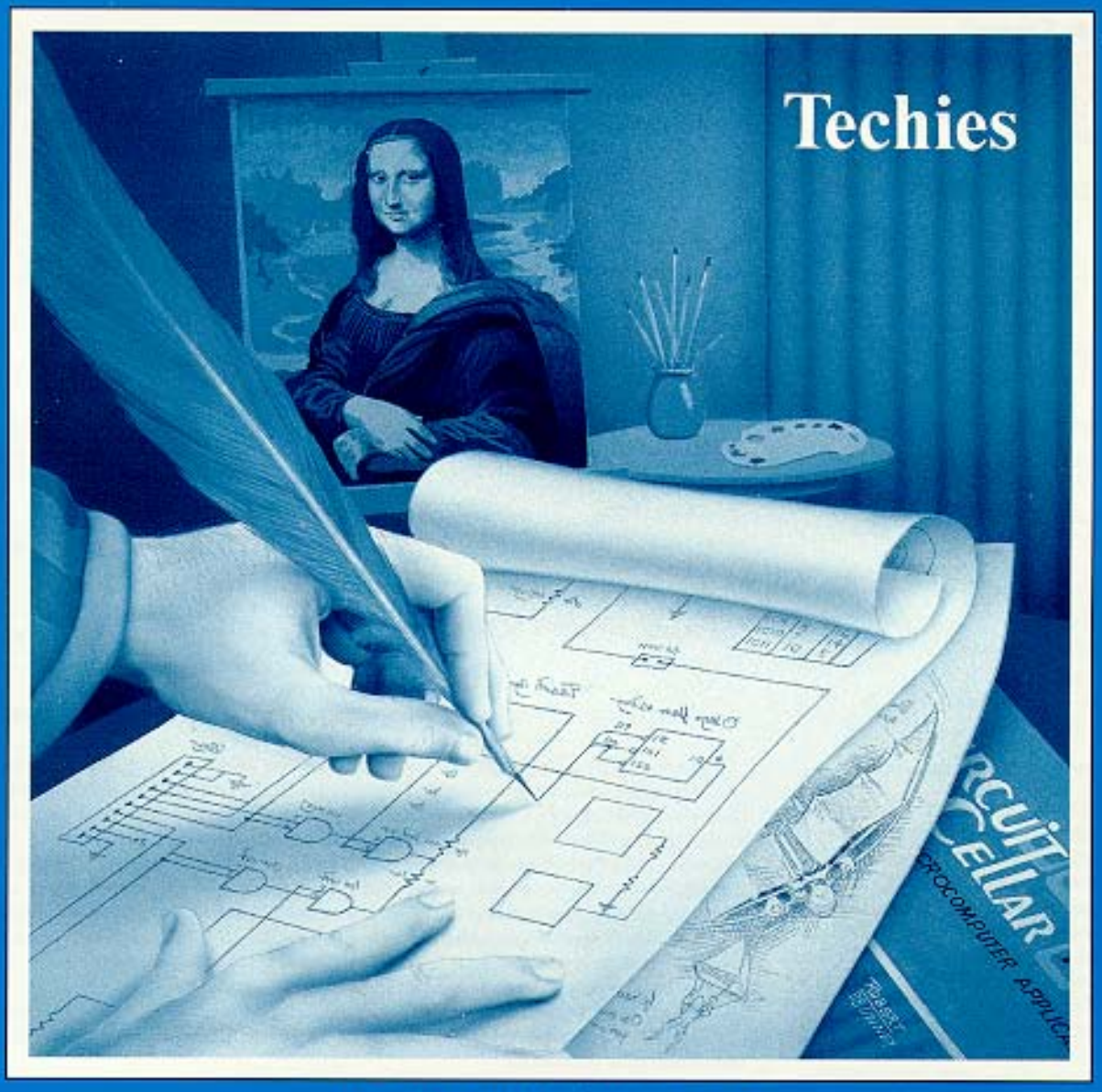
$3.95

CIRCUIT CELLAR INK

MICROCOMPUTER APPLICATIONS



March/April 1988 -- Vol. 1, No. 2

CIRCUIT CELLAR INK
MICROCOMPUTER APPLICATIONS

CIRCUIT CELLAR. INK (ISSN 0896-8985) is published bi-monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT06066 (203-875-2751). Second-class postage applied for at Vernon, CT. One year (6 issues) charter subscription rate U.S.A. and possessions $14.96, Canada $17.95, all other countries $26.95. All subscription orders payable in US funds only, via international postal money order or check drawn on US bank. Direct subscription ordem to Circuit Cellar INK, Subscriptions, P.O. Box 3378, Wallingford, CT 06494 or call (203)875-2199.

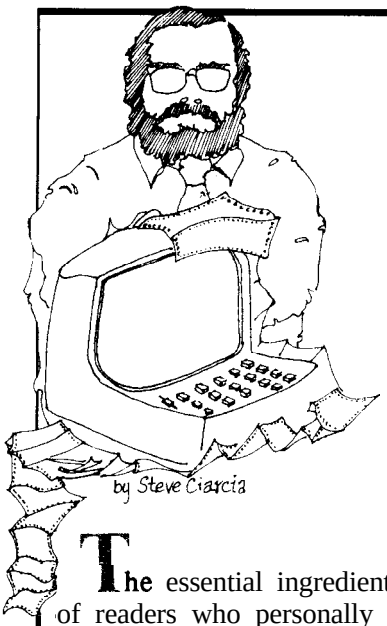
TABLE of CONTENTS

Circuit Cellar BBS - 24 Hrs. 300/1200/2400 bps, 8 Bits, No parity, 1 Stop Bit, 205-871-1981

The schematics provided in Circuit Cellar INK are drawn using SCHEMA from Omation Inc. All programs and schematics in Circuit Cellar INK have been carefully reviewed to ensure that their performance is in accordance with the specifications described and programs are ported on the Circuit Cellar BBS for electronic transfer by subscribers.

Circuit Cellar INK makes no warrantier and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors. Furthermore, because of the possible variation in the quality and condition of materials and workmanship of reader-assembled projects, Circuit Cellar INK disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from plans, descriptions, or information published in Circuit Cellar INK.

POSTMASTER: Please send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 3378, Wallingford, CT. 06494



EDITOR'S INK

The Core Audience

The essential ingredient in any successful publication is a dedicated core audience. The core audience consists of readers who personally identify with the purpose and value of a publication. In the extreme, publications with strong identities even develop unique "personalities" that core readers form lasting relationships with.

Depending upon a publication's purpose, the composition and necessity for a dedicated core audience varies. I don't generally think of Good Housekeeping or TV Guide as being particularly concerned about core audiences, for example. But, a technical magazine like Scientific American, which is designed for a minimum level of scientific understanding and interest, has almost a cult following.

Publishers have to be careful when they change their goals. Unlike time-critical news or event-based magazines, core-audience-critical technical publications have to be constantly aware of their prime directives so that they don't dilute or ignore the core audience in a frenzy to meet financial objectives.

A circulation manager once described the delicate process of increasing publication revenues to me in the following way: Think of carefully breaking an egg in the center of a spinning turntable. At the center is a well-defined and boundaried area (the yoke) surrounded by an almost infinitely expandable unboundaried area (the white). Metaphorically speaking, the yoke is the core readership and the white is the general-interest audience of the publication.

Expansion and contraction of readership and revenue is dynamic. The rotation rate of the turntable represents a complex compromise of editorial, reader, and revenue objectives. When the platter spins at a reasonable rate, following a well-planned path, the yoke will remain intact and the white will expand out uniformly. If, as the result of an abrupt change in objectives, however, the platter is spun very fast or with sudden acceleration, the situation changes radically.

Initial appearances can actually misrepresent actual results. The white will indeed expand out very quickly and appear to demonstrate substantial gain but, left unchecked, this applied energy can rupture the yoke causing the core audience to spill out at the same rate of expansion.

Reader profiles are not just emotionless demographics. Specialized journals like INK have a vested interest in a cohesive and dedicated core audience. The uniqueness of a technical journal is that its readers are also its authors. While we are often disparagingly called "techies," INK truly has a nonsuperficial core reader and experience shows most of these readers are also experts at the center of innovation. As a core reader of INK you are in good company!

Successful technical journals are those that have forged a proper balance between business and editorial objectives. I for one am dedicated to keeping INK's direction consistent with its significance. I am proud to be a techie, and I make no bones about it!

Steve

READER'S INK

Letters to the Editor

Dear Steve,

As a nonelectrical engineer, I have followed your articles in BYTE for many years. I've put together a few of your kits. I have always admired your ability to write about electronics in an understandable way. Occasionally I have plagerized portions of your work or made a few copies for the students in my classes.

I've been interested in instrumentation throughout my professional career (as an agricultural engineer) and have always played with electronics. Several years ago I struck out on my own (nonengineering) but found that I missed it so I returned to another university.

While there, I homed in on the BCC52 and taught a course in experimental data acquisition using it. The more I used it the more I liked it. I am now developing a course in Sensors and Control for students headed into industrial jobs, and food engineering.

I have always wished there was more of what you write about. I have subscribed to Popular Electronics since high school and now Radio Electronics and Computer Smyth. I welcome Circuit Cellar INK and just thought you should hear from one of the silent (electronic) majority.

Keep up the tutorials; I always learn something new.

Kenneth A. Jordan
Tucson, AZ

Dear Ken,

Thanks for the vote of confidence, Ken. I will endeavor to keep Circuit Cellar INK both entertaining and informative. I looked over the class materials and BASIC-52 programs you sent. The way it is presented could benefit many people. Perhaps you should upload these class materials to the Circuit Cellar BBS and share them with other readers.

-- Steve

Dear Steve,

It has been a long time coming, but you've finally got it in published form...BRAVO! I can't wait for the next issue. I would like to comment on Mr. Tom Cantrell's article on "RISC vs Reality, an Exercise in Acronyms." It appears as if the one-sided, so called "facts" about RISC CPUs and the performance they can provide is something that Mr. Cantrell doesn't seem to fully understand. RISC CPUs may not be for every user, but if it is speed and performance you desire in your computing needs you just may have found what you've been looking for.

RISC doesn't mean cutting corners on the hardware, but I will agree that the software will need to be much smarter on these types of systems. For some of us advanced "hackers," computational performance is all that matters. Software or no. I hope he reads your opening editorial. It is what's inside the box that counts! Not to mention, your specific needs and/or applications when it comes to the RISC. Steve, if I may be so bold, let me direct my additional comments to Mr. Cantrell personally.

Frankly, I'm shocked Mr. Cantrell. RISC -- is for student computers? Acronyms! What are people going to think about RISC when they read your article? The power that can be achieved from a RISC CPU offers tremendous and dramatic performance improvements over traditional computer architectures. By reducing the use of microcode, limiting references to main memory, and using RISC optimizing compilers that manage and analyze the instruction flow, the results are quite clear. Compilers maximize the concurrency of such operations as storage, branching, and floating-point instructions. The machine code they produce improves application performance by maximizing register usage, minimizing memory references, and grouping similar operations.

Notice the "key" word here is "application"! This optimization of instruction flow reduces wait states and increases the efficiency and performance of the RISC CPU. Using simpler instruction sets and minimizing microcode translations result in CPU instructions being executed in one machine cycle and increased throughput respectively. In addition, if instructions are uniform in length, the need to order or schedule instructions prior to their execution is reduced. The result is a smoother flow of instructions through the CPU and faster execution times. Such architectures most often depend less on memory referencing for instruction fetching and translation as well; that further increases

system efficiency.

Speaking of memory, RISC CPUs employ data and instruction caches that greatly reduce the need for argument and instruction fetching thereby accelerating system speed. Instruction set complexity is a VERY BIG DEAL! System performance may only be one factor in achieving success in the market, but depending on what market you're in, it may be everything you need.

Don't get me wrong, I enjoyed your editorial and look forward to a response from you or Steve concerning my somewhat biased view of the RISC CPU. In closing, I thought you might be interested in my favorite toy, which is what I've been referring to throughout this letter.

It is the Prime PCXL 5521 Graphics Workstation. A supermini VME computer utilizing a fully customized CMOS VLSI, IO-MIPS, 32-bit RISC CPU; 4-MFLOPS coprocessor; 12 MB ECC MOS memory; 17 VLSI 10-MHz highly pipelined graphics engines handling 145,000 3D 32-bit FLOP coordinates per second, accelerated by 14 custom-VLSI processors for graphics management, providing 16.7 million RGB, 24-bit displayable colors, 1280 x 1024-pixel resolution for true, real-time interaction with 3D graphics and complex 4D transformations.

Enough acronyms for you? My business would be next to impossible without the RISC CPU. I'm now an independent designer of ultra high resolution computer animated graphics for the post video production industry, and previously have been a systems telecommunications engineer for several years, working specifically with the D.O.D. For writing, I use a PC/XT compatible, manufactured by Ericsson Information Systems and Microsoft. Word processing by MultiMate Advantage 3.6.

Rich Gaskill, President - The Graphics Workshop
Garland, TX

Dear Mr. Gaskill,

I'm sorry if you were disturbed by my editorial. I went out of my way to say a few good things about RISC. I'm all for anything that helps keep the big guys honest. I concede that as a marketing issue, RISC has a big impact (it is giving a lot of people an excuse to change). Witness all the excitement over the AT&T-Sun affair.

My original statement, "Instruction set complexity is not that big a deal," is open to interpretation. I'll restate it as "Instruction set complexity has little to do with system performance."

I agree with most of what you say, because most of it isn't about instruction sets. Rather, as is often the case, what starts as a pitch for RISC evolves into a discussion of cache, multiregister sets, and optimizing compilers. These were all deemed worthy long ago. Thirty lashes with an Ethernet cable (and not the thin stuff)!

"RISC CPUs employ data and instruction caches that greatly reduce the need for argument and instruction fetching, accelerating system speed." Arrgghh -- more RISCSpeak! Do you mean to imply that CISCs don't, or can't, have cache ? What does data cache have to do with instruction set complexity?

As for "Such architectures most often depend less on memory referencing for instruction fetching...," even RISC proponents admit the architecture exhibits more, not less, instruction traffic than CISC. For example, comparing two memory locations requires three instructions on a RISC (LD,LD,CMP) versus one instruction on a CISC (CMP). The real argument is how much the increase in number of instructions executed is offset by faster instruction execution.

Even your workstation RISC CPU needs a floating-point coprocessor (or a more optimal optimizer). You have complex instructions after all!

The key factors for system performance are multi-processing (architecture) and process/speed (implementation). These, not instruction set complexity, are what separate a CRAY from a PC.

I appreciate your comments as a fellow INK reader. "What's inside the box still counts," not what they say.

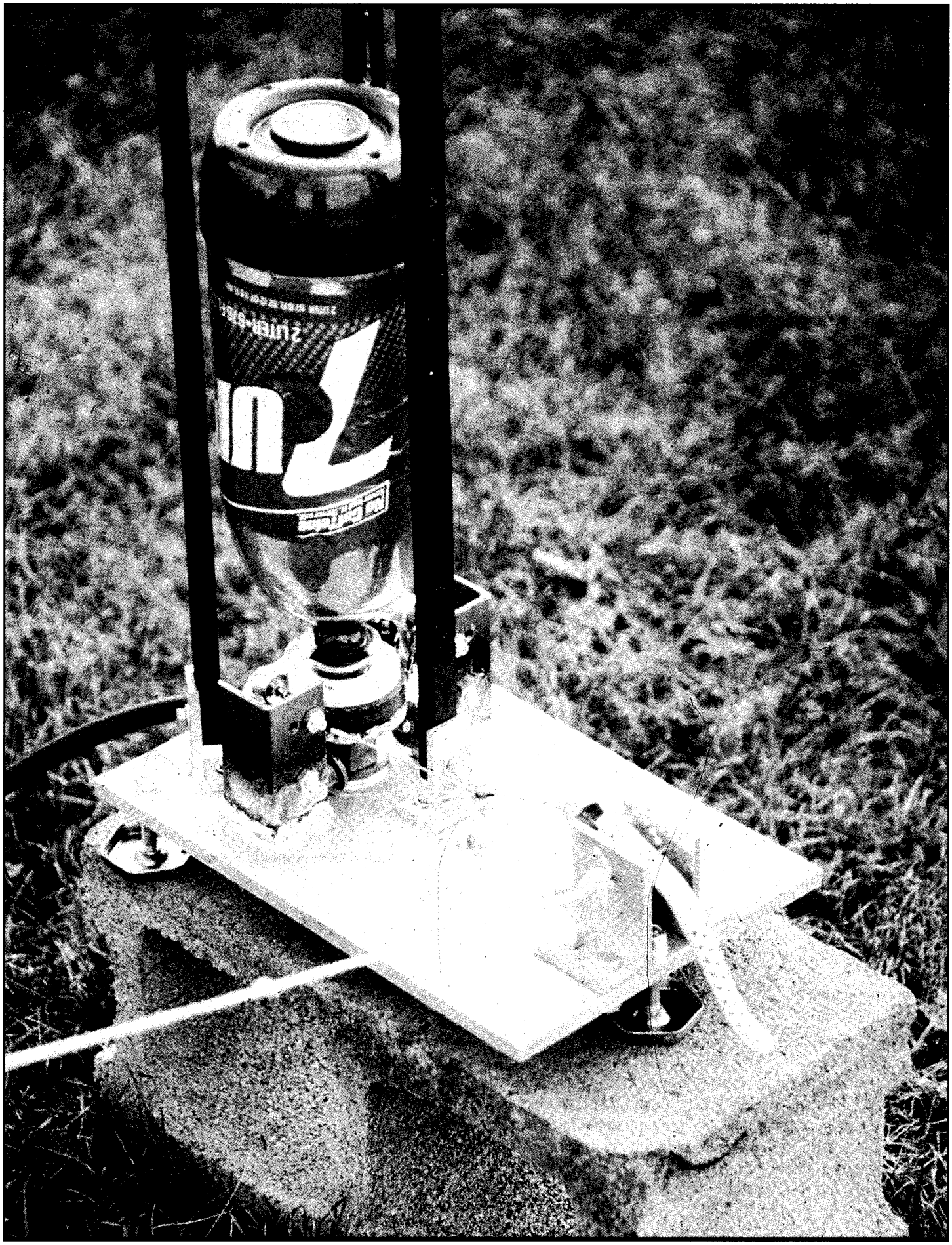
Tom Cantrell

Dear Steve,

Congratulations on your new publication, and I wish you great success. It is badly needed because most of the other magazines contain nothing but low-level applications and advertising hype.

I don't hesitate to tell our readers about other good publications (a little different than BYTE's attitude!!), and I'll announce it [INK] in the next issue.

Best of Luck,
Art Carlson
Editor/Publisher
The Computer Journal
Columbia Falls, MT ■



Circuit Cellar Neighborhood Strategic Defense Initiative

The Ballistic Dynamics of Plastic Soda Bottles

by Steve Ciarcia and Ed Nisley

You know how a. picture is worth a thousand words? Well, a personal visit can amount to an experience of a lifetime. Let me explain.

You may have seen mention of the Circuit Cellar Research staff here and in other publications. No, it is not tongue-in-cheek oxymoron attempting to suggest that I do the work of many people. The Research group is a living, breathing bunch of high-tech individuals who do a lot of the background research (some might call it the dirty work) and correspondence for Circuit Cellar and INK. These people live all over the country but the majority of them live in the Northeast.

Last summer I decided to have a picnic/party/meeting at my house for as many of this group who could show up. While we communicated regularly via BBS and mail, it would give a few of us an opportunity to "press the flesh." This would also be my first meeting with Ed Nisley.

From what I am told (they wouldn't lie, would they?), people like Ciarcia-hosted parties. I don't bring in any dancing girls but if you have a tendency to overeat, you're at the right place. I love to cook and I always use parties or entertainment as an excuse to cook all the things heart specialists execute people for even thinking about. You know: hors d'oeuvres, scallop clam chowder with pounds of butter and cream, shrimp provencale in butter and garlic, eggplant Parmesan, chocolate mousse with whipped cream, trifle...Wait...I'm getting hungry even describing it.

In any case, about 25 of us were sitting out on the deck sipping drinks and eating hors d'oeuvres when a Toyota Camry with New York plates wheeled in. Ed Nisley had made it after all.

You never quite know what to expect of ex-IBMers and Ed was no less surprising. He climbed the walk and introduced himself to everyone there (no shy guy here, folks). As we shook hands we stared intently at each other. While slight of stature I sensed this man was a bundle of energy who rarely rested. As he watched me I couldn't help but feel that he was reducing me to prime components through a Fourier analysis. I shouldn't have been surprised; all of us engineers do that. We look at a vase of flowers and draw a schematic.

As we continued to exchange pleasantries, I couldn't help but notice that his car was full of "stuff." Not luggage, this "stuff" looked more like he had stopped at a junkyard someplace. I decided to keep an open mind. "How about a drink, Ed. You're probably thirsty after your trip."

"Well actually, Steve, I don't drink. I still have some refreshments in the car cooler if you need any more, though. I think I still have orange juice, ginger ale, and root beer."

Well, we weren't going to get anything out of this guy by plying him with booze. With my luck he was probably one of those guys who lived on rabbit food, too. One spoonful of chowder was his total caloric intake for the day...Ed interrupted my next thought.

Ed wet his forefinger and held it up into the air, then rotated quickly around toward me. "What do you think, Steve. Six miles per hour from the south-southwest? Do you still have that anemometer from that wireless weather instrument you did as a project a few years ago?" Checking his watch and a small instrument from his pocket he continued almost in the same breath, "What are we, about 1100 feet above sea level? You aren't in any airline flight paths are you?"

I wasn't the only one on the deck who thought this conversation was getting a little strange. I took a sip of my Tanqueray martini, furrowed my brow as a nonverbal "say-what?" and answered, "Actually we're at 1120 feet, if it makes any difference."

Before I could finish, Ed spied something of particular importance on the deck and took off like a shot toward it. He grabbed a 1/4-full 2-liter Coke bottle from the cooler and yelled, "Hey, Steve, can I use this?"

That seemed a bit confusing to me but people also say "can I borrow the bathroom." In actual-

ity, you don't want your bathroom to go anywhere, but a bottle of Coke seemed harmless enough not to question semantics. Perhaps Ed had tired of root beer and orange juice. "Would you like a glass and some ice for that, or do you prefer it straight?"

"Neither! I just want the bottle." We all watched aghast as Ed took the cap off the bottle, held it at arm's length, and inverted it. He poured its contents into the shrubbery surrounding the deck and then called back, "Got any more?"

A quick thought of Ed stopping and picking up 5-cent-deposit soda bottles along 100 miles of highway between Connecticut and New York flashed through my mind. "I have a barrel of soda cans that I've been saving for the neighborhood kids that you can have."

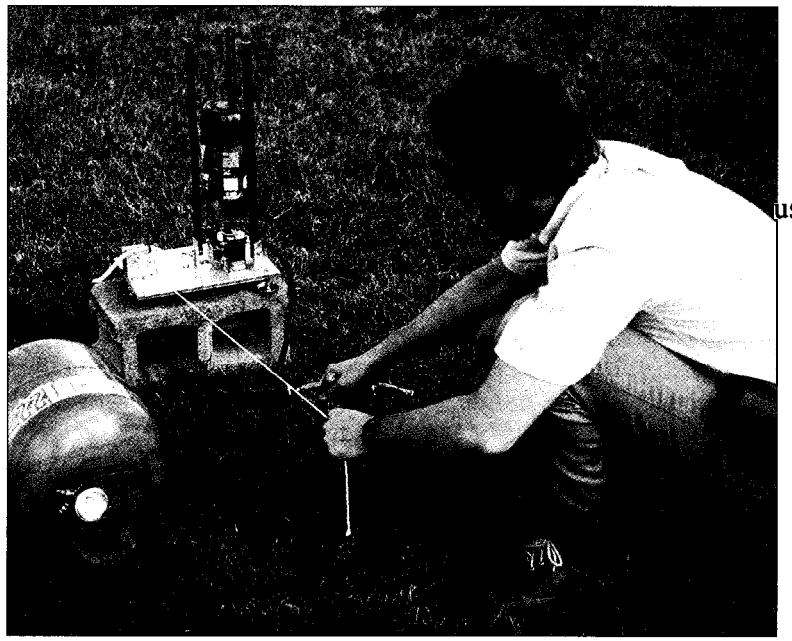
I took a long drink of my martini as a few guests politely chuckled. Most ex-IBMers I had met were exceptional people. The "ex" connoted that they could not live in a homogenized community and needed to speak as an individual, but they usually applied themselves brilliantly in other endeavors. Was I seeing a new side to the coin?

"No, Steve. I need plastic bottles. Got any l-liter?"

Ed finally ceased his exhuberant behavior long enough to exhibit some established social skills. He looked around at all the blank faces and started to realize that perhaps he had left out some important details. "Wait! I brought along a little toy to liven things up. Do you mind if I set it up?"

Before I could say anything, Ed bolted for his car with a few semi-willing conscripted volunteers. I wiped my forehead which was now sweating profusely from stress and turned toward the house to make another drink.

I took my time in the kitchen eating a few spicy hors d'oeuvres that might replace the growing throb in the back of my head with a more manageable case of indigestion. When I finally stepped outside I found myself pressed back toward the house by a group of guests who were backing up en masse. Gently elbowing my way to the front of the throng, I could clearly see Ed Nisley crouched beside a strange contraption holding a rope as if he were about to fire a cannon.



A closer examination revealed that it was a gantry and rocket launcher! Or, sort of a launcher. There was a base "launch pad" with iron guides that appeared to be a gantry and a hose running from the launcher to a metal cylinder that looked like the propane bottle from a barbecue. As my eyes went back to the launcher, I realized that the launch vehicle was the plastic soda bottle!

"Hey, Steve. Watch this!" Before I could jump back, protest, or take cover, Ed yanked the lanyard. "Whomp!!!!" Twenty-five people stood there (actually, some ducked) with their mouths open as this "bottle rocket" blasted off like it was shot out of a cannon.

I barely had time for the shock to wear off when somebody in the crowd yelled, "Heads Up!" Obviously, what goes up must come down and everyone scurried for cover as the plastic bottle which had been blown almost completely out of sight suddenly appeared to be on a direct trajectory for us. Fortunately, they take off with a lot more velocity than they fall. I backed up flat against the side of the house so that I was protected by the overhanging roof.

The "bottle rocket" finally landed. It missed the edge of the roof and bagged a perfect bullseye on a large hanging plant spraying dirt in all directions. As luck would have it, I was the one closest to the plant. Everything I'd eat or drink for the rest of the afternoon was now destined to taste like Miracle-Gro.

Ed looked at me. I took my time shaking the dirt out of my beard and pouring the freshly made Martini into the bushes. The people on the deck were silently waiting for my next move.

Before Ed could say anything I picked up the spent soda bottle rocket and walked toward him. As I reached my hand out to grab the lanyard from him he jumped back

Photo Inset: Ed Nisley demonstrating his bottle rocket.

as if to stay out of range. Had this been the wrong "toy" to bring to a Ciarcia party?

I knew I had him and I could have carried on the charade longer, but there wasn't any use rubbing it in. When I saw the blastoff, I recognized instantly that this was a grown-up version of the compressed air and water hand-held rockets that I used to love playing with as a kid. I grabbed the lanyard and yelled, "All Right!!! What a shot! Let's do it again!"

Ed, of course, breathed a sigh of relief. His opinion of my sense of humor had been formed by reading Circuit Cellar for many years and he had not guessed wrong. We were birds of a feather and instant friends. As for how the party turned out, we emptied the cooler into the bushes and had a great time all afternoon.

Ballistically Speaking

While our experiences leading up to that point were humorous, it eventually evolved into something more serious. That many technical minds can't witness something like this without asking questions or wanting to get involved. You actually have to see one of these things blasting off to appreciate the speed. We all were interested in the scientific principles and realities of these projectiles. How much pressure was in the bottle? What was the take-off speed and what were the G-forces?

With a few more drinks and some quick calculations on a place mat, the Circuit Cellar Ballistic Missile Research Division was born. Its charter was to analyze this compressed-air missile system.

The first thing we did was to check out Ed's launcher.

It turns out that 1/2-, l-, and 2-liter plastic soda bottles (airframes) share a common opening diameter (nozzle) and cap. Launching one merely consisted of filling the inverted bottle with compressed air and quickly releasing it. Holding it against the pressure and instantly releasing it is no easy task, however.

Ed had two different launchers with him. One used a lever-arm clamp around the neck ring on the bottle (Photo 1). The clamps were adapted from a set of bicycle brakes. The hand lever pulls the clamps down against the neck ring and pushes the nozzle against a rubber seal in the baseplate.

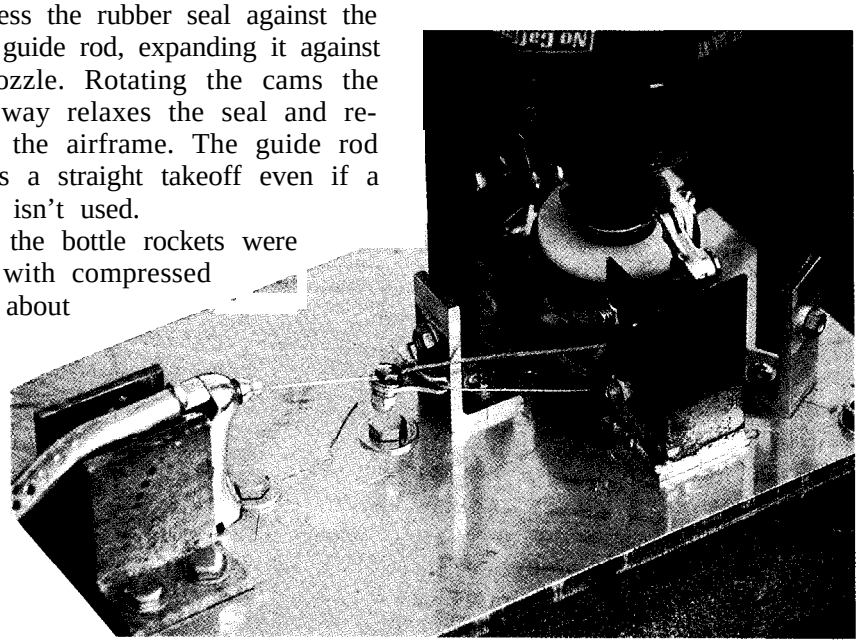
Another used an expanding rubber seal to grip the inside of the airframe's nozzle (Photo 2). A pair of cams pushes a metal plate upward to comoress the rubber seal against the nylon guide rod, expanding it against the nozzle. Rotating the cams the other way relaxes the seal and releases the airframe. The guide rod ensures a straight takeoff even if a gantry isn't used.

All the bottle rockets were filled with compressed air at about 60-100 psi. It's hard to convey just how exciting a full-power missile launch is. It starts with a satisfying BLAM! that puts an airframe a few hundred feet in the air in a fraction of a second. Next, there is a search for the airframe silhouetted against a clear blue sky, and finally finishes with a tightened stomach when you realize that the launcher is much too close to the parking lot.

An extensive series of evaluations conclusively demonstrated the superiority of the expanding rubber seal launcher. Although the clamp launcher looks wonderful and is a triumph of the machinist's art, it simply could not hold the airframes against the nozzle seal with enough force to prevent leaks.

At the end of the launcher evaluation tests, we had a dozen airframes scattered around the Circuit Cellar Missile Test Range, many lost forever in trees and bushes. We retired to the Bar-B-Q and restarted the airframe production line. The time was ripe for a study of launch parameters.

Photo 1 - Level-arm launcher



Instrumentation

Designing and building the airframes and launchers was easy enough, but we wanted to know just how these missiles performed. It's not enough to be awed by a good launch; we needed some hard numbers by which to measure and evaluate our options.

All of us were interested in the

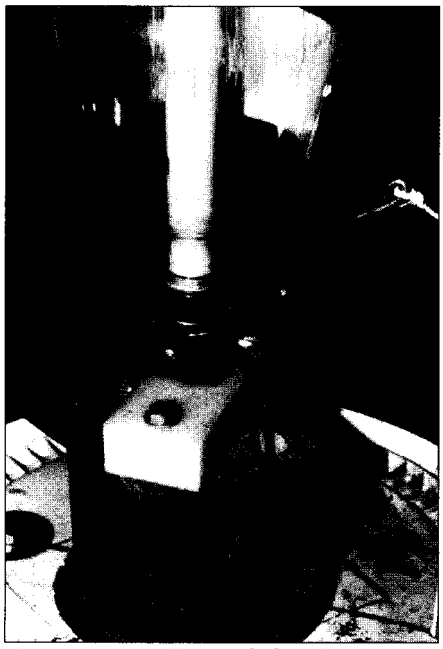


Photo 2 - Expanding seal launcher

airframe's launching velocity and acceleration. Both of those values can be computed from the time required to travel a known distance during the launch.

The first test flights verified that the missiles travelled "mighty fast" but we needed a better estimate than that to decide how to time the flights. We fished out the place mat that started the project and made a few more computations.

Newton's Second Law of Motion provided the basis for the calculations: the force required to move an object is given by the product of its mass and the acceleration imposed on it. We can measure the mass of the airframes easily enough, and the force imposed on it is determined by the propellant pressure. The resulting acceleration is simply the launch force divided by the airframe mass. Of course, all these values have to be expressed in the appropriate units for the answer to mean anything.

We measured several airframes and found that the 2-liter versions weighed about 2.0 ounces, or 0.125 pounds. The neck diameter was 0.85 inches, giving a nozzle area of 0.57 square inches.

When an airframe is pressurized to 100 psi there are 57 pounds (100 pounds per square inch x 0.57 square inches) pressing down on the launcher through the nozzle. This force will decrease as the propellant expands through the nozzle during flight, until finally the airframe is in free-fall near the top of its trajectory. We used 57 pounds for the thrust throughout the gantry, even though we knew that it would give optimistic results.

Figure 1 shows the calculations for a 2-liter launch. A 57-pound thrust applied to a two-ounce bottle produces an initial acceleration of 470 times the force of gravity! Contrast that with the Space Shuttle's leisurely 3 G...it's a good thing the Circuit Cellar Ballistic Missile won't be used for manned flights!

Using that estimate, we can determine the airframe's velocity at the end of the 3-foot launch gantry shown in Photo 3, as well as the amount of time it takes to get there. Figure 2 shows these calculations, with the surprising results that the airframe reaches the end of the gantry in 20 milliseconds, travelling at 300 feet per second.

Three hundred feet per second is about 200 miles per hour! Now that's performance!

Even though the airframes may not move that fast for very long, it's enough to rule out some of the simpler ways to find the airframe's position along the gantry. We originally thought of locating switches along the gantry, but you can imagine what happens when an airframe hits a switch at 200 mph!

Ed came up with the system shown in Photo 4. A pair of connectors holds a wire across the center of the missile's path. The tip of the airframe pulls the wires out of the sockets, breaking the circuit. The connectors are made from cut-

Newton's Second Law: acceleration = force / mass

Airframe mass = 2.0 ounces
= 0.125 pounds mass

Airframe nozzle diameter = 0.85 inches
area = 0.57 inches^2

Launching pressure force = 100 pounds force / inch^2
= 57 pounds force

Conversion between pounds force and pounds mass:

$$1 \text{ lbf} = (1 \text{ lbm}) / (32.2 \text{ lbm-ft/lbf-sec}^2)$$

$$\text{So acceleration} = \frac{57 \text{ lbf}}{(0.125 \text{ lbm}) / (32.2 \text{ lbm-ft/lbf-sec}^2)}$$

$$= 15{,}000 \text{ ft/sec}^2$$

Conversion between ft/sec^2 and sea-level force of gravity:

$$\text{accel in "G"} = \frac{\text{accel in ft/sec}^2}{32.2 \text{ (ft/sec}^2\text{)/G}}$$

so launch acceleration = 15000 / 32.2
= 470 G

Figure 1 - Initial Launch Acceleration

down wire-wrap IC sockets, with three contacts at each location.

These sensors were simple enough that eight were installed along the length of the gantry. The first is at the bottom, just above the nose of a 2-liter airframe secured in the launcher. The remainder are at four-inch intervals starting nine inches above the bottom sensor. Timing starts when the bottom sensor is tripped by the first inch of the airframe's flight.

Schematic 1 shows the gantry sensor electronics. Each sensor wire holds one input of a 74LS244 bus buffer at electrical ground. The buffer's input goes high when the airframe yanks the wire free. A red LED connected to each buffer gives a visual indication that all the wires are in place before launching.

As you might expect, those eight sensor outputs are cabled to a single input byte on a computer. The computer records the time when each bit goes high and computes the velocity and acceleration from those times. The only problem is finding an IBM PC with a parallel input port (Ed wasn't fond of converting over to another computer).

The IBM PS/2 printer port can be used as a bidirectional parallel

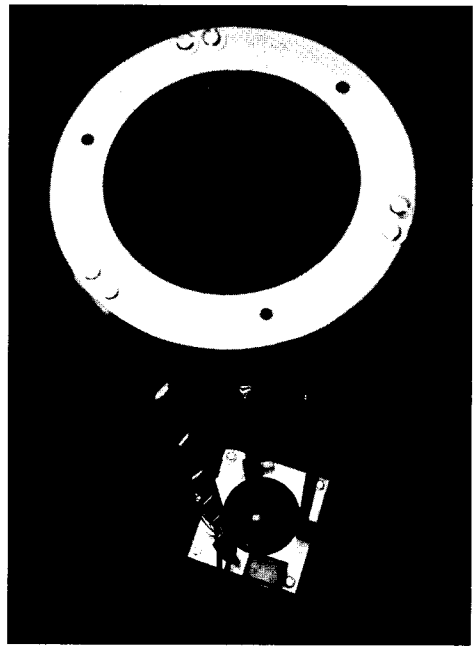


Photo 3 - Instrumented launch gantry

port (it can both read and write data). The original IBM PC and AT printer ports can only write data to the outside world and are thus useless for input.

After he pored over the schematics for a while, Ed determined that a one-wire change to a standard IBM PC printer port would convert it into a bidirectional data port. That's more along the lines I was expecting, so I authorized the butchery posthaste (it was his machine).

Schematic 2 shows the required changes for standard IBM Parallel Printer Adapters. The Monochrome Display Adapter uses the same circuitry, so you can modify it using the same method. However, there are enough different variations of the basic circuits around that the pin numbers shown on the ICs are probably irrelevant.

Essentially, the change connects an unused bit in the control port to the output enable line of the data buffers. The enable line must be disconnected from ground, which may require trimming the pin off if it is soldered to an internal ground plane (Many of the newer multifunction cards use ASIC [Application Specific Integrated Circuit] chips to implement the parallel port function, so there's no way you can "get inside" them to make the change. If you trace your parallel port's data lines back to a large IC that doesn't look like a bus buffer, you're out of luck). Most software should be unaffected by the change.

Another caveat is worth mentioning. If you simply cable the gantry sensor buffers to the printer port, you'll be wiring a pair of bus buffer outputs together. The cable should be connected only after the PC program has disabled the

For uniform acceleration from a standing start,

$$\text{time} = \frac{\text{Sqrt}(2 \times \text{acceleration} \times \text{distance})}{\text{acceleration}}$$

$$\text{velocity} = \text{Sqrt}(2 \times \text{acceleration} \times \text{distance})$$

For a 3 foot gantry,

$$\text{time} = \frac{\text{Sqrt}(2 \times 15000 \text{ ft/second}^2 \times 3 \text{ feet})}{15000 \text{ ft/second}^2}$$

$$= 20 \text{ milliseconds}$$

$$\text{velocity} = \text{Sqrt}(2 \times 15000 \text{ ft/second}^2 \times 3 \text{ feet})$$

$$= 300 \text{ feet/second}$$

Figure 2 - Launch Timing and Velocity

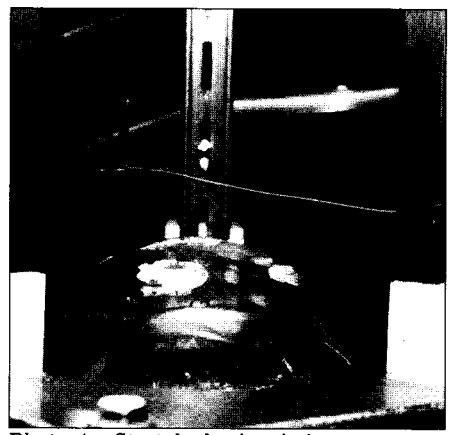
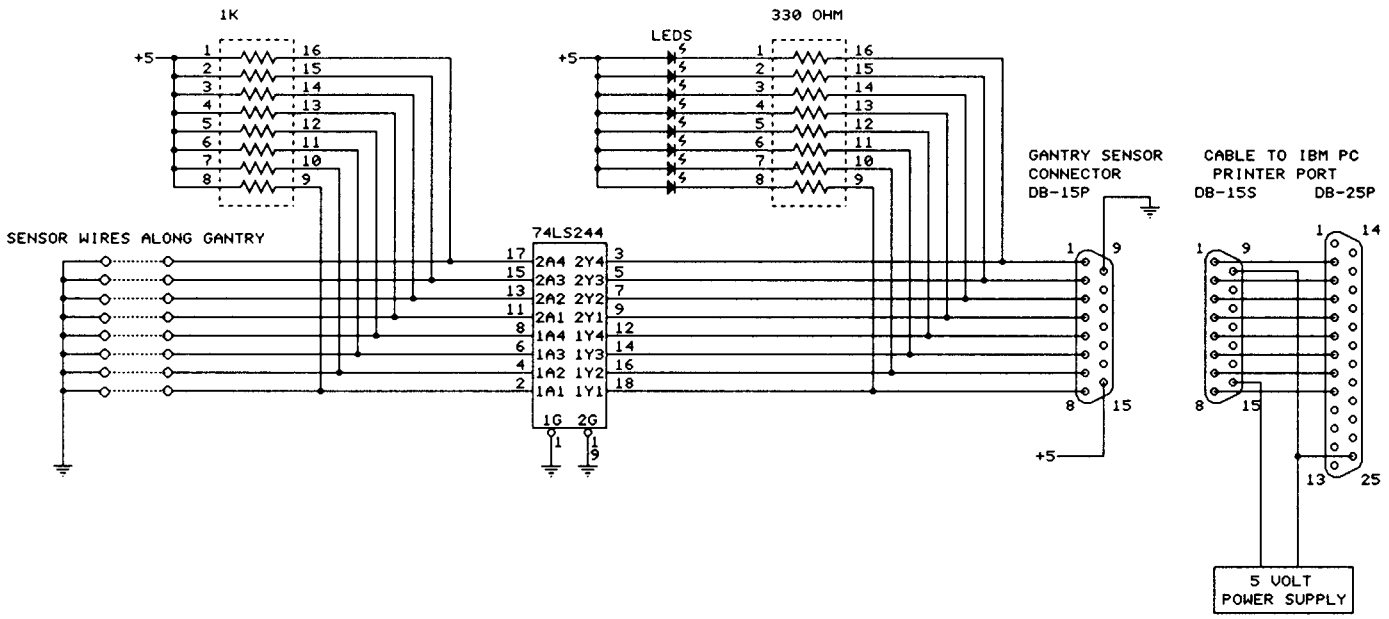
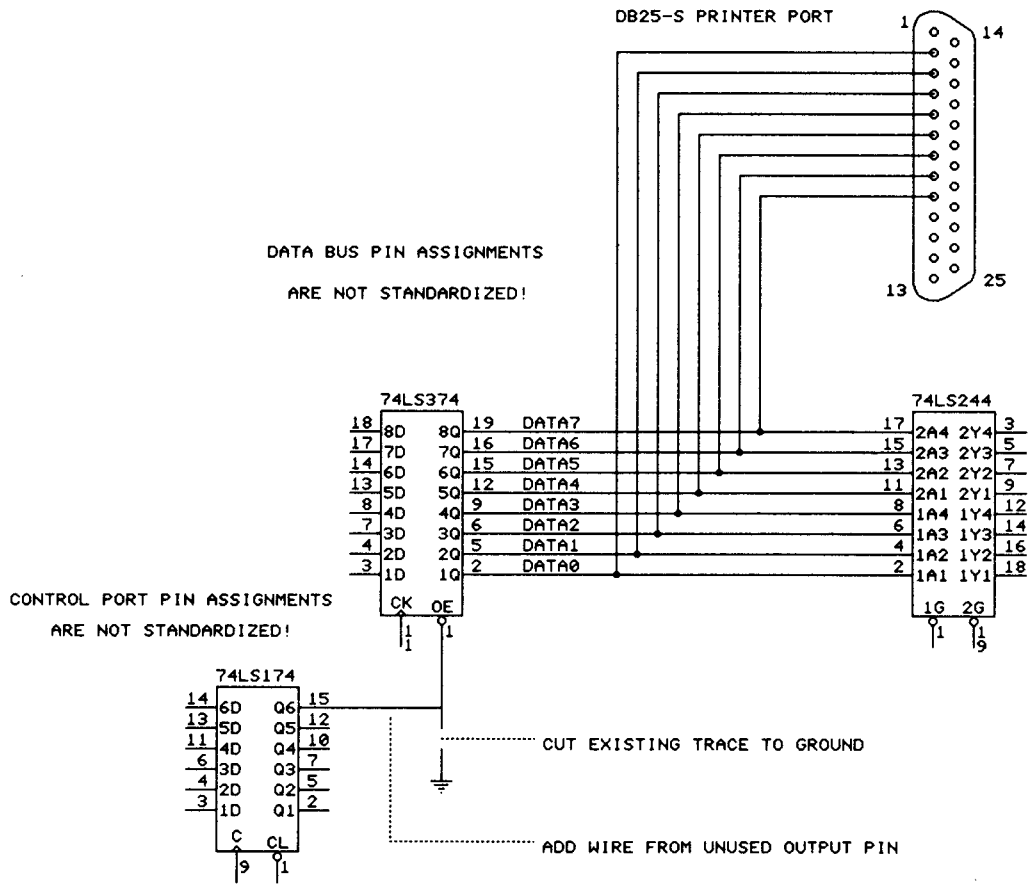


Photo 4 - Stretched-wire timing sensor



Schematic 1 - CCBM Gantry Sensors



Schematic 2 - Printer Port Modifications

printer port's buffer!

Precision Launch Timing

The IBM PC uses an Intel 8253 programmable interval timer to provide three hardware timers. A common 1.19-MHz clock input derived from a 14.3-MHz crystal oscillator provides a fundamental "tick" of 840 ns. The timers are programmed by the BIOS when the PC is turned on and rarely changed by ordinary programs.

The timers count down from a prestored maximum value to zero. When the count hits zero, a hardware output signal becomes active, the maximum count is reloaded, and the timer continues counting. Setting the maximum value defines the rate at which the output signal is generated. The timers can also be programmed to stop counting when they reach zero, but this mode is not usually used in the PC.

Each of the three timers is assigned to a different task in the PC. Software measures time using Timer 0, which generates interrupts 18.2 times a second. The BIOS counts the interrupts and updates a time-of-day value in RAM that's used whenever a program needs the current "wall clock" time. A resolution of 1/18 second is enough for normal timing, but 54.9 milliseconds is a rather long time compared to a 20-millisecond launch. An airframe can completely vanish between two clock ticks!

Timer 1 generates the signal used by the RAM refresh DMA channel. It's set to produce an output every 15.1 microseconds, which is a little too often for our purposes. Because PS/2 system boards use a different method to generate RAM refreshes they do not implement Timer 1 at all. Messing with the system RAM refresh rate is fraught with peril, so we didn't even consider using Timer 1.

Timer 2 is used to generate the beeps and boops you hear from your PC's speaker and (in the original PC, at least) it was the basis of the cassette tape interface. Although it's tempting to use Timer 2 for the launch instrumentation, there's one fatal flaw: the output cannot generate an interrupt, so there's no way to tell when the timer has "hit bottom" and recycled to the maximum value.

Tricky Ed to the Rescue

Although none of the timers are quite what we needed, Ed came up with a tricky solution for this obstacle.

A program can read the state of the 8253's counting registers at any time. Because the timers count down, the value read back equals the number of ticks until zero. This is easily converted to the number of counts since the previous zero. This is exactly what's needed for better timing resolution.

Recall that when Timer 0 hits zero, the BIOS code updates the time-of-day value in RAM. That value provides the time to the nearest 54.9 milliseconds, and the count read for the whole cycle. from Timer 0 provides the exact time with microsecond resolution.

The BIOS programs Timer 0 into "Mode 3" to generate a square wave output signal. In this mode the count is decremented by two instead of one for each input clock pulse. When the count reaches zero the output signal is inverted and the maximum count reloaded. The effect is a square wave signal with a period set by the maximum count.

Because interrupts are generated only on the rising edge of the output signal, the timer goes through two complete counting sequences for each interrupt. Therefore, you can't tell the exact time just by reading the current count value. There is a way to read the actual state of the output signal using the 8259 Programmable Interrupt Controller, but it's hardly worth the bother.

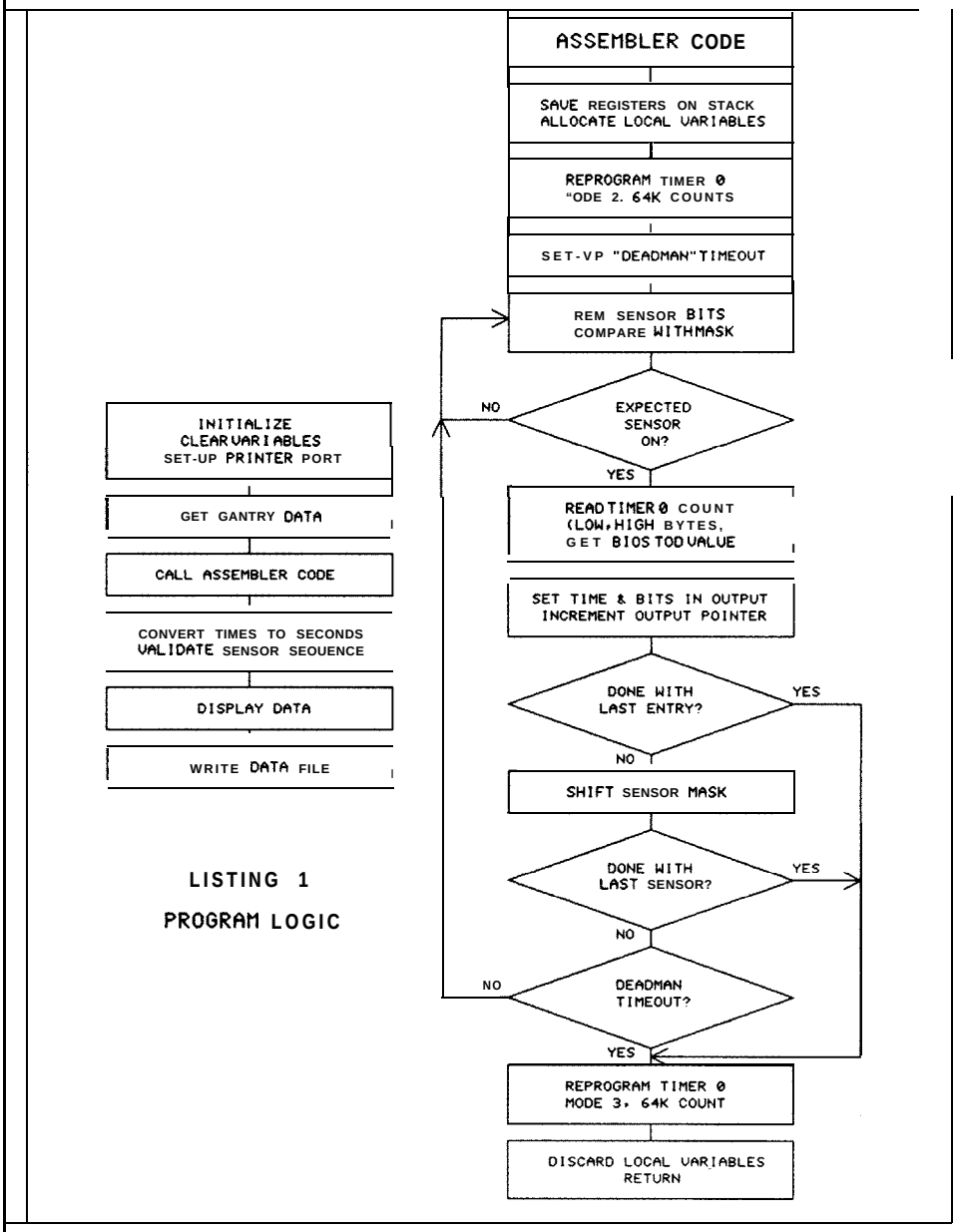
An adventurous programmer can reprogram Timer 0 to operate in Mode 2, which produces a narrow pulse instead of a square wave. The interrupt controller doesn't care how wide the pulse is, so a pulse will work as well as a square wave. The only restriction is that the CPU (8088, 80286, 80386, what have you) must respond before the signal becomes inactive. Fortunately, because we aren't running much else concurrently, this isn't a problem here.

The only remaining task is to calculate the maximum count value that will produce an output pulse every 54.9 milliseconds. This is simply the ratio of the desired output period and the input period: 54.9 ms / 840 ns, which is 65536 or 64K. For the purists, the actual values are 18.2065 Hz and 838.095 ns.

The 8253 uses the value 0 to represent 64K. In effect, the timer reloads a zero when it reaches zero! The next count is 65535, and so on

With precise timing in hand, it was easy enough for Ed to throw together a program that reworks Timer 0 and records times to the nearest microsecond or so when a sensor changes state. Listing 1 shows a flowchart of the launch timing programs. The main program is written in Turbo Pascal, with an assembly language routine to handle the fast timing loop. Listing 2 shows the core of that assembly code.

The time required to complete one loop determines the actual timing resolution. For example, if a sensor changes just after the program has read the input port, the



LISTING 1

PROGRAM LOGIC

program must travel around the whole loop before it will find the change on the next port read. A stock IBM PC running at 4.77 MHz takes about 15 microseconds to complete one "idle" loop, so that's the true resolution.

The sensors sometimes glitch during launches (remember that 200-mph projectile ripping through the gantry!) so the code waits for each sensor to be activated in turn. If a sensor glitches open momentarily the code will ignore it.

The program records the state of the gantry sensors and the time at which each sensor changes. After the last sensor wire has vanished the program sanity-checks the results and displays them on the screen. If everything looks OK, it creates a file containing the appropriate data for later analysis.

Although we'd like to pretend that the Circuit Cellar Ballistic Missile Research Division has only successful launches, that's simply not true. The assembly code includes an escape clause that times out after about 20 seconds of no gantry action. This prevents a computer hangup if a launch fails. The code implements this by adding 20 seconds to the BIOS time-of-day value at the start, then checking the current time on each loop.

Test Flights

With the gantry wired up and an IBM PC converted to work as a data logger, we were faced with the task of validating our assumptions and Ed's code. I insisted that the program should generate reasonable numbers before we made claims about actual launches.

Ed checked his software by filling an airframe with water and dropping it through the gantry (holding it upside down for the purpose). The measured times (shown in Figure 3) matched the predicted values closely enough to convince me that we knew what we were up to, so we made a series of instrumented test launches.

Ed imported the gantry timing program's output files into a series of Lotus spreadsheets to compute the velocities and acceleration from the distance and measured times. If you remember that velocity is just distance divided by time, and acceleration is velocity divided by time, you'll have no trouble following the formulas shown in Figure 3. The spreadsheets simply apply these equations to the data obtained from the timing program.

However, the example used in Figure 3 shows that you can't take the exact numbers too seriously. The average acceleration between sensors 2 and 4 is 3 1 ft/sec^2 instead of the 32.2 ft/sec^2 you'd expect from high-school physics. Differ-

Listing 2 -- Core of assembly language timing routine
<<< code precedes this section >>>

```
;-----------------
; Reprogram timer for mode 2 operation
; This sets 64K counts in each BIOS TOD tick
            cli                     ; ensure no interrupts!

            mov     al,mode2cmd
            out     ctrlport,al
            punt

            mov     al,0            ; set up 64K count
            out     timeport,al
            punt
            out     timeport,al
;-----------------
; Set up timeout value to avoid hangup if unplugged

            mov     ax,TODlsw       ; grab data
            add     ax,maxtime      ; add timeout value
            mov     [bp].endtime,ax ; save for later

            sti
;-----------------
; Run the timing loop until all sensors change or we give up
; Sensors are '0' until rocket passes
;  BH = mask to select next sensor bit expected to change
;  BL = temp storage for input bits
;   CL= filled entry counter
;  DX = printer port
;  DI = ptr to output data array

            mov     bh,sensor1      ; sensor change mask
            mov     cl,nvals        ; number of sensors
            mov     dx,[bp].prtport ; set up port address

looper      label   near

            in      al,dx           ; get sensor bits
            mov     bl,al           ; save 'em for later
            test    al,bh           ; right one tripped?
            jz      timecheck       ;  = 0 -> none

;--- got a new sensor change, grab current time

            cli                     ; ensure no interrupts

            mov     al,latchcmd     ; latch 16 timer bits
            out     ctrlport,al
            punt
            in      al,timeport     ;  ... lsb
            xchg    al,ah           ;
            punt                    ; delay a bit
            in      al,timepo*      ;  ... msb
            xchg    al,ah           ;

            mov     si,TODlsw       ; get millisecond count

            sti                     , interrupts OK again
;--- save value in output record for each new sensor

            not     ax              ; get up count
            mov     es:[di].lotime,ax

            mov     es:[di].hitime,si
            mov     es:[di].bits,bl ; store raw bits
            mov     es:[di].valid,true; note good data

            add     di,TYPE  results  ; tick ptr by 1 entry
```

(continued on page 14)

ent pairs of sensors give different results, sometimes much farther off than this sample.

The cause is easy to understand. As the airframe passes through the gantry some wires stretch or bend before they're pulled from the contacts. The break time recorded for that wire is late, making the time interval "below" that sensor too long and the one "above" too short. The velocities will be too low and high, respectively, and the computed acceleration will exaggerate the error because it depends on the incorrect times, too! In effect, that one bad time skews the acceleration formula in four places: both velocities and both time values.

As you can see from the graph of Figure 4, though, the basic velocities are quite repeatable despite any problems with individual wires. The curves show the velocity at each sensor plotted against the time required to get to that sensor for six flights, each with the same starting pressure. Apart from two obvious glitches in the first and second flights the results are very stable.

Figure 5 shows the results of several 2-liter launches at various pressures. The Research Division noted that the last six velocities lie in a more-or-less straight line, so they "eyeballed" a straight line to reduce the inaccuracies in each individual measurement. (Using Lotus 123's least-squares function agreed quite well with our eyeballs!) The line slope gives the average airframe acceleration through that part of the gantry. The same procedure can be applied to the first few points to get an idea of the initial acceleration, albeit with somewhat less accuracy.

Test Results

The curves in Figure 5 illustrate the (not too surprising) fact that increasing the launch pressure increases both the acceleration and velocity through the gantry. For 2-liter airframes, a 40-psi launch gives a 50-G acceleration and a final velocity of 100 ft/sec; increasing the pressure to 95 psi boosts it to 270 G and 220 ft/sec.

The initial accelerations measured at the first three sensors range from 150 G at 40 psi to 380 G at 95 psi. The back-of-the-envelope prediction was 470 G at 100 psi, which translates into 450 G at 95 psi. Because the envelope had nothing about air resistance, friction along the gantry, or propellant pressure drop during launch, I'd say that the results are in excellent agreement with the prediction!

A second test launch series compared the three airframe sizes at a constant 60-psi launch pressure. The results are summarized in Figure 6. The i/2-liter airframe gets up to 100 mph faster, then noodles along at a mere 12 G because it's literally run out of gas. The 2-liter airframes are considerably slower because they weigh more.

Surprisingly, the 1- and 2-liter airframes are still accelerating at the end of the gantry, so the velocities shown are not maximums! An informal series of static tests indicated that 2-liter airframes require about 250 milliseconds to exhaust their fuel supply. Allowing for thrust reductions, the ultimate velocity may exceed 300 mph!

Conclusions

We considered ways to measure both the altitude and velocity of the airframes in free flight, but de-

(continued from page 13)

```
                dec     cl                      ; tick entry counter
                jz      goback                  ; and exit if done
                if      direction
                shr     bh,l                    ; move to next sensor
                else
                shl     bh,l
                endif
                cmp     bh,0                    ; if all zero, done!
                jz      goback
                jmp     short  looper           ; else keep on checking

  ;--- check for overall timeout

  timecheck     label   near

                mov     ax,TOD1sw               ; get current TOD value
                cmp     ax,[bp].endtime         ; hit timeout value?
                jb      looper                  ;  no, repeat again
  ;----------------
  ; end of timing loop, return to caller
  goback        label   near

  <<< more code follows >>>
```

Figure 3 -- Velocity and Acceleration Computations

$$\text{Velocity} = \frac{\text{change in distance}}{\text{change in time}}$$

$$\text{Acceleration} = \frac{\text{change in velocity}}{\text{change in time}}$$

Sample sensor data from a drop test run

Sensor	Distance	Time
0	0.000	0.0000
1	0.750	0.1596
2	1.083	0.2020
3	1.417	0.2376
4	1.750	0.2706
5	2.083	0.3014
6	2.417	0.3303
7	2.750	0.3502

Average velocity between sensors 1 and 2:

$$\text{velocity} = \frac{1.083 - 0.750 \text{ feet}}{0.2020 - 0.1596 \text{ second}}$$

$$= 7.9 \text{ ft/sec}$$

Average velocity between sensors 3 and 4:

$$\text{velocity} = \frac{1.750 - 1.417 \text{ feet}}{0.2706 - 0.2376 \text{ second}}$$

$$= 10 \text{ ft/sec}$$

Average acceleration between sensors 2 and 4:

starting velocity = 7.9 ft/sec at 0.2020 sec
ending velocity = 10 ft/sec at 0.2706 sec

$$\text{acceleration} = \frac{10.0 - 7.9 \text{ ft/sec}}{0.2706 - 0.2020 \text{ sec}}$$

$$= 31 \text{ ft/sec}^2$$

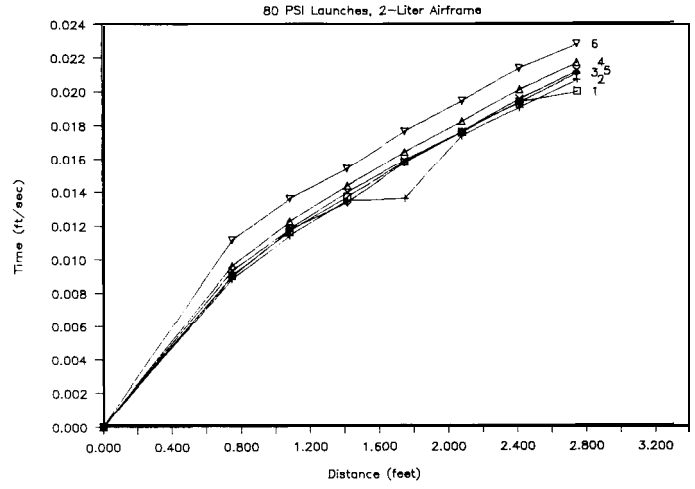


Figure 4

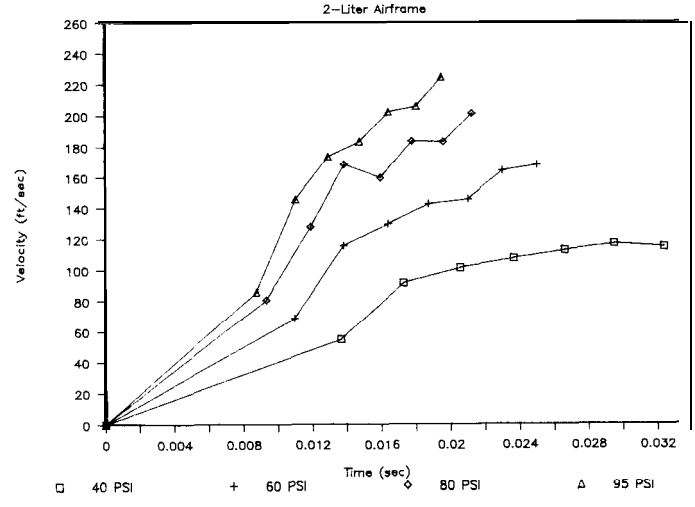


Figure 5

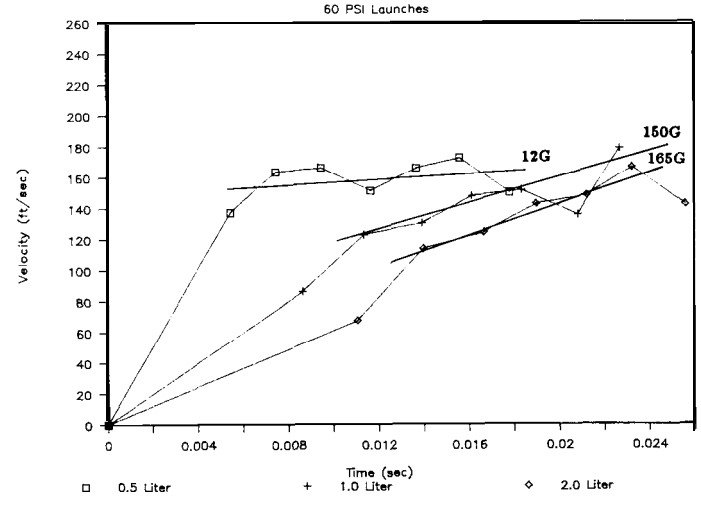


Figure 6

cided that this is a tough problem and would probably take another picnic. If any of you decide to install a CCBM system at your house, you may be able to solve the problems before we do. If so, we'd like to hear from you!

A particularly interesting research project would investigate the effect of adding water to the propellant mix. It seems that the missile should go faster because it's expelling more mass out the rear. Or would it be slower because the airframe weighs more? You should take some waterproofing precautions. Our preliminary trials indicate a significant overspray problem!

And, of course, anyone figuring out how to add a payload to one of these screamers should be in a good position in the neighborhood arms race.

Finally, this is no April fool's article and a word of warning is in order. A 200-mph projectile can cause significant damage despite the fact that it's "only" a plastic bottle. Keep your hands and faces clear during launches and use it only in unpopulated areas. Dropping a bottle on your neighbor's Porsche from 200 feet straight up may strain diplomatic relations to the breaking point.

[All the software used for this project is available for download from the Circuit Cellar BBS.] ■

WRITE FOR INK!

Writing technical articles may not make you rich and famour but it might be just what you need as incentive to finish that 100-MIPS computer you started last summer. Or, if your expertise is software, perhape it's time you presented your talents to the world.

Unlike most narrowly specialized publi-/cations, Circuit Cellar INK's charter is to /cover a wide variety of hardware and software technology and ideas.

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.

VISIBLE INK

Answers; Clear and Simple

Letters to the INK Research Staff

Dear INK,

I recently saw a neat CAD drawing system for the IBM PC. The system allows you to create great screen graphics easily. The screen images are stored in data files in DXF format. I would like to be able to display these files in my Turbo Pascal programs.

Can you recommend a good source of information on the DXF format, and writing or acquiring the necessary Turbo Pascal code?

John Panagacos-East Williston, NJ

Dear John,

As near as I can tell, files ending in DXF are drawing files used by AutoDesk's AutoCAD CAD program. Because the file extensions are free for the naming, two vendors may (and have!) used the same extension for completely different types of files, but I'm pretty sure that's what you've got.

I'm also sure that it's not in the normal CGA or EGA bit-image format supported by most drawing programs. CAD files contain a great deal of in formation in addition to the simple graphics primitives: a typical entry might be "there is a double- width broken line in color I from (1.25,2.50) to (4.5,10.13) with break spacing .05", with other entries filling in all the details of color and width and so forth.

As you can see, decoding this into a picture might turn into something of a project. I don't know where you'd find source code for a file decoder, but a call (or letter) to AutoDesk might produce some interesting results on the file format. They're at:

AutoDesk
2320 Marinship Way
Sausalito, CA 94965
(800) 433-0100 or (415) 332-2344

Good luck!
-- INK

Dear INK,

I have several questions about my Tandy 1000A which came with an NEC CPU chip (I assume it's a V20).

1. How can I find out which model of CPU it is? I think it's a 5-MHz model. The chip has this label on it: "NEC JAPAN D8088D8433KX"
2. Will an 8- or IO-MHz model work, and if so how much speed improvement can I expect?
3. The unit has an 8087 math chip socket. Should I use a regular 8087, or can I use a 8087-3, or 8087-2? Also should the CPU chip and the math chip speed be the same?
4. Drive A is very noisy when turning. I have a drive B and it is quiet. The noisy operation is very annoying. I tried to switch the two drives on the logic that I use drive A more often than B, but the system still went to the original drive A to boot. What can I do?

I would be thankful for any information and/or advice you can give me.

Mark Danner-Sherman, Texas

Dear Mark,

I believe the chip you describe is actually an NEC 8088, which is the same as the Intel 8088, not a V20. The V20 is also a di jjerent design using CMOS, rather than the 8088 which uses NMOS. Besides the part number "8088" another clue is the date code which indicates your chip was made in the 33rd week of 1984 -- over three years ago, before the V20 was available.

Also, if you haven't heard, Intel and NEC have been embroiled in a major legal battle over the "V" series -- Intel claims NEC has violated a "microcode copy-right." The outcome remains to be seen.

Finally, we get many questions about "speed up" for PCs. Personally, we would only consider a speed up based on a well-designed "kit" or "board" from a well-known company. Check PC magazines, user groups

and/or Tandy to see if someone makes a "turbo" setup. I do not recommend a "homebrew" approach: swapping chips, cutting traces, soldering, and so on. It is just too easy to get into real trouble.-

As for the 8087 issue, a well-designed "turbo" option must include provision for a fast 8087. However, if you don't elect to speed up, you don't ***need a faster*** *8087 either.*

Though not familiar with the specific disk/cable setup of the 1000A, certainly drive A and B can be switched -- unless Tandy has ESP built into its computer! You need to figure out which cable, disk drive jumper, or switch will do the job.

--INK

Dear INK,

I desperately need your help! I recently purchased a Centronics Model 306 (C) printer. I am trying to confirm which signals go on which of the 44 wires that form a ribbon cable assembly into the unit (22 twisted pairs). I've called all over Omaha and no one knows anything about the Centronics Model 306. Centronics has gone out of business so I can't turn to them. This wiring info is so simple but I can't determine a source. Could you please send me an address of some individual or company which might still have such information.

Bill Cashman-Omaha, Nebraska

PARALLEL INTERFACE PIN FUNCTIONS

Pin No.	Signal Name	Function
1	STROBE	Signals when data is ready to be read. Signal goes from HIGH to LOU (for at least 0.5 micro seconds when data is available.
2	DATA1	These DATA signals provide the
3	DATA2	information of the first to eighth
4	DATA3	bits ofparallel data. Each signal
5	DATA4	is at HIGH level for a logical 1
6	DATA5	and at a LOU level for a logical 0.
7	DATA6	
8	DATA7	
9	DATA8	
10	ACK	A 9-microsecond LOU pulse acknowledges receipt of data.
11	BUSY	When this signal goes LOW the printer is ready to accept data.
12	PAPER OUT	This signal is normally LOU. It goes HIGH if the printer runs out of paper.
13	SELECTED	This signal is HIGH when the printer is on-line.
14-15	N/C	Unused.
16	SIGNAL GND	Signal ground.
17	CHASSIS GND	Printer chassis ground.
18	+5VDC	External supply of +5VDC from the printer.
19-30	GND	Twisted-pair return-signal ground level.
31	RESET	Uhen this signal goes LOU the printer is reset to its power-on condition.
32	ERROR	This signal is normally HIGH. It goes LOU to signal that the printer cannot print due to some error condition.
33	EXT GND	External ground.
34-36	N/C	Unused.

Dear Bill,

The Centronics parallel interface is the defacto "standard" for many of the popular printers available for today's microcomputers. It is designed to transfer information to the printer electronics 8 bits at a time and contains not only the data lines but several handshaking signals as well.

The attached list gives the pin numbers of an Amphenol-type 57-30360 connector, which is the common 36-pin connector used on most parallel printers, and a brief description of what each line is used for. Keep in mind that not all computers will use every handshaking signal available: some such as the SB180 use only the STROBE and ACK lines and will perform quite satisfactorily.

Thanks for writing. I hope this information will cure your "no output device" blues.

Dear INK,

I recently sold my 6502C-based personal computer. Since then I've been in the market for an IBM AT compatible. One in particular that has attracted my attention is the Jameco JE1003 AT motherboard. My questions are:

1. Just how "compatible" is this AT?
2. Can an AT run PC and XT software?

I would appreciate any help or sources you could direct me to.

Stefan R. Kelley-Havelock, NC

Dear Stefan,

Almost all of the XT and AT clones available now are 99 and 44/100% compatible with IBM -- including Jameco. Some early 8088 machines (such as Sanyo and

Zenith) were not very compatible, but the industry has learned its lesson: full IBM compatibility is a must.

There are very few incompatible programs now. Those that are usually have a very specific form of copy protection which does not allow them to run at high speeds. An AT compatible will run all PC and XT software, although some games may become unplayable due to the higher speeds. Some machines can be switched to a lower speed to eliminate this problem.

-- INK

Dear INK,

I am considering building a Mobil Weather Station Monitor (MWSM) using the BCC-52 computer/controller. This monitor would collect, process, and display weather conditions like temperature, humidity, pressure, and elevation. Can you suggest ways to electronically measure and record these parameters?

Orlando A. Moreno-Stockton, CA

Dear Orlando,

The BCC-52 controller would lend itself well to your Mobile Weather Station Monitor project. Many of the atmospheric conditions can be sensed by transducers, converted to digital information, and processed by a controller. Here are some of the sensors and transducers you might consider:

Temperature measurement: National Semiconductor's LM135-series precision temperature transducers.

Barometric pressure: National's L X05xxA series Monolithic Pressure transducers.

Elevation: same as Barometric pressure.

Wind direction and speed: Hall-effect sensors and permanent magnet.

The transducers will need to be processed in an analog environment, then converted to digital information with an A/D circuit. The Hall-effect sensors can be used digitally for count or direction.

--INK

Dear INK,

I am interested in the possible deleterious effects that populations of cockroaches may inflict on computers and electronic equipment over time.

Deborah A. Gust, Ph.D-Atlanta, GA

Dear Deborah,

Your interest in the effects of cockroaches on electronic equipment is somewhat unusual. I've seen an occasional dead roach inside equipment -- older tube-type amplifiers are quite adept at cooking stray insect visitors -- but I am unaware of any formal studies on the matter.

While purely speculative, there are some obvious lines of inquiry to pursue regarding possible effects. Our initial speculations are summarized below:

1. chemical contamination -- possible corrosive effects of insect secretions or substances transported by the insects such as insecticides, food particles, and so on.

2. fecal contamination -- possible corrosive effects of roach feces ("roach dirt").

3. component damage due to chewing by roaches, short circuits resulting from insulation being eaten from wires, and so on.

4. thermal effects due to insulating properties of roach feces and carcasses -- the resulting increased operating temperatures would certainly shorten component life.

These speculations are, of course, merely preliminary and should not be construed as possessing any remarkable validity -- perhaps some laboratory tests would be appropriate.

--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.

Ctrl
Guest Editorial
INK SPOT

Leonardo the Techie

by Phil Lemmons

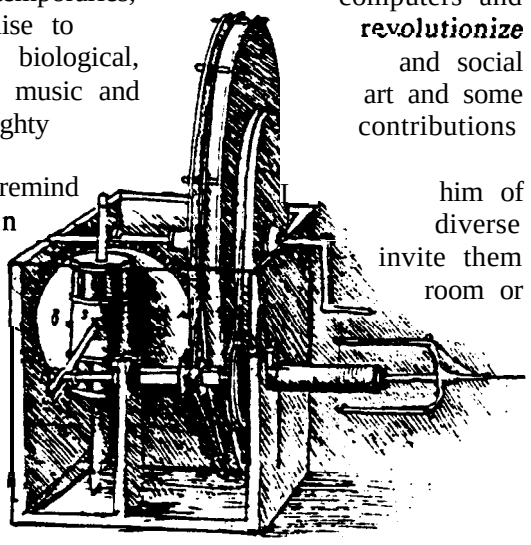
Technical people often fall victim to stereotyping. Once their technical gifts become obvious, technical people may be regarded as being capable only of technical work, and somehow disabled for all other purposes. The terms "techie" and "nerd" are sometimes used to cast aspersions against the technically adept. Yet my impression after years of working with technical people is that they are more likely than most to have a wide variety of interests and abilities. Indeed, if anyone in our time deserves the title of "Renaissance man," he is more likely than not to be a "techie."

Consider the case of a true and literal Renaissance man. Leonardo da Vinci designed flying machines, both winged and helicopter style. His plans for a car used internal springs to drive gears. He designed a type of chain that resembles those used in twentieth century machinery. His flyer spindle resembles those later used in textile manufacturing. His design for a steam-driven machine was the first instance of a piston moving within a cylinder. He designed mortars with projectiles intended to shatter into fragments on impact. He designed the world's first military tank, which had armor and a cannon and was to be propelled by eight men turning cranks to drive gears within. He studied the optics of mirrors and he sketched plans for life preservers and floating shoes. He designed mills for rolling sheet metal; a centrifugal pump; and a new type of oil lamp, using a water-filled glass globe to magnify the light. He failed to develop the field of electricity and electronics, but we should forgive him. He was, after all, born in 1452.

Did Leonardo's technical inventiveness disable him for all other purposes? No, indeed. Like many technical people, he had strong gifts and interests in music. He was both a singer and a lute player. According to the sixteenth century art historian Giorgio Vasari, Leonardo was also noted for his physical strength: "with his right hand he could bend the clapper of a knocker or a horse-shoe as if they had been of lead." And oh, yes, Leonardo could paint a little.

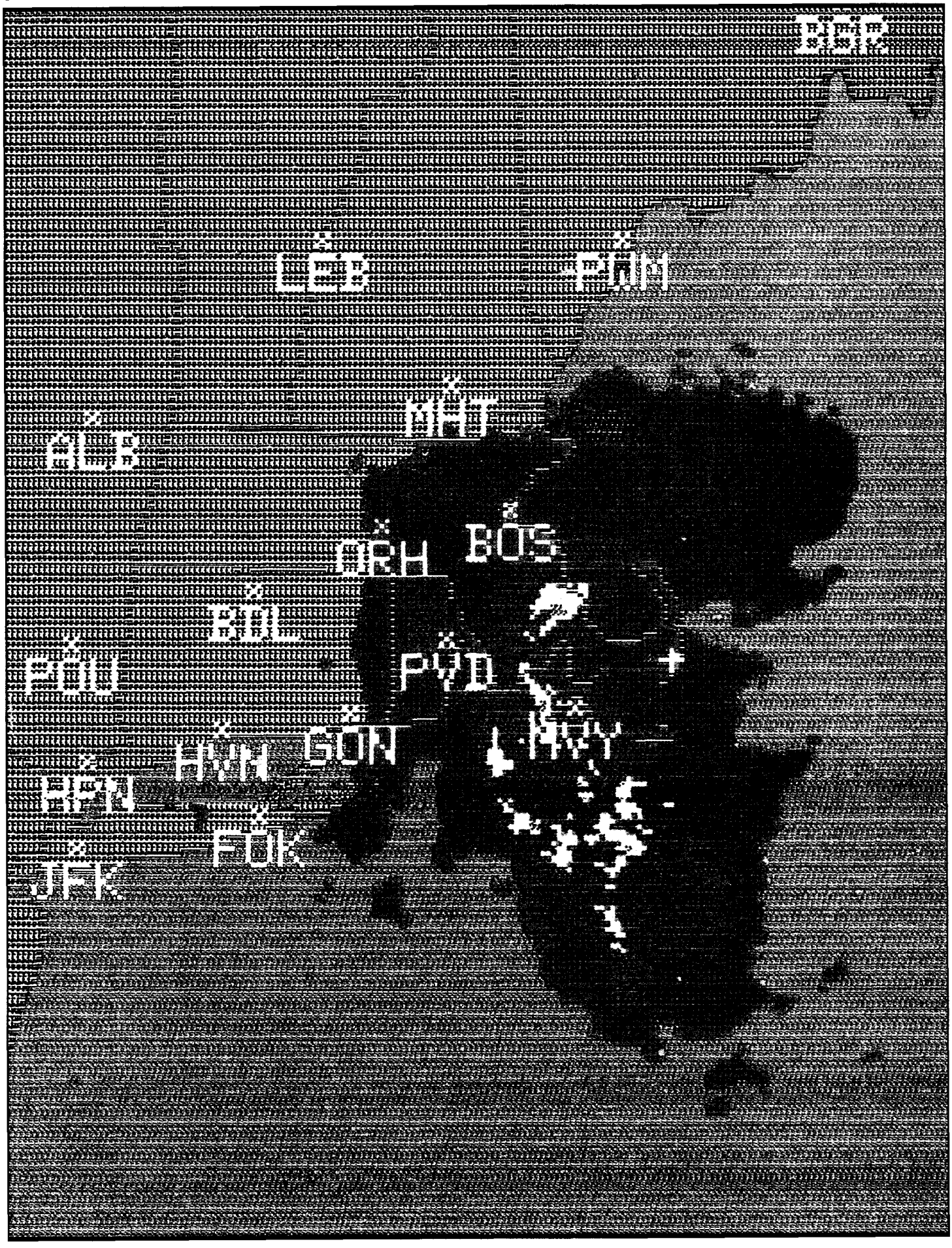
Our lifetime has seen a flowering in computers and electronics that could be called a Renaissance if this were not its first birth. Thanks to the inventiveness of many of our contemporaries, computers and electronics are revolutionizing business and manufacturing and promise to revolutionize education by making possible simulations of all manner of physical, biological, and social processes. Computers and electronics are also providing new vigor to music and art and some wild experimentation in literature. Technical people are making mighty contributions in all these fields.

The next time you hear someone stereotyping technical people, remind him of Leonardo's technical side and, with due modesty, point out your own diverse activities and those of your technical friends. You might even invite them over to Steve Ciarcia's to listen to fine music in his entertainment room or take a ride in an exotic sports car and above all to sample his fine cuisine. Never miss a chance to shatter the tiresome stereotype of the one-dimensional techie.



flyer spindle

Phil Lemmons is a former Editor-in-Chief of BYTE and an accomplished techie.


BOS
PVD
BUR

The Home Satellite Weather Center

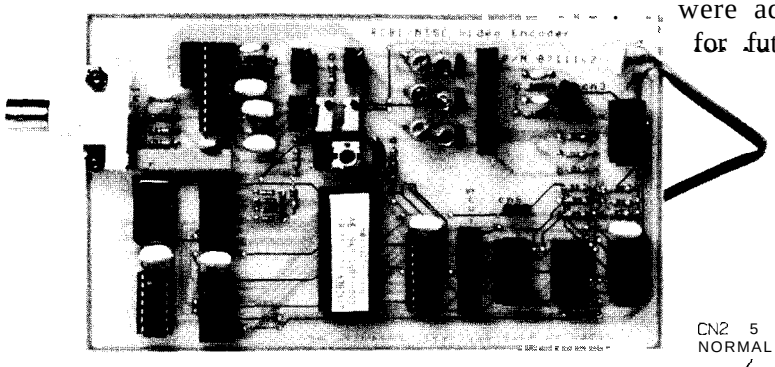
Part 2

NTSC Encoder Alignment and System Overview

by Mark Voorhees

In our last segment, we began the Weather Center project with the design of an RGBI-to-NTSC encoder board. This encoder will be used to convert weather map information displayed on an IBM PC (or PC compatible) EGA card into a VCR-recordable format or to view the data on a standard NTSC color monitor. Feeding these graphics images to a VCR allows archiving interesting radar and facsimile pictures, while freeing up valuable disk space for more current graphics.

In this part, I'll describe how to align the NTSC encoder and we'll also build an accessory which will allow you to use one or two TTL color monitors and the encoder simultaneously. Following this introductory dip into the hardware, I'll describe more about the ultimate configuration of the Home Satellite Weather Center.



Constructing the Encoder

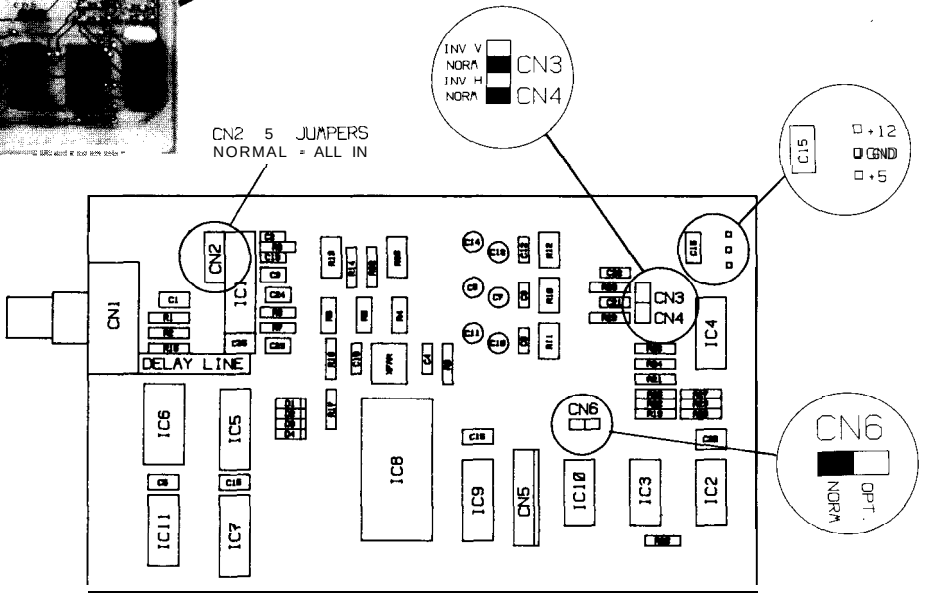
The schematic for the RGBI-to-NTSC encoder was presented in the last installment (INK Vol 1, No. 1). The schematic contains sufficient information to build this project, but printed circuit boards are available. I encourage you to build the Weather Center; the contents of the 27256 EPROM for the encoder as well as the BASICA test programs listed in this article are available for download from the Circuit Cellar Bulletin Board (203-871-1988).

The unit can be built on a breadboard, but I must caution you to be sure to use good layout practices for wiring to avoid any possibility of crosstalk or degradation of performance. Some of the header connections were added to allow for future options.

For a point of reference and ease of discussion, the printed circuit board layout and actual board are shown below. Note specifically the position of all jumpers and headers.

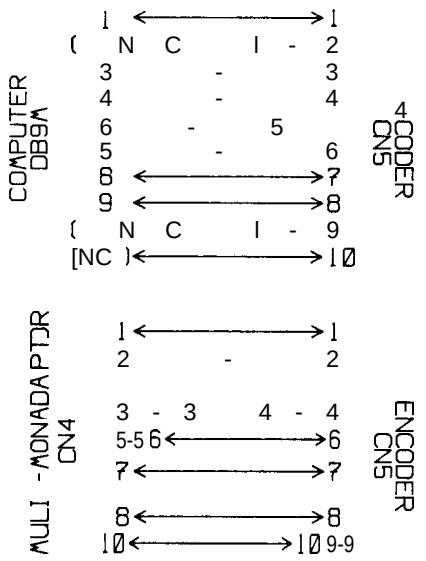
CN2 allows direct access to the MC1377 input pins, and will operate as a connector for a future subassembly. For now, however, jumpers should be in all five positions. CN6 allows for an interlaced sync to be configured in the EPROM output pattern if this is needed in the future. CN3 and CN4 allow for inversion of vertical and horizontal sync inputs from the graphics card, if necessary. All these jumpers are shown in the normal positions in the schematic.

You will need to supply +5VDC and +12VDC to the board from an external, regulated power supply.



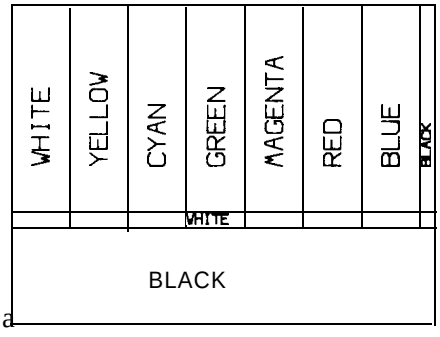
Board power requirements are minimal, so a utility supply of 500mA rating at each voltage should be sufficient. I will not offer suggestions on an enclosure at this time, but if you plan to build the entire Home Weather Station, you should keep in mind that I'll be doing something about a cabinet during the construction phase of the peripheral processor. I'll also discuss system power requirements so that all of these projects can be integrated into a single unit.

The RGBI signal from the computer connects to the encoder via CN5. The diagram below shows the connection information for making the cable to go directly to an IBM PC DB9 RGBI connector. If you decide to build the TTL signal adaptor described later in this article, you will need a DB9 male to DB9 female cable to connect between the computer and the adaptor, and you will not need the direct input cable shown.



To align the encoder you will need an oscilloscope, a composite monitor, and a PC with a CGA or EGA video card. The BASIC program in Listing 1 will generate a color-bar pattern similar to the standard NTSC color-bar test pattern. If your video card supplies R,G,B and I signals, the BASIC program in Listing 2 allows you to display 16 vertical bars across the screen. However, if your video card does not generate the intensity (I) signal, you will only see 8 bars. This second program is primarily for demonstration purposes.

To simplify alignment, we will use the program in Listing 1. Our goal at completion of the alignment is a display of color bars as shown in th diagram below.



Listing 1

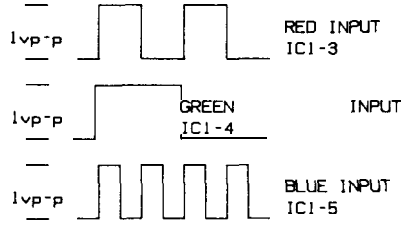
```
10 REM COLOR BARS TEST FOR RGB/NTSC ENCODER BOARD / 8 BAR
   ALIGNMENT  VERSION
20 REM COPYRIGHT 1987 BY MARK VOORBEES
30 REM ALL RIGHTS RESERVED
40 CLS:KEY OFF
50 FOR X=1 TO 22
60 LOCATE X,1,0
70 COLOR 0,7,0:PRINT SPACE$(11);
80 COLOR 0,6,0:PRINT SPACE$(11);
90 COLOR 0,3,0:PRINT SPACE$(11);
100 COLOR 0,2,0:PRINT SPACE$(11);
110 COLOR 0,5,0:PRINT SPACE$(11);
120 COLOR 0,4,0:PRINT SPACE$(11);
130 COLOR 0,1,0:PRINT SPACE$(11);
140 NEXT X
150 LOCATE 23,1,0:COLOR 0,7,0:PRINT SPACE$(80);
160 LOCATE 1,1,0
170 GOT0 170
180 END
```

Listing 2

```
10 REM COLOR BARS TEST FOR RGB/NTSC ENCODER BOARD/ RGBI 16
   BAR  VERSION
20 REM COPYRIGHT 1987 BY MARK VOORBEES
30 REM ALL RIGHTS RESERVED
40 CLS:KEY OFF
50 FOR X=1 TO 22
60 LOCATE X,1,0
70 COLOR 0,15,0:PRINT SPACE$(5);
80 COLOR 0,7,0:PRINT SPACE$(6);
90 COLOR 0,14,0:PRINT SPACE$(5);
100 COMR 0,6,0:PRINT SPACE$(6);
110 COLOR 0,11,0:PRINT SPACE$(5);
120 COLOR 0,3,0:PRINT SPACE$(6);
130 COLOR 0,10,0:PRINT SPACE$(5);
140 COLOR 0,2,0:PRINT SPACE$(6);
150 COLOR 0,13,0:PRINT SPACE$(5);
160 COLOR 0,5,0:PRINT SPACES(G);
170 COLOR 0,12,0:PRINT SPACES(S);
180 COMR 0,4,0:PRINT SPACE$(6);
190 COLOR 0,9,0:PRINT SPACE$(5);
200 COLOR 0,1,0:PRINT SPACE$(6);
210 NEXT X
220 LOCATE 23,1,0:COLOR 0,7,0:PRINT SPACE$(80);
230 LOCATE 1,1,0
240 GOT0 240
250 END
```

Connect the unit to a power supply and composite monitor. Run the test in Listing 1 (after it creates the screen display, it will loop until <CTRL><BREAK> is pressed), then connect the graphics card output to the encoder unit. Don't be surprised if the picture is monochrome, or, if in color, the colors are "strange." There are several adjustments to be made. You should, however, see some image at this time. If not, recheck the board for proper component placement and soldering, and recheck your input cable and power sources.

If you have achieved success to this point, set all controls to the center of their range (the recommended potentiometers are 20-turn, so the easiest way to set them is to turn the adjustment 20 turns in one direction, then 10 turns in the opposite direction). Connect the oscilloscope probe to IC 1-3 and adjust R11 for a 1 volt peak-to-peak (P-P) maximum signal. In a similar manner, adjust R10 for 1 volt P-P at ICI-4, and R12 for 1 volt P-P at IC1-5 (see the diagram below for an example of the waveform).



While observing IC1-17 with the scope, set the level of the 3.579545-MHz square wave to approximately 1 volt P-P (this and all other adjustments will be fine-tuned later). Next, look at the output of the encoder chip at the rear of the BNC connector with the monitor connected. You should see a waveform similar to that shown in Figure 1.

As you make the following adjustments, refer to Figure 1 for guidance as to the affected waveform elements. Adjust R33 in one direction slowly until burst appears on the waveform, and note the adjustment position. Continue turning in the same direction until burst just disappears, then turn the pot in the opposite direction until you are half-way between the burst on and off positions. This is called "bracketing" the adjustment, or centering it in its operating "window."

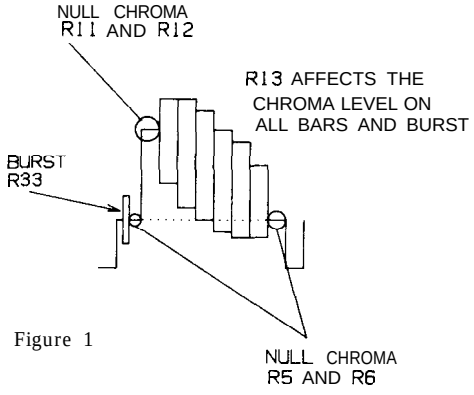


Figure 1

Adjust R5, then R6, to minimize the chroma on the video baseline. These controls interact, so you may have to adjust them several times to null the chroma, and you may not be able to truly null it yet. Noting the proper area of the waveform in Figure 2, fine-adjust R11 and R12 for minimum chroma on the white bar. Once again, these interact, so they may need to be adjusted several times to null the white bar. Readjust R5 and R6 as above to renull the chroma on the baseline.

The next adjustment is the fine adjustment of R13. Turn the control slowly in one direction until you see a decrease in all chroma levels (if the levels are increasing, turn in the other direction until you see all levels decrease). Next, turn the control slowly in the opposite direction until you see the chroma levels reach a "plateau" (i.e., a point at which the levels don't increase as you continue turning R13). What we are doing is setting R13 to a level at which the encoder performs well without overloading the chroma mixers in the MC1377.

The final instrument adjustment is R4, which sets the "geometry" of the color bars. If you have access to a vectorscope and are familiar with its operation, this adjustment will be simple. It consists of tuning until the bar vectors fall close to their respective targets. If you have only the oscilloscope, adjust this control to meet the criteria of Figure 2 as closely as possible. C6, C9, and Cl2 are roll-off adjustments to minimize ringing of sharp edges (such as those caused by small characters) on the composite monitor. This adjustment is empirical, and is usually made through observation of the composite monitor in use. I suggest that you repeat this adjustment procedure several times to produce optimum signal quality.

Troubleshooting

If you should encounter problems with the alignment, check for the following possible causes. You should be able to identify any error based on the schematic and this information:

No **picture** -- Check power and signal connections. Check for 3.579545-MHz square wave at IC7-10; check for sync signal at IC1-2. Recheck construction.

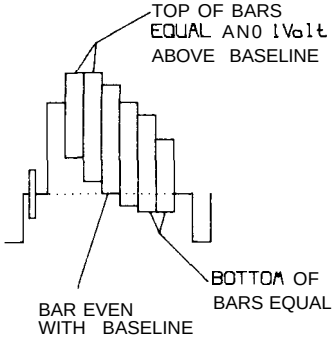


Figure 2

Picture rolls vertically (horizontal O.K.) -- Check for positive-going vertical-rate pulse at IC5-11. If pulse is negative-going, move CN3 to other position.

Picture rolls vertically and rolls or jitters horizontally -- Check for positive-going horizontal-rate pulse at IC5-10. If pulse is negative-going, move CN4 to other position.

Colors do not match color **bars** -- If minor hue difference, repeat alignment procedure. If colors are in the wrong place in the pattern, recheck cabling to the computer graphics card.

Monochrome picture only -- Check for 3.579545-MHz signal at IC1-17, and positive-going horizontal-rate pulse at IC1-1.

Extra horizontal lines in the video -- The EPROM (IC8) is too slow. Only a 27256 or 27C256 with an access time of 200 nsec or faster should be used.

The multimonitor accessory operates as a simple TTL signal repeater, taking its power from the encoder card and delivering a buffered TTL signal in return. The schematic is shown in Figure 3. The device is quite straightforward and requires no alignment. If a second TTL monitor output is not needed, the components shown within the dashed lines can be omitted.

The cable to connect the adaptor and the encoder cards is wired "straight through" (i.e., pin 1 to pin 1, pin 2 to pin 2, etc.). A DB9 male to DB9 female cable (also wired "straight through") may be used to connect the graphics card signal to the adaptor.

Let's take a moment to look at our Weather Center project in a little more depth, and also see what's ahead in the next few installments.

As noted in the last issue, our completed system will be a Home Weather Center capable of maintaining local instrument records, receiving National Weather Service Facsimile pictures directly from satellite, and handling your access and processing of information from various weather databases.

Your IBM PC, XT, AT, or IBM compatible will be your link to the peripheral processor. The peripheral processor is our next major hardware project. It will handle all the instrument and facsimile data 24 hours a day. The PC software will take care of access to the dial-up databases, transfer of data from the peripheral processor (and transfer of commands to it), the chores necessary to maintain the instrument database, as well as all

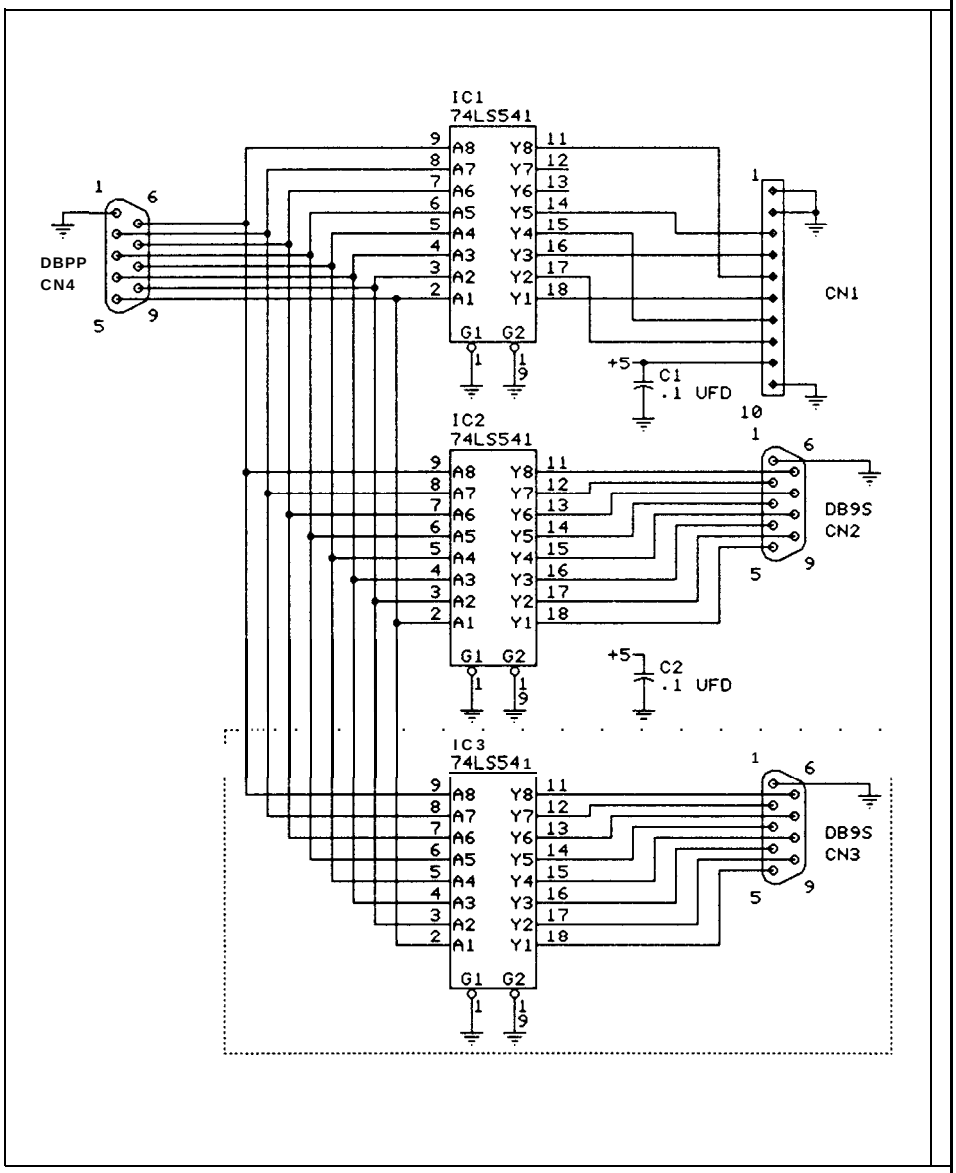


Figure 3 -- Multimonitor Adaptor

interaction with you, the operator. By using the PC in this way, you can have full-time monitoring and graphics capabilities without dedication of a PC to this application.

The RGBI-to-NTSC encoder which we've just completed will allow you to create recordable, composite NTSC video from your PC's video display card output. Not only will this provide virtually unlimited storage for the weather graphics you produce, you probably will find many other uses, such as titles for home videos, classroom demonstrations, and so on.

Our next hardware project is the peripheral processor, a device which will handle the full-time job of storing instrument data samples and current facsimile picture data from satellite. The unit is a dedicated 68000-family-based single-board computer, which has 1 Megabyte of RAM on-board for data storage, and a 32K-byte EPROM for the dedicated operating system. Depending on the sampling rate set by the user, the unit will hold one to four days of data before it must be transferred to the PC for processing.

Construction of the local weather instrumentation itself will not be covered here, although I will cover interfacing to two target systems: the Heathkit ID-4001 Digital Weather computer (with ID- 1795 rain guage and ID-2295 relative humidity instrument), and the new Heathkit ID-5001 Advanced Weather Computer (note: this is so new that, as of this writing, I have just received the unit; I'll include a short appraisal of the ID-5001, as well as the older ID-4001, in a future issue). I've chosen to support these instruments since they appear to be the most accurate and widely accessable instruments available to the experimenter for the money.

After completion of the main processor unit, we'll cover the construction of the ports and interfaces to the instruments, and then move on to the facsimile demodulator. I'll approach that accessory in two ways: use of an integral, dedicated receiver to feed the converted RF to the demodulator, and use of a scanner-type receiver. We will not delve into the RF down-conversion from 1691 MHz in this series. This requires very precise, microstripline techniques something not easily done at our level. I will, however, detail the installation and setup of a modular downconverter and antenna system distributed by an East-Coast supplier, and available for much less than you would expect to pay for the materials to build it yourself.

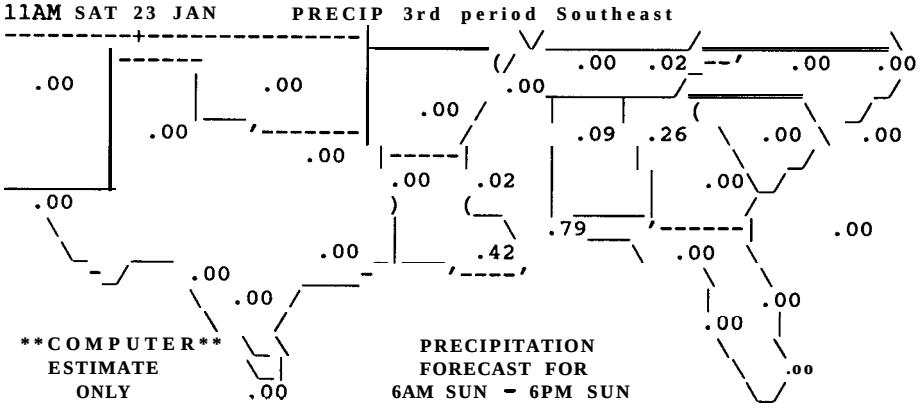
Remember that your suggestions for other general-purpose accessories for the system may be incorporated as additional future projects. The peripheral processor is designed to be very flexible, and should easily handle additional weather-related tasks beyond the instrument and fax housekeeping. If you have a suggestion, write to me at the address listed in the ordering information.

For the next installment, we'll let the soldering irons cool a bit while we discuss weather databases which are available via modem. I'll also introduce the first of the PC/weather support software -- the basic processor system, with modules for accessing the databases, as well as capturing and printing the data. The first accessory module will be the Advanced Radar Display System, which will convert National Weather Service Radar Station text data to a U.S. map graphic with the overlay of the radar echoes. You'll be able to store the screen on disk, or on video tape using your NTSC encoder.

All of the PC software will be modular in nature. Each additional module (introduced as we progress through the system) will automatically interface with the basic processor software.

Shown below is a typical ASCII data image. It is an example of the kind of data available to you from a dial-up data service. This map gives the computer predictions of precipitation in the southeast U.S. for the 24-hour period following the noted forecast time. (Of course, quite a bit of text data is also available). We'll explore these services in depth in our next segment. ■

(Ordering information - page 42)



Typical ASCII data image

CONNECTIME *Excerpts from the Circuit Cellar BBS*

THE CIRCUIT CELLAR BBS
300/1200/2400 bps
24 hours/7 days a week
(203) 871-1988 -- 4 incoming lines
Vernon, Connecticut

Sysop: Ken Davidson

Before we get to the messages, I want to direct your attention to something I ran across while editing this month's batch. Anyone who has done enough on-line computing, whether it be on one of the big conferencing systems or networks, or on local BBSs has probably seen the sequence of characters that looks something like :-). If you haven't seen them before, put your left ear on your shoulder and take a closer look. Out pops a little smiley face!

During normal face-to-face conversation, the mere words being spoken often don't tell the whole story about the thoughts or ideas being expressed. The way the speaker SAYS the words makes a big difference. When a person shouts "You jerk!" and has a stern look on his face, there's no mistaking that you had better run and hide. If the same phrase is said with a chuckle and smile, you know the person saying it is poking a little fun at your dumb mistake and you can laugh right along with him.

When those little visual and audible cues are removed, we're left with a lot of ambiguities. Conversing by computer does just that. Punctuation can help, but only to a certain extent. Enter the smiley face.

Often when a person is leaving a message on the BBS that is poking fun at something or someone, but could be taken conversely if read the wrong way, the sender will tack a :-) to the end of the sentence or message to show that what he's saying should be taken with a grain of salt. Over the years, creative soles have expanded on the original smiley to come up with some real cute faces.

A wink might look like ;-). If you're sad about something, you might type :-(. Someone with a mustache might leave a :-{). When you're in awe of something, your face might look like :-O. Or if you want to stick your tongue out at someone, try :-Q. Add some glasses and your face becomes 8-).

The list goes on, limited only by your imagination. So if you see a face in one of these messages, remember that it's there to help you to better understand what the person is trying to say.

Enough about BBS etiquette, back to the business at hand. Last month there seemed to be a lot of talk about digitizing audio signals. The phone company has been doing it for years, the record industry has recently started distributing records in digital form (CDs or compact discs), and the ever-curious experimenter is busy down in the basement. Here are some of those conversations:

Msg **#8934** by Jim **Thorton** To: ALL USERS
Re: Audio Digitizing

I need some info on building an audio digitizer. Is there possibly a text file around here somewhere that could help? Or maybe this project has already been undertaken in the Circuit Cellar (quite likely)? If so, could someone point me at the correct back issue? Or maybe even a good book on the subject?

-- Jim **Thorton**

Msg **#8983** by Bob Paddock To: Jim **Thorton**
Re: Audio Digitizing [Reply to msg **#8934**]

What type of audio do you want to digitize? Voice, music, noise? Do you want something that anyone can listen to and know what it said, or just something that only you could understand?

Msg **#9099** by Jim **Thorton** To: Bob Paddock
Re: Audio Digitizing [Reply to msg **#8983**]

Bob,

To answer your questions, I'd like to be able to digitize voice, music, AND miscellaneous noise. It's just something that I find intriguing and would love to do. I considered buying one until I saw what one for

an IBM PC would cost. I'd like to get fairly high quality. I have a friend working on putting an SID chip (the sound chips in the Commodore 64) onto an IBM PC expansion board, so (if he gets his timing problems worked out) I'd like to use that for playback. The SID has only three voices though, so if the SID project works, it should be fairly simple to use one of Ensoniq's many-voice chips in a similar design.

I guess I need to find (first of all) some METHODS of digitizing audio that will produce high-quality digital representation.

-- Jim

Msg **#9152** by Bob Paddock To: Jim **Thorton**
Re: Audio Digitizing [Reply to msg **#9099]**

What you want to do is not as easy as it first appears. If you want to digitize just voice at communication quality (i.e., telephone/radio quality), you can use one of the many chips that are available for the telephone system. Intel, National Semiconductor, Texas Instruments, and many others make such chips. The problem they all suffer from, if you want to store the digitized audio in your computer, is that they take a LOT of memory for very short periods of speech.

If you want to digitize music, like CD or DAT (digital audio tape) quality, then you start needing things like **16-bit** A-to-D converters, (one recent CD player uses an 18-bit system), and wide-bandwith audio circuits, none of which come cheap. A 1 -second sample of music could take up more memory than most computers have right now.

Philips (Signetics) makes a few CD chips (all playback, I don't know if they have any for recording).

I never really thought about digitizing noise before. As a first guess, I would say that it is harder to digitize than music or speech. Since noise can conceivably contain all frequencies at once, you would need to use really wide-band sampling and a lot of memory.

I don't know much about the SID, but to get it to play back something that was digitized in some other format would probably take a lot of number crunching to translate the data.

Msg **#9191** by Jim **Thorton** To: Bob Paddock
Re: Audio Digitizing [Reply to msg **#9152]**

Yes, having since found a book on the subject I have learned quite a bit about digitizing. You're not kidding about it being expensive--the specs required for the A/D and D/A circuits and the low-pass filters are fairly rigid. I think that I'll be able to scrape up at least enough parts to build (or attempt to build) one of them though.

Somebody on the West Coast built a fairly nice digitizer for the Commodore 64 a few years ago, but that person seems to have fallen off the edge of the earth after building and selling about 50 of them. With some good editing, you can take a few small samples and construct a fairly interesting "song" with them, even with only the 64K in the C-64. I'm not shooting for CD quality by any means (16-bit converters don't come cheap) but something that could be understood by anyone that happened to hear it would be nice.

-- Jim

Msg **#9023** by Ben Barlow To: Jim **Thorton**
Re: Audio Digitizing [Reply to msg **#8934]**

Jim, I built an audio digitizer that is based on an **OKI** Semiconductor chip (MSM5218) and was described in the June '83 Circuit Cellar article. It worked pretty well, but I never got quite the sound out of it that I wanted. The microphone and analog stuff delays me; I'm OK at digital, but I'm a hobbyist, not an engineer. I'm interested in discussing the topic as well if you get other approaches.

-- Ben

Msg **#9127** by Ben Barlow To: Jim **Thorton**
Re: Audio Digitizing [Reply to msg **#8934]**

Jim, some more information. OK1 has a new ADPCM chip with an evaluation board offered for testing. The board is $290, though. My previous experience with the **OKI** chip set that Steve used in his June '83 article was that it worked pretty well, but some of the parts (the filters) were unobtainable in 1987. I don't know whether the new OK1 set would encounter similar problems or not. (Of course, if I knew what I was doing, I could make my own filters, but...)

Motorola offers a chip combination (MC341 7 or - 18, a CVSD digitization chip, and MC145414 programmable filter) that looks like it might be a good deal (and available). The Motorola "Telecommunications Device Data" manual contains information about the chips and a couple of application notes with complete serial and parallel interfaced setups.

-- Ben

Msg #9332 by David Lovelace To: Jim Thorton
Re: Audio Digitizing [Reply to msg #8934]

Jim, your problem could be simple to solve if you could supply some additional information regarding your requirements. The digitizing can be accomplished through first filtering the audio input to reduce noise and to reduce bandwidth (if required), and then feed the filtered input into a video-rate A/D converter for digitizing. How you handle the digitized signal is left open to your needs. The digitized form of the audio signal can be converted to an analog audio form through the use of a D/A converter. The output of the D/A converter would then be filtered to reduce the high-frequency harmonics that accompany D/A converters.

I would like to suggest two low-cost Signetics products for use as the converters. The PNA7507 is an A/D converter with 7 bits of resolution and a conversion rate of 15 MHz. D/A conversion could be accomplished with a PNA7518 D/A converter with 8 bits of resolution and a 30-MHz conversion rate. Low cost is the common denominator in the selection of these components.

Perhaps I have over-simplified your application. The method used in the digital section is where the most sophisticated techniques for handling the audio would be applied. Examples are: error detection and correction for high reliability of the data stored, and digital signal processing for filtering, time delays, and so on. If any digital signal processing other than that accomplished by a standard microcomputer is attempted (e.g., signal processing with an add-on card using a DSP (digital signal processor), dedicated DSP) the cost increases along with the cost of any available software.

Digital signal processing with a microcomputer could yield nice results, but not in real time. Texts on digital signal processing abound as well as several tried and reliable computer algorithms. These algorithms could be easily converted to FORTRAN, Pascal, or C. I would be very interested in your problem and would like to help in any way possible.

Msg #9244 by Scott Whittle To: Jim Thorton
Re: A/D-D/A for audio

Jim, I've read some of your messages on this board regarding the digitizing of audio signals. I too have been very interested in such a device and have done much research on it. I was wondering if you could tell me what book you have on the subject. Perhaps we could exchange thoughts and ideas as well as techniques regarding the A/D and D/A processes. Keep in touch. Thanks.

-- Scott Whittle

Msg #9282 by Jim Thorton To: Scott Whittle
Re: A/D-D/A for audio [Reply to msg #9244]

Scott, the book I found is called "Principles of Digital Audio" by Ken Pohlmann. It's published by SAMS. It's well-written and easy to follow. The latter part of the book goes into some of the coding schemes, different medias, and how they work, including much about CD players.

It's helped me to be able to conceptualize what I need to do, but I am still a novice experimenter (very novice).

One thing that I'm wondering about is the dither generator required for antialiasing purposes. What sort of device would be used for that purpose?

Another is the resolution of the A/D and D/A converters. Pohlmann says that 16 bits is the industry standard for CD and Digital Audio Tape (DAT). What do you think an acceptable resolution would be? Eight-bit converters seem to be very common/cheap, but will the quantization error be too great? I think 12 bits would be more realistic, but what does that do to the price of this sort of thing?

-- Jim Thorton

Msg #9304 by Eric Bohlman To: Jim Thorton
Re: A/D-D/A for audio [Reply to msg #9282]

As far as resolution goes, you should have at least 12 bits for a linear DAC or ADC. In the telephone industry, the bit rate is usually reduced by using companding DACs and ADCs. These have a nonlinear resolution; the ADC's step size increases at the higher values. This works because at high values a small step is much smaller in percent than the same size step at a low value. I believe that digital telephony normally uses 8-bit codes. The logarithmic companding scheme is either A-law or mu-law; one is used in the U.S. and the other in Europe (I can't remember which is which).

Msg #9320 by Bob Paddock To: Jim Thorton
Re: A/D-D/A for audio [Reply to msg #9282]

Texas Instruments just came out with a couple versions of an A/D-D/A CODEC that is an 8-bit part with 12-bit dynamic range. They are the TCM29C18 and TCM29C19; one works at 1.536 MHz and the other at 2.048 MHz.

We've just finished installing a brand new multiline BBS to keep up with the terrific response we've had to the Circuit Cellar BBS. In the coming months I'll be taking you on a "guided tour" of the various facilities available while on-line. In the meantime, the following messages reflect the change in format of the message headers.

There are many clock/calendar chips on the market, each seeming to have its own interface characteristics. The following is an attempt at making two incompatible interfaces compatible:

Msg#: 534 *GENERAL*
From: RICHARD ANDREWS
To: ALL USERS
Subj: CLOCK/CALENDAR CHIPS

I am in the process of adding an MC146818 clock/calendar to a Z80-based system. The timing information available is somewhat hazy concerning interfacing to a processor that does not use a multiplexed bus (i.e., address and data share the same pins) or that does not use Motorola timing (the Z80 naturally is Intel based in that respect). One of the reasons that this IC is appealing is that it is the same one used in the IBM PC/AT so second sources and availability should be no problem. The 80286, to the best of my knowledge, does not multiplex its bus, so I am left wondering if there are any hints that anyone can give on doing this. Many thanks

Msg#: 542 *GENERAL*
From: DON NOWAKOWSKI
To: RICHARD ANDREWS (Rcvd)
Subj: REPLY TO MSG# 534 (CLOCK/CALENDAR CHIPS)

Rich,

I have interfaced the MC146818 to an MC68000 with success. I have heard someone comment that the 6818 must have been designed by someone from Intel rather than Motorola just because of the multiplexed address and data bus. In any case, use a '245 device (LS or HCT) with the direction controlled by whether the RD\ or WR\ line is active. Use a '244 for the address line.

The outputs of the '245 and '244 are connected to the ADx lines of the RTC. The best timing arrangement in dealing with this interface is to leave both devices in the tri-stated mode until the RTC is actually accessed. At the beginning of the cycle, develop your CS\ signal and drive AS high for the time specified (I don't remember off-hand) to latch in the address. During this time, the output enable on the '244 is low. After this, drive AS low, drive the '244 OE\ high, then drive the '245 OE\ low and strobe DS high to latch the data. This nonoverlapping sequence is essential to making the I/F work. There will be those who claim it is not necessary, but when you want 5000 pieces to work guaranteed, you arrange it as I have described.

I have only outlined the necessary elements; the design is not done (it is left to the student, but you get the idea). Two pieces of advice: if you want to maximize your backup time (if battery life is critical), drive CS\ from an open-collector driver that has its output pulled up to the battery voltage. Also, do not connect a 32.768-kHz oscillator directly as shown by Motorola; a much less power-hungry solution is to use a nonbuffered CMOS inverter such as the 4069 to build an external oscillator and drive the RTC with that. The total circuit draws less than 65 microamps. For a more forgiving interface, use the MC14681 8A instead of the "non-A" version. I have never interfaced to a Z80 so I can't go any further, but I think you'll succeed.

-- Don Nowakowski

Msg#: 589 *GENERAL*
From: RICHARD ANDREWS
To: DON NOWAKOWSKI (Rcvd)
Subj: REPLY TO MSG# 542 (CLOCK/CALENDAR CHIPS)

Thanks for the suggestion, I will mull over what you said and see if it will apply to the 280. The Motorola data book shows an example of the RTC being used with a 68000, but I'm not that familiar with that CPU so I'll have to give look. Thanks.

The Circuit Cellar BBS runs on a IO-MHz Micromint OEM-284 IBM PC/AT- compatible computer using the multiline version of The Bread Board System (TBBS 2.0M) and currently has four modems connected. We invite you to call and exchange ideas with other Circuit Cellar users. It is available 24 hours a day and can be reached at (203) 871-1988.

FIRMWARE FURNACE

Digitizing Infrared Signals

by Ed Nisley

Here's a secret that personal computing benchmarking articles never mention: your computer spends most of its time waiting for you. If you'd just push the keys faster its overall performance would perk up quite a bit. If you buy a faster PC it just executes more NOPs while it's waiting.

On the other hand, in this column I'll describe firmware that must keep up with high-speed input data. As I mentioned in the last installment, I am using the firmware developed for the Circuit Cellar Infrared Master Controller (BYTE, March '87) as the example for this series of articles. The IR Master Controller imitates other remote controllers by digitizing and recording their infrared signals. Because this data is both intermittent and high speed, it presents some unique problems in its acquisition. Once the IR data signal starts, there's no way to go back and take another look at it!

Before diving directly into the firmware, however, much can be learned about the task at hand by reviewing IR signals in general and looking at the IR Master Controller's hardware.

Master Recordings

You might think that there is a design standard controlling IR communication such as the transmissions between remote controllers and consumer electronics, perhaps similar to the RS-232 standard for serial communications. Of course, when you try to hook up any two serial devices, you soon realize that there's no "standard" in "RS-232 standard" serial communications. The situation is even worse in IR communications. There isn't even the hint of a standard. In fact, it's not unusual for a single manufacturer to use different transmission methods in each of their various remotes!

Figure la illustrates one typical IR transmission coding technique. As in all schemes, the IR LED is either on or off. Here, a high-level logic 1 bit turns on the IR LED during the first half of the bit time, while low-level logic 0 bit leaves it off. Both "1" and "0" bits are the same length, so this type of transmission is particularly easy to decode.

The more complex signal shown in Figure lb turns the LED on during both "1" and "0" bits, but distinguishes the logic level by pulse width. The time between bits is usually constant. The total time required for a message depends on the data values, and it's not so easy to decide which bit is which within the message.

In both cases above, the IR transmission is actually a group (called a burst) of much shorter pulses. This is similar to amplitude modulated (AM) radio, where the audio message modulates a carrier signal. It's different from the modulation used in FSK (Frequency Shift Keying) modems, which alternates between two distinct frequencies for "1" and "0" bits. The lower "traces" in Figure la&b show the state of the IR LED during the designated bit times.

There are no time scales in Figure la&b for a very good rea-

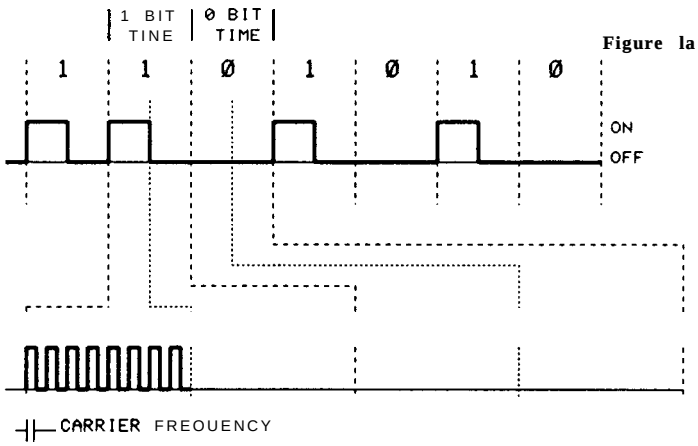


Figure la

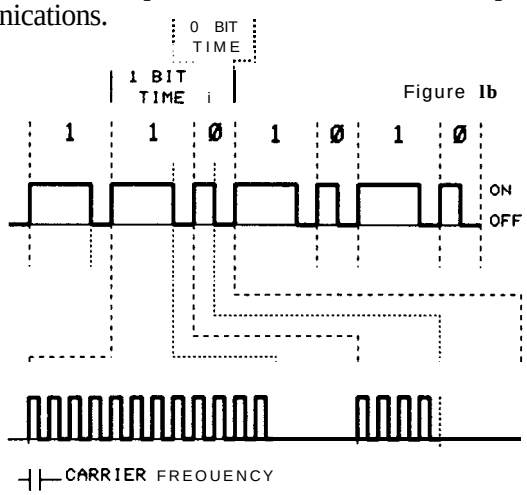


Figure lb

son. Every manufacturer uses different values! The IR carrier frequency is usually about 40 kHz, but it can typically be anywhere between 30 and 70 kHz. Bit times and the gaps between them can be a few hundred microseconds to well over 50 milliseconds. Total message lengths may be as short as a few milliseconds or as long as a quarter second.

There is also no agreement on what the bits actually mean. Some controllers produce a simple numeric code for each key, and others combine shift prefixes and suffixes to modify certain keys. Still others send out variable length messages depending on the function requested. The situation is a true Tower of Babel!

The overall message, though, resembles a single character sent over a serial data link, with perhaps several dozen bits instead of the conventional seven or eight. There is usually a very long start bit at the beginning and one or more stop bits at the end of each message. The exact formats, of course, vary between manufacturers just like the IR signal parameters.

Figure 2 shows how a typical TV set receives and processes the IR signal from its remote unit. After the photodiode converts the infrared light into an electrical signal, a highly selective bandpass filter passes the demodulated carrier as a received "envelope'* and rejects any IR carrier signals from remote controls out of the passband. The detector and level shifter convert the analog envelope signal into TTL logic levels, which are then ready for the TV's microprocessor-controlled tuner. Usually all of these functions will be integrated onto one or two chips.

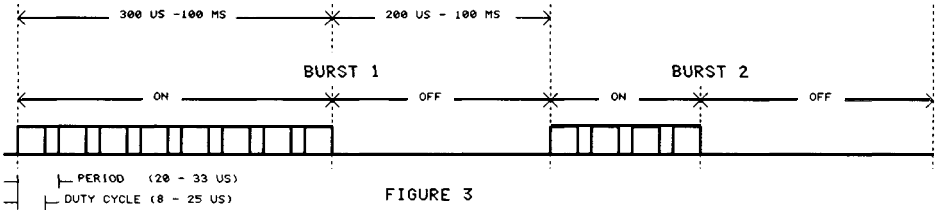


Figure 3

The signal coming out of the level shifter reconstructs the original bit stream from the remote unit with all the details of the IR transmission removed. That is exactly right for a TV microprocessor, but it is entirely inadequate for the IR Master Controller. The TV set designer chooses each of the components to enhance the set's own specific remote control signal and suppress any interference from other remotes and extraneous IR sources. As a result, any single circuit will work well with only one (or, perhaps, a few) remote units.

The IR Master sidesteps this problem by digitizing the received IR signal without any analog filtering or processing and using digital signal processing to analyze the results. Because the IR Master's output duplicates the input signals, it doesn't have to waste any effort deciding what they mean; there is no need to determine how the bits are encoded.

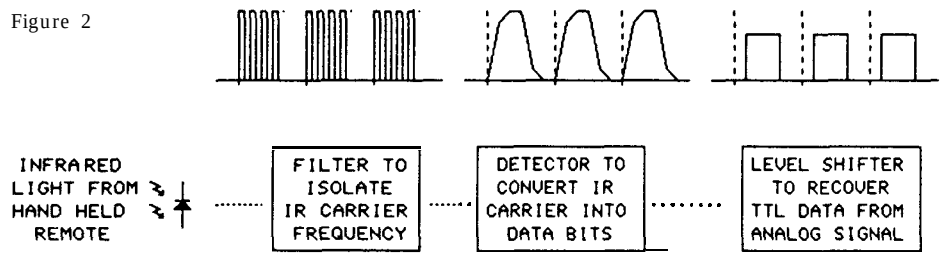
Measuring an IR signal's timing gives enough information to reproduce it even without knowing what function it might activate in the TV set or VCR. Figure 3 shows the values for a typical signal; remember that there are wide variations among actual signals.

The carrier period is the reciprocal of the carrier frequency. A 40-kHz carrier has a 25-microsecond period. The carrier duty cycle is the fraction of each carrier period that the LED is active. Most remote units have a 50% duty cycle, but we found variations between 40 and 75 percent of the period. Both the period and duty cycle are fixed for any given remote unit, so they can be measured once for each signal and used for the entire burst.

Recording the burst and gap times are enough to duplicate either of the coding schemes shown in Figure la&b, as well as many others. This system handles the start and stop bits at the beginning and end of each message even though those bit times may be different from all the rest.

The only complication comes at the end of a complete message, when the IR Master must decide when the message is finished. The IR Master's program assumes that messages are always less than 250 milliseconds long and simply times

Figure 2



out after the last burst.

The firmware's task is simple. It must measure the IR carrier signal period and duty cycle and then time each of the data bursts and gaps within the message. Those values will eventually be used to control an IR LED in the Master Controller's transmission section where it emits a convincing copy of the original signal.

A **Blizzard** of Bits

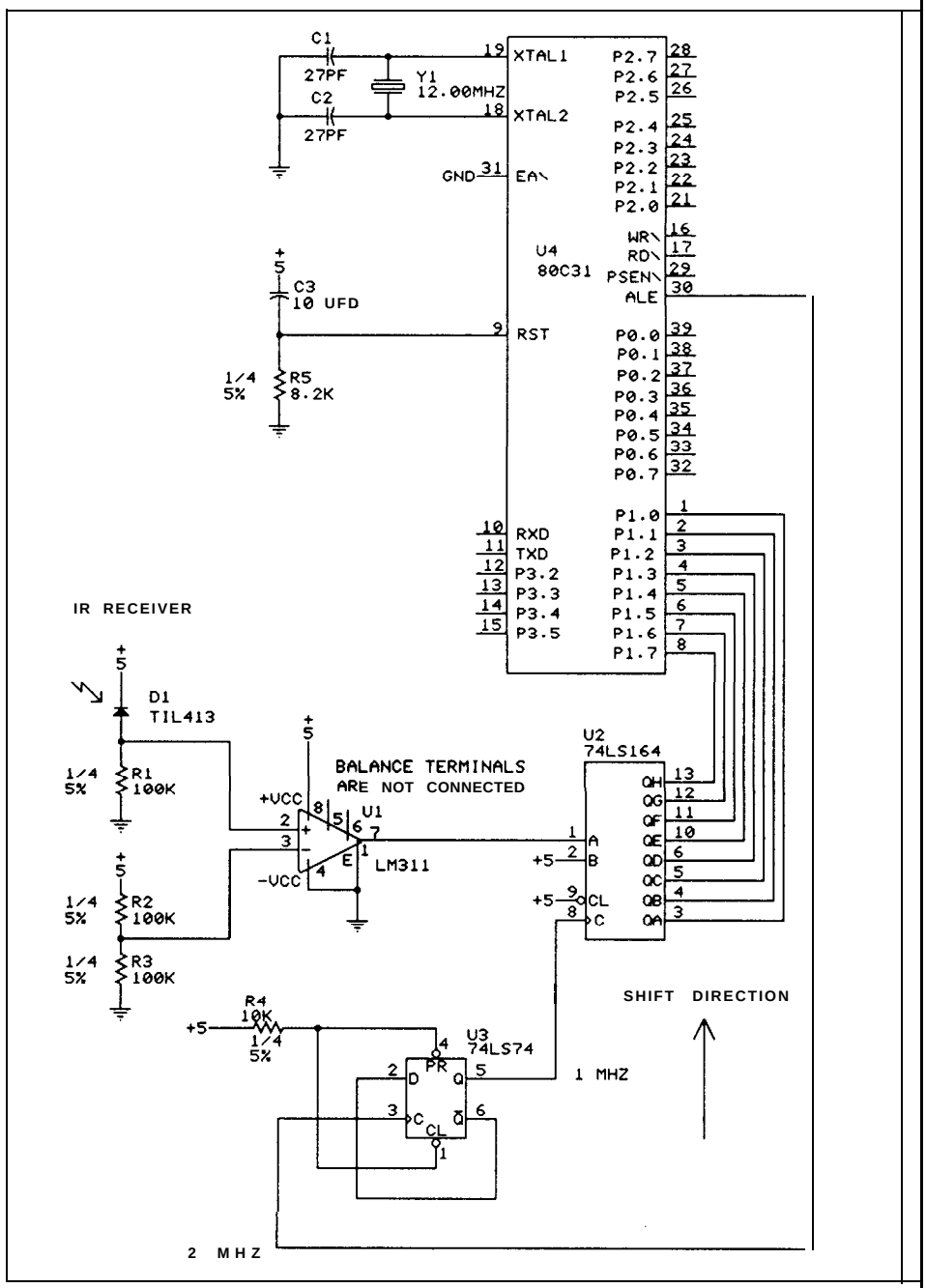
A quick and dirty way to measure time intervals with a microprocessor is with a software loop which detects rising and falling edges on an input pin. By recording the number of loops between edge detections, the interval can be calculated. Unfortunately, when the interval to be measured is shorter than the fastest program loop this technique cannot be used.

Carrier periods typically range between 20 and 33 microseconds. Within this period the IR LED "on" times are between 8 and 25 microseconds. One microsecond of measurement error can produce a 5% error in the reproduced frequency or duty cycle!

The 8031 executes most instructions in one or two machine cycles. A machine cycle requires 12 clock cycles, which at 12 MHz, is one microsecond. The smallest useful timing loop requires four or five instructions and takes up to ten microseconds to execute. Obviously, the 8031 can't directly time the IR pulses in firmware, no matter how clever the programmer.

However, because the input signal is a single bit, we can use an external shift register to extend the timing capability of the 8031. By shifting the input signal into an 8-bit register which is clocked at 1 MHz, each bit represents a one-microsecond sample of the signal. The 8031 then reads the shift register through a parallel port every eight microseconds. Now, instead of an impossible timing constraint, we are left with a strict but manageable requirement.

Schematic 1 shows the IR signal processing hardware. The photodiode converts the IR signal to an analog voltage which is proportional to brightness. The LM311 comparator compares that signal to a 2.5-volt threshold reference set by a simple resistor divider. When the signal exceeds that threshold, the LM311 produces a TTL output indicating that a suitable IR signal is present. The LM31 l's output is



Schematic 1

connected to a 74LS164 eight-bit shift register, which in turn is read through the 8031 P1 port.

The IR Master doesn't need elaborate amplifiers and gain controls in its receiver section. Unlike the CD player or TV set, IR signals for the Master Controller come from remote units which are held within a foot or so of the photodiode during training. The receivers in commercial remote control units use esoteric amplifiers, modulated carrier frequencies, and elaborate message structures to punch the signal across a room full of interference. Mercifully, none of that applies here!

The 8031 produces one pulse on the ALE (Address Latch Enable) output for every six clock pulses, or at 2 MHz for a 12-MHz crystal. A simple flip-flop divider produces a precise l-MHz signal to drive the shift register in sync with the 8031's clock oscillator. The shift register thus contains the last eight microseconds of IR input data, sampled at each rising edge of the 1 -MHz clock.

There is a problem, however. The IR Master stores signal timing data and menu text in 32K of RAM, which is referred to as External Data Memory. The ALE signal is produced at 2 MHz except during external data memory reads or writes. These operations take two cycles and omit the ALE pulse during the second cycle. The 1-MHz signal will slip by one microsecond every time the firmware attempts to record its measurements!

Although this sounds catastrophic, the solution is simple: the firmware cannot read or write External Data RAM while it expects good data from the shift register. If this sounds contradictory, you're not used to firmware yet!

The trick is to measure the carrier signal and the data bursts in two separate steps using two distinct routines, each optimized for a particular purpose.

Getting the Carrier

Recall that because the carrier period and duty cycle are constant throughout a given message, the firmware can measure any part of any burst with confidence that the values are representative of the entire message. There's not enough time to extract the values on the fly, but only a few cycles suffice for the whole message.

The carrier timing subroutine records 32 bytes of sampled IR data in the 8031's Internal Data RAM. With no references to External Data Memory during the subroutine, ALE is a stable 2 MHz and no samples are "dropped" along the way. Those 32 bytes thus represent 256 continuous microseconds of IR signal, stored without analysis. Hidden within the samples will be 7 to 12 complete carrier signal cycles.

Once the samples are stashed in Internal Data RAM the firmware can analyze them and store the results in External Data RAM. The shift register isn't used when the results are stored, so there's no conflict.

Figure 4 shows the relationship between the IR input signal and the data read from the shift register. There is no chance of reading the shift register during a shift because the read occurs near the falling edge of ALE. Notice that the leftmost data bit (bit 7) is the "oldest" sample, so the signal reads from left to right in the byte.

GETPULSES, shown in Listing 1, is the 8031 subroutine that collects the IR signal data. Each instruction in the time-critical section is marked with its starting time relative to the shift register reads as well as its duration.

A brief description of 8031 assembly language is in order before we walk through the code in detail. The 8031, unlike more complex microprocessors, can test nearly any bit of any register, so the statement

JB A.7,WAIT1

indicates a test on Accumulator bit 7 and a branch to location WAIT1 if that bit is set to "1". A JNB instruction branches if the bit is not set.

A pound sign (#) denotes an immediate data value; thus

MOV B,#nsamples-2

loads the B register with 30 when nsamples (number of samples) is 32. An instruction without the pound sign treats the number as an Internal Data RAM address, so

MOV B,nsamples-2

loads B with the contents of RAM location 30. As you might expect,

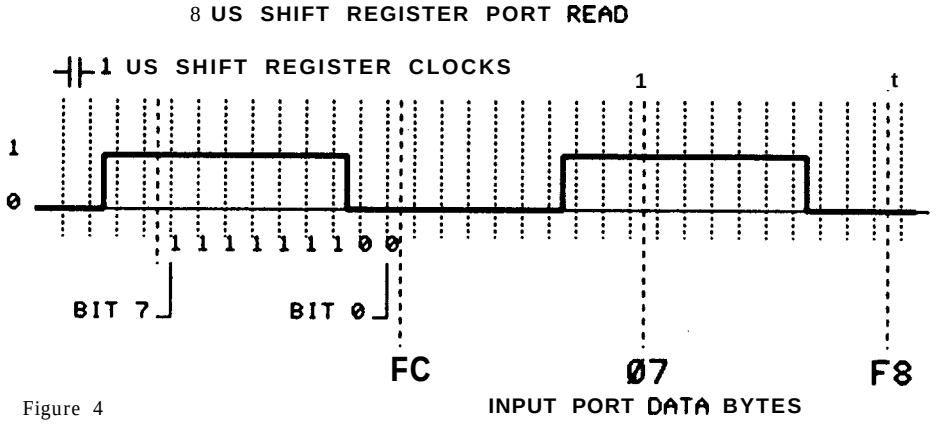


Figure 4

```
Listing 1 -- IR Carrier Sampling

;----
; Accumulate samples in internal RAM
; Assumed to be between 30-50 KHz, thus 33 - 20 us period
; Samples from P1 must be spaced at 8 us intervals exactly
; ALE cannot be interrupted, thus no external data references!

getpulses  EQU $
           PUBLIC  getpulses

           MOV     RO,#samples  ; internal RAM pointer

;--- wait for a sample that has a rising edge
; These samples don't have to be synchronized with shift reg

,           start of instruction -||- instruction timing, us

wait1      MOV     A,P1         ; 0 1 get sample
           NOP                  ; 1 1 stay in sync
           MOV     @RO,A        ; 2 1 save for later
           JB      A.7,wait1    ; 3 2 reject if starts high
           JNB     A.0,wait1    ; 5 2 reject if ends low

;--- take the remaining samples into storage

           NOP                  ; 7 1 get into sync with shift reg
           MOV     A,P1         ; 0 1 get next sample
           MOV     B,#nsamples-2 ; 1 2 have two samples already

taker      INC     RO           ; 3 1 tick pointer
           MOV     @RO,A        ;4 1 save previous sample
           NOP                  ; 5 1 stay in sync with shift reg
           NOP                  ; 6 1 stay in sync with shift reg
           NOP                  ; 7 1 stay in sync with shift reg
           MOV     A,P1         ; 0 1 get the next sample
           DJNZ    B, taker     ; 1 2 continue in loop

           INC     RO           ; save the last sample
           MOV     @RO,A

           RET
```

omitting a # causes baffling errors.

An "at" sign (@) means that the contents of the register is used as an address, so

MOV @R0,A

takes the contents of RO as an Internal Data RAM address and moves the Accumulator to that location. This is called an "indirect" move. The contents of RO and A do not change.

Finally, the CJNE (compare and jump if not equal) instruction compares its first two operands and branches to the third operand address if they're not equal. The 8031 does not include the corresponding CJE instruction, which often leads to tangled coding with jumps around jumps.

GETPULSES starts with a loop at WAIT1 to sample the shift register. The code is looking for the rising edge of a carrier pulse. The edge must occur within one sample, so the byte must have bit 7 = 0 and bit 0 = 1. Successive samples are written into the same Internal RAM location every five or seven microseconds until an acceptable edge shows up.

The samples at WAIT1 don't have to be taken every eight microseconds because the firmware is looking for the start of a pulse. But when a sample passes the tests, microseconds suddenly become critical!

It takes seven microseconds to verify the first sample. The NOP instruction following the last test accounts for the eighth microsecond, and the MOV A,Pl grabs the second sample on schedule. The code loads the remaining count (30 bytes) into the B register and the TAKER loop gets control three microseconds after sampling.

TAKER looks like a backwards loop. It starts by incrementing the target address in RO, stores the accumulator into RAM at the address in RO, gets the next sample at the bottom, and then loops. The reason it's written this way is to match timing with the instructions after **WAIT1 .** There isn't much choice in the matter.

More advanced than microprocessors, the 8031 "help" your program by prefetching and overlapping instructions, with the result that you cannot count cycles to determine the timing relation between two instructions. Even the lowly 8088 has an instruction prefetch queue and a small amount of overlap;when you get to the 80386 it's nearly impossible to figure out what's going on.

Once all 32 samples are stored in RAM, a routine called TALLY analyzes the data. Each carrier cycle is composed of two phases: "up" when the IR signal is on, and "down" when the signal is off. The number of bits in all "up" phases divided by the number of cycles gives the average duty cycle in microseconds per cycle. Similarly, the total number of "up" and "down" bits divided by the number of cycles gives the average period in microseconds per cycle. Because only complete cycles contribute bits to the totals, the averages do not include extraneous data at the beginning and end of the samples.

Listing 2 shows TALLY's logic, which is perhaps more complex than you might expect at first. A pair of loops (UPCOUNT and DOWNCOUNT) count successive "1" and "0" bits, passing control

back and forth as they work through the samples. Each loop fetches a new sample byte if it consumes all eight bits of the previous one. The two loops continue until all 256 sample bits are accounted for.

The totalizing variables receive new values at the beginning of each carrier cycle, when the first "1" bit of a new cycle appears. This ensures that only complete cycles contribute to the totals. A partial cycle at the end of the data is simply ignored, regardless of whether it ends with "1" or "0" bits; there is no transition from "0" to "1" to trigger a count update.

TALLY also computes the minimum and maximum number of "1" and "0" bits found in any cycle. If these numbers are within one count of the average values the entire sample is considered good. Samples that fail the test are rejected; the IR signal must be digitized again.

The end result of all this work is a pair of numbers: the IR carrier period and duty cycle expressed in microseconds. After storing these in External Data RAM, the code proceeds to the next step: timing the bursts making up the message.

First with the Burst

GETPULSES got away with storing samples in Internal Data RAM because 32 bytes is enough to completely characterize the IR carrier. A different method is required for the data burst measurements because messages can contain (many) dozens of bytes and last for hundreds of milliseconds. The timing must be accurate to a few microseconds, but recording one sample every eight microseconds could produce tens of thousands of data bytes.

The trick here is realizing that the IR bursts are at least a few

Listing 2 -- IR Carrier Analysis

```
;----
; Analyze pulses and return counts

tally       EQU     $
            PUBLIC  tally

            MOV     upsum,#0            ; initialize counters
            MOV     upmin,#254
            MOV     upmax,#0
            MOV     downsum,#0
            MOV     downmin,#254
            MOV     downmax,#0
            MOV     pulses,#0

            MOV     R0,#samples         ; point to first sample
            MOV     R1,#1               ; count samples

            MOV     A,@R0               ; get first sample
            INC     R0                  ; point to next one
            MOV     B,#8                ; bit counter
getrisel    JB      A.7,rise1           ; '1' bit is rising edge
            RL      A                   ; move next bit in
            DJNZ    B,getrisel
                                        ; B=0, this is error, ignored
rise1       EQU     $

;----
; Sample is positioned with first 1 bit in A.7
; B has number of bits left in A
; Count number of 1 bits left in A, add to upsum at end of pulse
; When B becomes zero, get a new sample and tick sample counter
; When sample counter hits zero, all samples are done
; When a zero bit occurs, it's the end of the up phase

newpulse    EQU     $

            MOV     R2,#0           ; total up bits this pulse

upcount     EQU     $
            JNB     A.7,endup       ; found end of pulse
            INC     R2              ; count this bit
            RL      A               ; shift in the next one
            DJNZ    B,upcount       ; step through the byte

;---B is zero, need another sample
;    first check to see if we're done

            CJNE    R1,#nsamples,ct1OK
            SJMP    ctdone

ct1OK       INC     R1              ; tick sample counter
            MOV     B,#8            ; reset bit counter
            MOV     A,@R0           ; pick up new sample
            INC     R0              ; tick sample pointer
            SJMP    upcount

;--- come here when we find a 0 bit in A.7

endup       EQU     $

;----
; Sample is positioned with first 0 bit in A.7
; B has number of bits left in A
; Count number of 0 bits left in A. add to downsum
; When B becomes zero, get a new sample and tick sample counter
; When sample counter hits zero, all samples are done
; When a one bit occurs, it's the end of the down phase

            MOV     R3,#0           ; total down bits this pulse
```

(continued on page 38)

```
(continued from page 57)
dncount     EQU    $
            JB     A.7,enddn       ; found end of pulse
            INC    R3              ; count this bit
            RL     A               ; shift in the next one
            DJNZ   B,dncount       ; step through the byte

;--- B is zero, need another sample
;    first check to see if we're done

            CJNE   R1,#nsamples,ct0OK
            SJMP   ctdone

ct0OK       INC    R1              ; tick sample counter
            MOV    B,#8            ; reset bit counter
            MOV    A,@R0           ; pick up new sample
            INC    R0              ; tick sample pointer
            SJMP   dncount

enddn       EQU    S

;--- come here when we find a 1 bit in A.7
;    the pulse is complete, so update totals & averages
;    go back to 1 loop above
; Note that there are only 256 bits in 32 bytes, so...
;  none of the additions can generate a carry!

            PUSH   A               ; save sample bits
            INC    pulses          ; tick pulse counter

;--- up phase of pulse

            MOV    A,upsum         ; get up count
            ADD    A,R2            ; add for this pulse
            MOV    upsum,A         ; return to storage
            MOV    A,upmax         ; max up time so far
            SUBB   A,R2            ;  upmax - upcount - 0
            JNC    upmaxOK         ; no carry -> max > count
            MOV    upmax,R2        ; set new high count
            CLR    C               ; reset carry

upmaxOK     MOV    A,upmin         ; min up time so far
            SUBB   A,R2            ;  upmin - upcount - 0
            JC     upminOK         ; carry -> min < count
            MOV    upmin,R2        ; set new min count
upminOK     EQU    S

;--- down phase of pulse

            MOV    A,downsum       ; get down count
            ADD    A,R3            ; add for this pulse
            MOV    downsum,A       ; return to storage

            MOV    A,downmax       ; max down time so far
            SUBB   A,R3            ;  downmax - downcount - 0
            JNC    dnmaxOK         ; no carry -> max > count
            MOV    downmax,R3      ; set new high count
            CLR    C               ; reset carry

dnmaxOK     MOV    A,downmin       ; min down time so far
            SUBB   A,R3            ;  downmin - downcount - 0
            JC     dnminOK         ; carry -> min < count
            MOV    downmin,R3      ; set new min count
dnminOK     EQU    S

            POP    A               ; restore sample bits
            SJMP   newpulse        ; begin counting the 1 bits

;-----
; All sample bytes processed, so we're done with counting
; The partial bit counts accumulated up to this point are discarded
```

(continued on page 39)

hundred microseconds long. Because the burst will not start or stop immediately after an edge occurs, the firmware can locate the edges of the burst with considerable precision, then take a relatively long time processing and recording the data. Best of all, there is no need to write External Data RAM during the edge timing loops, so the ALE glitch problem simply goes away, along with constantly monitoring the shift register.

The 8031 includes two on-chip timers that can count up to 64K microseconds. The firmware simply notes the timer reading at the beginning and end of each burst. The end of the message occurs when the timer has gone through four complete 64K count sequences, about 1/4 second from the start of the first data burst.

It's easy to find the start of a burst -- simply look for the first "1" bit in the input data. But when does a burst end? An LED transmitting a 30-kHz IR carrier with a 40% duty cycle will be off for 20 microseconds at a time. This translates into at least two completely zero sample bytes. The solution requires that we define four all-zero samples at the end of a burst as "the end," and then reduce the timer reading to compensate for the overshoot.

The final step processes the recorded data to find the duration of each burst and gap by subtracting successive time stamps. The final result is a pair of integers for each burst. A message with 30 bursts is completely described by only 120 bytes: 30 bursts x 2 integers per burst x 2 bytes per integer. Compare that with the tens of thousands of bytes created by the brute force digitizing!

Listing 3 shows the code for GETBURSTS, the subroutine that handles burst timings. It uses two

linked loops in much the same fashion as GETPULSES, with one loop timing the burst and the second timing the gap.

The instruction

leader JNB P1.0,leader

is a single instruction that branches to itself if bit 0 of port P1 is a zero. Because P 1 .O reads the first bit of the shift register, this is a two-cycle test for the beginning of a burst: PI.0 is "1" immediately after the IR signal becomes active.

The next instruction starts Timer 0, which was preloaded to compensate for three microseconds of delay. Because the input may become active just before the JNB tests it the correction may be off by one microsecond, but that's close enough for burst measurements.

Although GETBURSTS uses Timer 0 to measure time intervals, the shift register sampling loops must still maintain the 8-microsecond sequence to ensure that each test operates on fresh, valid data. The loops are complicated by tests for timer wrap and the quarter-second final timeout.

The code illustrates a useful technique for time-critical sampling: if the interval between samples is too short for any useful work, you may be able to combine two intervals and get something done. For example, the loop at BURSTING is two samples long and accommodates all of the required tests with a whole microsecond to spare.

The test at BEND ("burst end") counts all-zero samples at the end of a burst. If a nonzero sample shows up, the code rejoins the burst timing loop in exact sync with the shift register inputs. The NOPs provide delays to keep everything in step, so they're not really No Operation instructions.

Extracting the timer value "on the fly" presents a problem. The

'continued from page 88)

```
ctdone      EQU    $
;----
; Compute the averages based on the number of bits and pulses

            MOV    A,upsum            ; find average up time
            MOV    B,pulses
            DIV    AB
            MOV    upavg,A

            MOV    A,downsum          ; find average down time
            MOV    B,pulses
            DIV    AB
            MOV    downavg,A

            MOV    A,upsum            ; find average period
            ADD    A,downsum          ;  max 255 bits!
            MOV    B,pulses
            DIV    AB
            MOV    pdavg,A

            RET
```

Listing 3 -- Burst Timing Measurement

```
;----
; Analyze IR bursts in real time
; Stores burst start and end times in external data RAM
;  DPTR points to start of RAM data buffer
; A,B = last address + 1 for burst data (high/low)
;    must be multiple of 4 bytes from starting address
;    samples are stored as start time, end time pairs
;    high byte first in each item
; DPTR returned pointing to last byte + 1
;    unchanged from entry values if RAM is full...
; Uses Reg Bank 1

getbursts  EQU  $
            PUBLIC  getbursts

            SETB  RSO              ; select reg bank 1 (or 3!)
            MOV    R2,A            ; set up pointer
            MOV    R3,B

            PUSH   DPH             ; save RAM pointer for fixup
            PUSH   DPL

;--- set up timer 0 to start when burst is detected

            CLR    TRO             ; stop timer 0
            CLR    TFO             ; reset overflow flag

            MOV    TH0,#00H        ; reset timer 0
            MOV    TL0,#03H        ;  to start at 3 microseconds

            MOV    R1,#maxtime     ; set wrap counter
            MOV    B,#burstend-1   ; # zero bytes at end - 1

;--- wait for start of burst
;    turn timer 0 on when first bit is found
;    may be off by 1 microsecond due to jump timing
;    add two microseconds due to SETB timing

leader      JNB    Pl.O,leader     ; sample 1st shift reg bit

            SETB   TRO             ; start timer

;--- burst in progress, wait for end...
;    need 32 us without any signal
```

(continuedon page 40)

(continued from page 39)

```
;     samples must be taken 8 us apart in sync with shift reg
;     check for timer overflow and tick wrap counter
;     if 256K us have transpired, bail out

;            start of instruction -| |- instruction timing, us
bursting    MOV     A,P1            ; 0 1  get sample
            JZ      bend            ; 1 2 decide if all zero

            JNB     TF0,T0novl      ;3 2  check for timer 0 overflow
            CLR     TF0             ; 5 1  yes, reset overflow flag
            DEC     R1              ; 6 1  tick wrap counter
            NOP                     ; 7 1  stay in sync
            MOV     A,P1            ; 0 1  get sample
            JZ      bend            ; 1 2  decide if all zeros
            MOV     A,R1            ; 3 1  check if 256K us is up yet
            JZ      timeupb         ; 4 2  if yes, bail out
            SJMP    bursting        ; 6 2  if not, keep testing

T0nov1      NOP                     ; 5 1 keep in sync
            SJMP    bursting        ; 6 2 dive back into loop

;--- detected one zero sample, so begin tests for next few
;     B contains count for required number of zero samples

bend        NOP                     ; 3 1  stay in sync
            NOP                     ; 4 1
bend5       NOP                     ; 5 1
            NOP                     ; 6 1
            NOP                     ; 7 1
            MOV     A,P1            ; 0 1  get sample
            JZ      bendcount       ; 1 2 if zero, there's hope for
                                          end
            MOV     B,#burstend-1   ; 3 2 alas, reset zero detect
                                          counter
            NOP                     ; 5 1
            SJMP    bursting        ; 6 2  rejoin the top loop

bendcount   DJNZ    B,bend5         ; 3 2 if zero, tick zero counter

;--- got the right number of zero samples, burst is over
;     grab current count from timer 0
;     ignore error from fixing phase error between bytes
;     subtract 40 us to correct for overcounting zero bytes
;     save time in RAM at current pointer, low byte first
;     don't need to sample shift reg for a while...
;     leading edge of next burst defines next sync start

reget       MOV     A,TH0           ; get high byte
            MOV     B,TL0           ; get low byte
            CJNE    A,TH0,reget     ; ensure valid sample

            XCH     A,B             ; get low byte in A
            CLR     C               ; knock off overcount
            SUBB    A,#40
            XCH     A,B             ;    ... and high byte
            SUBB    A,#0
            MOVX    @DPTR,A         ; stash high byte
            INC     DPTR
            XCH     A,B             ;    ... and low byte
            MOVX    @DPTR,A
            INC     DPTR
;--- now wait for the start of the next burst
;     record starting time of burst as end of gap
;     time out if more than 256K us has elapsed

burst2      MOV     A,P1            ; 0 1 pick up a sample
            JNZ     newburst        ; 1 2 decide if gap is over
            JNB     TF0,T0nov2      ; 3 2 check for timer 0 wrap
            CLR     TF0             ; 5 1 reset if so
            DEC     R1              ; 6 1 tick wrap counter
```

(continued on page 41)

8031 must read the two timer bytes with two separate instructions, so it's possible for the timer to change between them. For example, suppose the code finds FF in the low byte and 23 in the high byte. The correct value might be 22FF, 2300, or 23FF, depending on the order in which the bytes are read and whether a count occurred between the reads.

The code at REGET and NEWBURST makes sure that the high timer byte is unchanged after a read, which will be true if the timer did not count during the read. If the high byte changes, the code simply rereads the timer.

The timer value at the end of a burst is too high by about 32 microseconds because of the four all-zero samples at the end of each burst. An additional error arises because the code does not attempt to locate the burst's exact trailing edge within the final nonzero sample. Reading the timer occupies a few more cycles, so the code subtracts 40 microseconds from the timer value to compensate for most of these.

The maximum timing error for any one burst is about 8 microseconds, which is close enough for events that are usually a few hundred microseconds long. In any event, this is much less than half a cycle of the IR carrier signal, so there is little benefit from attempting to remove all of the errors.

The duration of the gap after a burst is set by the start of the next burst, so the adjusted timer values for a burst and gap are stored just after the next burst begins. The shift register can be safely ignored while the data writes into External Data RAM glitch ALE.

A message can end in one of two ways: exceeding 256 milliseconds or reaching the end of External Data RAM. Most messages are

much less than 256 milliseconds long, so the most common ending is a timeout while waiting for a burst that never arrives. If, instead, a burst is in progress when the loop times out, the code discards that burst and begins fixing up the stored timer values.

Although the timer may have wrapped three times during the message, simply subtracting successive values will always return the relative time between them. The only tricky part is doing a 16-bit subtraction in an 8-bit machine. The code works through all of the values stored in External Data RAM, then returns.

The Bottom Line

The IR Master Controller, using just the simple hardware and firmware in this article, can record and play back the signals from all of the IR remote units we've been able to track down. This is particularly interesting because of the stringent timing requirements that, at first, seem to rule out using a "slow" microprocessor for real- time data collection.

In fact, it's been able to cope with some encoding schemes we never expected. One remote even dispensed with the IR carrier signal entirely: it just turned the LED on for the entire duration of the "1" bits. How the firmware sorted that out, I don't even want to know!

The techniques of cycle-counting I/O synchronization are most useful with microcontrollers that do not overlap and prefetch instructions, but if you're using a PC to collect data, you may find the interleaved loops a useful way to analyze the incoming signals. There's obviously an upper limit to the bandwidth you can handle with a micro, but 1 megabit/second may be good enough! ■

(continued from page 40)

```
            NOP                 ; 7 1 stay in sync
            MOV     A,P1        ; 0 1 pick up next sample
            JNZ     newburst    ; 1 2 decide on gap ending
            MOV     A,R1        ; 3 1 check on 256 K us limit
            JZ      timeupg     ; 4 2 and bail out if so
            SJMP    burst2      ; 6 2 get back into the loop

T0nov2      NOP                 ; 5 1 stay in sync
            SJMP    burst2      ; 6 2 rejoin the loop

;--- found a burst, so grab the current count from timer 0
;    ignore few microseconds that the time is off
;    don't have to sample in sync right away

newburst    MOV     A,TH0       ; pick up high byte
            MOV     B,TL0       ; and low byte
            CJNE    A,TH0,newburst     ; make sure they're OK

            MOVX    @DPTR,A     ; store high byte
            INC     DPTR
            MOV     A,B         ; . . . and low byte
            MOVX    @DPTR,A
            INC     DPTR

;--- check to make sure we're not at the end of RAM

            MOV     A,DPL       ; do DPTR - end limit
            CLR     C
            SUBB    A,R3
            MOV     A,DPH
            SUBB    A,R2
            JNC     endRAM      ; no carry -> DPTR = end limit

;--- all clear, rejoin burst loop

            MOV     B,#burstend-1 ; set burst ending counter
            SJMP    bursting

;--- come here when 256K us have expired during burst
;    this "should not happen", but if it does we
;    discard this burst...
;    DPTR now points to the place we would have put the
;    starting time, so it's OK to just return

timeupb     KQU     $
            SJMP    setend
;--- come here when 256K us have expired during gap
;    this will happen after the last burst
;    DPTR now points to the place we would have put the
;    ending time, so add a fake gap & call it quits

timeupg     KQU     $

            MOV     A,#FFH      ; FFFF is as good as any
            MOVX    @DPTR,A
            INC     DPTR
            MOVX    @DPTR,A
            INC     DPTR        ; point beyond end

;--- come here from either timout ending

setend      nov     A,DPH
            MOV     R2,A
            MOV     A,DPL
            MOV     R3,A
            SJMP    endRAM

;--- come here when RAM is filled up

endRAM      EQU     $
```

(continued on page 42)

(continued from page 41)

```
            CLR     TR0             ; turn off timer 0

;--- change the absolute time data to relative times

            POP     DPL             ; get starting RAM pointer back
            POP     DPH

            MOV     R6,#0           ;set up starting time
            MOV     R7,#0

fixit       MOV     R4,DPH          ; save current RAM address
            MOV     R5,DPL

            MOVX    A,@DPTR         ; get high byte of time
            INC     DPTR
            PUSH    A               ; save for later
            MOV     B,A
            MOVX    A,@DPTR         ; get low byte
            PUSH    A               ; save for later

            CLR     C               ; time - start (low)
            SUBB    A,R7
            MOVX    @DPTR,A         ; save fixed low byte

            MOV     DPH,R4          ; get first RAM address back
            MOV     DPL,R5

            MOV     A,B             ; get high byte back
            SUBB    A,R6            ; time - start (high)
            MOVX    @DPTR,A         ; save fixed high byte

            POP     A               ; get low byte back again
            MOV     R7,A            ; remember for next loop
            POP     A               ;   ... ditto for high byte
            MOV     R6,A

            INC     DPTR            ; step to next pair of bytes
            INC     DPTR

            MOV     A,DPL           ; compare pointer with last byte
            CLR     C
            SUBB    A,R3            ; DPTR - end address
            MOV     A,DPH
            SUBB    A,R2

            JC      fixit           ; carry -> more to do

            CLR     RS0             ; return to Reg Bank 0

            RET
```

Ordering Information for Home Satellite Weather Center (from page 27)

Bare Circuit Board	$18.00
Pre-Programmed 27C256 EPROM	$10.00
TOKO Transformer	$2.75
TOKO Delay Line	$9.75
Oscillator module	$7.40
Kit of partslisted above	$42.50
Complete kit of all parts to build the RGBI to NTSC Converter	$91.00

Add $2.00 Postage and Handling for each plied order.

Sendcorrespondence 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 supwith your order, as well as being noted in a future installment.

Personal-Computer-Based Instrumentation

Build a 4-Channel Temperature Logging and Data Reduction System

by Tom Riley

We rely on our personal computers for a great many tasks. With the device constructed from this article, your personal computer can also be turned into a powerful tool for doing many scientific experiments. For about $35.00, this device will let your computer read and log four independent temperatures. The resulting data is useful to students and amateur scientists but it can also be used to evaluate a home heating system or even control an industrial process.

The device in question is called the Quadtherm and its construction and electronic circuitry are quite simple. The supporting software, however, is considerably more sophisticated for a reason. This project goes beyond a simple construction article and is presented instead as an introduction to a new and growing interest area: Personal-Computer-Based Instrumentation.

A number of personal instrumentation systems are commercially available for the Apple and IBM PC. Some are bare boards with a specification sheet or two, while others are full-blown systems with hardware, software, and full documentation. The latter, of course, have much higher price tags. The Quadtherm is an inexpensive starting point, yet one which has practical applications.

A personal instrument system is more than hardware attachments. It must include supporting software for every step of the experimental process. A good personal instrument system must aid the user in the design of the experiment, the calibration of the sensors, the logging of data, the reduction of that data into comprehensible form, the writing of the report, and the generation of supporting graphics. At every step along the way it must also encourage good scientific method.

Simple Hardware -- Comprehensive Software

The Quadtherm's sensors consist of four tiny thermistors about the size of sesame seeds. These devices are ceramic resistors which change their resistance exponentially with applied temperature. The Quadtherm's operating range is selected and automatically calibrated through a program-initiated step-by-step procedure which gives it an accuracy equal to temperature-measuring devices that cost an order of magnitude more.

Quadtherm can be easily adapted to any personal computer which has timer-type paddle inputs. The prototype was tested on an Apple][+ but Atari, Commodore, and IBM all use similar systems with only -slight changes in component values and cable connectors. The software can be easily adjusted to different machines by entering the correct parameters in Program #1.

The Quadtherm software is provided primarily as a learning tool as well as an introduction to personal instrumentation. I invite you to modify the programs to suit your own needs. The programs are written in structured BASIC (as discussed by Arthur Luehrmann in the May-July 1984 issues of Creative Computing) and can be easily translated for use on any of the above-mentioned computers.

To further simplify the task, the programs are broken down into self-contained blocks which have only one entrance and one exit. The main program merely functions as an outline giving the order in which these blocks are to be executed.

The experiment I've outlined to demonstrate the Quadtherm measures the temperature of key components inside an Apple][+ (with and without a fan) and the room air temperature. The expected temperature range is about 76°F (22°C) 200°F (90°C), measured with a resolution of 1.5°F (1.0°C). With these measurements we will ascertain whether using the fan reduces the likelihood of component failure.

There are five BASIC programs used in this project which select and calibrate the thermistors and then display their readings. Unfortunately, these programs are much too long to publish as listings here. Instead, the programs can be downloaded from the Circuit Cellar BBS (see the ConnecTime section). The programs *are* referenced *in the article either by number or by a descriptive title indicating their function.*

Choosing the Thermistors

The thermistors I used were small glass-coated beads made by Fenwal Electronics (see parts list). I tried cheaper thermistors without the glass coating, but their readings were affected by moisture even when the units were covered with epoxy. Certainly, thermistors from other manufactures and of other designs will work. They should, however, be single thermistors intended for temperature measurement.

Program #1, Choosing Thermistors, selects thermistors for a specific temperature range using a manufacturer's catalog. First, choose a thermistor from the catalog that you guess is about right, run Program #1, and answer the questions. The calculated parameters Beta and Alpha are used to compare various thermistors.

Thermistors

Thermistors are nonlinear resistors. They exhibit their highest resistance at low temperatures. This resistance decreases exponentially as the thermistor's temperature increases.

When read by the computer game port, there is a temperature below which the change in resistance can no longer be read. Interestingly, though, a thermistor's point of greatest resolution is just above this low limit. A thermistor has no high limit except that defined by its packaging and cabling.

Like the potentiometer in a joystick, the thermistor functions as the resistance element in an RC (resistor/capacitor) timing circuit of the computer's game port. Basically, the game port circuit is nothing more than an NE555-based low-frequency oscillator whose period is set by an internal capacitor and an external resistor. By measuring the period of the oscillator, the computer can empirically determine the attached resistance.

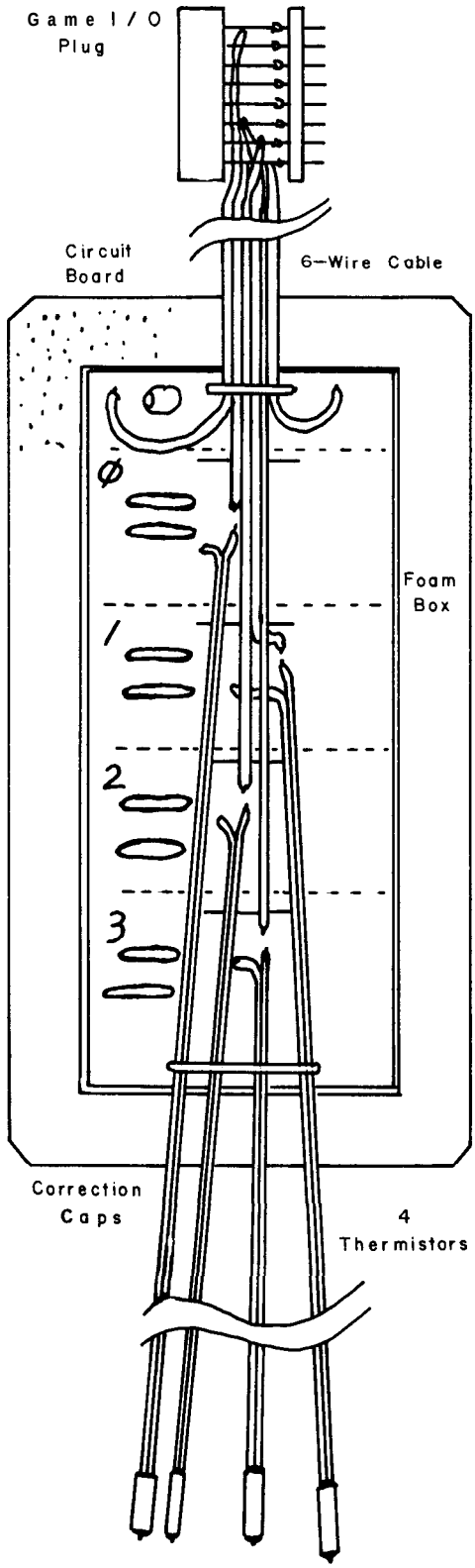


Figure 1 - Quadthermometer Prototype constructed on perforated PC stock and mounted in an insulating foam enclosure.

If you are using an Atari or Commodore computer, the value for the paddle resistance (Program #1, step 105) should be RX = 1000000 and the internal capacitor (step 110) CN = 0.01. IBM PCs use RX = 100000 and CN = 0.01. The Apple][+ computer has an RX = 150000 and an internal capacitor CN = .022 (these default values are preset in the program). Some thermistor manufacturers use different reference temperatures for the resistance ratio so be aware of catalog entries regarding this.

The game port functions as an 8-bit analog-to-digital converter. The computer samples the period of the game port oscillator and returns an 8-bit "pot position" value (0-255 decimal) to the application program. A pot position of "0" indicates minimum resistance while "255" is maximum. A thermistor at its minimum operating temperature will display a maximum output reading. Depending upon the characteristic dR/dT of the thermistor, the minimum output count should correspond to the maximum desired temperature.

While the basic paddle port connection makes use of the internal timing capacitor on the game port, to better adjust the frequency range of the game port oscillator for a selected thermistor, additional trimming (or correction) capacitors are attached in parallel with the thermistor. With the thermistor set at its low-temperature operating point, the correction capacitor(s) are added so that this minimum temperature will read out as a pot position of about 250 (it should not be set for 255 because there is no way of knowing when it is beyond the adjustment point).

Accuracy and resolution depend upon the applied temperature. Just above the low-temperature limit, 0.1 °C of resolution and accuracy are possible. At 80°C or higher, 5°C resolution is all that can be expected. As a result, you must carefully choose the low temperature as close to the expected experimental low as possible, but not so close as to lose data. The combination of thermistor parameters and correction capacitor value set this low limit.

You may either give Program #1 a value for the correction capacitor or give the low-temperature limit and the correct capacitor will be calculated.

A chart of the resistance and potentiometer reading versus temperature is then calculated. This chart will give you a good idea of the performance of your chosen thermistor and correction capacitor over the range of the experiment. Note the change in resolution (i.e., number of pot reading steps per degree) over the temperature range.

I suggest that you try several thermistors to find one requiring only a small correction capacitor for your temperature range. You can expect that the numbers on the data sheet are only accurate to within about 10%. Also, it is best to plan on at least a O.Ol-microfarad correction capacitor to give yourself some adjustment capability.

Constructing Quadtherm

The construction of the Quadtherm hardware is straightforward. Figure 1 shows the main parts: a game I/O plug, connecting cable, circuit board, and thermistors.

The circuit board is 1/3 of a general-purpose predrilled PC board from Radio Shack. The area for the correction capacitors is designated but nothing should be installed initially. The exact capacitors required will be determined in an automatic calibration test later. The foam block around the board is a single piece of shipping foam, flexible but stiff, which is held with rubber bands. The foam protects the board and helps keep the correction capacitors at a stable temperature.

The cables can be any wire size that will mechanically survive your experiment. The prototype uses #24 wire. The main cable needs a minimum of six wires, but you may wish to add more if your application has use for pushbuttons and annunciators. All the cables on the prototype were 4 feet long since it was intended for use only on a workbench. Longer cable lengths will work just as well but I would not leave them permanently attached since they also serve as antennas where electrical noise can enter the computer.

The only difficulty encountered in constructing the thermistor sensors was due to the very small size of the devices. Figure 2 details the assembly procedure. First separate, trim, strip, and tin the cable wires. Place a 1/4-inch piece of heat-shrink tubing over the long wire. Wrap either lead of the thermistor around this wire and solder. Trim off the extra wire. A pair of pointed tweezers is very helpful at this point. Shrink the tubing with either a small flame or a heat gun.

The second solder joint is then made and the outer heat-shrink tubing installed. A small amount of epoxy seals the end of the tube, but the tip of the thermistor is allowed to stick out. The outer heat-shrink tubing may be any length from 1/4 inch to several inches to serve as a handle. As an alternative, you can mount the thermistors in metal or glass tubes.

Figure 3 shows the schematic of Quadtherm. C4 is included to reduce electrical noise. The Apple][+ game port connector is a 16-pin DIP header. For computers other than the Apple][+, you need only change the type of connector and the pin numbers.

The exact value needed for each correction capacitor is usually easier to determine by connecting several small capacitors in parallel. Allow some extra room on the circuit board. The capacitors themselves will be installed during the calibration procedure.

Correction Capacitor Adjustment

Using just catalog specifications, the absolute accuracy of the thermistor and the game port timing circuit is hardly worth mentioning (about 10%). However, when properly calibrated, the ac-

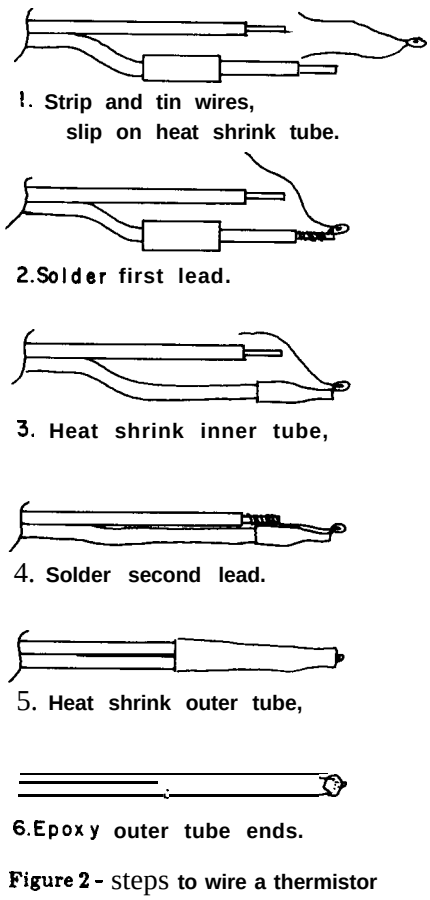


Figure 2 - steps to wire a thermistor

curacy and repeatability of any measurement is substantially higher. By calibrating the thermistor and computer together as a system using a known accurate reference, we can achieve measurement accuracies of 0.5%.

Low-temperature calibration and correction-capacitor adjustment are done at the same time. The calibration procedure uses a Thermos bottle of cold water and an accurate reference thermometer. The thermometer I used was a Fisher Scientific 14-983-10B glass bulb costing about $20.00. In the calibration procedure this thermometer sets the precision of the finished thermistor sensors. The computer will be able to provide no greater accuracy than this reference standard so you want to use a good thermometer. The correction-capacitor adjustment and calibration is done on the computer by running Program #2.

I attached the sensors to the thermometer with a rubber band and wrapped the assembly in plastic film closed with another band to keep out the water. I then placed them in the Thermos bottle and adjusted the water temperature by adding warm and cold water until the thermometer read 2 degrees below the lowest temperature of interest for my experiment, 76°F (20°C). I waited 10 minutes to be sure the temperature stabilized and then ran Program #2.

Next, turn the computer off and disconnect the Quadtherm I/O cable. Using the information derived from Program #2, select correction capacitors which are just a bit smaller then the values suggested by Program #2. Install these trim capacitors in the designated locations.

Check your bath temperature and run Program #2 again. If the results are optimum, you can stop with a single correction capacitor. If the setpoint is still not correct, you should repeat the same procedure and add additional smaller capacitors to the sensors which need them. These capacitors do not affect the accuracy of the finished Quadtherm. They merely set its low-temperature setpoint.

Calibration

The final calibration procedure is a bit involved, but accuracy always takes work. Program #3 is the calibration program. It guides you through the calibration procedure, then calculates calibration parameters for each thermistor and stores them on the disk.

If you are using a computer other than an Apple][+, you will need to rewrite the disk storage routine (subroutine 1400) here and in Programs #4 and 5. The calibration file consists of: (1) a description of all the thermistors with the calibrator's name and the date, (2) the game I/O reading at 25°C, and (3) the Beta parameter for each of the four sensors.

Instructions are given in the program, but let me describe the process. The procedure requires immersing the thermistor sensors in a water bath again and raising the temperature in steps. The thermistors' readings are taken automatically but the bath temperature must be monitored with a reference thermometer and its temperature entered via the keyboard. As mentioned earlier, the accuracy of the sensors will be the accuracy of this reference thermometer.

The program suggests that you start with the coldest temperature and work your way up. You can, if you like, start at any place in the temperature range of interest and enter the temperature steps in any order, but cold to warm is easiest. With each step, pour a little water out of the Thermos and replace it with more hot or cold water. Wait

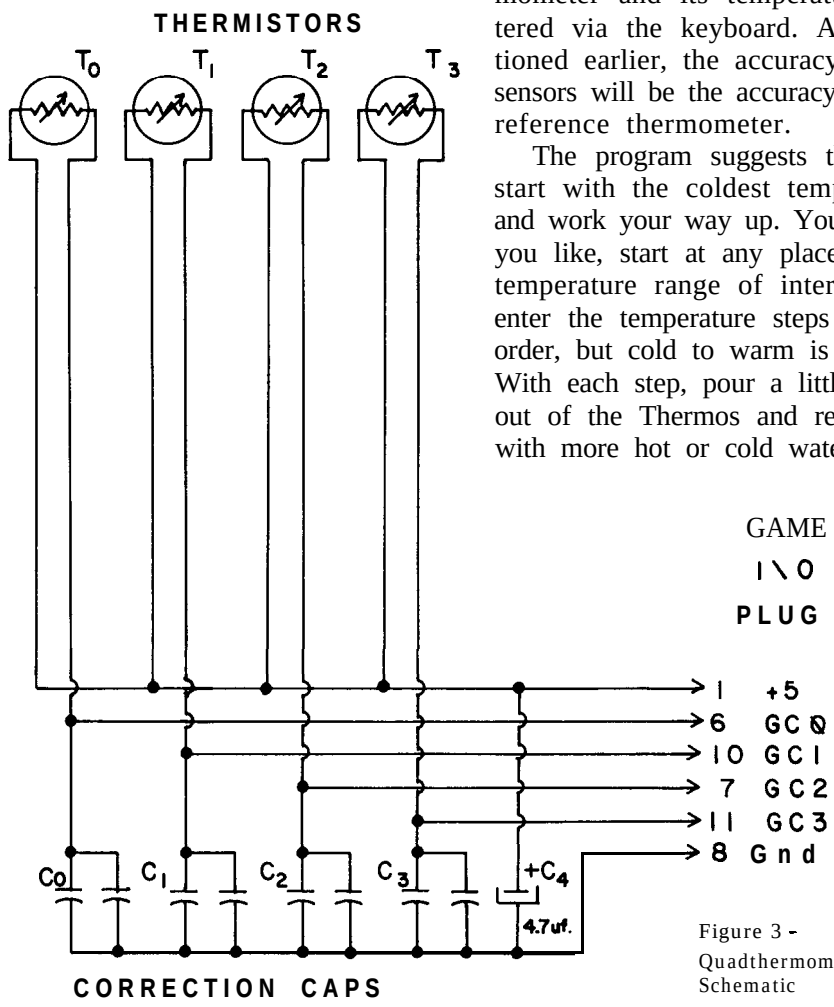


Figure 3 - Quadthermometer Schematic

at least two minutes for the temperature to stabilize before taking readings.

The smallest number of data points that will run is three. The largest is 50. Any number above 10 should produce acceptable accuracy. Taking many small steps near the cold end of the range improves accuracy.

When you complete the temperature steps (type "E" on the keyboard to end calibration), the program will fit the points to an exponential curve and calculate two key parameters for each sensor. The first is GC(25C), the game control reading for that sensor at 25°C, and the second is Beta, the exponential parameter. The Beta calculated here should agree with the Beta obtained from Program #1 to within about 10%. These values may then be printed out or stored on disk.

The mathematical procedure for calculating the parameters of the sensors (Figure 4) uses the Least-Squares Fit of an Exponential Function Technique. The data is first linearized by taking the logarithm of each pot reading. The linearized data points are then fitted to a straight line. The slope of this line becomes Beta and its value at 25°C becomes the first calibration parameter. Even more sophisticated math may be desired for experiments having large temperature ranges to ensure that the accuracy of the reading is uniform over the entire range.

```
800 NP = N
805  PRINT  :  PRINT
810   PRINT "DATA COMPLETE - CALCULATING"
815   PRINT
820   REM   *  CALCULATING  SUMMATIONS
825 SIT = 0:SST = 0
830 FOR I = 0 TO 3:SLR(I) = 0:SP(I) = 0: NEXT I
835 FOR I = 1 TO NP
840 T = T(1)
845 IF TU$ = "F" THEN T = (T - 32) * 5 / 9
850 T = T + 273.15
855 SIT = SIT + 1 / T
860 SST = SST + 1 / (T * T)
865 FOR M = 0 TO 3
870 IF GC%(M,I) = 0 THEN GC%(M,I) = 1
875 SLR(M) = SLR(M) + LOG  (GC%(M,I))
880 SP(M) = SP(M) + LOG  (GC%(M,I)) / T
885  NEXT  M
890  NEXT  I
892 :
895  REM  *  CALCULATE  BETA  & GC(25C)
897 :
900 FOR I = 0 TO 3
905 TP(I,1) = (NP * SP(I) - SIT * SIR(I)) / (NP * SST - SIT * SIT)
910 TP(I,O) = SLR(I) / NP + TP(I,1) * (1 / 298.15 - SIT / NP)
915 TP(I,O) = INT ( EXP (TP(I,O)) * 205 + .5) / 100
920  TP(I,1) = INT (TP(I,1) + .5)
925  NEXT  I
930   PRINT
935  PRINT  "CALCULATION  COMPLETE"
940   PRINT
950   RETURN
```

Figure 4 - BASIC subroutine used by calibration program to calculate a least-squares fit to an exponential data curve.

Data Logging

Program #4 is the data logging program. It allows the user to set up the experiment, record reference data, retrieve the calibration parameters from disk, automatically read the data, add running notes, and record the completed data on disk.

In my example experiment, I mounted the Number 0 sensor on the Apple's 6502, Number 1 on a central RAM chip, and Number 2 on the center of the power supply. Number 3 was mounted by its wire so that the sensor was in the room air outside the computer near the fan intake. The first three were attached with tape and heat sink grease (Radio Shack #276- 1372). The heat sink grease helps transfer the heat from the target to the sensor.

The results of running Program #4 are shown in Figure 5. For the first 30 minutes the fan was on. It was then turned off and a note to that effect was entered in the data. During the next hour and a half 73 data points were taken at one-minute intervals. Data was stored to disk at the end of the experiment.

Program #4 keeps track of the time between data points with a FOR/NEXT loop (Program #4, lines

Quadtherm Parts List

4 Thermistor GA45J1 by Fenwal - Glass coated beade Resistance at 25C - 50K Ratio OC to 50C - 9.53
Fenwal Electronics - 63 Fountain St. Framingham, MA 01701

1 Circuit Board - Radio Shack #276-157 or equivalent

20ft Cable twin wire #22 stranded Radio Shack #278-1385 or equivalent

5ft Cable 8-conductor #24 stranded two rune of Radio Shack #278-365 or equivalent

lft Heat-shrink tubing polyolefin 1/16" ID

2ft Heat-shrink tubing polyolefin 1/8" ID

8 Disc capacitors (values calculated by program)

1 DIP Header 16-pin - Radio Shack #276-1980 or equivalent

845 to 875). It also monitors the keyboard and looks for the experimenter note entries. The delay constants are set in lines 120 (printer off) and 125 (printer on). They should be adjusted for your machine.

It is best to start by setting up a simple trial experiment which just measures the air temperature. Set the sample rate for the 10 data points at one-minute intervals and adjust the delay constants until the time intervals are correct. Since the printer is slower than the monitor, the printer constant P2% will be the slightly smaller of the two.

Of course, it would be far better to use a real-time clock. The use of the FOR/NEXT loop for timing may prove to be inadequate, particularly if notes are being typed in from the keyboard. A programmable interrupt from a real-time clock would be a most useful feature for this program.

Data Reduction

One of the biggest problems with computer-aided experiments is the reams of raw data they produce. Data reduction programs must be customized to the needs of the experiment. Program #5 is a data reduction program designed specifically for evaluating hot electronic equipment. It reads the data from the disk, prints out the data, and calculates statistical means for selected sections of the data.

The personal computer is capable of quite sophisticated data analysis although some computers are slower than others. I have had success fitting equations to data points (lines, polynomials, and nonlinear functions), taking Fast Fourier Transforms for spectral analysis, and drawing contour charts. The major limitation I have observed besides speed is memory. There is always a trade-off among the number of data points which can be stored in memory while leaving enough room for the program and math operations.

Experimental Results

To evaluate my fan experiment, I reviewed all the data graphically and chose two blocks of data which best represented steady-state conditions with the fan on and off. The mean and standard deviation for these data blocks were then calculated. (Figures 6a & 6b)

The 3-inch fan mounted inside the Apple][+ (equipped with 16K memory card, printer card, and disk controller card), was shown to lower the 6502 and power supply surface temperatures by less than 10 degrees, but it lowered the memory chip by over 20°F. I view this as a significant temperature reduction, particularly in the chip memory, and it justifies the expense, noise, and dust accumulation associated with the use of a fan.

Of course, the hardware and programs I've presented are a long way from the ideal personal instrumentation system, but we have to start someplace. The Quadtherm is quite adequate for temperature-measuring experiments, educational applications, and above all, teaching good scientific method. ■

```
6a   STATISTICAL PARAMETERS
     LOW LIMIT OF POINTS 0. HIGH - 29 LIMIT 28
     NUMBER
     ELAPSED TIME - 28 MINUTES
     # MEAN      MIN       MAX          S
     0   85.6      84.9      86.2       .29
     1   81        79.9      81.6       .41
     3 2 70.9 90.4 69.2 89.4 90.7 74.5 .37 1.55

6b   STATISTICAL PARAMETERS
     LOW LIMIT - SO. HIGH LIMIT - 72
     NUMBER OF POINTS - 23
     ELAPSED TIME - 22 MINUTES
     #  MEAN MIN       MAX          S
     1  91.7      90        92.8       .92
     21 99.5 104.5 97.9 102.7 105.8 100.5 .99.86
     3   76.7      76        77.2       .36
```

Figure 6a & 6b

```
EXPERIMENTAL DATA
DATA FILE NAME - APPLE FAN TEST 6
DESCRIPTION OF EXPERIMENT - FAN ON / FAN OFF. TO-6502. T1-MEMORY. T2-POWER SUPPLY. T3-ROOM AIR.
EXPERIMENTER - J.T.RILEY
TIME INTERVAL (MIN) - 1
TEMPERATURE UNITS - F
THERMISTOR PARAMETER FILE NAME - T1
DESCRIPTION OF THERMISTORS - JTR 0 - GA45J1 .02 / 1 - GA45J1 .02 / 2 - FA41J .25 / 3 - GA45J1 .02

#    TIME TØ      T1      T2      T3      PB
     SYSTEM &FAN ON 10 MIN BEFORE TEST
Ø  1169  85.    17    79.91   09.44   73.16  4
1
2  1170 1171 85.43 85.7 80.37 80.84 89.44 89.69 73.16 73.37  4
5  4 1172 1173 85.97 85.97 81.08 81.32 89.94 90.2 73.16 72.95  4
6  1174 1175 86.24 85.97 81.32 81.56 90.2 90.45 74.45 73.59  4
7
   1176  85.43  81.56  90.45  71.51  0
     ROOM LIGHTS REDUCED
8
9  1177  85.43  81.56  90.45  70.91  4
   1178  85.43  81.56  90.71  70.71  4
   1179  85.7  81.56  90.71  70.71  4
1Ø 1180 81832 7  90.71  78.12  4
11 1181  85.7  81.32  90.71  70.12  4
14 1182 1183 85.7 85.7 81.32 81.08 90.71 90.71 69.93 70.12  4
15                 81.08  90.71  70.32  4
   1184 85.7       81.08  90.71  70.12  4
16 1185 85.7 .  81.08  90.71  69.93  4
18 1186 8585.7  80.84  90.71  70.12  4
19  1188  85.7  80.84  90.71  70.12  4
2Ø 1189  85.7  80.84  90.71  69.74  4
21
22  1190  85.7  80.84  90.45  69.74  4
    1191  85.7  80.84  90.71  69.74  4
23  1192  85.43  80.84  90.45  69.55  4
24  1193  85.7  80.6  90.45  69.74  8
25  1194  85. 17  80.6  90.45  69.55  4
26  1195  85.17  80.6  90.45  69.17  4
27 1196 85.17 80.6 90.45 69.36  4
   28 1197 84.9 80.6 90.45  69.93  4
29  1198  84.9  80.6  90.45  69.55  4
   FAN OFF
3Ø 1199  84.9  82.78  91.24  78.51  4
31  1200  85.17  85.37  92.04  71.11  4
                                     4
                                     |
32 43 1201 1212 85.43 88.5 99.75 87.31 96.34 92.86 72.12 76.03  4
44
45 1213 1214 88.5 88.79 100.56 100.15 96.64 96.95 76.49 76.96  4
46
47 1215 1216 89.09 89.39 100.97 101.39 96.95 97.26 76.96 77.2  4

48 49 1217 1218 89.69 89.69 101.81 102.24 97.58 97.58 76.49 76.96  4
50  1219  89.99  102.67  97.89  77.2  4
51  1220  90.29  102.67  97.89  77.2  4
52  1221  90.29  103.11  98.21  76.96  4
    1222  90.6  103.11  98.53  76.96  4
53  1223  90.6  103.55  98.53  76.96  4
55  1224  90.91  103.55  98.86  76.96  4
56  1225  90.91  103.99  98.86  76.96  4
57  1226  91.22  103.99  99.19  4
58  1227  91.54  104.45  99.19  77.2  76.96  4

6Ø 1228 1229 91.54 91.54 104.45 104.45 99.19 99.52 76.49 76.72  4

61 62 1230 1231 91.86 91.86 104.9 104.9 99.52 99.85 76.96 76.49  4
63  1232  92.18  104.9  99.85  77.2  4
64  1233  92.18  104.9  99.85  76.96  4
65  1234  92.18  105.37  100.19  76.72  4
66  1235  92.5  105.37  100.19  76.03  4
67  1236  92.5  105.37  100.19  76.26  4
68  1237  92.5  105.37  100.19  76.26  4
69  1238  92.83  105.37  100.53  76.49  4
70  1239  92.83  105.37  100.53  76.49  4
71  1240  92.83  105.84  100.53  76.26  4
72   1241  92.83  105.84  100.53  76.26  4

DATA COMPLETE 73 DATA POINTS TAKEN.
```

Figure 5