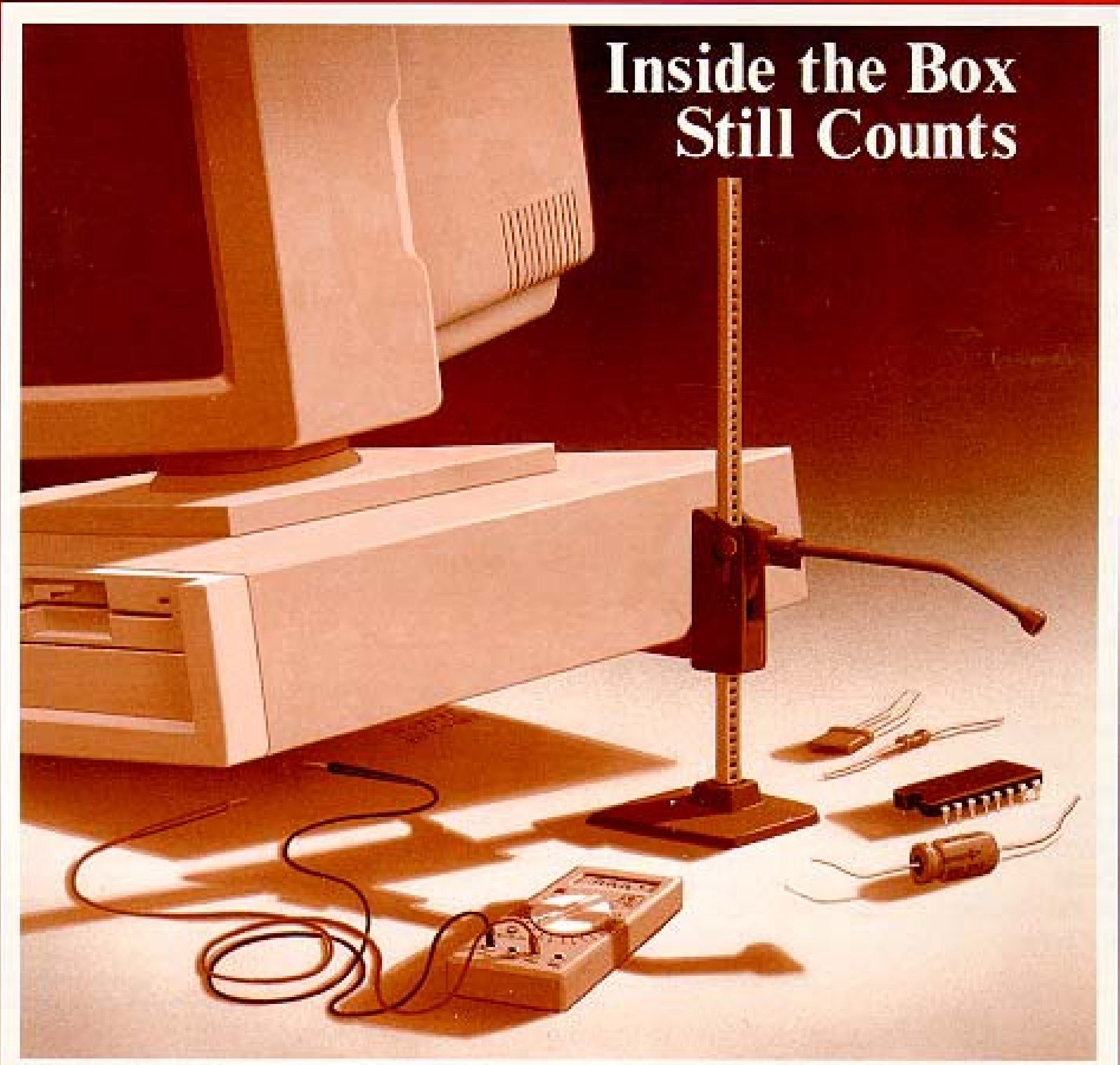
Circuit Cellar INK

MICROCOMPUTER APPLICATIONS



January/February 1988 -- Vol. 1, No. 1 -- Premier Issue

$3.95

CIRCUIT CELLAR INK
MICROCOMPUTER APPLICATIONS

PUBLISHER
Stephen J. Walters

EDITORIAL DIRECTOR
Steve Ciarcia

EXECUTIVE EDITOR
Harv Weiner

TECHNICAL EDITORS
Kenneth Davidson
Je jj Bachiochi

CONTRIBUTING EDITORS
Thomas Cantrell
Edward Nisley

CIRCULATION DIRECTOR
Jeannette Do jan

CIRCULATION ASSISTANT
Diane Morey

PRODUCTION MANAGER
Tricia Dziedzinski

BUSINESS MANAGER
Daniel Rodrigues

STAFF RESEARCHERS

Northeast
Eric Albert
William Curlew
Richard Sawyer
Robert Stek
Midwest
John Elson
Tim McDonough
West Coast
Frank Kuechmann
Mark Voorhees

Cover Illustration by Robert Tinney

TABLE of CONTENTS

CIRCUIT CELLAR INK is published bi-monthly by Circuit Cellu Incorporated, 4 Park Street, Suite 12, Vernon, CT 06066 (203-875-2751). Second-class postage paid at Vernon, CT and additional mailing officer. One year (6 issues) charter ● ub8cription rate U.S.A. and possessions $14.95, Canada $17.95, all other countries $26.95. All subscription orders payable in US fund8 only, via international postal money order or check drawn on US bank. Direct subscription orders to Circuit Cellar INK, Subscriptions, P.O. Box 3378, Wallingford, CT 96494.

POSTMASTER: Please send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 3378, Wallingford, CT 66494.

Circuit Cellar BBS - 24 Hrs. 300/1200/2400 bps
8 Bits, No parity, 1 Stop Bit
203-871-1988

EDITOR'S INK

Inside the Box Still Counts

Chinese restaurants are interesting places. I could say that the only reason I go is to get my latest fix of General Tso's spicy hot chicken and Tangerine beef, but, any escape from the humdrum regularity of business responsibility must be seen as an opportunity. To disguise the necessity of this occasional, but necessary adventure, I often invite colleagues along and pass the event off as business with social overtones. Amid the confused scents of odoriferous garlic and pepper-laden concoctions I get the necessary inspiration to return for an afternoon of work.

On one of these recent business luncheons, we were interrupted by a very boisterous individual at the next table. Chinese restaurants are usually like libraries - waiters and patrons talk almost in whispers. Perhaps that's what made this such a relaxing place. In any case, I would have ignored this very loud individual except that he was talking about the computer business. Unfortunately, his volume combined with a sensitive subject generated a priority interrupt and I found myself listening to his conversation.

Generally speaking, loud restaurant patrons take very little time to divulge their entire life's story and this man was no exception. He used to be a copier salesman and now sold personal computers for one of the computer store chains. The people with him were prospective large volume purchasers of PC clones for a medium size business. What struck me most vividly about his conversation was the way he described his informed view of the computer industry.

"The world is divided into appliance PC users and hackers (his use of the word "hacker" was not complimentary). Computers have evolved to the point where they are merely appliances supported by thousands of plug-and-go application programs. True computer users these days are the programmers who design and sell software which further fosters this appliance concept. Real computer people don't want to know what's in the box and don't care."

Unfortunately, this man has the mistaken impression that the world is only made up of volume business computer purchasers. Like a child who thinks that milk comes from grocery stores, our transplanted copier salesman and too many others like him, neglect the innovative designs and engineering talents that led to the production of their adopted appliance. Using his logic, we too could easily decide to eliminate teaching people to read now that television is the major medium for disseminating information.

It took a lot of effort not to stand up and jam a piece of garlic chicken in the man's ear. What it did do however, was strengthen my resolve to further the education for and communication with those individuals intelligent enough to recognize that sometimes it is very important to know EXACTLY what's in the box and how it works. Without a continuous effort at understanding and improving present achievements, we cannot progress to higher levels of achievement.

Circuit Cellar Ink is a publication designed to increase that awareness. We must not ignore the fact that it is a combination of people AND machines that create intelligent personal systems. Whether they be applied as toaster-like appliances thoughout an industry, serve as the control system of a CAT scanner, or function as a video arcade game, the basic ingredients of computers are similar. It is the continual evolution of a computer's concept, design, and application which ultimately results in the perfection of the truly Intelligent Personal System.

Steve Ciarcia

VISIBLE INK

Answers; Clear and Simple

Dear Ink,

I am a student of Electrical Engineering at Brigham Young University and have been studying the Image Wise gray-scale video digitizer. I would like to do a similar project for one of my classes.

First, on the transmitter/digitizer board: How does the circuit or microprocessor distinguish between horizontal and vertical sync pulses? The one-shot does not seem to make any distinction, and I don't know if the processor can determine the length of a pulse. Second, in the Turbo Pascal program that reads the picture data into a **PC,** how do you get around the 64K-byte restriction on the stack for program variables? It seems to me to compare two pictures, at least twice that much would be necessary, unless a messier linked-list approach was used.

Robert B Smith
Provo, Utah

Dear Robert,

It turns out that the way the ImageWise tells a vertical sync from a horizontal sync is by timing. Horizontal syncs are a few microseconds wide and vertical syncs are a few tens of microseconds wide, roughly half a line.

The ImageWise transmitter gets an interrupt on the leading edge of each sync pulse. It delays about a third of a line and looks at the sync input: if it's still active, it must be a vertical sync pulse. That's all there is to it!

The Turbo Pascal programs allocate 62K image buffers from the heap using the New () function to get a pointer. The Turbo heap has a few hundred kilo bytes available in a 640K machine, so there's enough space for a few images. The code looks something like the following:

```
TYPE
 pelrng    = 0..255;              { pel indexes
 linerng   = 0..243;              { line indexes

pelrec   = RECORD              { one scan line    }
           syncL  : BYTE:
           pels   : ARRAY[ pelrng] OF BYTE:
           END:

linerec  = RECORD              { whole picture    }
           syncF  : BYTE:
           lines  : ARRAY[linerng ] OF pelrec;
           syncE  : BYTE:
           END:

picptr   = ^pictype;           { picture pointer )
pictype  = linerec;            { picture data     }

VAR

pic1 :     picptr;             { pointer variable }
pic2   :   picptr;             {  another  one    }
rownum : linerng;              { array indexes    }
pelnum  : pelrng;

BEGIN

 ... lots of code here...

New(picI);                     { get heap storage }
New(pic2);                     { and some more    }

 ... more code here to load a picture...

 ... to refer to the picture data, you do this...

FOR rownum := 0 TO 243 DO
 FOR pelnum := 0 TO 255 DO
  IF pic1^.lines[rownum].pels[peInum] =
pic2^.lines[rownum].pels[pelnum]
    THEN BEGIN
     and so forth
    END;

--INK
```

Dear Ink,

My question(s) regards pattern recognition software. After reading several references made to it in recent consumer magazine articles I have developed a great interest but have precious little information about it.

Is there such a thing as a general pattern recognition software program; one that might be applied to a subject where there exists a recognizable pattern of events? Are any more specific pattern recognition programs available? Would you please recommend a good book on the subject? Thanks.

Bruce **Miller**
Los Angeles, CA

Dear Bruce:

As far as I know, pattern recognition programs must be written for each application: there's no pattern to the way patterns** are **recognized.

The problem is that each topic requires a different way to represent its data: musical notes use frequency, overtones, timing information, and a pitch reference, while technical stock market analysis uses values and dates. The same program can't handle both problems.

There are some tools available to simplify the pattern recognition task, but they tend to run on computers considerably more expensive than the usual PC clone. Indeed, the software alone tends to cost more than the typical house.

And, as always with computer topics, the journals dealing with the subject are clogged with fancy mathematical notation that (we think) tends to obscure what's really going on. You'll need a solid background in calculus and logic to make any headway.

The best way for you to get acquainted with the field is to drop into the UCLA or USC library and look in the file cards under "pattern recognition" -- then wander through the stacks until you find the wall devoted to the subject!

--INK

Dear INK,

I develop software applications (mostly custom) for the IBM PC and compatibles. I currently use a ZENITH XT clone. However, with the announcement of IBM's PS/2 and Microsoft's OS/2 I would like to start preparing for the oncoming changes. From what I read OS/2 is only for 286 or higher based machines. What I want is to be able to upgrade my system to cope with the new DOS, giving me the possibility to develop new applications under OS/2 Are there any 286 or 386 boards available that would do this to my computer?

Jose Carlos do Santos Mendes
Portugal

Dear Jose:

As of right now, you can buy a PS/2 machine from IBM along with DOS 3.3 to run all (well, almost all) of your favorite applications. And if you've got $3000 to burn you can get the Microsoft OS/2 developer's kit to start writing those protected-mode programs. But that's it.

The official Microsoft position is that OS/2 will run on any IBM AT clone, regardless of what's in the proprietary IBM PS/2 hardware. IBM, of course, isn't saying anything.

There are a number of subtle issues that may get in the way of** OS/2 **compatibility. For example, the standard AT BIOS that you're used to won't work for OS/2: it doesn't run in protected mode. The PS/2 machines include additional BIOS routines that will run in protected mode. So, will the clone makers have to provide a protected-mode BIOS, will one be included with the Microsoft version of OS/2, or what? We don't know yet.

We suspect that the add-in boards that promise to turn your XT into an AT clone will also run OS/2, but there's absolutely no way to tell without the final code. Buying one now is almost certainly a guarantee that you'll have one of the few boards that simply doesn't work.

If you want to get started with OS/2 right now, the only way to do it is to buy an IBM PS/2 and drop $3000 on the OS/2 developer's kit. The only other choice is to wait until the dust settles, then buy a board and OS/2 that are guaranteed to work together.

Anything else is just vaporware. -- INK

In Visible Ink, the Circuit Cellar Research Staff answers microcomputing questions from the readership. The most representative questions are published each month as space permits: Send your inquiries to: INK Research Staff, c/o Circuit Cellar INK, Box 772, Vernon, CT 06066. All letters and photos become the property of CCINK and cannot be returned.

Motion Triggered Video Camera Multiplexer

by Steve Ciarcia

One of the most successful Circuit Cellar projects ever was the ImageWise video digitizing and display system (BYTE, May-August '87). It seems to be finding its way into a lot of industrial applications. I suppose I should feel flattered that a whole segment of American industry might someday depend on a Circuit Cellar project, but I can't let that hinder me from completing the project that was the original incentive for ImageWise Let me explain.

How **it all started**

When I'm not in the Circuit Cellar I'm across town at INK or in an office that I use to meet a prospective consulting client so that he doesn't think that I only lead a subterranean existence. Rather than discuss the work done for other clients to make my point, however, I usually demonstrate my engineering expertise more subtly by just leaving some of the electronic "toys" I've presented lying around. The Fraggle Rock lunchbox with the dual-disk SBI 80 in it gets them every time!

ImageWise was initially conceived to be the "piece de resistance" of these hardware toys. The fact that it may have had some commercial potential was secondary. I just wanted to see the expressions on the faces of usually stern businessmen when I explained that the monitor on the corner of my desk wasn't a closed-circuit picture of the parking lot outside my office building. It was a live video data transmission from the driveway at my house in an adjacent town.

Implementing this video system took a lot of work and it seems like I've opened Pandora's box in the process. It would have been a simple matter to just aim a camera at my house and transmit a picture to the monitor on the desk but the Circuit Cellar creed is that hardware should actually work, not just impress business executives.

ImageWise is a standalone serial video digitizer (there is a companion serial input video display unit as well) which is not computer dependent. Attached to a standard video camera, it takes a "video snapshot" at timed intervals or when manually triggered. The 256x244-pixel (64-level grayscale) image is digitized and stored in a 62K-byte block of memory. It is then serially transmitted either as an uncompressed or run-length-encoded compressed file (this will generally reduce the 62K bytes to about 40K bytes per picture, depending upon content).

An ImageWise digitizer/transmitter normally communicates with its companion receiver/display at 28.8K bits per second. Digitized pictures therefore can be taken and displayed about every 14 seconds. While this might seem like a long time, it is quite adequate for surveillance activities and approximates the picture taking rate of bank security cameras.

"Real-Time" **is relative**

When we have to deal with remote rather than direct communication, "freeze-frame" imaging systems such as ImageWise can lose most of their "real time" effectiveness as continuous-activity monitors due to slow transmission mediums. Using a 9600-bps modem, a compressed image will take about 40 seconds to be displayed. At 1200 bps it will take over 5 minutes!

Of course, using such narrow logic could lead one to dismiss freeze-frame video imaging and opt for hardwired direct video, whatever the distance. However, unless you own a cable television or telephone company you might have a lot of trouble stringing wires across town.

All humor aside, the only reason for using continuous monitoring systems at all is to capture and record asynchronous "events." In the case of a surveillance system, the "event" is when someone enters the area under surveillance. For the rest of the time, you might as well be taking nature photos because, according to the definition of an event, nothing else is important.

Most continuous surveillance video systems are, by necessity, real-time monitors as well. Because they have no way to ascertain that an event has occurred they simply record everything and ultimately capture the "event" by default. So what if there is 6 hours of video tape or 200 gigabytes of useless video data transmission around a 4-second "event."

If we know exactly when an event occurs and take a freeze frame picture exactly at that time, there is no difference between its record of the event and a real-time recorder or snap-shot camera at the same instant. The only difference is that a freeze-frame recorder needs some local intelligence to ascertain that an event is occurring so that it knows when to snap a picture. Sounds simple, right?

To put real-timing into my driveway monitor, I combined a video camera and an infrared motion detector. When someone (or something) enters the trigger zone of the motion detector it will also be within the field of the video camera. If motion is detected, the controller triggers the ImageWise to capture that video frame at that instant and transmit the picture via modem immediately.

The result is, in fact, real-time video, albeit delayed by 40 seconds. Using a 9600-bps modem, you will see what is going on 40 seconds after it has occurred. (Of course, you'll see parts of the picture sooner as it is painting on the screen.) Subsequent motion will trigger additional pictures until eventually the system senses nothing and goes back to timed update. With such a system you'll also gain new knowledge. You'll know that it was the UPS truck that drove over the hedge because you were watching, but you aren't quite sure who bagged the flower bed.

Of course knowing a little bit is sometimes worse than nothing at all. While a single video camera and motion detector might cover the average driveway, my driveway has multiple entrances and a variety of parking areas. When I first installed a single camera to cover the main entrance all it did was create frustration. I would see a car enter and park. If the person exited the vehicle they were soon out of view of the camera and I'd be thinking, "OK, what are they doing?"

Rather than laying booby traps for some poor guy delivering newspapers, I decided to expand the system to cover additional territory. Ultimately, I installed three cameras and four motion detectors which could cover all important areas and provide enough video resolution to specifically recognize individuals. (Since I have four telephone lines into my house and only one is being used with ImageWise, I suppose the next step is to use one of them as a live intercom to speak to these visitors. A third line already goes to the home control system so I could entertain less-welcome visitors with a few special effects).

Motion Triggered Video Mux

Enough of how I got into this mess! What this is all leading to is the design of my motion triggered video camera multiplexing (MTVCM) system. I am presenting it because it was fun to do, it solved a particular personal problem, and if I don't document it somehow, 1'11 never remember what I've got wired the next time I work on it.

The MTVCM is a 3-board microcomputer-based 4-channel video multiplexer with optoisolated trigger control inputs (see figure 1). Unlike the high-tech totally solid-state audio/video multiplexer (AVMUX) which I presented a couple years ago (BYTE Feb '86), the MTVCM is designed to be simple, lightning-proof, reliable, and above all flexible.

The MTVCM is designed for relatively harsh environments. To minimize wire lengths from cameras and sensors, the MTVCM is mounted in an outside garage where its anticipated operating temperature range is -20°C to +85"C. The MTVCM operates as a standalone unit running a preprogrammed control program or can be remotely commanded to operate in a specific manner. It is connected to the Imagwise and additional electronics in the house via a twisted-pair RS-232 line, one TTL "camera ready" line, and a video output cable.

At the heart of the MTVCM is an industrial temperature version of the Micromint BCC52 8052-based controller which has an onboard full floating-point 8K BASIC, EPROM programmer, 48K bytes of memory, 24 bits of parallel I/O and 2 serial ports (for more information on the BCC52 contact Micromint directly, see the Articles section of the Circuit Cellar BBS, see my article "Build the BASIC-52 Computer," BYTE, Aug '85, or send $2 to CIRCUIT CELLAR INK for a reprint of the original BCC52 article). Because the BCC52 is well documented, I will not discuss it here.

The MTVCM is nothing more than a specific application of a BCC52 process controller with a little custom I/O. In the MTVCM the custom I/O consists of a 4-channel relay multiplexer board and a 4-channel optoisolated input board (Micromint now manufactures a BCC40R 8-channel relay output board and a BCC40D direct decoding 8-channel optoisolated input/output board. Their design is different and should not be confused with my MTVCM custom I/O boards). Each of my

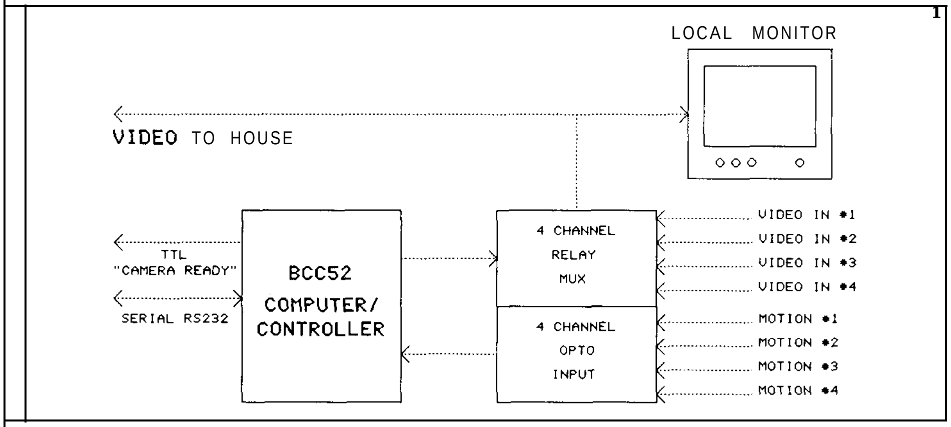


Figure 1

custom circuits is mounted on a BCC55 decoded and buffered BCC-bus prototyping board.

Figure 2 details the basic circuitry of the BCC55 BCC-bus prototyping board. The 44-pin BCC-bus is a relatively straightforward connection system utilizing a low-order multiplexed address/data configuration directly compatible with many standard microprocessors such as the Z8, 8085, and the 8052. On the protoboard all the pertinent busses are fully latched and buffered. The full 16-bit address is presented on J19 and J20 while the 8-bit buffered data bus is available at J21. J22 presents eight decoded I/O strobes within an address range selected via JP2.

The Multiplexer Board

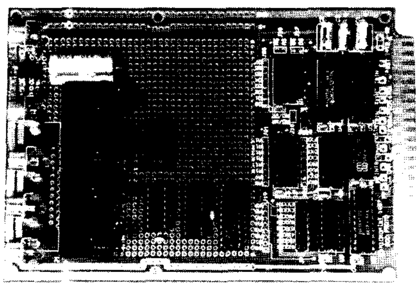


Figure 3 is the schematic of the relay multiplexer added to the prototyping board. The relay circuit is specifically addressed at C900H and any data sent to that location via an XBY command [typically XBY(0C900H)=X] will be latched into the 74LS273. Since it can be destructive to attach two video outputs together, the four relays are not directly controlled by the latch outputs. Instead, bits DO and Dl are used to address a 74LS139 one-of-four decoder chip. The decoder is enabled by a high-level output on bit D3. Therefore, a 1000 (binary) code selects relay 4 while a 1011 code selects relay 1. An output of 0000 shuts off the relay mux (eliminating the decoder and going directly to the relay drivers allows parallel control of the four relays).

All the normally-open relay contacts are connected together as a common output. Since only a single relay is ever on at one time, that video signal will be routed to the output. If the computer fails or there is a power interrupt, the default output state of a 74LS273 is normally high. Therefore, the highest priority camera should be attached to that input. If the system gets deep-sixed, the output will default to that camera and will still be of some use (I could also have used one of the normally-closed contacts instead but chose not to).

Fools and Mother Nature

I'm sure you're curious so I will anticipate your question and answer it at the same time. With all the high-tech stuff that I continually present, how come I used mechanical relays?

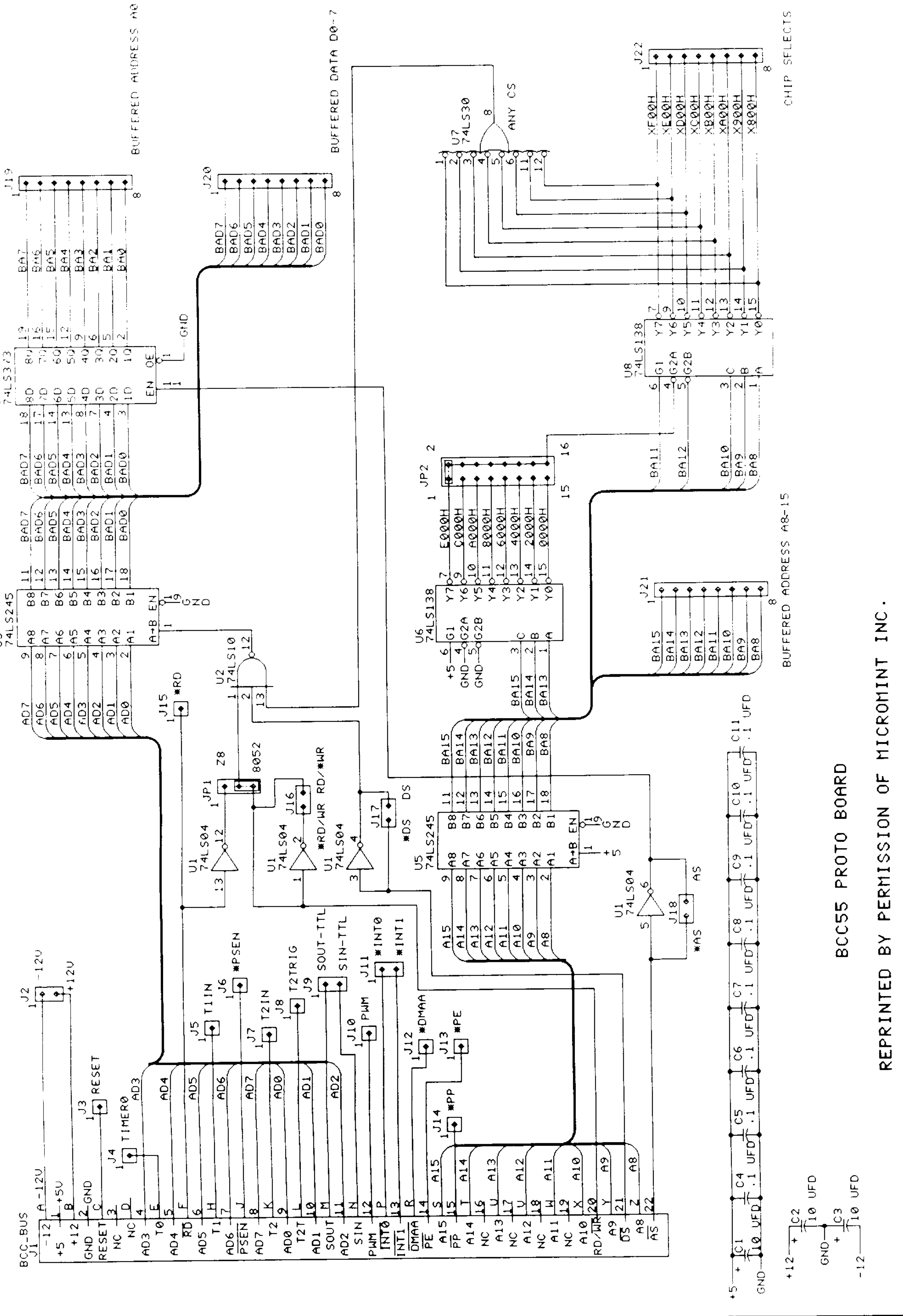
The answer is lightning! Anyone familiar with my writings will remember that I live in a hazardous environment when it comes to Mother Nature. Every year I get blasted and it's always the high-tech stuff that gets blitzed. Because the MTVCM has to work continuously as well as be reliable I had to take measures to protect it from externally-induced calamities. This meant that all the inputs and outputs had to be isolated.

In the case of the video mux, the only low-cost totally isolated switches are mechanical relays. CMOS multiplexer chips like the ones I've used in other projects are not isolated and would be too susceptible. (Just think of the MTVCM as a computer with three 150-foot lightning collectors running to the cameras.) Relays still serve a useful purpose whatever the level of integrated circuit technology. They also work.

Because the infrared motion sensors are connected to the AC power line and their outputs are common with it, these too had to be isolated to protect the MTVCM. Figure 4 details the circuit of the 4-channel optoisolator input board which connects to the motion detectors.

The Optoisolator Board

The opto board is addressed at CAOOH. Reading location CAOOH [typically X=XBY(OCAOOH)] will read the 8 inputs of the 74LS244. Bits O-3 are connected to the four



BCC55 PROTO BOARD

REPRINTED BY PERMISSION OF MICROMINT INC.

Figure 2

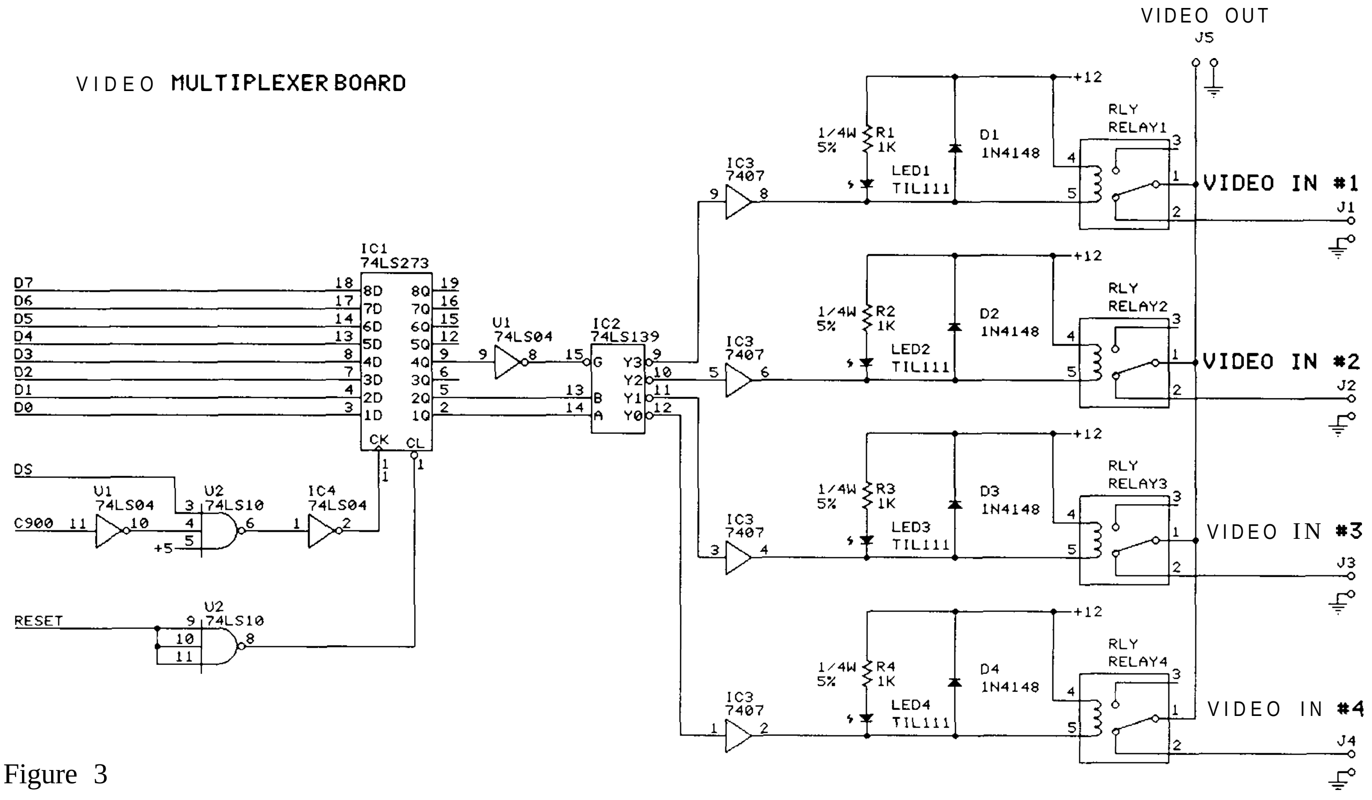


Figure 3

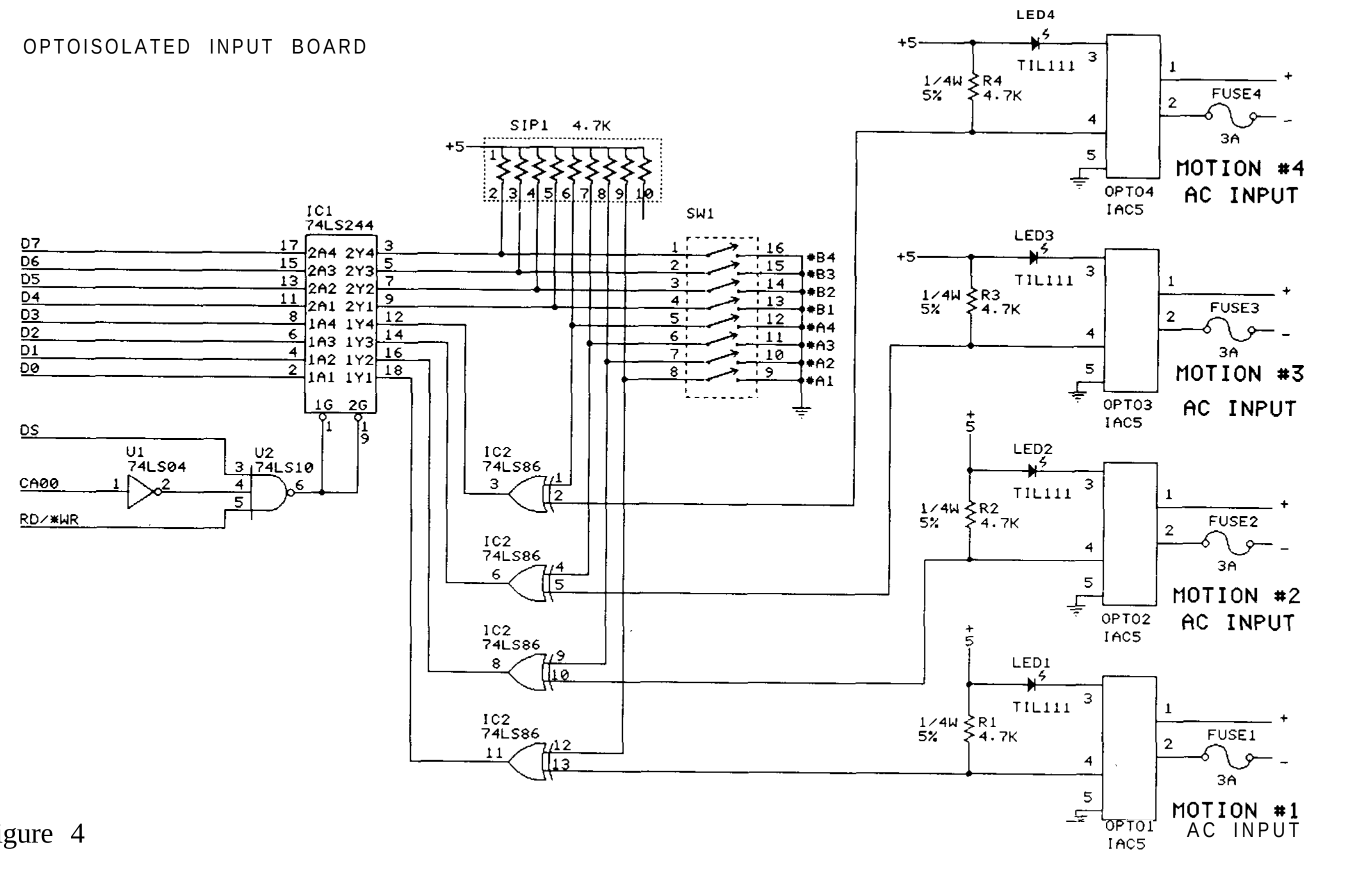
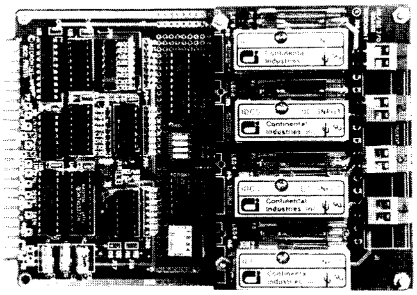


Figure 4

optoisolators and bits 4-7 are connected to a 4-pole dip switch which is used for configuration and setup. Between the optoisolators and the LS244 are four 74LS86 exclusive-OR gates. They function as selectable inverters. Depending upon the inputs to the optoisolators (normally high or low) and the settings of DIP SW2 you can select what level outputs you want for your program (guys like me who never got the hang of using PNP transistors have to design hardware so that whatever programming we are forced to do can at least be done in positive logic).



The optoisolators are common units sold by OPT022, Gordos, and other manufacturers. They are generically designated as either IAC5 or IDC5 modules depending upon whether the input voltage is 115 VAC or 5-48 VDC. Since the motion detectors I used were designed to control AC flood lights, I used the IAC5 units connected across the lights.

Simple Software

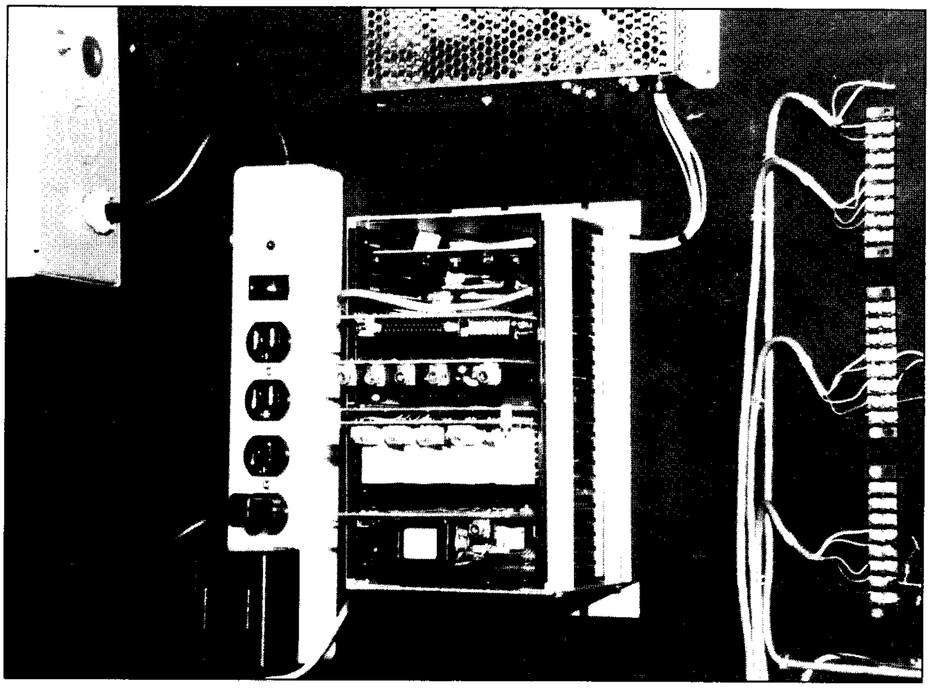
Now that we have the hardware I suppose we have to have some software. For all practical purposes, however, virtually none is required. Since the MTVCM is designed with hardcoded parallel port addressing, you only need about a three-line program to read the inputs, make a decision and select a video mux channel; you know, something like READ CAOOH, juggle it, and OUT C900H. I love simple software.

Of course, I got a little more carried away when I actually wrote my camera control program. I use a lot of REM statements to figure out what I did. Since it would take up too much room here, I've posted the MTVCM mux control software on the Circuit Cellar BBS (203-871-1988) where you can download it if you want to learn more. Basically, it just sits there looking at camera #l. If it receives a motion input from one of the sensors, it switches to the appropriate camera and generates a "camera ready" output (TTL output which is optoisolated at the other end) to the ImageWise in the house. It stays on that camera if it senses additional motion or switches to other cameras if it senses motion in their surveillance area. Eventually, it times out and goes back to camera # 1.

Basically, that's all there is to the MTVCM. If you are an engineer you can think of it as a lightning-proof electrically-isolated process-control system. If not, just put it in your entertainment room and use it as a real neat camera controller.

Now I've opened a real bag of worms. Remotely controlling the ImageWise digitizer/transmitter from my office through the house to the MTVCM is turning into a bigger task than I originally conceived. Getting the proper picture and tracking someone in the driveway is only part of the task. I can already envision a rack of computer equipment in the house which has to synchronize this data traffic. My biggest worry is not how much coordination or equipment it will involve, but how I can design it so that I can do it all with a three-line BASIC program! Be assured that I'll tell you how as the saga unfolds. ■

Call Circuit Cellar Inc. at (203)875-2751 for information and pricing on BCC52 and peripheral boards.



Complete MTVCM system as installed, including BCC52 and Optoisolator boards.

Ctrl INK SPOT

RISC vs Reality

An Exercise in Acronyms

by Tom Cantrell

Arrgghh--the RISC hype (and puns) are really getting out of hand. Let's face facts:

a) RISC has little to do with CPU performance. The bottom line: instruction set complexity is just not that big a deal.

b) CPU performance has little to do with overall system performance. Ask an HP 'Spectrum' designer whether RISC makes I/O faster.

c) System performance is only one factor in achieving success in the market. Ask IBM, DEC, or Apple.

Frankly, I'm shocked at the one-sided claims of some RISC (Reduced Instruction Set Computer) proponents and the apples-and-oranges RISC/CISC (Complex Instruction Set Computer) comparisons. The worst are made by those who--surprise!--sell RISC computers. Typically, it's a RISC with multi-register sets (MRS), giant on-chip cache and optimal, optimizing compiler vs. an old off-the-shelf CISC (VAX, 68000) with none of these. Proponents are glad to expand the definition of 'RISC' to accommodate every new chip -the more, the merrier! It is amazing how 'CISC'y some 'retro-marketed' RISCs are--CLIPPER, AMD 29000, even the Novix FORTH chip! 'RISC's as useful as 'MIPS (Meaningless Information Provided for Sales) - about the same as the 'New-Improved' label on soap.

Here is the true meaning of RISC in easy-to-remember acronyms...

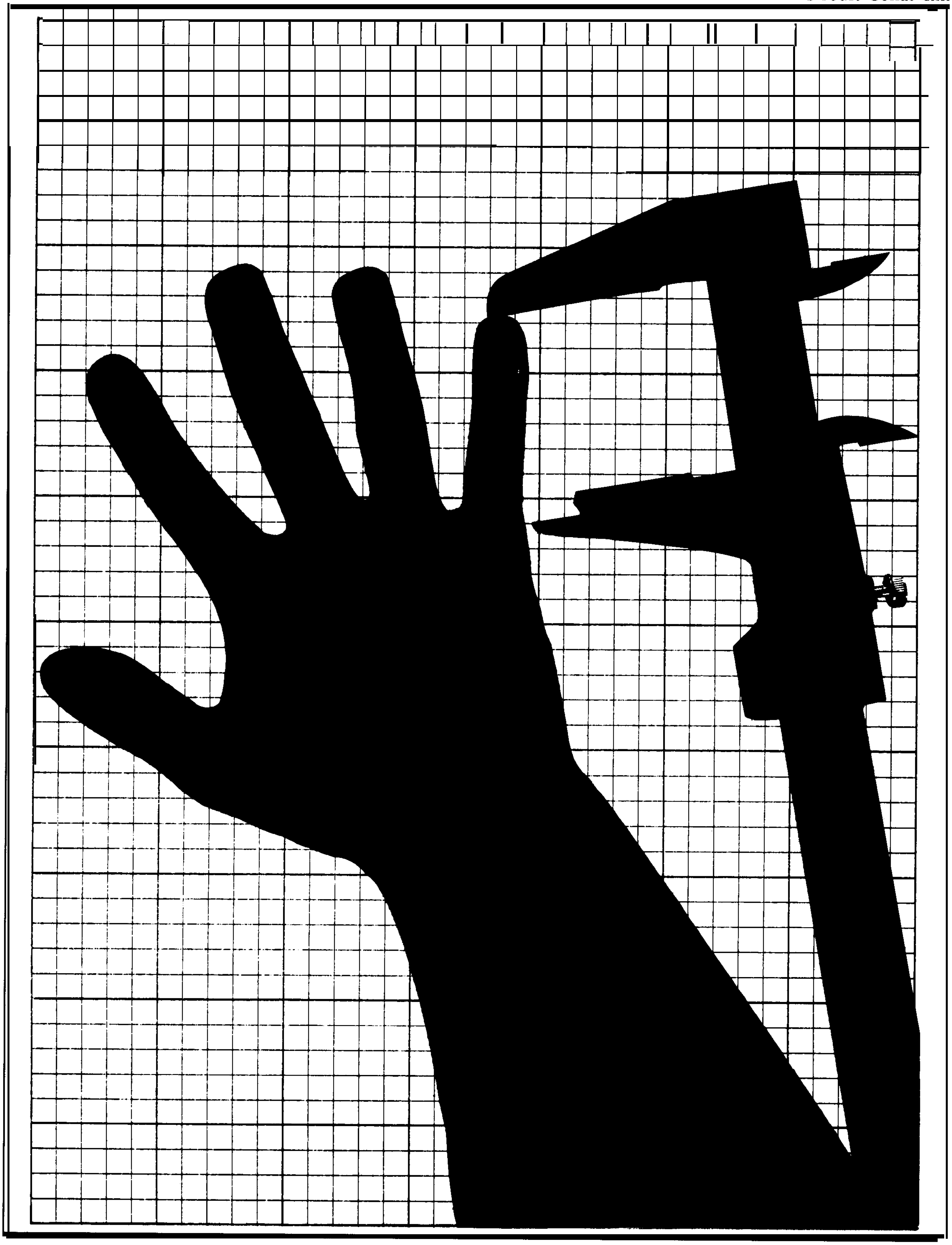
-RISC Is for Student Computers. Between football games and demonstrations, a few grad students can't come close to the hundreds of person-years needed to design a CISC like the '386, 68020 or VAX, and even if they could, who would make the chips? RISC allows students, and tiny companies, to build their own small, simple CPU chips--a worthy goal. It's only true for the original 20-30K transistor RISCs, not 'CRISCs' ('Complex' RISCs' like CLIPPER 3-chip set which is 850K+ transistors).

- Relegate the Impossible Stuff to the Compiler writer. Cutting corners on the hardware means the software has to be much smarter in areas like register allocation, instruction reorganization, pipeline scheduling, etc. Yes, smarter software is good--for both RISC and CISC.

- RISC Is an excuSe to Change. Though usually considered a no-no, there are a number of reasons a company may wish to abandon an existing architecture and switch to a new one, including NIH (Not Invented Here), 'locking' customers into a proprietary architecture and differentiation from competitors. For instance, which sounds better... 'Moon Microsystems is switching to RISC because it goes a hundred MIPs and the RISC/CISC war is over and RISC is the winner', or... 'Moon Microsystems is switching to RISC so you'll think we're better than 32-bit PCs which offer the same performance at half the price, more selection and lower cost hardware, software add-ons, and 1000x our installed base.'

Don't get me wrong--I'm not a CISC evangelist and welcome the good work and discoveries of the RISC camp. In fact, I'm trying to protect the 'good' of RISC from falling with the 'bad and ugly' hype. Witness what happened to yesterday's over-promoted hot-button, Artificial Intelligence--anyone want to invest in an AI ('That which cannot be done on a computer') company? Maybe a backlash is already starting--check out the article in the September 1987 issue of COMPUTER--'And Now A Case For More Complex Instruction Sets.' Another is 'Performance Evaluation Of Multiple Register Sets,' Eickemeyer and Patel, 14th Annual International Symposium on Computer Architecture. RISC zealots--read and heed! ■

Biography -- As an independent designer in Silicon Valley, Tom Cantrell has worked on chip, board and software projects, and written numerous articles. For writing, **Tom uses a 512K MAC but his favorite toy is an SB180.**



High Security on a Budget

Build a Video Hand Scanner/Identifier

by Ed Nisley

Let's suppose you're in charge of security for a building, controlling access to rooms full of valuable equipment. You must devise a method that will allow only authorized people to enter, and, as is usually the case, you only have a limited budget. Human guards used to be excellent for this sort of thing, but with the pressure of social security, pension plans, medical care, vacation, sick leave, lunch breaks and so on, most economy-minded companies keep moving to electronics for relief.

While I suppose that someday this article could conceivably be titled, "Build Your Own RoboCop," today we have to be concerned with more basic issues dictated by the state of current technology.

All kidding aside, the most obvious security advantage for having a human guard is visual recognition. The guard is there to "see" and "recognize" people and take appropriate action. Electronically speaking, most of this can be done by present computer technology. A computer can digitize a person's face and compare this picture to a library of images via image processing algorithms. How fast and accurate it does this of course is purely an economic decision. A CRAY-XMP does the job very quickly and effectively.

Generally speaking, however, more of us own PC/ATs than CRAYs. This doesn't mean that we are back to contending with unions and coffee breaks. We can still perform useful image processing with an AT provided that we limit the amount of visual data that we process.

Full-face images, while certainly different enough between individuals, contain voluminous amounts of data that take a considerable amount of time to crunch. The trick is to find some visible feature that will both identify each authorized person and reject every unauthorized one. Whatever feature you pick should be difficult to counterfeit and easy to verify. Furthermore, the whole process should take only a few seconds so that the system isn't an irritant.

In true Circuit Cellar fashion, however, we are not here to philosophize about the elements of such a security system. We are here to build and test it. This article describes the HandScanner, a prototype device that identifies people using images of their hands.

Using an ImageWise video digitizer (described in a construction project by Steve Ciarcia in the May-June '87 BYTE) and an IBM AT, the HandScanner takes digital TV pictures of hand prints and analyzes and identifies the person in only a few seconds.

What does it do?

The basic idea behind the HandScanner was the (not very stunning) observation that people's hands look different: some are long, some short, others fat or skinny. The overall hand shape may not be as unique as a fingerprint, but it seemed reasonable that there would be enough difference to make the project feasible. Best of all, hands are easy to get into a TV picture.

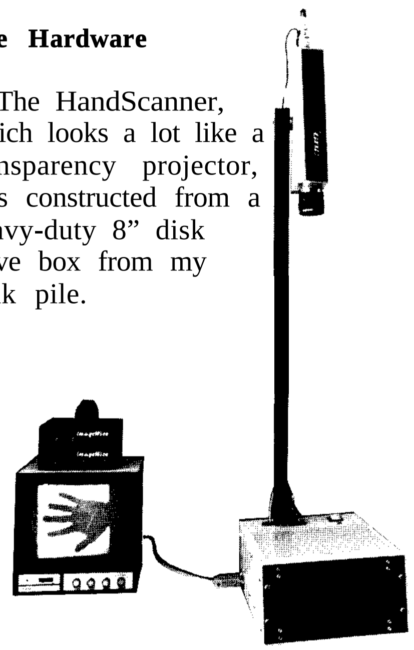
Unfortunately, there is a big step between a human's almost instantaneous recognition of the difference between two images and a computer program's ability to do the same thing. It's not at all clear just how people recognize things, so there's no hope of duplicating that feat in software. But there might be a way to extract a few significant numbers from each hand image that could serve as a signature.

The first chore was taking the picture. It turned out that the only way to get a reliable picture of a hand was to backlight it against a bright screen to form a silhouette. This eliminated any superfluous information provided by color or texture. Finger lengths and the overall width of the palm were distinctly apparent and relatively easy to measure. After a bit more experimenting, I had the basic elements of the finger measurement algorithm, and all the pieces were in place.

The HandScanner is composed of three parts: the scanner hardware, the firmware on the ImageWise board, and the software to handle the image analysis and recognition. I'll cover them in sequence so you can see how the project evolved, then discuss how well it works in actual practice.

The Hardware

The HandScanner, which looks a lot like a transparency projector, was constructed from a heavy-duty 8" disk drive box from my junk pile.



The power switch in the corner is the only external control. Near the switch is a two-color LED that glows green when the scanner is ready to take a picture and red when the AT is analyzing one. Power is supplied through a standard three-prong line cord. The female DB-25 connector is wired as a modem, so a straight cable can connect it to a PC serial port.

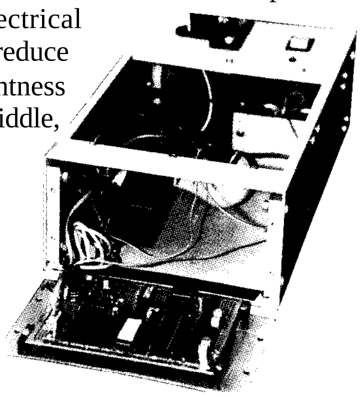
The camera on the overhead arm is wired to a power supply and ImageWise board inside the box. It uses a standard 16 mm f/ 1.4 lens, although a zoom lens would have simplified the setup quite a bit. Almost any camera and lens will work, but the image at about two feet must be big enough to encompass a hand.

The square scanning panel is made of white translucent plastic. A 1 00- Watt incandescent bulb behind it provides enough backlighting to silhouette a hand against normal office lighting. A bulb that size throws off quite a bit of heat, so it is only turned on when the ImageWise digitizer/transmitter board is digitizing a picture.

The bulb is controlled by a 5A solid-state relay driven directly from the RS-232 RTS signal. It is connected so that a high level (about +lO Volts) turns the light on. It is off for a low-level signal or an unplugged connector. The relay must be sized to handle the bulb's cold inrush current, which can be much higher than the steady-state rating.

The camera is suspended over the center of the scanning panel, and aligned so that the upper left corner of the image corresponds to the corner of the panel nearest the power switch and LED. The distance is set so that the panel nearly fills the image vertically. The high-contrast lighting ensures that only the hand over the panel is visible, with the rest of the image appearing completely black.

This photo shows a view of the Scanner with the scanning panel removed and the end plate detached. The bulb is covered with strips of black electrical tape to reduce the brightness in the middle, of the scanning panel.



Schematic 1 shows the circuitry inside the HandScanner box.

The scanning panel is suspended by thin mounting strips along two edges, with supports on the other two. A friction fit keeps it from falling out.

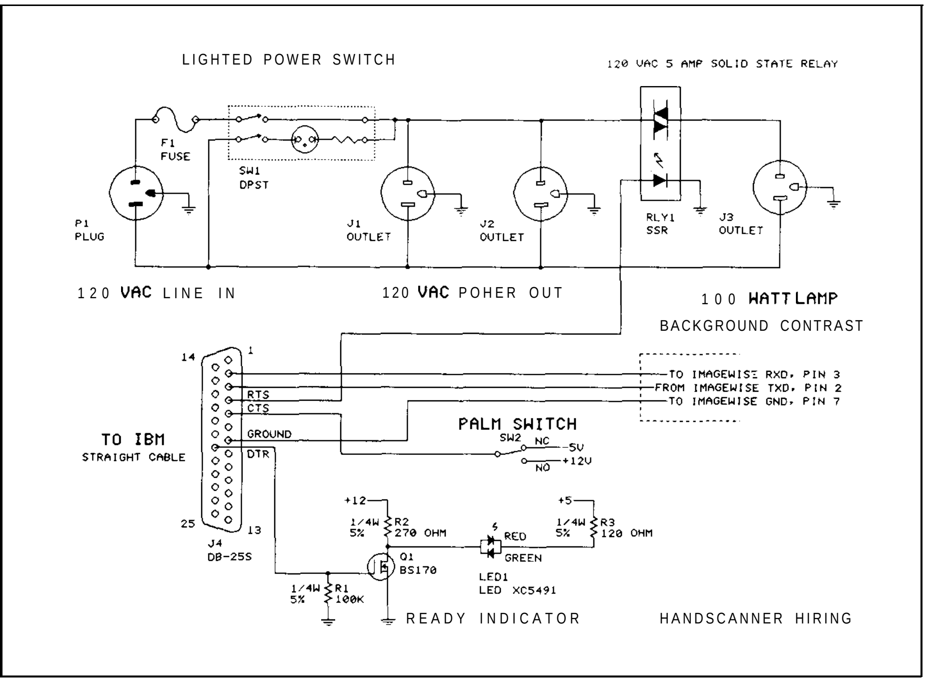
The arm extending over the panel opening holds a microswitch connected to the RS-232 CTS line. The normally-closed contact supplies 5 volts and pressing gently on the panel bends it just enough to trip the switch, pulling CTS to +12 volts. The AT monitors CTS to detect when it's time to take a picture. Pressing one's hand on the panel simultaneously flattens the hand and triggers the switch!

The two-color LED is red when the RS-232 DTR signal is low (about -lOV) and green when it's high (about +lOV). The LED required more current than the DTR line could supply, so I used a FET buffer and a few resistors to provide about 25 mA in each direction.

The overall function of the hardware is straightforward: the AT sets DTR high to make the LED glow green, then waits until CTS becomes high when someone presses the microswitch under the scanning panel. The AT sets RTS high to turn on the 100W bulb, delays a few hundred milliseconds to allow the bulb to warm up and the camera to stabilize, then sends an XON to the ImageWise to grab a new image. Because the ImageWise takes only l/60 second to store the image, RTS and DTR can be set low to turn off the light and set the LED back to red immediately after the first byte is received.

The Firmware

The starting point for the



Schematic 1

Handscanner images is a standard video camera picture processed by an ImageWise Digitizer/Transmitter unit. Because there is no need to display the images, a Receiver/Display unit is not used.

A standard ImageWise Digitizer converts a picture into an array of 244 lines with 256 pels in each line, each pel having one of 64 brightness levels. The Digitizer's run-length compression algorithm can give a 25 - 50 percent reduction in the amount of data, but a typical image will require about 20 seconds to receive and occupy 20K to 40K bytes of disk space.

That amount of time was acceptable while I was initially building the HandScanner, but some improvement was necessary before persuading people to "lend me a hand" for the test pictures. Fortunately, since I colaborated with Steve on the ImageWise article and wrote the original ImageWise firmware, I knew exactly what to change!

The process of image analysis works best when the desired part of the image is very different from the background. The HandScanner uses high-contrast backlighting to silhouette the hand, which gives considerable definition. There's no use carrying excess brightness information along that will be discarded later.

I added a small piece of code to the Digitizer firmware to force each pel's value to 15 if it was originally any non-zero value. The resulting pel data, with values of 0 and 15, can be compressed efficiently by the RLE algorithm, resulting in 244x256-pel images that occupy about 3 K bytes of disk space and take only a few seconds to transmit at 28.8 K bits/second.

Because position 7 on the Digitizer's DIP switch was unused by the standard EPROM, I appropriated it to control the new threshold function. The definition of switch 6 is changed (when switch 7 is on) to set the threshold to 1 or 4, the latter being useful to compensate for very bright room lighting that may produce highlights on the back of the hand.

Surprisingly enough, the code in the AT doesn't have to know if the special firmware is in use! All of the image processing routines expect to see zero-value pels inside the hand, surrounded by non-zero pels. Those border pels can have any value, so 15 serves as well as any other.

Photos 3 and 4 show the view from the video camera with thresholding OFF and ON, respectively. You can see why the run-length compression algorithm will be much more successful with the latter image!

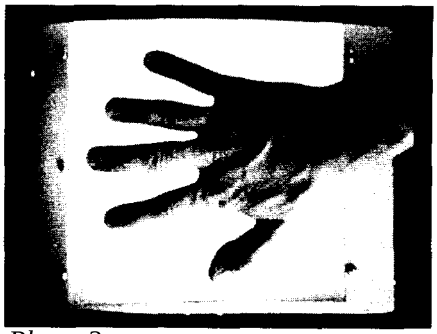


Photo 3

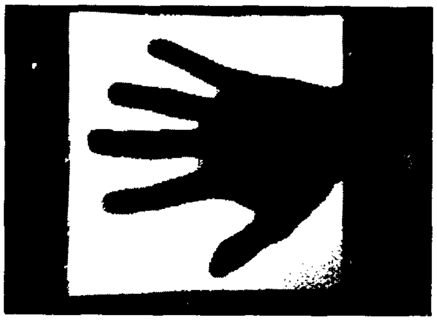


Photo 4

The software for this project, including the hex code for the modified HandScanner ImageWise EPROM and the IBM AT programs described can be downloaded free from the Circuit Cellar BBS (203-871-1988). Alternatively, a diskette containing all the pertinent files is available from CCI (4 Park St, Suite 12, Vernon, CT 06066; 203-875-2571) for $10 postpaid. Please add $9 for a set of Image-Wise Digitizer/Receiver manuals. This software is distributed solely for noncommercial personal use. Licensing is required for other uses.

Hand Images

The software in the AT must handle three main functions: grab a hand image, analyze it to get a few numbers, then compare those numbers against a database of “authorized” hands to decide whether the hand is attached to an authorized person. Obviously that’s a lot of code!

First of all, I realized that I’d have to work from pictures stored on disk rather than from live images simply because I couldn’t very well ask a dozen people to hang around waiting for me to get the code perfected. Another advantage of disk files was that I could use exactly the same images over and over again until the right answers came out.

So the starting point was grabbing pictures from the HandScanner. The GRAB program distributed with the ImageWise boards wouldn’t quite work because the HandScanner uses the RTS line to control the LED and the CTS line to tell when to take a picture. But it was pretty close, so I chainsawed

Listing 1 -- key parts of HANDGRAB.PAS

```
PROGRAM   Grab(input,output,picfile);

{$U- control-break checking during execution          }
{$C- control-break checking during I/O operations     }
{$R- array range checking                             }
{$I- turn off I/O checking (we do our own)            }

(*** some code omitted here ***)

{--------------------------------------------------}
{ Get a picture from the transmitter               }
{ The bit rate depends on which PC you're using... }
{ An 8 MHz AT-can handle 28.8 K bits/sec           }
{ Assumes hand scanning box with                   }
{  RTS = lamp control                              }
{ CTS = palm switch                                }

PROCEDURE GetPicHand(pic : picptr; resol : BYTE);

VAR
 picbyte   : BYTE;              { byte from transmitter }
 bptr      : byteptr;           { fake pointer to pic   }
BEGIN

 Port[comMCR] := $03;             { turn on the light      }
 bptr := Ptr(Seg(pic^),Ofs(pic^)- 1);  { preset for loop }
 SendByte(reso1);                 { specify resolution     }
 SendByte(XON);                   { prompt transmitter     }

 WHILE ((Port[comLSR] AND DataReady) = 0) AND
       NOT KeyPressed DO;              { stall for data }
 Port[comMCR] := $00;             { turn off the light    )
 REPEAT                           { for each line          }
  bptr := Ptr(Seg(bptr^),Ofs(bptr^)+l);    {  tick  ptr  }
  WHILE ((Port[comLSR] AND DataReady) = 0) AND
        NOT KeyPressed DO;             { stall for data }
  bptr^ := Port[comdata];         { snag the byte          }
 UNTIL (bptr^ = fldend) OR KeyPressed;

END;

(*** some code omitted here ***)
```

(Continued page 17)

```
{-------------------------------------------}
{ The Main Routine!}
BEGIN

 Port[comMCR] := $01; { turn on READY light }

 IF manual
  THEN BEGIN
 Write('Press scanner plate to start scan... ');
   REPEAT
UNTIL KeyPressed OR (Port[comMSR] AND$lO) <> 0);

  END;

 Port[comMCR] := $03;     { turn on scan light    }
 Delay( 500);
 picl := NIL;             { ensure new alloc      }
 PicSe tup(pic 1);        ( set. up picture array )
 Writeln('Loading');
GetPicHand(pic1 ,fullres);( get the picture       }

 filespec := GetFSpec(ParamStr( 1));

 pic2 := NIL;             { ensure new alloc      }
 PicSetup(pic2);          ( get second array      }
 Writeln('Expanding');
 Expand(pic 1 ,pic2);     { expand image          }
SavePicture(filespec,pic2);{save it away          }
END.
```

the Pascal source code to come up with HANDGRAB.PAS.

Listing 1 shows a key section of HANDGRAB, which gets a picture from the board and stores it as a disk file. The disk files are stored in compressed form to conserve disk space (even 20 MB hard disks fill up eventually!) and the modified ImageWise EPROM ensures that only about 3K of space is used for each image.

To find enough people to make a decent database, I coerced about a dozen people into giving me five hand pictures each.

Hand Measurement

The next task was a little trickier: a program to read a hand image file and measure finger lengths and palm widths without human intervention. After all, if the security system required someone to point out the interesting parts of the picture, you might as well hire a guard!

Although the natural way to think of finger length is from the tip of the finger to the middle of the knuckle on the palm, it's easier to measure from the end of the nail to the midpoint between the webs on either side of the finger. Obviously, a few fake fingernails can confuse the issue beyond recovery.

The measurement software starts by assuming that the hand in question is aligned with palm to the panel, fingers pointing to the left, middle finger roughly along the middle of the screen, and thumb at the bottom of the screen. If you think about it for a moment, you'll realize that the algorithms work only with right hands; I decided that a left-handed option could be added later on.

The code scans for the tip of the middle finger by looking for the first black spot on the left edge of the image. A reserved border ensures that it scans only the illuminated background rather than the darker surrounding area at the very edges of the image.

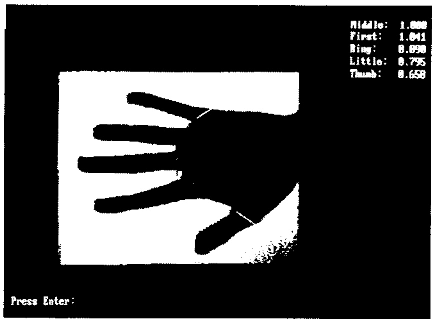
Once the middle fingertip is found, the code scans upward for the ring and little fingers, then downward for the index finger and thumb. It was possible to guess where the fingers might be because the images are restricted to right hands; the left-handed option would be exactly reversed.

With the fingertips located, the program next traces along the finger boundaries to locate the webs between each one. The webs are assumed to be the first point where the edge curves back to the left after the vertical section.

Knuckles between webs are assumed to be at the midpoint of the line connecting the webs. The thumb and little finger knuckles are located by extending lines at 45 degrees from the last web on each side, with the midpoint of that line marking the knuckle. The actual knuckle locations can't be found from a silhouette, but these are close enough to serve for our purposes because they're measured the same way for each hand.

The complete hand analysis program is far too long to reprint here, but the small section in Listing 2 should give you an idea of how it works. The coordinates for each measured point are organized into an array of records with the array elements indexed by finger name. Thus, the coordinates of the tip of the little finger are held in the variables HAND[LITTLE].TIP.PEL and HAND[LITTLE].TIP.LINE.

Photo 5 shows a hand image with the significant points marked. I used different colors to indicate the fingertip points, the tracing leading to the webs, and the knuckles; these colors show up much better on an EGA display than on the printed page. The routine highlights each point as it is found to simplify debugging. One of the nice features of working with graphic programs is that it's painfully easy to see gross mistakes, particularly if they're marked in glaring red.


```
Middle:  1.000
First:   1.041
Ring:    0.898
Little:  0.795
Thumb:   0.658

Press Enter:
```

Photo 5

Incidentally, the EGA was used just to keep an eye on the measurement program. It's not a critical part of the process, and, in fact, drawing the pictures on the screen takes far more time than the analysis!

Finger lengths are simply the distance between the tip and the knuckle at the edge of the palm. The palm width is the distance

Listing 2 -- Locating middle fingertip

```
{*** extracted from a PROCEDURE ***)
{--------------------------------------------------}
{ Locate middle finger                             }

 pndx := border;
 REPEAT                        ( for each pel column   }
  lndx := border;
  REPEAT                       { for each line         }
   IF (pic^.fmt.lines[lndx].pels[pndx]  = 0) AND
      (pic^.fmt.lines[lndx].pels[pndx+l] = 0)
    THEN BEGIN                   { find midpoint        }
     top := lndx;           { remember top line     }
     REPEAT
      pic^.fmt.lines[lndx].pels[pndx] := maxbit;
      IF showit
       THEN BEGIN
        EGASetPelAddr(lndx,pndx);
        EGASendByte(tracecolor);
       END;
      lndx := Succ(lndx);
     UNTIL (pic^.fmt.lines[lndx].pels[pndx] <> 0);
     avg := (top + lndx - 1) DIV 2;
     hand[middle].tip.line  := avg;     { remember this! }
     hand[middle].tip.pel  := pndx;
     lndx := maxline - border; { exit from loop        }
    END
  IF KeyPressed
    THEN GOT0 bailout;
   lndx := Succ(lndx);
  UNTIL (lndx > (maxline -  border));
  pndx := Succ(pndx);
 UNTIL (hand[middle].tip.line <> 0) OR
       (pndx >  (maxpel-border));

 IF hand[middle].tip.line <> 0
  THEN WITH hand[middle].tip DO BEGIN
(* Writeln(' found at ',pel,',',line); *)
   IF showit
    THEN BEGIN
     EGASetPelAddr(line,pel);
     EGASendByte(markcolor);
    END;
  END
  ELSE BEGIN
   Writeln('Could not find tip of middle finger!');
   Halt;
  END;
```

between the knuckles at the bases of the thumb and little finger. The finger lengths are computed in units of pels.

Because the actual pel size depends on the hand-to-camera distance and the lens focal length, it's reasonable to normalize the finger lengths to a standard. The middle finger can be measured most accurately, so I divided all the other finger lengths by the length of the middle finger. The resulting ratios are independent of the actual picture size.

The program appends these results to the end of an ASCII file that can be read and written by a standard text editor. By concentrating all the data in one readable file, I could manually delete incorrect records without having to regenerate the whole file from scratch. Listing 3 shows a part of that file.

Listing 3 -- part of the hand data file

The numbers represent the length of the middle finger in pel units and the thumb, first, ring, little finger, and palm sizes expressed as a ratio to the middle finger.

```
"       JEFF1",67,0.761,0.985,0.925,0.806,1.684
"       TRISH5",64,0.703,1.000,0.922,0.781,1.490
"       KEN1",64,0.719,1.016,0.969,0.828,1.789
"       MERRILL5",69,0.768,1.043,0.942,0.855,1.712
"       DAN1",75,0.627,0.947,0.933,0.827,1.564
"       STEVE4",65,0.723,0.923,0.985,0.846,1.672
"       CAROLYN2",69,0.710,0.928,0.928,0.768,1.512
"       JANE4",60,0.800,1.050,0.917,0.800,1.577
"       MARY5",69,0.652,0.971,0.928,0.797,1.543
"       TRACEY4",67,0.776,1.030,0.940,0.896,1.524
```

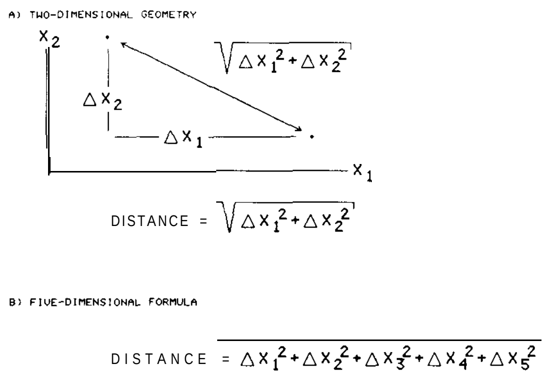

A) TWO-DIMENSIONAL GEOMETRY
x_2
$\sqrt{\Delta x_1^2+\Delta x_2^2}$
Δx_2
Δx_1
x_1
DISTANCE = $\sqrt{\Delta x_1^2+\Delta x_2^2}$
B) FIVE-DIMENSIONAL FORMULA
DISTANCE = $\sqrt{\Delta x_1^2+\Delta x_2^2+\Delta x_3^2+\Delta x_4^2+\Delta x_5^2}$

Figure 1

Hand Recognition

The remainder of the problem was more of a database exercise than anything else, so I switched languages from Pascal to dBase to take advantage of the latter's data manipulation functions. The file of hand data was easily converted into a dBase III database file and the dBase routines look suspiciously like pidgin Pascal.

The database file contains one record for each hand image, with each record including the image file-name and the key hand measurements described above. I had about 50 records to work with, which isn't enough for a true statistical analysis, but was about as many as I could get without exhausting everyone's patience.

I selected one data record at random for each person to serve as a test image; the four remaining records were used to create the database against which those test images are matched. While it's tempting to use all of the images to create the database, that tends to give unrealistically high recognition rates because there are no "unknown" images that haven't contributed to forming the matching criteria.

The key assumption in this project is that images of one person's hand will resemble each other more than they do anyone else's hand. If that is true, then all of the records for that person should be numerically close to each other and distant from other records.

Each database record can be regarded as the coordinate of a point in five-dimensional space. A simple extension of the familiar Pythagorean formula used to measure distances in two- or three-dimensional space will serve to find the distance between a pair of records. Figure 1 shows the two-dimensional formula and geometric interpretation, as well as the five-dimensional formula used to measure the distance between two database records.

You might feel a little queasy about five-dimensional points and distances. This is actually a standard mathematical process: the dimensions don't really correspond to the ordinary ones of length, width, and depth... we're not venturing into Special Relativity for this project! The dimension of a point is just the number of values needed to specify it: in our case there are five values, so we need five dimensions.

The database records (but not the test record!) for each person were combined by simple averaging to get a single record representing that person's hand images. I assumed that each measurement was contaminated by random noise, so that the averaging process would remove some of the noise and provide a better estimate of the true location of the ideal hand image in that five-dimensional space.

Listing 4 shows the distances between Steve's average and those of everyone else. The smallest numbers represent people with relatively large hands like Steve's; larger values are less similar. Remember that the single distance number summarizes all the information in the difference between a pair of four-finger and palm sizes.

The process of identifying a new hand image requires computing the distance between its data point and every record in the database. The smallest distance indicates the database record that is most similar to the new point, so the name associated with that record has the best chance of being the right one.

Listing 5 shows the distances between Steve's "test" hand image (the one that was not included in the database records) and all of the points in the database. The smallest distance was to Steve's averaged point, so his new image was correctly identified.

Listing 4 -- distances from Steve's averaged hand point

[CAROLYN]	0.157
[DAN]	0.136
[JANE]	0.162
[JEANNETTE]	0.177
[JEFF]	0.063
[JOHN]	0.273
[KEN]	0.095
[MARY]	0.139
[MERRILL]	0.095
[STEVE]	0.000
[TRACEY]	0.175
[TRISH]	0.223
[VAL]	0.186

Listing 5 -- distances from Steve's test hand point

[CAROLYN]	0.187
[DAN]	0.165
[JANE]	0.166
[JEANNETTE]	0.195
[JEFF]	0.07 1
[JOHN]	0.286
[KEN]	0.074
[MARY]	0.168
[MERRILL]	0.100
[STEVE]	0.044
[TRACEY]	0 167
[TRISH]	0.245
[VAL]	0.211

Positive ID

Unfortunately, it turns out that there is just not enough difference between hands to reliably identify everyone. Steve can't pass for Trish (indeed!), but it's difficult to choose between him, Jeff, Ken, or Merrill because all their hands are similar. Noise and measurement errors can cause one hand to look much like another.

But all is not lost!

Instead of trying to identify a person based on a hand image alone, we can verify that they are who they claim to be. For example, if Steve provides both an image and his name, the program can measure the distance between the image point and his database record. If there's a close match between the two, he's presumed to be Steve.

In Listing 5 you can see that the next closest match is 0.071 (for JEFF), with the others above 0.100. It turns out that matching hands tend to be within 0.080 of the matching record, so requiring an unknown hand to be within 0.090 of the record will give some margin for error.

Security systems can fail in one of two ways: false positives and false negatives. The false positive failures admit strangers to the building, the false negatives reject authorized people. In general, you can reduce one type of failure only at the cost of increasing the other.

Because the unknown hand is tested against only one record in the database, the minimum distance allowed for a match determines the error rate. Too large a distance will permit mismatches, but a very small distance will reject an unreasonable number of authorized people. The correct value must be determined by experimentation and will surely depend on the exact collection of hands in the database.

Of course, someone typing in a name that's not in the database can be rejected immediately, simply because all authorized people are in the database. An intruder must know which person on the authorized list has a similar hand, which is rather hard to figure out by just looking -- we humans are easily distracted by details that the HandScanner algorithms simply ignore.

Conclusions

The HandScanner is not as good as a fingerprint matching system, that much is certain! However, it may well provide enough reliability for low to moderate security areas, particularly if you have a small number of people who are authorized for access.

I suspect that some changes would greatly improve the performance and reliability. In particular, a better means of extracting the hand measurements would reduce the errors in each length and reduce the differences between measurements of the same hand.

Optically, the hand images use only about half of the picture area; a cylindrical lens would extend the image and increase the resolution. A zoom lens would allow some modification to the working distance, and a power zoom lens would allow the size of the image to be tailored to each hand.

As I suggested at the start, you might be able to adapt the HandScanner to industrial inspection applications rather than security systems. For example, if you're involved with punching sheet metal, the HandScanner can tell you whether you've got a correctly sized hole in the right place in a few seconds. Some changes to the mechanics would be needed to fit it into your production line, but the principle remains the same. ■

Acknowledgement

Thanks to Jeff Bachiochi, who was the first to say "Hey, I bet everybody's fingers are different!" It was all downhill from there...

CONNECTIME

THE CIRCUIT CELLAR BBS
300/1200/2400 bps
24 hours/7 days a week
(203) 871-1988
Vernon, Connecticut

Sysop: Ken Davidson

And so the welcome file begins as your modem connects with the Circuit Cellar BBS. If you keep reading, you'll find that the CCBBS is set up to provide support to Ciarcia's Circuit Cellar and Circuit Cellar Ink readers. It is here to answer questions about articles, to distribute files related to the articles, and for people to discuss just about anything concerning computers and electronics.

We have a devoted group of regular callers who provide a rich and diverse background in the fields of electronics and computer science. More than once I've seen an off-the-wall question posted to "ALL USERS" that I thought nobody could answer, just to find a knowledgeable response to it the next day.

Connectime's purpose is to provide you with a sampling of what's being discussed on the CCBBS. The CCBBS has been operational for over two ***years and we're lucky to have some very gifted and talented callers who have contributed to its success. Unfortunately, it is very difficult to synopsize two years of BBS traffic in a few pages of INK (especially the massive initial message traffic on the SB180). The following is just a sampling of some of the BBS conversations:***

Msg *#3006*
by Andrew Coile
To: Steve Ciarcia
Re: AVMUX Circuitry

Steve,

I read your construction articles with much interest and, for the first time, I am attempting to build an audio routing switcher (8x8x2) based on the AVMUX (Feb '86). After plotting out the schematic, some questions occurred to me.

In the AVMUX, you provide a 10K resistive load to the input and then AC couple it through a 0.1 uF capacitor to the input of the 74HC22106 cross-point switch. The output of the cross-point has a 1 Meg pull-down to ground at the input of an inverting (practically) unity-gain buffer.

My questions is, what biases the inputs? If there are internal biasing resistors on the 74HC22106 (I don't have a data sheet on them, they're too new), then the input will get centered around Vcc/2, or 2.5 volts. Fine.

When there are no cross-points active for an output, is it safe to assume it is high impedance? (I assume yes, otherwise why would you have the pull-down resistors to prevent the inputs from floating all over the place.) If this is true, wouldn't the pull-down resistor pull the input toward ground, and then when a cross-point is activated, suddenly the 74HC22106 output goes up to Vcc/2, giving you one massive glitch? (I wouldn't want to be standing in front of my speakers when a glitch like that goes through.) Is this the case? If so, how can the 74HC22106 be biased to operate noiselessly? Perhaps by running it on +/- 5V and then playing games with the addressing and control lines?

I plan to run the cross-point with a Z8 using MM5486 display drivers to run a 64-LED cross-point display. The routing switcher will also have the capability to accept remote keyboard inputs. (I'll send you a copy of the schematics and a photo of the board when I get it done.)

I am a Systems Engineer by profession, and specialize in dedicated microprocessor-driven industrial controllers. Thus I am familiar with this type of circuit.

Thank you very much for answering this question.

--Andrew Coile

P.S. Doesn't the Z80 have an I/O space of only 256 ports? I thought the contents of the A register are placed on the low-order address lines, but that is not really true 16-bit addressing. Plus, that "feature" of the Z80 is not documented (I think). Good article on interfacing, though. Keep up the good work! . . ..ACMC

Msg #3022
by Steve Ciarcia
To: Andrew Coile
Re: AVMUX Circuitry
[Reply to msg #3006]

Andy,

You pretty much answered all your questions. An unselected channel is a high impedance. I did find, however, that this was not a good idea and led to cross talk. All output devices should have something selected as its source, even if only a grounded input channel. The input impedances of the monitors and amplifiers end up with garbage if left open-circuited.

While the Z8 is a good device, you may want to consider the BCC52 after all. There is a new 4x20 and 8x40 LCD display peripheral (BCC25) for the 52. It also has provision for a parallel keyboard attachment. There will be a spec sheet in the articles section one of these days. Finally, to my knowledge the Z80 addresses 64K bytes of memory AND 64K bytes of I/O. There are four control lines: RD\ (read), WR\ (write), IOE\ (I/O enable), and ME\ (memory enable).

--Steve

P.S. If there are any loud "pops" when switching channels, I haven't experienced it. I'm not using much audio, however. My switcher is primarily video.

Carrier current systems is always a popular topic when dealing with retrofitting a house for automated control. The following series of messages addresses the issue:

Msg #3054
by Richard Hinton
To: ALL USERS
Re: Carrier current systems

I am interested in a carrier current intercom and also an appliance control device ala carrier current. My name is Richard Hinton from Homewood, Illinois. If anyone has any information about same please send along or I will be calling back this BBS in a few days.

--Thanks.

Msg #3101
by Marc Bacon
To: Richard Hinton
Re: Carrier current systems
[Reply to msg #3054]

Richard,

I am also interested in carrier current systems. Radio Shack makes an FM wireless intercom that uses carrier current and BSR makes the X- 10 home control system. These two are not compatible and the intercom can cause some bizarre things to happen. I asked Steve about this a few months ago and he stated that it is hopeless to run the two. Steve's Home Control System allows you to control appliances in the BSR X-10 system using the HCS computer. I hope this might be of some help. Additional information can be found in earlier Circuit Cellar projects including April and May, 1978, January-March 1979, Jan, 1980 (BSR X-IO), Aug 1983, April-June 1985 and others I might have overlooked. These are in BYTE magazine and copies can be obtained from libraries and computer information services such as the Source et al. They will also be available from long term hackers (don't talk to crackers...they are the bad guys) who, like myself have collected BYTEs from the early days. Good luck. I am currently developing a project along these lines and I will try to keep you informed on this BBS as I promised Steve.

Later, --Marc

From carrier current in general, we move on to more specific discussion about X-10 and HCS. We had a run on HCS questions during one week with a good deal of information being exchanged.

Leo Taylor, who helped in the development of the HCS, had just answered a question for Jack Olivieri. The following is what transpired:

Msg #3275
by Jack Olivieri
To: Leo Taylor
Re: HCS Modules

Leo - thanks for the info. Since you seem to be using a lot of modules, just like me, I have a question. What have you done about the erratic nature of operation when the module you want to control is on one side of the 220VAC line and the controller is on the other side? I was thinking of bridging the two 1lOVAC lines with a O.luF, 600-Volt capacitor. Any comments or suggestions?

Msg #3279
by Leo Taylor
To: Jack Olivieri
Re: HCS power bridge
[Reply to msg #3275]

A common question! I had all my modules working dependably sometime ago, then I had my house upgraded to l00-Amp service. All of a sudden things were missing commands. I have tried a 1uF oil-filled cap, and it does work. I was uncomfortable with it though; what if someday (perhaps years from now) the cap fails? 100 Amps at 220 Volts can do wonders to a large cap! So I rearranged the circuits in my breaker box, putting all computer and BSR stuff on one side, and refrigerator, heat, and air conditioners (except bedroom which is controlled) on other side. I now have dependable action from the HCS, though my BSR-brand Control Box on the nightstand can't hit the living room. Also, be aware that some devices with caps across the power will lower the output of BSR transmitters and the HCS. I had to remove a cap from my STAR SG- 10 printer: I missed events when the printer was on!

Msg #3295
by Jack Olivieri
To: Leo Taylor
Re: HCS power bridge
[Reply to msg #3279]

Leo - thanks for the info. I have 200-Amp service and with 40+ circuits it would very difficult to re-wire. Unquestionably there needs to be some protection for a capacitor if shorted (fuse?). Interesting comment about the fact that some devices have caps across their power inputs. That could explain some things I have been seeing. I would assume that this problem has been solved in a variety of ways.
Again, thanks!

Msg #3328
by Leo Taylor
To: Jack Olivieri
Re: HCS power bridge
[Reply to msg #3295]

Jack -
Final word on power bridge. I had a fuse in series with the cap while it was installed, found it had to be about 3 Amps! Of course, that's "funny" current; it actually may have reduced my bill by power factor correction. I fixed my STAR printer by clipping out the cap. Now I have to figure out what part of my 68000 Atari dulls the BSR!

Msg #3293
by Bob Munck
To: ALL USERS
Re: X- 10 on 2 "sides"

I'm running a fairly extensive set of BSR X-10 modules controlling lights, furnace, fans, hifi, and so on, programmed by a Radio Shack Color Computer (CoCo) and a number of controllers. The CoCo and some of the controllers have trouble reaching various appliances unless I turn on the oven. My conjecture is that they are on different sides of the 220V line feeding the house and that the oven provides a signal path across from one 11OV circuit to the other. Is there any more permanent and less expensive way to provide this signal path? For instance, the clothes dryer has 220V/3-wire connections. Could I put a capacitor across the two hot wires? What size and rating? Second question: has anyone tried to RECEIVE X- 10 signals with a computer. I'd like to use my controllers for input (on one set of house codes) and have the CoCo do all control signals (on another set). I'm going to try to write code in Pascal for OS-9 to do this. Any suggestions?

Thanks, --Bob Munck

Msg #3298
by Steve Ciarcia
To: Bob Munck
Re: BSR
[Reply to msg #3293]

Bob,
Both you and Jack Olivieri seem to be having BSR problems. I may not have a solution for you but let me tell you what I've done in my house. I have about 22 BSR modules in operation with a Micromint HCS. The modules are on both sides of the line. I used two 1 -microfarad mylar capacitors with series fuses across the hot lines. The place that I inject the transmissions (the HCS transformer) is very close to the breaker box which contains the crossover caps.

Results are relatively positive. One side of the line always works while only about 2/3 of the other side will function continuously. I avoid using the outlets that are intermittent.

Regarding spurious on/off BSR modules, I don't have that problem because that is one major reason for using the HCS. The HCS has the capability to restore all modules to their proper setting, either on or off, every four minutes. (It just retransmits the whole output table every four minutes when commanded to do a restore). It also does a restore after any power outage (the HCS has battery backup).
--Steve

P.S. There is one other way to handle both sides of the AC line and that is to transmit into both at the same time. The HCS power transformer is the transmitter coupler to the line. I suppose it would be possibe to drive two transformers, one on each AC line, from one HCS. I haven't tried this, however, so I'm not recommending it as a solution.

Msg #3313
by Bob Munck
To: Steve Ciarcia
Re: BSR
[Reply to msg #3298]

Steve,
Thanks for the info on the capacitors. 1'11 try it. For receiving, I'm planning to tap into a spare module to get the detected signal, and tie it to an input port on my CoCo. I've already tied a house circuit zero-crossing detector to a non-maskable interrupt, so I know when to watch for the "BSR signal received." I'll write software to assemble the incoming codes and check validity.

BTW, I have about 20 modules and a bunch of other more direct controllers, including direct control of the furnace (and several temperature inputs), some control of the house telephones, a speech synthesizer that drives the hifi, and an IR output that can tell my remote-controllable receiver what to do. Future plans include better phone control and using the touch-tone pads on all phones to send commands to the CoCo, switching of audio and video signals, and handling inputs from a weather station. A friend is looking into converting an old 45 juke box to handle CDs for me, and the CoCo will drive it. I hope to get random access to about 100 CDs and to be able to pick selections explicitly from the keyboards or set up a random selection of pieces of a given mood. My expertise is the programming -- I do Modula-2 and Ada at work.
-- Bob Munck

Msg #3342
by Bob Munck
To: Steve Ciarcia
Re: BSR Bridging
[Reply to msg #3298]

Steve,
I didn't quite understand your solution to getting BSR signals from one 11OV circuit to the other. Did you mean

```
     hot                  hot
      o--cap--fuse--cap--o
                 0
              neutral
```

or

```
 hot                              hot
  o--cap--fuse--o--fuse--cap--o
              neutral
```

or some other arrangement? Thanks -- Bob Munck

Msg #3355
by Steve Ciarcia
To: Bob Munck
Re: BSR Bridging
[Reply to msg #3342]

Bob,
The first circuit you presented is the way I wired it.
--Steve

The Infrared Master Controller presented in the March 1987 Circuit Cellar article was a big hit among the hard-core experimenters.

Msg #7413
by Ken Mandelberg
To: Steve Ciarcia
Re: Master Controller

Steve

Right after the article on the Master Controller came out, a few people (including me) asked you if the command set on the controller could be extended to allow the PC to ask the controller to send out one of the IR sequences. The idea was to let the PC use the Master Controller as a dedicated peripheral. You expressed some interest in this and said you would talk to the programmer. Has anything come of this? More generally, have your plans for the future of the Master Controller developed any?

Msg #7424
by Ken Davidson
To: Ken Mandelberg
Re: Master Controller
[Reply to msg #7413]

You might be interested in the file called IRCMDS.DOC in the PROJECTS: area. It turns out that the capability for the IBM to remotely press a key was always there. It just took documenting the relatively untested and undocumented function to make it a feature. Programmers sneak the strangest things into their code when you're not looking.

--Ken

Msg #7516
by Burton Freeman
To: Steve Ciarcia
Re: IR Master Controller

When I try to download a menu to my Master Controller I get "RAM size = - 16" and "Menu will not fit into remote RAM!" Do you know what this means?

Msg #7568
by Jeff Bachiochi
To: Burton Freeman
Re: IR Master Controller
[Reply to msg #7516]

Burt,

Seems as though your MC unit thinks that there is more than the 32K RAM installed. Check for a possible short or open in one of the address lines. A RAM size calculation of greater than 32K would give a negative value. Get out your ohm-meter and magnifying glass. Good Luck!

-- Jeff

Msg #7681
by Lloyd Prindle
To: Burton Freeman
Re: Master Controller
[Reply to msg #75 16]

Burt,

I noticed you had the same problem with your Master Controller. I, too, get "Ram = - 16". I scoped all the address lines and there wasn't an obvious problem. Did you fix yours yet? If you did I would appreciate any help. Thanks.

--Lloyd

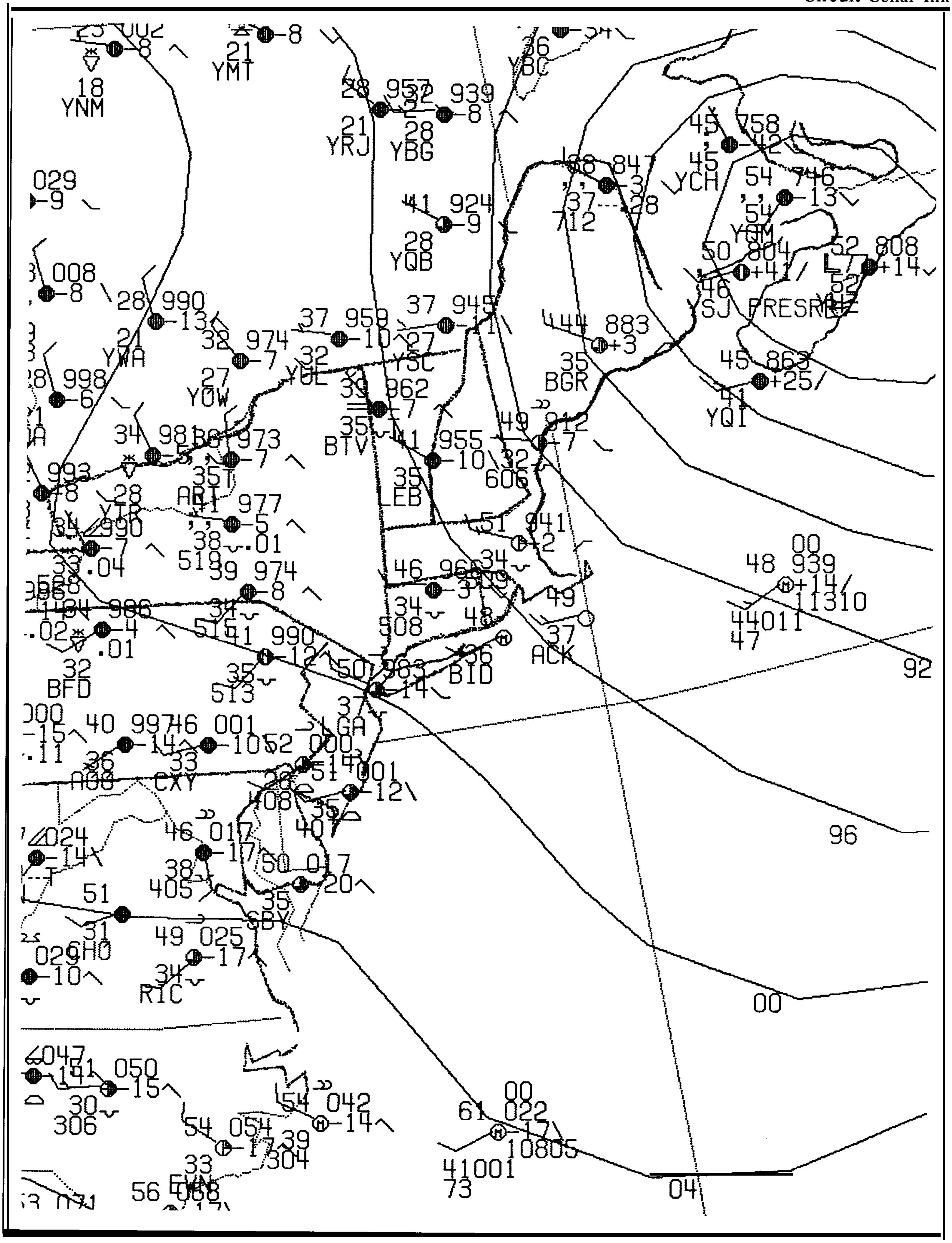
Msg #7871
by Lloyd Prindle
To: Burton Freeman
Re: Master Controller
[Reply to msg #7681]

Burton,

I found the problem with my Master Controller. Address bit 13 was shorted to ground. I used a meter and checked all the address lines. I suspect you have the same short I did. The circuit board design ran this address line along a ground trace. I did not have a solder bridge. It is very hard to see; I spent two hours looking right at it. You must take a sharp knife and scrape away the green mask to see the short. My short was located on the solder side next to the function scroll up switch. Hope this helps you. Let me know if it did.

The following discussion concerns the design of DC-to-DC converters.

(continued on page 40)


YMT
YNM
YRJ
YBG
YBC
YQB
YCH
YQM
YSJ
PRESRVE
YWA
YSC
YUL
YOW
BTV
BGR
YQI
LEB
ART
YTR
BFD
ACK
BID
LGA
CXY
SBY
CHO
RIC
EWN

The Home Satellite Weather Center

Part 1

RGBI to NTSC Converter

by Mark Voorhees

Whether you're planting the spring crops, planning a day at the lake, or just wondering if you should wear your raincoat to work, local, regional, and national weather conditions play an important part in our lives. Most of us just listen to the radio or TV for the weather report, while some of the "older generation" claim to predict the weather by the occurrence of ailments or by folklore. Yet, some of us are not satisfied; we yearn to explore the technology of weather data and forecasting. It is to this group that this series of articles is dedicated.

I spend my "40-hour week" as a broadcast television engineer at a local station, and specialize in the care and feeding of sophisticated technical equipment (especially that which is "computer"-based). I became interested in weather technology when we began exploring the use of specialized weather graphics and data in our newscasts, and have created several home projects -- some of which will be detailed in this and subsequent articles.

The series will describe the construction and operation of a home weather center. When completed, the system will maintain a record of daily statistics, access various weather-related databases, and display graphics. The graphics, generated by your own processing of data and satellite facsimile information, will be similar to that seen on a local newscast! You will even be able to output your graphics to a VCR for long-term storage!

The system will interface to an IBM PC or compatible, but will also operate "stand-alone" in its data monitoring and WEFAX (Weather Facsimile) modes, so that the PC need not be dedicated to the weather functions.

The unit will be built in several parts, each designed to fulfill a given need. The center of our system will be a single-board 68000-series microcomputer, which will handle the routine functions of timed sampling of data, reception and storage of WEFAX and other information, and communication with the PC. Serving this "peripheral processor" will be several interfaces: parallel ports, serial ports, WEFAX demodulator and A/D, and other accessory devices. Interfacing to instruments such as Heathkit's Digital Weather Computer, Relative Humidity Indicator, and Rain Gage will be discussed, as these instruments can provide local data, and are probably more reasonably priced and readily available than most professional electronic instruments.

Several other accessories are "on the drawing board," awaiting your input. As you construct your Home Weather Center, would these features enhance its operation?

- A graphics display card contained within the "Peripheral Processor"?

- Some form of floppy or hard disk storage?

- A built-in Receiver for the WEFAX signals?

- A built-in interface to allow you to use the WEFAX feed from your C-Band Satellite dish?

- A receiver for NWS wire data from satellite?

- Some other accessory of general interest?

As you can see, we'll be covering a wide selection of subjects, and it's my hope that, when we're finished, you not only will have completed a fully functioning system, but you will have also gained knowledge in areas not previously investigated.

I will follow certain conventions throughout the projects that will help in modifying the device, or software, if desired:

- Graphic images will be processed and stored in the Compuserve-sponsored GIF (Graphics Interchange Format). This will allow your graphics files to be transmitted to another machine (even a non-PC compatible, such as a Macintosh) via phone line, and will minimize the disk storage space.

- Operating programs for the PC will be written in "C". On occasion, a BASIC-type program will be included if it serves the application better, but this will be the exception rather than the rule. The purpose here is to maximize portability of the software between machine formats.

- Components necessary to build these projects will be of standard types and values. Where specialized devices are required, I will give sources for those devices.

I intend to offer these projects in kit form (that is, all parts and circuit board). Additionally, bare circuit boards, EPROMs or disks containing the operating programs or other software will also be available. Within each article, a sidebar will give ordering information, including pricing, for the various items. The objective of the kits is to help establish a user base so that the system can be expanded, enhanced, and provide a basis for future articles and discussion.

We will not discuss the science of meteorology or weather forecasting in this series. Several books and publications are available to further your interest. Some sources of useful material include:

-WEATHERWISE Magazine
Heldref Publications
4000 Albemarle Street, NW
Washington, DC 200 16

-The American Meteorological Society
45 Beacon Street
Boston, MA 02108-3693

A good selection of books on weather and related subjects is described in a catalog available from:

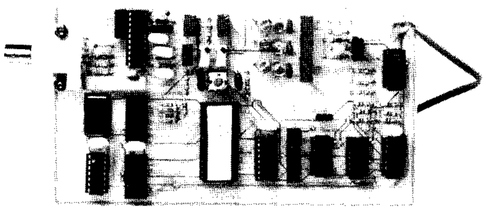
-Wind and Weather
The Albion Street Water Tower
P.O. Box 2320
Mendocino, CA 95460

It's important for you to realize that your input and questions are vital to this series. Send your ideas and questions to the address listed; it is my intention to answer reader questions and discuss submitted ideas during the course of the series, possibly devoting a full installment to your letters.

That said, and having plotted our future course, let's warm up the soldering irons and get started with our first project.

The RGBI/NTSC Encoder

Graphics will be a significant feature of our system, so an RGBI-to-NTSC encoder will be a valuable device. This encoder, with interlaced composite sync, will allow you to record graphic images from the PC on a VCR for later viewing or archival purposes. It can be used with any TTLRGB graphic display card operating at the normal horizontal sync rate of 15.734 - 15.750 KHz. The encoder can serve as a stand- alone unit to record the output of your favorite graphics programs or as a video training device using actual screen outputs.



The encoder combines the separate Red, Green, Blue, and Intensity (RGBI) signals generated by your graphics card into the composite NTSC (National Television Systems Committee) video signal. This signal can then be displayed on a standard composite video monitor or recorded with a VCR. To understand why this encoder is significantly more complex than the $60 devices advertised for this purpose, let's briefly look at the NTSC video signal.

Video Signals

The video image on a television set or monitor is "painted" by an electron beam moving across and down the screen. The beam, starting at the upper left corner, sweeps across at a rate of approximately 15.75 KHz until it reaches the right corner. The beam is then turned off (blanked) as it returns (retraces) to the left edge, and the cycle continues until the bottom of the screen is reached. At the bottom of the screen, the beam is again blanked, and returned to the upper left corner. The downward motion occurs in l/30 second, so the beam "paints" 30 frames per second. The image on the screen is, therefore, "refreshed" at a rate of 30 Hz and is composed of 525 lines. See Figure 1.

However, the persistence of the human eye is such that a noticeable flicker occurs at this refresh rate. To eliminate this flicker, a technique known as interlacing is used. Interlacing gives an effective rate of 60 Hz by increasing the downward rate of the video beam so that every other scan line is displayed. The video image is made up of two fields of 262.5 lines, the first containing scan lines 1, 3, 5 . . . . . and the second containing scan lines 2, 4, 6,.. Each field is displayed in half the time, or l/60 second, and the total frame is displayed in l/30 second. The two fields are staggered, or interlaced, between each other and the flicker is eliminated. See Figure 2.

In addition to being swept across and down the screen, the beam intensity is varied (modulated) to produce the light and dark areas of the video image. The information necessary to control the beam is included on the video signal as blanking and synchroniz-

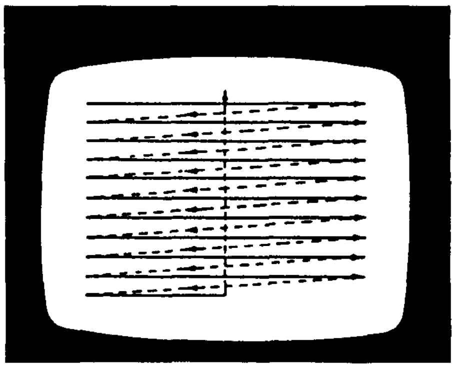


Figure 1

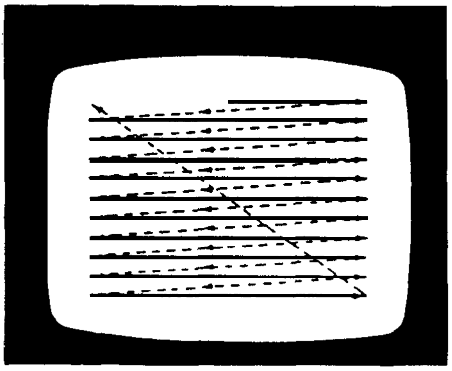


Figure 2

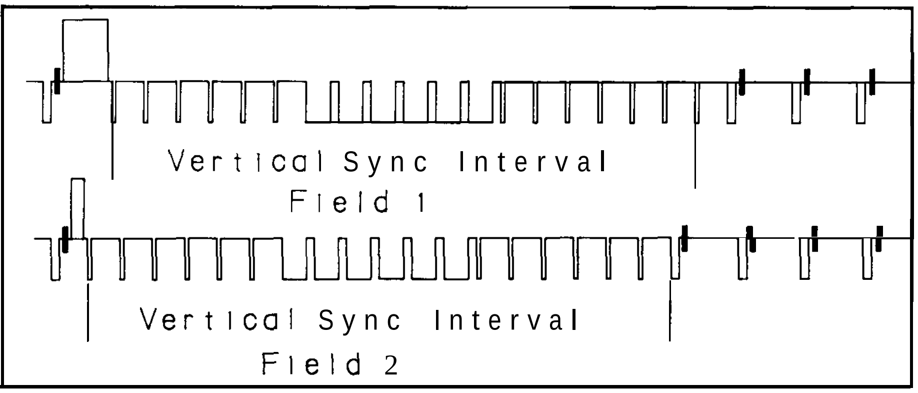


Figure 3

ing pulses. The interlacing of even and odd fields (line 2 placed between lines 1 and 3, for example), is a timing function of the "Vertical Interval" sync signal (Figure 3).

Most computers use a non-interlaced video signal and display the entire frame with a 60 Hz refresh rate. All video information is presented in one vertical sweep and uses timing generated by "Block Sync" (Figure 4). This, in reality, is the lack of Vertical Interval information, and just overwrites the previous field with the new field. The result is possibly a slight picture distortion at the top of the screen, but little more.

Now that we understand the basic video signal, we see that the complexity of our circuit is a result of conforming to the NTSC standard. This circuit is designed to process any TTL-Level RGBI output which operates at the standard rate of 15.734 to 15.750 KHz horizontal line frequency. Therefore, IBM-compatible CGA cards, as well as EGA, PGA, VGA, etc., running in 16-color normal sync modes (for example, EGA's 640 by 200-pixel, 16-color format) will provide normal NTSC output. The 32 KHz horizontal rate signals will not injure the unit, but will not display a visible picture, as this would require that we retime the signal itself. The retiming could be handled by a frame-storage-type circuit, and we may look to that for a future project.

Fortunately, much of the work is done by the Motorola MC1377 chip, which is the heart of the unit. This device handles the matrixing, phase modulation, and timing requirements of the video signal itself, given the necessary input information. The remainder of the circuit converts the signals provided by the Video Graphics Card into usable inputs for the MCI 377.

The primary problems we must solve in conforming the video card output are:

- Matrix the Red, Green, and Blue signals with the Intensity signal to create 2-level (intensified and non-intensified) Red, Green, and Blue signals.
- Turn the Red, Green, and Blue signals on at the proper time so that other information required by NTSC is protected.
- Use the Horizontal and Vertical sync signals from the Video Card to generate the NTSC composite synchronization signal.
- Also, because we want to be able to record the NTSC output, we must synthesize the "vertical interval" area to perform as an "interlaced" sync signal.

The hardest problem to solve is the last one.

A video recorder uses the information in the Vertical Interval to provide timing information to its servo systems (those circuits controlling tape speed, tape position, video head speed, and video head position). Depending on the recorder, and on the width of the Block Sync pulse, many machines will record poorly, and play back poorly, if at all.

To overcome this, we generate the NTSC interlace sync format using a pattern which is programmed in an EPROM. We then combine this (in the MC1377) with our Red, Green, and Blue data, and color reference information, to form a signal which is, for all intents and purposes, a noninterlaced video/interlaced sync composite NTSC video signal.

Theory of Operation

The schematic of the Encoder board is shown in Figure 5. The TTL signals from the PC graphics card enter the circuit at CN5. The Vertical- and Horizontal-rate sync signals are inverted at IC4 and also appear, with their inverted counterparts, at CN3 and CN4, respectively. A jumper at each of these headers selects the polarity of the corresponding reset pulse (some unusual versions of graphics cards generate inverted sync signals). The CN3 and CN4 output signals (selected for normal high, with negative-going pulses) are then differentiated to provide sharp, leading-edge pulses. The horizontal pulse, occurring once during each video line, resets IC7, the element address counter, and increments IC5, the line address counter. The vertical pulse is used to reset IC5.

ICI 1 is incremented by a 14.3 18 1 SO-MHz TTL oscillator module. ICl1-8, the QC output, provides color subcarrier through the SC level control, R13, to the Motorola MC1 377 encoder, ICl. IC7 provides addresses to the 27C256 sync format EPROM, IC8. Thus, there are up to 256 possible sync format addresses in each line of video information; in actuality, about 228 addresses are used. These addresses are used to store the latch triggers which control the sync pulse widths and positions.

IC5 provides the remaining addresses to the EPROM, designating which line is currently being

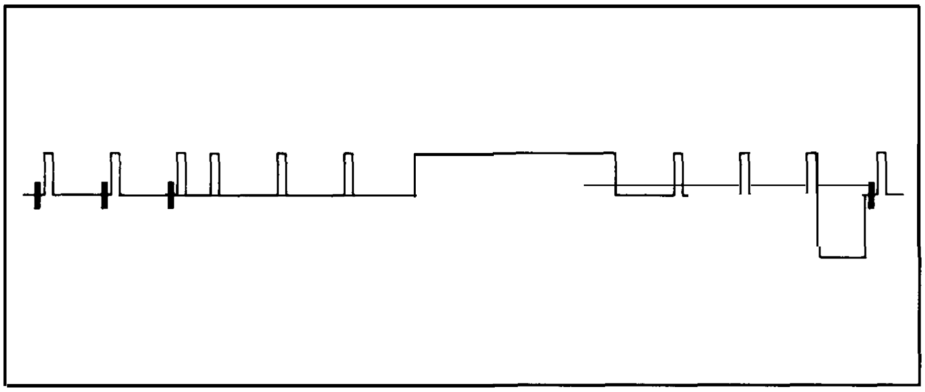


Figure 4

encoded. The line count addresses the EPROM as necessary to properly format the vertical information in the vertical interval. The Q4 through Q7 outputs are diode-OR'ed onto one EPROM address line to simplify the EPROM format.

The actual phase locking of our unit to the graphics card output is a function of the formatting of the EPROM itself. Sync trigger outputs of the EPROM (IC8-13 and IC8-15), blanking control trigger outputs (IC8- 12 and IC8- 18), and burst flag trigger outputs (IC8- 16 and IC8- 17) are count-referenced to the reset address of IC7, so video information will be properly positioned in relation to sync on the output. Additionally, IC8- 11 and IC8- 19 provide vertical holdoff option triggers, if needed (I'll explain further during the testing and setup routines).

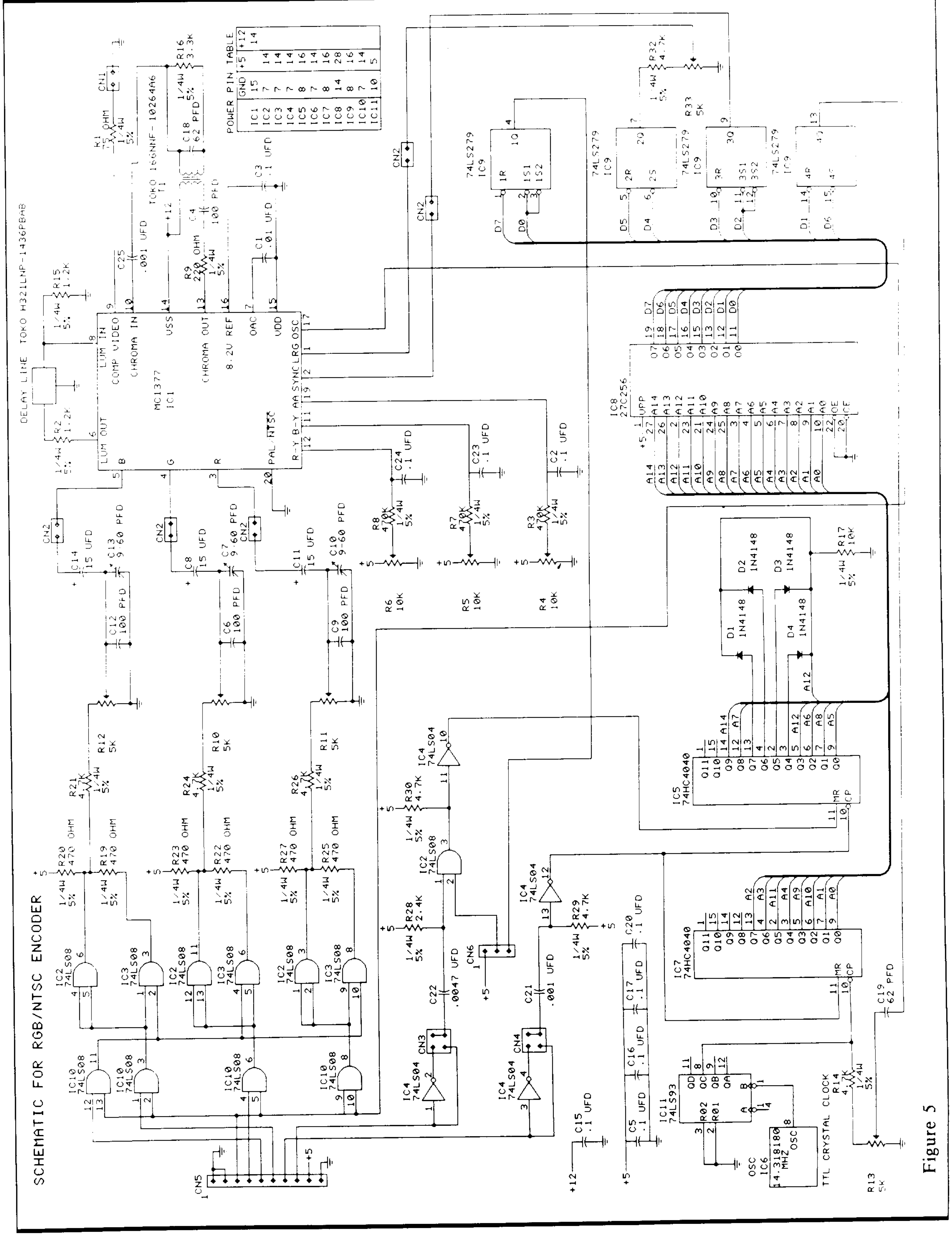
IC9 uses the triggers from the EPROM to generate the composite sync pulse (which routes via CN2 to the encoder), burst flag (routed through R32, CN2, and to the encoder), blanking (which gates the video information in IClO), and vertical holdoff.

Red, Green, Blue, and Intensity video information from CN5 are individually gated by blanking through IC10. The gated Intensity signal is now used as control information for gating of the Red, Green, and Blue through IC3 (Red, Green, and Blue also route to IC2). The respective outputs of IC2 and IC3 (open-collector devices) are used to drive the voltage divider for each color signal. They provide OV when the color is not selected, approximately 2.5Vp-p (peak-to-peak) when the color signal only is selected, and SVp-p when only the intensified color signal is selected.

We have now created simple analog Red, Green, and Blue signals. We feed these signals, after level adjustment, high-frequency rolloff, and decoupling, to CN2, which (for now) is used as a jumper header to the inputs of ICl, the encoder. (This header will be used to connect to features in future parts of our project).

ICI, the MC1377, is the heart of our video encoding and processing system. It contains the matrixing, modulating, and luminance/chrominance circuitry necessary to add the proper video information to our locked sync signal to produce the NTSC format, which is output, via Rl, to the BNC connector CNl.

Only three adjustments exist in relation to ICl: R4, which controls the 90-degree phase relationship



POWER PIN TABLE

	GND	+5	+12
IC1	15		14
IC2	7	14	
IC3	7	14	
IC4	7	14	
IC5	8	16	
IC6	7	14	
IC7	8	16	
IC8	14	28	
IC9	8	16	
IC10	7	14	
IC11	10	5	

Figure 5

basically controls the hue or phase relationship of the blue, green, and red elements), and R5 and R6, which balance the modulators for minimum carrier leakage (minimum color content in whites and blacks).

That covers our circuit overview of the NTSC encoder. In our next installment, we'll look at construction of the unit and alignment. We'll also include a short BASIC program to generate a color-bar signal for setup purposes. ■

Ordering Information:

Bare Circuit Board	**$18.00**
Pre-Programmed 27C256 EPROM	**$10.00**
TOKO Transformer	**$2.75**
TOKO Delay Line	**$9.75**
Oscillator module	**$7.50**
Kit of parts listed above	**$42.50**
Complete kit of all parts to build this project	**$91.00**

Add $2.00 Postage and Handling for each order.

Send correspondence and orders to:

Mark Voorhees
P.O. Box 27476
Phoenix, AZ 85061-7476
ATTN: Encoder

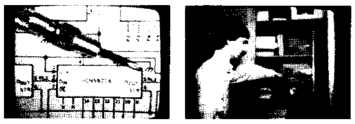
Please include your name, mailing address, and check or money order to cover total amount. Sorry, we do not accept credit cards. Allow 30 days for delivery (money order will speed shipment).

Any updates or corrections will be supplied with your order, as well as being noted in a future installment.

BECOME A CIRCUIT CELLAR PROJECT BUILDER!

Circuit Cellar Inc. kits are a proven vehicle for accomplishing a very special goal. With well designed circuits, pretested key components, documentation, and a knowledgeable support team you can have the thrill of making something you built yourself actually work! This is a CCI project! Call (203) 875-2751 to order your kit or for information.

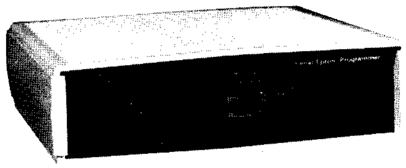
IMAGEWISE - Serial Digital Imaging System


unretouched photos

The Circuit Cellar ImageWise Serial Digital Imaging System was designed to function intelligently as a stand-alone digitieer or as an integral component of a complete tele-imaging system.

DTOl-Full ImageWise Transmitter Full kit	$249.00
DROl-Full ImageWise Receiver Full kit	$249.00
Both Units purchased together	$489.00
DTOl-Exp ImageWise Transmitter Exp. kit	$99.00
DROl-Exp ImageWise Receiver Exp. kit	$99.00
Both Units purchased together	$179.06
Case & power supply for either unit	$49.00

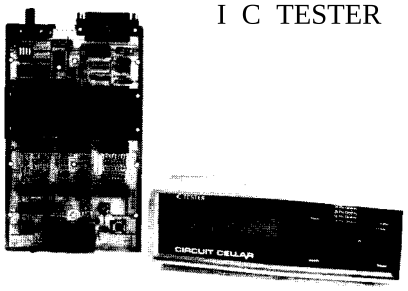
SERIAL EPROM PROGRAMMER



The Serial EPROM Programmer provides a fast and efficient way of programming, verifying and copying a large variety of EPROM types. Supports 27x16 thru 27x512.

SEP27/2 Serial EPROM Programmer complete kit	$199.00
SEP27/1 Serial EPROM Programmer Exp. kit	$89.00
Power Supply	$19.00

I C TESTER



The IC Tester has the ability to identify unmarked ICs as well as designate specific pin failures of hundreds of **74xx00** logic chips.

ICT01-EXP IC Tester Experimenters kit	$99.06
ICT01-FULL IC Tester complete kit	$179.00
ICT02 Complete kit with enclosure	$349.00
2x20 LCD for **ICT01**	$32.00

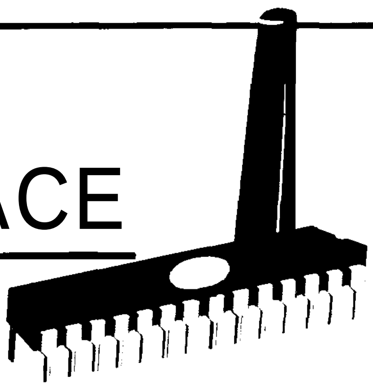
BCC180 - Multi-Tasking Computer

The BCC180 is a 9 **MHz** single board computer with **384K,** 6 parallel ports, and 3 serial ports **onboard.** Multi-tasking BASIC-180 runs 32 simultaneous tasks.

BCC180-Kit-20 .... $296.00

Circuit Cellar Inc-4 Park St., Suite 12-Vernon, CT 06066 (203) 875-2751

FIRMWARE FURNACE



by Ed Nisley

Quickly, now: what's firmware?

Single-chip microprocessors are inexpensive, so they are often used to replace a board full of ICs. Unfortunately, these chips need some instructions to make them perform properly. This column introduces the technique of writing software that works like hardware.

Don't expect machine-independent high-level code, fancy CRT displays, or "enhanced" keyboards. This is lean and mean code, built to run right down on the bare metal. You might think of firmware as epoxy: mix the right combination of hardware and software and wait for it to set up like a rock.

Firmware is generally found in EPROMs and ROMs mounted close to the processor, but 1'11 stretch its definition to include the low-level code that works with any hardware at the "logic gate" level. We'll make occasional forays into IBM PC territory and even high-level languages to handle an interesting topic.

These columns will blend both hardware and software descriptions because you must know some of each to write good firmware. If you feel a little unsure about either, this may be a good way to get your feet wet! Furthermore, you'll be exposed to a variety of microprocessor architectures and programming languages, so there's sure to be something new for everyone.

There won't be any complete, ready-to-run programs. Firmware is created to match a specific hardware configuration, so your system (whatever it is) surely won't be compatible. Of course, if you've been paying attention, you can adapt both the code and circuitry to your needs... that's what we're here for!

I'll use parts of the Circuit Cellar IR Master Controller as the basis for several columns. It uses minimal hardware to solve interesting problems, such as high-speed signal detection and generation, keyboard controls, LCD data output, and even some digital signal processing. The IR Master Controller was described in the March '87 issue of BYTE, so you might want to refresh your memory before diving into this column.

Minimalist Keying

The Master Controller's keyboard interface will serve as a simple beginning. Figure 1 shows the complete keyboard schematic diagram, which uses one IC to support a six-key keyboard. There are two pairs of keys to scroll, in either direction, through the lists of appliances and functions, a "do it" key to trigger the infrared signal, and a "learn" key to begin learning a new signal.

The scrolling keys are used to search through a list, and should repeat rapidly when they're held down. An initial holdoff forces a delay to make it easier to scroll one entry at a time while still keeping a fast scroll. This holdoff also eliminates the need for an explicit debounce function: its period is long enough to bypass any glitches.

Separate routines are used for: holdoff and repeat timings, waiting for a new keypress, and waiting for release. Each calls a single routine that returns the number of the current key (0 indicates that no key is pressed). This technique allows the keyboard's physical layout to be rearranged by changing a single routine to assign new numbers to the keys. That's exactly what happened between the prototype and final Master Controller circuits.

The schematic diagram in Figure 1 shows that the six keys are (electrically) arrayed out in two rows of three keys. Each key can be identified by its location at the intersection of a row and column, as well as a unique key number. A common practice is to make the key number equal to:

key#=(row #)(totalcoiumns)+(column #)

or

key # = (column #)(total rows) + (row #)

The Master Controller uses the first equation. A technique to create a table to relate the row/column key location to the final key number is often used. You can produce any number for any key by changing that table.

A matrix keyboard has many keys connected to each row and column, so there's no way to uniquely identify a single key in the matrix with one action. The process of determining which key is pressed is called "scanning" the keyboard. Each row in the Master Controller is scanned in turn for active keys. It's also possible to scan each column.

The physical key layout need not match the schematic. It makes no difference to the circuitry whether the holes in the front panel are in a rectangular array, a single row, or even a triangle. You can choose whatever arrangement best suits the functions.

A particularly bizarre key layout may complicate the wiring, so you might want to re-assign the key numbers to simplify the PC board layout. The code doesn't really care which number goes with which key, so only the table of values need be changed.

Hardware Connection

Each column of the Master Controller keyboard is connected to a single bit of the 8031's data bus through a bus buffer. That buffer is controlled by an address decoder that is activated whenever the 8031 reads a memory location between addresses COOOH and DFFFH. Devoting 8K of address space to six keys may seem excessive, but the decoder was already there and the address space wasn't being used for anything else.

Although there are three key columns, there are four data bus buffers, and the data bus has a total of eight bits available. The spare buffer allows a fourth column of keys to be added, but the firmware must know how many columns are actually there to remove any noise that will appear on the floating bus lines. The 10K resistors ensure that the three column lines are pulled up to a solid +5-volt level when no keys are pressed.

Recall that the 74LS240 is an inverting buffer. When no keys are pressed, the three low-order bits of the data bus will be binary zero whenever the 8031 reads an address that activates the bus buffer.

Detecting a keypress in a row involves setting that row line to a logic low level (about 0 volts) and reading the column lines. If a key is pressed, the corresponding column line will be pulled to a logic low level because the key connects the column to the row. The 'LS240 inverts that to a binary 1, so the 8031 will see a logic 1 for that column.

The firmware scans the keyboard by activating each row in turn and examining its data. If no keys are active, it deactivates that row and turns on the next one. The scan stops when a key is found, or all of the rows have been tried.

The rows could be selected by writing the appropriate bit pattern to an output port, but there's a simpler way. The bus buffer is activated by reading any address in the COOOH through DFFFH range, so some of the remaining low-order address bus bits can select the rows directly. The Master Controller uses A3 to select matrix Row 1 and A2 for Row 2. Because the 'LS240 inverts the address bits, a binary 1 address bit produces the required low logic level on the row lines.

As with the matrix columns, there are provisions for two more rows driven by bits Al and AO. The Master Controller could support 16 keys in a 4x4 array with no additional circuitry.

The net result is that the state of the keys in Row 1 is determined by reading address C008H and Row 2 by reading C004H. That's all there is to scanning!

What happens if two rows are activated at once, perhaps by reading address COOCH? The schematic indicates that there's no way to tell which row holds the active key. The solution to this is obvious: simply write the firmware so that it only reads from addresses that activate one row at a time.

If two keys in the same row are pressed, the result is simply two '1' bits in the column data. But if two keys in the same column are pressed while the 803 1 is scanning the keyboard, the outputs of the two bus buffers will be connected together through the keys. One buffer will be high and the other low, so the short circuit will impose some high currents. Fortunately, the buffers can withstand momentary abuse, and the scanning duty cycle is so low that no damage will occur.

Software Shenanigans

Listing 1 shows what's involved in GETKEY, the keyboard scanning routine. There are several similar blocks of code in the middle of the routine (sometimes code can be analyzed by taking off your glasses and standing back a bit). Each block handles the scanning of a single row and encoding the results. A branch to the end of the routine occurs when a key is found.

Notice that there are four blocks and only two rows. The reason for this will be discussed shortly.

The Intel 8031 has a rather clumsy interface to its 64K bytes of data memory. The memory address must be loaded into the 16-bit DPTR (Data PoinTeR) register for use with the MOVX (Move External) instruction. This curiosity occurs because the 8031 is an offspring of the older 8048 and was never really intended to use much RAM. Remember, 64K of RAM is still a lot for most controller applications.

The addresses required to scan each row are defined in EQU statements so that if the hardware changes, it's easy to reassign the

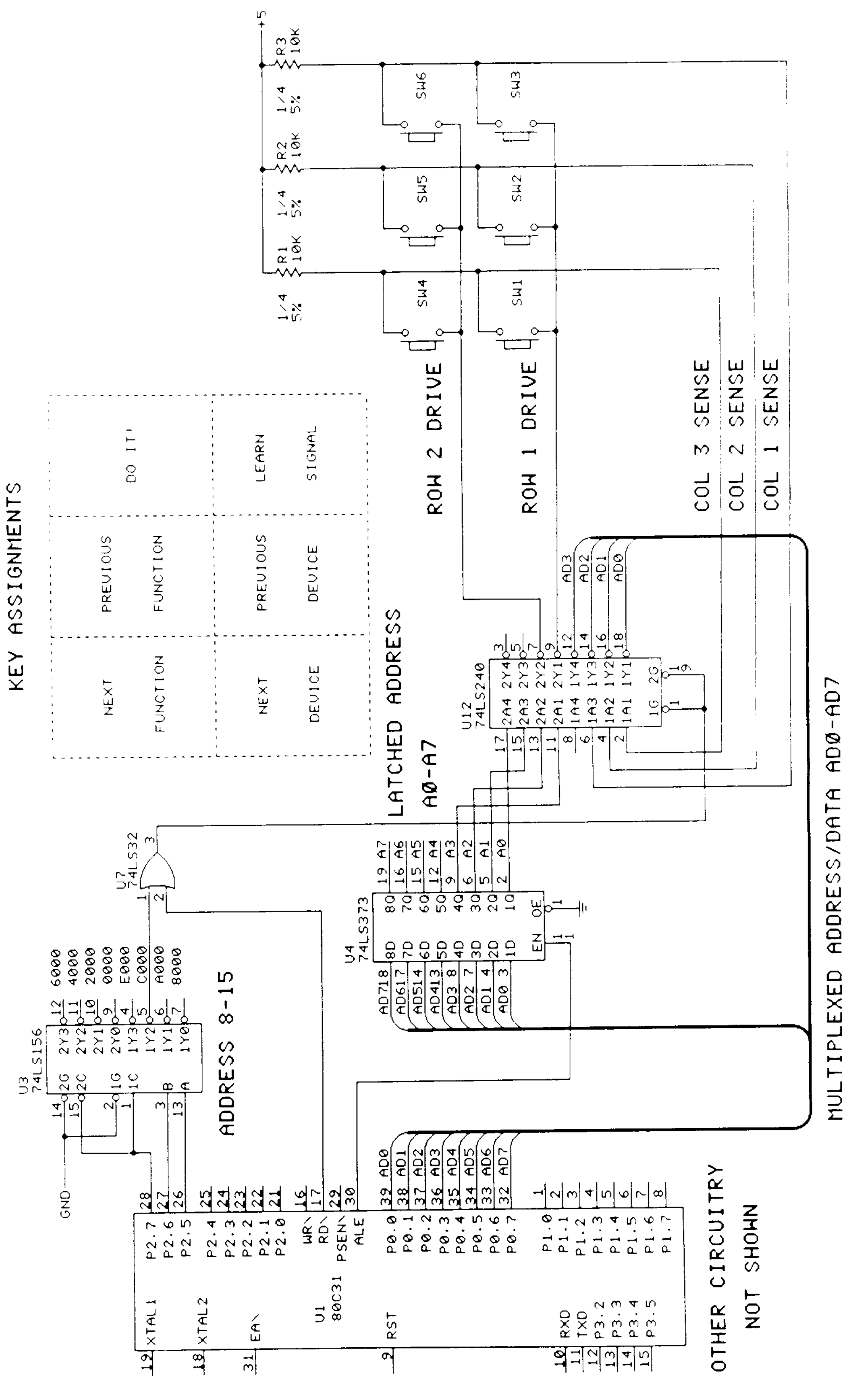

KEY ASSIGNMENTS
NEXT FUNCTION
PREVIOUS FUNCTION
DO IT!
NEXT DEVICE
PREVIOUS DEVICE
LEARN SIGNAL
ROW 2 DRIVE
ROW 1 DRIVE
COL 3 SENSE
COL 2 SENSE
COL 1 SENSE
LATCHED ADDRESS AØ-A7
MULTIPLEXED ADDRESS/DATA ADØ-AD7
ADDRESS 8-15
OTHER CIRCUITRY NOT SHOWN
U1 80C31
U3 74LS156
U4 74LS373
U7 74LS32
U12 74LS240
R1 10K 1/4 5%
R2 10K 1/4 5%
R3 10K 1/4 5%
+5
GND
SW1
SW2
SW3
SW4
SW5
SW6
6000
4000
2000
0000
E000
C000
A000
8000

Figure 1

addresses.

Each block of code sets up the row number to be scanned in the B register, loads the appropriate memory address to activate that row, reads the bus, then strips off the five extraneous bits that aren't connected to the key matrix. Binary 1 bits in the result indicate pressed keys in that row.

It's possible that two keys may be pressed at once, so the raw data value isn't directly useful as a column number. The table at KEYTAB translates the eight possible values into a true column number. If more than one key is pressed, it will return the column number of the leftmost key. The entries in this table can be changed to match a new layout of the columns.

In addition to 64K of data memory, the 8031 can use up to 64K of entirely separate program memory. The MOVC A,@A+PC instruction returns the byte in program memory located at the address formed by adding the contents of the Accumulator to the current PC (Program Counter). The Master Controller includes a special circuit that combines the two address spaces into one, but this doesn't affect the MOVC instruction.

The ADD A,#keytab-$-3 instruction just before each MOVC creates the appropriate offset in the Accumulator to reach the first entry in KEYTAB. The offset is the sum of the column bits in A, the base address (KEYTAB) minus the address of the ADD instruction, minus the length of the ADD and MOVC instructions. There are times when I wish that the 8031 was just a trifle easier to use...

If the column number obtained from the table isn't zero (indicating that at least one key was indeed

Listing 1 -- Keyboard Scanning Subroutine

```
;----
; Keyboard addresses

row 1       EQU  C008H          ; kbd top row input
row2        EQU  C004H          ;.. next
row3        EQU  C002H          ;.. next
row4        EQU  C001H          ; kbd bottom row

;----
; Get current key number in A
; Numbers start at 1 and end at 12
; Zero is returned if no key pressed
; Multiple presses are not always detected

getkey     EQU    $
         PUBLIC getkey

         PUSH   DPH              ; save bystanders
         PUSH   DPL
         PUSH   B

         MOV    B,#0             ; indicate top row
         MOV    DPTR,#row1       ; pick up top row
         MOVX A ,@DPTR
         ANL    A,#07H           ; strip off unused bits
         ADD    A,#keytab-$-3    ; correct for code offset
         MOVC   A,@A+PC          ; pick conversion from table
         JNZ    gotkey

         MOV    B,#3             ; indicate second row
         MOV    DPTR,#row2
         MOVX   A,@DPTR
         ANL    A,#07H
         ADD    A,#keytab-$-3    ; correct for code offset
         MOVC   A,@A+PC          ; pick conversion from table
         JNZ    gotkey

         MOV    B,#6             ; indicate next row
         MOV    DPTR,#row3
         MOVX   A,@DPTR
         ANL    A,#07H
         ADD    A,#keytab-$-3    ; correct for code offset
         MOVC   A,@A+PC          ; pick conversion from table
         JNZ    gotkey

         MOV    B,#9             ; indicate bottom row
         MOV    DPTR,#row4
         MOVX   A,@DPTR
```

(continued page 39

```
        ANL     A,#07H
        ADD     A,#keytab-$-3; correct for code offset
        MOVC    A,@A+PC     ; pick conversion from table
        JNZ     gotkey

        MOV     B,#0        ; no key, ensure zero out

gotkey  ADD     A,B         ; add row and column numbers

        POP     B           ; restore bystanders
        POP     DPL
        POP     DPH
        RET

;----
; Translate key column number into true column
; Right and left depend on which way you're facing...

keytab  EQU     $
        BYTE    0         ; 00 - no key
        BYTE    3         ; 01 - rightmost key
        BYTE    2         ; 02 - center key
        BYTE    2         ; 03 - center + right
        BYTE    1         ; 04 - leftmost key
        BYTE    1         ; 05 - left + right
        BYTE    1         ; 06 - left + center
        BYTE    1         ; 07 - all keys down

        END
```

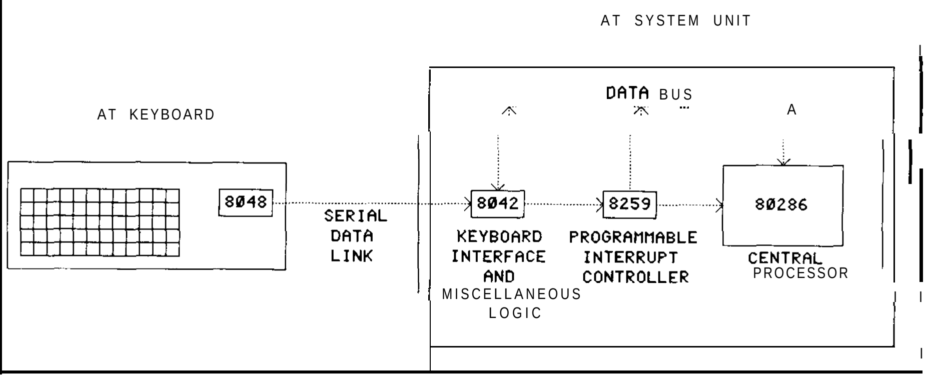


Figure 2

pressed), a jump to the end of the routine will prevent any more rows from being scanned. Finally, the row number in B and the column number in A are added to form the key number returned in the Accumulator.

Now, why would the code scan four rows when there are only two in the hardware? It turns out that the prototype version of the Master Controller used those six additional keys to provide debugging functions that were not intended to be available in the final product. Those functions remain in the firmware, waiting patiently for a key code that can never occur.

At the beginning of this article, you may have wondered why the Master Controller used a matrix arrangement instead of six buttons connected directly to an input port. The reason should now be obvious: a matrix allows easy expansion (and contraction) of the number of keys with minimal hardware changes. A single port could access up to eight switches; the matrix can handle 16 with no new hardware.

The Key Point

Figure 2 shows how keyboard input is handled on an IBM PC/AT. The key matrix is scanned by an Intel 8048 in the keyboard enclosure. The resulting "scan codes" (sound familiar?) are transmitted over a serial link to an 8042 on the AT's system board, where they are buffered before being processed by the BIOS routines. The BIOS code translates the scan codes into ASCII characters and special function codes.

You may never have thought of an AT as a three-way multiprocessor, but that's the way it works. All that for just one keystroke! The principle remains the same: the 8048 in the AT's keyboard is doing much the same thing as the 8031 in the Master Controller.

If you need a keyboard, you could do far worse than a simple one-IC design and some firmware to fit. ■

(continued from page 26)

Msg #8712
by Dale Nassar
To: Steve Ciarcia
Re: Inverter

Steve,
What is the best way to get - 12V @lOOmA and -5V @lOOmA (both regulated) from a 12V battery? Thanks.

--Dale

Msg #8745
by Steve Ciarcia
To: Dale Nassar
Re: Inverter
[Reply to msg #8712]

5 mA is easy, 100 mA takes a little skill. What you need is a DC-to-DC converter with inverted output. I have done two articles on the subject, Oct. '78 and Nov. '81, which have some applicable circuits. Another option is to call Maxim and get their applications manual (Maxim's number is (408) 737-7600). In it there are lots of circuits. You'll also need a source for chokes.

--Steve

Msg #8720
by Bob Paddock
To: Dale Nassar
Re: Inverter
[Reply to msg #8712]

Look at Maxim's MAX6xx family of parts. I don't have the data book right here, but I think it's the MAX635 that will do positive-to-negative switching.

Msg #8784
by Dale Nassar
To: Bob Paddock
Re: Inverter
[Reply to msg *#8720]*

Bob, thanks for the info. I'm not sure that the MAX635 will give me enough current; the MAX680 gives 10 mA. But I have ordered the data sheet.

--Dale

Msg *#8824*
by Bob Paddock
To: Dale Nassar
Re: Inverter
[Reply to msg #8784]

I finally remembered to bring the Maxim book up here. The following are DC-DC voltage inverters with +2V to +16.5V input: The MAX635 output is set for -5V, MAX636 = - 12V, MAX637 = -15V. With the addition of two resistors all three can have their output adjusted. On page 2-4 of Maxim's "1987 Analog Data Acquisition Applications Seminar," there is a circuit for +5V in to +15/-15V @ lOOmA out. With alittle modification you could get your +12V to - 12/-5V.

Msg #8856
by Dale Nassar
To: Bob Paddock
Re: Inverter
[Reply to msg #8824]

Bob, thanks for your information on the Maxim ICs. I learned a lot this week winding my own coils but am glad to hear that I don't have to. I'll be using the part you suggested.

--Dale

Well, that's all we have space for in print this month. But, if I've sparked your interest and you want to read the hundreds of other messages on line, give the CCBBS a call some time (203-871-1988). Remember, my name is Ken, and I'm your sysop.

WRITE FOR INK!

Writing technical articles won't necessarily make you rich and famous but it might be just what you need as incentive to finish that lOO-MIPS computer you started last summer. Or, if your expertise is software, are current approaches to artificial intelligence or database manipulation all wet and you have an improved technique? Perhaps it's time you presented your talents to the world.

Unlike most narrowly publications these days, CIRCUIT CELLAR INK's charter is to cover a wide variety of hardware and software technology and ideas. If you have a software or hardware project worthy of publication in INK we'd like to discuss it.

Send your project outline to Harv Weiner, CIRCUIT CELLAR INK, PO Box 772, Vernon, CT, 06066, or contact him on the CIRCUIT CELLAR BBS at (203) 871-1988.