributing Editor

A little while ago, I had to develop some software that would run in an EPROM in the CoCo. So, I took out my EPROM programmer and a few blank EPROMs and started to work. Soon after, I ran out of blank EPROMs and had to wait for my eraser to do its work. What a drag.

I started thinking that there must be something I could do about this. The easiest way to solve my problem was to buy a whole lot of EPROMs. It was a quick solution, but the problem was only delayed. And it was also costly. I sat and thought about it for a while, then came up with this idea. Why not build something that would take the place of an EPROM and not take so long to erase?

I had to make some type of RAM that looked and programmed just like an EPROM. RAM does not need to be erased. You just have to write over the top of it and, bang, it is done. If you have a choice of programming times (or can get into the program to change it), you can save a lot of time programming the emulator, too. While it takes time to program an EPROM, you can do an EPROM emulator in no time flat.

What about this chip? It shouldn't be too hard to do these days, with the amount of memory chips available. So I looked into my favorite memory reference manual for some ideas. The EPROM I wanted to emulate was the 2764. I use them a lot and they are quite popular. The 2764 is an 8K-by-8 EPROM in a 28-pin package. I had to find the closest match possible.

After looking through the memory

manual, I found an almost perfect chip. It is an 8K-by-8 Static RAM chip in a 28-pin package. I couldn't have come closer if I'd designed it myself. Next, I needed to find some pins that matched. I looked up the pinout for the 2764, compared the two and, presto, found an almost pin-for-pin match. What is this mystery chip? It's a 6264. Many companies make it, and it's not expensive.

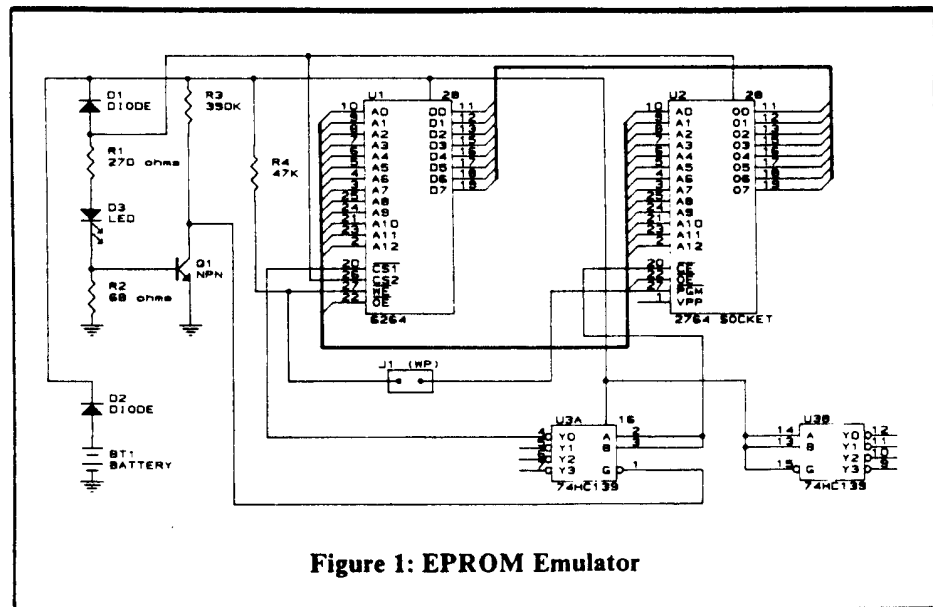
That is the first part. But what about retention? As soon as you power down static RAM, you have no more memory. Now you need a battery (a small coin-size battery will do) and a little bit of support circuitry.

Figure 1 shows a diagram for an EPROM emulator. It takes an 8K RAM chip and a battery and turns it into an EPROM. The circuit is quite simple, but still needs some explanation. Also the construction of this is a bit tricky. I will start with the circuit and

finish with construction tips.

But first, you must remember that you are working with CMOS memory chips. These are very sensitive to static electricity. Your work area must be static free, and the best way is with a static-free mat. If you don't have one, take the following precautions. Don't wear running shoes on a shag carpet; it's better if the room is damp; and before you start to work, ground yourself by touching a water pipe. I also recommend that you use a grounded soldering iron. You will need just the standard project building kit for this one.

In Figure 1, you will notice that U1 is the RAM chip. To access 8K of memory, you need 13 address lines, A0 to A12. These lines are directly connected to the socket. Instead of drawing each wire separately, they are tied into what is known as a bus. The bus is indicated by a dark line. When members



Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

go in and out of the bus, they have a name. Members of the same name connect together. For instance, A2 on the 6264 connects to A2 of the 2764 socket and so on. All of the eight data lines are also connected together.

In order for the RAM chip to retain its data, it must be kept powered at all times. This is where the battery comes in. It keeps the power to the chip at all times. The battery also keeps U3, R4 and Q1 (via R3) powered by way of D2. This way the memory chip cannot lose data. But this presents another problem. Touching the pins of a powered chip may change the contents of its memory. If you happen to touch the R/W line and the CE line, you've just changed memory, and you can't afford to change even one bit. The rest of the circuit is dedicated to protecting the memory.

When the emulator is powered from the 2764 socket (i.e., it is in circuit and the circuit is on), D1 is forward biased and conducting, and D2 is reversed biased and is shut off. The LED D3 is on and supplies base current to Q1, and Q1 is conducting and brings Pin 1 of the 74HC139 low. This pin is the enable to half of a 2 to 4 decoder. Many different kinds of chips will work here, but I picked this one because it is easy to get. With this chip enabled, the CE of the 2764 will pass through and enable the RAM chip. When there is no power to the 2764 socket, D2 is forward biased and keeps power to the RAM chip. On the other hand, D1 is reversed biased and is shut down. This turns off the LED and Q1 no longer gets base current and becomes high impedance. This lets Pin 1 of U3 go high. That disables the chip, and when the chip is disabled, you cannot change its data.

R4 is used to tie the R/W line of the RAM so that it defaults to a read operation. This is so that no writes to

memory can be made when the power changes from external to battery or vice versa. The jumper J1 protects the chip from writes. When you are programming the emulator, you must have that jumper in. But, when it is time to use the emulator, you must remove this jumper. Again, this is another method to prevent writing to the RAM chip when you don't want to.


The other half of the 74HC139 is not used. The inputs are just tied high. This is necessary to prevent the chip from doing things on its own, like oscillating by itself and using up power for nothing. The ground connections to the ICs are not shown. You must connect Pin 14 of U1 and Pin 8 of U3 to Pin 14 of the 2764 socket U2. Normally, the power connections to these chips are not shown, but in order to get the power to the right places, I put in wires. Also, it may seem that not all pins are connected from the RAM chip to the EPROM socket; it is true, a couple are missing. Don't worry about it because they are not connected (N/C).

All the parts should be available at your local electronics parts shop. There are no hard-to-get parts. Your local Radio Shack store may have most of the parts, but not all. Note, R1 and R2 are in a delicate circuit and should not be substituted for a "close" value. Q1 is any "high gain" switching transistor such as a 2N2222 or a 2N3904. The battery voltage is about 3.6 volts. It can be just one or a combination of many, as long as it is about the right voltage. If you decide to use rechargeable batteries, a 300 ohm resistor across D2 will recharge them when the emulator is in use. The diodes should be low-leakage types with low, forward-voltage drop.

As far as the construction of the unit goes, I used one trick that works well. You need a printed circuit protoboard. I didn't have any around, so I cut a piece

of protoboard from one of my previous projects. If you don't have one, Radio Shack does. You need one that's about 2 by 3 inches. You can cut it down to size later. It is better if you get the kind that does not have any bus lines. Get the type that has just pads; it is easier to work with.

Now, for the socket. There are many techniques, but the way I do it is quite simple. Take a 28-pin socket made for wire wrapping. Insert the socket halfway into the protoboard and solder all the pins. This way you get a 28-pin socket on top and a 28-pin "plug" on the bottom to put into your EPROM programmer. After all the pins are soldered, locate all the pins that are not the same on both. Solder a piece of wire just under the top socket. Solder another piece of wire to the bottom of the pin. Now cut the wire in between the two wires. This way, you get a clean cut without reheating the pin and taking a chance that the solder melts and the pin goes crooked.

If you write a lot of machine language programs that run in ROM or EPROM, this emulator will save you hours of time. Using the emulator is quite simple. After the construction is made, remove the jumper J1. Plug the emulator in an EPROM programmer and set the programmer for 2764. Insert the jumper and program the emulator. Normally, it takes a few minutes to program. It takes time for the EPROM circuits to change those 1s to 0s. But, it doesn't take any time to program RAM. If you can find where in the software the program delays are, change them to 0. You will see a great improvement in the time it takes to program the thing. Now, remove the jumper and remove the emulator. All you have to do is insert the emulator into a 2764 socket and you have the equivalent of a 2764 EPROM. 

Dissecting the Disk Controller

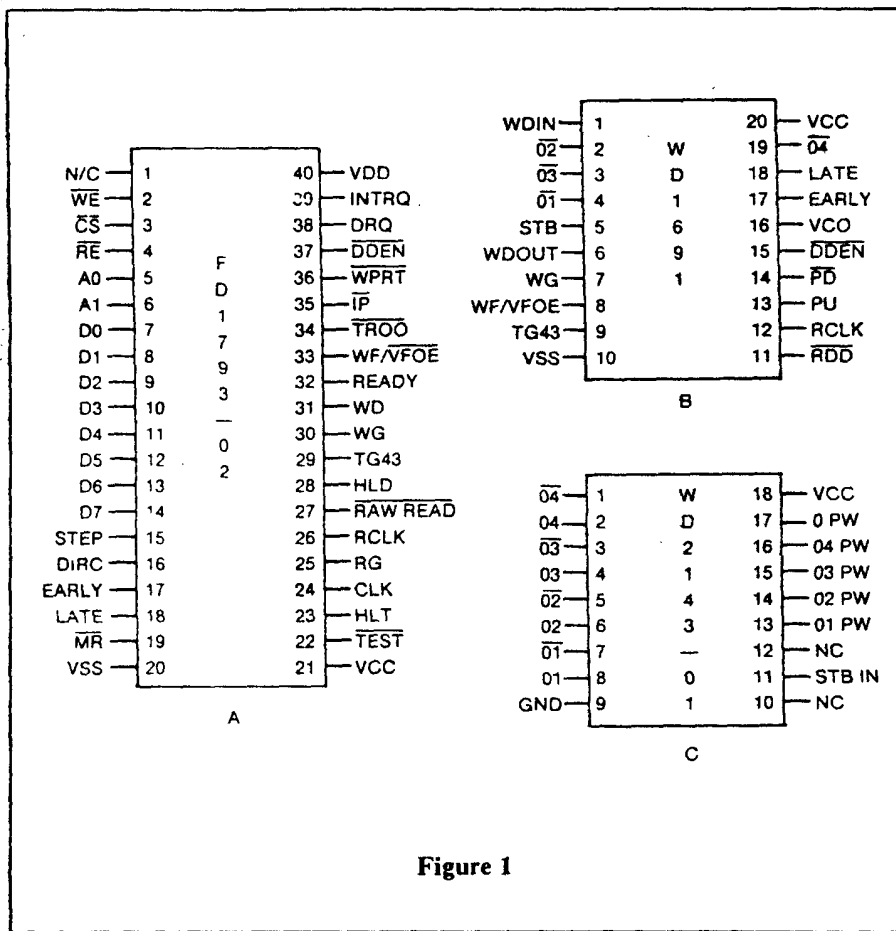


Figure 1

- 1 — NC. This pin has to be left N/C though it connects to a back bias generator.
- 2 — WE. The Write Enable pin tells the FDC to write data.
- 3 — CS. The Chip Select is used to map the FDC into the CPU's memory area.
- 4 — RE. The Read Enable pin tells the FDC that a read cycle is being done.
- 5 and 6 — A0 and A1. Two address lines select one of the four registers of the FDC.
- 7 to 14 — D0 to D7. The eight data lines that transfer data to and from the FDC and the CPU.
- 15 — STEP. This output steps the

- disk drive to the next track.
- 16 — DIRC. This output tells the disk drive which direction to step.
- 17 — EARLY. This indicates an early write precompensation.
- 18 — LATE. This indicates a late write precompensation.
- 19 — MR. A low on this pin resets the FDC completely.
- 20 — VSS. This line is the ground line for the FDC.
- 21 — VCC. This line requires +5 volts.
- 22 — TEST. This pin is used for testing and should be kept high during normal operation.
- 23 — HLT. The Head Load Timing signal is high when the head is engaged.

- 24 — CLK. This input requires a free-running 1MHz. clock.
- 25 — RG. The Read Gate is used to synchronize the external data separators.
- 26 — RCLK. The Read Clock is a square-wave signal derived from the data stream.
- 27 — RAW READ. Data stream directly from the drive.
- 28 — HLD. Head Load controls the loading of the disk head against the floppy disk.
- 29 — TG43. This outputs tells the support circuits that the head is sitting on a track greater than 43 (for 80-track drives).
- 30 — WG. Tells the drive that a write is to be done.
- 31 — WD. The Write Data output contains the data and address marks to be written to the drive.
- 32 — READY. The Ready input tells the FDC that the disk is ready for a read or a write operation.
- 33 — WF/VFOE. A bi-directional signal. When WG=0, a low will terminate any write command. When WG=1, this pin remains low until the end of the data field.
- 34 — TR00. This input tells the FDC when the head is positioned over Track 0.
- 35 — IP. This Index Pulse input tells the FDC that the index hole has just gone by.
- 36 — WPRT. The Write Protect input tells FDC that you cannot write to the disk.
- 37 — DDEN. Double Density pin tells the FDC if you want double or single density.
- 38 — DRQ. This output indicates that the FDC is ready for another byte in the write mode and that the buffer is full in the read mode.
- 39 — INTRQ. The Interrupt Request indicates that any command has been finished.
- 40 — Vdd. This input requires +12 volts.

A little over a year ago, I started a series of articles describing the LSI (Large Scale Integrated) circuit chips of the CoCo. There was the CPU, the SAM, the PIAs and the VDG along with a whole lot of other TTL support chips. One thing that I did not touch upon is the disk controller. The controller from Radio Shack also has some LSI chips. In fact, the first Radio

Shack controller, Catalog No. 26-3022, used a three-chip set. The later controllers used more up-to-date parts. What I intend to do this month is to describe the older controller and the newer controller. In both cases, you will learn more about the FDC (Floppy Disk Controller).

The controller chip that Radio Shack used in their first controller is a part

made by Western Digital. The FD1793-02 is a floppy disk formatter/controller. That means that the controller can format a disk as well as read and write to it. This chip had many features: soft sector format compatibility, automatic track seek with verification, single and double density, and IBM 3740 and 34 densities, just to name a few. This was a wonderful chip. It came in a 40-pin

- 1 — \overline{CS} . The Chip Select is used to map the FDC into the CPU's memory area.
- 2 — \overline{RW} . The Read/Write pin tells the FDC what cycle is being done.
- 3 and 4 — A0 and A1. Two address lines select one of the four registers of the FDC.
- 5 to 12 — D0 to D7. The eight data lines that transfer data to and from the FDC and the CPU.
- 13 — \overline{MR} . A low on this pin resets the FDC.
- 14 — GND. The ground return for all signals.

- 15 — Vcc. Power supply of +5 volts only.
- 16 — STEP. This output steps the disk drive to the next track.
- 17 — DIRC. This output tells the disk drive which direction to step.
- 18 — CLK. This clock input requires an 8MHz. clock.
- 19 — \overline{RD} . This input requires raw data from the disk drive.
- 20 — PRECOMP. This input tells the FDC when to use the write pre-compensation circuit.
- 21 — WG. Tells the drive that a write is to be done.

- 22 — WD. The Write Data output contains the data and address marks to be written to the drive.
- 23 — $\overline{TR00}$. This input tells the FDC when the head is positioned over Track 0.
- 24 — IP. This Index Pulse input tells the FDC that the index hole has just gone by.
- 25 — \overline{WPRT} . The Write Protect input tells FDC that you cannot write to the disk.
- 26 — DDEN. Double Density pin tells the FDC if you want double or single density.
- 27 — DRQ. This output indicates that the FDC is ready for another byte in the write mode and that the buffer is full in the read mode.
- 28 — INTRQ. The Interrupt Request indicates that any command has been finished.

package, very compact for its day. But it required at least two other support chips — the WD1691 Data Separator and the WD2143-01 Four Phase Clock Generator. Together these three chips and a half-dozen or so other support chips made up the controller. The power requirements for this setup is 5 volts and 12 volts.

The chips in Figure 1 are the pinouts of the three Western Digital parts that make up the heart of the controller followed by a pin-by-pin description of the FD1793-02 controller. Overlines indicate that the signal is an active low pin.

The other two chips are used to support the FDC. They connect to each other in various ways and connect to other TTL circuitry. It would be a little too long to explain each pin of these two chips and maybe even useless. Yes, useless, because the three-chip FDC combination is now obsolete. Western Digital has since redesigned the FD1793 and made a new chip called the WD1773. This chip has the WD1691 and the WD2143 built right into the new FDC. That's right, three chips in one. Another welcomed feature of the WD1773 is that it does not require +12 volts. It will run on a +5 volt supply only.

This development was great for Radio Shack because they had just released the new CoCo 2. It was smaller, lighter and less expensive than the CoCo 1. Following the new CoCo 2 came a new controller. Radio Shack had to come up with a new controller that did not use +12 volts. It was easy with the new FDC. Not only did it not use +12 volts but was less expensive than the older three-chip set. It also required less

support circuitry. Another plus for the new FDC was that it did not have any adjustments. The older 1793 had three trim pot adjustments.

This new controller was great all around. Less money, no +12 volts, only one part, and no adjustments. It also had one more feature: It came in a 28-pin package. Figure 2 shows the pinout of the WD1773 and a pin-by-pin description. Notice that functionally, the parts in Figure 1 and Figure 2 are the same.

Since the introduction of this new FDC chip, many companies have used it to make their own version of Radio Shack's controller. Though the exact circuitry may vary from design to design, they have to follow certain rules. First, the FDC has to be mapped to the same memory area. That requires some sort of memory-mapping chips. The way in which the FDC and the CPU transfer data has to be the same if it is to be compatible.

The technique used in a Radio Shack or compatible controller to transfer data (a complete sector) is not too difficult to understand. It starts off with the CPU setting up the FDC registers for the right track and sector. It then turns on the proper drive motor and drive select. Next, it gives the read or write sector command to the FDC. It checks to see that everything is all right, then, it flips the bit that halts it. This is done by hardware that pulls the HALT line of the CPU low. When the FDC has data ready or needs data from the CPU, it unhalts the CPU via the DRQ line of the FDC. The CPU then stores that byte of data into memory on a read or fetches another byte from memory on a write and then halts itself again. This procedure

is repeated until all the data of that sector is transferred. At this point the FDC fires the INTRQ; this signal is connected to the IRQ of the CPU. The IRQ routine then gets the CPU out of this loop and continues to the rest of the read/write sector software.

This procedure is the same for all controllers that hook to the CoCo. Until now that is. Disto is soon to announce a new controller called the Super Controller II. This controller will do everything the Radio Shack controller can and more. It also has a

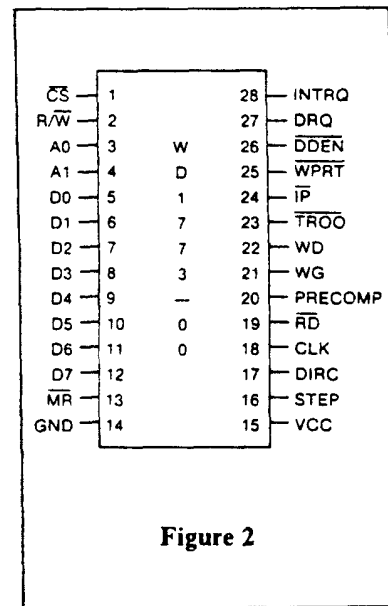


Figure 2

different way to transfer data. It has built-in RAM memory and the support circuitry to transfer data to and from this RAM without the use of the CPU. In the OS-9 operating system, this is a big boost. Data is transferred without the use of the HALT line. The CPU

does not have to mask the interrupts. That means that once the CPU gives the command to the FDC, it is free to do other things and return to get the data when the FDC is finished. That means no more missed characters on the keyboard when a disk operation is running.

Information for this column was taken from *Storage Management Products Handbook 1986*, Western Digital Corporation, Literature Department, Irvine, California, and *Color Computer Disk Interface*, Tandy Corporation, Fort Worth, Texas. ☺

A New, Improved Printer Adapter

By Tony DiStefano
Rainbow Contributing Editor

I have watched our computer grow from a 4K CoCo 1 to a 512K CoCo 3. BASIC has improved from ho-hum simple Color BASIC to Extended Color BASIC, to Disk Extended Color BASIC. That is some improvement. The hardware has gotten faster and the software has gotten better. There is, and always will be, a close relationship between software and hardware. It's a closed loop. The hardware cannot work without the software and the software cannot work without the hardware. This is where I sometimes have a dilemma. I have many ideas for hardware, yet do not have the time or the skill to implement the proper software.

DOS (Disk Operating System) or, for that matter, any software in ROM (Read Only Memory) is a lot harder to deal with than software in RAM (Random Access Memory). This is because ROM cannot be changed, but RAM can be. So, if there's a little piece of hardware you want to add on, it must be supported by software. To add on some hardware, you can plug it into the cartridge slot or you can plug it into a multipack. If you are like me, you can also solder it right in. All you need is the hardware.

The software, on the other hand, can be loaded from cassette or disk, or typed in from the keyboard (if it is not too long). But, whatever the method, soft-

ware may cause you problems. If it resides in memory, no matter where it is, it will be erased by something else. In the CoCo 1, 2 and 3, there is only 32K of memory available for BASIC programs. The other 32K is reserved for BASIC itself. In the case of the CoCo 1 and 2, this 32K of memory space is taken up by ROMs.

If, for example, you want to make changes to BASIC, you need to have 64K of memory. Then you need a routine to transfer ROM into RAM. Only then can you make changes to BASIC. For instance, if you don't like the word PRINT you can change it to SPLAT. If you want to change a routine, it can be done. Of course, you will need some knowledge of how BASIC works. But the fact is you can do it.

When the CoCo 3 came out, good ol' Radio Shack made it a little easier for us. First, the CoCo 3 comes with lots of memory — a whole 128K of it. But BASIC can still use only 32K. The main difference is that BASIC itself is in RAM, which makes it a lot easier to modify. There is one less step to do in the CoCo 3. Also, since it comes with all CoCo 3s, there is no problem with, "Will it work with mine?" And it doesn't use up memory for BASIC programs.

Now it comes down to, "What am I going to do with this?" Well, a while back, I made a parallel printer port that plugs into the cartridge port. It was a PIA (Peripheral Interface Adapter). There were a couple of things wrong with this adapter. First, if you had a disk

drive attached to your computer, it didn't work. Second, it had to be re-initialized every time you pressed the reset button. And the CoCo 3 was not available then, so you needed driver software, which was always in the wrong place.

This time I am making a new parallel printer adapter — a better one, in many ways. First, it will be inside the CoCo. Second, since I am not using a PIA, it will not be necessary to re-initialize after a reset. And, if you are installing it in the CoCo 3, which always works in the all-RAM mode, the driver will not be erased by other software. The rest of this month's article will be taken up by the construction of the adapter board itself, and next month we'll finish by hooking it up to a printer and a software driver.

As you can see in Figure 1, this is not a big project. It only requires three chips. I did it this way because I did not want to use a 40-pin PIA chip, for a couple of reasons. I've already discussed one reason; the other is size (the smaller the better). I think it is a little cheaper, too, and those are magic words. Anyway, the first chip is an eight-bit latch. It is used to latch the data that is to be printed. Without a latch, the data would not be held long enough for the printer to receive it. The chip I used in this case is the same chip Radio Shack's newest controller uses to set the active drive. It is the 74LS273.

The second chip in the circuit is a tri-state buffer. Before data can be sent to

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

a printer, it is up to the software driver to determine that the printer is not busy. This is done with software that reads the busy line on the printer. The second chip in the circuit is connected to the busy line of the printer. The output of this buffer is connected to Bit 7 of the data bus. When a READ to that memory location is done, the status of the printer is easily known. The chip in question is a 74LS125.

The third chip in the circuit is very important. It is used to memory map the printer data latch and the busy indicator into the picture. The chip I used for this is a 74LS139. It is a dual 2-to-4 decoder. Memory mapping extra devices into the CoCo's memory area is a very delicate operation. There are not very many locations available that don't violate someone's real estate.

But, I have a trick up my sleeve. The I/O area used for the disk drive hardware is mapped from \$FF40 (65344) to \$FF5F (65375). That area takes up 32 bytes. You need only five of those 32 bytes to operate the disk drive. The other bytes are wasted because they are mirrored. "Mirrored" means you access more than one byte but get the same hardware being activated. In the case of the CoCo's map, the five bytes are all located between \$FF40 and \$FF4F (65359). The first thing this chip does is separate the upper half of the I/O area from the lower half. This is done using half of the 74LS139. It separates the \overline{SCS} line into two sections. The first section, \$FF40 to \$FF4F, will go to the disk controller. That is needed if you are to use a disk drive. We will use the second section for the printer I/O.

The second half of this chip is used to further decode the section into two more sections. The first of the two sections is used for data. This signal is also used to strobe the data into the printer. This is done by running a line from this output to the Strobe input of the printer. The second section is used for the busy line. It is used to activate one of the tri-state buffers of the 74LS125. The other buffers of this chip are not used.

To recap, the new memory map looks like this: The untouched area is from \$FF40 to \$FF4F. This area has to go to the disk drive. The next area is \$FF50, which is used for the data latch. Finally, the third area is \$FF58, and it is used to monitor the busy signal.

Constructing this board is not a big deal. You will need the three chips mentioned above. It is recommended that you use sockets for the chips (a

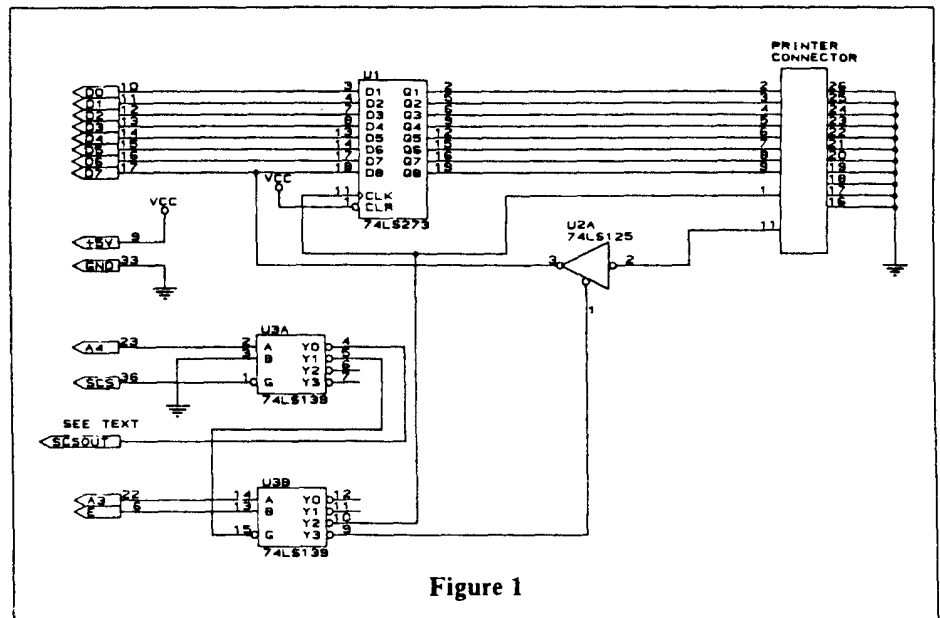


Figure 1

20-, a 16- and a 14-pin socket). You will also need a small board to mount the chips on. The way I decided to put it in, the PCB will not need an EDGE connector. A 2-inch by 3-inch board is more than enough to fit all the parts on. Radio Shack has such a board.

As usual, there is more than one way to skin a cat. Some may like to solder directly, and some may prefer to use connectors. This time I'll use a connector for the output and direct wiring for the input. As a connector for the output, I used a dual in-line header. This is a connector that has two rows of pins that are spaced at one-tenth inches between the rows and at one-tenth inches between the pins. You will need a 26-pin connector.

The connector should mount on the same side as the components. It is numbered as follows. Look at the pins lengthwise. Pin 1 is the bottom left-hand pin. Continue counting counter-clockwise till you get them all. See Figure 2 for its position. All pins not mentioned are N/C. The construction of the board is simple and requires only the standard project kit. In Figure 1 the 5 volts and ground pins are not indicated. The following is a list of the 5 volts and ground connections:

IC	+5 Volts	GND
74LS273	20	10
74LS125	14	7
74LS139	16	8

Also not shown on the schematic are three decoupling capacitors. The value of the caps is one-tenth uf at 25 volts. They go between +5 volts and ground,

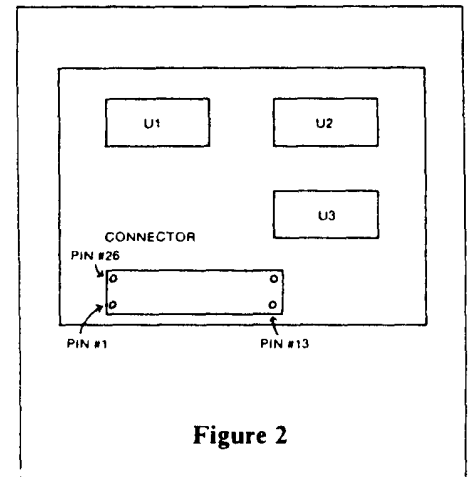


Figure 2

and as close to the chips as possible. These caps are used to decouple the supply to the chips. There is one more thing to do. Since the \overline{SCS} line has to be decoded to a different state, it has to be cut. The best place to cut the line is right at the connector. In fact, that is the best place to get all of the signals — right at the connector. Use the connector numbers, but solder the wires directly to the connector on the inside.

Cut the connector and pry the two ends apart so they do not touch. The end that goes to the connector is the \overline{SCSOUT} and the side that goes to the PCB is the \overline{SCS} . Build the circuit first, then connect the wires to the connector. Make the wires as short as possible so that they won't be in the way of anything. Use four plastic screws and some rubber cement to fix the board to the computer.

Next month I'll finish up by making and installing the printer cable and getting the different drivers for CoCos 1, 2 and 3.

Finishing the Printer Adapter

Last month I started something I now have to finish, a parallel printer adapter for your CoCo — something internal to your CoCo that will give you a parallel printer port without using a Multi-Pak or special controller.

Last month was the hardware side of this two-part project, which I'll review quickly. A small PCB that goes inside your computer has three ICs on it and connects to the inside of the cartridge connector. It also has a 36-pin connector. This connector can be connected to any Centronics type parallel printer. The hardware uses two bytes to talk to the printer. The first, at \$FF50, is the latch to which the character to be printed is located. The second, located at \$FF58, is a readable bit that shows the state of the printer, busy or not busy. The data at the latch is auto-strobing, which means the second the data is latched, the printer is told about it. You don't have to strobe the printer separately.

This month I will do two things: first, I'll describe how to build (or buy) a printer cable; second, I'll describe the software required to drive this parallel port.

You can get the cable in one of two ways. The simple way is to run to your local Radio Shack and buy a cable. Just ask for a cable to connect a Model 100 computer to any Radio Shack parallel printer (Catalog Number 26-1409).

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

“With the proper driver, the parallel printer adapter would work with all OS-9 software.”

The second way is a bit harder but also less expensive. You need three parts: a 36-pin ribbon printer connector

(Radio Shack carries it, but at a stiff price), a length of 26-conductor ribbon wire (if you cannot get a 26-conductor,

```

NAM POUT
ORG $0000
START LDX $168
      STX PT2+1,PCR
      LEAX POUT1,PCR
      STX $168
      LDA #$39
      STA START,PCR
      RTS
POUT1 PSHS B
      LDB $6F
      CMPB #$FE
      PULS B
      BEQ POUT2
PT2   JMP $CB4A
POUT2 TST $FF58
      BMI POUT2
      STA $FF50
      LEAS 2,S
      PSHS B
      CMPA #$0D
      BEQ POUT3
      INC $9C
      LDB $9C
      CMPB $9B
      BLO POUT4
POUT3 CLR $9C
POUT4 PULS B,PC
      END
      FOR PRINTER
      IS IT CR?
      YES
      INC CHR COUNT
      CHECK END OF PRINT LINE
      END?
      NO
      RESET CHR COUNT
    
```

Figure 1: Driver routine for any CoCo

```

ORG $A2F7
FCB $21
CODE FOR BRN
ORG $A2C3
POUT TST $FF58
      BMI POUT
      STA $FF50
      JMP $A2DF
      PRINTER BUSY
      YES
      PRINTER DATA
      CONTINUE
    
```

Figure 2: Driver routine for CoCo 3 only

get a higher number and split the difference. A common ribbon wire available is a 25-wire. This will do just fine since the 26th wire is not used. Just make sure that the missing wire is not on the pin number 1 side), and a female 26-pin dual inline header. To assemble the cable, start by locating pin number 1 on both connectors. Usually, the ribbon wire will come with a red stripe on the side. Line up Pin 1 of one connector to the red stripe. Push the wire into the connector and crimp the connector. Be careful that the wires align up with the connector teeth. Next, do the same thing with the other connector. That's it, your cable is done. Now it's time to get into that "Do I really have to?" part of the project, yes, the software.

Deep in the ROMs of the CoCo lies software. This software is called BASIC, Extended BASIC and Disk BASIC. Also in these ROMs are drivers that control the computer. Reading the keyboard, displaying a character on the video screen, getting a file from disk and printing a character on a printer are all software functions built into these ROMs. These functions are sometimes called Basic Input Output Subroutines, or BIOS, for short.

In the case of the CoCo's printer routine, it is in the BASIC ROM. Without going into too many details, the printer routine has what is called a RAM HOOK. If you look in the "Machine Language Subroutines" section of your BASIC manual, you will find one routine that is called `CHROUT`. This routine will output a character to the device specified by the contents of a byte in memory. The value of that byte will determine which device the character will be sent to. If that value is `-2` or `$FE` as a signed eight-bit integer, that character is destined for the printer. But before this character is sent to the printer routine, it goes through the RAM HOOK. This is a few bytes in RAM that, if changed, can re-route the character to your own driver. This is where my routine comes in.

Look at Figure 1, the driver routine for my parallel printer port. It will work with any CoCo. I wrote it using the *Micro-Works* editor/assembler. You may have to change some things around if you use another package. The first part, called `Start`, initializes the software by changing the RAM HOOK to

`POUT1`. It then puts an RTS at the beginning of the routine so it cannot be done again. The new printer routine starts at `POUT1` and checks the device number to see if the character in question is for the printer. If it is not, the routine continues to where it would normally go had we not changed the RAM HOOK. If the character is for the printer, the routine then moves to `POUT2`, where the printer is tested to see if it is busy. If it is busy, the software waits in a loop until the printer is free. If the printer is not connected, the software will wait forever.

After it is established that the printer is no longer busy, the software proceeds by sending the character to the printer. By now you would think that your job is finished. No way, there's a little more to go. First, we get rid of the return address, because the character has been processed and must return to whatever software called the printer routine to begin with, avoiding the serial printer routine.

To stay compatible with the regular printer routine, this software must do one more thing — deal with carriage returns. There are two variables used with the regular printer routine: character count and printer line length. Every time a character is output to the printer, the character count is incremented and checked with the printer line length. If it is equal, it is then cleared. When a carriage return is issued, the character count is again cleared. You may ask yourself what use this routine might have. Well, the printer routine itself does not use it, but other routines like `TAB` use these variables. After all this is taken care of, the routine is finished and returns to its caller.

A few notes to this program are necessary at this time. If you noticed, at the beginning of the routine there is an `DRG` statement. This tells the assembler where the software is to be loaded in memory. The value after the `DRG` statement is 0 to make things a little easier for the user. While the program will not function properly when it is assembled, calculation of the offset is made easier.

The loading address of a machine language consists of adding its regular address to the offset. If the regular address is 0, then the offset address becomes the loading address. It is up to the user to determine where this routine must end. Usually, machine language

routines are loaded in the top portion of memory, protected by the `CLEAR` command. An offset address must be used, in any case. Another point to this driver is that while all BASIC programs should work fine, machine language programs that choose to ignore RAM HOOKS will not work. The reason is simple — the program does not use the hook; therefore, there is no way that the program will know you have added the extra hardware.

If you use a higher level of software such as OS-9, with the proper driver, the parallel printer adapter would work with all OS-9 software. But, unfortunately, I know little about OS-9 drivers. If there is someone out there who knows enough about it and can write such a driver, send it to me, via `THE RAINBOW`. I'll check it out, and if it works fine, I'll print it in a future issue.

Figure 2 is another printer driver with a twist. It works only with the CoCo 3. You see, the CoCo 3 always works in the all-RAM mode. When you turn the computer on, it transfers all the ROM data into RAM. While it is impossible to write to ROM, it is possible to write to RAM. This routine is in two parts. The first part is one byte long and checks to see if the serial printer is ready. We don't need this with the parallel port; this byte defeats that routine. The second part is the printer driver itself. It is not very long — it does not need to be. First of all, it is loaded directly on top of the old serial driver. It does not need to be relocated in memory, nor does it need to be hooked into the RAM HOOK. Next, it does not need to check to see if the character is for the printer; if the software gets this far, it has already determined that it is for the printer. And finally, it does not have to deal with carriage returns, because the rest of the routine does that for you. Another advantage to this is that more machine language programs will work, because it is at the address normally taken up by the serial driver.

Again, some notes for this driver are necessary. The assembler I used for this routine allows for more than one `DRG` value. Many assemblers allow this, but the area in between must not be filled with zeros if your assembler does not allow it. You can poke the value into memory. Enjoy your parallel printing!

Beginners — Add an LED to Your Controller

By Tony DiStefano
Rainbow Contributing Editor

Thinking about the Princeton RAINBOWfest still excites me. If this RAINBOWfest is any indication of how the CoCo is doing, then long live the CoCo! This show was one of the best I've been to in a long time. The CoCo 3 seems to be doing very well. There were lots of new things for the CoCo 3 — both hardware and software. Look forward to seeing a few projects from me for the CoCo 3. I talked to a lot of people who read this column, and I would like to thank all my readers for their support, without which I would have stopped writing a long time ago.

Talking to RAINBOWfest goes gave me a few insights on the direction this column is heading. I received a lot of requests for "simple-to-do projects." Some people want to build something useful. Others say they want challenging projects. Well, why don't you send me your "Hardware Projects Wish List"? I'll look them over and make the ones I think other people might like. Send them to THE RAINBOW, with attention to me or "Turn of the Screw."

This month, as I promised several readers, I am doing a beginners project. In the past, I have done LED (Light Emitting Diodes) projects that have lit up just about everything on the CoCo. I even did a project that lit up different colors on your disk drive when you accessed different sides of your drive. Well, I'm doing another LED project,

one I saw done on a disk controller a long time ago and have not seen since: an LED to indicate when the disk controller is writing to the disk.

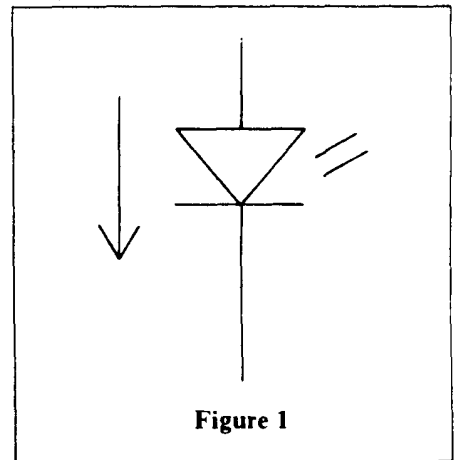
This is a simple project requiring a minimum amount of tools and parts. The parts are available at your local Radio Shack, and there are only two needed. The first, of course, is an LED. Radio Shack has lots of them. I suggest you buy one that comes with its own panel-mount holder, as it is easier to install. The other part is a resistor. That's it — a simple project that costs under a dollar.

Before I get into the construction of the project, let's look into the theory of the LED. Figure 1 is the electrical diagram of an LED. An LED, as the name implies, is first a diode. A diode is an electronic component that lets current flow in only one direction; let's call it the positive direction, which is shown by the arrow in Figure 1. The diode presents little resistance to the current flow. When the diode is conducting, it is said to be "Forward Biased."

In the other direction, the negative direction, the diode presents a high resistance. Current does not flow through the diode in the negative direction. When this happens, the diode is considered to be "Reversed Biased." When a light emitting diode is forward biased, it emits light. Quite simple, isn't it?

When an LED is forward biased, it conducts current. If we were to put an LED, forward biased, across the 5 volts found in the CoCo, it would cause trouble. The diode would act like a

short and cause the 5 volts to blow a fuse, as well as the LED itself.



We need something to limit the amount of current flowing through the LED. This is where the resistor comes in. Current flow is measured in amps. A typical LED can handle up to 50ma. The term "ma" stands for "milliamp." It means 1/1,000th of an amp. To have 50ma means to have 50/1,000ths of an amp, or .05 amp. Without getting into too many formulas, we want the LED to have about 10ma. The formula for calculating resistance from voltage and current is $R = V / I$, where the voltage (V) is 5 volts and the current (I) is 10ma. The resistance is 500 ohms. The closest value for this resistance that Radio Shack has is 470 ohms, which will do just fine. So, to recap the parts, you will need one LED with panel-mount holder and one resistor, 470 ohms ¼ watt.

Next, you will need some tools. Not many are required, but check to make sure you have them all before you start. There is nothing more frustrating than

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

starting a project and finding out that you are missing something. You need a soldering iron and solder, cutter/strippers, screwdriver (to match the screws that open your controller), and a drill and bit (to match the size of the LED mounting hardware). You will also need a few pieces of thin wire and electrical tape or shrink tubing.

Now we have all the parts and theory we need to start. It's time to get practical. In the controller circuit, there is an output that tells the disk drive hardware to go into the record or write mode. Like other signals that control the disk drive, it must reach the drive itself. This is done by the 34-wire connecting ribbon cable that plugs into the end of the controller. We will monitor this write signal with our LED. We want to hook up our LED so that it lights up when the controller is writing to the disk. The write signal is on Pin 24. This signal is available many places in the controller, but I chose this one because it is the only place common to all controllers, Disto, Radio Shack or any other.

When the controller is idle or reading, the level on Pin 24 is high, about 5 volts. When the controller is writing, the pin is low, or ground-level. We want to hook up our LED and resistor in such a way that the LED is on when this signal is low. Before reading on, think about it and try to design it by yourself. Does your design look like the one in Figure 2? If it does, reward yourself with a visit to the fridge. If it doesn't, study the circuit and see where you went wrong.

Here is the theory behind why I wired it up this way: As I stated previously, when the controller is reading, the signal is high (5 volts). The LED is also hooked up to 5 volts. Disregarding the resistor, if a diode (or our LED) has 5

volts on both sides, it cannot have any current flow. Therefore, the LED is off.

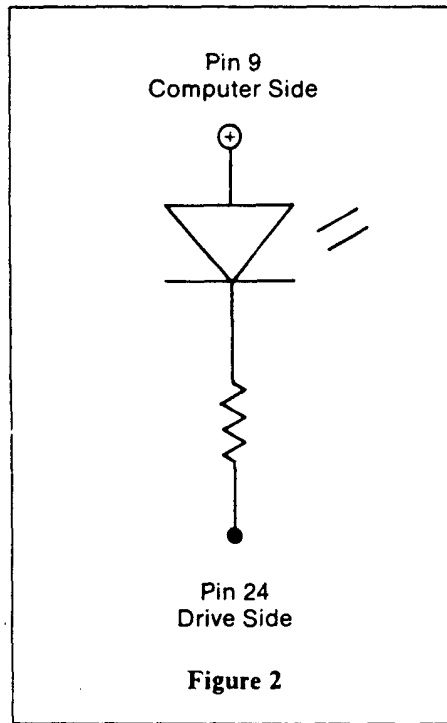


Figure 2

volts on one side and ground on the other side, it becomes forward biased and conducts. Therefore, the LED is on and shines brightly.

OK! Time to start constructing. Turn the computer off and remove the controller from the computer. Remove the controller's cover using the proper screwdriver. Locate Pin 24 on the connector that connects to the drives. Locating this pin may be a bit of a pain. On the top part of the connector are all the odd numbers. On the bottom part are all the even numbers. So, unless you can see where the pin leads, you will have to remove the controller from the bottom part of the case to get to Pin 24.

If you are looking at the bottom part of the connector, and the connector is pointing upward, Pin 2 of the connector is on your right. Count by two toward the left until you reach 24. Solder a piece of wire to that pin. Make sure you solder at the base of the pin and not at the tip. You will not be able to plug in the connector, otherwise. Run the wire out over the side of the controller and replace the bottom cover. Make sure the wire is long enough to reach the LED. Cut the ends of the resistor to leave just enough room to solder. Solder the other end of the resistor to the short lead of the LED. Now solder another short piece of wire to the long end of the LED.

At this point you must find 5 volts somewhere. One place where I know that all controllers must have 5 volts is at the connector that plugs into the controller: on Pin 9 of the connector. It is on the top this time. Pin 1 of the computer connector is on the same side as Pin 1 of the drive connector. Solder the wire that comes from the LED to this pin on the computer side of the connector. That is all the soldering you have to do. Use black tape or shrink tubing to hide all of the exposed wires, including the resistor.

The only thing you have left to do is mount the LED. Find a suitable place on the cover to mount it. But you have to be able to see it when the controller is plugged into the computer, and the back side of the LED cannot touch the controller. If you have a Multi-Pak, you may want to make your hole on the end of the controller, so that the LED will be pointing up when it is plugged into the Multi-Pak.

Now close up the cover, and test it out. Set up your system and turn it on. Make sure you get your normal message. Put a blank or otherwise "non-useful" disk in the drive. If this circuit doesn't work right, you don't want to destroy a good disk. If all is OK, try entering DIR. The LED should not come on. If all is OK, try using DSKINI to format the disk. The LED should go on and blink for every track the controller formats. If the controller formats the disk properly and the LED works, all is OK. If not, go back and check your work. If you cannot find anything wrong, try reversing the LED. It may be in backward.

Enjoy your new LED. I hope you have learned a little more about the hardware in your computer. Till next time. Don't forget to send in that "Hardware Projects Wish List." ☺

In the September 1987 issue of RAINBOW (Page 150), I wrote an article on how to build an EPROM emulator using a RAM chip backed up with a battery. That was all well and good, and I thought that was the end of that. But it wasn't. A reader called me up and told me about his problem with the emulator — he had a problem erasing it.

A regular EPROM has a specific method of erasing — you need an EPROM eraser. All EPROMs have a window on top that allows you access to the chip's memory cells. Exposing that window to ultraviolet light erases all data in the EPROM.

When an EPROM is new, and every time you erase it, the EPROM memory cells contain "logical 1"; or, in the case of an eight-bit EPROM, a Hex value of \$FF (that is, eight logical high levels). When you program an EPROM, the logical 1 changes to a logical 0. And there is only one way the programmer can change that cell back to a logical 1 — use an eraser.

Since the chip I used was a RAM instead of an EPROM, my EPROM emulator had no window. You could not erase it with an EPROM eraser, but that did not seem to be a problem. Unlike an EPROM, a RAM chip cell can be changed to a logical 1 just by writing to it. In most cases, all you had to do was plug the RAM-based EPROM emulator and run the programmer software. No problem, the emulator was programmed.

There are always exceptions to the rule. In sync with today's world of "faster is better," the people who wrote EPROM programmer software were looking for faster ways to program an EPROM, each cell of which has to be programmed separately. Each EPROM cell takes a small fraction of a second to program, which does not seem like a very long time; but with EPROMs getting bigger and bigger, those "fractions" add up, and it takes longer and longer to program them.

The software experts thought of one way to shorten the programming time: Since an EPROM contains all \$FFs when it is new and just after it is erased,

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

Build an . . .

Electronic EPROM Emulator Eraser

By Tony DiStefano
Rainbow Contributing Editor

why not use that fact when programming? So, when the software is programming an EPROM, it first checks to see if the present data byte to be programmed is \$FF. If it is, the software doesn't bother to program that byte, as it is already an \$FF on the EPROM. Instead, it goes on to the next byte. The more \$FFs there are in the data to be programmed, the faster it goes. Makes sense, doesn't it? Right! To further aggravate the problem, some EPROM programmers check for \$FFs and won't even start if your EPROM isn't right.

Now, that is a problem. You can't erase the EPROM emulator with an eraser, and you can't program \$FFs into it. You can't even unplug the battery to let the memory "forget"; that would make the EPROM emulator all zeros. What are you to do? Well, here is the answer. Build an Electronic EPROM Emulator Eraser. Wow, what a mouthful! But it will solve your problem.

Building It

To start with, you will need the standard tools you usually use on a project: soldering iron, solder, cutters, screwdrivers and the like. The parts list shows you what you will need. Some of these parts are not available at your local Radio Shack store, but they

should all be available at a good electronics store.

Note that this project does not have to be plugged into a CoCo to work, and is completely self-contained. However, it does need a 5-volt supply. If you build it on a CoCo-compatible proto-board, you can get 5 volts from the CoCo's power supply. The 5-volt supply is available on Pin 9, and ground is on Pin 33.

First, let's start off with some theory. The EPROM emulator is mainly a RAM chip, so let's review our knowledge of RAM chips. Basically, this RAM chip has 13 address lines (A0 to A12), eight data lines (D0 to D7), one read/write line and some ChipEnable lines. Since this chip is emulating an EPROM, all lines are about the same except for the read/write line. It changes to the program (PGM) line. What we have to do is program the chip for \$FFs, so all DATA lines are tied to Vcc, which is 5 volts and logical 1, using the PGM pin (See Figure 1) to strobe this data (always \$FF) to the chip.

Every memory location has to be programmed this way. The easiest way to access every location is to do them in sequence, one at a time. For that you need some binary counters. Two of them will have enough bits to cover all addresses. In fact, if you study Figure 1, you will see the counters I am using are 74LS393. Each of these packages have two 4-bit counters. I am using two chips to give us a total of 16 bits. That is more than enough for us to use.

Setting up these counters is quite easy. The last bit of the first counter, QD (most significant bit), connects to the clock of the next, and this is repeated two more times to include all counters. The clock to the counter comes from a free-running clock. The LM555 is a versatile timer that can be used as a "one shot" or resettable timer, but I am using it as a free-running timer. That means that the output clocks high and low continuously, which is necessary in our case.

So, the output of the 555 is connected to the clock of the first counter. The clear (CLR) of the counters and the reset (R) of the 555 are connected together to an RC constant, which is just a capacitor that charges through a resistor. When you first turn the power on, the cap is discharged. Therefore, the

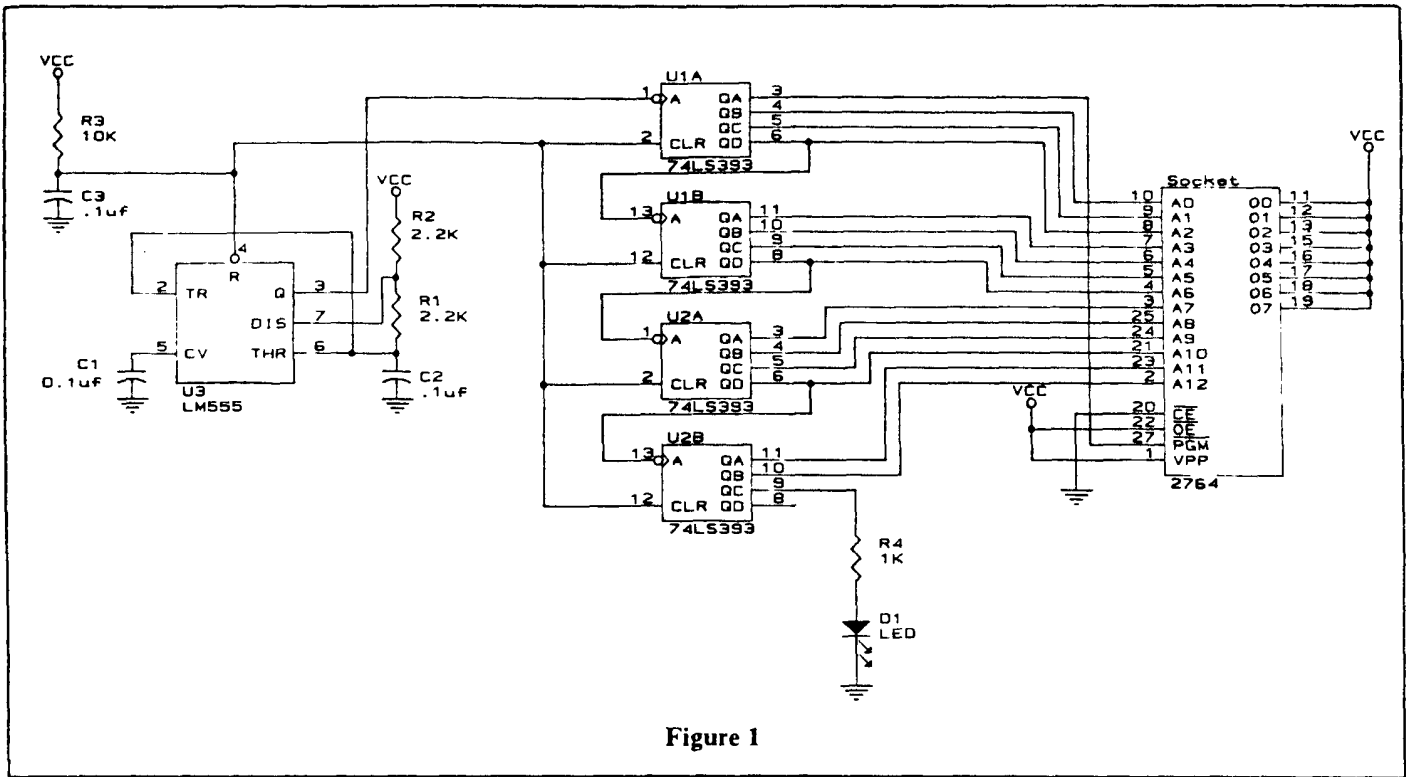


Figure 1

555 and the counter are held inactive. When the cap charges to 5 volts, it activates the 555 and counters. This is done in order to give the power supply time to stabilize and to make sure that all the chips are properly powered before starting. It also clears all the counters to zeros.

Once the power is stabilized and the reset releases, the 555 starts to clock. That starts the counters. If you notice, the first bit is connected to the PGM pin of our 2764 socket. That programs (or pulses) the data (\$FF) into each chip. The next 13 bits of the counter are connected to address lines. It should be clear to you by now that all address lines have to be used.

The next bit on our counters is connected to an LED and a resistor. Last month I covered the theory on LEDs, so everyone should be up on it. This LED is used as an indicator to tell you that the process is finished. If you let the process continue, the LED will go off again and then on again. This will not hurt the chip, but it is not necessary to do it twice — once is enough.

Constructing the project is not too hard. It is recommended that you use sockets for all the chips. Use a 28-pin, ZIF socket for the 2764. If your budget does not allow for one, use a good quality socket, at least. Some of the cheap sockets are good only for one or two insertions. Figure 1 shows all connections except power and ground.

The following is a power and ground connection list for this project:

Chip Number	Power (5v)	Ground
U1	14	7
U2	14	7
U3	8	1
2764	28	14

It would be a good idea to run a few tests before you plug the EPROM emulator into the eraser. Turn the power on and check with a digital probe or meter to see if the 555 is working and if all the address lines are clocking. You should also see if the LED lights up after a while. Check for the proper 5 volts and ground on the 2764 socket. That should be all there is to it.

Erasing an EPROM emulator is simple. With the power to the eraser

turned off, insert the EPROM emulator. Turn the power on until you see the LED go on. Then turn the power off and remove the EPROM emulator. And that's that. □

Part	Description
U1	74LS 293
U2	74LS 293
U3	LM 555
C1	.1uf 25 V
C2	.1uf 25 V
C3	.1uf 25 V
R1	2.2K ¼ W
R2	2.2K ¼ W
R3	10K ¼ W
R4	1K ¼ W
D1	Red LED

Parts List

A long time ago, when computers for the consumer were just starting to come on the market, large amounts of memory were unheard of. My first computer was a Sinclair ZX-80. It had only 1K of Random Access Memory, or RAM.

RAM is a temporary storage place for data — as long as the computer is on, RAM will remember what is put into it. When you first power up a computer, RAM has no set pattern. The data in it is not valid data. When you turn the computer off, all RAM data is lost.

Anyway, imagine only 1,024 bytes of memory, and half of that used for video display — a far cry from our present CoCos. BASIC was in Read-Only Memory, or ROM, and that was a whopping 4K ROM at that. Later, they came out with 8K of ROM, which was a big improvement.

ROM is memory that has been permanently etched into the chip at the factory. It cannot be changed or lost. When you power up with ROM, instant data (or a program) appears. Every computer needs a bit of ROM (no pun intended). How much is a “bit”? Well, that all depends on what that ROM has to do.

When a computer is first powered up, a hardware reset line delays the start of the CPU until the power supply is stable. Then, when the reset line lets go, the first thing the CPU does is load a starting address from a predetermined area of memory. It loads this address into its program counter and then starts to execute the code pointed to by this program counter. Now, what is wrong with this picture? If this area of memory is RAM, we're in trouble. On power-up, RAM has no definite pattern; the CPU would certainly get confused and hang up. But if ROM were there in place of RAM, then the CPU would see valid code and run merrily on its way. Hurray for ROM!

ROM is great — instant software, and no way to lose it. But for hackers like you and me, ROM is a downer. Why? For the same reason that makes ROM great — it locks us in. It cannot

Bigger and Better Eproms

By Tony DiStefano
Rainbow Contributing Editor

be changed. The code that is in a ROM is for keeps.

The manufacturer of ROMs saw a need for the user to be able to program his or her own ROM. From that need came the PROM. The PROM is a Programmable ROM. In other words, a PROM is a blank ROM. A special device lets you program your own data code into the PROM. That was great, but if you made an error in your code, you had to throw that chip away and start with a new one. The chip was fine for small runs of a proven code: It had all the advantages of ROM and none of the high costs of mask programming a ROM.

But there remained a need for a reusable chip that was easy to program. The EPROM was introduced — an erasable PROM. Just what the doctor ordered. Easy to use, inexpensive and able to be used over and over again. When I first started learning about computers, I wanted to customize mine. When I turned it on, I wanted it to say “HI TONY.” It was that desire that made me want to learn more about EPROMs.

Back then, the most capacious EPROM I could find was only a 2K by 8-bit EPROM. Its part number was 2716. The “16” represents the number of bits in that chip. There are 16K (16 thousand) bits. Most microprocessors then were only eight bits wide, so

EPROMs were also eight bits wide. Dividing 16,000 bits into 8-bit-wide bytes gave us 2K (2,000) bytes of memory. But that was then, and this is now. As technology improved, so did EPROM capacities. After the 2716 came the 2732. Yes, you guessed it, the 2732 has 32K bits or 4K by 8 bits — twice the capacity of the 2716.

Still improving, technology then allowed for a reasonably priced 2764. To me that was the breakthrough, a 64K bit EPROM and 8K to play with. This was great because it was the same size as the BASIC, Extended BASIC and the Disk BASIC ROMs to EPROMs. I was able to customize these ROMs with EPROMs.

Things didn't stop there. The prices for these EPROMs started very high, but soon dropped very fast. Again, the industry came out with another EPROM — another doubling of capacity. Yes, a 27128, a whole 16K of data in one chip. Impressive as it was, it did not stop there. Next came the 27256 and then the 27512. The 27256 is a 32K EPROM and the 27512 is a 64K EPROM. Just think of it. The 6809 CPU inside the CoCo can access 64K of memory — that is the whole 6809's memory address in one chip! If you think back to the 2716, it would take 32 of these memory chips to make up the capacity of one 27512. I know that manufacturers are making 231024s, which are 128K by 8-bit ROMs (but I don't think they have them in EPROMs — just yet, anyway).

The Project

What can you do with these bigger and better EPROMs? Well, I have a few ideas. The easiest place to put EPROMs

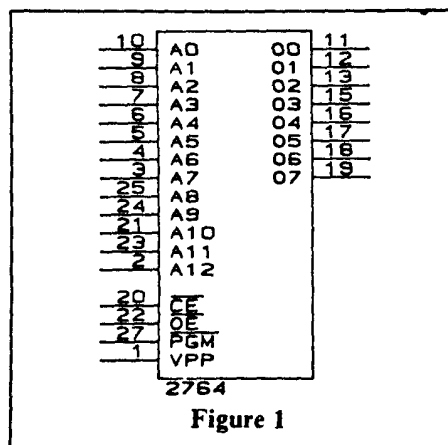


Figure 1

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

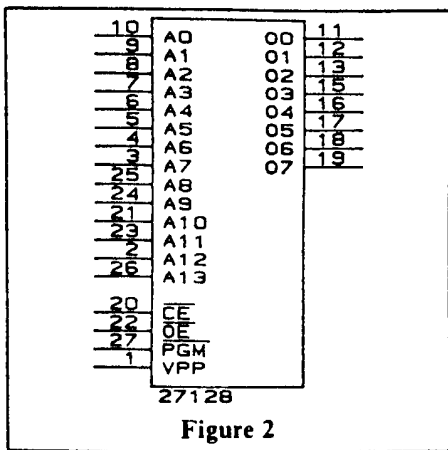


Figure 2

is in the Multi-Pak. And the easiest place to map them is in the Disk BASIC area, located from \$C000 to \$FEFF in the memory map of the CoCo 1 and CoCo 2. With the CoCo 3, you are a little bit more limited. The mapping is from \$C000 to \$FDFF, just one page less only 256 bytes at the top of the memory map. That is to accommodate the extra functions of the GIME chip. Anyway, for all intents and purposes, this area is 16K long. Just remember the top two pages are not usable.

Look at Figure 1, a pinout of a 2764. I started there because I figure it is the smallest memory chip (8K long) that is worthwhile hooking up. Accessing this amount of memory requires 13 address lines, A0 to A12. The CTS pin on the CoCo's bus accesses a total of 16K, requiring 14 address lines to properly decode. This leaves us with one address line left over. In this case, we can't use it. Leave it unconnected. This will cause a memory mirror. If the CPU accesses the first half of the 16K memory area, it gets the data. When it accesses the second half of the memory, it gets the same data. The only difference is that the last address line, A13, does not control anything. Such is the case of the Disk BASIC ROM in the Radio Shack Controller; it is only 8K long and is mirrored to the second half of the 16K area.

Now look at Figure 2, the pinout of a 27128. It has 14 address lines, making it 16K long. It is a perfect match for the CTS area of the CoCo. There are no leftover address lines. The CPU can access a full 16K of memory with no memory mirroring.

Figure 3 shows the pinout for a 27256. This one has one more address line than we can handle. That is the number of address lines it requires to access 32K. This presents a problem. The CTS cannot handle 32K, and we have one address line left over, with nowhere to connect. Figure 4 is the

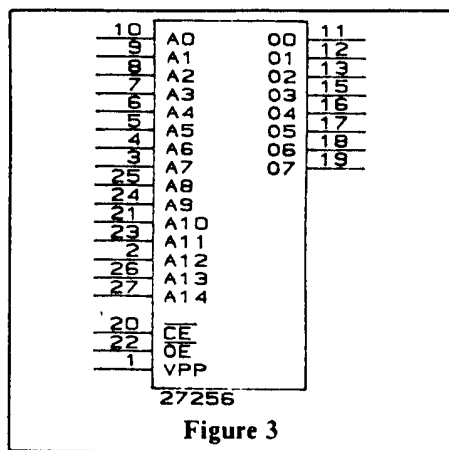


Figure 3

pinout of a 27512. It has double the problem, with yet another address line we don't know what to do with.

The 27256 represents 32K of data, but that is just one way of looking at it. Another way of looking at it is as two banks of 16K. For example, let's say you have two pieces of software that are each 16K long. You can put both of them on one 27256 and select which you want to use when you turn on the computer. This can be done quite simply.

Figure 5 shows a small (I mean *small!*) circuit that can select between the two banks of a 27256. It consists of a single pole, single throw switch and a resistor. The resistor acts as a "Pull Up." When the switch is in the off position, current is fed from the 5-volt supply to the address line via this resistor. The XX means whichever address line is connected to it, making the address line a logic level of 1, or HI. When the switch is on, the current is shorted to ground, making the address line in question a logic level of 0, or LO. The switch and resistor become your manual bank selector. When this circuit is connected to A14 on a 27256 and the switch is on, you get the first half of the EPROM. When the switch is off you get the second half. So, when you turn the computer on, it will see one or the other. If you happen to turn the switch when the computer is on, chances are the computer will get confused and hang up. However, this does not hurt the computer.

If you are thinking of using a 27512, you can have four banks of software, each bank 16K long. In that case, you have to build another circuit like the one in Figure 5. Connect the second switch to A15. When both switches are on, you get the first 16K bank of software. When the A14 switch is off and the A15 switch is on, you get the second. When the A14 switch is on and the A15 switch is off, you get the third bank. When

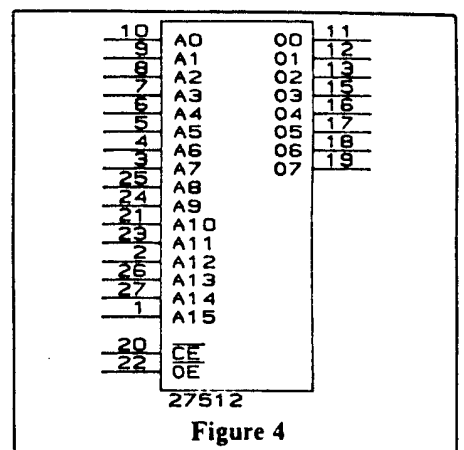


Figure 4

both switches are off, you get the last bank of software.

So far, the switches have been switching 16K banks of data. If most of your software is in 8K blocks or less, you might want to switch these EPROMs in 8K banks instead of 16K banks. You will need yet another circuit like the one in Figure 5.

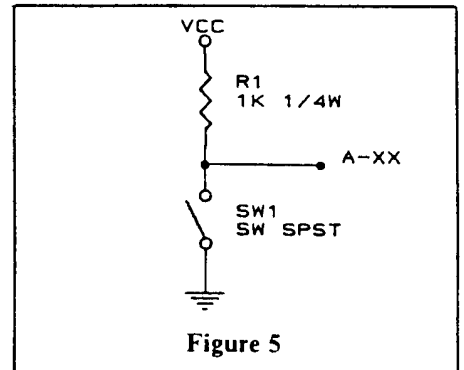


Figure 5

In either the 27128, 27256 or the 27512, disconnect A13 from the computer side. Connect A13 to the third switch. This switch now controls 8K banks. When the switch is off, you are seeing the first, or lower, 8K bank of data at \$C000 to \$DFFF. When this switch is on, you see the second, or upper, bank also mapped at \$C000 to \$DFFF. When you use a 27128, you get two 8K banks. A 27256 gets you four banks, and a 27512 gives you a whopping eight 8K banks of software. Remember, though, that each one of these banks starts at memory location \$C000 and that for this software to work properly, they must be written for this area or be in complete relocatable code. Also remember that to autostart software that begins at \$C000 you must short out Pin 7 and Pin 8 on the CoCo bus. Software that looks like a DOS must have the first two bytes the same as RS-DOS in order to function properly and be recognized by BASIC. DOS-like software must not have pins 7 and 8 shorted.

Last month we talked about high-capacity EPROMs from 8K (the 2764) all the way to 64K (the newest member of the family, the 27512). We also talked about hooking up these chips to your CoCo.

The 2764 and the 27128 (16K) can be hooked up directly to the CoCo. The 2764 is easy to hook up, as it has only 8K of memory. In any ROM chip, only the first 8K of memory is valid — the rest of it is memory mirrored.

As we get into more memory per chip, we are faced with more options. For example, do we use the 27128 as one complete 16K package, or do we split it up into two packages of 8K and select between the two? Do we make it selectable in software or in hardware, or both? Look at the Radio Shack Multi-Pak, which does all of this.

All these possibilities can be overwhelming to a novice hardware hacker. For that matter, the ramifications of EPROMs aren't all that clear to the pros, either. What I am attempting to do is take a close look at large capacity EPROMs and describe how to hook them up to the CoCo and have a big enough EPROM package to make a ROM disk.

The CTS pin can access up to 16K on CoCos 1 and 2, and can access up to 32K on the CoCo 3. But because only the CoCo 3 can access 32K, I am limiting my possibilities to 16K — otherwise, we'd have to throw in another variable, which would only add to the confusion. Besides, the 32K mode of the CoCo 3 is rarely used, if at all.

Let's start with something we are already a little familiar with, DOS. Now, the DOS ROM that Tandy offers is called Disk Extended BASIC. While we are on the subject of DOS, let me clear up a little misconception. There are only two versions of this DOS — the older 1.0 and the newer 1.1. When you power up a CoCo 3 with a Tandy DOS in the controller, you see one of two messages: If you have Tandy DOS 1.0, you get the message "2.0"; if you have Tandy DOS 1.1, you get "2.1." You see, the '2' part of the version belongs to the

A DOS expansion project for experienced hackers

Build a Half-Megabyte ROM Disk

By Tony DiStefano
Rainbow Contributing Editor

version of Hi-Res BASIC you have, not to DOS — the ".0" or ".1" part of the version belongs to DOS. I just thought I would clear this up because I hear too many times that someone has DOS Version 2.1.

Well, back to work. Whatever the version, Tandy DOS is contained in an 8K ROM. ROMs are masked at the factory and cannot be changed, but EPROMs are user-programmable. You can change them any time you want, as they are erasable. I think I have said enough about the structure of EPROMs. Read last month's article for more details.

There are a lot of people who are familiar with DOS and would like to expand it — add in their favorite utility, for example. To expand DOS, you need more memory space, so the only thing to do is change to a 16K EPROM. That gives you about 8K of extra space to work with. To have more than that requires more space.

This is where you have to start with bank switching. Bank switching means that you have more than one memory chip mapped in the same area, but only one of them is active at a time. Last month we looked at a technique that required a hardware switch to physically change the access to the EPROMs. That is a simple technique, but there are some limitations, the biggest one being that the software will most likely get lost

and cause the computer to crash when you switch it. It is OK when you want to completely change and power down anyway, but not too practical when you have a lot of software already loaded and need just a little utility.

Preventing a crash that may occur when you turn the switch is not too difficult if you know how. Have the CPU turn the switch for you — this is called a "softswitch," and requires a latch, some decoding and a circuit. With this latch we can switch between quite a few things. What I want to show you is a way to access eight EPROMs of varying sizes. Figure 1 shows the circuit required to wire up eight 27512 EPROMs. That gives you a total of half a megabyte of EPROMs, or, in other words, one big ROM disk.

Before you run out and buy all the parts and try building the ROM disk, keep in mind that this is one heck of a big project. A project that should not be tried by everyone. First of all, you must have a lot of patience — to solder eight 28-pin EPROM sockets takes many hours. Second, you must have lots of money to buy eight 27512 EPROMs. In addition to a disk drive, you must also have a Multi-Pak. And lastly, you must have a lot of knowledge about machine language drivers for disk drives. So, you see, this is a big one. If you have all the prerequisites, let's start.

The first thing to do is get acquainted with the circuit. U1 simply gates the SCS with the Read/Write line. All this does is prevent you from switching the data in the latch just by reading that memory area. So, this becomes a "Write Only" byte. Since it uses the SCS pin, this byte is mapped at \$FF40. In fact, it is mirrored from \$FF40 to \$FF47. U2 is a six-bit latch. The diagram says that the inputs are from D1 to D6, but they are in fact connected from D0 to D5, respectively. The latch is connected to the output of U1 and is cleared to all 0s when the reset button is pressed.

The output of U2 is six bits that are controlled by writing to it. Let's look at the last three bits first, Q4, 5 and 6. They go to the inputs of a 3-to-8 decoder. These three pins select one of eight outputs. The other inputs to U3 are the CTS pin and the E clock. The E clock is needed to make sure the data is in sync with the CPU.

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec.

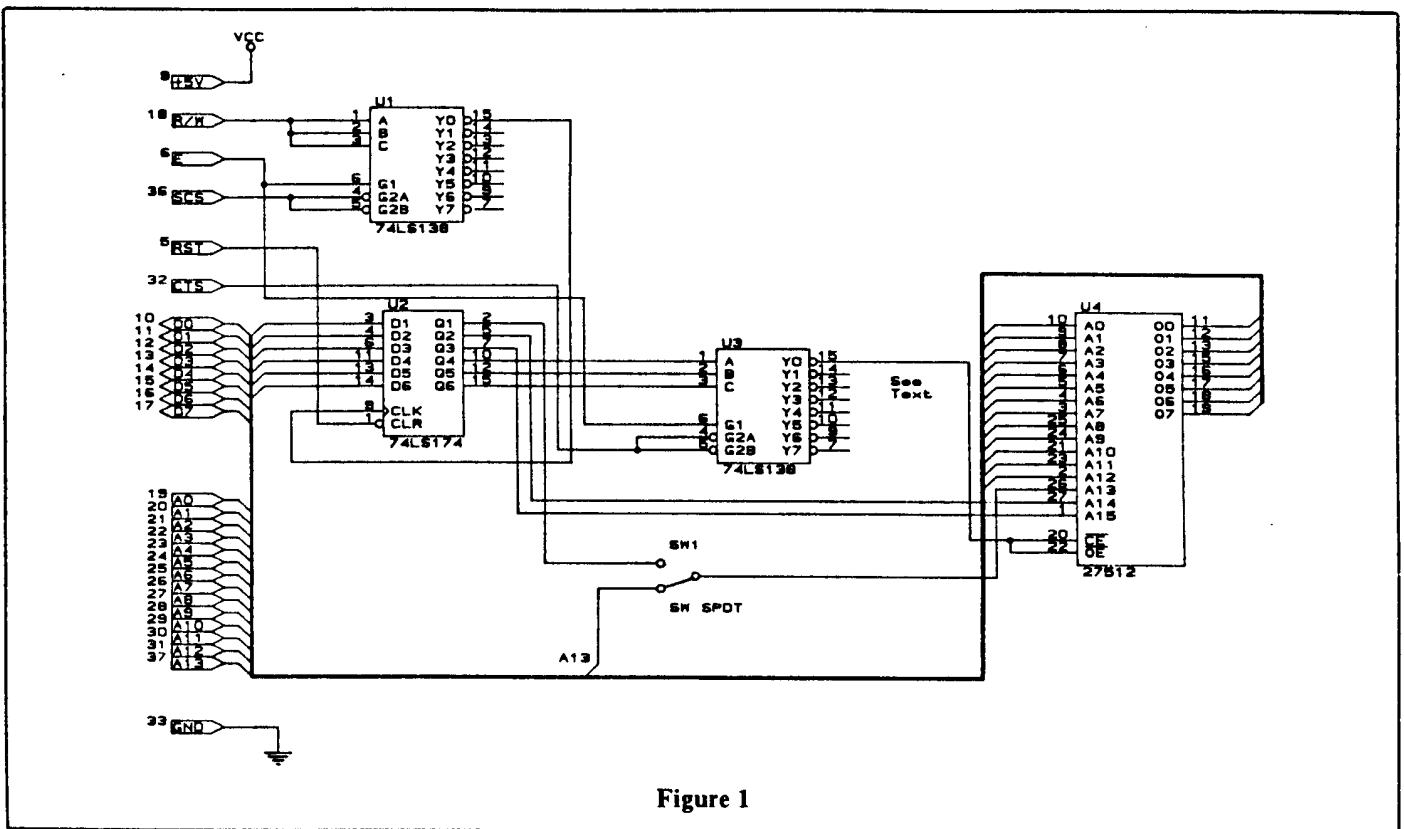


Figure 1

The CTS pin is the main select for the external ROM area. The output Y0 is connected to the Chip Enable and Output Enable of U4. Only one EPROM is shown. The other seven EPROMs are all wired in parallel except for these two pins. Y1 to Y7 of U3 connect to pins 20 and 22 of chips U5 to U11, respectively. Wow! What a mouthful! Depending on what the binary number is at the A, B and C inputs of U3, one of the eight EPROMs will be selected when the CTS pin goes low.

Now let's look at the next two bits, Q2 and Q3 of U2. They connect to A14 and A15 of all the EPROMs. If you put on your binary thinking hat, you'll realize A0 to A13 comprise 14 address lines. Two to the power of 14 gives us the amount of data 14 address lines can access — 16K. These two bits that are connected to the EPROMs select four banks of 16K. A 27512 has 64K of memory. These two bits connected to A14 and A15 will divide the 64K EPROM into four banks of 16K. OK, here comes the tricky part. A13 of the EPROMs can be connected to one of two sources via SW1. The way it is connected in Figure 1 is the way it is required to switch 16K banks. Each of the eight EPROMs has four 16K banks; that gives you 32 16K banks of memory.

There is another way to wire things up. When the switch is turned the other

way, it no longer gives you 16K banks. With one less address line to work with, the CPU will see two 8K banks mirrored with the same data. By putting this address line to another bit (Q1 of U2), we now have three bits of bank switching. In binary, three bits give you eight banks to choose from. You now have eight EPROMs with eight banks each, which gives you 64 banks of 8K of memory. That's a total of 512K of memory.

Well, that about does it for the theory part. The construction of the ROM disk, like I said before, is a big task. You will need eight 28-pin sockets for the EPROMs and three 16-pin sockets for the other support chips. The best way to go with this one is to get the proto-board from CRC Inc. That is the one I used, and it has plenty of room for all the chips. Also needed for this project are eleven .1 uf capacitors, one for each chip; connect them between +5V and ground as close to each chip as possible. Not shown on the diagram are the +5V and ground pins for these chips. It is simple. For the three TTL chips, the +5V pin is 16 and the ground pin is 8. For the EPROMs, the +5V pin is 28 and the ground pin is 14. That is all you need to know to construct this board.

Now that I've shown you the hardware part of this project, it's time for the software. You all know how much I hate that. But, without software, hardware

would not be much good. Though I will not be writing any software, you will need to know something about the hardware to write it yourself. The control byte, as I call it, for which bank is active in this circuit is at \$FF40.

There are two different ways the control byte works, depending on which way the switch SW1 is set. The two options are this — 32 16K banks and 64 8K banks. For the option of 64 8K banks, D0, D1 and D2 of the control byte select eight banks per EPROM. D3, D4 and D5 select one of eight EPROMs. So, U4 has bank numbers 0 to 7, U5 has 8 to 15, U6 has 16 to 23, and so on. Each bank will appear from \$C000 to \$DFFF. The 16K banks are a little different. D0 is not used; D1 and D2 select four 16K banks; and D3, D4 and D5 again select one of eight EPROMs. This time U4 has bank numbers 0 to 3, U5 has 4 to 7, U6 has 8 to 11, and so on.

The choice to use 8K or 16K banks is yours, of course, but think of this: If you use 16K banks, you lose 256 bytes per bank in CoCos 1 and 2 and 512 bytes per bank in CoCo 3 because of the addressing of the CoCo. Those bytes are reserved for I/O.

I hope that I have given you enough information to think about and act on. It is a big project, but for the right people, it can be quite rewarding.

Back in February '85, I wrote an article describing how the Tandy Multi-Pak worked. I followed that up with a project involving a little circuit that could decode the latched bits and drive some LED digits to tell you which slot was active. Since there were two active areas of memory available in the Multi-Pak, you needed two LED digits and two driver chips. It worked well for a time but, as always, Tandy likes to throw some curves — they changed the insides of the Multi-Pak.

In order to make the Multi-Pak less expensive to make, and therefore less expensive to buy, they took many of the chips, grouped them together and made one big custom chip. This was great for them, as the price for the Multi-Pak went down and they sold more of them. Good for them, but not so good for my circuit — it no longer worked. Those latched bits I used to get the data to drive the LEDs are no longer there.

When this new Multi-Pak came out in '86, I got a few letters asking if there was anything I could do. At the time I thought it would be too much trouble to redesign the circuit, too many chips and too much work for it to be worthwhile to build. But lately I've had some calls about this one again. So here goes another big project.

Reviewing the Multi-Pak

Let's recap what was said in that article. The first half of the article described the functions of the Multi-Pak and the second half described how to hook up LEDs to tell you which slot is active. The two active areas in the Multi-Pak are the CTS and the SCS areas. The CTS is mapped from \$C000 to \$FEFF for CoCos 1 and 2 and from \$C000 to \$FDFF for the CoCo 3. The SCS is mapped from \$FF40 to \$FF5F on all three CoCos. These mapped areas can be switched to any of the four slots of the Multi-Pak, together or independently. That means you can have the CTS in Slot 4 and the SCS in Slot 2 if you want.

These memory areas can be switched both in hardware (via the switch on the front of the Multi-Pak) or in software

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouesi, Quebec.

Using LEDs to see which slot on your Multi-Pak is active

Multi-Pak LED Update

By Tony DiStefano
Rainbow Contributing Editor

(via one memory location). The switch is simple to operate; before turning the computer and Multi-Pak on, slide the switch to the desired slot number. When you turn the computer on, the active area (or slot) is identical to the slot number on the switch in front. The hardware switch cannot change the SCS and CTS separately, only both of them at the same time. Sliding the switch to another slot number with the power on will change slot access, and probably crash your software program at the same time. There is, however, a time when the switch no longer works to switch these areas.

Let's go back to the software switch. It, too, can change the active slot area; it does so by writing a number to a read/write byte in the memory map, the byte at \$FF7F. This byte is divided into two nibbles (a nibble is four bits); the lower nibble controls the SCS area and the upper nibble controls the CTS. A value of 0 to 3 in any of these nibbles selects slots 1 to 4, respectively. One interesting thing this memory byte does, once you've written to it, is lock out the hardware switch. Sliding the switch on the front panel does not work after you have changed the slot access from software.

Now, both the newer and older Multi-Paks do the same thing. But because we don't have access to the latched bits on the newer board, the project I did back in 1985 will not work

on the new Multi-Pak. The circuit I have come up with now works just like the older Multi-Pak's circuit, allowing you to hook up the LEDs as before.

The Project

This circuit requires just five chips. These chips can be mounted on a small PCB that you can get from any Radio Shack. It does not require an edge connector, because it does not connect to the slots of the Multi-Pak. Instead, you have to open up the thing and insert this circuit inside. This is not too bad, because you have to open it up anyway to get the LEDs in there.

Figure 1 shows the circuit in question. A step-by-step description will help you understand it. Let's start with U1 and U2: These chips are used to decode the memory map into one byte, Byte \$FF7F, which is 16 bits long. Out of these 16 bits, 15 of them are 1s and only one of them is 0. The 74LS133 (U1) takes care of 13 of them. When all of these are high, the output goes low. This output goes to U1, a 74LS138, where the E clock, read/write line and the rest of the address lines are decoded. Only one output is used to write to the 74LS173, which is a four-bit latch used to record what slot is active.

The 74LS368 is a six-bit buffer (we will use only four bits, however) whose input comes from the switches on the front of the Multi-Pak. You can get these signals from the 64-pin chip inside the Multi-Pak. The A pin in Figure 1 connects to Pin 21 of IC 6, and B connects to Pin 22 of the same chip. These two signals are split to form the four bits necessary for the LEDs. This is where the four connections of the LEDs project connect to. Here is the connection list for these pins:

Pin No. U13 of LEDs Project

C	2
D	3
E	14
F	13

This chip (U5) will output the status of the front switch when the Multi-Pak is first turned on, due to the U4, a 74LS74. This is a D-type flip-flop. On power-up or reset, the Q (Pin 5) output of this chip is low, which activates U5. At the same time, *Q (Pin 6) is high,

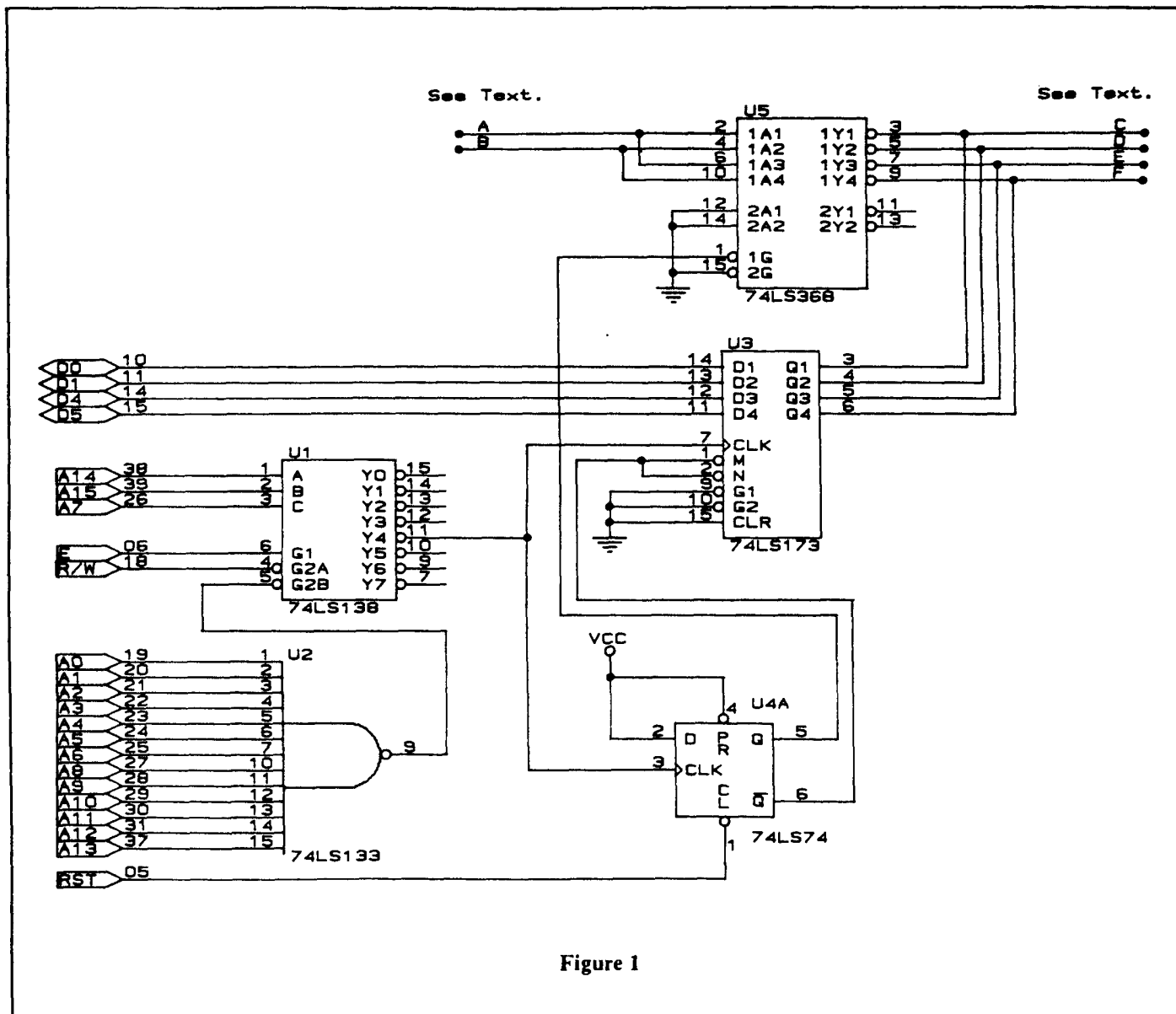


Figure 1

which keeps U3 in activated (tri-state). U3 has to be kept in this state because no data has been assigned to it; that would give random values to the LEDs. On the other hand, U5 is activated to give the status of the switch, which conforms to the old Multi-Pak.

The output of U1 also goes to the clock input of U4, so when your software program writes to \$FF7F, to change the active slot for the first time, it flips the outputs Q and *Q. This, in turn, deactivates U5 (connected to the switches) and activates U3. The new values just entered into the latch at U3 are now valid, and the flip-flop action of U4 brings this data out to the LEDs. From then on, changing the switch has no effect on the LEDs. The switch will have no effect until one of two things happens: Either a reset occurs or the power is turned off. Pressing the reset button will again flip U4 back to its

original state and therefore re-enable the switches. Turning the power off also flips the condition of U4.

This project for the newer Multi-Pak is not very difficult, but you must have done (or do now first) the project from 1985 for this one to be useful. The standard project builder's kit is necessary. These parts are not available from Radio Shack, but are at most well-stocked electronics shops. Active Electronics is my best source for almost all the electronics parts I buy.

There is one more thing yet to do; The program I use to generate the circuit diagram in Figure 1 does not put in the pin numbers for 5 volts and ground. Figure 2 shows a list that explains which pin goes where in the power and ground department.

In the Multi-Pak, you can get 5 volts from Pin 9 of the connector and ground connections from pins 33 and 34.

Chip #	+5 Volts	Ground
U1	16	8
U2	16	8
U3	16	8
U4	14	7
U5	16	8

Figure 2

In my January 1988 column (Page 144), I requested that my readers send in a hardware projects "wish list." I have gotten a few responses.

Some have been good, and I will get to work on them, but some are a bit far-fetched. Try to keep your ideas limited to small projects — some guys asked to do a project that would cost several times the price of the computer, the Multi-Pak, my drives and then some!

CoCo 1 and 2 users, remember when the CoCo 3 came out? One of the good things built right into the CoCo 3 is an 80-character-by-24-line display screen. I guess you must have felt left out in the cold. If you have a CoCo 1 or 2 with a Multi-Pak and are using OS-9, your luck will change; you too can now have an 80- by 24-character display.

I say this is a big project not because it is hard to build, but because the overall project will take a bit of time and some hardware considerations. For instance, if you want an 80-by-24 display, you must have a monitor capable of displaying 80-by-24 characters. That requires an RS-70 compatible monitor of about 20 MHz resolution. You will also need a Multi-Pak and software. (You know what I think of software — leave it to the programmers.) CRC will send you OS-9 Level I Version 2 software to drive this display for about \$10.

In the old days, an 80-by-24 character display required many chips, starting with the display chip. For many years, the most common display chip was the Motorola MC6845, back then a very powerful chip. It had a bunch of registers and counters that would divide a high-frequency clock into two lower frequencies. The higher of the two was the horizontal frequency, usually about 15 KHz; the other was the vertical frequency, at about 60 Hz. Also coming from this display chip was a character address. Part of this address went directly to a character-generator ROM. Now, a character-generator ROM is nothing more than a ROM with bit-mapped graphics of what letters and numbers are made of.

The other part of the character address went to the address lines of one side of some dual-ported RAM, which was ordinary RAM with some extra circuitry allowing two devices to read and write data to the same RAM. Also included in this circuitry was a circuit to switch between the two.

The other side of the dual-ported RAM usually was connected to a CPU,

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec. Tony's username on Delphi is DISTO.

*An 80-column
adapter project for the
CoCo 1 and 2*

Increasing Character Display

By **Tony DiStefano**
Rainbow Contributing Editor

like the MC6809 CPU that is in all CoCos. The data lines of the RAM fed the rest of the address lines of the character ROM, and the ROM's data lines fed into a parallel-to-serial shift register. This shift register is the dot pattern that flows out of the display adapter and onto your screen. This dot pattern is mixed with the vertical and horizontal frequencies, called sync signals, into one signal that is called composite video.

Sound a little complicated? It might be at first, but read it again a couple of times and you'll understand it. Think of it like this: The CPU writes data characters into RAM, one character per byte of memory. The display chip, along with its support circuitry, reads this data and converts it into a stream of dots. With these dots come the signals necessary to control your monitor's circuitry to keep these dots in sync so that, to you, they look like characters such as letters and numbers.

What I have described requires a display chip (such as the MC6845), a character ROM, a RAM chip, about 20 other TTL support chips like the shifter and dual port circuitry. That makes quite a big job to design, let alone to do the board space and the wiring. But in the past, that's the way that technology was used.

Today things are different. Super LSI (Large Scale Integration) chips are here:

The CoCo 3 is proof of that. The GIME chip is a video display adapter, a memory map decoder and a memory management adapter all in one. Chips like that contain thousands of TTL equivalents.

To make this 80-column display, I will use an LSI chip made by Standard Microsystems Corporation (SMC), the CRT 9128. This chip by itself does most of the work I described above. It has built-in character ROM, all porting circuitry, and all decoding and shifting chips. The only support chip it requires is RAM. Add that and a couple of decoding chips, and you are done.

As you can see from the diagram in Figure 1, there are not many components in this project. U1 is the SMC chip, U2 is the RAM chip. A 6116 is a 2K-by-8-bit RAM chip. U3 is used for decoding, and U4 is used for the video output and mixing to form composite video. In the diagram all the pins of U1 are numbered and have abbreviated names. Most of the names are self-explanatory; the ones that are not are listed below:

Pin No.	Name	Function
7	VID	serial video data
18	HS	horizontal sync
19	VS	vertical sync
20	CS	composite sync, (HS and VS combined)
8	INT	intensity pin

Before I get into the project's construction, a little information on the SMC chip is needed. The SMC chip has internal registers that control the many aspects required to display characters on the screen. In order to talk to this chip, we must know where it is in the memory map below:

Location	Direction	Function
\$FF54	Write	Writes to data register
\$F554	Read	Reads from data register
\$FF55	Write	Writes to address register
\$FF55	Read	Reads status register

The CPU communicates with the video chip via seven registers. Access to these registers is made by first storing the register number in the address

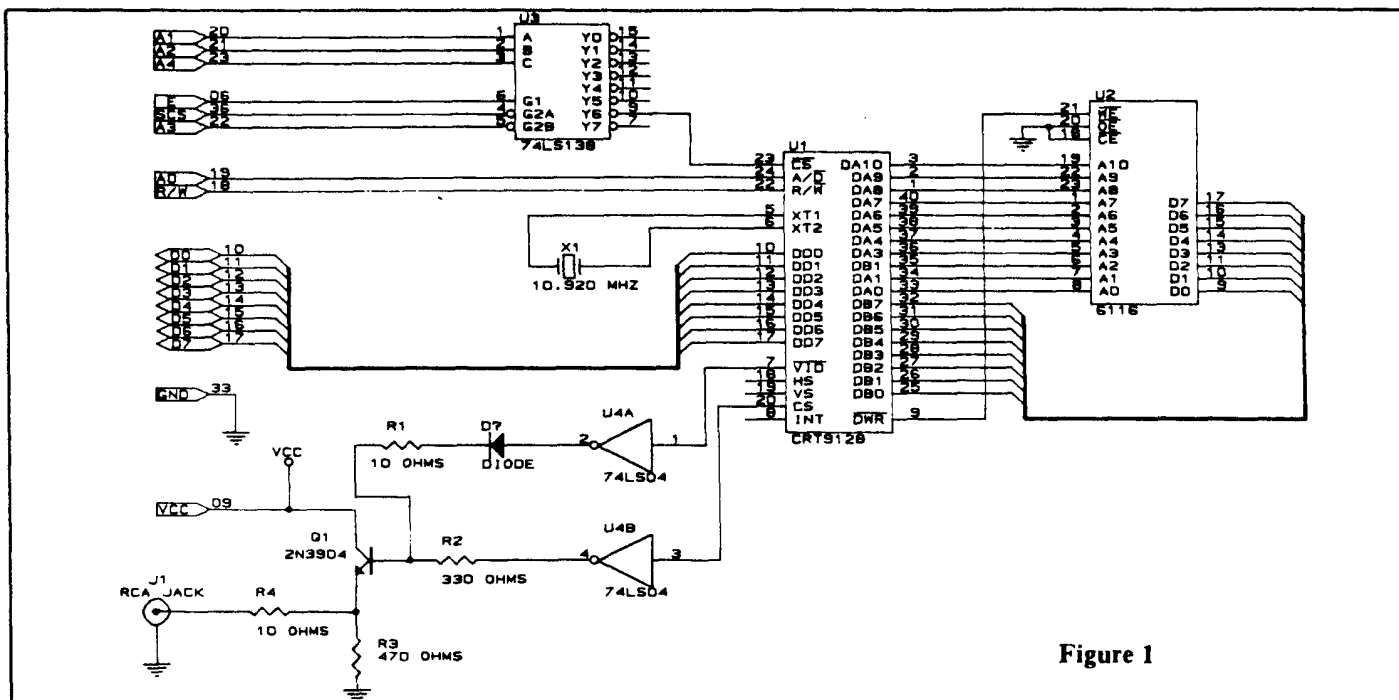


Figure 1

register and then accessing the register's data through the data register. The following is a list of the addresses and functions of available registers.

Address	Register Function
\$6	Chip Reset
\$8	TOS Add
\$9	CUR Lo
\$A	CUR Hi
\$B	Fil Add
\$C	ATT Dat
\$D	Character
\$E	Mode Register

For example, if you want to address the CUR Lo register, store the value \$9 at \$FF55 then store the CUR Lo byte at \$FF54. Each of the seven registers has a specific function:

Chip Reset — The first thing done after powering up the chip. Stores \$6 in \$FF55 then stores a 0 value in the data register.

TOS Add — TOS stands for Top Of Screen. Top of screen address bits are DA10 to DA4, for D6 to D0, respectively. DA3 to DA0 are internally set to 0, forcing the first address at the beginning of each row to be 00, 16, 32 and so forth.

CUR L0 — Cursor low address position of flashing cursor. This is the first eight bits of the cursor address.

CUR Hi — Cursor high address position of flashing cursor. Bits D2 to D0 are DA10 to DA8, respectively. Other bits set to 0.

Fil Add — Fills address locations starting from cursor position to the fill address. Bits D6 to D0 are addresses from DA10 to DA4, respectively. As

with TOS, the least three bits are always 0.

ATT Dat — Attribute Data, a register that changes the way things appear on the screen. The attribute byte:

- D7 = 1 Enables block graphics
= 0 Enables Alpha Mode
- D6 = 1 Disables cursor (Invisible)
= 0 Enables cursor (Visible)
- D5 = 1 Underlines cursor
= 0 Blocks cursor
- D4 = 1 White screen and black characters
= 0 Black screen and white characters
- D3 = 1 Enables video suppress
= 0 Allows character blinking
- D2 = 1 Hi intensity
= 0 Lo intensity
- D1 = 1 Character underlined
= 0 Character not underlined
- D0 = 1 Character in inverse video
= 0 Character in normal video

Character — Register where the ASCII character is placed to appear on the screen. If Bit D7 is set, the attributes described in the above byte (bits D3 to D0) will take effect on that character.

Mode — Auto increment mode. If Bit D7 is set to 1, the cursor address will automatically increment by one every time a byte is written to the character byte. If D7 is set to 0, the auto increment is disabled.

The basics for this display chip ought to be enough to get you started. If you want more detail, contact SMC at 35 Marcus Blvd., Hauppauge, NY 11788.

For this project you will need all the parts shown in Figure 1 and sockets for all the chips. The following is a list of socket sizes and the pin numbers for +5V and ground:

Chip No.	Socket Size	+5V	GND
U1	40	21	4
U2	24	24	12
U3	16	16	8
U4	14	14	7

You can get the SMC chip, project board, OS-9 software driver and RAM chip from CRC. Call (514) 383-5293 for prices.

There is one more interesting thing about the project. If you happen to have a Disto Super Controller or Disto Super Controller II, you can wire this project to the MEB connector. Two changes to the diagram in Figure 1 are necessary: Instead of A4, connect Pin 3 of U3 to VCC. Then, instead of A3, connect Pin 5 of U3 to GND. The rest of the connections appear on the bus. Instead of a project board, you can use just about any double-sided PC board. You will need, however, a 17-pin single inline female header. This way, you will not need a Multi-Pak.

Regarding the Multi-Pak, remember that when using the addresses from \$FF40 to \$FF5F, you must do a slot swap to whatever slot your hardware is, and swap back after you are finished. If you are in a multitasking environment remember to turn off the interrupts before swapping slots, and turn them back on again afterward.

Five years ago I introduced to the CoCo Community a piece of hardware called the Disto controller. It is compatible with Radio Shack's controller, as well as others. One of its interesting features is an internal mini expansion bus (MEB). This bus allows internal expansion of a peripheral card. Two of the adapters available for this controller are more popular than ever these days. The first is the clock/parallel adapter. This allows the user under OS-9 to have the real time at hand without having to type it in every time and to be able to connect a parallel printer to the CoCo without having to use an adapter. The second is a hard disk/serial adapter, which allows the user to connect a hard disk to the CoCo. It also has an RS-232 interface that is somewhat compatible with the Radio Shack Deluxe RS-232 Pak.

Until now, only one of these adapters would fit into the controller at one time. If you wanted a second, you needed an MEB carrier or a RAM disk along with a Multi-Pak Interface. Very expensive! If you had a CoCo 3, you also had to have the Multi-Pak modified. More bucks. As for myself, I have two systems, a CoCo 1 with an unmodified Multi-Pak and monochrome monitor, and a CoCo 3 with no Multi-Pak and a Sony RGB monitor. I don't intend to buy another Multi-Pak for my CoCo 3 system, so where does that leave me?

There were a couple of reasons for writing this article. The first is that if I do something for myself and find that it helps me do something else better, faster or more easily, I think that other people must have the same needs; most of the time I am right. This is why I began writing articles in the first place. The second reason is that Radio Shack has discontinued the RS-232 Pak and may discontinue the Multi-Pak in the future. What will we do?

If you take a look at the two adapters described above, they represent a lot of I/O: serial, parallel, hard disk and clock. To be able to have all those things without the Multi-Pak would be great. Getting the Super Controller or the Super Controller II is a good start, but you can still only put one of the two adapters inside the controller. This is

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec. Tony's username on Delphi is DISTO.

*A project to fit two
adapters into your
controller —
at the same time*

Two for One

By Tony DiStefano
Rainbow Contributing Editor

where I come in. I decided that I wanted both of these adapters in my second system's controller. So I took out my soldering iron, and this is what I came up with.

Before you get started, let me give you the drawbacks to this project. First of all, when all is said and done, you can no longer close the cover of the controller. An even bigger problem is power: When both of these boards are plugged in, the current draw is a little over the recommended limit of 300mA. A separate regulated supply must be built to handle the extra demand on power. Apart from these hurdles, a little soldering experience is needed.

Let's review some theory before taking out the ol' soldering iron, however. The MEB is a 17-pin connector that has data, address and control lines. The following is a description of these pins:

Pin #	Description
1	Reset
2	E Clock
3	A0
4	A1
5	D0
6	D1
7	D2
8	D3
9	D4
10	D5
11	D6
12	D7
13	CE (Chip Enable)
14	GND
15	R/W
16	+5V
17	A2

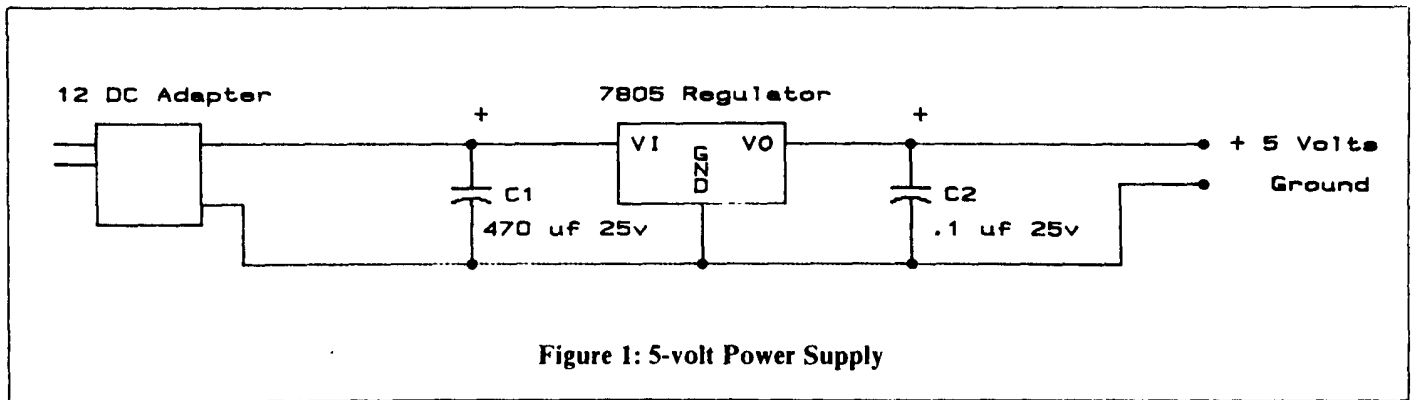
Study the pins carefully; it is a standard memory-mapped area. If we added another area to this, the only thing to change would be the CE. All other lines — data, address and control — would be the same. A piggyback technique here will do fine, except for the CE pin, which will go to another memory-mapped area. This is not too hard since the controller is already decoded; all you have to do is fish out the CE. Later, I'll tell you how to patch the OS-9 software, as well.

Not much to the theory, is there? In fact, this project is more mechanical than anything else. Now, it is time to get started. Please don't do any of these modifications with the power on. All the modifications are done on the hard disk/serial adapter. There are two cuts to do on this board, or only one if you have a modified power supply and it can stand the extra drain.

The first cut is to disable the CE from the board. Look at the component side of the board. Locate Pin 13 on the MEB connector. Follow the trace to the first hole and cut the trace just before that hole. For the +5V, locate Pin 16 on the same connector. Follow its wide trace to the first hole about one inch away, and cut the trace just before you reach that hole.

On the solder side, solder a set of 17 short male single inline header pins to the bottom of the MEB connector. The clock/parallel adapter board will sit on these pins. Now, solder one side of a 4-inch wire to the hole just after the first cut. For all versions of the Super Controller I, solder the other end of this wire to Pin 7 of the 74LS139 chip just below the 74LS04. For the Super Controller II, solder the wire to Pin 3 of J3 on the controller; you also have the choice of putting on a jumper instead of soldering it. One limitation is that you must use the alternate, eight-byte area for this modification; the other area is only four bytes long, so it cannot be used.

For the power, solder the plus side of a +5V regulated power supply to the hole above the second cut you made on the adapter. Locate Pin 14 on the MEB connector, follow it to the first hole, and connect the ground return of the power supply to it. Insert the clock/parallel board piggyback on Pin 17 that you just installed. Plug the hard disk/serial board into the MEB connector. Connect the controller into the computer. That is all there is to the hardware part of this project.



Now for the software patches for the OS-9 drivers. One of the great things about OS-9 is the ability to adapt software to hardware. In most cases, the way designers connect devices to a computer is very similar. *Where* these devices are connected, as far as the memory map goes, can be very different. The writers of OS-9 had this in mind when they wrote it. Along with the necessary software drivers, the fathers of OS-9 created small blocks of memory called descriptors. These descriptors have information on the physical aspects of the hardware they control — things like how many tracks on a disk or what baud rate the device works at.

One of the pieces of information included in these device descriptors is the memory location of the hardware. This tells the software driver exactly where in memory the hardware can be found. Now, what I did above is change the hardware location of the hard disk registers and the serial (RS-232) registers. The only way the software driver knows this information is through the device descriptor. All we have to do now is change the values in the proper device descriptors to the new memory locations, and we are home free.

Since the clock and parallel hardware is not changed, no changes to the de-

scriptors are needed. However, we do need to change the hard disk and serial descriptors. Let's start with the hard disk adapter. A little knowledge of OS-9 is needed to make these changes. On the disk that came with this adapter are drivers and descriptors. The $\angle h0$ descriptor used for the hard disk adapter needs to be changed. To change it, we will use the OS-9 command Debug. As part of the descriptor, there is a three-byte address that represents the area in the memory map where the hardware resides. This data is set for the hardware memory; but since we changed the hardware, we must now change the software. The third byte in this address is \$53. You now have to change this value to \$5A. To do this, execute Debug and link to the $\angle h0$ module. Press ENTER until you pass the series of two bytes, \$07 and \$FF; when you see the next value, \$53, type =5A to change it to the right value. Press Q to exit.

The other device descriptor to change is the serial one. Follow the same procedure as above, except use the $\angle T2$ descriptor. The byte to change may be one of two values. If it is the original, unmodified Tandy descriptor, the value to look for is \$68. If you have already changed this value, you will know that it is \$54. In either case, change it to \$5C.

If you want to make this change permanent, the OS-9 manual will describe just how to do this.

There will be a lot of cables protruding from this contraption: the disk drive cable, the hard disk cable, the printer cable, the RS-232 cable and the power cable. I bent and shaped all the cables so that they were parallel to the drive cable, and then I bundled them together with a tie-wrap. As I mentioned before, the cover will no longer fit; so I made another cover from a small piece of tin, bending, cutting and shaping it to fit. I did not bother to paint it, but you might.

The only thing left is the power supply. Radio Shack has all the parts necessary to build a regulated power supply. You will need all the parts listed in Figure 1. Most of the parts are not too critical and can be substituted for the nearest part. The transformer you must use is a DC adapter. A 12-volt adapter at about 150mA will do just fine.

I have recently joined Delphi. You can find me there as DISTO. Drop me a line if you have any problems or if you just want to say "hi." I'm not on at any regular time, but look for me in the OS-9 and CoCo SIGs. ☺

Communication is important in today's world. We understand what other people are saying because we all know the rules of communication. This set of rules is a sort of English protocol. When we hear the word "apple" (perhaps a bad example!) we immediately think of a red, ball-like object that can be eaten. If you say the word to anyone who knows the English protocol, he or she too will think of a red, ball-like object that can be eaten. This is a form of communication.

A set of rules has to be followed in communicating with a computer, too. This time you cannot use the English protocol, because the computer does not understand that — yet! To communicate with most computers, you have to press a number of switches arranged in a way that is familiar in human communications: the keyboard. We press these switches in an order that makes sense to us, but to the computer this is just a sequence of pressed switches. It compares this sequence to a known sequence in its memory banks. If a match is found, the computer then proceeds according to its programming.

The keyboard is an interface between a person and a computer, but there are times when we want one computer to communicate with another computer in order to transfer some kind of information the user needs or is sending. This computer-to-computer communication also has to follow a certain protocol.

There are many of these, ranging from simple serial communications to high-speed networks to parallel mainframe workstations. The protocol most used in the CoCo is serial. In this case, serial means to transfer data one bit at a time. The CoCo's internal memory is organized in eight-bit chunks called bytes. To transfer one byte of data from one computer to the other serially requires eight bit transfers. But that is just the data. In order to keep errors at a minimum, a start bit and a parity bit must also be included.

The CoCo has no special hardware to communicate in a serial fashion. Instead, it has a few bits on a PIA that is used by the CPU to simulate a real serial

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec. Tony's username on Delphi is DISTO.

Communicating computer-to-computer

All About Serial Packs

By Tony DiStefano
Rainbow Contributing Editor

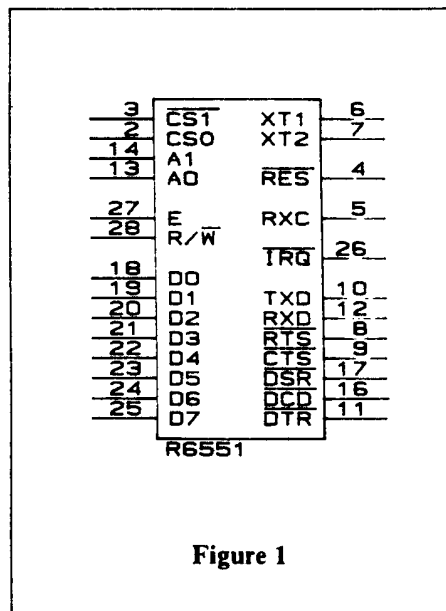


Figure 1

port. This makeshift port is limited in speed and performance. Also, with the exception of the CoCo 3, there doesn't seem to be any good software that supports this "bit banger," especially if you want to communicate at 1200 baud. The CPU simply does not have enough time to take care of the serial I/O and still do the rest of its chores. This led Tandy to introduce the Deluxe RS-232 Pak.

Inside it lies the hardware for a real serial port and true RS-232 protocol. At its heart is the Rockwell R6551 ACIA

(Asynchronous Communication Interface Adapter) chip. This chip has all the necessary circuitry to interface the parallel data of the CoCo's CPU to the standard RS-232 serial protocol and is capable of baud rates of 50 to 19,200. (Baud rate is the speed at which the bits are transferred.) It is also capable of word lengths from five to nine and has a programmable number of stop bits and parity detection. In fact, it is a great chip for our use. Figure 1 shows the pin-out of the R6551; a pin-by-pin description of this 28-pin chip appears in Figure 3 on the next page.

RS1	RS0	WRITE	READ
0	0	Xmit Data Register	Rmit Data Register
0	1	Reset	Status Register
1	0	Command Register	
1	1	Control Register	

Figure 2

From Figure 2, we see that the R6551 has four registers. The first is the data register. This is data going to and from the different computers. The next register is the Control Register. Bits 0 through 3 control the baud rate of the ACIA. Here is a list of the baud rates:

Bits	Baud Rate
3 2 1 0	Generated
0 0 0 0	EXTERNAL
0 0 0 1	50
0 0 1 0	75
0 0 1 1	109.92
0 1 0 0	134.58
0 1 0 1	150
0 1 1 0	300
0 1 1 1	600
1 0 0 0	1200
1 0 0 1	1800
1 0 1 0	2400
1 1 0 0	3600
1 1 0 1	4800
1 1 1 0	9600
1 1 1 1	19200

Bit 4 controls the external clock, with 1 being baud rate and 0 being external. Bits 5 and 6 are word length. 00 is 8, 01

is 7, 10 is 6 and 11 is 5. Bit 7 high is two stop bits, and Bit 7 low is one stop bit. The next register, the command reg-

ister, is used to control the specific transmit and receive functions shown in Figure 4.

Pin No.	Name	Description	Pin No.	Name	Description
1	GND	Signal and power ground. All signals are referenced to this pin.	12	RXD	Receive data input pin used to transfer data from the external device.
2	CS0	Active low-input chip selects the device. When this pin is low and CS1 is high, the chip is selected.	13	RS0	First of two register select lines connected to CPU address lines. Used to select various internal registers. See Figure 2.
3	CS1	Active high-input chip selects the device.	14	RS1	Second of two register select lines. See Figure 2.
4	RES	Active low input resets and initializes internal registers to zero.	15	Vcc	Input is connected to +5 volts. It powers the chip's internal circuits.
5	RSC	Receive clock pin is bi-directional; serves as the receiver of 16X clock input or output.	16	DCD	Data carrier detect input pin used to indicate to the chip the status of carrier detect output of the external device.
6	Xtal1	This pin and Xtal2 are normally directly connected to an external crystal to derive various baud rates. Crystal frequency for these baud rates must be 1.8432 MHz.	17	DSR	Data set ready input pin used to indicate readiness state of the external device. A low indicates a "ready."
7	Xtal2	Connected to other side of the crystal.	18-25		Data bits D0 through D7, respectively; bi-directional lines used to transfer data to and from the CPU to the chip.
8	RTS	Request to send output used to control the modem from the processor. Output of this pin is determined by contents of the command register.	26	IRQ	Interrupt request pin is an open collector (drain) output used to flag the CPU when the chip has finished using data. IRQ status bit allows many pins to be connected to the same IRQ line to the CPU.
9	CTS	Clear to send input pin used to control transmitter operation. Transmitter section of the chip is automatically disabled if CTS is high.	27	E	E clock input to this pin used to gate all data transfers to and from the CPU.
10	TXD	Transmit data output pin used to transfer serial data to the external device. The least significant bit is transmitted first, with rate determined by baud rate selected.	28	R/W	Read/write input pin used to control direction of data transfers between the CPU and the chip. A low on the R/W pin allows a write to the chip.
11	DTR	Data terminal ready output pin used to indicate status of the chip. A low on DTR indicates the chip is enabled. This bit is controlled via Bit 0 in the command register.			

Figure 3

Bits	Description
0	Hi= Enabled DTR
-	Lo= Disabled DTR
1	Hi= IRQ Disabled
-	Lo= IRQ Enabled
3 2	Xmit IRQ RTS Other
--	
0 0	Disabled Hi -
0 1	Enabled Lo -
1 0	Disabled Lo -
1 1	Disabled Lo Xmit BRK
4	Hi= Echo
-	Lo= Normal
7 6 5	Operation

X X 0	Parity Disabled
0 0 1	Odd Parity
0 1 1	Even Parity
1 0 1	Mark Parity Xmit Check Disabled.
1 1 1	Space Parity Xmit Check Disabled.

Figure 4

The final register is the status register. These bits in the status register indicate to the processor the status of the various

Bit	Low	Hi
0	No parity error	Parity error detected
1	No framing error	Framing error detected
2	No Overrun error	Overrun error detected
3	Receive buffer -Not full	Receive buffer -full
4	Transmit buffer -Not empty	Transmit buffer -empty
5	DCD detect	DCD not detected
6	DSR ready	DSR not ready
7	No IRQ	IRQ has occurred

Figure 5

R6551 functions as outlined in Figure 5.

The R6551 is the heart of the pack, but not the only part. Its job is to take the eight-bit data to and from the CPU and transmit it at the right baud rate and parity, but that is not all. This chip has a high level of 5 volts and a low level of ground, or 0, volts. RS-232 standards require that the voltage for serial communications be a high of +12 volts and a low of -12 volts. This is done through two chips known as level shifters. The first, the MC1488, is a shifter that changes 5/0 volt levels to 12/-12 volt levels. The other, the MC1489, does the opposite: It shifts the 12/-12 volt inputs to 5/0 volt.

Other parts include decoders and buffers, resistors and capacitors. Software in a ROM is also included. This software gives the CoCo the ability to communicate with other computers. It is OK as far as "dumb terminals" go, but it lacks the power for good data transfers. Most people use other third-party software to drive this pack.

I have designed an equivalent to the above-described RS-232. It functions the same except that it has no built-in software — no great loss, since most people do not use it. If you are using OS-9, the software driver is already included and is compatible with my pack. For prices and delivery, call CRC at (514) 383-5293. ☺

Summer Cleanup

An update on the parallel interface and a hardware patch for the Multi-Pak

This month we will look at two short items. The first is an update on an earlier project, and the other is a hardware patch for the Multi-Pak's IRQ problem.

In a two-part project in November and December of 1987, I described making a parallel interface and building it right into the CoCo. It turns out that some people are having problems. Paul Anderson of SD Enterprises said that the DWP 430 printer from Radio Shack requires a longer strobe pulse width than my circuit delivered. At 2 MHz (the double speed for the CoCo 3), the problem was even greater.

I have an Epson FX-80 printer. When I tested my circuit on it, it worked fine. The pulse width for the strobe signal

(even at 2 MHz) is wide enough to make it work. If you look back to the circuit in the November '87 RAINBOW, you see that the signal connecting the data into the latch is also the signal that drives the strobe signal of the printer. That signal is derived from the memory mapping of data from the CPU. This makes the width of the signal directly proportional to the clock speed of the CPU. The faster the CPU is clocked, the shorter the strobe pulse is. For my printer this is no problem, but for slower strobe printers like the DMP 430, it is a problem that must be addressed.

To solve this problem, I looked at the spec sheets of several popular printers. Much to my surprise, I found out that printers have a wide range of strobe

pulse widths, from .5 microseconds to a full 2 microseconds. Not only that, I also found out some printers require that data be valid up to 1 microsecond before the strobe line goes active. If you look again at my circuit, the strobe line is active at the same time as the data. *Oops!* I guess I should have done my homework before putting out that article. Well, fortunately, I have a good fix. After looking through my TTL data books, I came up with a circuit that will give the strobe signal both a 1-microsecond delay and a pulse width of 2 microseconds. That should be enough to satisfy any printer's needs.

The chip I decided to use is a 74HC123, which is a *dual retriggerable monostable multivibrator* — a mouth-

ful, but not that complicated. Basically, there is an input signal and an output signal. An R/C (resistor/capacitor) constant determines how long the pulse is. Every time the input is strobed, the output becomes active for the duration of the pulse width, which is controlled by the R/C constant. I chained the output of the first multivibrator to the input of the second, which then goes to the strobe of the printer. The first gives me the delay to set up the data; the second gives me a pulse width that is controllable by the R/C constant and not the clock speed of the CPU.

Construction of this, I hope, won't be too hard. If you have already built the parallel printer adapter and have enough room to fit one more chip and four components, you're home free. If you have not built it yet but want to, just make sure that you have enough room to place one more chip. The rest of it is the same as in the November '87 issue.

If you don't have enough room, you have two choices: Start over again, or make a small piggypack board. I suggest that you start over, since it makes for a cleaner job and is easier to trace if you have a problem.

The circuit in Figure 1 is the fix only and does not include the rest of the circuit needed to make the complete parallel adapter. The 74HC123 chip requires +5 volts on Pin 16 and ground on Pin 8. To interface it into the rest of the original circuit, follow these instructions:

- 1) Remove the wire that goes to Pin 1 of the printer connector.
- 2) Connect that wire to the point marked "input" in Figure 1.
- 3) Connect the wire marked "output" in Figure 1 to Pin 1 of the printer connector left vacant by Step 1.
- 4) Connect +5V and ground to the chip.

With this modification, no other changes are required; the software remains the same, and all printers should work at either slow or fast speed.

The second part of this article deals with interrupts and the Multi-Pak. Many people may never come across this problem, which will show up only in certain cases. First, I think that explaining what the Multi-Pak does will help you understand the problem.

The Radio Shack Multi-Pak has four slots and was Radio Shack's original idea to expand the CoCo; the idea was that people who bought the expander would plug four game packs into it. As we all know, Radio Shack game packs auto start. That means when you plug in a pack (without a Multi-Pak) and turn the computer on, the game (or whatever) starts to play all by itself. To do that, the computer must be able to sense the presence of the pack. One pin on the connector connects to the CPU's interrupt pin via a PIA. On an auto-starting game pack, this pin is connected to the Q clock. The Q clock is a signal coming from the internal cir-

cuits that runs at 1 or 2 MHz, depending on the mode of the computer. This signal is fed into the interrupt pin of the CPU. The CPU responds to this interrupt by a small routine in ROM that jumps to the software inside the game pack.

In making the Multi-Pak, Radio Shack wanted to be able to handle four packs instead of one. To choose which one of the packs works requires a switch, so a four-position switch was added. The first part of the switch is a block of memory known as CTS. This block, as long as 16K, is found from \$C000 to \$FEFF for a CoCo 1 and 2 and from \$C000 to \$FDFF for the CoCo 3. The second is another block of memory known as the SCS area and is mapped from \$FF40 to \$FF5F on all CoCos. The third part of the switch reroutes the interrupt signal from the selected slot or game pack to the CPU.

To make the Multi-Pak software selectable as well, Radio Shack made one memory location, \$FF7F, into a software switch. The 8-bit location was divided into four 2-bit decoders, two of which control which of the four slots are active. Since the two memory blocks are controlled separately, the CTS block can be selected to one slot and the SCS block to another. This was a good idea, since the CTS block usually contained software and the SCS block usually contained hardware I/O.

At this point Radio Shack decided to tie the interrupt router to the same circuitry that controls the CTS, so that

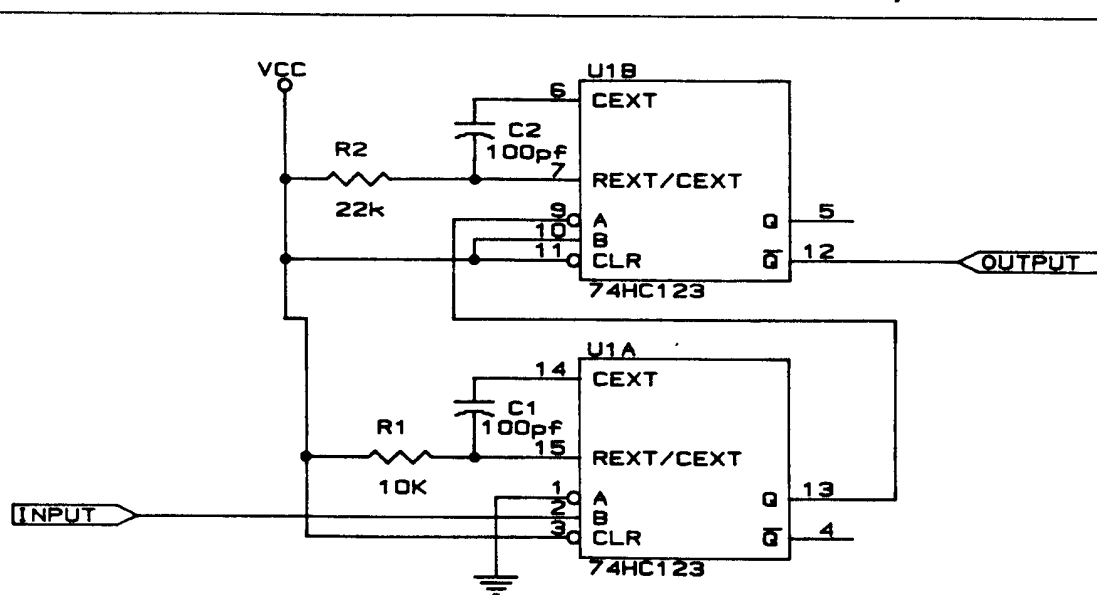


Figure 1

whatever CTS slot is active originates the interrupts. This arrangement is OK for game packs, since changing the switch to another slot means that whichever slot has the interrupts also has the right software. Good for game packs, but not so good for OS-9 users — OS-9 relies heavily on interrupts. Most hardware handshaking is done with interrupts. OS-9 uses the all-RAM mode, so the CTS signal is not used. But with the Multi-Pak, the software switch still switches the interrupt signal. The problem is mostly seen when someone uses the Deluxe RS-232 Pak.

Under OS-9, drivers and hardware devices can be added and left out to suit the owner's particular needs, but no one driver knows what else is using the hardware. When one device driver needs the interrupt line, it changes the software switch to the slot the hardware is in. If another driver needs the inter-

rupts, it switches it back; this is where the problem starts. When you change the software switch away from one slot, the interrupt has a chance of getting lost. The problem gets worse when a device like the RS-232 Pak is online. The registers for this pack are memory-mapped in an area not covered by the software switch, while the interrupts are covered by the software switch. So if one driver switches the software switch away from the slot the RS-232 pack is in, it can no longer produce an interrupt. Even though the registers are still in the memory map, data is lost and things start to get confused.

One solution for this is a small modification in the Multi-Pak. Interrupt signals in non-game packs are usually "open collector," meaning that more than one signal can be connected together to form an "OR" type of configuration. A simple way to avoid the

problems is to connect all the interrupts together, so that no matter which slot the interrupt comes from, the signal comes through. This mod is simple and quick. A little soldering experience, a few tools and a short piece of wire are all you need. Unplug the Multi-Pak and remove the bottom screws. Remove the top and disconnect the power to the board. Now, remove the screws that hold down the PC board. Carefully remove all the pins that hold the bottom shield to the board. Locate Pin 8 on each of the four slot connectors. Solder a piece of wire from one to the other until all slots are done. Reassemble the Multi-Pak in reverse order, and that's all there is to it.

With this modification you should be able to use the RS-232 Pak under OS-9 with any one device that changes the software slot switch and without losing characters on the Pak. ☺

Ever notice that my articles run in patterns? Usually, I start with a simple project for the beginner, move on to a harder, longer project and then finish with an electronic lesson. Well, it's time, once again, for a beginner's project. It is always hard to design a simple project that actually does something. As an electronics student in college, I did a lot of labs. They were simple, but they were *boring*. (Set the power supply to 10 volts. Put two resistors in series. Measure the voltage across the two resistors. Compare the values to that of the calculated voltage values.) Those labs were enough to put you to sleep in the middle of a lab.

For this column, I had to design a project that is simple but not boring. I checked to see what beginners wanted as a starter project. Most said they wanted something that worked in front of them — something that buzzed, beeped, moved or lit up. In the past, I have had projects using an LED to indicate that power is on, the disk drive is on, etc. LEDs are always a good project, and this beginner's project makes the computer control up to eight

*Finally, a beginner's
project that does
something*

A Simple, Expandable LED Project

By Tony DiStefano
Rainbow Contributing Editor

LEDs. (Note: Even though this project is for beginners, some electronics knowledge is required. Read the article and judge for yourself if you understand enough of it to try it.)

I will continue this project for a few months and make it grow into a miniature control center. This project will show the beginner how to turn on LEDs, small motors, relays, sensor devices, etc. If you come up with a few ideas, let me know. You can write to me

in care of THE RAINBOW or reach me on Delphi.

As with any project, you need tools. How far you want to go with this project will determine how many tools and parts you will need. To begin the project, you will need the following parts:

Part #	Description
U1	74LS273
C1	.1uf 10 volts
R1 to R8	470 ohm ¼ watt
D1 to D8	LED (just about any kind)
Misc.:	20-pin socket and wire.

You may already have some of these materials, and most are available at your local Radio Shack. You may need to get some parts through a mail order service. Many companies that have the parts advertise in RAINBOW.

The first thing you need is a project board. Radio Shack has dropped this item. I suggest you check RAINBOW's advertisements to find a board. I get my boards through CRC, but the board is available through other companies. At this time, the only tools you will need are a soldering iron and some solder.

It should take less than two hours to assemble this project. We will do it together, step by step. Don't start until you have all the parts. It's no fun to let a project sit, incomplete, because some of the parts are missing.

Before we begin, it is important to

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec. Tony's username on Delphi is DISTO.

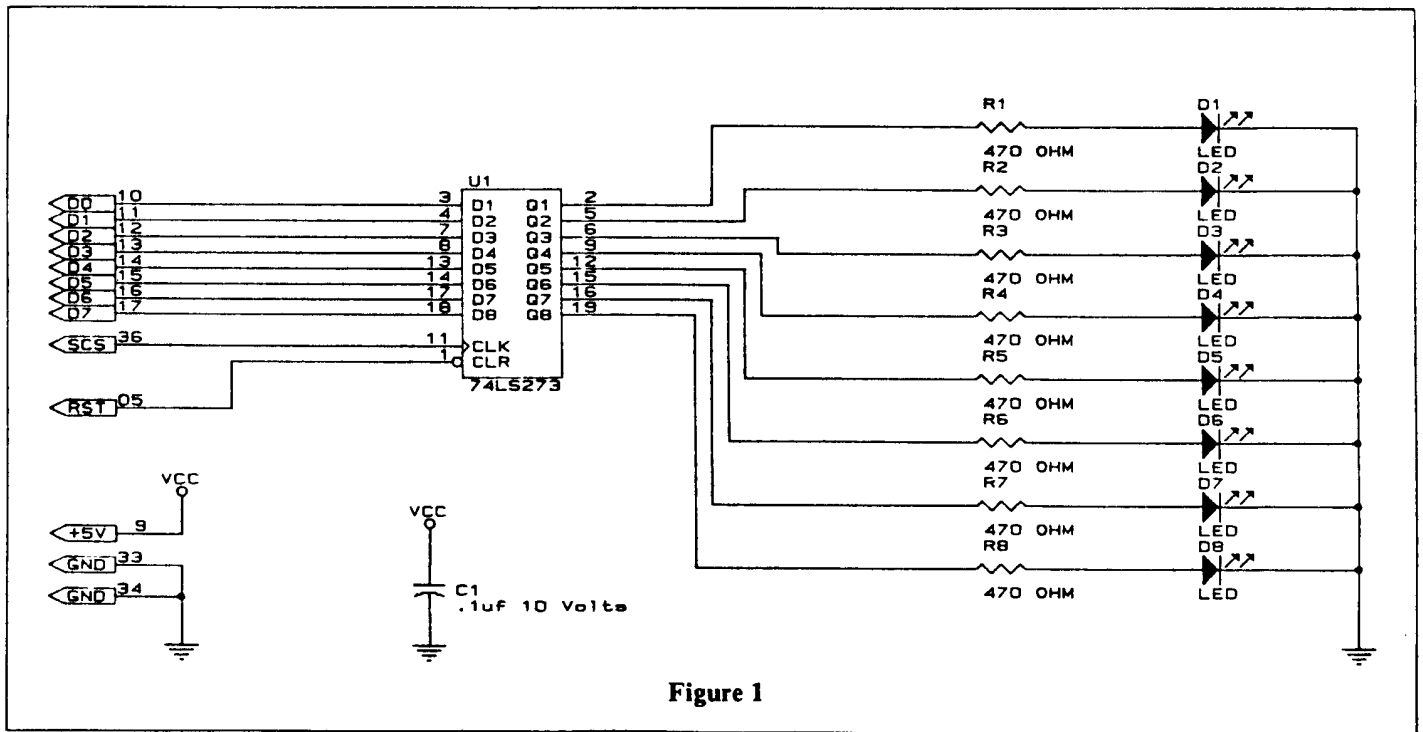


Figure 1

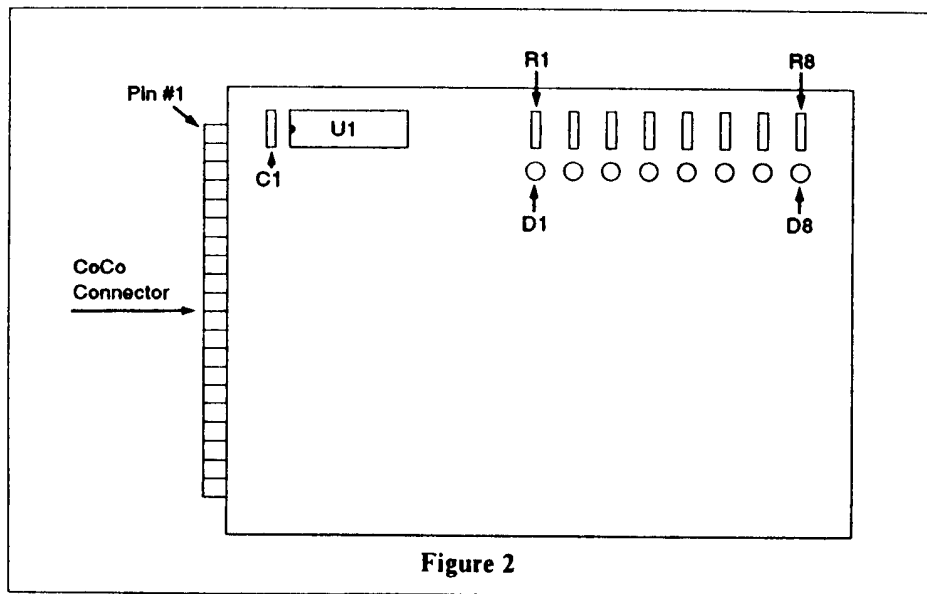


Figure 2

understand how a schematic diagram works. Look at Figure 1, and examine U1. The pin numbers are not drawn in any order. They are arranged so the diagram is easy to understand. All the inputs are on one side, and all the outputs are on the other.

On the actual board, the pins are arranged in order. Begin with Pin 1, which is identified by a notch or dimple.

The next pin in a counter-clockwise direction is Pin 2. The other pins are in the same counter-clockwise order. The boxes on the left of Figure 1 are the pin descriptions for the CoCo's pin connector. The numbers above the wires are the pin numbers. Pin +5V leads to a box labeled VCC. That means every point in the diagram hooked up to VCC is really hooked up to that pin. This also applies

to Box GND. All points marked GND are connected.

While it is not obvious on this small diagram, the way the diagram is presented makes the schematic easier to read. Instead of wires everywhere, labels are used. (Please note: Though not marked on the diagram, U1 has a VCC at Pin 20 and a GND at Pin 10.

Now, let us begin the project.

First, put all the parts on a clean table. If you are using a CRC project board, make sure you have the right side up. A small #1 is printed next to Pin 1. This is the top. Pin 2 is directly below Pin 1. Pin 3 is next to Pin 1, Pin 4 is below Pin 3 and next to Pin 2, etc. All parts will mount on the top.

Mount the 20-pin socket in the top of the protoboard. For proper placement, follow the plan in Figure 2. Make sure that Pin 1 is the pin closest to the edge connectors. Solder all the pins of the socket, and mount the resistors and LEDs. Make sure that the short lead of the LED is positioned away from the resistors. They are polarized, and the short lead is the negative side. Bend the leads so that no part falls out. Insert the capacitor next to the socket, and bend the leads of this part as well.

The rest is just wiring. You know the pin numbers and positions. One at a time, solder a wire between the points in the schematic. Every time you place a wire, mark it off on the diagram. This serves two purposes: that you don't miss any points and that you don't try to do any point twice.

Let's do the first few together. Following the schematic, solder one end of the wire to Pin 10 on the connector. Cut the wire so that it just reaches Pin 3 of U1, and solder that end of the wire to Pin 3 of U1. Mark off this wire on the schematic. Next, solder an end of the wire to Pin 11 of the connector. Cut the wire so that it just reaches Pin 4 of U1, and solder that end to Pin 4. Mark off that wire on the schematic. Now finish off the rest of the wires one at a time. When you are finished, recheck all your work. Remember to check the VCC and GND of U1. Insert the 74LS273 into the socket, and make sure that Pin #1 is in the right place.

That's all there is to the hardware part of it. Plug it in, turn on your computer and check for the normal power-up message. If you do not, turn off the computer and check your work again.

Now that you have built it, let's see

how it works. Look at Figure 1. The main part in this project is U1, an eight-bit D-type flip-flop. All the D's are inputs and all the Q's are outputs. When the CLK input is strobed, the binary level on D is transferred to Q. Thus, if all D's were at Level 1 when the CLK was strobed, all the Q's (outputs) are now at Level 1. The D's are now at Level 0. The CLK that I am using is the CoCo's SCS pin. It is mapped at \$FF40 to \$FF5F. Since I am not using any address lines, mirroring will occur throughout this area. Next month, when we expand, I'll use the address lines to add more to this project.

Since they are all connected to identical circuits when any Q has 0 volts, no current can flow because the other end of the circuit also has 0 volts (GND). The LED is off. When any Q is high, roughly three to five volts, current flows through the resistor and the LED.

Since each LED is represented by one bit on the CoCo's bus, D0 on the CoCo controls LED 1, D1 controls LED 2, etc. Since it is memory-mapped on the CoCo's bus, a simple BASIC poke command will turn on the LEDs. Thus, if you type POKE &HFF40, 255, all the LEDs should go on. (Wow! It works.)

If it doesn't work, check all your wiring. Did you put all the LEDs in the right direction? Try reversing one and see.

If it is working, continue by typing POKE &HFF40, 1. Only one LED should be on. Now try typing 2 instead of 1, then 4, 8, 16, 32, 64 and finally 128. Each LED should light up, one at a time. Now try 72 (8 + 64). Adding two LED values together will cause both LEDs to come on. Use a FOR/NEXT loop to write a program that makes a chaser.

Those of you with Multi-Pak Interfaces must remember that the SCS pin is switched. In order to poke the values at \$FF40 in the right slot, you must change the slot access. You can do this by going into the all-RAM mode and turning the switch in front of the Multi-Pak to the project's slot. You can also make sure that your disk controller is in Slot 4, then put your project in Slot 1 and type POKE &HFF7F, &H30. This will change the SCS access to Slot 1 and leave the CTS, or DOS, access in Slot 4. Remember to return to &H33 before trying to access the disk.

In my next column, I'll expand this project to include more goodies that beep, boop and buzz.

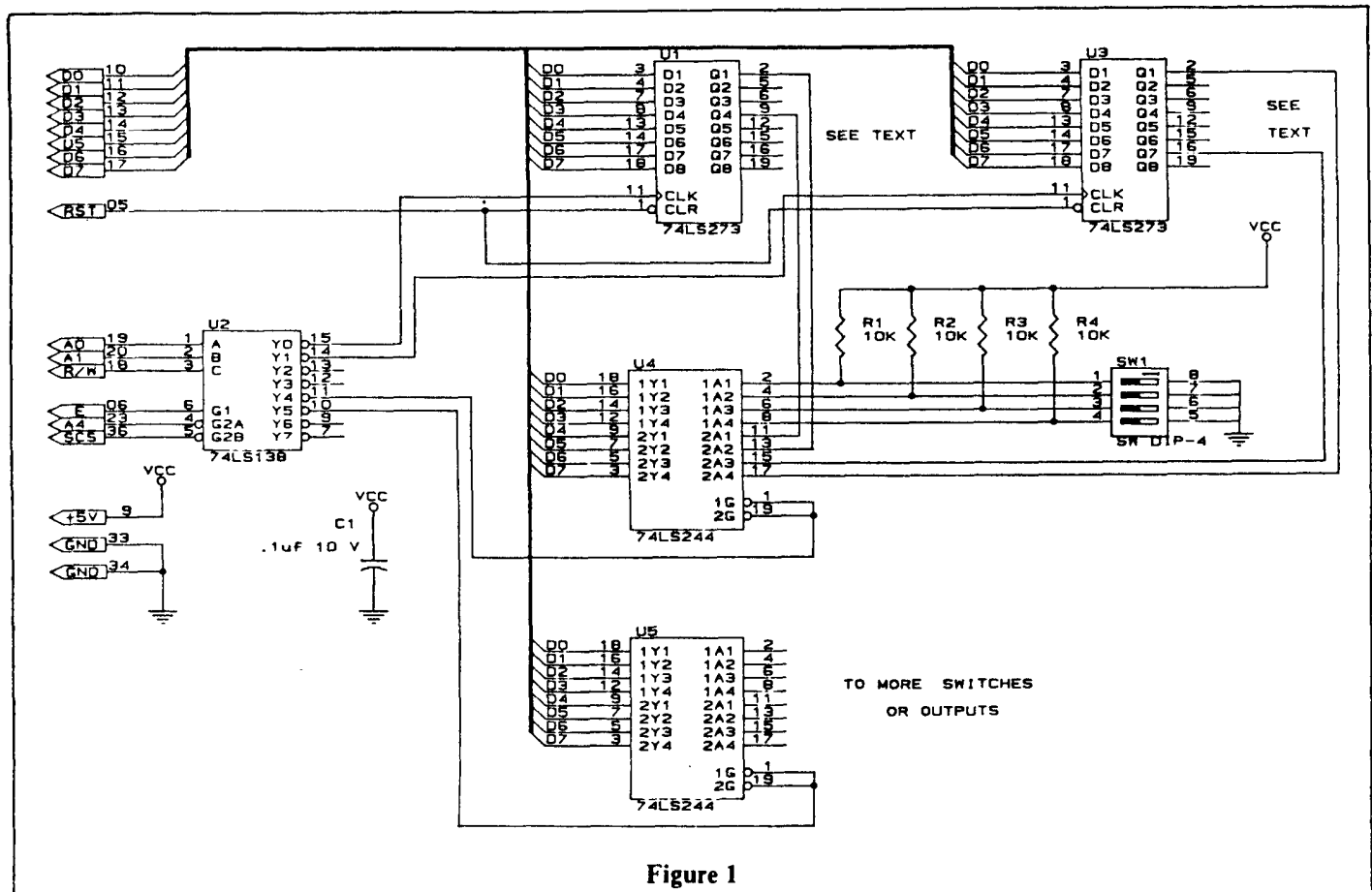


Figure 1

Last month, we started a one-chip beginner's project that turned on some LEDs. Let's expand that idea to a four-chip project that controls more than a few LEDs. We will begin with a short explanation of the electronic theories used in this project. Once you understand what we are doing, all you will need is a little patience and a few parts to complete this project.

If you look at the diagram we will use for this project (Figure 1), you will see that it differs in several ways from the one we used for the first part of the project. First, because there will be no changes in the circuit that involve the LEDs, I removed all the LEDs and their resistors from the diagram. This gives me more room to work and makes the schematic less cluttered. Leave the LEDs on your board, just expand it. Next, in the original diagram I used separate wires to connect the pins on the connector to the corresponding pins on the computer (i.e., D0 on the computer to D0 on the chip). During that phase of the project, each wire went to only one place.

When I expand, however, I must use a technique known as bussing to connect one pin to more than one other pin. To illustrate this change in the diagram, I used a thick line called a Bus line. This line indicates that several wires are grouped together. In such a grouping, the wires generally have something in common. In this case, all the wires are data lines. Bus lines may also carry address lines, control lines, etc. This technique saves space and makes things look neater. To identify these wires as they enter or exit the bus line, the wires must be labeled (see Figure 1).

In this project, we will use the same chip we used in the last phase, and we will change only one wire on this chip. If you begin with last month's project, the only wire you will need to change is the one connected to Pin 11.

Now look at U2 — a TTL chip 74LS138. It is a 3-to-8 decoder. In binary, one bit has two different conditions, two bits have four and three bits have eight. U2 takes a three-bit binary input and decodes it into eight different combinations. The three inputs are A,

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec. Tony's username on Delphi is DISTO.

*Use last month's project
to power your
imagination*

Project Expansion

By Tony DiStefano
Rainbow Contributing Editor

Y7. Normally, all but one of the outputs are high. The low output depends on the condition of the three inputs and the three control lines. The output is disabled (all high) unless G2A and G2B are low and G1 is high.

Look at the six inputs and their connections on the schematic. We can see from the three control lines that the outputs will work only when the following conditions are met:

1 — The SCS pin (G2B) is low. When this occurs, we can access the I/O area of the CoCo, located from \$FF40 to \$FF5F.

2 — The A4 pin (G1B) is low. This limits access. When A4 is low, we can access \$FF40 only to \$FF4F — half of the previous area. If we decode more address lines, we can limit it to a smaller area, but that is not required now.

3 — The E pin (G1) is high. This ensures that the data is valid when we use more than one chip. The CPU specifications manual states that data and address is valid during the high portion of the E clock.

Let's look at what we have so far. The chip select is properly active between \$FF40 and \$FF4F. Inputs A and B are connected to A0 and A1 respectively. This decodes to one of four memory locations (represented by Y0 to Y3 if our third input (R/W) is low, and Y4 to Y7 if R/W is high). If you look at the function of the R/W line, you will understand the final stage of this IC. In

the CoCo, when the R/W line is high, the CPU reads in data from whatever address area the address bus dictates (represented by the PEEK command in BASIC). When the R/W line is low, the CPU writes data to whatever address area the address bus points (represented by the POKE command in BASIC).

Instead of the one memory location to which you could write in last month's project, you now have four memory locations to which you can write (Y0 to Y3) and four from which you can read (Y4 to Y7). (More about the read locations next time.) Looking at Figure 1, you see that Y0 is connected to U1's CLK. Writing (or poking) data to \$FF40 will transfer that data to U1 and, in turn, light up the LEDs. That much of our project remains the same. Now, however, we have another data latch — U3. Because U3 is the same chip (74LS273) as U1, it presents the same output characteristics as U1. However, we want to control more *fun* things than LEDs with this chip.

Unfortunately, the 74LS273 chip cannot supply much current, so we will need another *buffer* chip that can. We will use the 7406 chip, which is a hex open-collector inverter/buffer chip. As an open collector, the chip can only act like a SPST (Single-Pole, Single-Throw) switch with one side connected to ground. It cannot supply voltage. As an inverter, the chip inverts the incoming signal, and as a buffer it can supply a larger sum of current. When the input to one of these inverters is high, the switch (output) is considered closed. When the input is low, the switch is opened. With this information, we can use our circuit to control small DC devices.

Look at Q1 of U3. It is connected to an input of one of the hex buffers (Pin 1 of U4). The output (Pin 2) goes to the negative side of a small DC motor. The other side of the motor connects to VEE. Connecting VEE to the CoCo's VCC puts 5 volts on the motor.

Before you connect *any* motor to our circuit, however, there are a few rules to follow. These important rules *must not be broken*. If they are, *permanent* damage may occur to your circuit and to your computer.

The chips on this board have a 5-volt control voltage. This voltage comes from the CoCo through Pin 5 on the connector. According to Tandy, the

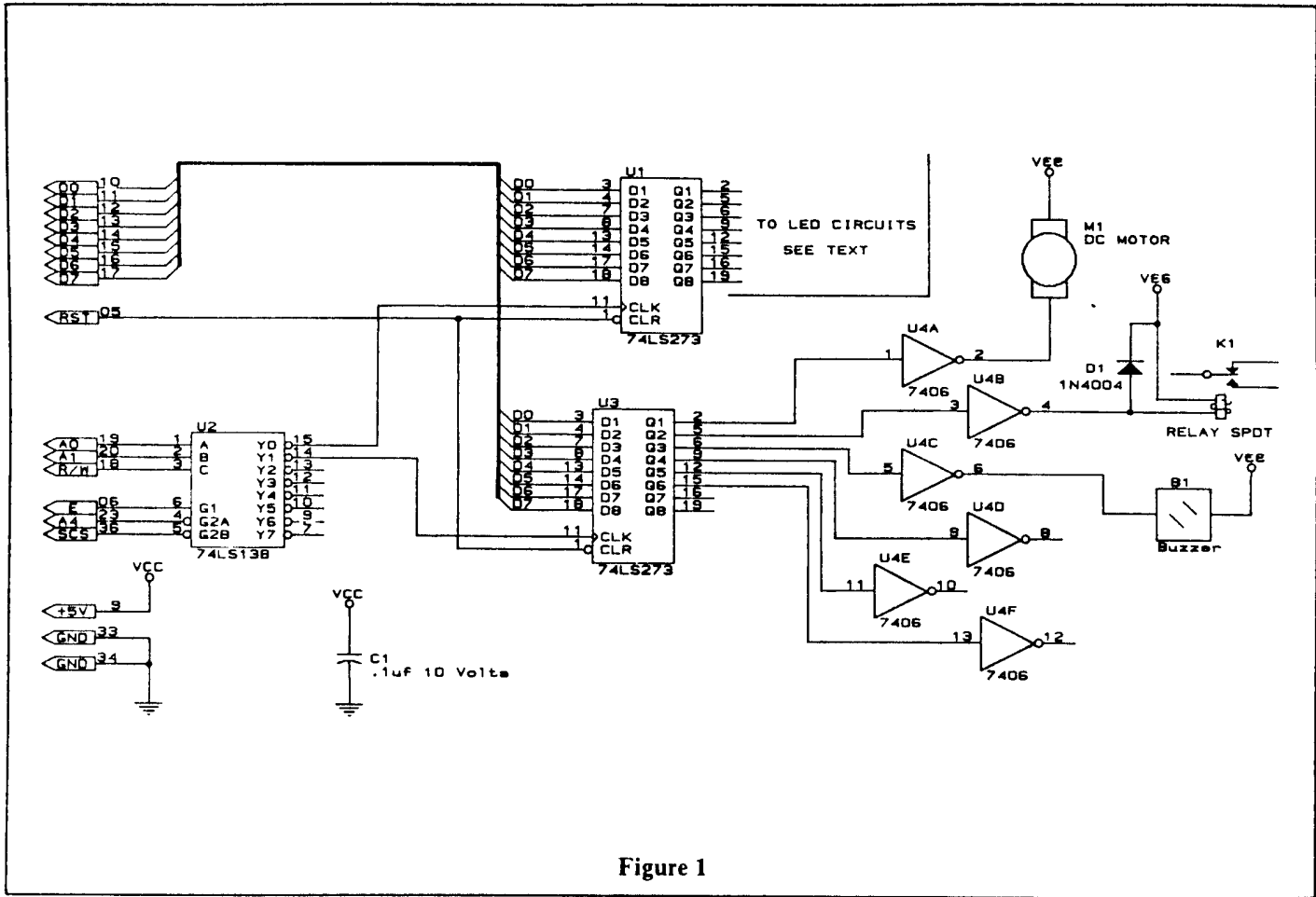


Figure 1

current limitation on this supply is 300 ma (milliamps). It takes 1000 milliamps to make 1 amp, so 300 ma is .3 amp. Drawing more than 300 ma from the computer may damage the power supply. So how does one know when the limit is reached? When the computer smokes — just a joke, but drawing more than 300 ma isn't. If you have a meter that can measure current, you're in luck; if you don't, you'll have to calculate how much current you are using. TTL chips generally draw about 10 ma each. Depending on what you have on at the same time, you are left with about 250 ma. When the LEDs are on, they draw approximately 50 ma more. That leaves you with about 200 ma for the rest of the circuit.

The amount of current drawn by small devices (e.g., motors, relays and buzzers) is usually marked on the device. To be completely safe, you should not go over 300 ma for all connected circuits. Unfortunately, that may not leave you with many connected circuits. In that case, make sure you turn on the circuits one at a time.

Another solution is to power the devices with an external power supply.

Radio Shack sells several DC adapters — some with multi-voltages. If you power your devices externally, make sure the device and adapter you use are the same voltage. Connect the negative side of the adapter (usually black wire) to the ground of the project circuit and connect the positive side (usually red wire) to the point marked VEE on the device — not to the VCC of the computer. The maximum voltage you can use externally is 15 volts. More than that risks damage to the buffers. In addition, each buffer can sink only about 50 ma.

I got the small devices that I used from the Radio Shack catalog. I used the relay (Cat. No. 275-243), but look through the catalog; there are many things you can hook up. Use your imagination to control a robot arm, electric race car, train set, etc. But remember, it's important to match the voltages and not exceed current limitations. Most Radio Shack items mention voltages and currents.

Anything you use will connect in the same way — the negative (black wire) connects to the outputs of the buffers, and the positive (red wire) connects to

the VEE source (either the VCC of the computer or the plus of an external DC adapter). The schematic shows only six buffers, because there are only six buffers in one chip. If you need the other two outputs of U3, you will need another 7406 chip.

To construct this project, continue as you were instructed in the last column. If you plan to use many small devices, leave room for other control circuits by using a multi-pin connector and mounting the devices on a separate board. When you build this, remember that Figure 1 does not show the +5 volt and ground connections shown for U1 to U4. Those connections are listed below:

IC	+5 volts	GND
U1	20	10
U2	16	8
U3	20	10
U4	14	7

Well, that's it for this time. Enjoy your new toys. Next time we'll look at some input devices the computer can read. ☺

Part 1 of this project (November '88, Page 157) explained the basics of start up. We started with a big project board and put two TTL circuits and a few LEDs on it. I used the first part of this project to show you how to output to the board and turn each LED on and off. In Part 2 (December '88, Page 146), I expanded the board to control things that required more current (like relays, buzzers and motors). This required another TTL chip like those used in Part 1 and an additional chip capable of carrying more current.

The first two parts of the project dealt only with outputs. You could turn devices on and off, but then you could not read the condition of the devices (like switches). In order to do that, you need a circuit able to read in data via the data lines D0 to D7. This, in turn, requires the proper decoding circuitry and a device that will buffer the switches. Study the circuit in Figure 1. It is a continuation of the circuit used in the last part of our project. In order to save space, I removed the details of the first and second parts. Any parts that will not be changed, I removed. The LEDs of Part 1 and the motors and buzzers of Part 2 have been removed. I left the buffer chips there, so you can see how the circuits work.

The first thing we need in order to be able to read in some data is a decoder able to decode the Read/Write (R/ \bar{W}) line. Chip U2 of Figure 1 is the decoder chip we have been using. It is a 74LS138, a three-to-eight decoder. By now, you should be familiar with this chip, but let's review what lines are connected to it. The most important line is the SCS from the computer. This is connected to one of the *select* lines of U2, the G2B. This line is used to select a block of memory from \$FF40 to \$FF5F, which is the normal I/O area for disk drives. The second line going to G2A is an *address* line. Since this is an active low input, when A4 is low, the chip will be selected. When A4 is high, the chip de-selects. This limits our memory area to 16 bytes and leaves the other 16 for future expansions. The third connection to our chip is the E

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec. Tony's username on Delphi is DISTO.

Adding input devices to an expansion board

Do You Read Me?

By Tony DiStefano
Rainbow Contributing Editor

clock from the CPU. It connects to G1 of our chip. This is an active high input. So when the E clock is high, our chip is selected again. The E clock signal from the CPU is sort of a "data valid" indication. All data is valid when the CPU is writing to a device and the E clock is high. When the CPU is reading, the data is latched (or swallowed) on the falling edge of the E clock.

Those three signals control the selecting of the chip. The next three lines I describe determine which of the eight outputs will be selected, a three-to-eight decoder. Inputs A and B are connected to A0 and A1, respectively. Two address lines in binary represent four locations. The third line is connected to the R/ \bar{W} line of the CPU. Connected to the C input, it divides the eight outputs into two groups of four. The R/ \bar{W} line of the CPU is high for reading and low for writing. This makes one group a write-only select and another group a read-only select. Y0 to Y3 is the write-only group. We know this because we have already used two of the four lines with the controls for the LEDs and motors. The other group, Y4 to Y7, are read-only selects. We will use one of these read-only lines today, to read in data.

That takes care of the decoding part of today's project. We now have a read-only chip select. For the second part, we need a chip we can use as a buffer. Since this chip interfaces to the CPU's data bus, it must conform to some rules. The main rule is that when it is not selected, it must not interfere with the data bus. This condition is called *tri-state*. That

means when the chip is not selected, it must be electrically disconnected (high impedance). Since the CoCo uses an 8-bit bus, we might as well use an 8-bit buffer. Looking through the TTL parts manual, I came across a chip that meets all our requirements — a 74LS244. It is an 8-bit, tri-state buffer.

U4 in Figure 1 is a 74LS244. It has eight outputs connected to the CPU's data bus. It also has eight inputs. These are our eight readable bits. Let's look at the two control lines. There are two because this chip can be controlled as both two 4-bit buffers and one 8-bit buffer. This makes the chip a little more versatile. For our project, we want it to be a single 8-bit buffer, so we will tie both control lines together. The TTL manual states that when the control line of a 74LS244 is high, the outputs are in tri-state mode. This is good because when the 74LS138 is disabled, all outputs are high. The manual also states that when the control line of this chip is low, the signal level appearing on the chip's inputs will appear on the chip's output. This is perfect for our project.

When the CPU is reading the proper location, the 74LS138 will respond by putting Y4 low. This will cause the 74LS244 to generate whatever level (high or low) it has on its inputs to the CPU. If we tied all the inputs of the 74LS244 to ground, the CPU would read \$00 or all zeroes. On the other hand, if we tied the inputs to +5 volts, the CPU would read \$FF or all ones. This is good, but soldering the wires to this chip every time we want to change the condition is a drag. Let's use a switch instead. SW1 in Figure 1 is a quad switch. The diagram shows that it is a PC board-mount DIP switch. This type of switch is generally found on a modem or printer as an option switch, and you can get them at a good electronic shop.

A switch is not the only thing needed for this project. You also need a resistor. Look at the diagram again, and you'll see why. One side of the switch is connected to the input of the 74LS244, and the other is connected to ground. When the switch is on, a direct connection to ground is made. The chip will see that as low, but when the switch is off, no connection is made anywhere. The input to the 74LS244 is just floating — a condition of uncertainty. When the chip is called upon to give the state of

the input, it may give a reading of high or low. It all depends on exterior conditions, such as how close it is to another wire. In order to make sure the input is high, we use a resistor to tie it high. Therefore, when the switch is off, the resistor supplies +5 volts to the input of 74LS244, and the chip reads high. When the switch is on, the current is shunted to ground, and 74LS244 reads low.

The SW1 switch is only a quad switch. That means there are only four switches in that package. The 74LS244 chip has eight inputs. As you can see in Figure 1, I have connected the other four inputs to the outputs of the other chips. This is a way to monitor the output conditions of the other circuits in this project. The wiring in Figure 1 is just an example. You may not want to monitor the LEDs or motors I have selected; you can make any changes you want. For instance, you have a program that turns the first LED on and off in U1 in several places. (See Part I of this project for proper connections of the LEDs.) Using this read-only circuit, you are not certain at any time if the LED is on or off. Using the circuitry discussed in this column, you may now determine the condition of your LED. The same can be done with motors and buzzers.

Now that the theory is clear (I hope), let's look at the construction. You will need different parts for any application, so I'll just describe them and let you decide what you need. First, you need the board you used for the first two parts. For this application, you need one or two 74LS244 chips and one or two 20-pin sockets, depending on how many bits you need to read. For 1 to 8 bits, you need one; for 9 to 16, you need two.

Next, you'll need switches. You can

use any quantity of DIP switches. The diagram shows four, but you can use any number from one to 16. You can also use individual switches and run them off the board, but the wires should

Bit	Decimal	Hex	Binary
D0	1	01	00000001
D1	2	02	00000010
D2	4	04	00000100
D3	8	08	00001000
D4	16	10	00010000
D5	32	20	00100000
D6	64	40	01000000
D7	128	80	10000000

Table 1: Bit Values

be no longer than about 10 feet. In addition, don't run the wires outside. If lightning hits the switches, you'll find yourself shopping for a new computer. You'll need one resistor for every switch you use. As the diagram says, a 10K, ¼-watt resistor will do.

Mount the ICs, switches and resistors close to each other and close to the CPU's data bus. Construction is not too critical, but keep your work neat — it's better for trouble shooting. Try not to spread out your work. Next month I'll add something you might want to add as well. Check your work before turning on the computer. If something feels wrong, turn the computer off right away and check it again. Remember, my diagram does not include power and ground to the ICs; they must be connected. The two ICs you are adding this time require +5 volts at Pin 20 and ground at Pin 10. Also, use two more .1uf capacitors close to the ICs.

Finally, let's discuss the software.

This project uses the CoCo's SCS pin. This maps all I/O from \$FF40 to \$FF5F. (Remember, the dollar sign means it's a Hex number.) To enter a Hex number on the CoCo, just put the characters &H in front of the number. Now, when you want to read the 8 bits connected to U4, the address is \$FF40. The following is an example of a line in BASIC to read the 8 bits at U4:

```
100 X = PEEK(&HFF40)
```

The value returned in X is a value from zero to 255 or \$FF. Each of the 8 bits contribute to the value. If the value returned is zero, then all bits on that IC (U4) are off. In order to find out which particular bit is on or off, you can use the AND command in BASIC to mask the other bits. This command will change any bit that is zero to zero. A full explanation of the AND command can be found in your BASIC manual; I will not go into detail here. I will, however, give you an example of how to do it. Look at U4 in Figure 1. I have connected Pin 13 of U4 to Pin 2 of U1. That means reading U4 and looking at D4 will give you the condition of whatever you poked at U1 D0. If U1 Pin 1 is high, then when you read U4, D4 will also be high. The following is an example of this:

```
10 POKE &HFF40,1
20 X=PEEK(&HFF40)
30 IF X AND &HB <> 0 THEN PRINT "D4 IS HI"
```

The first line makes D0 of U1 high; the second line reads U4; and the third line masks all bits except D4. If D4 is equal to zero, then there is something wrong. To check other bits one at a time, use the values in Table 1 with the AND command.

That's it for now. See you next time when we'll add new input devices. ☺

Note: See Page 140 for Figure #1.

This is the end of our project. Remember, though, we've just touched the surface of CoCo's abilities. You can go beyond this simple project — the possibilities are endless. This time I'll show you how to connect a few more inputs. To do this, we'll delve into the world of optics — light. We can use light to monitor time or trespassers (i.e., determine when it grows dark outside or when someone walks into the light).

Let's start with some electronic theory. Look at Figure 1. Q1 is a symbol for a photo transistor. This one is an NPN. (The *N* stands for Negative and the *P* for Positive.) A transistor has three pins — a base, a collector and an emitter. Figure 2 shows a typical NPN transistor switching circuit. I use the term *switching* because we use it as a simple transistor switch. A simple switch is an SPST (Single-Pole, Single-Throw). The two contacts are the collector and the emitter. Current can only flow from the collector, through the transistor, to the emitter. Examine the circuit in Figure 2. If Point A were connected to ground, there would be no base current flowing from the base of the transistor to the emitter. This causes a high impedance between the collector and the emitter of the transistor (no collector-emitter current). The voltage at Point B would be about the same as VCC.

We'll introduce a base current by raising Point A to VCC. Current will now flow through Resistor R5 and the transistor base and out the emitter, which causes the transistor to conduct. The impedance of the collector-emitter will lower, and current will flow from the collector to the emitter. When this happens, the voltage at Point B lowers as well. If there were enough current flow through the transistor, the voltage at B would drop to 0 volts. The amount of collector current depends on the amount of base current and the *gain* of the transistor. The gain of a transistor is the amplification factor.

The transistors and opto-isolators we will use work in the *saturation* mode,

Tony DiStefano is a well-known early specialist in computer hardware projects. He lives in Laval Ouest, Quebec. Tony's username on Delphi is DISTO.

*Light detectors really
brighten this final
modification*

Lights Out!

By Tony DiStefano
Rainbow Contributing Editor

where we design the transistor to be either fully on or fully off. Look at Figure 2 again. When you ground Point A (no base flow), point B is high. When you make Point A high (when base current flows), Point B is low.

Now that we understand the switching transistor, let's look at the photo transistor. The photo transistor is like a regular transistor. It has two pins and a window. The two pins are the collector and the emitter, and the window is like the base of a regular transistor. Examine the circuit surrounding Q1. It looks like the transistor circuit in Figure 2 but has a window instead of a base circuit. This window acts like the base circuit but uses light instead of current. When there is no light in the window, there is no base current; when there's no base current, there's no collector current. The point at which the photo transistor and resistor meet is high. When there is light, that same point is low. We now have a light-activated switch.

The output of this light switch is connected to Pin 2 of U6 (one of eight inputs of a 74LS244). The circuit in Figure 1 is similar to the circuits in the previous three parts. I just deleted a few ICs to make room for the new circuits. U2 is the same; I just added another 74LS244 chip to Pin 9. The software created in the previous parts of this project is also the same. However, today's additions will use different addresses.

Now that we have the photo transmitter, we need an *opto-isolator*. An opto-isolator is a photo transistor and an LED (Light Emitting Diode) together in one package. As the name implies, this device is used to isolate an incoming signal. This device is used in many places. The most common is in televisions with separate video and audio inputs. In today's TVs, there are no line-voltage transformers. Therefore, many components inside a modern TV can have the potential of 117 volts. This is dangerous and can shock you. Any connection made to the TV is made using isolators similar to the one used here. Electrical signals are converted to light signals by an LED and are returned to electrical signals by a photo transistor.

As in all TVs, my circuit is powered by a separate supply. This supply has to be isolated from the 117-volt AC via a transformer. The circuit surrounding ISO1 in Figure 1 is used in places requiring isolation. It is just a switch (SW1) isolated from the rest of the Color Computer. This switch can be used outside or over long distances of wire without the worry that static electricity or lightning will damage the CoCo. The DC adapter is a standard toy adapter found almost anywhere. If you use more than one opto-isolator, you can use the same adapter. I used a 1K resistor in R3, but this resistor may be a different value, depending on the maximum current for the LED inside the opto-isolator and the voltage of the adapter. To calculate the resistor value, use the equation $R = V/I$. In this equation, *R* is the value of the resistor needed; *V* is the voltage of the adapter; and *I* is the current needed to turn on the LED. You will get this value from the specs on the opto-isolator. When the isolator is wired up, close the switch. This causes current to flow through the LED, which in turn activates the photo transistor. When on, the output is low. When the switch is open, the LED is off and the output is high.

There are many photo transistors and opto-isolators on today's market, and they all work the same. You may have to change the values of resistors to match the different types, but you'll need only a volt meter to make sure it's running right. A wide variety of transistors and isolators are on the market; pick one for yourself. They come in

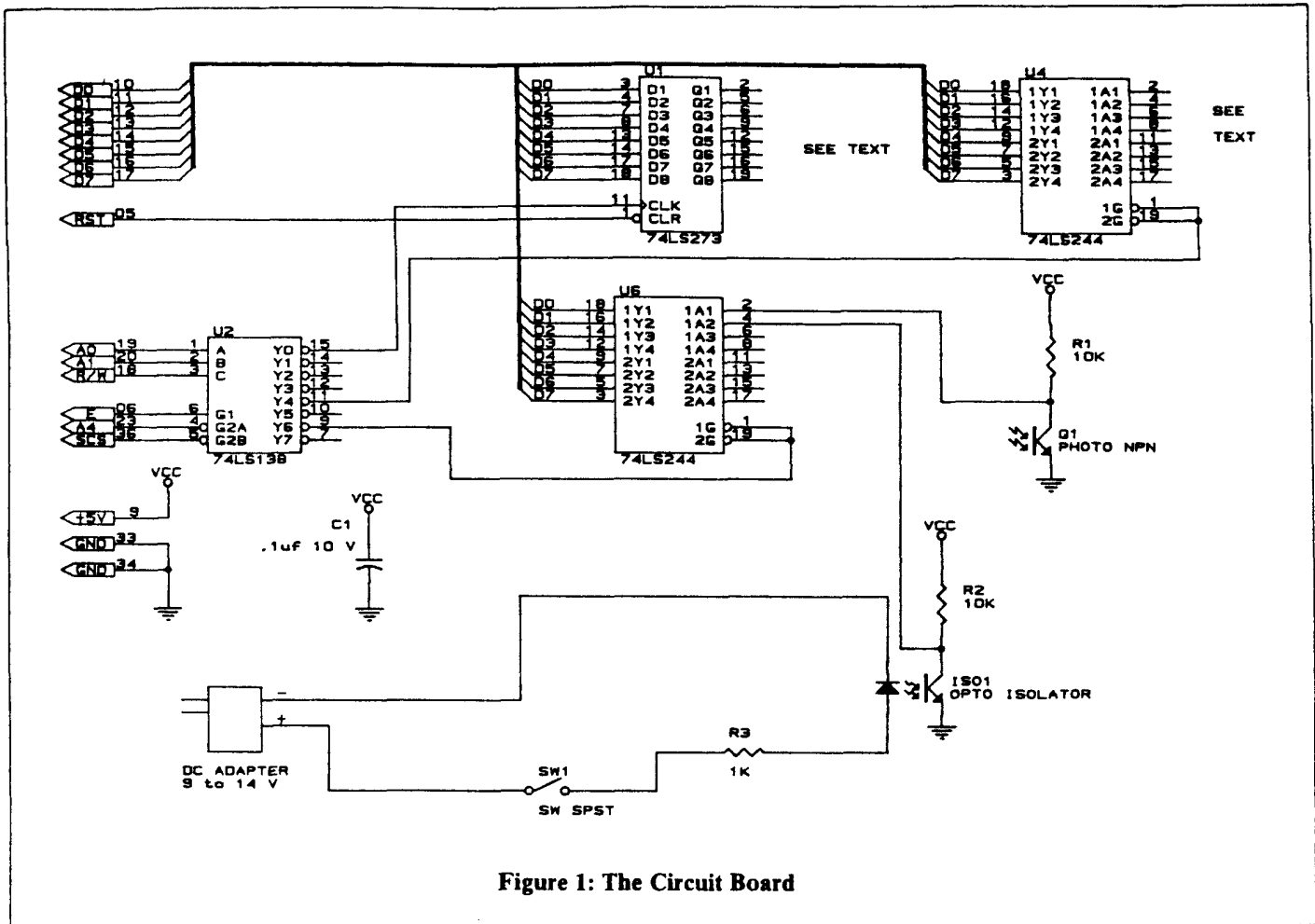


Figure 1: The Circuit Board

different shapes and sizes. Some have built-in lenses or tubes. Some can be mounted on doors or motors, and some come with reflective mirrors. Choose the one you need or want to try. There is an entire series of infrared photo transistors and LEDs. You can build a gadget and write software that reads your television's remote control and duplicates it, so your computer controls your television or VCR. Wire your house for security. You can use a couple of IR pairs and have two CoCos talk to each other without wires — your imagination is the only limit.

Now we need only to create the software. Since we are still using the same SCS pin on the CoCo, the addressing area remains from \$FF40 to \$FF5F. U6 is a read-only device, so only the PEEK command will work in BASIC. If it is connected to Y6 of U2, U6 is located at \$FF42. The same software that read the other locations works here. The same condition applies with the bit positions. In Figure 1, Q1 is connected to D0, and ISO1 is connected to D1. Mix and match these inputs as you like. If eight inputs are not enough, you can use another 74LS244 and get

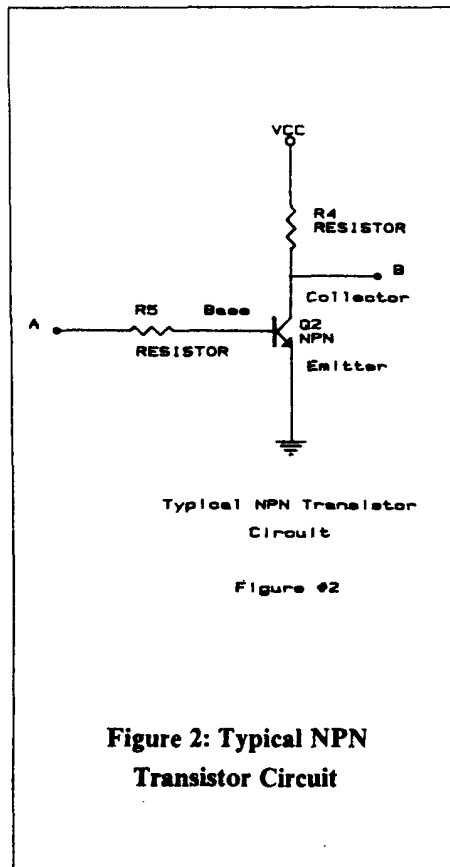


Figure 2: Typical NPN Transistor Circuit

another eight inputs. With U2, you can have 32 (4 x 8) inputs and 32 (4 x 8) outputs. If you need more, add another 74LS138 and an inverter.

Constructing the project is simple — just add to the existing board. Add more sockets and chips as you need them. Many electronics stores carry photo transistors and opto-isolators. Radio Shack stores have a limited selection.

In Part 1 of this project, I told you to keep things neat and tight, and this is why. If you have many wires coming off the board, look into a multiwire connector. It helps prevent wires from breaking when you turn the board upside down to work on it. You may want to start again. Design your own circuit to suit your needs. With the experience you now have, you can make it the perfect size. If you are having trouble reading the photo transistors, use a volt meter to measure the output. Make sure the voltage on the collector is at least 3 volts when no light shines on them and no more than .5 volts when there is light. If this is not the case, use a different value resistor between the collector and the VCC.

For many enthusiastic computer users, understanding the mechanics of their hardware is as essential as pen and paper to a writer. The following article will begin a basic explanation of the 40-track disk drive. Articles to follow will elaborate on various other drives.

First, to define a disk drive: A disk is similar to a cassette tape and a drive is like a cassette player. Both systems use the principal of magnetism, and in both cases the media is made of plastic material coated on one or both sides with a substance containing iron oxide. This makes it sensitive to an electromagnet, called a *head*. Both cassette players and disk drives have heads.

In a cassette player the tape is dragged across the head by a motorized mechanism. In the record mode, a magnetic field is created by the record electronics. This field varies in intensity proportional to the signal it is recording. The varying intensity leaves iron particles in the tape aligned in a specific order. Simply stated, the tape is magnetized while in the record mode. Then the tape dragging across the play head makes tiny magnetic fields that are transferred to electrical signals. These are then amplified to an audible level.

A disk drive's electronics works much the same way. The mechanism, obviously, is different in that it is made with a computer in mind. A cassette is made for continuous music, which makes it inconvenient when you want a small piece of data at the end of a tape. A disk drive, though, is made with the ability to access any part of it quickly.

Let's take a closer look at a disk. It is commonly known as a *floppy disk*, because of its flexibility. The disk most used by the CoCo community is 5¼-inch square and consists of four parts.

The first is the actual media. It is a round piece of plastic, a little over 5 inches in diameter, with a 1⅛-inch hole in the center. Better-quality disks have a second piece of plastic glued to the inner side of the disk to reinforce the mechanism that holds and spins the disk. More on that later. It also has a second hole, about 1/16 inch in diame-

*Heads, sleeves, jackets
and index pulses . . .*

The ABCs of Disk Drives

By Tony DiStefano
Rainbow Contributing Editor

ter, that is about a half-inch from the edge of the inside hole. This is called the *index* hole.

The third part of the disk is called the *jacket*. The jacket serves two purposes. First, it is a protective cover for the media. Touching or bending the media can damage it or completely destroy data. Except for one slot, the jacket completely covers the media. This slot has to be left open so that the read/write head can access the media. The jacket also has a hole on both sides to expose the index hole and another hole to expose that part of the media pinched by the mechanism.

The second purpose is to protect the media from being erased. In the upper right-hand corner of the disk is a small notch. When this notch is left uncovered, the disk drive is able to write to the disk whenever the software "tells" it to. When it is covered with opaque tape the disk drive cannot write to the media, even if the software "tells" it to.

The fourth part of a disk is the *sleeve*, a paper envelope that protects the media from fingers or dust and cigarette smoke. Most people don't realize it, but cigarette smoke creates a thin film of tar

that attracts dust, putting extra wear on the drive heads. Sleeves cover everything from the index hole to the access hole. Whenever a disk is not being used, it should be stored in its sleeve. Never leave a disk in a drive with the door closed over a long period of time. It puts a dent in the media.

Now let's discuss the drive. It is a mechanism used to read and write data to the disk. The first thing a drive does is spin the disk inside the jacket. When you close the door of a disk drive, a plastic hub pinches the disk to the metal hub and shaft of a motor. Older drives had a capstan and were belt driven by a separate motor. Now drives have the motor built right into the hub. When the drive is selected, the motor spins the disk at about 300 rpm (revolutions per minute), give or take 5 rpm. Older drives took up to five seconds to come up to speed; the newer drives can come up to speed within two revs. That's about two-fifths of a second.

The next responsibility of the drive is to properly move the head. The read/write head is mounted on a movable assembly that can move across the access hole in the disk jacket. The heads rub on the moving media. Open the door of a drive and peek in just after a DIR and you will see the back-and-forth motion. The assembly moves with the help of a stepping motor. The head movement is done in steps, with each step being called a *track*.

With 40 of these tracks on each side of its disk, the 360K drive is today's most commonly used drive. The drive is double-sided, meaning that there are two read/write heads, one for each side of the disk. Tracks are numbered from 0 to 39, Track 0 being on the outermost area of the disk and counting up as tracks move toward the center. The head can move back and forth on a pair of rails controlled by a stepper motor that receives one of two signals from the controlling hardware.

The two signals are "step" and "direction." The direction is set according to where the head is and where you want it to go. Then the step pulse is applied, and the head moves the distance of one track in the specified direction. In the case of the 360K drive, the distance between two tracks is about one-forty-eighth inch. That is 48 tracks per inch.

A hardware switch positioned to turn on when the heads are at Track 0 tells

Tony DiStefano is a well-known specialist in computer hardware projects. He lives in Laval Ouest, Quebec. Tony's username on Delphi is DISTO.

the controller where the head is. The proper way to position the head to Track 0 is to give the controller a restore command or to step and test for the switch until Track 0 is detected. Some software steps in 40 times without testing; but if the head is not at Track 40, then it bangs against the Track 0 stopper and can possibly become misaligned. A register in the controller keeps track of where the head is. If the controller confuses where the heads are, it restores to Track 0 and then steps to the desired track.

Another duty of the drive is detecting *index* pulse. The little hole in the disk is used to give the controller a reference point. Inside the drive on one side of the hole is an IR (infra-red) LED. On the other side there is an IR detector. When the disk is spinning, most of the time the light emitted by the LED is blocked by the disk. Every revolution of the disk, the hole appears in the path of the LED and detector. This in turn gives a short pulse to the controller. By this signal the controller can determine a reference point to the rotational position of the disk.

This position reference is used when formatting new disks. Formatting divides the disk into small blocks called *sectors*. Each sector has a unique address or ID number. They are assigned by track number, sector number and side. Some controllers, however, do not use side but, instead, have greater sector numbers.

As mentioned earlier, tracks are numbered 0 to 39. In CoCo's case,

sectors contain 256 bytes of data each. There are 18 sectors per track per side. Radio Shack DOS is written to handle a single-sided drive with 35 tracks at 18 sectors per track. That gives you a total of 256 bytes x 18 sectors x 35 tracks = 161,280 bytes per disk.

Since most drives today can step 40-tracks and are double-sided. This is a waste of data area. Some third-party DOSs get around this by changing it to

Pin #	Function
2	N/C
4	N/C
6	D4 Select
8	Index Pulse
10	D0 Select
12	D1 Select
14	D2 Select
16	Motor On
18	Direction
20	Step
22	Write Data
24	Write Gate
26	Track 00
28	Write Prot
30	Read Data
32	Side Select
34	N/C

Table 1: Standard Connector for a 360K Drive

handle double-sided and 40 tracks.

When formatting, the controller does one complete track at a time. The index pulse is used to start the writing head up and then to shut it off. This keeps the write head from writing over the part already written on.

So far, I have been talking about the mechanical parts of a disk drive, but there is more — the electronics part.

A disk drive has several electronic sections in it. Though the actual electronics varies, there are standard protocols that make drives made by different companies compatible. This is called the interface. All drives use a 34-pin edge connector to transfer all electronic information to and from the controller. All the pins do basically the same thing. You can virtually unplug a Panasonic 360K drive and plug in a Tandem without any problems. Table 1 shows a pin list of the standard 360K drive connector.

All odd pins are ground returns. These signals completely control the drive. The electronics needed for this task are speed regulation for the spinning of the drive, stepping the head in and out, electronics to power the write head and erase head, and amplifiers to read the small signal of the read head and to light the "drive in use" LED.

Now you should have a good idea of how a disk drive works. Next time, I'll discuss how an 80-track drive is different and include a circuit on how to double-step the drives so it can read standard 40-track disks. ☺

*Stepping into the world of
40- and 80-track drives*

The DEFs of Disk Drives

In last month's column, I covered the ABCs of how a disk drive works — its mechanical parts and how it accesses the data available on the disk. I'll continue on that track, giving more detail to the differences between 40- and 80-track drives. Part of the article will concentrate on designing a small circuit that allows CoCo users to read standard CoCo disks with an 80-track drive.

The need for 80-track drives came about with the need to store more data on one disk. If a 40-track double-sided drive can hold 360K of data, then an 80-track double-sided drive should hold 720K of data. In fact, it does. But instead of going back to

the 8-inch drive, which has more data storage, the manufacturer decided to double the amount of data by doubling the amount of tracks on the same-size disk. The only problem with this is that it becomes incompatible with the 40-track drives. The differences make it impossible for an 80-track drive to read a 40-track disk.

One difference between the two is obviously the number of tracks. But how is that possible, when both are 5¼-inch drives? Well, the difference is in track size. On a 40-track drive the track density is 48 TPI (Tracks Per Inch). At 48 TPI, it takes just under one inch to make 40 tracks. If you look at a disk, one inch is about enough

room to fit 40 tracks. If 40 tracks take up one inch, then 80 tracks take up two inches; that's too much to fit on a 5¼-inch disk. So the disk drive manufacturers decided to make the tracks thinner and closer together. To make them fit on the same size disk, the track density was doubled to 96 TPI. That allows 80 tracks to fit on the same size disk.

This, however, causes a few problems for both the drive and disk manufacturers. So the read and write head had to be made thinner and the stepping mechanism more accurate. This adds to the cost of the drive. In addition, the disk has to hold twice the data and be of better quality. Since the track size is smaller (thinner), the magnetic surface is smaller. In order to get the same reliability, the quality must be better — both with the heads and disks. When using 80-track drives, it is recommended that you use 96 TPI-rated disks. If you don't, you may not have any problems while the disks are new, but in the long run valuable data is safer with this type of disk.

Now, let's step back a little. The mechanism that steps the head back and forth is usually a motor called a stepper that can precisely rotate within certain speed limits. When Radio Shack first started selling drives, it took 30ms. (milliseconds) to make each step, but as motors improved, drives had shorter stepping times. Today an average 40-track drive has a stepping time of 6ms. When the 80-track drives came out, the manufacturer wanted it to be just as fast, so they increased the stepping time again to 3ms.

Look at Figure 1. It shows a few tracks on a typical disk. On the left side of the drawing are tracks made by a 40-track drive at 48 TPI. The track on the outer edge is Track 0; the next is Track 1, then Track 2 and so on. Tracks made by an 80-track drive are twice as thin as those of a 40-track drive. Notice, though, that Track 0, is on the outer edge on both sides.

Take a disk formatted in a 40-track drive and place it in an 80-track drive. If you step the 80-track drive to Track 0, you can read it; trouble starts when you want to read the next track and so on. Look again at the right-hand side of Figure 1. Imagine that you step the 80-track side one track inward to Track 1. Now move over to the right-hand side and see where you are. On the 40-track side, you are still on Track 0, yet the software expects Track 1. Now step

in again. The software expects Track 2 but gets Track 1. For every track stepped, the result is half of what you expect. If you step up to Track 10, then you only get Track 5.

Stepping in or out, the ratio is always 2 to 1. Knowing this, I thought I could make a circuit that would generate two pulses for every one that came in. It would then be possible for an 80-track drive to read a standard 40-track disk. After a few experiments I came up with a doubling circuit. For every step pulse coming into the drive, two pulses come out.

When stepping a standard 40-track drive, the CoCo's controller waits a minimum of 6ms between steps. For an 80-track drive with a 3ms step rate, this is relatively slow. In fact, it can step twice as fast, so the circuit has time to step between steps.

Examine the circuit in Figure 2 used to make the double stepper. It consists of two TTL chips and a handful of passive components.

First it takes one pulse that comes in and changes it into two pulses. U2B acts as a buffer so that the second pulse doesn't trigger the circuit into oscillation. U1 is a dual *monostable multivibrator*. The first part (U1A) is used as a delay. The pulses that come in on STI are very short and are coming in at every 6ms. I say short because they are short compared to the circuit's delay of 3ms between pulses. That is

half the time between incoming pulses. (Remember that an 80-track drive can step every 3ms.) When a pulse enters into the A input of U1A, Q* (Pin 4) goes low and stays low for 3ms. Nothing happens until Q* goes high again. The B input of U1B circuit starts on the rising edge of Q*. When this pin gets a rising edge, it starts timing a much shorter pulse, about 4µs, the same pulse length as the incoming step pulse.

Now let's look at what happens to the STO point in the circuit. The first (original) pulse happens; STO sees one pulse; that triggers a pulse at U1A; about three milli-seconds later, a pulse triggers U1B. If the switch S1 is closed, the short pulse generated by U1B (4µs) goes through U2A and appears at STO. At that point the drive gets a second pulse to step. If S1 is opened, the pulse goes nowhere.

Construction for this project is not difficult. Besides parts, it requires opening your drive case and modifying the drive, which takes some electronic skills and should be done only by someone with experience in soldering and circuit modifying.

Concerning parts, look at the circuit in Figure 2. These are all the parts you need — four resistors, four capacitors, two chips and one switch. You'll need a small protoboard on which to mount all the parts. These are available at any Radio Shack

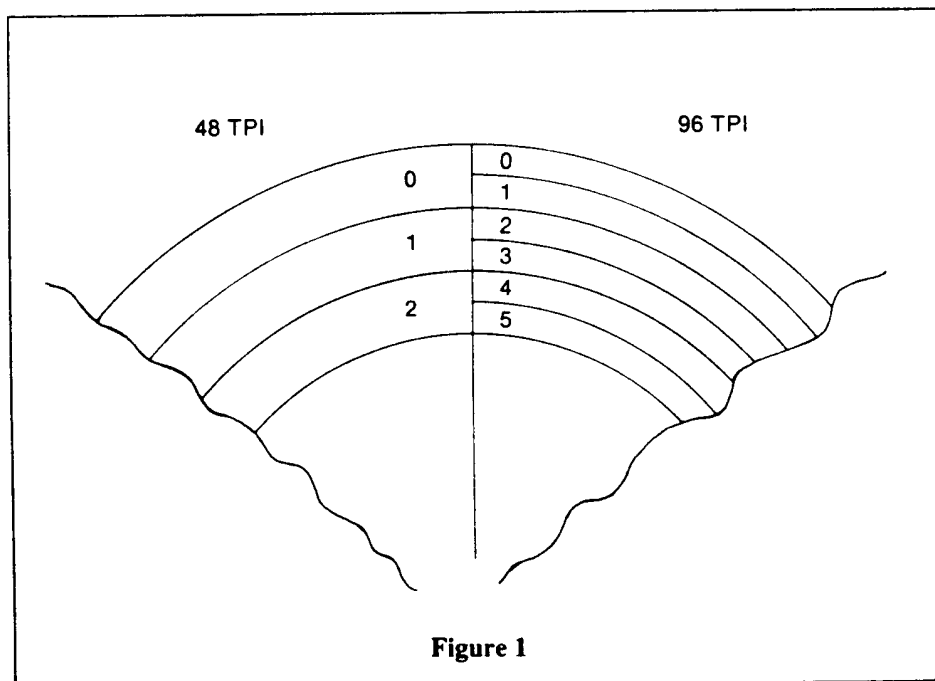


Figure 1

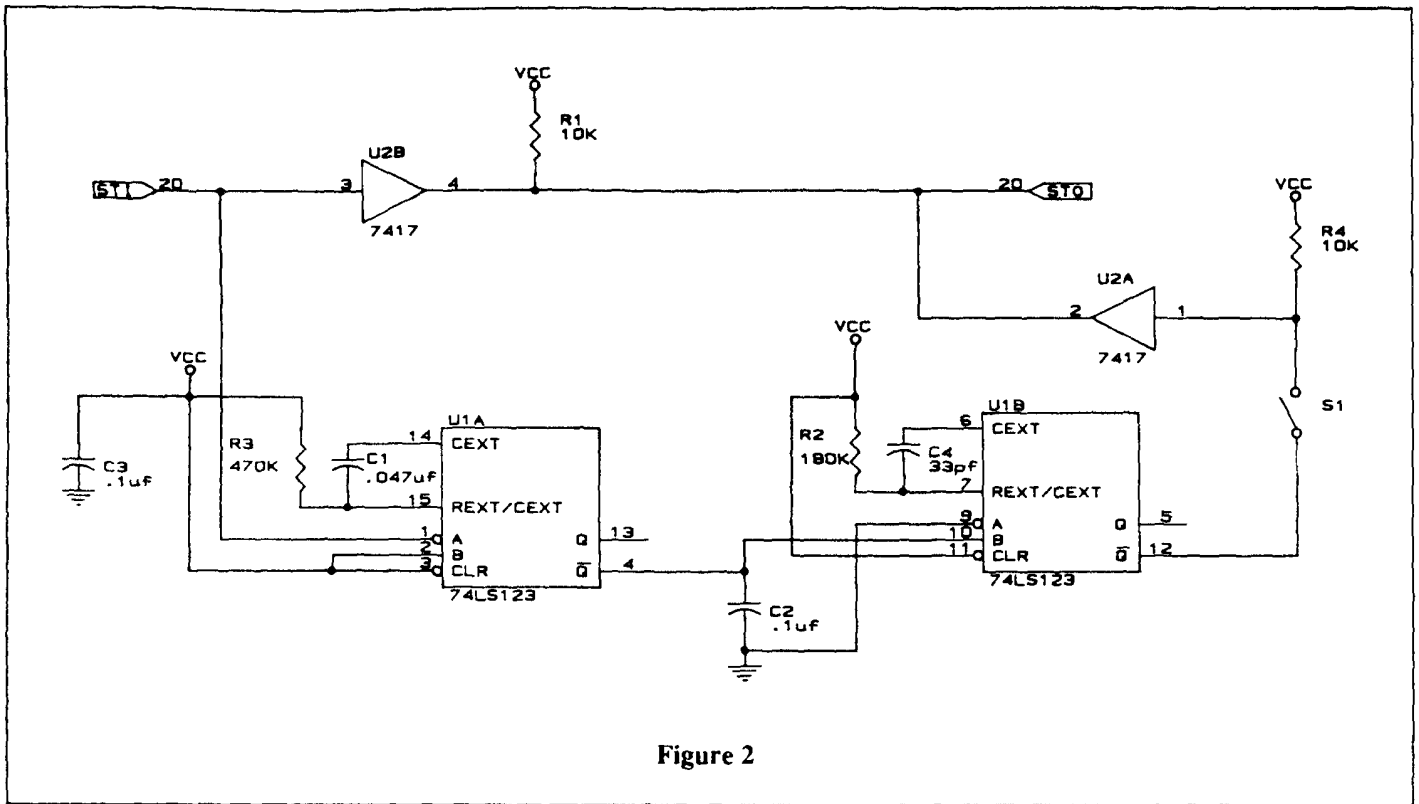


Figure 2

store, unlike some of the other parts.

Connect all the pins to the chips; unmentioned pin numbers should be left unconnected. Pin 16 of U1 and Pin 14 of U2 should be connected to +5 volts. Pin 8 of U1 and Pin 7 of U2 should be connected to the ground. After all the components are mounted on the small board, it's time to mount the whole thing into your drive. I can only give you guidelines since the great variety of 80-track drives makes it difficult to be exact.

First you need to find a place to fit the board — once fitted, you have to connect

5 volts and ground. A voltmeter here is handy but not necessary. Locate the power connector to the drive. There are 5 volts, 12 volts and ground at the connector. Pin 4 is 5 volts and pins 2 and 3 are ground. The next step is to find the 34-pin edge connector. Locate Finger 20 and a convenient location, then cut the trace that leads to it. Solder the connection labeled STI to the side of the cut that leads to the finger, and solder another connection labeled STO to the other side of the cut. Mount the switch somewhere on or near the front of the drive, then reassemble the drive assembly

and turn everything on.

Now insert a 40-track disk in the 80-track drive, turn the switch on, and type DIR. If it's not working, check your work; if you have a digital probe, use it.

Now that you have the circuit working, you need to know how to use it. While in OS-9, leave the switch off. This allows you to access all 80 tracks. (You must use the 80-track descriptor.) When you want to read standard 40-track disks, turn the switch on, use a 40-track descriptor and read the disk. Do not try to write on a 40-track disk with an 80-track drive. It will not work properly. ☺

Dynamic Random Access Memory Explained

Making refresh and page modes everyday conversation

Just about everyone and his brother in the computer business knows about RAM, Random Access Memory. But how much do you really know about it? Most users know enough about it to get by and how much RAM is needed to do certain things. Some years ago, many programs required only 16K. Then there was the 32K memory craze, with everyone using the piggy-back technique. Moving on to 64K was then the limit for the CoCo. When the

bank-switching technique arrived, everyone used it, breaking the 64K barrier. The CoCo 3 brought 128K, expandable to 512K. But as a hacker, you must know more than just how much memory your computer has. It is important to know the kinds of RAMs available and how these work. I will quickly review the basic concepts of RAM, then discuss the finer details of DRAM, or Dynamic RAM.

Let's start by reviewing a static RAM. Figure 1 shows a 2K-by-8 static RAM chip in a 24-pin package with Vcc and GND. The Vcc is 5 volts, all that is needed to power this chip. There are eight data lines

labeled D0 to D7, then 11 address lines, A0 to A10. To understand why there are 11 address lines, remember binary numbers. Each line has two states, Hi and Lo; for every extra address line added, the amount of memory doubles. For 11 lines it is 2 times 2 times 2, eleven times. That gives a total of 2K or 2048. There is also a sole Read/Write line and two Chip Enable (CE) lines. This accounts for 24 lines.

That is how a static RAM chip works. When the CPU reads or writes to RAM, it puts out an address first. Any data written into a static RAM chip stays there until power is removed from it or it is changed by the CPU or other device. Each memory location is made up of a flip-flop circuit. When flipped, it stays flipped; when flopped, it stays flopped — thus the name *static*. It takes up two transistors and a support circuit for each cell, as well as a lot of room on the chip, adding to its cost. This is one of two major differences between static and dynamic memory.

In general, dynamic memory has a much higher capacity than static memory, over 100 times greater than the 2K static RAM chip. There is not enough room on a small IC chip for all those transistors so the IC designers made a small change in the design to save both room and money. The standard flip-flop memory cell was changed to one transistor and capacitor, the capacitor becoming the new memory cell. When the memory cell was given a Hi, the capacitor was charged; when requiring a Lo,

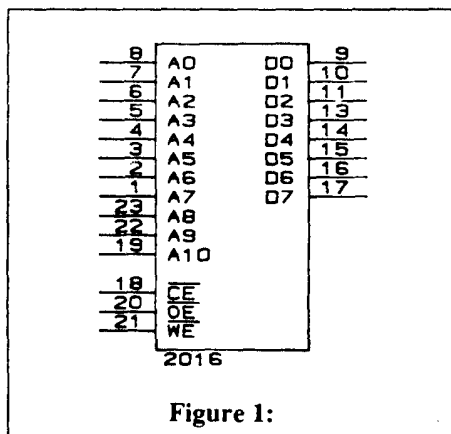


Figure 1:

it was discharged. When reading the data, a sense amplifier reads voltage across the capacitor, which, if above a certain voltage, is considered to have a Hi. If not, it has a Lo. This worked well to lower cost and real estate.

However, one small problem is that when the capacitor memory cell is not accessed for a while, the capacitor dis-

charges due to leakage. When the sense amp reads the voltage, it is not high enough to convince the amp that it is Hi, so data is lost. The designers added extra circuitry to refresh (recharge) the capacitor occasionally before voltage gets too low. The voltage across the capacitor is dynamically changing, dropping when it leaks and rising when it's recharged — thus the term *dynamic refresh*.

This took care of price and space for higher-capacity memory chips, but there is also another problem. The small chip needs a small package, but with high-memory capacities come many address lines. For instance, a 256K-by-1 memory chip requires 18 lines for addressing alone. Add the data and control lines and you have a big package. In order to cut down on address pins, the chip multiplexes these lines. The dictionary definition of multiplex is: "equipped to transmit two or more sets of signals in one or both directions simultaneously over the same wire or radio band." We are not dealing with radios, but the rest of the definition applies, cutting the address lines almost in half. There now is a need for other control lines to allow the chip to recognize when it's the first set of address lines and when it's the second. The savings are great enough to warrant the extra circuitry both inside and outside the package.

Those are the major differences between static and dynamic memory. For more details on how dynamic memory works, study the diagram in Figure 2, which shows the pin-out of the well-known 41256 memory chip. It is the 256K-by-1 memory chip commonly used in the CoCo 512K, IBM PC, AT, PS/2, Atari ST, Commodore Amiga, Apple MAC, SE, MAC II and all the clones. It is also used in video processors, VCR electronic pauses, TV Screen on Screen, video freeze frames, laser printers, electronic typewriters, telephone systems, musical electronic keyboards and so on. No wonder there was a shortage! But this chip has just 16 pins and only one data bit. That is to say, it requires eight of these chips to make 256K-by-8 memory.

When we compare this chip with the 2K-by-8 static RAM chip in Figure 1, there are many similarities. Both share Vcc and GND, address and data lines, as well as the R/W line. But instead of Chip Enables, there are RAS and CAS lines that serve many uses. They are used for refresh, multiplexing address lines, and serve as Chip Enables. Information about these

areas is necessary for a good understanding of the dynamic memory chip.

Since the address lines are multiplexed and are the first thing the memory chip needs to operate, let's look at these first, while following the block diagram in Figure 3. Fully decoding 256K requires 18 address lines, A0 to A17. The 41256, with only nine address lines (A0 to A8) uses the RAS (Row Address Strobe) line to strobe

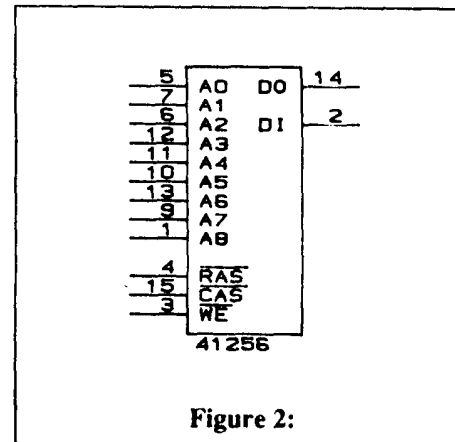


Figure 2:

the addresses A0 to A8 into the DRAM chip. It is the responsibility of the computer's support circuitry to generate all the signals required by the DRAM chip. (Critical timings are not discussed here in order to keep things simple. Remember, though, timing is very important and must be respected by the support circuitry if the DRAM chip is to work. For example, in the CoCo the GIME chip takes care of all timing requirements for the DRAM.)

First the address signals must be stable on the address bus, then the falling edge of RAS locks the lower address into the row address buffer. After the RAS line has done its job, the CPU's A0 to A8 must be taken off the chip and replaced with the CPU's A9 to A17, which is accomplished by the CPU's support circuitry. When A9 to A17 appears on the DRAM's address bus, (A0 to A8) the CAS (Column Address Strobe) does its job. On the falling edge of CAS, the upper address is locked into the column address buffer. These buffers (row and column) feed into the row and column decoders that access the sensing amps and then the memory cells themselves.

When the RAS/CAS sequence occurs, the chip is selected and, depending on the Read/Write line, a read or write cycle is completed. If a read cycle was executed, the DOUT pin will have the data from that cell; if a write cycle was executed, the data present at DIN is transferred into the accessed memory cell. When all is finished,

the cycle starts again, first with the RAS, then CAS; then data becomes valid. All this is known as one memory cycle.

A typical DRAM chip can handle several different modes: Read-Modify-Write, RAS Only Refresh, Hidden Refresh, Page Mode, and Nibble Mode. They are all slight variations of the same Read/Write cycle, which you will understand better as we continue.

Reading and writing data is all the CPU does as far as memory goes, but the DRAM has one more requirement — refresh. I explained why the DRAM needs refresh and will now show you how it's done.

Most DRAMs on the market today require RAS Only Refresh. If you look at Figure 3, you can see that the Row Decoder has only eight lines, meaning that only 256 refresh cycles are required in order to keep all data refreshed in the DRAM chip, keeping the refresh circuitry to a minimum. An eight-bit counter along with its support circuitry is required.

There are many ways of refreshing a DRAM chip, depending on design factors. As long as each of the 256 RAS locations are accessed once every 4ms, the refresh is satisfied.

In software the CPU simply has to make 256 reads or writes every 4ms. This is low-cost but not very practical because it takes up a lot of CPU time. If the CPU has the time, great. The most common way is to let the video circuitry do the work since most video circuits are bit-mapped, or have at least one bit-mapped plain. If video is unavailable, an independent circuit usually does the trick. Again, there are a couple of ways to approach this. One is to put in a refresh cycle when the CPU doesn't need the memory. The only problem is that there needs to be at least 256 free spots every 4ms. Another way is to make the CPU wait every time you refresh.

The Hidden Refresh method involves strobing in a refresh cycle in the middle of the CAS cycle. Since the CAS buffer is latched relatively early in the CAS cycle while the DRAM is fetching data, the cycle can be squeezed in. With CAS al-

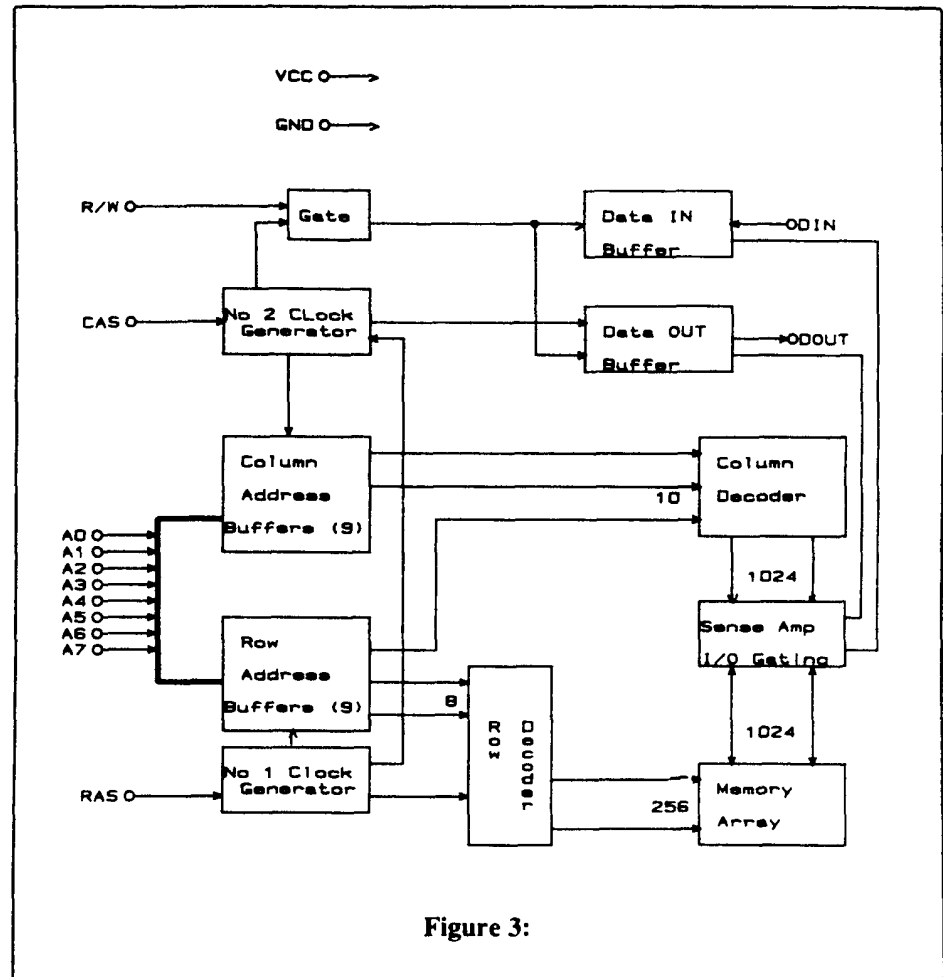


Figure 3:

ways low, the CAS address is taken off the address bus, and the Refresh data counter is presented to the DRAM's bus. The RAS strobe is then fired and the refresh cycle is completed.

Page Mode is for faster I/O more than anything else. The mode may be used when many column accesses are needed within the same RAS area. This is done by latching the RAS as usual but then doing many CASes without deselecting the RAS signal. This mode is used when speed is needed without an increase of power.

The Nibble Mode operation allows faster successive data operation on four bits. The first of four bits are accessed as usual.

Then by keeping RAS low, CAS can be accessed four times to get the four bits each in the next three pages at a rate faster than accessing them separately.

Not all these modes are available on all DRAMs. You must refer to the data sheets of each particular chip in order to see if the feature you need is available. This article by no means includes all the data on DRAMs. I have left out timings, chip loads and many other small details. If you want to design a circuit involving DRAM, make sure you know a lot about the chip itself and the system you are designing it for before starting. More specific details can be found in the DRAM data manuals. ☺