EMBEDDED PC MONTHLY SECTION

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

#98 SEPTEMBER 1998

EMBEDDED APPLICATIONS

Data Acquisition Goes Ballistic—Recording Rocket Acceleration

Audio Spectrum Analyzer

MicroMonitor Your Car's Performance

DeviceNet—Fundamental FAQs

$3.95 U.S.
$4.95 Canada

TASK MANAGER

The Final Rendition

Like any kid, when I studied music, I had my favorite pieces and those I practiced so many times that the rest of the family had strains of "Happy Birthday" or "Three Blind Mice" intermingling with recipe ingredients, calculus assignments, crop-rotation decisions, and so on. My mother, an accomplished pianist and organist for one of the larger congregations in the nearby city, however, knew the real secret to success, "Janice, let's hear your scales." She was equally fond of those inane Hannon exercises, not to mention arpeggios, triads, chords, harmony, history, etc. She had quite a list.

Surprise, surprise, with time, a little patience, and some good old-fashioned parental supervision, I actually got to be not too bad at churning out a little Bach, Chopin, Kabalevsky, and many Hannon exercises. In preparation for my last music exam, I had pushed my way up to five hours of practice a day (now there's a new definition of tedium, eh?!).

Probably the most nerve-racking experience was preparing for recitals. In a sense, exams weren't quite so bad. Yeah, you could fail and spend several more months repreparing. But, a recital happened once a year, in front of many people, people you knew…you flub up then, that's it!

In the weeks before a recital, I protected my fingers—no knives, no door jams, no volleyball. As a result, I would arrive at my recital with my hands intact, but really it did no good. At that point, nerves would strike. My hands would be so cold that I could barely move them and they'd sweat so profusely that I'd lose my grip and slip all over the keys.

I suppose the final product sounded good, but it was always a hair-raising experience. I finished the concert, people congratulated me, but I was still in some kind of echo-chamber reality. So worn out from the rigors of preparation and recital that everything sounded a little hollow and undone.

As I talk to engineers, it seems to me that embedded applications is just about the same. There are all those years of engineering studies, then the first few years of getting your feet wet in some lackey first-time job, and then design, development, debugging, testing, and production. After a product is on the market and ready to sell, many company owners have that same exhausted, hollow look that I had following each recital. One part of them can't believe they actually got it finished, while another is deathly afraid that someone, somewhere will find that fatal timing error in the seventh measure, or perhaps I should say, module.

And, in many ways, the final product—the concert, the invention—is the crowning glory. It shows how well you learned all the bits and pieces that go into the final product. You know, how well you learned your scales.

This month's feature articles are real concert winners. Tom Consi and Jim Bales, researchers at MIT, launch the issue with a look at acquiring acceleration data from rockets. Robert Lacoste, the first-place winner for the PIC17XXX family in Design98, then unveils his audio spectrum analyzer. And, Robert Priestley is back with a car performance monitor—just what you need for your next high-speed, high-tech car rally.

In *EPC*, Jim Brady introduces us to DeviceNet, a new automation network, while Ingo checks out what we need to know about TCP/IP to design real-time applications, and Fred shows us how to solve RF problems, even if we're not RF experts.

Stuart Ball wraps up his two-part MicroSeries on building a PROM programmer, Jeff pokes about in the nether-world of phosphorus, while Tom tells us about his latest and greatest finds at Sensors Expo.

All in all, it's a high-quality recital from engineers determined to keep delivering what they're the best at. I, on the other hand, having completed my last piano exam, changed from music to English, moved from that campus and haven't lived with a piano since. Now when I play, my ears know what to expect, but my fingers can no longer deliver. My many years of training are now put to work picking out fine compositions and recordings.

Janice

janice.hughes@circuitcellar.com

CIRCUIT CELLAR INK®

THE COMPUTER APPLICATIONS JOURNAL

EDITORIAL DIRECTOR/PUBLISHER
Steve Ciarcia

MANAGING EDITOR
Janice Hughes

SENIOR TECHNICAL EDITOR
Elizabeth Laurençot

TECHNICAL EDITOR
Michael Palumbo

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
Ken Davidson
Fred Eady

NEW PRODUCTS EDITOR
Harv Weiner

ASSOCIATE PUBLISHER
Sue (Hodge) Skolnick

CIRCULATION MANAGER
Rose Mansella

BUSINESS MANAGER
Jeannette Walters

ART DIRECTOR
KC Zienka

ENGINEERING STAFF
Jeff Bachiochi

PRODUCTION STAFF
Phil Champagne
John Gorsky
James Soussounis

EDITORIAL ADVISORY BOARD
Ingo Cyliax Norman Jackson David Prutchi

Cover photograph Ron Meadows—Meadows Marketing
PRINTED IN THE UNITED STATES

ADVERTISING

ADVERTISING SALES REPRESENTATIVE
Bobbi Yush
(860) 872-3064
Fax: (860) 871-0411
E-mail: bobbi.yush@circuitcellar.com

ADVERTISING COORDINATOR
Valerie Luster
(860) 875-2199
Fax: (860) 871-0411
E-mail: val.luster@circuitcellar.com

CONTACTING CIRCUIT CELLAR INK

SUBSCRIPTIONS:
INFORMATION: www.circuitcellar.com or subscribe@circuitcellar.com
TO SUBSCRIBE: (800) 269-6301 or via our editorial offices: (860) 875-2199
GENERAL INFORMATION:
TELEPHONE: (860) 875-2199 FAX: (860) 871-0411
INTERNET: info@circuitcellar.com, editor@circuitcellar.com, or www.circuitcellar.com
EDITORIAL OFFICES: Editor, Circuit Cellar INK, 4 Park St., Vernon, CT 06066
AUTHOR CONTACT:
E-MAIL: Author addresses (when available) included at the end of each article.
ARTICLE FILES: ftp.circuitcellar.com

For information on authorized reprints of articles, contact Jeannette Walters (860) 875-2199.

CIRCUIT CELLAR INK®, THE COMPUTER APPLICATIONS JOURNAL (ISSN 0896-8985) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (860) 875-2751. Periodical rates paid at Vernon, CT and additional offices. **One-year (12 issues) subscription rate USA and possessions $21.95, Canada/Mexico $31.95, all other countries $49.95. Two-year (24 issues) subscription rate USA and possessions $39, Canada/Mexico $55, all other countries $85.** All subscription orders payable in U.S. funds only via VISA, MasterCard, international postal money order, or check drawn on U.S. bank.
Direct subscription orders and subscription-related questions to Circuit Cellar INK Subscriptions, P.O. Box 698, Holmes, PA 19043-9613 or call (800) 269-6301.
Postmaster: Send address changes to Circuit Cellar INK, Circulation Dept., P.O. Box 698, Holmes, PA 19043-9613.

INSIDE ISSUE 98

EMBEDDED PC

www.circuitcellar.com
★ September Password: Arthur ★

READER I/O

BEFORE YOU SPEAK, WE LISTEN

I enjoy the articles on RTOSs, but I'm surprised that Ingo hasn't mentioned RealTime Linux. RT-Linux offers sub-100-ms real-time response, Posix soft-real-time tasks, a tremendous development environment with free compilers, RTOS, and debuggers, and remote logging, monitoring, and debugging via a PPP link.

James Linder
jam@woozle.dialix.oz.au

Keep an eye out for upcoming articles on Linux in Embedded PC, *especially in Real-Time PC, where Ingo will be presenting a series of applications using RT-Linux.*

WITH JUST A LITTLE MORE HARDWARE...

Thanks for Dan Cross-Cole's article about turning your PC into a multichannel analyzer ("Using a PC for Radiation Detection," *INK* 96). The calculations to determine the percentage of radon in a poured concrete basement were enlightening. Working from counts to U-235 to U-238 to radon to concentrations to flow rates to a muffin fan is a lot more interesting than sending a canister off to be measured by a lab.

However, a couple of significant calibration issues weren't addressed because there wasn't enough hardware applied to the project.

First, any pulse amplitude spectra measured by the analyzer will be biased toward higher energy pulses. The sample-and-hold circuit is really a peak-hold circuit, so what's measured is the largest pulse that has occurred since the circuit was reset. This wouldn't be a problem if all the pulses were from the 185-keV gamma emission of U-235, but as Dan notes, higher energy pulses come from the decay chain of U-238. For example, Bismuth 214 emits 609 keV, 1120 keV, and 1764 keV gammas. These pulses mask many of the gammas produced by U-235.

The second issue—dead time—relates to the amount of time the analyzer is unable to detect a pulse and the average number of pulses per second. For example, if the C++ program `portdata` takes 10 µs to acquire and store a pulse in its array and the radioactive source is producing 6000 pulses per second, the analyzer is dead for 6000 × 0.0000010 = 0.06 s per second, so 6% of the pulses are lost. As Dan demonstrates, this dead time can be compensated for by calibrating the analyzer with a radioactive source of a known activity.

This only works if the multichannel analyzer PC does nothing else. Anything that increases the amount of time `portdata` takes to handle the data (e.g., interrupts) substantially changes the dead-time and, consequently, the measured pulse rate. In other words, don't expect good results if you run this task on Windows NT.

Minimize the first issue by using a gated peak-hold where the gate is turned off shortly after a pulse is detected. The second issue can be handled by using the pulse detector to generate an interrupt and a timer to track the amount of time between pulse detection and ISR data collection.

Stephen Lloyd
slick@lanl.gov

MICROCHIP TAKES ON MOTOROLA

Tom Cantrell's hint of a looming 8-bit OTP MCU war in "The Micro Price is Right" (*INK* 96) was on target. Three weeks after Motorola introduced the 49¢ 8-bit MCU, Microchip Technology fired back with an 8-bit MCU priced at 40¢ for a ROM and 49¢ for an OTP in large quantities.

The PIC16C505 OTP microcontroller provides design engineers with a higher value solution by offering 1024 × 12 words of program memory, 2:1 code compaction over competitive solutions, 72 × 8 bytes of user RAM, 12 I/O pins with 25-mA sink and source capability, wakeup on I/O change, and 4-MHz internal clock oscillator.

According to Dataquest, Microchip now ranks number two in worldwide 8-bit MCU shipments—thanks largely to *INK* readers. In turn, these leading designers can rest assured that Microchip will respond to future industry battle cries and deliver the most appropriate 8-bit MCUs for their design needs.

Eric Sells
eric.sells@microchip.com

BACK TO SCHOOL

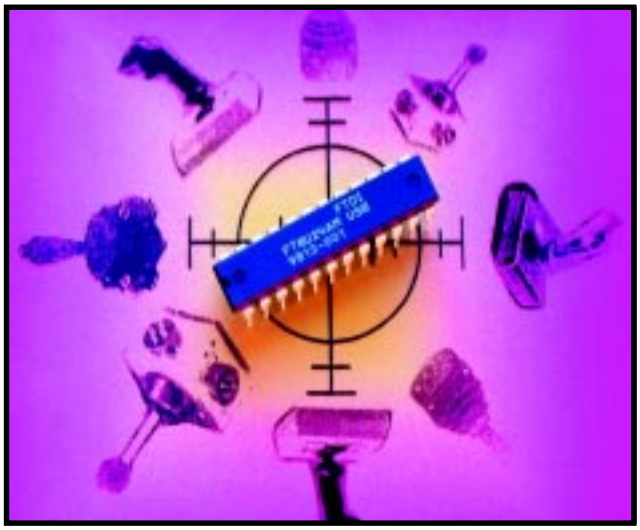
I was struck at how well thought out Ken's recent editorial was ("A Mind is a Terrible Thing...," *INK* 95). It gives good tips and directions for helping students.

This past year, I was very pleased to have directed my focus on one agent useful towards lymphoma. This small success reminded me to set my goals carefully, and I am happy to incorporate your method for reaching students.

Marc McGary
mcgama8@elwha.evergreen.edu

NEW PRODUCT NEWS

Edited by Harv Weiner



USB JOYSTICK CONTROLLER

The **FT8U24AM USB Game-Port Joystick Controller IC** incorporates a four-channel ratiometric measuring circuit for determining the position of the analog controls. Traditional low-cost analog joysticks track position with an RC time constant, using low-tolerance components (i.e., a potentiometer and a capacitor). Resulting threshold values vary with temperature and voltage, producing inaccurate joysticks that need frequent recalibration. Digital joysticks use optical sensors and electronics to eliminate these inaccuracies, but they can be more expensive.

The FT8U24AM measures the angle of the potentiometer rather than its resistance to eliminate the tolerance effect and provide a typical accuracy of ±1%. It features an integral EMCU microcontroller with preprogrammed mask ROM, 128-byte data memory and 12 I/O pins. The four ratiometric analog channels measure *x* and *y* movements as well as throttle and rudder functions. The IC supports a wide range of configurations with up to four analog controls, four stick and four base buttons, as well as support for a POV hat switch. The IC is packaged in a 24-pin PDIP.

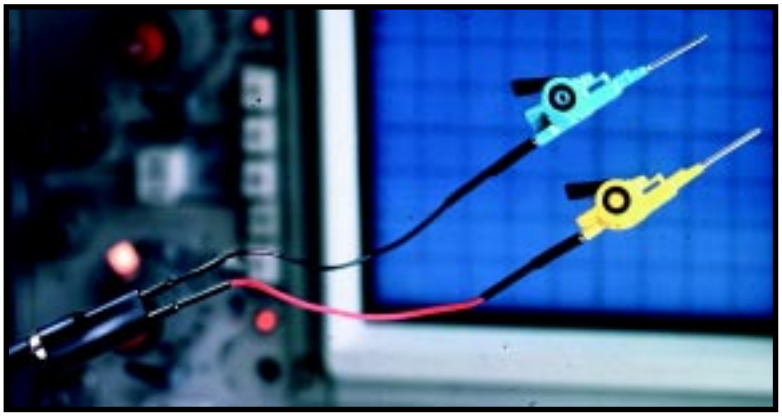
The FT8U24AM sells for less than **$2** in quantity.

Saelig Co.
(716) 425-3753
Fax: (716) 425-3835
www.saelig.com

NEW PRODUCT NEWS

OSCILLOSCOPE PROBE ADAPTER

The **MicroGripper Oscilloscope Kits** from Emulation Technology provide easy probing of fine-pitched devices. They are uniquely designed to provide testing of PQFP and SOIC packages from 0.8- to 0.3-mm lead pitch. Combined with a dual-lead adapter, the MicroGripper enables hands-free testing. The devices work with all popular oscilloscope probes.



The MicroGripper tip is insulated to prevent shorting out when attached side-by-side and features a wire-tip diameter of 0.08 mm. It is constructed of steel or stainless steel wire, nickel plated with an ABS plastic finish. It is rated at 500 WVDC. The kit is intended for use with voltages up to ±40 V. The probe capacitance is less than 2 pF between adjacent grippers and its inductance is less than 1 mH (both measured at 1 kHz). The 3-dB bandwidth is greater than 100 MHz.

Three separate kits are available, depending on the lead pitch. Each kit contains parts for use with two probes, including four MicroGrippers and two dual-lead adapters.

Pricing ranges from **$145** to **$199**.

Emulation Technology, Inc.
(800) ADAPTER
(408) 982-0660
Fax: (408) 982-0664
www.emulation.com

NEW PRODUCT NEWS

CALLER-ID PLUG FOR PC

The **PC Caller ID** plug decodes caller-ID data sent over a telephone line and delivers the name, number, date, and time to a computer. Enhanced services, like voice-mail notification and caller ID with call waiting, are also supported. The PC Caller ID Plug, which measures only 2.25″ × 3″, plugs directly into any spare PC serial port and requires no external power.

Included software (with source code) enables the user to transform a PC into an advanced caller-ID box and to integrate caller ID into other applications. The software has DOS- and Windows-based programs. Windows software source code is supplied in Visual Basic, and DOS software source code is supplied in C. The Windows software supports Windows 3.11, 95, 98, and NT.

The unit is FCC Part 68 certified and can supply caller-ID information to microcontrollers, SBCs, and other embedded systems. Full documentation, including the data format and a schematic diagram for interfacing to other hardware, is included.

The ITU PC Caller ID Plug is available for **$34.95**. A 30-day money-back guarantee and a 90-day warranty are supplied with each unit.

ITU Technologies
(888) 448-8832 • (513) 661-7523
Fax: (513) 661-7534 • www.itutech.com

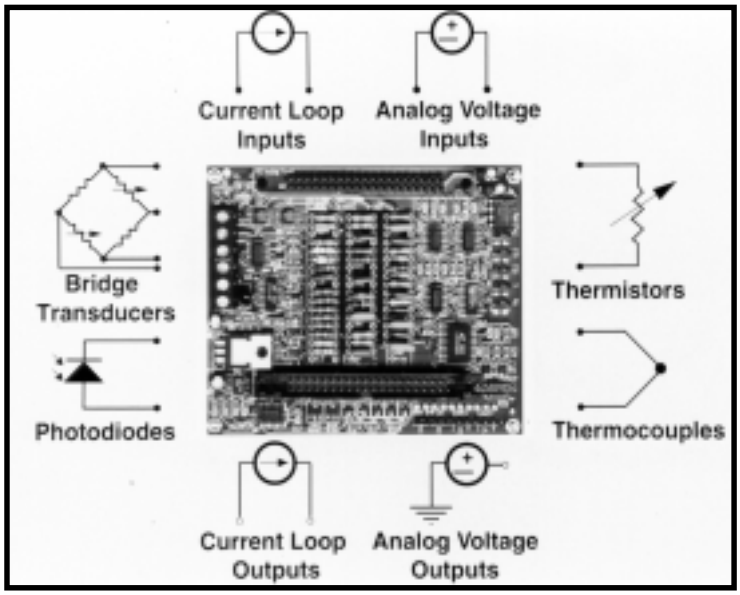
NEW PRODUCT NEWS

ANALOG SIGNAL CONDITIONING BOARD

The **QED Analog Conditioning Board** enables direct monitoring and control of 16 analog inputs and 8 analog outputs from thermocouples, RTDs, strain gauges, pressure sensors, photodiodes, bridge sensors, analog actuators, and 4–20-mA transmitters. Onboard amplification, attenuation, and filtering can be easily customized with plug-in components to provide flexible signal conditioning. Analog outputs range from 0 to 10 V.

Typically controlled via a direct cable interface to the C-programmable QED board controller, the QED Analog Conditioning Board also features built-in thermocouple cold-junction compensation to simplify temperature measurement and control.

The QED Analog Conditioning Board sells for **$249**.



Mosaic Industries, Inc.
(510) 790-1255 • Fax: (510) 790-0925
www.mosaic-industries.com

FEATURES

Smart Rockets

FEATURE ARTICLE

Tom Consi & Jim Bales

Data Acquisition in Model Rocketry

Model rockets have always served as ideal vehicles for teaching physics and engineering basics. With this new data-acquisition device, Tom and Jim show how to trace the rocket's acceleration and store the entire flight and its data.

Model rockets have inspired generations of students to pursue careers in engineering and science. Indeed, many of you probably went through a rocket phase in your formative years.

Model rockets are popular with aspiring engineers for good reason. They're exciting, they're fun to build and launch, and they offer a number of significant engineering challenges that can be tackled with simple tools and small budgets.

From an educational perspective, model rockets introduce a number of fundamental concepts in physics such as Newton's laws, lift, and drag. Inexpensive microcontrollers and solid-state sensors add an exciting new dimension to model rocketry.

It's now possible to build tiny devices that can measure and record the performance of a model rocket in flight. This article describes just such a device that we designed, built, and flew as part of our "Smart Rockets" seminar at MIT.

Our system measures just one flight characteristic—acceleration—although it could be easily modified to measure other aspects of a rocket's flight. We designed the system to fly in a small, single-staged, model rocket that could be launched in a relatively small area.

The entire system, including battery, is 4¼″ long, slightly under an inch wide, and weighs about 1 oz. (32 g). Photo 1 shows a picture of the system mounted in the rocket.

Inside the payload compartment is a small circuit board that contains an acceleration sensor, power supply, microcontroller, and nonvolatile memory chip. A pair of leads brought out of the payload run down to the tips of two fins and touch corresponding contacts on the launch pad. The launch-pad contacts attach to a cable that leads to the launch control box.

When the launch button on the control box is pressed, two things happen. First, the leads brought out to the fins are shorted together, triggering the data-acquisition system. Second (and electrically isolated from the first), a current passes through the igniter, starting the rocket's engine and setting the vehicle into flight.

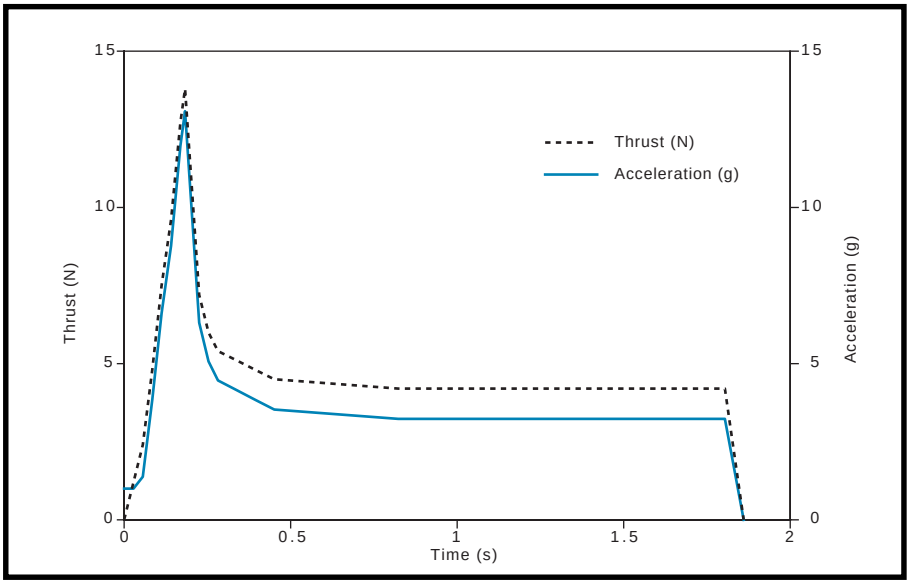


Figure 1—*This graph shows the predicted performance of the rocket. The thrust data was extracted from a performance curve of the Estes C6-5 engine provided by the National Association of Rocketry. The acceleration curve for the rocket is calculated. Note that at* t *= 0, the system experiences Earth's gravity, which is equivalent to a rocket accelerating upward at 1 g in free space.*

INTRODUCTION TO MODEL ROCKETRY

Model rockets are lightweight, low-cost, solid-fuel rockets designed to permit safe experimentation with the principles of rocketry. Factory-made, single-use, solid-fuel engines are a key feature in model-rocket safety.

A model-rocket engine consists of, from bottom to top, a nozzle, propellant, delay charge, ejection charge, and an end cap. This design governs the overall flight characteristics of the model.

The acceleration phase (i.e., when the propellant burns) lasts from a fraction of a second to up to 2 s for typical rocket engines (see Figure 1). It is followed by a longer (several seconds) tracking phase when the rocket coasts up to maximum altitude. The model is not powered during this phase, as the delay charge that generates smoke for visibility has negligible thrust.

At the end of the tracking phase, the ejection charge ignites at the peak altitude, when the velocity of the rocket is near zero. The ejection charge blows forward, bursting the end cap, and pressurizing the interior of the model, which pushes the nose cone/payload section out of the rocket body.

Both pieces, connected by an elastic shock cord, then drift safely back to earth via a parachute. During this recovery phase, the rocket is at the mercy of the wind which can blow it onto rooftops, into trees or power lines, or, in our case, dump the model in the middle of the Charles River! If all goes well, however, the rocket is recovered, the engine is replaced, and the model can be flown again.

Model-rocket engines are designated by a letter and two numbers. The letter indicates the total impulse of the engine, which is just the integral of the thrust versus time curve of the engine (the dashed line in Figure 1).

We used A-, B-, and C-sized engines with total impulses of 2.5, 5, and 10 N, respectively. The first number of the engine designation is the average thrust in newtons, and the second number is the period of the tracking phase in seconds.

Our rocket—the Estes Nova Payloader—is a typical single-stage model rocket. It's about 21″ long, 1″ in diameter, and weighs 1.8 oz. (50 g) without a payload or engine.

The rocket consists of a paper tube, balsa wood fins, and a 4¾″-long plastic payload section topped by a nose cone. The wood and paper parts are assembled with wood glue, and the plastic parts with model cement. The rocket body and the nose cone/payload section are connected by an elastic rubber shock cord, which attaches to the 12″ diameter parachute.

Prior to launch, an engine is inserted into the rear of the rocket and held in place with a spring clip. A piece of recovery wadding (fireproof tissue) is placed in the rocket body just forward of the engine to protect the plastic parachute and shock cord from hot gasses during ejection.

Next, the parachute and shock cord are coiled up and inserted into the rocket. The nose cone/payload section is inserted into the forward end of the body, and the model is placed on the launch pad, ready for flight.

Our launch platform—an Estes Porta-Pad II—consists of a plastic tripod, a steel blast deflector, and a meter-long launch rod. The rocket has a small tube attached to its side, the launching lug, that holds the model to the launch rod during the initial phase of powered flight.

Model-rocket engines are fired electrically using an igniter that typically consists of a piece of nichrome wire coated with a flammable material. A current passed through the wire ignites this material, which then ignites the propellant.

The igniter connects to a long pair of wires that lead to the launch control box at a safe distance (~20′) from the rocket on the pad. The launch control

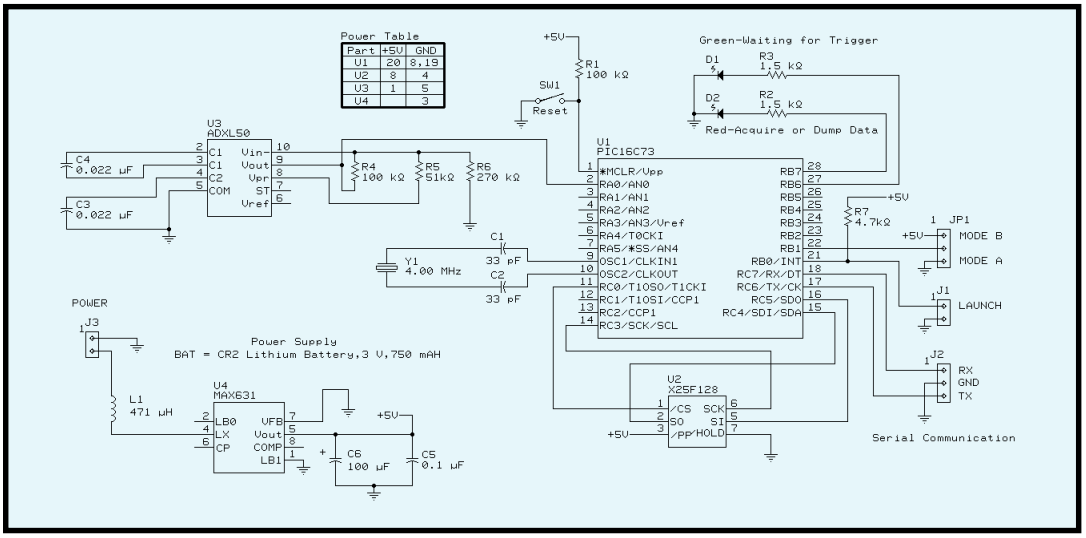


Figure 2—*Here's the schematic for the model-rocket data-acquisition system. During the rocket's flight, the PIC16C73 microcontroller reads the ADXL50 accelerometer and stores the data in the X25F128 serial flash memory.*

box has a switch (the launch button) that connects the igniter to the 6-V battery (four alkaline AA cells). We discuss the details of the launch control box later.

ROCKET SCIENCE 101

Newton's First Law, $F = ma$, contains all the physics you need to predict the accelerations that the system will record. All we need is the mass of the rocket (m) and the applied force (F), right?

The force exerted by a rocket engine is not constant as you see in the thrust versus time curve for an Estes C6-5 engine (the dashed line in Figure 1). We assume the rocket moves straight up while the thrust is applied, so the total force is the engine thrust minus the gravitational force (mg, g = 9.8 m/s^2).

If the mass of the rocket were fixed, then the acceleration is the dashed line scaled by the mass. However, as the propellant burns, the mass of the rocket decreases.

How important is this effect? For NASA, it's critical—85% of the space shuttle's take-off weight is fuel!

In our case, only 12 g of the 105-g take-off weight is fuel (11%). So, we can treat the rocket mass as a constant, which gives us the expected acceleration curve shown as the solid line in Figure 1.

Our students made this calculation on the first day of class. Their task for the semester was to determine, by direct measurement, if Figure 1 is correct.

CONTROL AND DATA LOGGING

We chose the Microchip PIC16C73 microcontroller as our data-logging engine (see Figure 2). This chip has all the I/O functions we need: DIO pins, a five-channel eight-bit ADC, a USART, and an SPI synchronous serial port.

Its 4 K words of data space (we used the reprogrammable windowed EPROM version) and 192 bytes of RAM easily accommodate our application code as well as provide some head-room for future expansion.

Size was a key factor in determining what microcontroller to use. The PIC-16C73 comes in a 28-pin skinny-DIP package that fits our space constraints.

The only thing the '16C73 lacks is nonvolatile data storage. This is provided by a Xicor 25F128 serial flash-memory chip that contains 16 KB of memory in an 8-pin DIP package.

The Xicor chip communicates with the '16C73 via the PIC's SPI serial interface. A 4-MHz crystal (Y1) was chosen so each of the PIC instructions executes in 1 µs, which is convenient for calculating code-execution times. The reset button (SW1) is a tiny SPST momentary On push button which grounds the MCLR pin of the PIC.

We considered two methods of triggering data collection by the PIC. One is to have the PIC continually read the ADC and use the flash memory as a ring buffer. The PIC waits for the accelerometer output to exceed a threshold and stops writing to the buffer just before completing the next loop around the ring.

The drawbacks are the added software complexity (a concern for students just learning PIC programming) and the finite number of write cycles allowed by the flash chip. Also, for the first launch, we would have to set the threshold based on our predicted acceleration curve, a golden opportunity for Murphy to intervene!

Instead, we chose to supply an external electrical signal when the launch switch is pressed. This technique removes all the objections listed above and also provides us with data on the lag between pressing the launch button and the liftoff.

However, it presents a new challenge: how to make a reliable electrical connection that can be broken with very little force. We need this type of connection in two places—between the launch pad and the rocket and between the rocket body and the nose cone/payload section (which must separate during ejection). Our solution is in the "System Construction" section.

Two digital input pins control the system operation. When grounded, the trigger input (RB0) starts the program running.

On triggering, the program enters one of two modes, determined by the jumper on digital input pin RB1. Mode A (RB1 low) causes the PIC to acquire accelerometer data. In mode B (RB1 high), the PIC immediately dumps its stored data to a host computer though its USART.

As Photo 1 shows, the trigger lines (RB0 and ground) are brought out of

the payload section and down the rocket body to the fin tips where they contact two plates on the launch pad, which are, in turn, connected to the launch control box (see Figure 3). The launch button (SW2) shorts these contacts together, triggering the system to start acquiring data (if it is in mode A).

Green and red status LEDs connect to digital outputs RB6 and RB7, respectively. The green LED indicates that the system is on and waiting for a trigger signal. The red LED means the system is either acquiring data (mode A) or dumping data through its USART (mode B).

Four other digital outputs (RB2–5) were used for debugging purposes.

To get the data onto a PC, we built a small interface box that has a push button to trigger RB0 while in mode B. This interface box contains a MAX-233 TTL to RS-232 level shifting chip that eliminates this chip from the payload's circuit board. The box also has its own battery so the payload's battery isn't drained during retrieval of data (see Figure 3).

Photo 1a—*Here's our smart rocket on the launch pad. Note the clear payload compartment containing the data-acquisition system and the copper contacts beneath the fins. The brown streak down the right side of the rocket is one of the two lines of conductive paint that bring the trigger signal to the payload.* ***b—****In this view of the data-acquisition system, you see the modifications made to the payload compartment. The brown patch on the rear bulkhead connector is conductive paint that forms a sliding electrical connection with the rocket body. To the left of the circuit board is a model-rocket engine.*

THE ACCELEROMETER

We chose Analog Devices' ADXL50 accelerometer, which requires a regulated +5-VDC supply and has an on-chip amplifier. The gain of this inverting final stage can be set by external resistors, which enables us to trade off range for sensitivity.

We selected the ADXL50's external resistors to condition the sensor output to have a zero-g bias level of about 2.5 V. Since the maximum acceleration is expected to be 14 g, we selected the gain resistors R4–R6 to create an inverting amplifier with a range of ±20 g.

An internal voltage reference makes the output insensitive to variations in the supply voltage. The output of the accelerometer connects directly to the PIC's 8-bit ADC (pin RA0), giving a resolution of approximately 0.2 g.

One potential source of error is fluctuations in the regulated power supply, which serves as the reference voltage (V_{REF}) for the PIC's ADC. A better design would be to use a voltage reference chip for V_{REF}, although that would, of course, consume precious circuit board space.

Another potential source of error is misalignment of the sensor with respect to the rocket's thrust axis. This error is small: one minus the cosine of the misalignment. So, for a misalignment of 5°, the error is less than 0.5%.

SOFTWARE

The code was written in PIC assembly language. We used Microchip's MPASM assembler and PICStart 16C programmer as development tools. A block diagram of the software is shown in Figure 4.

On powerup or after coming out of a reset, the software initializes the I/O facilities and internal resources of the PIC (DIO, ADC, SPI interface, USART, and Timer 0). The USART operates at 9600 bps, and the SPI clock runs at 1 MHz.

The program then polls RB0, which is normally high. A trigger signal pulls RB0 low and causes the program to check the mode and execute the appropriate code. After the selected mode executes, the program loops back to polling RB0 for another trigger signal (see Figure 4a).

Mode A is the real-time data-acquisition portion of the program. It is entered if pin RB0 is jumpered low. The ADC is read during an interrupt service routine that is governed by Timer 0, which provides an interrupt every millisecond, thus setting the data-acquisition rate (see Figure 4b).

How data is acquired is dictated by the behavior of the Xicor 25F128 serial flash-memory chip. It receives data in 256-bit packets via its SPI interface and stores it in one 32-byte sector of its flash memory.

Transferring 256 bits takes only about a quarter of a millisecond at an SPI clock rate of 1 MHz. The programming operation, however, takes 10 ms. Fortunately, this occurs automatically after the bits are received and it requires no further intervention of the PIC.

The program in mode A reads the ADC 32 times at a 1-kHz rate, storing these values in 32 bytes of buffer RAM on the PIC chip. Between the thirty-second and thirty-third A/D conversions, it shoots the contents of the buffer to the Xicor chip through the SPI.

It then proceeds to fill the buffer RAM again. During the 32 ms that the buffer is being refilled, the Xicor chip has plenty of time to write the data it just received into its flash memory.

In addition to sending acceleration data, the PIC sends the sector address that indicates where to store the data in the Xicor chip. The sector address is incremented after every SPI transfer.

This cycle repeats itself 256 times until 8 KB of data are stored on the Xicor chip. The program then ends mode A and goes back to polling the trigger pin.

The Xicor chip holds up to 16 KB of data. But, we found that 8 KB of data, corresponding to 8 s of time, was enough to capture a rocket flight from launch to recovery phase.

Triggering the system in mode B presumes that a host computer is connected to the USART of the PIC via the interface box. The PIC reads a single byte from the Xicor chip, converts it to a three-byte ASCII decimal number, and transmits it to the host computer (see Figure 4a).

Each transmitted number is appended with the ASCII codes for carriage return ($0D) and line feed ($0A). This cycle repeats itself until all 8 KB of data are transmitted to the host.

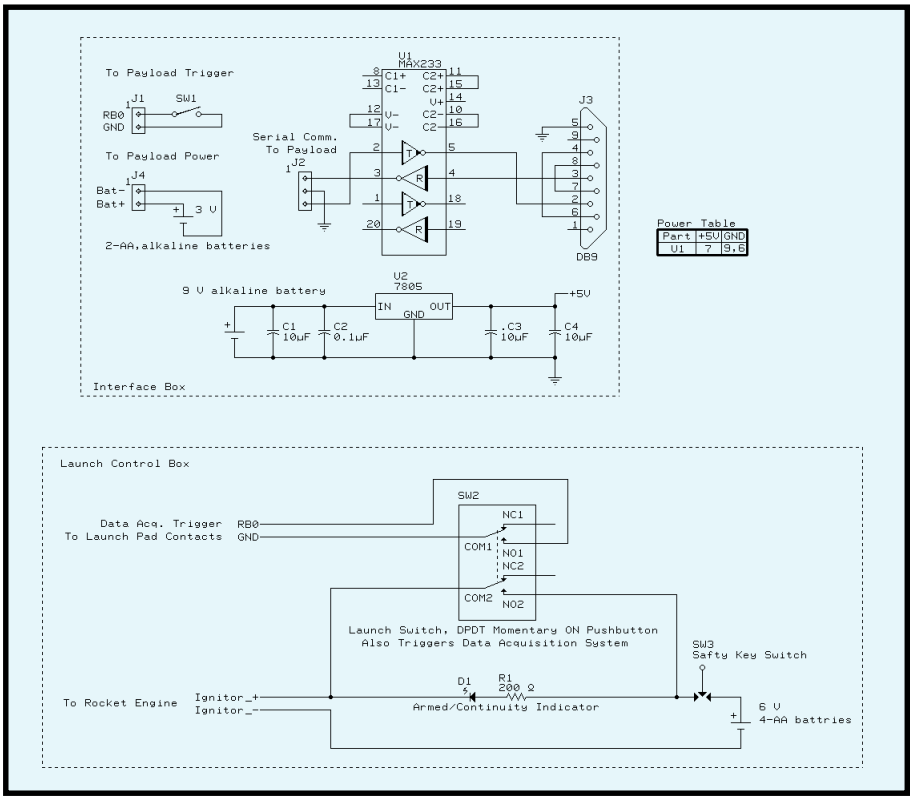


Figure 3—*In the ground equipment, the interface box connects the payload to a host computer for data transfer. Pressing the ignition switch on the launch control box (SW2) simultaneously ignites the rocket engine and triggers the payload to record data.*

The user must set the host computer to accept this ASCII stream before the PIC is triggered into mode B. There is no handshaking between the PIC and the host. As with mode A, on termination of mode B, the program loops back to polling the trigger input.

In the field, we used a laptop, running a terminal emulator set in screen-capture mode, as the host computer. The captured file is a 1-column × 8192-row matrix of ASCII decimal numbers.

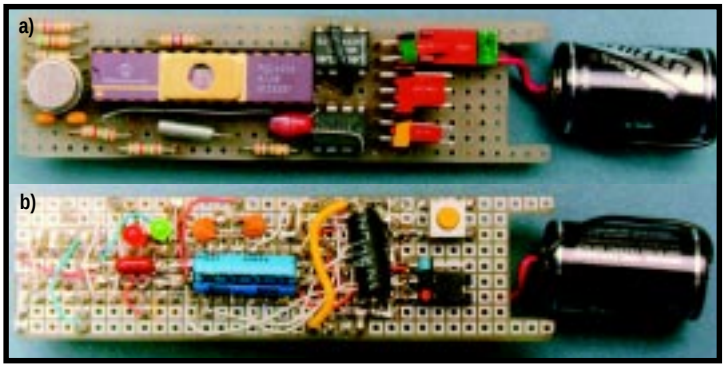


Photo 2a—*The accelerometer is in the TO-100 can at the forward end of the board. Its axis of sensitivity is along the diameter of the can that crosses the tab. The connectors are for the trigger signal (yellow), serial communication (central three-pin connector), and power (green), which is shown connected to the lithium battery.* ***b—****The red and green status LEDs, the large black inductor, the mode jumper, and the reset button are visible on the wire side of the board.*

As soon as the rocket was recovered and the data uploaded to a laptop, we plotted the data to ensure that the system worked properly. Thus, we could correct problems on site, instead of discovering them back in the lab. Being able to read the data in the field using a simple editor more than offset the extra time needed to transfer ASCII as opposed to binary data.

POWER

The power system, consisting of a battery and a voltage regulator, was one of the more difficult portions of the design because it operates under several constraints. First, the battery must source 30 mA at 3 V.

Second, the regulated 5-V rail is critical because it is the reference for the PIC's ADC. Any change in this voltage translates into a proportional error in the recorded acceleration.

Third, the power system must be compact and lightweight because 50 g is the rocket's maximum payload. Finally, it must be reasonably efficient because small batteries have limited energy.

We started by looking at high-energy batteries. The two obvious choices were alkaline cells and lithium primary cells. Lithium cells have, on average, three times more energy per unit mass than alkaline cells.

We settled on a CR2-style lithium battery designed for high-current applications. It provides 750 mAh at 3 V and weighs 11 g.

We selected Maxim's MAX631A step-up switching regulator because of its low part count (one inductor and two capacitors) and small quiescent

current. The selection of the inductor is critical because its resistance must be low (0.5 Ω or less) and it must not saturate when the current through it peaks. The power supply is shown in Figure 2.

SYSTEM CONSTRUCTION

The data-acquisition system was point-to-point wired on a $^{15}/_{16}'' \times 3\frac{1}{2}''$ piece of pad-per-hole perf board. The PIC chip was mounted in a socket so it could be removed for reprogramming. All other components were soldered directly to the board.

As you see in Photo 2 and Figure 2, three connectors were mounted on the board—two 2-pin connectors to connect to the battery (J3) and to link the trigger lines to the rocket body (J1), and one 3-pin connector for the USART connection (J2).

The big challenge in laying out this board was minimizing the component heights at the edges to accommodate the cylindrical shape of the payload compartment. Thus the PIC chip, accelerometer, and the large capacitor and inductor of the power supply had to be mounted centrally on the board.

We mounted components on both sides of the board, which had the added bonus of balancing the weight of the payload. Photo 2 shows both sides of the payload circuit board.

A slot was cut in the rear end of the plastic nose cone that acts as a guide to hold the circuit board in the payload section (see Photo 1b). Some material was removed from the plastic bulkhead that forms the bottom of the payload compartment to make a crude battery holder.

Pieces of foam wedged between the battery and the wall of the payload compartment help hold everything in place. The forward nose cone and rear bulkhead connector are taped to the payload tube to prevent the electronics from falling out during recovery.

We soldered wires onto the terminals of the battery and brought them out to a connector on the circuit board. Beware! This is the most dangerous part of building the system.

There is a lot of energy in the lithium cell, and excessive heat can cause it to catch fire or explode! Use a cell that has integral solder tabs, and don't linger on them with the soldering iron.

One of the more difficult aspects of the design was bringing the trigger lines (RB0 and ground) out of the payload compartment, down the rocket body, and out onto the fin tips. For this we used conductive paint, the kind used to repair windshield defrosters.

An electrical connection was established between the payload compartment and the rocket body by applying two patches of paint on the outside surface of the payload bulkhead connector where it slips into the rocket body, and two corresponding patches of paint inside of the forward end of the body (see Photo 1a).

Conductive lines were painted from these forward patches around the edge of the tube and down the exterior of the rocket to patches of paint at the tips of two of the three fins. These patches contact copper plates on the launch pad that connect to the launch control box (see Photo 1).

The paint patches on the bulkhead connector and the rocket body provide electrical continuity between the payload and the rocket while the model is on the launch pad. This connection is easily broken by the ejection charge.

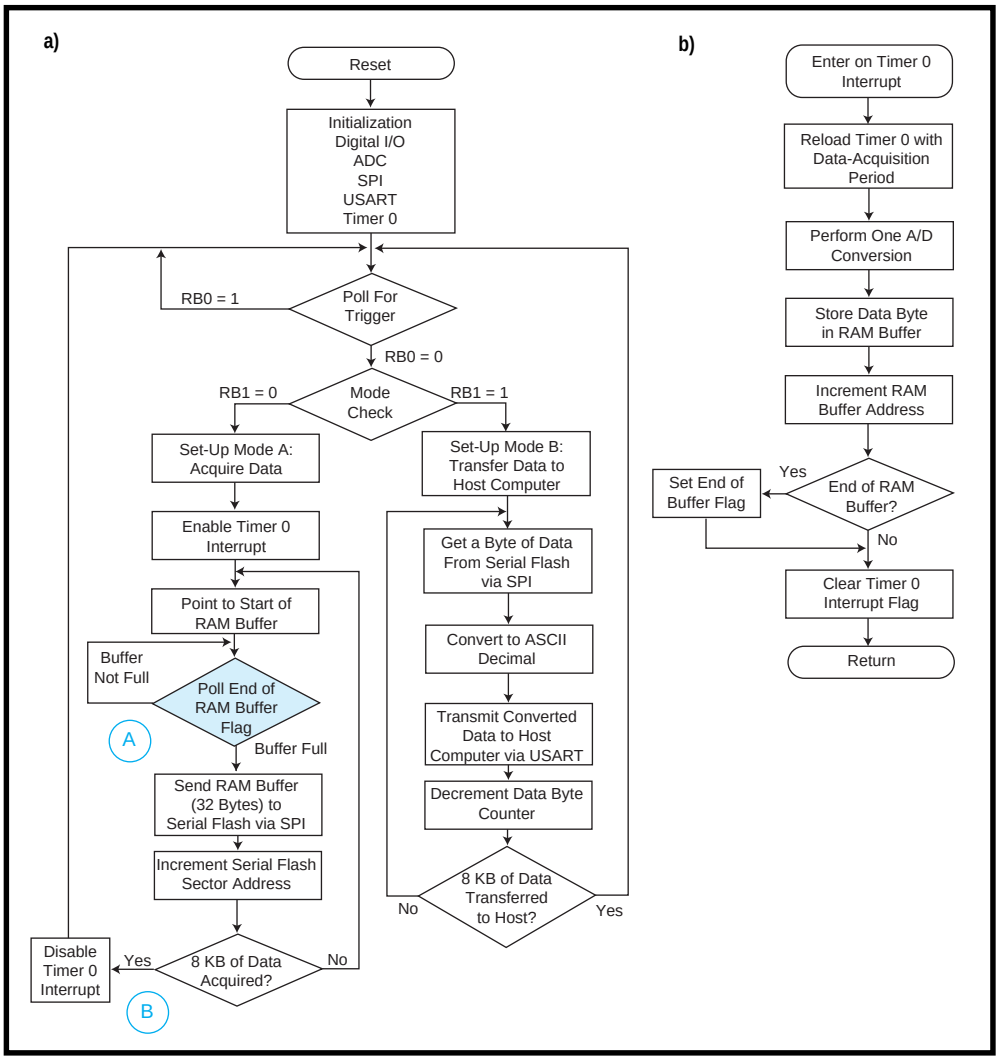


Figure 4a—*Data is acquired by an interrupt-driven routine that runs during the time indicated by the shaded diamond. Note that the code between the points labeled A and B must execute in less than 1 ms to maintain the 1-kHz data-acquisition rate.* ***b***—*In this flowchart of the interrupt-driven data acquisition routine, timer 0 sets the sampling period. Each time the routine is called, one A/D conversion is performed and the data byte is stored in the RAM buffer.*

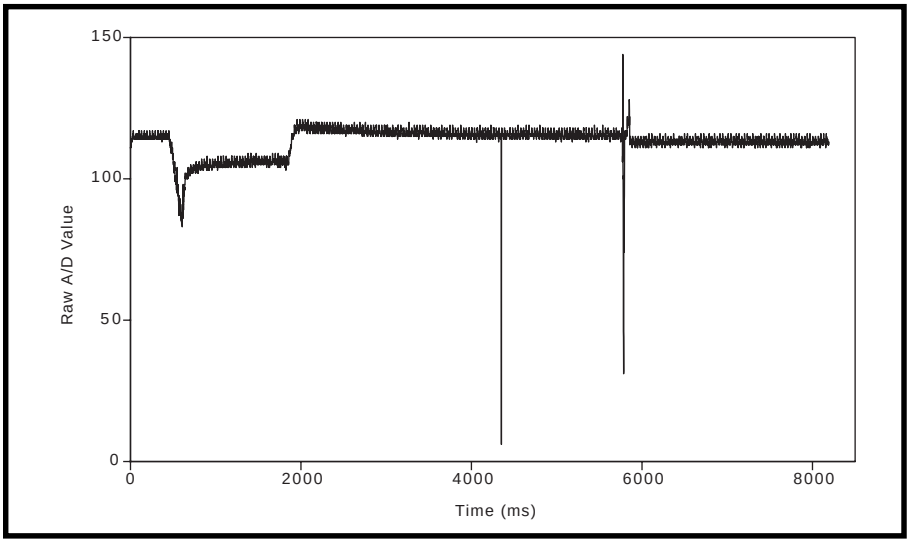


Figure 5—*Here's the raw data from an actual rocket mission using a C6-5 engine. Because this is the output of an inverting amplifier, upward acceleration causes decreased output voltage. This curve was plotted in the field, and it shows that the system functioned correctly. The spike around 4500 ms is spurious, while the structure just before 6000 ms indicates the ejection charge firing.*

Connector J1 brings the trigger lines to the paint patches on the bulkhead connector. It isn't possible to solder wires to the conductive paint, so we essentially pasted the stripped end of each wire to its patch with a thick coating of the conductive paint.

Clearly, conductive paint is not going to hold the wire. To provide strain relief, we glued the insulated portion of the wires to the bulkhead connector using model cement. Still, these connections proved quite delicate and sometimes broke in the field.

LAUNCH CONTROL BOX

The launch control box shown in Figure 3 has many of the features of the standard Estes Electron Beam launch controller. Between the batteries and the nichrome igniter are the safety key and the armed indicator (LED D1). The launch switch is in parallel with the LED.

The igniter is connected with the safety key removed, so the clips to the igniter are not hot. Inserting the safety key applies the battery voltage across the series combination of the armed indicator and the igniter, which enables enough current to flow to light the indicator but not enough to ignite the igniter's black powder coating. A nice feature of this design is that any break in the wiring prevents the armed indicator from lighting.

Our major modification to the Estes design was to replace their SPST launch switch with a DPDT momentary push-button switch (SW2). The second channel of this switch shorts together the two contacts brought out to the fins of the rocket. A 20′ four-conductor cable brings the igniter current and the payload-trigger signal to the launch pad.

Three small blocks of wood epoxied to the blast deflector of the launch pad (see Photo 1a) support the rocket, while electrically isolating the fins from the steel blast deflector.

Copper plates are attached to each block of wood, and two of the plates are connected by leads to the trigger signal lines of the launch cable. The two rocket fins with conductive-paint contacts are positioned over these blocks, and the third fin rests on the unconnected block to balance the model on the pad.

RESULTS

Before leaving the lab, we went through every step in the sequence of the setup, launch (with no engine), data recovery, and data visualization. We even shook the rocket up and down a few times after pressing the launch key, just to generate acceleration data. Once the procedure was perfected, we moved outside.

The raw data from a launch with an Estes C6-5 engine is shown in

Figure 5. The acceleration appears to be negative, but that's because the on-chip buffer of the ADXL50 is an inverting amplifier.

The initial kick and the sustained thrust are easily seen, as is the ejection charge (around 6000 ms). There is also a spurious data point (around 4500 ms). While the accelerometer output has not been converted into acceleration yet, this plot still tells us that the system worked.

Turning Figure 5 into the desired acceleration plot requires a few steps:

- removing the spurious data points
- smoothing the curve with a three-point sliding-window average
- converting the 0–255 A/D readings to 0–5 V
- converting the voltage to acceleration, exploiting the fact that the system saw 1 g just before ignition

The first three steps are straightforward. For the final step, we used the published response curve of the sensor as well as the calculated transfer function for the amplifier, given the values of resistors R4–R6.

We know that, just before ignition, the rocket was experiencing the 1-g gravitational force, so we defined the observed DC offset to be 1 g. Figure 6, shows the final acceleration curve for our system.

Compare Figure 6 to the solid line in Figure 1. The general shape of the curves agree, although the measured curve is compressed in time. The maximum measured acceleration was 17 g, while the maximum acceleration predicted from the mass of the rocket and the characteristics of the C6-5 engine was 14 g.

The correspondence between theory and experiment isn't bad, and it shows our system works as designed. There are several ways to improve the quality of the data.

First is to perform a careful calibration of the accelerometer. Calibration procedures for the ADXL50 are discussed in the datasheet. We calculated gains using the nominal values of our 5% resistors. Using 1% resistors in the accelerometer circuit would give better results.

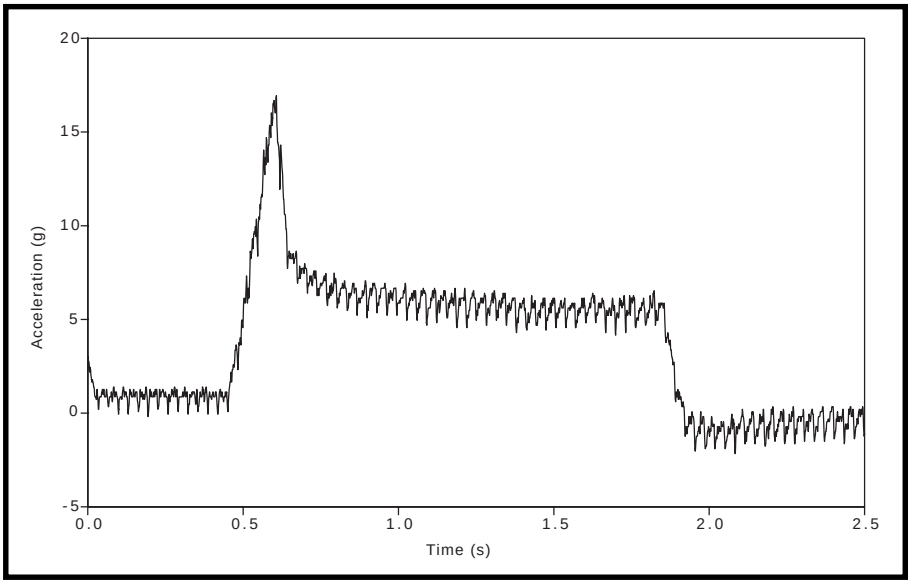


Figure 6—*Here's the processed acceleration data for the launch shown in Figure 5. The shape compares favorably to the predicted acceleration curve in Figure 1.*

Also, a more stable voltage source can be used as the reference for the PIC's ADC. Other questions can be addressed as well: What causes the spurious data? What is the source of the ripple seen even when the rocket is on the pad?

FUTURE DEVELOPMENTS

There are many ways to improve and expand this system. It could be miniaturized further by using a PCB with surface-mount components.

Earlier, we mentioned our decision to use a wire link to the rocket to

trigger the data acquisition as opposed to using a threshold acceleration to trigger the system. Two of our students managed to implement this idea and got it to work quite nicely. Other triggering schemes could use optical or radio signals as wireless triggers.

Also, more sensors could be added. An old model-rocket trick for measuring roll rate is to install a photosensor in the side of the model. As long as the model doesn't fly directly into the sun, there will be asymmetric illumination around the vehicle, causing a periodic signal from the photosensor as the rocket rolls. The PIC could easily measure the frequency of this signal.

A number of other sensors could be added—thermistors, pressure sensors, and even tiny gyroscopes. Another class of sensors consists of devices that indicate when critical events occur during the flight of the model.

For example, a fine wire could be placed across the nozzle of the rocket engine. When the engine ignites, the wire burns through and signals the PIC, which logs the exact time of ignition. You can imagine similar sensors that indicate when the rocket leaves the pad, when it clears the launch rod, and when the ejection charge occurs.

Additional sensor data could easily exceed our system's capacity. Greater memory can be obtained via high-capacity serial flash-memory modules (discussed by Tom Cantrell in "Serial Flash Busts Bit Barrier," *INK* 85).

We've only scratched the surface of the many possibilities for smart model rockets. We hope to explore some of these avenues in another course. This time we were more than satisfied that all of the students got to fly their rockets and collect real data.

This effort was supported by the MIT Sea Grant College Program and the MIT Department of Ocean Engineering. At MIT we were helped by the Undergraduate Seminar Office, The Safety Office, the Campus Police, and the Athletics Department. Analog Devices, Maxim, Microchip, and Xicor all donated ICs for the course. Microchip (through S-J New England Inc.) also provided PIC programmers and databooks. Finally, thanks to our students, whose interest and dedication were inspirational and who were just a lot of fun to be around!

Tom Consi is a research engineer and lecturer in the MIT Ocean Engineering department, where he develops biologically inspired marine robots. He received his Ph.D. in biology from Columbia University in 1987. You may reach him at consi@mit.edu.

Jim Bales is a research engineer with the MIT Sea Grant College Program. He received his Ph.D. in solid-state physics from MIT in 1991. His research interests include sensors for underwater robotics and power systems for autonomous platforms. You may reach him at bales@mit.edu.

SOFTWARE

Complete source code for this article is available via the Circuit Cellar Web site.

SOURCES

ADXL50
Analog Devices, Inc.
(617) 329-4700
Fax: (617) 326-8703
www.analog.com

Model rockets and rocket equipment
Estes Industries
1295 H St.
Penrose, CO 81240

MAX631
Maxim Integrated Products
(408) 737-7600
www.maxim-ic.com

PIC16C73
Microchip Technology, Inc.
(602) 786-7200
Fax: (602) 786-7277
www.microchip.com

Rocket-engine performance curves
National Assn. of Rocketry
P.O. Box 177
Altoona, WI 54720
www.nar.org

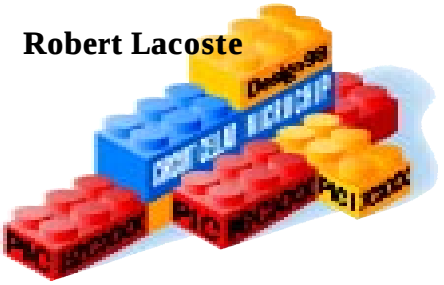
X25F128
Xicor, Inc.
(408) 432-8888
Fax: (408) 432-0640
www.xicor.com

FEATURE ARTICLE

Robert Lacoste

PIC'Spectrum

Audio Spectrum Analyzer



Sure, a DSP can make the calculations and generate the pixel bitmap for an audio spectrum analyzer, but at what cost? Robert gets the same result from a single PIC processor with a design so good he walked away with a Design98 first prize.

I remember being quite astonished when one of my professors explained the basics of frequency domain analysis, "Every periodic signal is the sum of pure sinusoidal signals, with given frequencies, amplitudes, and phases."

Hmm…every signal. That includes the light coming from the sun, the vibrations of my old car, even the tears of my two-month-old baby!

And, I bet that 99% of *INK* readers are like me. You want to understand, and you understand it better if you make it yourself.

So, years ago, I quickly wire-wrapped an analog acquisition board and wrote a small BASIC program on my old Apple II to display the frequency decomposition of an incoming audio signal. It was my first audio spectrum analyzer. Later, I did the same on my PCs, and the '*x*86s were soon powerful enough to get real-time performance.

Using a PC is OK, but how about an autonomous device? Something small enough to bring along anywhere but that has a VGA video output with a decent quality image and real-time refreshes.

My first idea was something like the block diagram in Figure 1a. An amplifier and low-pass filter suppress out-of-the-band signals before the signal goes to an ADC, which transforms it into numerical samples.

A DSP can calculate the frequency decomposition of this signal with the classic Fast Fourier Transform (FFT) algorithm and generate the pixel bitmap in video RAM, which is displayed by a CRT controller chip.

You'd need a fair amount of computing power, but is it possible to do the same with a simpler design? How about a high-end microcontroller? *INK*'s Design98 contest presented a great opportunity to try it with one of the newer Microchip devices—the PIC17C756.

This chip has enough horsepower not only to do an FFT in real time but also to eliminate the CRT controller. The video output can be made with some general-purpose parallel output lines, and the software toggles the corresponding bits to generate the video synchronization and color signals in real time.

With this controller, the block diagram of my logic analyzer (shown in Figure 1b) is, well, as simple as possible. PIC'Spectrum is born!

YOU SAY PIC17C756?

Before getting into more details about PIC'Spectrum (see Photo 1), let's have a look at the PIC17C756 and its stripped-down version, the PIC17C752. These new fully static CMOS chips are an enhancement of the existing PIC17C4X family.

The microcontroller core runs up to 33 MHz, giving a 121-ns instruction cycle. These chips are pure RISC design, and thanks to the two-stage pipeline, each instruction executes in only one cycle, except program branches and table reads/writes, which are two cycles long. That makes it near 8 MIPS—not bad for a microcontroller.

The 58 single-word instructions (coded on 16 bits) are easy to learn for any user of Microchip's smaller controllers. Direct, indirect, and relative addressing modes are supported. External interrupts are present, as is a 16-level hardware stack.

One interesting feature for compute-intensive applications: there is an integrated 8 × 8 bit hardware mul-

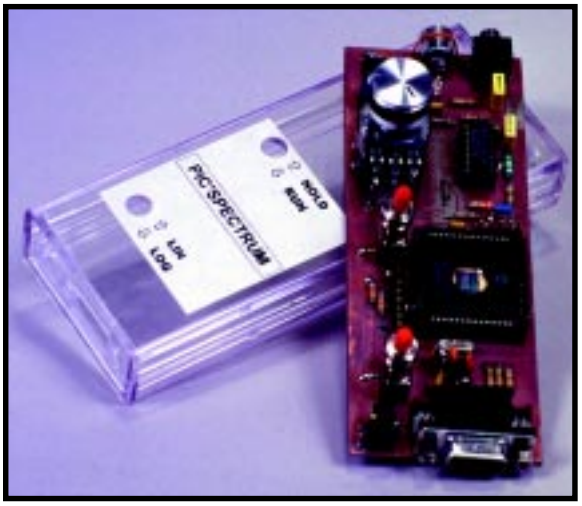


Photo 1—*The PIC'Spectrum prototype is built on a 1″ × 3″ PCB. A good ground plate is mandatory on a mixed digital/analog design like this one. I used a transparent box to prove that there are very few components inside.*

tiplier, working in only one instruction cycle. This multiplier offers a performance boost of more than three times (compared to a software-only version) for a complete FFT calculation.

The internal memory is impressive as well. The PIC17C756 has 16 K words of EPROM program memory as well as 902 bytes of general-purpose RAM, which accommodates quite large projects without the need of external memories, even if extended modes are available and support up to 64 K words of program memory.

The smaller PIC17C752 has only 8 K words of EPROM and 454 bytes of RAM. Microchip has also announced flash-memory versions (PIC17F75X). The memory map is shown in Figure 2.

One piece of bad news: the RAM and registers are still banked, and the working page is selected by some bits in the BSR register. Even if some specific assembler instructions help, this is a major headache for the programmer and a major source of bugs. I hope that Microchip switches to a linear address mode in the near future.

Hosted in 64- (DIP) or 68-pin packages (PLCC and TQFP), the PIC17C75X provides 50 I/O pins with individual direction control. As usual, each pin may be used for general-purpose I/O or dedicated to some on-chip peripherals.

Have a look of the pinout of the PLCC version in Figure 3 and you'll understand that this chip is quite flexible. Both OTP and windowed versions exist, even if they aren't so easy to find.

On the peripheral feature list, there are four timers (two 16 bits wide and two 8 bits wide, TMR0 having an internal 8-bit programmable prescaler), four capture input pins, and three PWM outputs with a 10-bit resolution.

Need more? Perhaps two asynchronous and one synchronous serial port, the latter configurable both in SPI and I^2C modes, master or slave? Or a 12-channel 10-bit ADC? Or an RC-clocked watchdog timer?

How about an integrated supply-voltage supervision? Or a configurable RC/crystal/ceramic clock system with an oscillator start-up timer? Name it, and it's probably there.

The PIC17C756X also has in-circuit programming hardware. By pulling the TEST and MCLR/V_{PP} pins to a 13-V programming voltage before powering up the chip, a specific ROM bootstrap code is launched and waits for orders coming from a dedicated serial synchronous interface (RA1/RA4/RA5 pins).

With a small interface connected to the LPT port of your favorite computer, you can read or write the internal EPROM without any specific programmer and without having to extract the chip from its socket.

I developed my own PC-based in-circuit programmer based on Microchip's programming specifications [2].

Electrically, the standard-grade PIC17C756 needs a supply voltage from 4.5 to 6.0 V, while a special extended device (PIC17LC756) accepts anything from 3.0 to 6.0 V. In both cases, the minimum RAM retention voltage is 1.5 V.

The supply current is 6 mA at 4 MHz but climbs to 50 mA at 33 MHz. Of course, sleep modes reduce consumption down to 1 µA, depending on the selected peripherals, but high performances and low consumption are still difficult to conciliate.

PIC'SPECTRUM HARDWARE

Now that we have a good microcontroller, let's look at the PIC'Spectrum hardware shown in Figure 4.

A small power supply, which is built around U1 (78M05), generates a clean 5 V from a standard 9-VDC power supply (the total power consumption is around 100 mA, which is mainly 50 mA for the PIC and 50 mA for driving the three 75-Ω video outputs). The LED D1 indicates powerup and generates a pseudoground 1.9-V level used by the analog section.

The analog input signal (from an onboard electret measurement microphone or a line input jack) is amplified and low-pass filtered down to 10 kHz

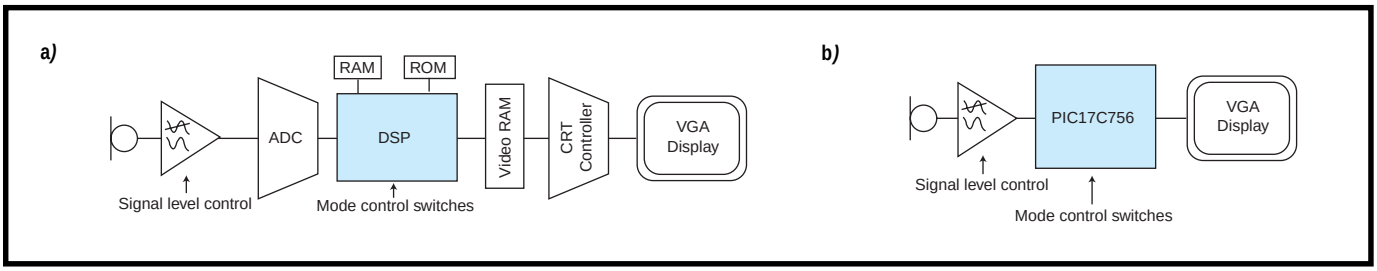


Figure 1a—*Here's a classic block diagram for a spectrum analyzer. The signal is amplified and low-pass filtered before going to an ADC. A DSP calculates the FFT and drives the VGA screen through a video controller.* ***b—****The PIC'Spectrum block diagram is much simpler. The microphone signal is amplified and low-pass filtered and goes directly to the PIC, which calculates the FFT and directly generates the video. It's all in the software.*

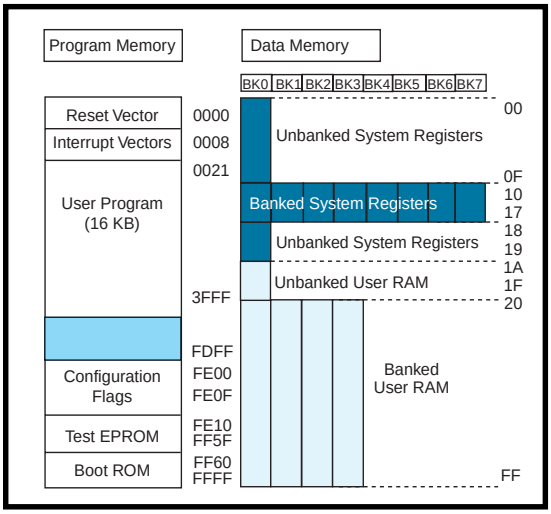


Figure 2—*From this memory map of the PIC17C756, you see that the program and data areas are separated, as usual on a Harvard architecture. The '17C752 has the same memory map, except that there is only 8 KB of user program memory and two user RAM banks.*

by four operational amplifiers (U2, LM324). A potentiometer lets you adapt the amplifier gain to the ambient sound level and serves as an on-off switch. The output signal, centered on the 1.9-V pseudoground level, connects directly to one of the PIC's analog inputs.

The PIC processor is clocked by a 32-MHz crystal (I wasn't able to find a 33-MHz crystal in time). As usual for these frequencies, this crystal is a 3× overtone model.

I needed a damper circuit (L1/C8) to select the correct resonant frequency. Without it, the crystal would oscillate on its fundamental frequency, and I'd end up with a 10.66-MHz clock!

Two switches (K1 and K2) control the current mode (run or hold) and the scaling of the display (linear or logarithmic). They connect directly to RB6 and RB7 because the PIC has selectable internal pullups.

The VGA-compatible video-output connector is directly driven by the PIC. The horizontal and vertical synchronization signals are TTL compatible, so there's no problem there. For the R, G, and B lines (0–2 V/75 Ω), a 150-Ω series resistor does an adequate 5-V to 2-V/75-Ω adaptation, thanks to the high power capacity of the PIC outputs (20 mA/line, 100 mA total for ports A and B).

The 8-pin header J4 handles in-circuit serial programming. Four additional output pins (including one UART output) are connected to a debug header. This provision is useful during the debug phase because it enables you to send debug information to a serial terminal to find out what's happening inside the box when you don't have an in-circuit emulator.

ON THE SOFTWARE FRONT

Of course, when you choose the simplest possible hardware, the software has more to do. Here, the software must:

- do the acquisition of a burst of the analog signal (typically 256 samples at a sampling frequency of 16 kHz)

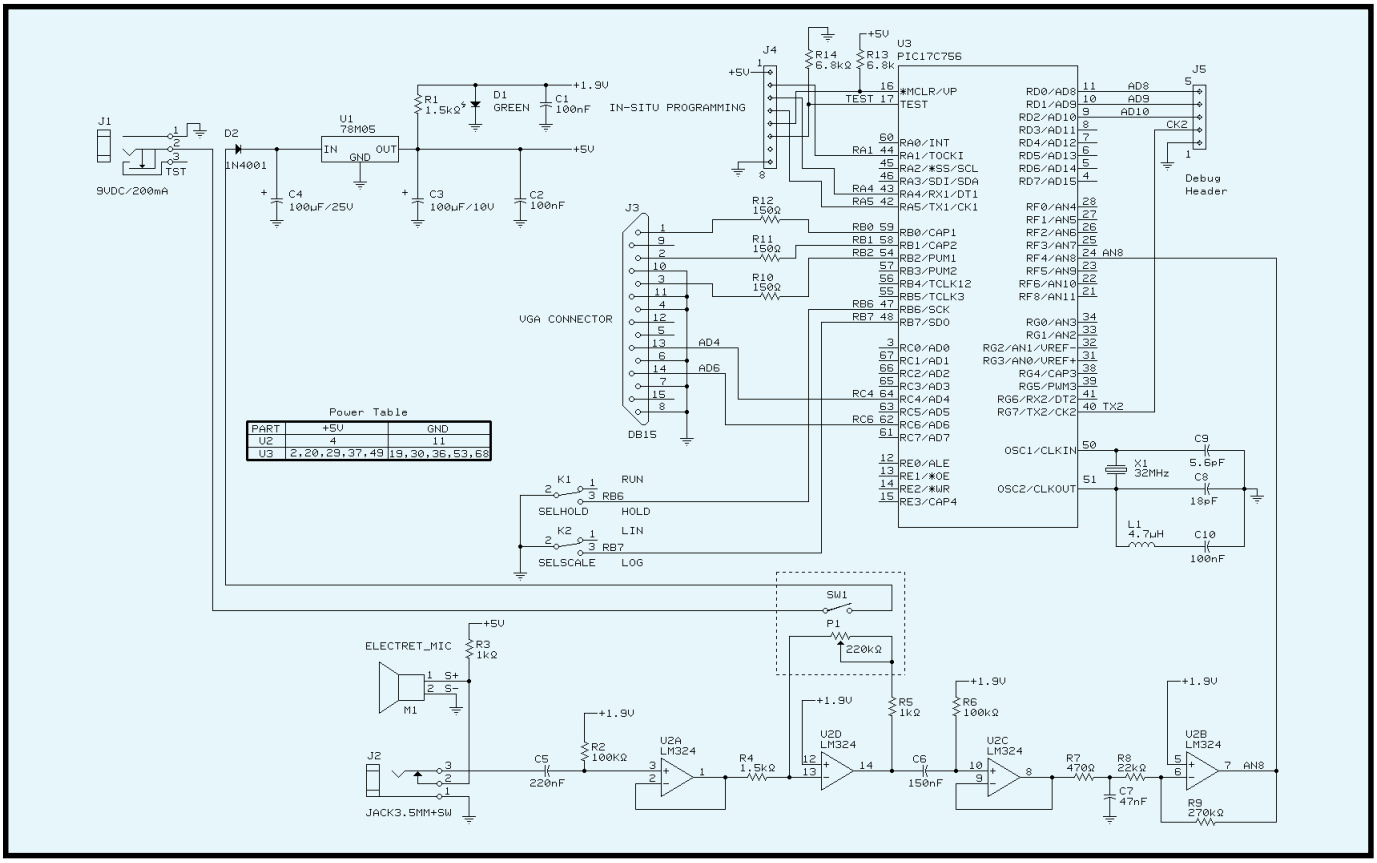


Figure 4—*Around the PIC, there is only a quad operational amplifier, a regulator, and some discrete components.*

- calculate the FFT of these 256 samples, giving an amplitude for each of the 128 frequencies
- calculate the power of each frequency and scale it (depending on the position of the log/lin switch)
- manage the run/hold switch
- do the generation of the VGA video signal in parallel

To do all these tasks while keeping real-time requirements, PIC'Spectrum software is divided into two tasks. The main program is in charge of initialization, device control, and numerical algorithms. The interrupt routine, executed each 31.77 µs (corresponding to the VGA horizontal video synchronization period), is in charge of analog signal acquisition and video generation.

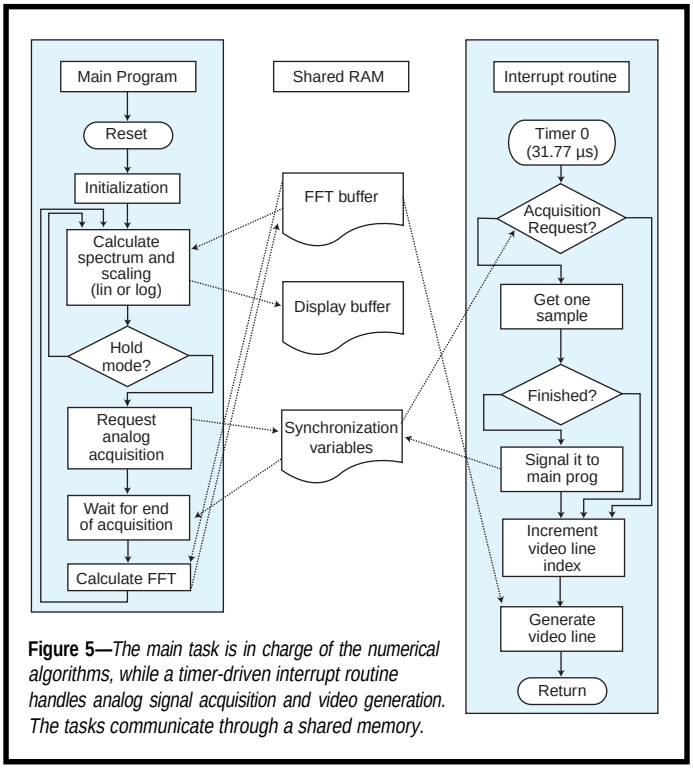


Figure 5—*The main task is in charge of the numerical algorithms, while a timer-driven interrupt routine handles analog signal acquisition and video generation. The tasks communicate through a shared memory.*

These two tasks dialog through three RAM shared structures:

- FFT buffer (256 words, 16 bits each), filled by the interrupt routine with analog samples and used by the main program for FFT calculation
- display buffer (128 bytes), filled by the main program with the length (in pixels) of each of the 128 horizontal frequency bars, and used by the interrupt routine for video generation
- synchronization variables

Figure 5 illustrates the information flow. Since you can get the complete source code from the Circuit Cellar Web site, I'll focus here under on the more specific codes, like FFT calculation and software video generation.

A standard FFT algorithm works on complex numbers (real plus imaginary parts). It takes as an input an array of *n* complex samples and gives an array of *n* complex frequency amplitudes. Here, the input signal is of course only real numbers.

The immediate solution to get its FFT is to add a zero imaginary part to each sample and to use the standard complex numbers algorithm. If you do that, only half of the *n* frequencies in the result are useful (in fact, each value is found twice).

This is a consequence of the well-known Nyquist rule. If you sample your signal at a period of 1/*n*, you can only analyze frequencies with periods above 2/*n*. More problematic, this simple approach uses twice as much memory as is useful and we have only 902 bytes of RAM.

Fortunately, you can use a more sophisticated algorithm, known as real-mode FFT. The idea is to pack two real samples in each complex input value, do a standard complex FFT, and decrypt the resulting complex values to find back the good real figures. You can get the details from *Numerical Recipes in C: The Art of Scientific Computing* [3].

To implement this FFT algorithm on the PIC, I developed a fixed-point mathematical library (source file is `fixed.inc`). It implements a virtual fixed-point machine operating on two 16-bit floating-point registers (RA and RB).

Routines are available for addition, subtraction, multiplication, division, sine, and logarithm, as well as access to the banked RAM. The fixed-point format used is S/2/13 (one sign bit, 2 bits for the integer part, and 13 bits for the fractional part). In fact, writing sinus and logarithm calculation routines in assembler is quite an interesting experience.

OPTIMIZING VIDEO INTERRUPTS

Writing the interrupt routine was another interesting exercise. One interrupt is generated by timer 0 each 31.77 µs. This period equals the horizontal refresh period of a VGA signal in mode 7 (640 × 350).

Because 31.77 µs translates into 254 instructions only (even at 32 MHz), the number of instructions used to do the analog signal acquisition and the video generation must be carefully optimized.

Figure 6 shows the VGA horizontal timing specifications, the corresponding number of instructions that the PIC17C756 could execute at 32 MHz, and the operations done by the interrupt routine in the corresponding time frame (more information on VGA timings is available on the Web [4]).

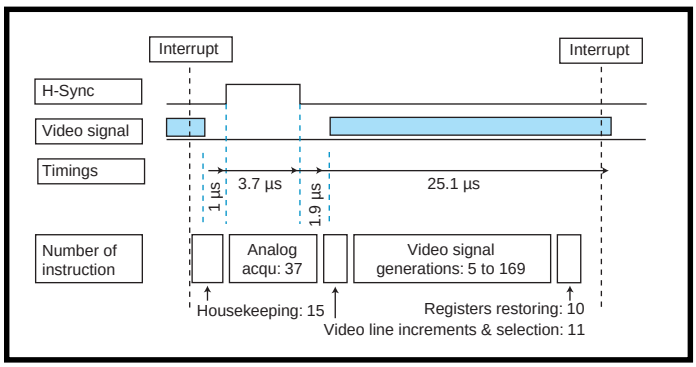


Figure 6—*The top part shows the timings of the video and horizontal synchronization signals, while the bottom part shows the conversion of these timings in number of instructions as well as what the PIC is doing during this time.*

The interrupt routine starts with housekeeping (register saving, timer reload, etc.) and switches on the horizontal synchronization signal. During the 3.7 µs needed by this synchronization, the software manages the acquisition of an analog sample (one signal sample is taken every two interrupts, giving a 15.7-kHz digitization rate).

The current video line number is then incremented and used as an index into a table, giving the address of the routine to use for displaying each scan line (vertical synchronization virtual lines, blank lines, frequency display line, scale or title display, etc.). Each routine manages the switching on and off of the three RGB video signals with good timing.

This brute-force programming method is needed to achieve the result in Photo 2. The longer the program takes between the interrupt and the start of the video display, the more black areas you have onscreen.

Real-time requirements have to be managed very carefully. In particular, the execution time from the interrupt to the start of the video display must be rigorously constant, whatever the results of the if/then tests are.

If it isn't, the video lines do not align. I was forced to add NOPs everywhere—in particular, in the analog acquisition routines—to ensure that the same number of instructions is executed whatever the situation is.

To get the best result, I implemented some strange programming practices. For example, to get a specific delay with 125-ns resolution, I do a calculated jump in the middle of a long sequence of NOPs. It's the only solution I've found, because managing a loop consumes several instructions.

DEVELOPMENT AND DEBUG

Debugging a signal-processing embedded application isn't easy. And, I don't have access to an adequate ICE. Since it might be useful for similar projects, let me briefly explain my methodology (which must not be too bad: PIC'Spectrum actually works!).

After some timing calculations to ensure the feasibility of the concept, I started with a prototyping phase on a PC using a PC sound board as input. I used floating-point calculations, with everything in C (see Listing 1). I then translated the software to use exclusively integer numbers and debugged it.

The third step was to develop a fixed-point library (still on the PC) that prefigures the future PIC-based fixed-point library. I also modified the FFT code to do all calculations exclusively through this library. It was quite easy to write this fixed-point library in PIC assembler, debug it on Microchip's simulator, and translate the FFT routine from C to assembler.

Using the simulator is useful for numerical-calculation software. In fact, it was possible to open two windows on my PC, the first being a standard PC debugger with my C-code FFT, and the second the PIC simulator with the manually translated assembler code. Single stepping between the two codes with a test signal as an input helped me quickly find the more tricky bugs.

When the signal-processing part was completed and debugged, I wrote the real-time part (signal acquisition and video generation) and debugged it as much as I could on the simulator.

This phase was helpful for correcting timing issues related to video synchronization. For example, with a

Listing 1—*Here you see my step-by-step process from traditional float C code on a PC down to integer assembler code on a PIC.*

```
Step 1: C code on PC, float numbers:
float data[NMAX],h1r;
…
data[j]=(data[i]-h1r)/2;   /* calculation step of FFT routine */

Step 2: C code on PC, integer numbers:
word data[NMAX], h1r;
…
data[j]=(data[i]-h1r)/2;

Step 3: C code on PC, use of pseudoregister based on fixed-point
        library:
fixed.h:
  word ra, rb;             /* pseudoregisters A and B */
  …
  m_sub ()                 /* subtraction function a-b->a */
  {ra=ra-rb;};
  …
main program:
  …
  /* data[j]=(data[i]-h1r)/2 */
  m_ldai(i);  /* load data[i] in A register */
  m_ldb(h1r); /* load h1r in B register */
  m_sub();    /* A = A-B */
  m_div2();   /* A = A/2 */
  m_stai(j);  /* store A in data[j] */

Step 4: Last, translation in PIC assembler
Fixed.inc:
  …
  ; m_sub :   subtract B to A (A = A-B)
  m_sub       macro
              movfp rb+1,WREG ; low byte first
              subwf ra+1,1
              movfp rb,WREG   ; and high byte with borrow
              subwfb ra,1
              endm
main program:
              ; data[j]=(data[i]-h1r)/2
              m_ldai i        ; m_ldai(i)
              m_ldb h1r       ; m_ldb(h1r)
              m_sub           ; m_sub()
              m_div2          ; m_div2()
              m_stai j        ; m_stai (j)
```

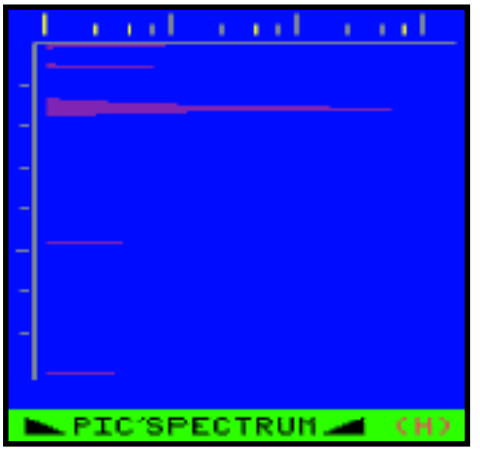


Photo 2—*This video image was generated by PIC'Spectrum. Here, we have a quite clean 1.6-kHz signal. The third and fifth harmonics are visible under the fundamental frequency.*

breakpoint set on each access to the horizontal synchronization pin, it's easy to verify if the timings on this pin meet the VGA specification.

I waited until this phase was successful before cabling the prototype. The first EPROM I burned didn't work 100%, but I got a working video image and something resembling a spectrum display.

Thanks to the in-circuit programming and some debug pins, the software was working soon. In fact, 90% of the bugs found in this last step were related to banking register issues, which are tough to simulate because hardware ports are involved.

WRAP UP

I got the display looking quite good, with an interrupt routine taking ~50% of the available CPU time for video generation. The main program has time to do more than 10 FFT calculations per second, giving a true real-time display.

With my technique, the time spent in the video-generation code depends greatly on the image being displayed. If the screen is full of information, the time left for the main program is near zero. So, you can't use this technique for all video-based projects.

To get satisfactory results with the video-display code, I sacrificed maintainability. Changes to the display may require tremendous efforts in keeping the real-time constraints unchanged. However, it works, and the basic principles may be used to get cheap video display devices like PIC'Spectrum.

For your next video-based project, try to do it with a software video only. If you have strict maintainability requirements, add a CRT controller chip or select a microcontroller with one built in. On the other hand, the FFT implementation is easily reused.

A final note: I'm sure this project wouldn't be possible without a good simulator or an ICE. Thanks to Microchip for providing a good simulator for free on the Web!

Robert Lacoste lives in France, near Paris. He works in the telecommunication industry as senior project manager for GSM wireless base stations development. Robert has 10 years' experience in real-time software and embedded-system development. You may reach him at rlacoste@nortel.com.

SOFTWARE

Complete source code for this article and freeware for the PIC-17C756 in-circuit programmer is available via the Circuit Cellar Web site.

REFERENCES

[1] Microchip Technology, *PIC17C-75X Datasheet*, DS30264A, 1997.

[2] Microchip Technology, *PIC17C-XXX EPROM Memory Programming Specification*, DS30274A, 1997.

[3] S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, New York, NY, 1992.

[4] "VGA Timing Information," www.hut.fi/Misc/Electronics/docs/old/vga_timing.html.

SOURCE

PIC17C756
Microchip Technology, Inc.
(602) 786-7200
Fax: (602) 786-7277
www.microchip.com

Photo courtesy of DVP Inc.

EMBEDDED PC
SEPTEMBER 1998

NPC

MULTIFUNCTION SBC

The Élan104 has been designed for a wide range of embedded control applications with functions specific to the implementation of PC-based, low-cost, medium-scale man-machine-interfaces (MMIs).

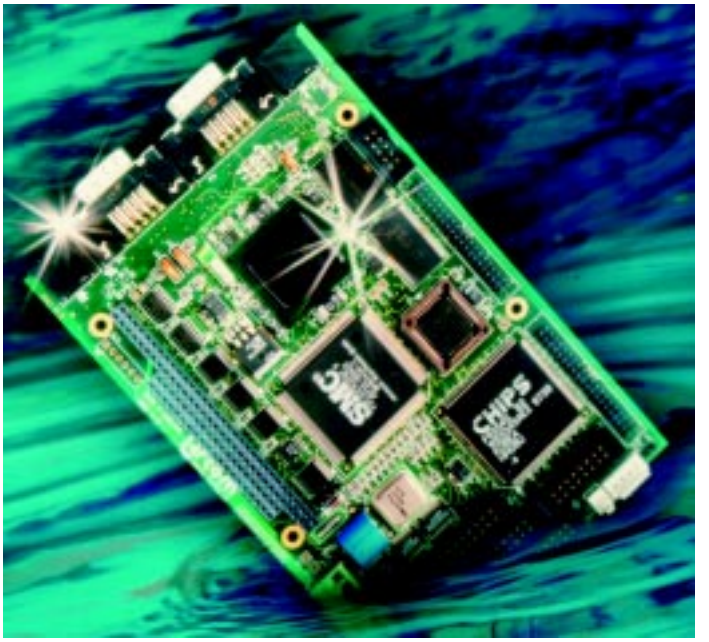
Its Eurocard-format board features the AMD 100-MHz Élan486 microprocessor (full power management) with fully licensed ROM-DOS running in flash memory, 4–16 MB of EDO DRAM, 4–8 MB of flash memory, and 128 KB of battery-backed SRAM.

Other features of the board include dual DMA controllers, interval timers, real-time clock, clock generation circuitry, DRAM controller, and ROM interface. Also included are a bidirectional IEPP/ECP parallel port, 16550-compatible serial port, ISA-bus interface, speaker interface, and JTAG test access port.

Interfaces typically suited to MMI design include matrix keyboard interface, mono LCD graphics (with ¼ VGA, 320 × 200 to 640 × 480 pixel resolution) interface, and SVGA CRT/LCD for interfacing to CRT monitors or color flat-panel displays.

Standard peripheral I/O ports include 16-bit PC/104 expansion interface, floppy and hard disk drive interfaces, two RS-232 serial ports, one RS-485 port for multidrop serial applications, and a single LPT parallel printer port. Standard PC-type mouse and keyboard ports are supplied via PS/2 mini-DIN connectors.

Arcom Control Systems, Inc.
(816) 941-7025
Fax: (816) 941-0343
www.arcom.co.uk

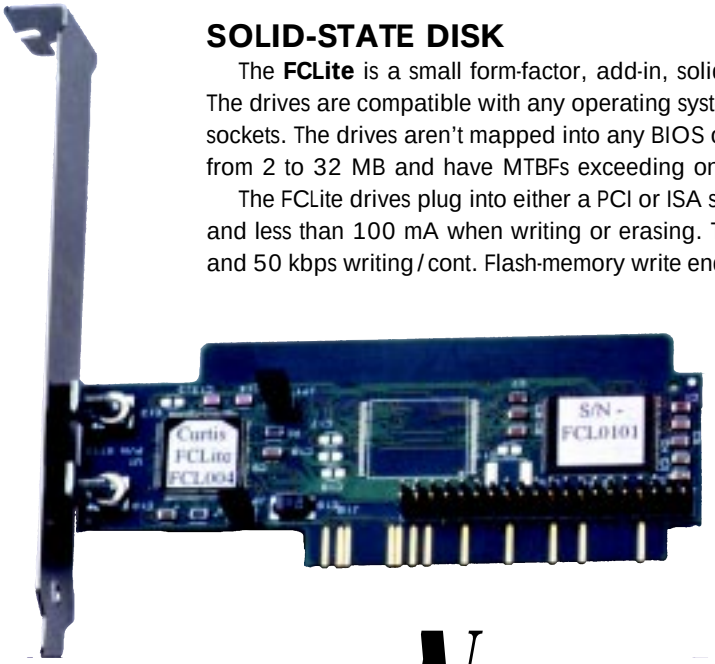


SOLID-STATE DISK

The **FCLite** is a small form-factor, add-in, solid-state flash-memory–based disk emulator that uses an IDE interface. The drives are compatible with any operating system, without requiring special software drivers or onboard BIOS ROM sockets. The drives aren't mapped into any BIOS or extended memory space. They are currently available in capacities from 2 to 32 MB and have MTBFs exceeding one million operating hours.

The FCLite drives plug into either a PCI or ISA slot from which they derive power. They consume 3 mA in sleep mode and less than 100 mA when writing or erasing. The data transfer rate is 4.0 MBps reading, 2.5 MBps writing/burst, and 50 kbps writing/cont. Flash-memory write endurance is in excess of 300,000 write/erase cycles. The FCLite drives can withstand operating shock of 500 G, temperature range of 0–700°C and 5–95% noncondensing humidity environments. Board dimensions are 4.3″ × 1.7″.

List prices for the FCLite drives range from **$129** to **$599**.



Curtis, Inc.
(612) 404-9081
Fax: (612) 404-9175
www.mncurtis.com

Nouveau PC

edited by Harv Weiner

SINGLE-BOARD COMPUTER

The **CMi586DX133 cpuModule** is an ultra-compact, high-speed, fully integrated PC/104-compliant CPU. The cpuModule features a high-performance 133-MHz Am5x86 processor, internal math coprocessor and cache, 8 MB of 32-bit wide surface-mounted DRAM, IDE, and floppy controllers.

Also included are two software-configurable RS-232/-422/-485 serial ports, Extended Capabilities Parallel (ECP) port, keyboard and speaker port, and a watchdog timer. A 32-pin solid-state disk socket supports 1-MB EPROM or 512-KB flash memory, SRAM, NVRAM, or MSystems' DiskOnChip.

The services and functions provided by its Embedded BIOS ensures PC/AT compatibility. BIOS enhancements include quick boot, virtual devices, and solid-state disk support with flash file system. A virtual device mode allows the operator to use the keyboard, video, floppy, and/or hard disk on another PC-compatible computer through the serial port. A nonvolatile configuration EEPROM stores the system setup without needing a battery.

The CMi586DX133 sells for **$598**.

Real Time Devices USA, Inc.
(814) 234-8087
Fax: (814) 234-5218
www.rtdusa.com

PC/104-COMPATIBLE AUDIO MODULE

WinSystems' **PCM-AUDIO-DLX** offers 16-bit stereo audio functions that can simultaneously record and playback (full duplex) voice, sound, and music on a PC/104-compatible module. Based on the ESS Technology ES1888 AudioDrive chip, this module supports PC audio that is AdLib and SoundBlaster Pro compatible.

A four-channel record mixer blends three 16-bit stereo and one microphone input under software control. An independent seven-channel mixer supports playback. Stereo output from the playback is up to 2 W per channel. The volume can be adjusted by software, an external 64-step push-button audio control, or onboard potentiometers. Two joystick inputs provide more input and control options.

The ES1888 chip operates at sample rates of up to 44.1 kHz to provide CD-quality sound. It has a 20-voice, 72-operator music synthesizer. The FM synthesis is backward compatible to the OPL-3 mode. An MPU401 serial port can be used to interface with external wave table synthesizers and MIDI devices.

Special drivers are not needed for newer OSs like Windows 95 and Windows NT. Windows 3.*x* is supported through supplied drivers. WinSystems offers a utility that allows record and playback of audio or voice with control of sample speed, volume, and compression for use with DOS or non-DOS applications. It also supports playback of Windows .WAV audio files in a non-Windows environment.

An optional KIT-PCM-AUDIO-DLX consists of a PCM-AUDIO-DLX board, microphone, two speakers, PC Bus Sound Blaster–compatible board, Windows audio-applications user guide, and DOS/embedded-systems tools and drivers. This kit enables the designer to develop and debug the application code on either a desktop PC or the target embedded CPU running in a Windows or DOS environment.

The PCM-AUDIO-DLX sells for **$250**.

WinSystems, Inc.
(817) 274-7553 • Fax: (817) 548-1358
www.winsystems.com

Nouveau PC

PC/104 PERIPHERAL MODULE

The **MM686** is a PC/104 form-factor board featuring two PCMCIA slots, four A/D video input channels, 5-W SoundBlaster–compatible stereo sound, and support for optional PCMCIA-based Zoom Video MPEG decoding. It also accommodates two joysticks or a MIDI input.

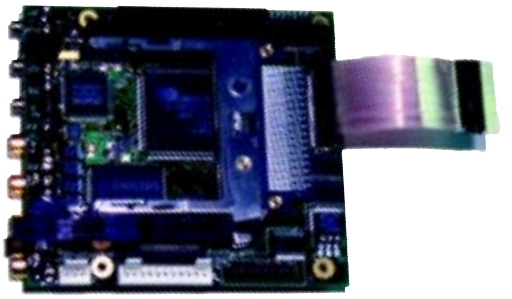
The video inputs support PAL, NTSC, and SECAM decoding and real-time video capture ability. The YUV video output plugs into the YUV connector on 686LCD/S or 686LCD/MG SBCs, allowing for extremely high frame rate video display and capture handled directly by the video controller.

The PCMCIA interface is PCMCIA-2.1 and JEIDA-4.1 compliant and supports all common PCMCIA cards (e.g., modems, ATA flash cards, MPEG decoders, and hard drives).

The SoundBlaster–compatible port has stereo-line and headphone output as well as a direct output from the onboard 5-W amplifier. The MM686 supports a microphone input, analog line input, and direct input from CD playback devices via the Zoom Video port. Volume control is supported in software or via the built-in up-down-mute plug.

This PC/104 board is fully scalable for applications that don't require all of its features. It can be provided as just a SoundBlaster board, just a dual-slot PCMCIA interface, just a video/image processing board, or in any combination.

Inside Technology USA, Inc.
(972) 390-8593
Fax (972) 390-8609
www.inside-usa.com



NouveauPC

Jim Brady

Networking with DeviceNet

Part 1: How DeviceNet Stacks Up

With all the debate on networks these days, it's easy to get confused about the differences between networks, buses, and field buses, particularly when a new technology comes along. Join Jim for the lowdown on DeviceNet.

Reading articles on networks raises a lot of questions—like whether to call DeviceNet a network, a bus, a fieldbus, or what?

The term "bus" is used for industrial networks that control things, as opposed to general office networks that move data. I decided to forget about impressive terms and stick to basics. I want to cover the meat and potatoes of what a developer needs to know to implement a DeviceNet network.

Why are there so many network protocols? Figure 1 shows 15 of them. Why not just use Ethernet?

In the beginning, I didn't ask. I developed DeviceNet interfaces because customers requested them. Then Profibus, then others. However, with time, I wondered: is there an ideal network for a given application?

This month, I show how DeviceNet compares to other device networks and explain how it works. In Part 2, I'll look at a real DeviceNet device, code and all.

Better dust off your C++ books because DeviceNet is object oriented all the way. C works, but C++ fits like a glove.

SORTING THEM OUT

Where does DeviceNet fit into the network menagerie? At first glance, all the networks look quite similar. However, each one has a unique set of features and is designed to move a specific type of data between certain kinds of equipment.

Here, "device" refers to small to mid-size equipment with multiple integrated sensors and/or actuators. Device networks are designed to interconnect these devices.

As devices get smarter and processing power cheaper, even simple devices may have high-level network ports, producing a consolidation toward the high end of Figure 1. But there will always be a need for more than one level of network. Plant engineers don't want to share their Ethernet office network with assembly robots!

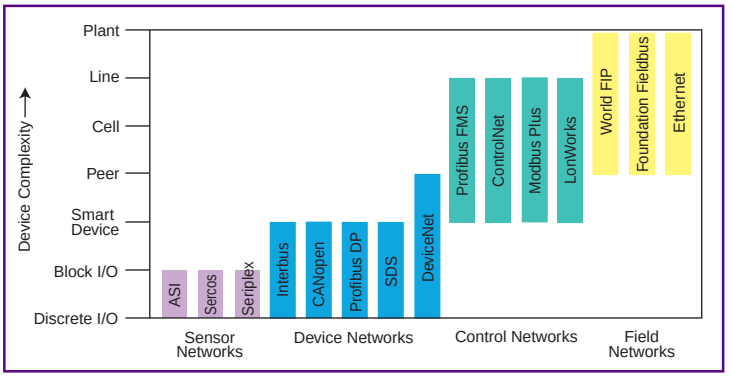


Figure 1—Automation networks at the low end move on/off messages between simple sensors and actuators. At the high end, complex equipment transfers large blocks of data plant wide. DeviceNet handles the middle ground.

NEW BREED

DeviceNet represents a new breed of device networks that offer nifty features, such as hot-plug capability, network-

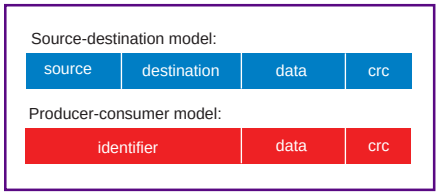


Figure 2—The new producer-consumer model identifies the data rather than the source and destination.

powered devices, peer-to-peer, fiber optics, and fault containment. And if you do it right, your device will be interchangeable with your competitor's. This may not seem good to you, but your users will like it.

Most device networks, including DeviceNet, are deterministic. In this context, "deterministic" simply means that the network can guarantee a drop-dead maximum delivery time for a critical message. Many peer-to-peer networks can't claim this because of the possibility of multiple, destructive collisions.

Ethernet using standard hubs, for example, is collision-based and therefore not deterministic. But it can be made so, thereby becoming a contender for real-time control automation.

If you have a fast processor, combined with Java, and perhaps Windows CE, most of the network code is done for you. I'm more of an 8-bit man myself, but with '386EXs at $16, it's worth considering.

Most new networks, again including DeviceNet, use a producer-consumer (also called data-centric) model as opposed to the older source-destination model. The data is considered central to the message and is what is identified, rather than the source and destination.

This situation increases the effective bandwidth of the network by permitting one-to-many broadcast messaging and time synchronization. Figure 2 illustrates the difference between the two message models.

MOTIVATION

Automation-equipment designers are eliminating old-style point-to-point wiring. They want devices from various suppliers to coexist on the same network. Ultimately, they want interchangeability between same-type devices made by different suppliers.

They're also asking suppliers to make their devices smarter, with better diagnostics. Idiot lights are no longer enough. Diagnostic sensors should provide both alarm and warning levels.

The hope is that the device warns of abnormal levels before it's too late. If a device does fail, it can easily be swapped out for another, possibly one from a different manufacturer, without powering down the network. When the new device comes up on the network, it tells you what it is, what it can do, and lets you know if it's OK.

Table 1 compares features of some popular device networks. At this level of comparison, many differences emerge. CAN-based networks have limited range because they are sensitive to time delay on the line. Most other networks use repeaters to extend their range.

With a 500-m range and a 64-node limit, you wouldn't use DeviceNet to network a large hotel. But, it is an excellent fit in a wide range of applications. The CAN protocol that DeviceNet was built on was originally designed by Bosch for use in autos and trucks. This harsh environment isn't so different from semiconductor fab tools and other automation equipment.

With a short message length, DeviceNet is well suited for time-sensitive messaging. At 500 kbps, a node doesn't have to wait more than 0.26 ms to send.

CAN

There is much literature on CAN [1], including articles by Brad Hunting ("The Solution's in the CAN," *INK* 58) and Willard Dickerson ("Vehicular Control Multiplexing with CAN and J1850," *INK* 69). Here, I'm only going to cover the most important points.

Network protocols such as DeviceNet, SDS, and CANopen—all built on top of CAN—inherit a well-established foundation for reliable real-time communication. At least five IC vendors make CAN controllers that handle all the details of the CAN protocol for you.

What makes CAN so good for real-time messaging? Short message length and priority-based collision resolution.

The latter feature is a major reason for CAN's popularity. If a collision occurs, the higher priority message still gets through intact! In fact, the lower priority colliding node ends up receiving the higher priority colliding node's message.

With most other protocols, if a collision occurs, no data gets through, and this can happen over and over. Not cool for a message to an automobile brake!

CAN requires nodes to listen before sending. If two or more nodes decide to send at the same time, it's the same time to within a small fraction of a bit time. That means the messages line up bit-for-bit.

Because the network line is essentially wire-ORed (see Figure 3), a low bit overrides a high bit. Nodes listen to what they send, and the node sending the high bit will realize that it's receiving a low bit.

At that instant, it knows there is a collision and switches from sending to receiving. The most significant bit of the Connection ID is sent first, so the message with the lower-numbered ID wins the bitwise arbitration and gets through intact.

For this to work, nodes at opposite ends of the cable must have their bits line up to within about half a bit-time. Thus, the round-trip delay of the cable is limited to 1.0 µs at the maximum DeviceNet speed of 500 kbps. The corresponding cable

	DeviceNet	SDS	Profibus DP	LonWorks
Max. range	500 m at 125 kbps	500 m at 125 kbps	9600 m at 94 kbps	2700 m at 78 kbps
Max. speed	500 kbps	1 MBps	12 MBps	1.25 MBps
Max. nodes	64	128	126	32,385
Max. message length	8 bytes	8 bytes	244 bytes	228 bytes
Bus access	Peer, M/S	Peer, M/S	M/S	Peer, M/S
Error resistance	15-bit CRC	15-bit CRC	16-bit CRC	16-bit CRC
Deterministic	yes	yes	yes	in M/S mode
Hot-plug capability	yes	yes	yes	yes
Fault confinement	yes	yes	yes	yes
Line-powered devices	24 VDC, 8 A	yes	optional	optional
Media	twisted pair	twisted pair	twisted pair, fiber, RF	twisted pair, fiber, RF, power line

Table 1—Repeaters are required for Profibus and LonWorks to achieve this range and node count. DeviceNet and SDS are two popular CAN-based networks. Others exist, too, such as CANopen and CAN Kingdom. Profibus is a master-slave protocol that is popular in Europe and gaining support in the U.S. LonWorks, widely used in building automation, is now seeing use in the equipment automation field.

length can be determined by:

$$0.5\ (1.0\ \mu s \times 300\ m/\mu s \times 0.72) = 108\ m$$

where 300 m/µs is velocity of light and 0.72 is the velocity constant of the DeviceNet cable. That's why you have the 100-m limit on cable length at 500 kbps.

MESSAGE RELIABILITY

The CAN controller calculates a 15-bit CRC value for the received data and compares it against the CRC it received. If an error is detected, the node originating the message is notified so it can resend the message.

If the originating node sends too many messages for which it gets an error back, it goes offline. That way, a bad device won't crash the entire network.

Your CPU detects this event by reading the CAN chip's status register. You have the option of staying offline or initiating an error-recovery sequence.

If the CAN controller originating a message doesn't hear back from at least one other device that the message was correctly received, it will resend. Thus, a lonely node will just sit there and send over and over.

When you do get a message from your CAN controller, you know it's correct. And when you tell the CAN controller to send a message, it keeps trying until the message gets through. Pretty good, considering all this is handled by one $8 chip.

To make CAN into a usable network, you need a way to string messages together, establish connections, and handle errors. That's where DeviceNet comes in.

DeviceNet CONNECTIONS

DeviceNet provides a structure for establishing logical connections between devices, releasing connections if they go unused for too long, and stringing messages together if you need more than 8 bytes. It also provides an object-oriented framework to tell you how to structure your network code.

If your device-type is in the DeviceNet library, it even tells you how your device should behave. That part is necessary to make devices completely interchangeable.

Central to DeviceNet is a concept called a connection. Think of it as a telephone connection. When you call someone, you establish a connection. That connection is yours, and other people talking on the same fiber have different connections. The connection breaks when you hang up or in some cases if you stop talking for a while.

In DeviceNet, each connection is identified by an 11-bit number called a message identifier or connection ID. This number includes your device's Media Access Control Identifier (MAC ID), which is a number from 0 to 63, usually set by a switch on your device.

DeviceNet provides a set of 11 predefined connections, called the predefined master/slave connection set (see Table 2). Wait a minute…master/slave?

Yes, that's a letdown after expecting peer-to-peer, but most DeviceNet products on the market today are slave-only devices. The implementation is much simpler and less memory consuming.

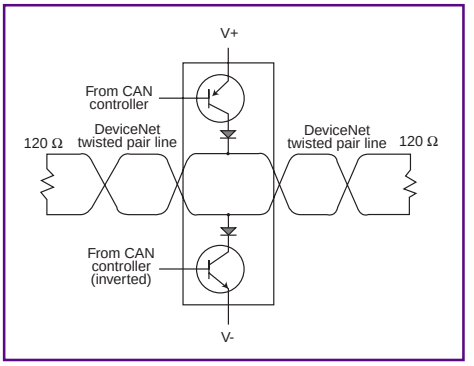


Figure 3—A DeviceNet line with a one-line driver uses a low output from the CAN controller to turn both transistors on while a high output turns both off. It works like a wire-OR, where any low output dominates all other nodes' high outputs.

A peer device must include a lot of code to dynamically establish and configure connections. If you really want to include peer capability, the standard enables you to put it in a device along with the predefined connection set.

In fact, the DeviceNet standard distinguishes between a slave device that also has peer capability, and a slave-only device. In this article, I'm sticking to the simpler slave-only device, which uses only predefined connections.

DeviceNet MESSAGES

DeviceNet has two basic message types: explicit and I/O. The predefined connection set includes one explicit connection as well as four I/O connections of different kinds.

Explicit messages include the path to locate the data of interest. This consists of the class ID, instance number, and attribute ID. They also specify an action to be taken (e.g., set or get). Finally, they include the master's MAC ID because a slave must respond only to its master.

With of all this baggage, explicit messages aren't efficient. They are used mainly for initial configuration, although in theory they could be used for everything. Your device must support this connection, but others are optional.

In an I/O poll connection, the master periodically sends a request saying in effect, "Hey! Send me your data." It is an efficient exchange because the master doesn't need to send any baggage.

When the slave device sees a message with this connection ID, it returns a prearranged set of data. On more complex devices, the master can usually select from various data sets. If only one I/O connection is supported, it's usually this one because it is general purpose.

In an I/O change-of-state (COS) connection, the device sends its data when it changes more than a selected amount. This choice is good for slowly changing data.

The I/O cyclic connection uses the same connection ID as the COS connec-

Table 2—These 11 connections come from two DeviceNet message groups, with the Connection ID made up differently in each case. Group 1 messages are higher priority, used for the slave's I/O messages. All messages originate from the master, except for the Duplicate MAC ID Check and the slave's COS/cyclic message.

Connection ID Bits											Description
10	9	8	7	6	5	4	3	2	1	0	
	Message ID				Slave's MAC ID						Group 1 Message
0	1	1	0	1							Slave's I/O Change-of-State/Cyclic Message
0	1	1	1	0							Slave's I/O Bit-Strobe Response
0	1	1	1	1							Slave's I/O Poll Response
		Slave's MAC ID						Message ID			Group 2 Message
1	0							0	0	0	Master's I/O Bit-Strobe Request
1	0							0	0	1	Reserved
1	0							0	1	0	Master's Change-of-State/Cyclic Ack.
1	0							0	1	1	Slave's Explicit Response
1	0							1	0	0	Master's Explicit Request
1	0							1	0	1	Master's I/O Poll Request
1	0							1	1	0	Unconnected Port
1	0							1	1	1	Duplicate MAC ID Check

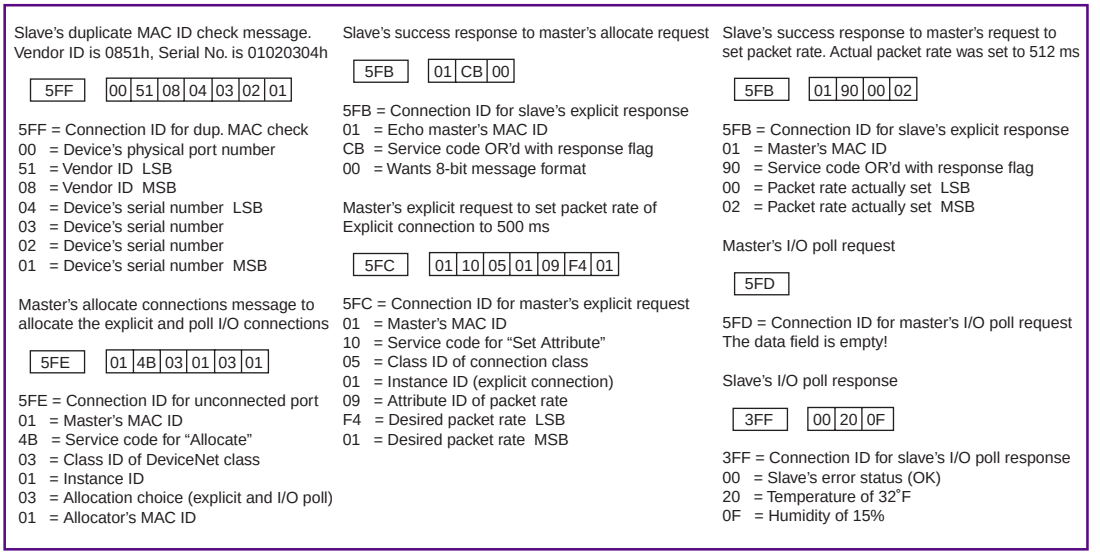


Figure 4—These are messages you see on a DeviceNet analyzer during a typical start-up sequence. The Connection IDs are based on a slave MAC ID of 03Fh. Note the variable-length data field. All message values are in hex, and the least significant bit is always sent first.

tion. The master chooses between the two when it allocates the connection. With the cyclic connection, the device sends its data at a selected rate. This choice works well for rapidly changing data.

In an I/O bit-strobe connection, the device sends only a few bits of data in response to the master's bit-strobe request. This is a good choice for simple on/off sensors.

With DeviceNet, no connections exist until they are allocated. How do you allocate a connection in the first place? I think the designers of DeviceNet must have struggled with this.

It turns out that a special connection always exists—the unconnected port. The master sends an allocate message to the unconnected port to allocate connections and specify which of the predefined connection set it wants to use. Once you have connections, you can send messages over them.

Many CAN controllers have individual mailboxes for incoming messages. You can assign each connection ID to a different mailbox. The Intel and Siemens chips have 15 and 16 mailboxes, respectively, enough for the predefined connection set with room to spare.

This makes your program modular from the start. Different messages come in different boxes.

STRINGING MESSAGES TOGETHER

An explicit message from the master uses five bytes of the eight bytes available for the path, service, and master MAC ID. This leaves only three bytes for actual data.

If the master sends a four-byte long int, two separate messages are needed. These messages are called fragments and are just like a regular message, except the first part of the data field contains fragment information.

For an explicit-message fragment, the first byte contains a fragment flag, and the second byte specifies the fragment type (first, middle, last) and the fragment count. The fragment count is only a six-bit value but can roll over any number of times, allowing for messages of unlimited length.

For an I/O-message fragment, only one byte is used for fragment information (the one specifying fragment type and count) to maximize the space for actual data. In this case, the fragment flag is implied if the produced connection size is greater than 8.

For explicit or I/O fragments, the receiver simply concatenates the data obtained from each fragment, stopping when it sees the flag indicating the final fragment.

With fragmented messages there is the question of whether the receiving device needs to send an acknowledge for each fragment received. It does for an explicit message but not for an I/O message. I/O message fragments are sent back-to-back for maximum speed.

SOME REAL MESSAGES

Figure 4 shows some real DeviceNet messages based on a typical out-of-the-box slave MAC ID of 63 and a master ID of 1. This situation is typical because it gives the slave device the lowest priority and the master the highest.

For clarity, Figure 4 shows only the connection ID and data fields of the CAN message frame. There are two additional fields. One is a length field that tells the receiver how many bytes of data to expect.

The other field is used for acknowledgment. The node receiving the message sends an ack bit if the message was OK.

CAN messages are variable length. Depending on the number of data bytes sent, a frame can range from 44 to 108 bits.

The first message your device deals with is a duplicate MAC ID check message. Before going online, your device must make sure it has a unique MAC ID.

To do this, your device broadcasts two duplicate MAC ID check messages, 1 s apart, addressed to its own MAC ID. All devices receive this message, but none respond unless your device addresses them!

After sending the duplicate MAC ID check message twice and hearing no response, you can go online.

But since you have no connections yet, you must ignore all messages with two exceptions—a message to your unconnected port to allocate connections, or a duplicate MAC ID check message that contains your MAC ID.

Duplicate MAC ID check messages would be from another device hoping to go online but set to the same ID as yours. Your response to this message prevents the offending device from going online.

When you get a message to your unconnected port it will be the master specifying which connections out of the predefined connection set it wants to allocate. The allocation choice byte contained in this message will have bits set which correspond to the connections it wants.

If you support these connections, allocate them and return a success response. Otherwise return an error and don't allocate any connections.

Keep track of the master MAC ID that allocated these connections, because from now on this is your master. A message containing a different master MAC ID is ignored.

For each connection allocated, start a timer that deletes the connection if it times out. For the explicit connection, the timer defaults to 10 s. For the I/O poll connection, it defaults to zero and must be set by the master before the connection is used.

The time-out value is controlled by an attribute called the expected packet rate (EPR), which the master can set. Your time-out value, in milliseconds, equals the EPR × 4. Thus, the EPR for the explicit connection defaults to 2500.

In the I/O message example shown in Figure 4, note that no data, path, or service code is sent with the master's request. The data set returned with the slave's response is already specified in the manufacturer's device profile or electronic datasheet.

More complex devices may have many sets of data the master can choose from. The Master selects the set it wants by changing the slave's produced connection path.

Because no baggage is involved in the I/O message, it's an efficient process. A device such as the one modeled in Figure 5 can send its entire sensor and status data in one I/O message. With explicit messaging, many full-length messages would be needed.

OBJECT LIBRARY

With DeviceNet, a device is modeled as a collection of objects. Each object has attributes and behaviors, and can be implemented directly as a C++ class. Figure 5 shows the object model for a DeviceNet device with two analog sensors.

Each class has a class ID, objects have an instance ID, and attributes have an attribute ID. By specifying these three ID numbers, any attribute in the device can be addressed.

The DeviceNet Object Library is contained in Volume II of the standard. In addition to network-related objects, it includes about 25 objects that model real-world switches, sensors, actuators, PID loops, position sensors, and controllers. There are more on the way, including an Analog Sensor Object that models advanced sensors, with capabilities such as calibration, auto-zero, offset, gain, and setpoints.

CONFORMANCE TESTING

When you complete your DeviceNet product, how do you know it meets the standard? The University of Michigan will test your product to see if it conforms to the rigors of the DeviceNet protocol.

If this sounds too intimidating, you can avoid embarrassment by getting the software and a DeviceNet interface card so you can test it yourself. When you do go to the test lab, bring your laptop and compiler along. The process is designed to be a fix-it-as-you-go experience.

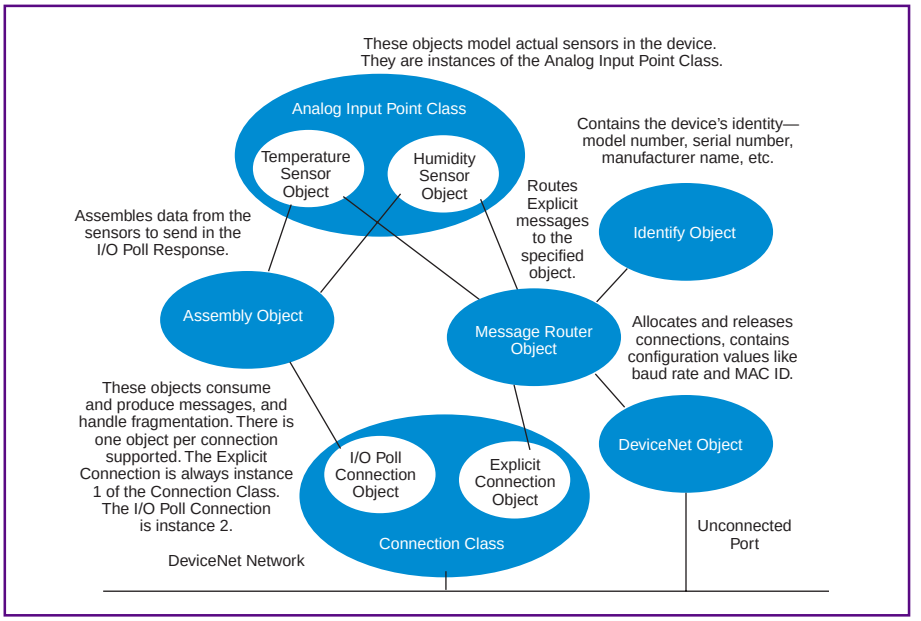


Figure 5—Here's the Object Model for a DeviceNet device with two analog sensors. These objects can be implemented as instances of C++ classes, based on the detailed models provided in the DeviceNet Object Library.

DeviceNet STANDARDS

The DeviceNet standard keeper is the Open DeviceNet Vendor Association (ODVA). It manages the evolving standard and assists vendors in developing their products and the Device Profiles for them. Within the association are 14 active special-interest groups, organized along product lines. The ODVA Web site lists these groups.

You can get your feet wet by getting a DeviceNet catalog at no charge from ODVA. The first chapter has an excellent overview of CAN and DeviceNet. If you decide to jump in, you can purchase the full DeviceNet specification (the CD version includes a good search engine) from ODVA. Later you can become a member, join a SIG, and play a part in defining network standards for your industry.

Next month, I'll turn some DeviceNet objects into C++ classes, embed them in a '386EX, and hang a DeviceNet interface on it. EPC

Jim Brady has designed embedded systems for 15 years. You may reach him at EbaraJim@aol.com.

REFERENCE

[1] J. Schill, "An Overview of the CAN Protocol," *Embedded Systems Programming*, p. 46, Sept 1997.

SOURCES

DeviceNet Information
Open DeviceNet Vendor Assn., Inc.
(954) 340-5412
Fax: (954) 340-5413
www.odva.org

DeviceNet Conformance Testing
University of Michigan
(734) 764-4336
Fax: (734) 936-0347
www.eecs.umich.edu/~sbus

DeviceNet Interface Cards
Huron Networks, Inc.
(313) 995-2637
Fax: (313) 995-2876
www.huronnet.com

National Instruments, Inc.
(512) 794-0100
Fax: (512) 794-8411
www.natinst.com

Softing GmbH
ICT, Inc.
(978) 557-5882
Fax: (987) 557-5884
www.softing.com

SST, Inc.
(519) 725-5136
Fax: (519) 725-1515
www.sstech.on.ca

Real-Time PC

Ingo Cyliax

TCP/IP Networking

With Ethernet chips becoming a dime a dozen, it's a lot easier to justify having your network Ethernet driven. Ingo walks you through all the nitty-gritty steps of how to get your Ethernet-based device working with real-time capabilities.

When I discussed network communication last month, I mentioned that Ethernet is starting to be viewed as a device interface bus. I also told you about the increasing availability of Ethernet-based data-acquisition devices. This month, let's see what it takes to build a prototype of an Ethernet-based device.

First question: what do I define as an Ethernet-based device? Basically, a device that has an Ethernet port as its primary interface. A temperature sensor with an Ethernet port is one example, but I'm not sure about the economics of such a project.

To make developing such a device more interesting to me, it needs to solve one of my problems. So, here goes.

I frequently use a prototyping system called a logic engine, which is essentially a system tester. It has 128 I/O ports and is controlled via a PC-compatible parallel port.

The protocol used over the parallel port needs nonstandard software to drive it. This situation wasn't a problem when a DOS-based machine controlled this board. I simply wrote a program to make the parallel port perform the function I wanted.

With the advent of Windows 95 and Windows NT–based systems, controlling the parallel port becomes an issue. I now need to write a device driver just so my program, which wants to control the logic engine, can talk to it. What's worse, this device driver will probably break when I try it with Win98 or when NTS comes out.

The device I want to discuss is a Ethernet-to-parallel port device that controls the logic engine. In a sense, the parallel port–based logic engine becomes an Ethernet-based logic engine.

This technique has several advantages. First of all, I don't need to write and install a device driver on the system controlling the logic engine. Secondly, I can share the logic engine between workstations or over the Internet.

Finally, I can use a variety of programming languages on many platforms to talk to this device. All my language on the workstation needs now is a network library. I don't have to write various libraries like I did for the old parallel-based interface.

But before getting started on this project, I want to take a look at one of its major components—TCP/IP.

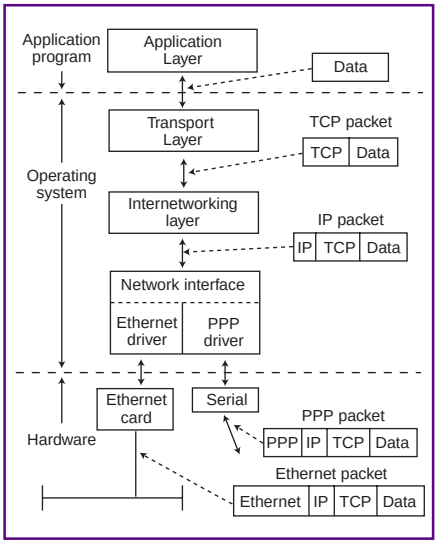


Figure 1—Each layer in a TCP/IP stack has a specific function and adds a header to the data from the application layer.

TCP/IP NETWORKING PRIMER

Recall that TCP/IP implementations use the socket API for application programs to interface to the stack (see "Network Communication," *INK* 96). In a nutshell, when two programs want to establish a TCP/IP connection, they set up a socket which is identified with an Internet address and a port.

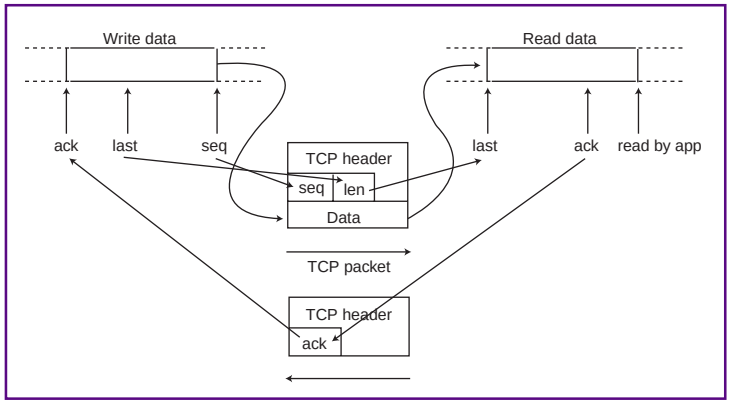


Figure 2—TCP uses a variety of sequence pointers to track how much data is sent and acknowledged out of the 32-KB window. TCP uses a sliding-window protocol, where each sequence number identifies a single byte.

Once the connection is set up, the programs write data to the socket as if it were a file, and the data comes out the other end of the socket, where the other program reads it like a file. The protocol stack treats the TCP/IP as an unstructured bytestream, so it's easy to program for TCP/IP.

But as you probably know, the actual network interface between computers usually consists of a LAN or serial interface, with data being transferred in packets. Let's look at what goes on in the TCP/IP stack to give us this illusion of a connection.

The protocol stack for TCP/IP is illustrated in Figure 1. At the top, we have the application layer, which is followed by the transport and Internetwork layers. The network interface and the network are at the bottom.

The application program, in the application layer, is considered part of the protocol stack. The application usually implements some kind of protocol on top of the unstructured bytestream provided by TCP/IP.

For example, Internet mail uses the Simple Mail Transport Protocol (SMTP) to exchange E-mail messages between different hosts on the Internet. You may also be familiar with the Hypertext Transport Protocol (HTTP) that Web browsers and servers use to communicate.

Note that application-layer protocols are implemented in the application and that the interface between the application layer and the transport layer is the socket API. The rest of the protocol stack—the transport layer through the network interface—is usually implemented in the OS because its implementation is the same for any application.

The transport layer implements the bytestream abstraction of the transmission control protocol (TCP) over the packet-switched Internetwork protocol (IP).

TCP implements what networking folks call a reliable virtual circuit. It is reliable because the implementation makes sure that the data, once received, is not corrupted by noise, arrives in order, and nothing is missing or duplicated.

A virtual circuit is kind of like a phone connection. Once the connection is made, it is maintained until disconnected.

TCP implements this reliable virtual circuit by maintaining a state on each end of the connection. There is state information for each active (or connected) port. The status information maintains the connection state (i.e., whether the circuit is being set up or torn down and whether it is connected).

Sequence pointers are kept to indicate which byte of the stream has been received, acknowledged, and delivered to the receiver. The transmitter also tracks which byte it sent, how much data is left to send, and the last byte acknowledged by the receiver.

Each time one side of the connection has data to send (TCP is full-duplex, by the way), it bundles up the data into a packet and adds a TCP header to the packet.

The packet header includes an end-to-end checksum to make sure that the TCP packet is received intact, a copy of the sequence pointer of the sent data, and a copy of the sequence number for the data it last received, which serves as the acknowledgment. Including the acknowledgement for the data received, while transmitting data in the other direction, is referred to as piggybacking the acknowledgment.

The header also contains the destination port number this TCP packet goes to and the source port number from which it was sent. Figure 2 shows how the various sequence pointers are related.

Once the TCP header has been added, the TCP packet is passed to the Internetwork layer. This layer gets packets to and from other hosts using the network interface.

This layer adds an IP header to the TCP packet. This header contains the destination host address and the source host address. These addresses along with the port numbers from the TCP header let us uniquely identify which TCP connection this packet belongs to as well as the direction it needs to go.

The Internet layer needs to send the packet out over a network using one of the one or more network interfaces. Internet packets can be up to 8 KB long, but most network interfaces can't handle packets of that size. For example, an Ethernet interface can only handle packets 1506 bytes long.

Therefore, the Internet layer often needs to fragment the IP packet into smaller packets. It splits the packet into smaller chunks and copies the IP header into each one.

A field in the IP header tells at what offset in the original packet the fragment packet belongs. The receiving host's Internet layer reassembles the fragments into the original packet before passing it on to the receiving transport layer.

For the Internet to work, we need to be able to connect various networks together via special hosts that have more than one network interface. These hosts are called routers or Internet gateways.

On a router, the Internet-layer implementation also needs to decide which network interface a packet needs to be sent on. A routing table does this job.

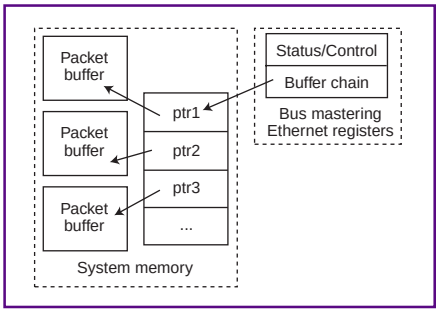


Figure 3—Bus-mastering Ethernet controllers can DMA the packets directly from system memory. The device driver simply sets up a DMA chain to tell the controller where to find the packets.

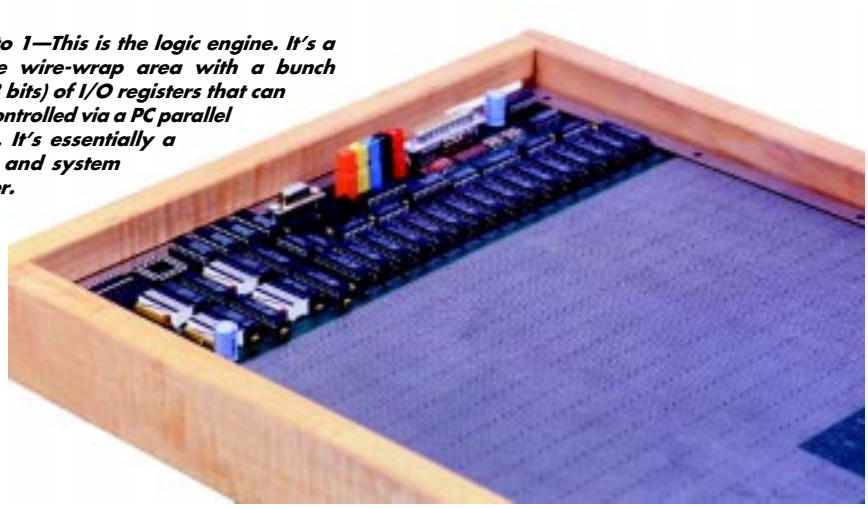


Photo 1—This is the logic engine. It's a large wire-wrap area with a bunch (128 bits) of I/O registers that can be controlled via a PC parallel port. It's essentially a chip and system tester.

Now we're at the business end of the protocol stack, the network interface layer, which has the device drivers for the network devices. A network device may be an Ethernet card, a serial port, or wireless Ethernet interface that connects the host to the network.

Probably the most common network device is the Ethernet interface. Its versatile LAN architecture can be connected to twisted pair, coax, or fiber-based media. An Ethernet card usually has some memory in which the outgoing Ethernet packet is constructed and received. The actual transmitting and receiving is done with hardware on the card.

The network interface driver copies the packet it received from the Internet layer into the Ethernet card's memory, adds an Ethernet header to it, and tells the card to send the packet. When the card transmits the Ethernet packet, it interrupts the network interface driver. Of course, the card also interrupts if a packet was received from the Ethernet.

The network interface usually communicates with the network driver queues. The transmit and receive queues decouple the upper level network code from the interrupt service code for the driver. In a real-time application, this task is especially important because we don't want to spend a lot of time in the interrupt service routine.

The receive interrupt routine simply pulls the packet from the Ethernet card's memory, puts it in the receive queue, and signals the upper level code. The transmit interrupt routine checks the transmit queue. If it has something to send, it pulls off the packet from the queue, copies it to the card's memory, and starts the transmission.

Newer cards—especially PCI-based cards that can do bus mastering and access system memory—use DMA to send the packets directly from system memory. For these cards, the network driver simply maintains a list of buffer pointers that point to receive and transmit buffer memory. These cards don't require the CPU to copy the data around during interrupt time. Figure 3 shows the basic idea.

TRANSFERRING DATA

We also need to discuss transferring data via TCP/IP. There are two Ethernet speeds, 10 and 100 Mbps.

Most100-Mbps cards automatically detect if the network they're plugged into is 100 or 10 Mbps. At 10 Mbps, Ethernet can transfer almost 1 MBps of data in one direction.

However, you most likely won't see that high of a throughput. In many cases, if there's more than one host on the Ethernet, there will be some overhead in trying to access the media and deal with contention.

For Ethernet, this situation can be really bad. The utilization may be as low as 30% when Ethernet is saturated with tens of hosts.

That means all hosts together are only able to transfer about 300 kbps on a highly congested 10-MBps Ethernet. Of course, if only two hosts are involved, we should be able to achieve almost full utilization of the Ethernet.

Another problem is latency. To send data to another host, the data has to traverse the protocol stack, possibly get copied into the Ethernet card's memory, be transmitted over the Ethernet, be received and copied form the receiving card, and passed back up the protocol stack on the receiver and then reverse this to get an acknowledgment back.

On a lightly loaded 10-MBps network with high-performance Ethernet cards, you

might see latencies down to 1.0 ms. However, if the network gets loaded, the latency can easily reach 10–20 ms or more. Basically, make sure the Ethernet is not very loaded if you expect high utilization or throughput. Also, this means there are upper bounds to what can be done with Ethernet as a device bus.

PUTTING IT TOGETHER

Let's look at the nuts and bolts of implementing a network-based logic-engine device. As I mentioned, I need to build an Ethernet device. In this case, it is an interface to the logic engine, a prototype developed by the Computer Science Department at Indiana University.

The logic engine (see Photo 1) is used in an undergraduate digital design lab and for research projects. In both cases, it serves as a sort of chip/system tester. Figure 4 gives an overview of the system we plan to build.

The logic engine is an array of 8-bit I/O registers which are read and written by a host via a parallel port interface. Figure 5 illustrates the logic engine's basic design.

The protocol controlling the logic engine is simple. An 8-bit address is latched by first writing it to the data port of the PC parallel port interface and then strobing the address strobe, which is one of the control signals on the parallel interface. Once the address has been latched, the host reads and writes the register.

Writing it is simple—the host sets the data in the data port of the parallel interface and strobes the write strobe. Reading is a little harder. The content is read by selecting the nibble (upper/lower) of the I/O register and reading the status port of the parallel port. It takes two transactions to read the port in the interface.

Now you have an idea of how the logic engine interface works, so let's go to work. As a minimum, you need a CPU board with a PC parallel port interface. You also need an Ethernet interface and enough memory to hold the system software.

For the prototype, I chose Versalogic's VSBC-1 motherboard and their PCM-3660 NE2000-compatible Ethernet module. While it's overkill for this application, I had one on hand. Using a PC-compatible single-board system with integrated Ethernet, such as Versalogic's PCM-4890, would be more economical.

Next, I selected an RTOS. My biggest concern was that it has a TCP/IP protocol stack and supports the Ethernet card.

I chose an NE2000-compatible Ethernet card, which is the most common. Nearly all TCP/IP implementations have a driver for it. Eventually I want to ROM this application, so a ROMable RTOS would be good.

For the prototype, I chose QNX, mostly because I can develop the code on the system while prototyping by booting the OS from the network or hard disk. Once it's done and I don't need the development system components, I can just ROM the essential modules needed from the OS.

Another OS to consider is RT-Linux, which is a freely available Unix-like Linux OS with a real-time extension. This also lets me develop software online and then embed it.

Although RT-Linux is not as easy to ROM as QNX, it is possible. I can also boot RT-Linux over the network.

I also need to talk about the application-layer protocol I implemented for this project. Recall that the application-layer protocol uses the socket API to communicate with the remote system—here, the logic engine interface. The protocol for the logic engine is simple.

The command is of the form:

```
<cmd><port><wdata>
```

where <cmd> can be r for read, w for write, or q for quit. <port> and <wdata> are two-digit hex numbers. Even though only the w command needs data, always send a data byte along. For the r and q commands, the data has no meaning. It just makes each command the same length, which makes buffer management easier.

For every command received, the logic-engine interface sends back a byte again in a two-digit hexadecimal encoding. If the command is a read, it's the value of the register addressed. So, the response is:

```
<rdata>
```

Listing 1 shows all of the application-specific code for this project. The program starts off by setting up a server port. When a connection from a client comes in, you spawn a process to handle the connection and it sits in a loop waiting for commands.

For each r and w command, you perform the transaction on the parallel port and return a byte. If the connection is broken or q is received, the process terminates.

What about the client side? Here, I use a task control language (TCL) program to test the interface. You can find free TCL

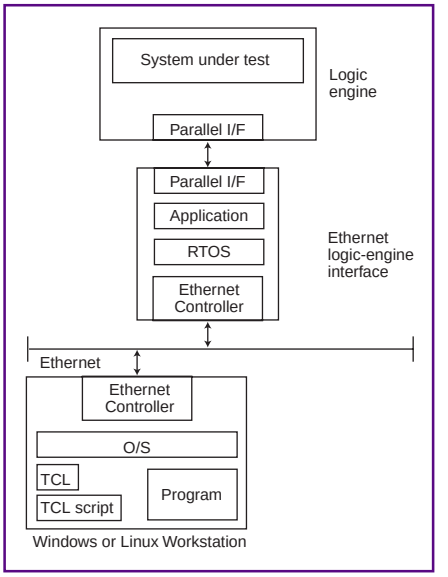


Figure 4—This is the system diagram of my Ethernet device. By putting devices on the Ethernet, I make devices more portable and accessible.

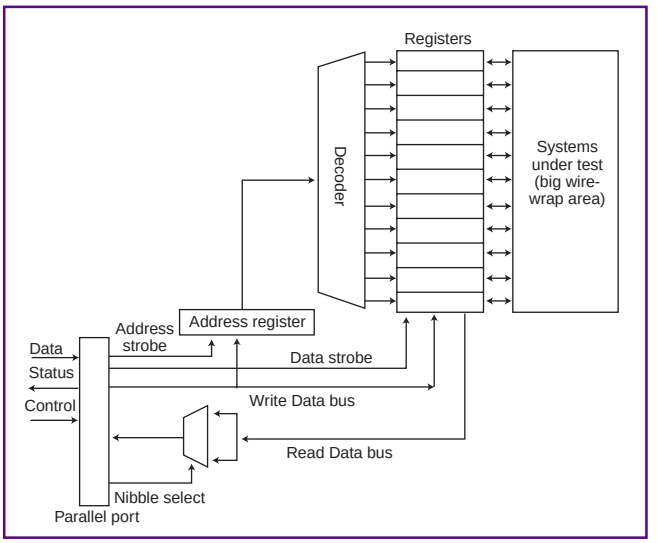


Figure 5—You can think of the logic engine as a big array of 8-bit registers that can be written and read via a PC parallel port. It serves as a test bed for digital circuits at Indiana University.

Listing 1—Here is all the application-specific code I need for the Ethernet device. The rest of the system implements the TCP/IP stack.

```
#include <sys/wait.h>
#include <netinet/in.h>
#include <stdio.h>
#include <asm/io.h>
#define LPTDAT 0x278
#define LPTSTAT 0x279
#define LPTCTL 0x27a
#define MyPort 4321

main(){
  int s,ns;
  struct sockaddr_in sin;
  int slen;
  /* establish command port using TCP/IP */
  s = socket(AF_INET, SOCK_STREAM, 0);
  sin.sin_addr.s_addr = htonl(0);
  sin.sin_port = htons(MyPort);
  bind(s,(struct sockaddr *)&sin,sizeof(sin));
  listen(s,1);
  while(1){
    slen = sizeof(sin);       /* wait for connection */
    ns = accept(s,(struct sockaddr *)&sin,&slen);
    if(ns < 0)
      continue;
    if(!fork()){   /* spawn off process to deal with connection */
      process(ns);
      exit(0);
    }
    close(ns);     /* close off socket in parent */
  }
}
#define CMDLEN 5
process(fd)
int fd;{
  char buf[128];
  int n,l;
  char req;
  int port, data;
  iopl(3);
  while ( 1 ){
    l = 0;
    while ( l < CMDLEN ){
      n = read(fd,&buf[l],(CMDLEN-l));
      if(n < 0) goto done_err;
      l += n;
    }
    buf[CMDLEN] = '\0';
    sscanf(buf,"%c%02x%02x",&req,&port,&data);
    switch(req){
      case 'q':
        goto done_normal;
      case 'w':
        le_write(port,data);
        break;
      case 'r':
        data = le_read(port);
        break;
      }
      sprintf(buf,"%02x",data);
      if(write(fd,buf,2)<0)
        goto done_err;
    }
  done_normal:
    close(fd);
    return(0);
  done_err:
```

(continued)

Listing 1—continued

```
    close(fd);
    return(-1);
}
le_read(p)
int p;{
  int x;
  outb(p,LPTDAT);      /* address */
  outb(0x01,LPTCTL);   /* address strobe */
  outb(0x00,LPTCTL);   /* reset address strobe */
  outb(0x02,LPTCTL);   /* read high nibble */
  x = inb(LPTSTAT)>>4;
  outb(0x0a,LPTCTL);   /* read strobe lower nibble */
  x |= (inb(LPTSTAT) & 0xf0);
  outb(0x00,LPTCTL);   /* reset read strobe */
  return(x);
}
le_write(p,x)
int p,x;{
  outb(p,LPTDAT);      /* address */
  outb(0x01,LPTCTL);   /* address strobe */
  outb(0x00,LPTCTL);   /* reset address strobe */
  outb(x,LPTDAT);      /* data */
  outb(0x04,LPTCTL);   /* write strobe */
  outb(0x00,LPTCTL);   /* reset write strobe */
}
```

Listing 2—Since the device is now accessible via TCP/IP, I can use scripting languages like TCL to control the device from several platforms. This program also runs under Windows or Linux without changes to test the logic engine interface.

```
# start socket to logic engine
set fd [socket "tmp2" 4321]
# turn on tristate buffers
puts -nonewline $fd "w00ffw01ff" ; flush $fd
puts [read $fd 4]
# now loop through a bunch of numbers
for {set d 0} {$d < 256} {incr d}{
  # write the sequence
  set p 4
  while { $p < 16 } {
    puts -nonewline $fd [format "w%02x%02x" $p $d ];
    flush $fd; read $fd 2;
    incr p;
  }
  # read and compare
  set p 4
  while {$p < 16}{
    puts -nonewline $fd [format "r%02x00" $p ];
    flush $fd
    set y [read $fd 2]
    set x [format "%02x" $d]
    if {$y != $x}{
      puts "data at $p read $y doesn't match written $x ."
    }
    incr p;
  }
  puts -nonewline "." ;  flush stdout
}
puts ""
# OK, all done.
puts -nonewline $fd "q0000" ; flush $fd
puts [read $fd 2]
exit
```

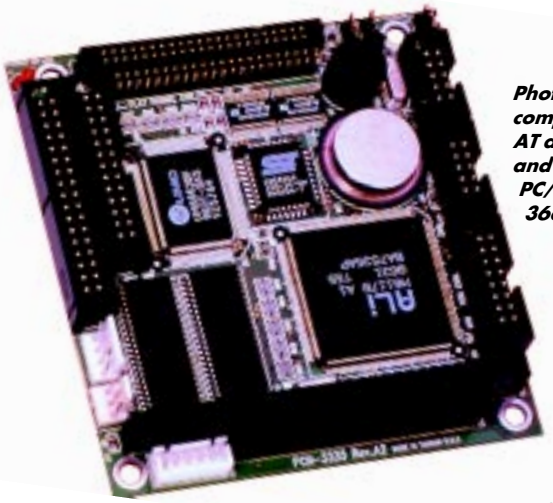


Photo 2—The PCM-3335 is a complete i386 computer module that includes common PC/AT devices such as floppy and IDE controllers and serial and parallel ports. Add an NE2000 PC/104 Ethernet module, such as the PCM-3660, and you're in business.

implementations for Windows, Unix, and Macintosh.

Listing 2 offers a simple test program. It connects to the server and tests some of the logic engine ports. That's all there is to it.

Well, almost. It's OK for a prototype, but there are some things that need to be addressed. As I have mentioned, the prototype is overkill for this application.

You only need a low-end '386-based CPU. Also, I haven't ROMed the application. Some of that depends on the RTOS we use as well as the particular CPU module.

I also haven't addressed how an Ethernet-based device finds out its Internet address. My development system boots from disk, so it's easy to configure the Internet host address in one of the start-up scripts via the console I use for development. In an embedded Ethernet device, however, there probably isn't a console.

But don't despair. There are solutions.

One option: include a serial port on the Ethernet device, which would run a small command-line-based interface that lets you configure things like the Internet host address and store data in a flash file system. But, this solution mars the beauty of having an Ethernet device—it would have a console and seem more like a computer.

A better solution is to use one of the protocols available for this purpose. There's the dynamic host configuration protocol (DHCP) and the boot protocol (BOOTP). The idea: you can use a Windows or Unix machine as an Internet host address clearinghouse.

When turned on, the Ethernet device sends a DHCP or BOOTP broadcast over the Ethernet. A workstation configured to be a DHCP or BOOTP server then assigns an Internet address to the Ethernet device and sends a response. The Ethernet device uses this address until it's turned off.

ON THE SEARCH

Although the logic engine interface isn't a very general example, it shows you the essentials for building an Internet device.

It would be easy to replace the logic engine interface with a PC/104 A/D board that plugs into to the Ethernet device, then change the code to read out the data from the A/D board and send it to the workstation using a TCP/IP stream. Voilà, an Ethernet-based data-acquisition device. I'm sure you can think of other applications as well.

One thing that bothers me is that it's still kind of expensive to implement this sort of thing. At one end of the spectrum, you can find reasonably priced '386-based PC/104 cards that require an Ethernet module to be added. Photo 2 shows a PCM-3335 module like this from Versalogic. There are also high-end '486- and Pentium-based super cards with integrated Ethernet and SVGA and so forth, but that's overkill.

It seems like there should be a no-frills i386-based PC/104 module with integrated NE2000-compatible Ethernet controller. Well, I'll keep looking. RPC.EPC

Ingo Cyliax has been writing for INK *for two years on topics such as embedded systems, FPGA design, and robotics. He is a research engineer at Derivation Systems Inc., a San Diego–based formal synthesis company, where he works on formal-method design tools for high-assurance systems and develops embedded-system products. Before joining DSI, Ingo worked for over 12 years as a system and research engineer for several universities and as an independent consultant. You may reach him at cyliax@derivation.com.*

SOURCE

CPU module with integrated Ethernet, Ethernet modules
Versalogic Corp.
(800) 824-3163
(541) 485-8575
Fax: (541) 485-5712
www.versalogic.com

Applied PCs

Fred Eady

Radio Frequency and Micros

Part 1: Transmitter and Receiver Modules

Ever take on a contract, then realize it's a speck over your head? Fred, too. However, DVP's RF devices are out-of-the-box solutions. They let you operate many transmitter/receiver pairs just like an Ethernet network. No need to be an RF guru.

Back in the early 1970s (before some of you youngling engineers were born), I have fond memories of traveling by car to the FCC field office in Atlanta, Georgia. Believe it or not, I used to be an RF head.

All those road trips weren't just for fun. I was taking FCC licensing exams. I cut my teeth in professional electronics by working in radio.

Yep, as you can tell from my writing, I did a lot of talking in my early years of broadcasting. I started out by doing the nightly rock show and the weekend graveyard shifts on the local AM station.

I tried television as a cameraman and even auditioned at WSM-TV in Nashville, Tennessee. Good thing I was young, dumb, and ugly. Couldn't see myself now doing weekend weather! However, I did manage to land a job at the big-city FM station working for National Public Radio.

By then I was a newsman, but management saw too much of me in the innards of the consoles. If you couldn't find me, just call the chief engineer. I was usually hanging out with him at the antenna site way up on the mountain.

Well, one thing led to another, and I finally made a trip to Atlanta to test for what was then called First Class Phone. Over the years, the First Class was replaced with General. I still have that rag hanging in the Circuit Cellar Florida room.

With all that, you're probably scratching your head wondering why all of a sudden you need to know my life story. Well, my point: there's a lot of mystery in RF product design and implementation. Odds are, if you don't do RF every day, you most likely can't do RF at all.

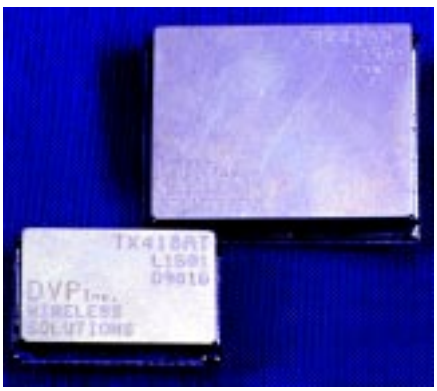


Photo 1—Even I could implement these tricky little RF modules.

If you think like I do, you see RF as a bunch of plumbing tied together with capacitors and coils. Yet RF is very much in our everyday lives.

Take your garage door opener, for instance. It's probably not infrared controlled. I bet it's RF. You can't use your TV remote control to disable your car alarm. That's RF, too.

I could go on and on about what is and what isn't RF. The fact is that in the end-user community, RF technology is used as much as microprocessor technology.

To stir in a little mud, RF and micros make a cute couple. So, if you're a micro head and need to develop an application that assumes you know RF, should you get in the car and head for Atlanta? Nope. Just pick up the phone or get on the 'Net and contact DVP.

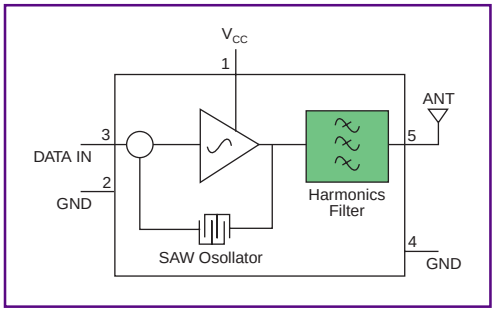


Figure 1—I'm pretty sure there's more than three parts in that little box.

NO-BRAINER RF

RF stuff always fascinated me. When I was a kid, I souped up my walkie-talkies by adding power stages and home-made antennae. It still amazes me that electronic components pumping electrons through listless wire become something useful.

In fact, I was so enthralled with RF, every year I purchased a copy of the *Radio Amateur's Handbook*. I even tried to build some of the RF circuits shown in its pages.

I had lots of fun with my two-way toys, but I was unknowingly breaking every FCC rule that applied to those types of devices. Now that I'm sorta a responsible adult and a professional, I can't afford to break the rules to make an RF application work.

To add pain to my RF misery, I write code. I don't design RF. About the closest RF energy thing I come into contact with is microprocessor clock speeds.

There must be a lot of people like me out there because DVP developed a line of what I call no-brainer RF modules that let me apply my micro knowledge in the RF galaxy. DVP offers off-the-shelf, out-of-the-box, work-right-away RF solutions.

I happen to have a few of their tricky RF modules in the *INK* Florida room. So, let's talk about them before we apply them.

SENDING IT OUT

These transmitter modules are low power and are designed for unlicensed wireless operation conforming to FCC Part 15 regulations. They're commonly used in car alarms and keyless entry systems. Micro heads like us are most likely to use these modules for telemetry and data-collection applications.

DVP's transmitter modules operate at five different frequencies—303, 315, 418, 433, and 916 MHz. Transmitter modules are designated as TXyyyAT or TXyyyAS. The *yyy* is the frequency and "AT" denotes through-hole. "AS" represents the surface-mount version.

The smaller module in Photo 1 is a TX418AT. As you can see, the transmitter module is contained within a rugged RFI/EMI shield. This puppy can go anywhere our embedded engines can.

The no-brainer design eliminates any tuning and requires no external components to effect the RF field. The only companion components needed are an an-

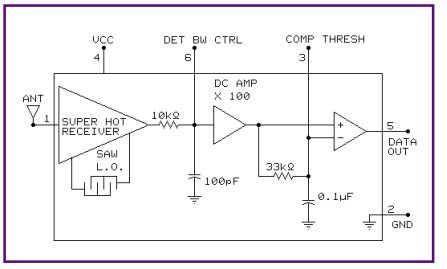


Figure 2—The documentation states that the 33-kΩ resistor here is really a 20 kΩ.... Anyway, note the SAW oscillator and superheterodyne front-end.

tenna, power, and some sort of data encoder.

All of the transmitter modules employ 100% on-off keyed (OOK) amplitude modulation. Also, each transmitter module is SAW stabilized and uses an internal harmonic filter to ensure compliance with spurious emissions limits. Figure 1 shows you that simple is as simple does.

The transmitter modules operate with power sources as low as 2 VDC and as high as 12 VDC. Power consumption at 3 VDC is around 5 mA. When used with a quarter-wave antenna and the matching DVP receiver module, this device can achieve transmission distances of up to 1000′ with a transmitter output power of 1 mW.

TAKING IT ALL IN

Matching receiver modules, like the larger module in Photo 1, are available for all DVP transmitter modules. Like the transmitter modules, the receivers are no-brainer components, too. They're designated by an RX prefix.

These little packages are extremely sensitive (–112 dbm) with good superheterodyne front-end circuitry that provides high selectivity. This feature is an absolute requirement if you plan to use these modules in heavy RF-traffic areas.

Unlike the transmitter modules, these may require a couple external capacitors in addition to the power source and antenna to effect RF reception. These added external capacitors let you tweak the sensitivity and lower the bit error rate. You select the values of these caps depending on the data rate, transmission length, and dead period between transmissions.

The data-recovery circuitry converts a variable-amplitude analog input signal to a fixed-amplitude analog signal. The amplitude of the incoming analog signal depends on the signal strength.

Because the data received is digital and thus time related, pulse width distortion and noise must be minimized during conversion. The capacitor connecting to pin 6 (C6) of the receiver module sets the low-pass filter cut-off frequency. Keeping the cut-off frequency as low as possible without distorting the data gives the best sensitivity.

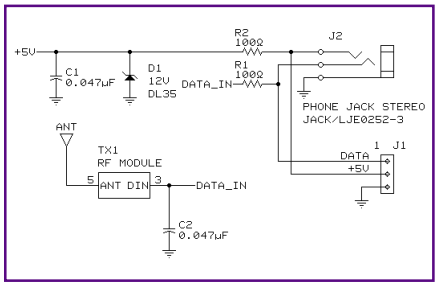


Figure 3—My antenna boards didn't come with the phone jack. Looks like its intended purpose was to accept higher voltages.

Photo 2—Two cans of RF whip-butt.

The receiver modules have an internal RC filter consisting of a 10-kΩ resistor and 100-pF cap. The cut-off frequency is derived from:

$$F_{cut-off} = \frac{0.16}{RC}$$

For a cut-off frequency of 16 kHz, a 0.001-µF capacitor is connected across pin 6 to ground. This also establishes a maximum data rate of 5 kbps.

A second external capacitor (C3) uses an internal 20-kΩ resistor and 0.1-µF cap to form a low-pass filter that finds the average value of the datastream. This sets the signal slicing level of the comparator.

The idea here is to avoid lengthy preambles. However, a preamble of some sort is necessary to assure that the comparator threshold is set above the noise level when the data is received.

This capacitor must be sufficiently large to prevent the comparator threshold from falling into the noise during dead periods

within transmission packets. If the transmission duty cycle is exactly 50%, then no cap is necessary across pin 3.

Interestingly enough, no cap is needed if the selected cut-off frequency is 16 kHz. At 16 kHz, the maximum data rate is 10 kbps. Otherwise, some experimentation is necessary to choose the correct value for the slice cap depending on the format of the data you're receiving.

The databook mentions a value of 2.2 µF for C3 and 0.001 µF for C6. A block diagram of the receiver module is shown in Figure 2.

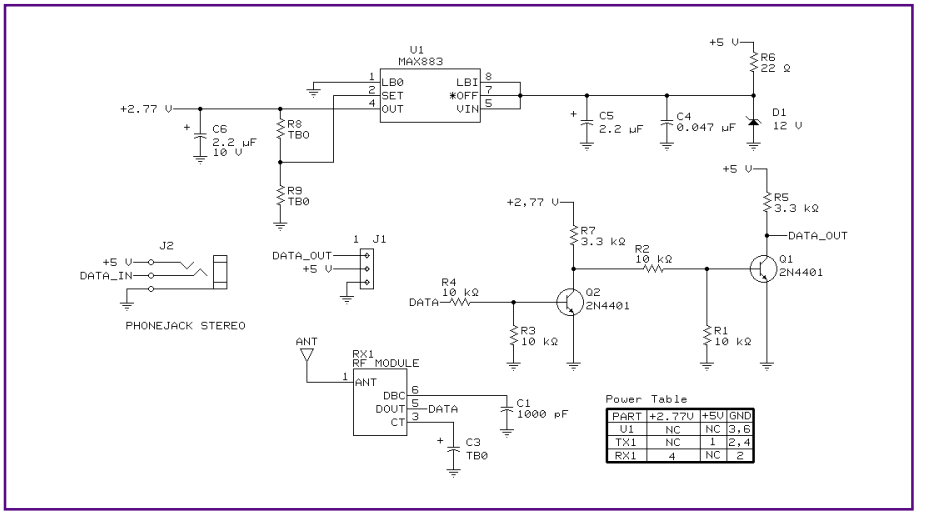


Figure 4—Note the circuitry to boost a couple of batteries to +5 VDC. Also notice the transistor level shifter/buffer.

BOARDS FOR ANTENNA HEADS

For us embedded types, the real work is done by the program. In RF, the real work is done by the antenna. Simply stated, RF performance is directly tied to the proper design of the antenna system.

Things I used to know about like gain, polarization, impedance matching, and coverage patterns all play a unique but important part in antenna design. Fortunately for you, I won't go into an antenna-crazed monologue. But, I do want to point out some antenna tidbits as they relate to these DVP modules.

First of all, the FCC and European regulations place no restrictions on the receiving antenna in low-power unlicensed devices. However, the bad news is that the transmitting antenna must be integral or internal, or it must use a unique connector. In other words, no BNC or RCA connectors from Radio Shack.

Radiated power is the key here, so you could pump a kilowatt into the antenna as long as the radiated power is within regulatory limits. This means that you as the designer cannot use an antenna that can be altered or replaced by the end user. The differing antennae may (and probably will) exhibit different characteristics that would alter harmonic and spurious emission levels.

To help the inexperienced RF designer, DVP supplies what they call Antenna Boards to ease the pain. I happen to have a pair of 315-MHz antenna boards, and they appear in Photo 2.

Antenna boards are ready-to-roll, matched-pair transmitter and receiver modules with integral loop antennae. All that's needed to put these little babies into operation is a +5-VDC power source and a data encoder or data input source.

These devices are perfect for the embedded RF designer because TTL-compatible data can be entered into and retrieved from the antenna boards directly. There is no onboard encoder or decoder, so the embedded designer must code his or her own data-transmission scheme.

Figure 3 shows the innards of the transmitter board. Figure 4 blows up the surface-mount components for the receiver board.

RF RELIEF

Thus far, I've discussed the basic building blocks necessary to get the RF-impaired embedded design engineer into the bounds of RF-ville. For those of you needing to travel into interstellar RF-space, DVP has the vehicle for you.

Photo 3—The batteries are included, and there are even blinking lights for me! Don't overlook the loop antenna etched onto the board.

Photo 3 shows you a matched set of receiver and transmitter evaluation boards. These boards differ from the antenna boards in that they are complete out-of-the-box units that include onboard hardware encoders and decoders from Holtek.

As an accomplished embedded coder or engineer, you're probably giggling at the thought of using a hardware encoder or decoder. As we all know, you can code that kind of stuff. I agree. But in this case, it's pretty neat what these guys and gals at Holtek and DVP have done.

To eliminate reinvention of the rolling device, the Holtek encoder/decoder performs neat little hardware tricks. Instead of the rushed-for-time embedded engineer writing data format code and preambles and worrying about timing, Holtek provides jumper-selectable address logic coupled with data entry logic on one IC.

The transmitter eval board uses the Holtek HT6014. The 6014 encodes 12 bits of info organized as 8 bits of address followed by 4 bits of data. For the blinking-lights types, a transmission in progress LED pin is also provided. The four 6014 data inputs are active low, and when any input goes low, data transmission starts. It gets better.

The 8-bit address and 4-bit data packet are assembled automatically and sent to the input of the TX module on the eval board. This pattern is sent until there are no active-low signals on the data pins.

No preamble is needed. You don't have to write one and you don't have to send one.

The 6014 sorta fakes a preamble by having the receiving decoder read the data twice for accuracy. Yep, you send double data, but it's dependable and, even better, the hardware saves development time.

The receiving decoder reads the eight-bit address twice to ensure it's picking up the correct data. If the addresses match, then the four bits of data are decoded and a data-available pin is enabled. All you, the embedded genius, must do is replace the switches in Figure 5 with logic levels from your embedded masterpiece.

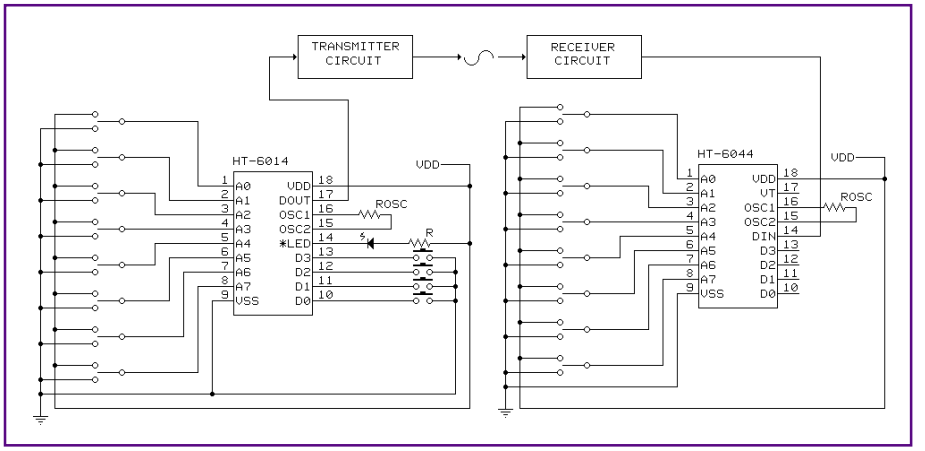


Figure 5—Sure, you could write the code, but in this case why? Just plug in at the switch points.

This type of addressing and data assurance scheme makes it possible to operate many transmitter/receiver pairs like you do an Ethernet network. Oh yeah, timing is a simple matter of placing a resistor across the encoder/decoder OSC pins.

FLOOBYDUST

Floobydust was a term coined by National Semiconductor in its *Audio Handbook*. It has no real meaning. It merely categorizes elements that have no category or that fit most everywhere you wouldn't expect them. So, with that, here's some RF floobydust.

First, even though the DVP modules are unlicensed, you still need FCC approval for your product. The antenna boards are designed to be implanted into your project, and DVP can assist you with getting your product approved.

Second, I wasn't able to get my paws on a 916-MHz radio set. The significance of this is that, at 900 MHz, lots of things can be done with the modulated information that can't be done below 900 MHz.

For instance, continuous transmission is okay at 916 MHz in the U.S., but 433.92 MHz isn't. However, European embedded types can transmit till the cows come home on 433 MHz. Hopefully, I'll be able to show you the 916-MHz radio set next time.

It would be an injustice to squeeze an RF app into the space left, so next time I'll have some RF apps using the DVP products and some wild embedded goodies.

In the meantime, get yourself a copy of the DVP databook and the Holtek remote-control databook. Next month, tune in as I prove that it doesn't have to be complicated to be embedded. APC.EPC

Fred Eady has over 20 years' experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems design and communications. Fred may be reached at fred@edtp.com.

SOURCE
TX418AT, Antenna Boards
DVP, Inc.
(818) 541-9020
Fax: (818) 541-9423
www.dvp.com

Automotive Travel Computer

FEATURE ARTICLE

Robert Priestley

Adding a Performance/Trip Monitor to Your Car

Start your engines! It's time for a high-tech car rally. This computer figures out your gas usage, distance, speed, RPMs, fuel, and more. It even has an alarm that lets you know when your heavy foot is tempting local police.

I originally got involved with this project when some friends asked me to build a device for their rally car to help them navigate during rally events.

They needed an accurate and reliable trip meter because, amazingly enough, their 1966 Ford Falcon XP Rally car didn't have an accurate odometer or speedo! They also mentioned that their fuel-tank gauge wasn't working, so I combined both of these requests and ended up with a trip computer that's suitable for all kinds of vehicles.

Sure, it would have been simpler to install an analog trip meter and fuel gauge, but they figured that installing something high-tech in their car was a surefire way to grab everyone's attention. I was certainly up to the task at the time, and now that the project has grown over the years, I want to give you a look at what I created.

FEATURES

The main features of the trip computer include a custom-designed four-color membrane keypad (see Photo 1) with acoustic feedback of each key press and a specially designed black anodized case, which gives a rigid construction that is shock, dust, and splash proof. The trip-computer dimensions are 145 × 70 × 55 mm.

The computer is programmed to make various calculations:

- distance—distance traveled journey, distance remaining journey, trip meter, and distance to empty based on average fuel consumption
- speed—current speed, average speed, and peak speed
- tachometer—RPM and peak RPM
- fuel—fuel used journey, fuel remaining in tank, liters per 100 km, kilometers per liter, average liters per 100 km, average kilometers per liter, liters per hour (flow rate), total fuel cost, and journey fuel cost
- timer—journey timer, time remaining to complete journey, time remaining at average speed to complete journey, and trip timer

All values can be displayed in metric, U.S., or Imperial formats on the 24 × 2 backlit LCD unit.

The computer also has a standing distance timer accurate to hundredths of a second typically used for quarter-mile sprint timing, and an over-speed alarm, which helps you avoid those speeding fines.

OPERATION

The computer performs its calculations by measuring the fuel flow and distance traveled and by counting the engine revs.

Engine revs are sensed by either making a connection to the engine distributor (for carburetor engines) or fuel injector (for electronically fuel injected [EFI] engines).

Regardless of the function being displayed, the computer continually updates all of these calculations. To show these functions, the computer is initialized at the start of each journey. Pressing the Clear key zeros the distance counters, the amount of fuel used, the internal timers, and the peak speed/tachometer display.

All distances are displayed with a resolution of 0.1 km (or miles), and time calculations are displayed with a resolution down to seconds. Fuel calculations are displayed with a resolution of 0.1 l (or gallon).

The over-speed alarm is an especially useful feature of the trip computer.

When you're driving, the Speed Alarm key can be pressed, which causes the current speed to be stored in memory. If you exceed this speed by 5 km/h (miles), an alarm sounds and the computer automatically displays your current speed until you slow down.

MEASURING FUEL FLOW

Fuel flow can be measured with a flow sensor for carburetor engines or by measuring the injector pulse of an EFI engine.

EFI engines work on the principle that a pressurized fuel line feeds the fuel injectors that are controlled by a computer. The fuel injectors are electromechanical devices that deliver precise amounts of fuel to the engine. By varying the injector-open time, the engine-management computer can precisely control the engine's performance.

With EFI engines and diesels, a flow sensor is typically unsuitable. The pressure of these fuel lines can exceed the specifications of the flow sensor, and fuel injector lines usually have a return feed back to the tank.

Therefore, measuring the actual fuel used by the engine requires two flow sensors (differential measurement) to measure the fuel being used. That is, you need one sensor to measure fuel flow into the engine and other sensors to measure the fuel returned to the tank.

The principle behind the flow sensors used is a coil that gives out sine-wave signals which are induced by permanent magnets. The sensor is an inductive type with a range of 1.5–200 l/h. The frequency of the output signal is proportional to the measured volume.

The flow sensor produces 780 pulses per 0.1 l of gasoline or 1100 pulses per 0.1 l of water. The sensor has a continuous pressure of 10 bar at 22°C and a bursting pressure of over 30 bar. It has a measurement accuracy of ±3%. The sensor is made out of Hostaform C, has a high resistance to chemicals, and weighs 25 g.

The V.1.1 firmware is used with one or two flow sensors, whereas the V.2.0 firmware is used for EFI engines only. However, the hardware is the same. Here, I want to focus on the V.2.0 computer.

Photo 1—*A specially manufactured color membrane keypad gives the car computer a professional look and feel.*

SPEED SENSORS

Many types of sensors can measure the speed of the vehicle by detecting the rotation of a wheel. A proximity sensor or a Hall-effect device are two examples.

Most modern vehicles already have a suitable pulse which is generated by the speedo sensor. For optimum performance, the computer requires 2–4 pulses per wheel rotation.

CALIBRATION

Before you can use it, the trip computer needs to be calibrated. To calibrate the distance, a known distance is driven (usually 1–5 km [or miles]) and the computer counts the number of pulses that are produced by the distance (speedo) sensor. The computer then calculates how far the car travels for one pulse of the sensor.

The V1.1 firmware requires a calibration factor to be entered into the computer for the flow sensor (supplied with the sensor). This number represents the number of pulses produced per 0.1 l of fuel used.

The V2 firmware is calibrated by measuring the total open injector time of a known quantity of fuel, such as a full fuel tank. The computer then calibrates itself over the time it takes to use up a full tank.

THE SOFTWARE

The microcontroller has a special timer input pin that is ideal for measuring pulses because it can capture the exact time at which a transition occurred on its timer input pin. Once the open time of the injector is calculated, it is added to a total.

Every second, the amount of fuel used is calculated based on the measured open-injector time. Listing 1 shows you a sample of the code required to measure a pulse for the 68HC705C8.

A 5-ms internal timer interrupt keeps track of the various timers and the keypad scanning routine. The sample program in Listing 2 generates 200 timer interrupts a second and increments a timer counter in 5-ms increments and a seconds counter.

The speed sensor generates a pulse that triggers the external interrupt pin (EXT) on the controller. When a negative edge is detected, an interrupt is generated which services the distance subroutine. In addition to the timer routine, Listing 2 demonstrates incrementing a simple counter (at RAM location $70) when the IRQ pin is triggered.

These routines are available for downloading from the Oztechnics Web site along with a simple PC program that simulates an injector pulse using the parallel port of a PC. You can use this program to check the trip-computer operation on the bench before installation.

Oztechnics' 68HC05 development system provides many features essential for developing applications for the 'HC05 series of controllers. The development system software, user manual, and trip-computer 68HC05 routines are available via Oztechnics' Web site.

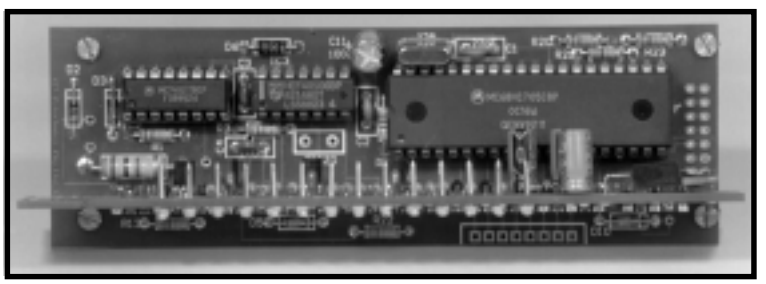


Photo 2—*The two compact boards solder together with the LCD mounted on the back of the processor board.*

The simulator, which forms part of the 'HC05 software package, can run Listings 1 and 2 on a PC.

CIRCUIT DIAGRAM

The heart of the circuit shown in Photo 2 is a Motorola 68HC705C8. Figure 1 shows you a schematic.

The display is operated directly by the controller. You can connect the display backlighting to either the incoming 12 V from the ignition switch (backlighting is therefore on when the ignition is on) or to the vehicle's headlight switch. The latter option means that the backlighting is on when the headlights are on. A jumper on the PCB selects the required option.

The matrix keypad also connects directly to the controller. The three input signals to the computer are buffered by the emitter-follower stages (TR1, TR2, TR3) and are then shaped and inverted by the Schmitt trigger inverters of IC3d, IC3e, and IC3f.

The distance and fuel pulses are interfaced to the controller via JK flip-flops (IC2) to give, in effect, an extra interrupt line. Serial data is available at pin 30 of the controller for interfacing to a PC via a level shifter (the MAX232 is ideal).

Listing 1—*Two 8-bit registers make up the 16-bit input capture register, which latches the value of the free-running counter after the corresponding input-capture edge detector senses a defined transition. The level transition that triggers the counter transfer is defined by the corresponding input edge bit (IEDG) in the TCR. Use the E key to toggle the timer input pin in the simulator.*

```
TCR       EQU   $12             Timer control register
TSR       EQU   $13             Timer Status register
ICAPH     EQU   $14             Input capture register high
ICAPL     EQU   $15             Input capture register low

EDGE1     EQU   $51             RAM: LSB 16-bit negative edge
EDGE2     EQU   $50             RAM: MSB 16-bit negative edge
PULSE1    EQU   $61             RAM: LSB 16-bit measured pulse
PULSE2    EQU   $60             RAM: MSB 16-bit measured pulse

          ORG   $160            Start program from $160 in EPROM
START     LDA   #%10000000
          STA   TCR             Enable input capture interrupt
          CLI                   Enable interrupts
          BRA   *               Loop here/Do nothing

TIMER_INT BRSET 1,TCR,POS_EDGE Determine which edge caused Int
NEG_EDGE  LDA   TSR             TSR must be accessed to clear flag
          LDA   ICAPH           Get MSB of timer counter
          STA   EDGE2           Store MSB of timer counter
          LDA   ICAPL           Get LSB of timer
          STA   EDGE1           Store LSB of timer
          BSET  1,TCR           Set input edge/Positive capture
          RTI                   End timer interrupt
POS_EDGE  LDA   TSR             TSR must be accessed to clear flag
          LDX   ICAPH           Temp store ICAPH/Freeze timer
          LDA   ICAPL           Load input capture register LSB
          SUB   EDGE1           (Time LSB - Pulse_Width LSB)
          STA   PULSE1          Store LSB pulse
          TXA                   Move ICAPH into accumulator
          SBC   EDGE2           (Time MSB - Pulse_Width MSB)
          STA   PULSE2          Store MSB pulse
          BCLR  1,TCR           Set input edge/Negative capture
          RTI                   End-of-interrupt pulse measure
EXT_INT   NOP                   Do nothing
          RTI                   End external interrupt
****                            EPROM vectors
          ORG   $1FF8           Timer interrupt vector
          FDB   TIMER_INT
          ORG   $1FFA           External interrupt vector
          FDB   EXT_INT
          ORG   $1FFE           Reset interrupt vector
          FDB   START
```

The Reset signal is applied to the controller via the +12-V supply from the ignition switch.

The piezo buzzer is operated by the oscillator around IC3a, which is pulsed on by the Beep signal from the controller. A regulated 5-V supply is provided by IC4 and filtered by C3, C7, and C8.

CONSTRUCTION

Constructing the trip computer is quite simple. The circuitry is contained on two small double-sided, plated-through, component masked PCBs, which solder together at right angles.

The LCD mounts on the back of the processor PCB and is held in place with

Listing 2—*Two 8-bit registers make up the 16-bit output compare register (OCR), which indicates when a period of time has elapsed. The OCR contents are compared with the contents of the free-running counter continually, and if a match is found, the corresponding output compare flag bit is set in the timer status register. The corresponding output level bit (OLVL) is also clocked to an output level register (TCMP pin). The I key in the simulator toggles the External IRQ pin.*

```
TCR         EQU     $12         Timer control register
TSR         EQU     $13         Timer status register
OCMPHI      EQU     $16         Output compare register high
OCMPLO      EQU     $17         Output compare register low
MOR         EQU     $1FDF       Mask option register

SECONDS     EQU     $50         Number of elapsed seconds
MILLI_SEC   EQU     $51         Number of milliseconds

EXT         EQU     $70         Counter for external IRQ

            ORG     $160        Start program from $160 in EPROM

START       LDA     #%01000000
            STA     TCR         Enable output compare register IRQ
            LDA     #$00
            STA     MOR         Enable neg trans on IRQ pin
            CLR     SECONDS     Reset seconds counter
            CLR     MILLI_SEC   Reset 10-ms counter
            CLI                 Enable interrupts
            BRA     *           Loop here/Do nothing
* A 4-MHz crystal produces 2-µs internal tick, 2500 ticks = 5 ms
* 200 x 5 ms = 1 second count, 2500 = $09c4
TIMER_IRQ   LDA     TSR         Clear OCMP flag by accessing TSR
            LDA     OCMPLO        and OCMPLO reg
            ADD     #$c4        Lower half hex equivalent of 2500
            TAX                 Temp store lower half of new OCMP
            LDA     #$09        Upper half hex equivalent of 2500
            ADC     OCMPHI
            STA     OCMPHI      Update upper half of OCMP reg
            STX     OCMPLO      Update lower half of OCMP reg
            INC     MILLI_SEC   Add 1 to millisecond counter
            LDA     #200        200 * 5 ms = 1 s
            CMP     MILLI_SEC   Test if reached 200 yet?
            BNE     TIMER1      Branch if not equal to 1 s
            CLR     MILLI_SEC   Clear 5-ms counter
            INC     SECONDS     Increment seconds counter
TIMER1      BRSET   0,TCR,TOGGLE Toggle OCMP pin
            BSET    0,TCR       Set TCMP pin to 1 on next compare
            RTI                 End-of-timer interrupt
TOGGLE      BCLR    0,TCR       Set TCMP pin to 0 on next compare
            RTI                 End-of-timer interrupt
EXT_IRQ     INC     EXT         Increment RAM
            RTI                 End-of-timer interrupt
****                            Define EPROM vectors
            ORG     $1FF8
            FDB     TIMER_IRQ   Define timer IRQ vector
            ORG     $1FFA
            FDB     EXT_IRQ     Define external IRQ vector
            ORG     $1FFE
            FDB     START       Define reset vector
```

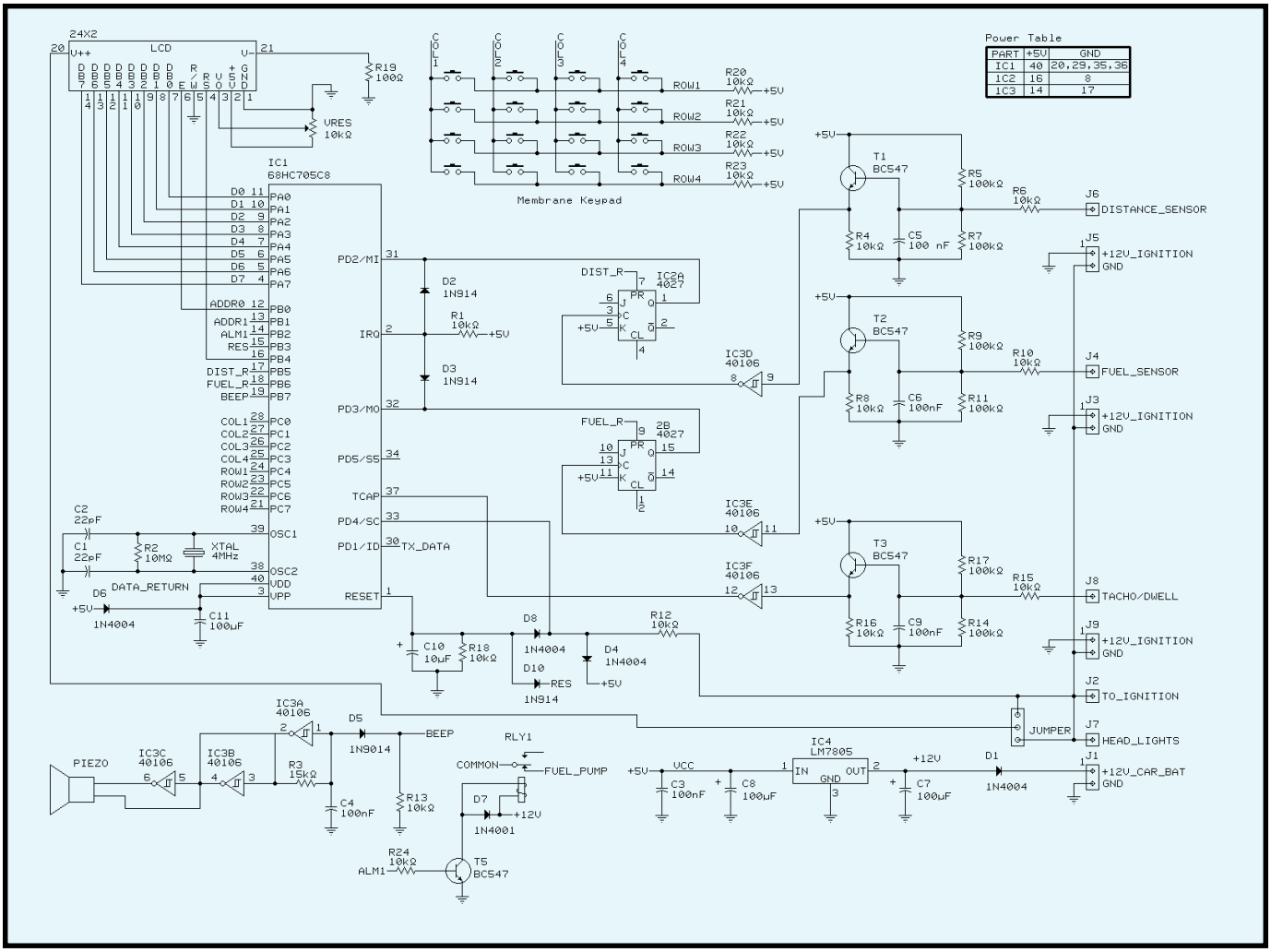


Figure 1—*The schematic for V.1.1 of the car computer is almost identical to V.2.0. The same PCB was used for both versions of firmware.*

four metal thread screws and nuts. The display is about 3 mm from the PCB.

The membrane keypad fixes to the clear Perspex front panel of the case using its self-adhesive back, and it connects to the PCB via a special membrane ZIF connector. The PCB assembly slides into the aluminum case, and the front and rear panels snap into place.

Use PCB pins to connect the wiring harness to the computer. Heat-shrink sleeving over each soldered connection increases reliability.

INSTALLATION

Installation is also straightforward. You need to locate permanent 12 V, ignition 12 V, Ground, and an optional headlight connection (for the backlit LCD). The back of the digital clock is a good source for these signals.

Once power is established, run a wire to an injector and connect the computer to the switched (grounded) side of the injector or install the fuel flow sensor in the fuel line. Make a connection to the electronic speedo.

You'll need 2–4 pulses per wheel revolution. If more are produced, either install a separate speedo sensor or divide the pulses down using a divider chip arrangement.

You need a digital pulse (0–5 V) to run the computer inputs. Inductive pickups won't work because they need to be amplified and signal conditioned.

APPLICATIONS

Of course, this system doesn't have to be used as a trip computer. Any application requiring a counter or quantity of liquid to be measured can be accommodated by this kind of device.

Robert Priestley is an electronics engineer specializing in embedded systems for Oztechnics, a producer of embedded system products and development tools for Motorola 68HC05 and 68HC11 microcontrollers. You may reach him at robert@oztechnics.com.au.

SOFTWARE

Complete source code for this article is available via the Circuit Cellar Web site.

SOURCE

V1.1 Trip Computer (requires flow sensor) $110
V2.0 Trip Computer (EFI) $110
Flow Sensor $75
Speed Sensor $25

Oztechnics Pty. Ltd.
P.O. Box 38
Illawong NSW 2234
Australia
+61-2-9541 0310
Fax: +61-2-9541 0734
www.oztechnics.com.au

DEPARTMENTS

Build a Serial Port PROM Programmer

MICRO SERIES

Stuart Ball

Two Adapter Modules

Part 2 of 2

After Stuart checks out the hardware setup for the low-cost serial port programmer he put together last month, he gives us an up-close look at how the software and programming algorithms make everything tick.

Last month, I gave you a look at the programmer hardware, construction, and operation for building a serial port PROM programmer. Now, let's check out the hardware and see how the programmer works, how it interfaces to the programming modules, and how the software handles the tasks required.

PROGRAMMER CHECKOUT

My first task is to verify that all the programmer functions are working. I start by applying power to the programmer and checking the +24 and +5 V to be sure they're correct.

Pin 2 of U10 (MAX232) should be –8 to –10 V, and pin 6 of U10 (MAX232) should be –8 to 10 V. I then verify that the bicolor LED is green.

Next, I connect the programmer to the PC and start the communication program on the PC. (For a description of setting up the communication program, see the section on "Using the Programmer" in Part 1.) I then install the 27*xx* EPROM programming module.

When I cycle power on the programmer, the command menu should come up on the PC screen. I select a 2764 by entering #T 1 and turn on the V_{PP} and V_{CC} voltages by entering #XV.

The LEDs on the programming module should be on. The voltage at pin 1 of the ZIF socket should be ~12.5 V, and the voltage at pin 28 of the ZIF socket should be about 6 V.

Signal	Usage
EPROM module	
A0–A14 (82C55 ports A and B)	A0–A14 (A14 is *PGM on 2764 and 27218)
D0–D7 (82C55 port C)	D0–D14
*OE (control register bit 2)	*OE
*CE (control register bit 3)	*CE
Control register bits 4-7	Unused
PIC module	
A0 (82C55 port A, bit 0)	Clock (RB6)
D0 (82C55 Port C, bit 0)	Bidirectional Data (RB7)

Table 1—*This table maps out the 8255 and control register bits on the two programming modules. The PIC module only needs two bits from the 8255 and does not use the control register at all.*

I turn off the programming voltages by entering #XR. My next task is to program an EPROM to make sure the programmer works correctly.

The 80C188 is somewhat picky about crystals. With some crystals, I've found that the oscillator doesn't always start. If you turn on the programmer and sometimes the LED stays off and the startup menu isn't sent, the oscillator may not be starting.

To fix this problem, lower the values of capacitors C3 and C4. Some crystals start best with C3 and C4 removed. Or, try installing a 1-MΩ resistor in parallel with the crystal. As a last resort, try a different brand of crystal.

INTELLIGENT ALGORITHMS

The programmer supports intelligent programming algorithms. In the early days, when a 2K × 8 2716 was a large EPROM, programming took a constant 50 ms per location.

When there are 2048 locations, programming this way takes only 100 s. But when there are 32,768 locations, programming by this method takes 30 min.

To reduce this programming time, the PROM manufacturers developed adaptive algorithms. First, apply programming signals to the device for a specified time (1 ms, 0.1 ms, etc.) and read the device to see if data is correct. Repeat these two steps until the data is correct. Once the data verifies, apply an overprogram pulse (typically 2×).

Consider an example where an EPROM uses a pulse width of 1 ms and a 2× overprogram pulse. In a typical programming sequence, you might first attempt to program four times, and the device programs on the fourth attempt. Then, you apply an overprogram pulse of 8 ms (2 × 4 × 1 ms).

The total programming time for this location in the device is the original 4-ms plus the 8-ms overprogram pulse time, for a total of 12 ms.

These algorithms also provide for a maximum number of retries before the location is considered bad. If this hypothetical EPROM allows up to 20 retries, then programming one location can take anywhere from 3 to 60 ms. Of course, most locations in most EPROMs take less than the maximum.

Different manufacturers use different voltages and program pulse widths. The 27*xx* devices can be broken down into broad categories—12-V programming voltage versus 21-V, and 1-ms program pulse width versus 0.1-ms.

The programmer simplifies things by using essentially the same algorithm for all devices. So, there are just a few generic devices to select:

2764, 12 V, 1 ms
2764, 12 V, 0.1 ms
2764, 21 V, 1 ms

There are similar generic selections for the 27128 and 27256. A commercial programmer may have a dozen different (but very similar) algorithms for programming PROMs from different manufacturers.

PROGRAMMING MODULES

As I mentioned in Part 1, the programmer saves cost by using adapter modules for different devices. There is one module for 27*xx* EPROMs and one for PIC devices.

Each module connects the programming signals from the 50-pin connector to the ZIF socket holding the device to be programmed. Table 1 defines the signals for the two modules.

To program a location in an EPROM, the V_{CC} and V_{PP} pins are raised to the values needed for programming—typically around 6 V and 12.75 V, respectively. The V_{CC} voltage for programming is usually higher than for normal operation.

Figure 1 shows the timing to program and verify a location in a 2764/27128 and a 27256. For the 2764/27128, after the voltages are stable, the address is placed on the address pins, and the data is placed on the data pins. In the programmer, this amounts to writing the address and data values to the appropriate 82C55 ports.

The *CE and *PGM signals are then driven low. This condition is maintained for 1 ms, or 100 µs, depending on the algorithm you're using. Then, the *PGM and *CE pins are driven high.

To verify the location just programmed, the data port of the 82C55 (port C) is switched to input mode. The *CE and *OE pins are driven low, while the *PGM pin is left high.

The EPROM reads the selected location and places the data on the data lines, where the programmer reads it by reading the 82C55 port. If the data does not verify, the programmer must program and verify the location again.

The timing for the 27256 EPROM is identical to the 2764/27128, except

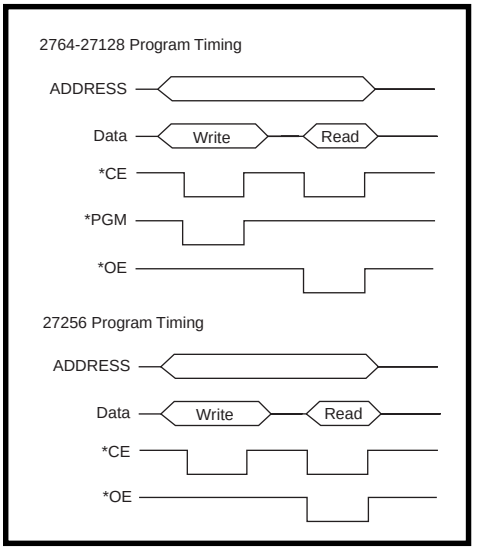


Figure 1—*In programming the 2764/27128 and 27256 EPROMs, the timing is identical, except for the *PGM signal, which is address input A14 on the 27256.*

that there is no *PGM signal. On the 27256, the pin that had the *PGM signal on it carries address bit A14.

The EPROM programs when *CE is low and reads when *CE and *OE are both low. To read a location, *OE must be driven low first so the EPROM doesn't see the *CE-only condition and try to program the location with the data lines floating.

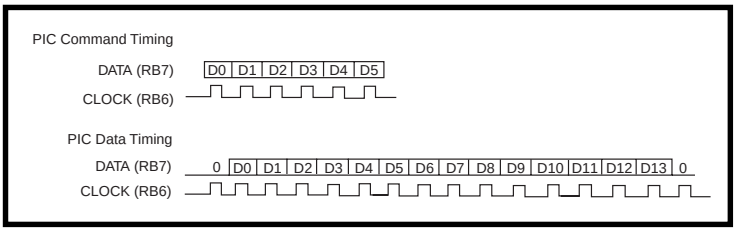


Figure 2—*The PIC devices are programmed and verified with commands that are transmitted serially. Each command is six bits long, and some commands are followed by 14 bits of data.*

PIC PROGRAMMING

The PIC devices use a different method of programming, which uses only five pins—Power, Ground, MCLR (which receives the V_{pp} voltage), RB6, and RB7. The PIC devices use a serial scheme where a six-bit command is clocked into the device. Some commands are followed by 14 bits of data.

Figure 2 shows the scheme for sending data to a PIC device. Pin RB6 functions as a clock, and pin RB7 serves as a bidirectional data pin.

Command	Code
Load Configuration	000000
Load Data	000010
Read Data	000100
Increment Address	000110
Begin Programming	001000
End Programming	001110

Table 2—*These PIC programming commands are loaded serially using two pins on the PIC device.*

The EPROM devices support six commands (listed in Table 2). The 16C84, an EEPROM device, has additional commands for bulk erasure and programming the user memory, but these are not used by the programmer.

The `Load Configuration` and `Load Data` commands are followed by data, and `Read Data` is followed by data read from the device. The data is 14 bits, but 16 bits are clocked in and out of the device, since the data is preceded and followed by a zero bit.

The programming sequence for the PIC devices goes like this:

1. Set V_{pp} (MCLR) and V_{CC}
2. Send `Load Data`, followed by the data to be programmed
3. Send `Begin Programming`
4. Wait 100 µs
5. Send `End Programming`
6. Send `Read Data`
7. Check the returned data against the original programming data

Repeat steps 2–7 until the data verifies.

Like the 27*xx* EPROM devices, the PIC EPROM devices need an overprogram pulse after the data verifies. After the overprogram pulse, the address is incremented using `Increment Address` and the next location is programmed.

The PIC devices don't have a way to randomly select a specific address, other than to reset the internal address pointer to zero. So, these devices must be programmed sequentially from location 0 to the top of program memory.

Because each PIC location is 14 bits wide, two bytes in the buffer are required to hold each location. A PIC program that is 1 K words long takes 2-KB locations in the buffer. The programmer expects data to be loaded least significant bit first, which is the sequence generated by the Microchip assembler.

CONFIGURATION MEMORY

The PIC devices have an additional feature that the EPROMs do not have—configuration memory. The configuration memory programs the type of oscillator that the PIC expects to use, the power-up reset characteristics, and other features. The programmer will program the configuration locations.

After the entire PIC program memory is verified, the programmer programs the value in buffer locations 400E and 400F to the configuration location (2007) in the PIC device.

Again, the address in the buffer is 2× the device address because the device counts words and the programmer buffer counts bytes. Like normal program data, the programmer expects the least significant bit of the configuration word to be first (in 400E).

To get the configuration memory programmed, you must include the configuration information in the hex file. The Microchip assembler provides for this with the `.CONFIG` directive.

If you're using an assembler that doesn't support the configuration memory, you can manually build a hex file that contains your configuration data and send it to the programmer prior to programming the device.

The PIC devices also provide locations that can be defined by the user for tasks like adding a serial number to each device. The programmer doesn't support this (although the necessary code changes would be fairly minimal).

PIC16C84

The PIC16C84 is an EEPROM device that is similar to other 18-pin PIC devices and that uses a similar command set. The primary difference in how the '16C84 is handled by the programmer is that the '16C84 performs an internal erase of the EEPROM location prior to programming, so no blank check is needed.

The '16C84 internally times the program cycle, so only one programming pulse is required, and no overprogram pulse is needed. Each location in the '16C84 takes a little over 10 ms to program.

Pin Number 18-pin devices	Pin Number 28-pin devices	Name	Function
13	28	RB7	Serial data I/O
12	27	RB6	Serial clock
4	1	MCLR	V_{PP} (prog. voltage)
5	8, 19	V_{SS}	Ground
14	20	V_{CC}	+5 V

Table 3—*With the PIC programming connector, the 18- and 28-pin devices are programmed the same way but use different pins. The 12-pin header on the module selects the type of device.*

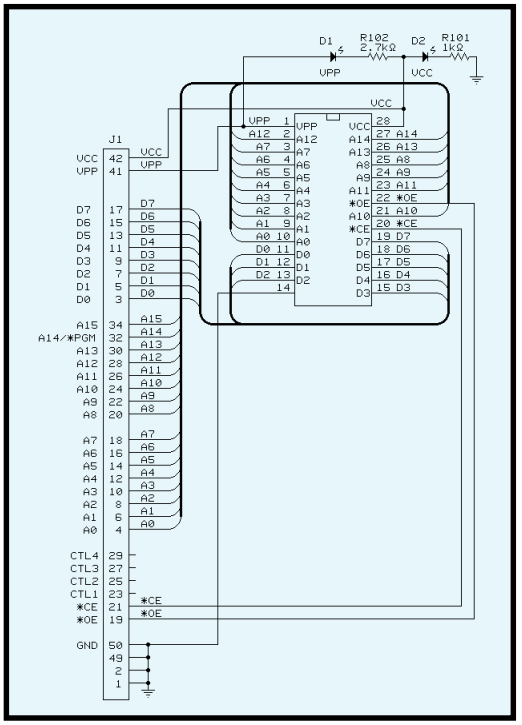


Figure 3—*The 50-pin connector mounts on the back of the EPROM programming module and plugs into the mating connector on the programmer.*

18- AND 28-PIN DEVICES

As I mentioned last month, the PIC module has a 12 × 2 pin header that connects the programming signals to the proper pins for 18- or 28-pin devices. A socket is installed on the header, on one side or the other, to make the connections. The PCB set includes a small circuit board for this purpose.

If you decide not to use the PCB to wire the circuit, you need to wire a socket header for this purpose. The header is wired as follows:

1 to 6
2 to 8
3 to 9
4 to 10
5 to 11 and 12

Pin 7 is a key to prevent reversal of the socket on the header. To prevent voltage from being applied to the wrong pins, the software checks to be sure the socket is installed in the correct position on the header before programming. Table 3 shows which pins serve which functions on the PIC devices.

Figure 3 shows the schematic of the EPROM module, and Figure 4 shows the schematic of the PIC module.

FILE FORMATS

The programmer accepts two formats—Intel hex and raw hex. The Intel hex format consists of a line like:

```
:nnaaaattdddddd…dddcc
```

where *nn* is a data byte count (the number of bytes in the line), *aaaa* is a 16 bit address (the location in memory where the first data byte goes), *tt* is a record type, *dd…dd* is the data in hex, and *cc* is a one-byte checksum.

The programmer can accept raw hex files, which consist of two ASCII-hex characters for each byte to be downloaded. The programmer always places raw hex data at location 0000 in memory, unless #M is used to force a different offset.

The programmer accepts raw hex data with or without spaces or commas for delimiters. Therefore, these three lines are the same to the programmer:

```
AA 55 12 34 56 78
AA5512345678
AA,55,12,34,56,78
```

The programmer also ignores carriage returns and linefeeds, so the data can be formatted for better readability. In short, almost any combination of raw ASCII-hex data is accepted in this format.

The one drawback to the raw hex format is the lack of an end-of-file indicator. The Intel format has an EOF record type to indicate when all the data has been transferred, but there is no such universal indicator for the raw hex files.

Some conversion programs generate an ETX at the end of a raw hex file, and the programmer can recognize this. In other cases, you have to terminate a raw hex download when the computer has sent all the data by entering the #Q command.

SOFTWARE

The programmer software can be broken down into five general components. First, the initialization code sets up the peripheral components, the 80C188 internal chip selects, and the data transfer rate.

Next, an interrupt service routine (ISR) handles UART I/O. Output data is buffered in a software FIFO and transmitted a byte at a time as the UART transmitter becomes available. Similarly, input data is read and buffered in a software FIFO for processing by non-ISR code.

A background loop performs one device operation (e.g., program, verify, blank check a byte) per pass. The background loop also checks for and processes data received from the PC.

Device support routines handle device operations (e.g., blank check, verify, program, setup) as well as generic operations like setting and clearing the control bits, writing a new address to the device, and so forth. Finally, device-specific routines perform the program, blank check, and verify operations.

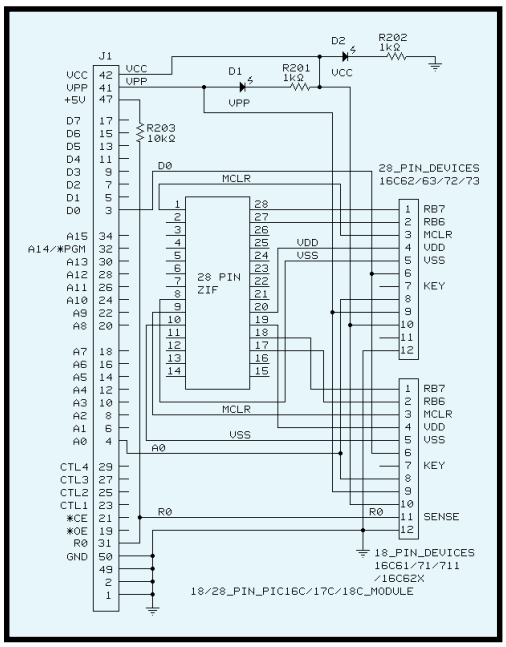


Figure 4—*The 50-pin connector mounts on the back of the PIC programming module and the mating connector on the programmer.*

Each device is described in a 16-word table. The table contains information like the device size and word width, a blank value, the device-specific routines to use, the DAC values to set correct voltages, and whether to perform blank check prior to programming.

The software can support both 8- and 16-bit wide devices. The 27*xx* EPROMs, for example, are all 8 bits wide. The PIC processors are 16 (really 14) bit devices. (The PIC itself is an 8-bit processor, but the programmer is concerned with instructions that go into the PROM, which are 14 bits wide).

Figure 5 shows the hierarchy of the main programmer routines. When I decided to turn this project into an article, I needed to break the software up in such a way that readers could assemble it with whatever assembler they have. Not all assemblers support modular assembly, and I didn't want to tie the source code to just one assembler.

As a result, the software source code is broken into text files that are assembled into one large source file by running a batch file (which is called, appropriately, COPYIT). With minor modifications, this source file should be compatible with just about any assembler.

The only catch is that the constants and variables are defined in separate files (due to a limitation in my assembler). If you need to have them included in the source file, you'll have to make some edits so they match your assembler format.

If your assembler supports modular assembly, you can modify the text files with the appropriate header information for your assembler and make each text file a separate module.

EXPANDING THE PROGRAMMER

The programmer is designed for expansion to include new device types. To do this, you have to design and build a new adapter module and write device-specific routines for the new device.

You then include the new device in the device tables, include it in the vector tables for setup, program, blank check, and verify, and include it in the message that's printed with the #L command. More detailed instructions are included in a text file with the source code.

The programmer includes a few features to simplify development of new modules. You'll remember that last month I mentioned the #XB and #XM commands. #XB lets you look at any 64-byte area of the 64-KB buffer, and #XM lets you look at any 64-byte area of the program RAM, so you can look at the variables and software FIFOs.

One specific software area—a rotating trace buffer—deserves special mention. Located from 1600 to 16FF, the buffer can give you a sequential history of what the software was doing.

You can write one-byte values to the buffer by calling a subroutine named DIAGNOSTIC. Your trace value is stored in the next buffer location, followed by a byte of FF.

You could also develop a Windows-based GUI to control the programmer. This type of approach could probably let you select devices by name, eliminating the #L command, as well as doing additional error checking (e.g., preventing you from specifying an operation size that is bigger than the device).

The programmer was intended to accommodate such expansions, and the only modifications you'd need to make to the programmer code would be to replace the return messages with numeric codes.

For this project, I needed 27*xx* EPROMs and PIC devices. You may have other devices you want to program.

Send me E-mail describing what PROMs or microcontrollers (no PLDs, please) that you would like to see the programmer support. If there is a consensus, I'll look at adapting the programmer to the most popular one or two choices.

That's it. Go forth and program.

Stuart Ball works at Organon Teknika, a manufacturer of medical instruments. He has been a design engineer for 18 years, working on projects as diverse as GPS and single-chip microcontroller designs. He has written two books on embedded-system design, both available from Butterworth-Heinemann (www.bh.com). You may reach Stuart at sball85964@aol.com.

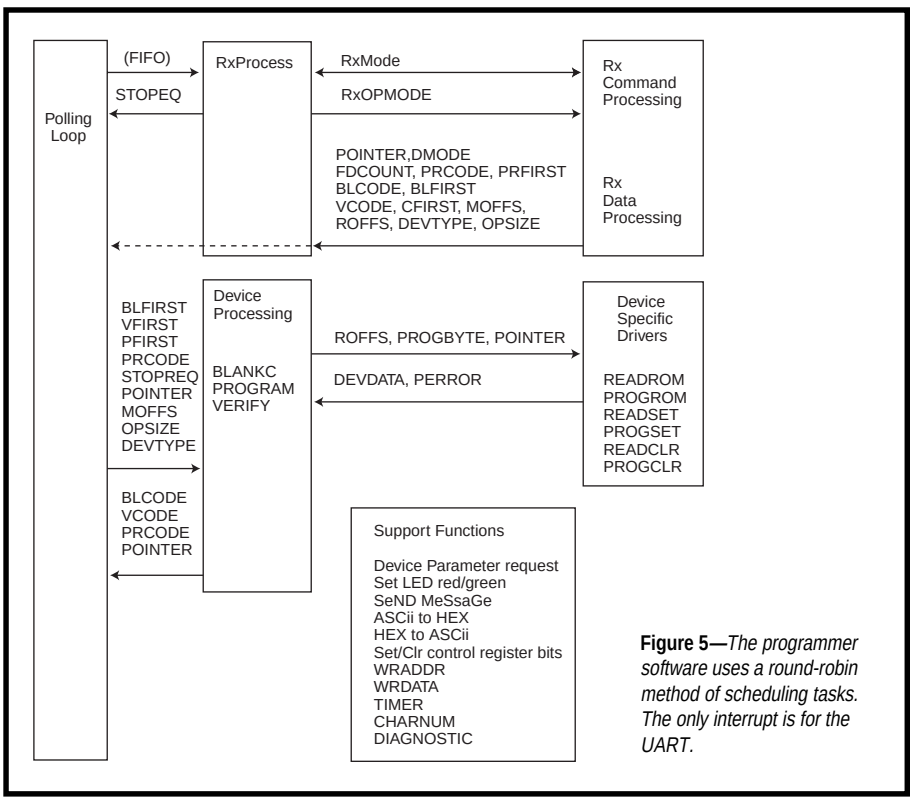


Figure 5—*The programmer software uses a round-robin method of scheduling tasks. The only interrupt is for the UART.*

SOFTWARE

Complete source code for this article is available via the Circuit Cellar Web site.

The Next Step in EL Driver Technology

FROM THE BENCH

Jeff Bachiochi

An alien? No, it's just Jeff wearing his new electroluminescent nametag. After debriefing us on the fundamentals of EL panels, he tells us how we too can get that strange green glow.

Even though I've converted to CDs, I just can't toss out my old vinyl LPs. Some have more than just sentimental value. Some of my favorites aren't even music.

Have you ever heard any of Bill Cosby's comedy albums? Albums like *To Russell, My Brother Whom I Slept With* keep me smiling even today.

Of course, back then it was different. I had to have a nightlight because of Bill and his storytelling. You see, a nightlight was the one thing that protected you from the snakes that slither around under your bed at night.

Bill told how if you put one foot on the floor after lights out you were done for. Daring the snakes to just take a little lick was foolishness. Thank heavens for nightlights! They had the power to keep those snakes under the bed where they belong.

I didn't think of our family as high tech at the time, but we didn't have a bulb nightlight—we had a mysterious panel that gave off the kind of light you'd expect from a full green-cheese moon. And it emitted no heat, unlike any other light source I'd ever seen. For years, I wondered what it was.

In my early teens, I did a lot of camping with the Boy Scouts. One night while camping far from civilization, I had my first encounter with an alien life form. What else could it have been?

It was glowing in the woods. Nothing I knew of on this planet could glow without batteries. I touched it, expecting to be transformed to another dimension. Nothing happened. It wasn't even warm. Flashback of snakes—yikes!

Nightlights, hmm…. Light and no heat. I needed to know more. But, the following morning, it was gone.

Well, as with many other mysteries of the world, Mother Nature had a hand in this one. The chemical process of a decaying tree stump had produced a phosphor that glowed continually.

During the day, the glow was undetectable, but at night, the phosphorescent glow was significant. Today, you find phosphorescent sticks and necklaces in your local store. The magic has gone commercial.

EL LAMPS

Electroluminescent (EL) lamps are everywhere. They are used to backlight devices ranging from IR remotes to lot of the audio and video equipment we own—pagers, cell phones, GPS receivers, and, yes, even some wristwatches.

It's no surprise that these light sources are being used more frequently. They're low in power, thin, flexible, and lightweight. They produce negligible heat and can be easily manufactured in custom shapes. You'd think, with all that going for it, EL lamps would be used even more than they are already.

There are two major drawbacks to EL backlights. First, EL panels experi-

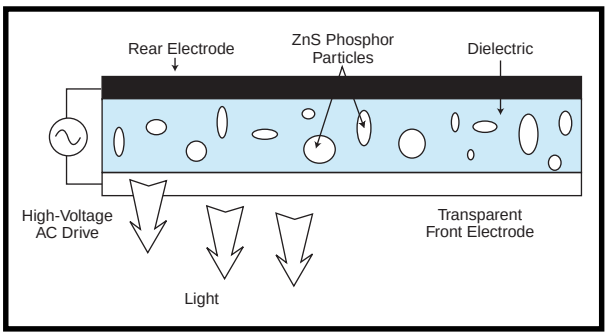


Figure 1—*High-voltage AC is needed to drive photons of light from phosphorous particles suspended between electrodes.*

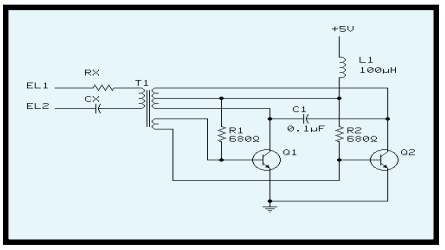


Figure 2—*As an integral part of this oscillator, the primary of T1 also boosts the voltage via its secondary.*

ence a typical brightness half-life due to aging. This half-life (i.e., the active life until the brightness diminishes to half its initial output) can be as little as a few thousand hours.

At 24 hours a day, that's roughly 9000 hours in a year. So, an EL panel running day and night at maximum brightness may deteriorate to half its brightness in only a few months.

Second, the EL panel often requires 100–200 VAC to operate. For portable battery-powered devices, this has been the EL panel's ball and chain.

Before we look at the support circuitry, let's see how EL panels are made.

THE EL SANDWICH

Mechanically, the EL panel looks like a large capacitor—typically, a few nanofarads per square inch.

As you see in Figure 1, two inner layers—a dielectric layer and a phosphor layer—are sandwiched between two conductive surfaces. The dielectric layer prevents the two conductive layers from shorting out while being subjected to high voltages.

When a high-voltage potential is placed across the two conductive layers, an electric field is generated. The phosphor particles are encouraged to give off photons (i.e., luminesce) as the electric field alternates.

During a voltage rise, the phosphor's electrons are excited from the valence band into the conduction band. Conversely, when the voltage decreases, these electrons return from the conduction band to the valence band.

Electrons moving to the lower energy bands can give off photons. If one conductive surface is transparent, the light escapes and produces the familiar glow.

ZnS phosphor, the most common compound used in the dielectric layer, is responsible for the blue-green glow of most EL panels. The spectral emissions of the phosphor can be altered by doping ZnS, CaS, or SrS with transition metals or nonmetals like magnesium or europium.

Ongoing EL development hopes to rival color LCDs someday. But, much of that is in the hands of chemists.

The two most significant variables used to generate light output from EL panels are voltage and frequency. Panel brightness is directly proportional to voltage and frequency.

Although each panel has maximum and minimum values for voltage and frequency, typical ranges are from 100 to 200 VAC and from 60 to 1000 Hz. An EL panel used as a nightlight at 120 VAC at 60 Hz may have a typical luminance output of a few candelas.

HIGH-VOLTAGE DRIVE

Unlike the couple of volts DC that are required for LED backlighting, their high-voltage requirements make EL panels far more difficult to use. In the past, transformer inverters were needed to translate low-voltage DC into useful high-voltage AC. Figure 2 shows a typical DC to unregulated high-voltage AC converter.

In this push-pull configuration, the transformer is not only a step-up device but also an integral part of the primary's oscillator. Normally, these devices were purchased for a particular-sized EL panel because there were no user-selectable components for adjusting the output.

Recent innovations have reduced the circuitry, so that a single IC can now act as a voltage step-up converter and a commutating bridge driver. Figures 3a and 3b show the IMP803 and SP4428 high-voltage lamp drivers, respectively.

The IMP803 step-up converter uses an internal high-frequency oscillator to gate a MOSFET switch (50–90 kHz). An external coil, L1, and diode, D1, combine with the MOSFET switch to step up Vin (as little as 3 V) to as much as 90 V across capacitor Cs (Vcs).

The coil's stored energy is released through the diode into the capacitor each time the MOSFET switch opens. The switching frequency is controlled by external resistor R_{SW}, and the efficiency varies with the selection of L1 and R_{SW}.

The second section of the circuit applies the high-voltage potential to a full H-bridge. The EL lamp is placed across the bridge. A second oscillator, tuned by external resistor Rel and running at 300–400 Hz, alternately gates diagonal sections of the H-bridge, producing an alternating potential on the EL panel twice that of the high-voltage input.

IMP's high-voltage EL lamp-driver IC comes with and without a high-voltage regulation section. When Vin is a battery whose output drops with age, Vcs (and the lamp's brightness) can remain constant if the device has regulation circuitry. When the lamp is driven from a Vin that remains constant, using the lamp driver without regulation has an advantage.

Since the EL lamp's capacitance goes up as the lamp approaches its half-life, an unregulated circuit self-compensates by creating a higher Vcs. Design considerations should be based on the type of power used and the duty cycle of the lamp (how often it needs to be on).

The Sipex SP4428 uses slightly different circuitry. Only a single oscillator is used, and its frequency is set by an external capacitor.

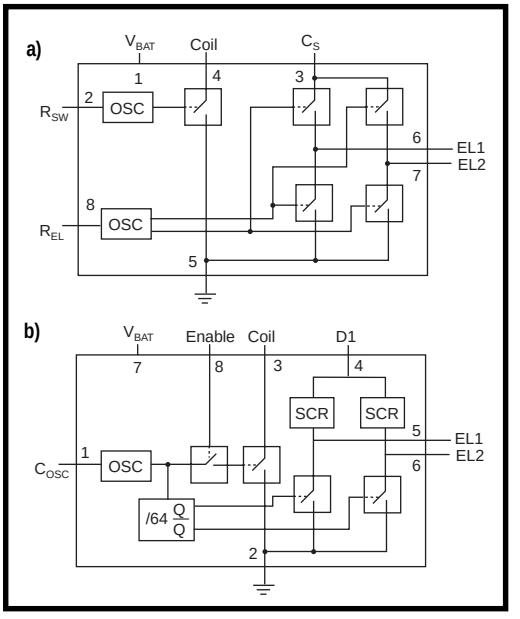


Figure 3a—*The IMP803 has separate oscillators for more user control.* ***b—****The SP4428 uses a single oscillator to provide both high voltage DC and switching for the EL panel.*

Energy developed in an external coil by removing it from ground at the 20-kHz (nominal frequency) rate is directed toward the second stage by a diode similar to the IMP device. But, instead of charging a separate external capacitor as in the IMP803, the SP4428 uses the EL lamp as the capacitor being charged.

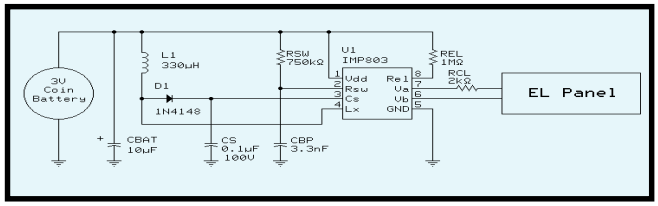


Figure 4—*My EL circuitry needed a bit of capacitance from RSW to Ground. This situation may be due to the layout currents.*

In the second stage, the H-bridge is switched by a second clock. Six flip-flops generate a second clock (~300 Hz) from the primary oscillator (frequency/64). The SP4428 requires more than 1.1 V to produce a minimum of 120 VAC across a 5-nF EL panel (roughly 1.5 $in.^2$).

NOT JUST LCD BACKLIGHTING

More and more uses for EL panels are arriving on the scene. I'm sure you've seen joggers or other people out at night wearing apparel fitted with EL panels for high visibility. Portable emergency and hazardous warning signs are gaining popularity, too, thanks to some of the new driver technologies.

At one point, I'd planned on making an LCD nametag. Now, with an EL panel and some circuitry, I can do the job more easily and less expensively.

I started with the basic circuit (see Figure 4), but I needed to add a couple of capacitors to get the best output. Using an IMP803, I found the HV section to work as advertised. Selecting a coil with low Rs gave higher Cs voltage.

The peak current draw from the 3-V lithium coin cell was above what the cell could sustain. Major voltage sags occurred at maximum currents, so I added a capacitor across Vbat, which also improved the high voltage on Cs.

On the H-bridge side, I saw spurious output that was insufficient to light the EL lamp. A note suggests adding a Cbp if the backlight flickers. Flicker wasn't the problem, but a capacitor here cured whatever noise was preventing the H-bridge from oscillating correctly.

These few parts fit easily on the back of a 1″ × 3″ PCB. I chose the size to fit across the back of an LCD bezel.

Because the parts were surface mounted to the PCB, I was able to use the other side of the PCB for mounting the EL panel with double-sided tape without worrying about protruding leads puncturing the EL panel. Extra holes in the PCB enable the bezel to sandwich the panel tightly to the PCB.

For the finishing touch, small adhesive or rub-on lettering lets me create any message I deem necessary.

So, if you happen to see a strange green glow at the next electronics show you attend, don't be alarmed. It won't be an alien…. It's just me.

Jeff Bachiochi (pronounced"BAH-key-AH-key") is an electrical engineer on Circuit Cellar INK*'s engineering staff. His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circuitcellar.com.*

SOURCES

IMP803
IMP, Inc.
(408) 432-9100
Fax: (408) 434-0335
www.impweb.com

SP4428
Sipex Corp.
(978) 667-8700
Fax: (978) 667-8310
www.sipex.com

EL lamps
Durel Corp.
(602) 917-6000
Fax: (602) 917-6049
www.durel.com

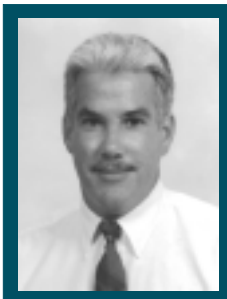
Lumitek International, Inc.
(800) 992-5149
(301) 831-1001
Fax: (301) 831-8221
www.us.net/quantex

MetroMark, Inc.
(800) 680-5556
(612) 935-8844
Fax: (612) 935-5718
www.metromark.com

SILICON UPDATE

Tom Cantrell

Sensory Overload



Sensors Expo always offers the latest in sensor technology, and Tom's here to report. Whether it's position sensing, voice recognition, or strain gauge devices, get a sense of what's best for your application.

I've always enjoyed the Sensors Expo. After all, sensors are at the heart of most embedded control apps, whether it's a simple thermostat or a highfalutin vision system.

I've been there before ("In the Realm of the Sensors," *INK* 49). However, the show is on a 10-city cycle (keep an eye out since it includes nontraditional venues like Cleveland, Baltimore, and Philadelphia), so it's been a while.

Anyway, I was a bit surprised at the sea of booths I encountered. Sure seems like sensors are getting to be a big deal. More power (and ground) to 'em, I say.

PCs WAKE UP

Even PCs are getting into the act with temp sensors, fan tachs, and intrusion and motion detectors. If chips like the National LM80, shown in Figure 1, are any indication, anybody thinking of lightfooting a laptop or lifting some SIMMs can expect the MIS police to come a-knockin'.

The chip starts with an eight-channel eight-bit ADC. Seven channels are brought out while one is devoted to an on-chip temp sensor.

Unlike a typical 0–5-V ADC, the LM80 A/D input range is 0–2.56 V (i.e., 10 mV/bit). This happens to be a perfect match with popular three-pin linear temp sensors like the LM50.

You can use the LM80 to monitor the temp of various components scattered around inside the box. There's also a digital input (BTI, which stands for Board Temperature In) that is suitable for connecting to thermostat temp chips like the LM56 or LM75.

The ADC is also intended to monitor the half dozen or so system voltages (e.g., 2.5, 3.3, ±5, and ±12 V) at work on the latest motherboard. It's a

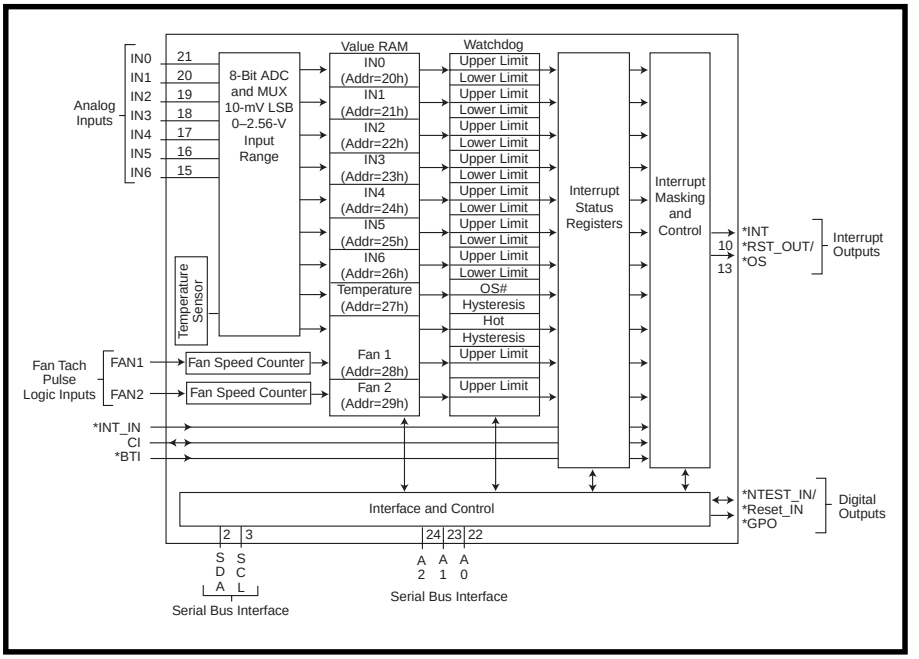


Figure 1—*Whether it's the various power supplies, fans, external and internal temp sensors, or even the chassis, the LM80, like Big Brother, is always watching.*

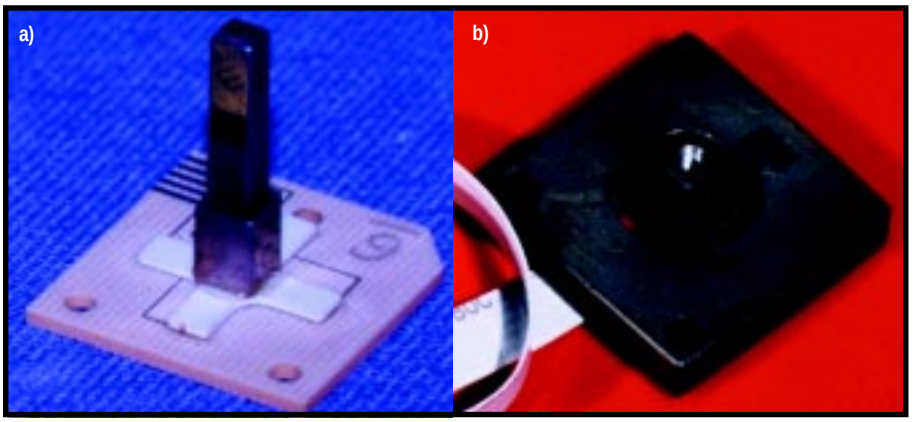


Photo 1—*Force joystick technology finds its way from laptops to the factory floor. The Bockam Engineering concept, screening strain gauges onto a substrate, is the same whether it's ceramic (**a**) or solid steel (**b**).*

simple matter of using resistor dividers to attenuate and center each voltage in the A/D range.

Whether it's the power-supply levels, external temperature, or on-chip temperature, the LM80's role is the same. It sits in a loop, scanning the inputs and comparing them against host-programmed (via I^2C) limits, raising an interrupt on excursion. The on-chip temperature sensor gets a little extra attention—a 12-bit conversion option, programmable hysteresis, and more versatile alarm modes.

If things are heating up, fans are a likely culprit, so the LM80 keeps an eye on them by monitoring the fan tach (note that fans with built-in tach outputs are available from major suppliers like NMB, Mechatronics, and Sanyo) and comparing against a user-programmed minimum-RPM limit.

Oh yeah, don't forget the ominous-sounding Chassis Intrusion (CI) line. Even when the LM80 isn't powered, it can be safely driven by an external circuit (National shows an example using an optosensor).

On powerup, the LM80 dutifully denounces any poor fool that deigns to pop the lid of their PC. Of course, the fate of such a subversive is left to the system designer. It could get ugly.

Another type of PC sensor that's close at hand, literally, are force sensors like those designed by Bokam Engineering for the IBM Trackpoint (see Photos 1a and 1b). I'm no expert, but I fiddled with the arrangement, which plants a small pencil-eraser–like fingertip joystick between the G, H, and B keys. It seems to work well.

The strain-gauge technology is basic, but Bokam's implementation is clever. The strain gauges are screened onto the joystick mounting platform, which acts as a substrate.

Besides low cost, this setup enables easy integration of conditioning electronics. For instance, versions with built-in amplifiers for high-level 0–5-V output are available, as well as ones with built-in micros for digital serial output (e.g., RS-232 or PS/2 keyboard).

Another advantage of this technique is that the substrate can be almost anything. For instance, there's a totally ceramic model that's especially well suited for harsh environment and temperature conditions. Meanwhile, solid-steel noncompliant units might be a boon for truly ham-fisted hackers but are more likely found in industrial force-measurement and motion-control applications.

WHERE AM I?

Oh well, two booths down means that I've got hundreds minus two to go, so I'd better get moving.

Digital compasses are all the rage. Most notably, they're finding homes in most new cars, boats, and presumably any other vehicle that can get lost. You can check out "Do You Know the Way to San Jose" (*INK* 53) for a refresher on the technology.

Honeywell has come up with a solid-state (i.e., no coils or flux gates) upgrade of the technology in their HM line of magnetoresistive sensors. The basic advantage is higher sensitivity and bandwidth for less size and power.

The technology is offered at various levels of integration, starting with the HMC1001 and '1002 (and less-sensitive, lower cost HMC1021 and '1022), which are just the single- and dual-axis MR bridge in 8- and 16-pin IC packages.

Combining the '1001 and '1002 with amplifiers, the hybrid HMC2003 shown in Photo 2 is a three-axis unit with high-level 0.5–4.5-V output (±2 G, which is 1 V/G vs. 15 mV/G for the bridge alone). The HMR2300 throws in the ADCs and a micro to deliver digital output via RS-232 or RS-485 at either 9600 or 19,200 bps.

Notable for virtual-reality applications, the unit can deliver samples at over 100 Hz, which is faster than the CRT refresh rate. That's good because if you update too slowly, the virtual gets out of sync with the reality, which results in an altogether unpleasant situation.

Another way of tracking movement is a gyro. As with compasses, the emergence of electronic equivalents to yesteryear's mechanical lashups promises to proliferate applications.

Consider, for instance, the MicroGyro100 from Gyration. This little goodie uses the electromagnetic equivalent of the Coriolis effect, perhaps best known for causing water to drain with a different spin north and south of the equator.

The MG100 outputs a voltage proportional to angular velocity (about

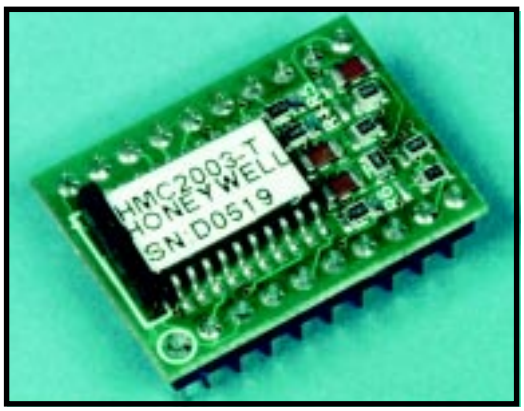


Photo 2—*The Honeywell solid-state magnetoresistive sensor cuts the size of their electronic compass modules. This unit, the HMC2003, provides three-axis high-level (0.5–4.5 V) analog output in a footprint smaller than 1-in.^{2}.*

1 mV per degree per second) in each of two axes.

The MG100 does exhibit some spec limits you should be aware of. First, the response rate is limited with maximum angular velocity of 150° per second and a 10-Hz bandwidth.

Also, the unit is rather sensitive to temperature over its –5°C to 45°C range, and it needs compensation. To help your design achieve the highest possible accuracy, the MG100, which runs on anything from 2.2 to 5.5 V, provides a 1.225-V reference output and includes a temp sensor.

There is also a lengthy power-up delay (almost 3 min., which is itself temperature dependent), and during this delay the output may drift a few degrees.

Instead of power cycling, consider using the sleep mode. This mode cuts power consumption from a few milliamps to only a few microamps, yet it can wake up and get going in less than a second.

If they aren't showstoppers, these spec foibles are a small price to pay because you're only paying a small price for the MG100 ($15 in high volume).

Photo 3—*Measurand Shape Sensors put a unique twist on position detection by translating curvature to force, acceleration, velocity, and flow.*

THE SHAPE OF TAPE TO COME

Measurand puts another twist on the position-sensing angle with its Shape Sensor (see Photo 3) and Shape Tape.

These sensing devices rely on treated optic fiber (laminated into a flexible tape-like ribbon) that modulates light intensity quite accurately, depending on the curvature. By mounting the tape to a cantilever beam of some sort (versions are available with integral spring steel beam), the unit can help measure force, acceleration, flow, and so forth.

The Shape Sensor comes with modular optoelectronics that handle the details and provide a high-level output (1–4 V for a 5-V supply).

Shape Tape goes further, managing up to 64 sensors in a single piece of tape by multiplexing eight LEDs and eight detectors. The sensors are paired to extract both twist and bend info.

After a calibration exercise (flat, coil, twist) and a bit of number crunching (Win95 software is provided), it's possible to determine relative position in free space between the ends of the tape. It works because the free-floating tape drapes into circular arcs between sensors, which along with known position, bend, and twist, make the geometry workable.

Truth is, I'm not so sure what the best application is for this technology. The company makes a case for all manner of positioning apps of the sort served by electromechanical LVDTs (Linear Variable Distance Transducers), with particular mention of biometrics, VR, crash testing, and so on.

However, I am pretty sure that, in the hands of a clever and imaginative designer, this stuff has got to be real good for something.

TALK IS CHEAP?

Talking chips have been around for many years, which is not to say the technology has always been well applied. I can still remember a '70s-era car that kept insisting its "door" was "a jar" until I taught it a lesson with some wire cutters.

While talk is cheap, listening is another story. In fact, practical voice recognition remains one of the holiest, and most elusive, grails.

On the desktop, I understand the technology has made great strides, I imagine largely as a result of PC-MIPS-to-burn brute force. Nevertheless, I don't foresee trading in my QWERTY for a microphone anytime soon.

However, other applications have quite different functionality and cost criteria that may lead to the acceptance of less grand, but more feasible, technology.

A suitable app is one that truly demands voice input (i.e., key entry is not an option) yet can sacrifice one or more blue-sky aspirations such as speaker independence, continuous speech, or huge vocabularies. Likely candidates include voice dialers, toys, security, appliances, and PDAs.

For example, while speaking to my desktop might be nice, the downside of QWERTY isn't exactly life threatening. By contrast, hand dialing a car

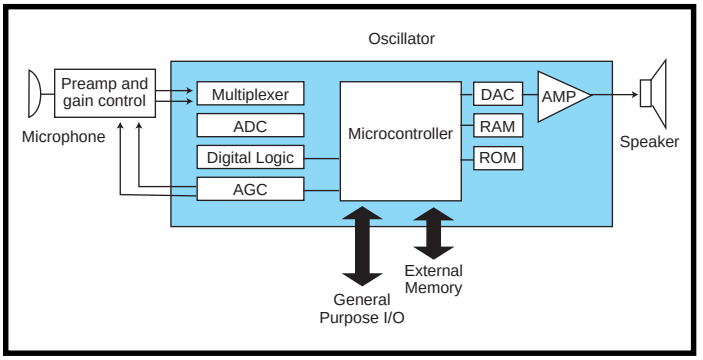


Figure 2—*The Sensory RSC-164 is a micro that talks and, more importantly, listens.*

phone is something that will probably be illegal in your area soon, if it isn't already. What's next, designated dialers?

Sensory Inc. (formerly Sensory Circuits) has carved a niche by supplying low-cost middleweight voice-recognition technology. The RSC-164 in Figure 2 starts with an 8051-like (without the accumulator and data pointer bottlenecks) MCU running at 14.342 MHz (a 32-kHz oscillator is an option for timekeeping).

It then adds the analog front and back ends, which enable the chip to both listen and talk. Both 10-bit DAC and PWM are output options, the latter able to direct drive a small (32 Ω) speaker.

The key to low-cost recognition is the use of neural-network software techniques rather than fancy hardware. The preprocessed patterns are stored in on-chip ROM (speaker independent, standard vocabulary) or external flash (speaker dependent, custom vocabulary).

In addition to recognition, a bit more software enables the chip to handle other useful audio tasks, including voice and MIDI-like music synthesis as well as DTMF and digital voice record and playback.

Sensory makes the RSC-164 available both for custom programming and preprogrammed for standard apps. An example of the latter is the Voice Dialer IC, which targets the previously noted drive-and-dial dilemma.

The Voice Dialer is designed to act as an autonomous and smart subsystem connected to a host CPU by a simple three-wire serial interface. The host merely sends high-level commands (e.g., adding a name to a directory), and the chip takes care of all the details, including voice prompting the user along the way.

As for handling a 60-name telephone directory, Sensory says the recognition rate exceeds 97% even in noisy environments. That's not bad at all for a chip that sells for under $5 in volume.

AN E TICKET

I remember going to Disneyland as a kid, way back when Walt was still in charge. Maybe you too remember how Disneyland used to sell tickets for each ride denominated A through E.

The A tickets were losers, kind of like pennies, good for only the wimpiest rides. Ahh, but the E tickets were good as gold, opening the door to high-speed thrills.

Much is being made of the various field buses that purport to bring order to the digital-control world. To tell the truth, I have trouble keeping up with them all. Like those Disneyland tickets, though, I'm sure the market will decide that some are an A and others an E.

In a rational world, nobody would think of specifying Ethernet as an industrial control network. But, who says anything in this business is rational?

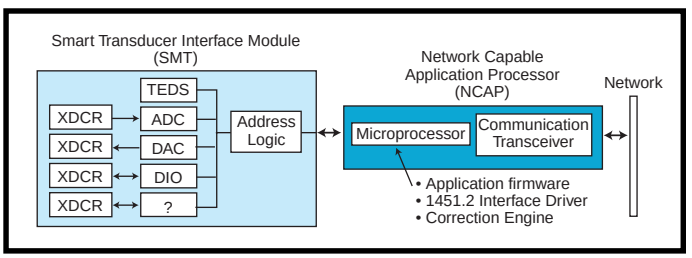


Figure 3—*HP answers the field-bus question with a Network Capable Application Processor (NCAP) that bridges the gap between IEEE 1451.2 sensors and standard networks.*

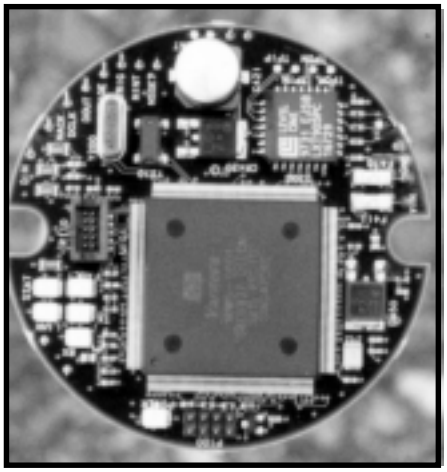


Photo 4—*In addition to a $995 IEEE 1451.2 developers kit, HP showed this prototype NCAP connecting 1451.2 (and RS-232) sensors to Ethernet and via built-in microserver onto the Web.*

Fact is, I think Ethernet is well positioned from a pragmatic, if not theoretical, point of view. A lot of PCs are finding their way onto the factory floor, and the incremental cost of an Ethernet port is approaching zero. In the real world, people just want to plug stuff in.

I feel less squeamish about going out on the Ethernet limb, thanks to backing from heavyweight HP. Their view of the world (see Figure 3), combines the recently finalized IEEE 1451.2 sensor interface standard with Ethernet as a field network.

For a view of what's to come, check out the prototype NCAP (Network Capable Application Processor) in Photo 4. It not only makes the IEEE 1451.2 connection but includes a micro Web server (i.e., easy browser access and control of the sensor subsystem).

SO MANY SENSORS, SO LITTLE TIME

Oops, out of pages already. Needless to say, there's plenty of neat stuff that has to go on the back burner for now.

How about the new line of mixed-signal ICs just introduced by Maxim? Judging by the company's success in other endeavors, I expect great things.

Then there's the cute IR-linked data logger from Lascar, not to mention a dizzying array of temperature, pressure, optical, and tilt sensors.

Whether or not I hit on just the gadget you need, I think there's one lesson we can all learn. Your next design can, should, and will be more sensational than your last.

Tom Cantrell has been working on chip, board, and systems design and marketing in Silicon Valley for more than ten years. You may reach him by E-mail at tom.cantrell@circuitcellar.com, by telephone at (510) 657-0264, or by fax at (510) 657-5441.

SOURCES

Force sensors
Bokam Engineering, Inc.
(714) 513-2200
Fax: (714) 513-2204
www.bokam.com

MG100
Gyration, Inc.
(408) 255-3016
Fax: (408) 255-9075
www.gyration.com

NCAP
Hewlett-Packard Co.
(425) 335-2654
Fax: (425) 335-2220

HMC2003, HMR2300
Honeywell, Inc.
Solid State Electronics Ctr.
(612) 954-2888
Fax: (612) 954-2582
www.ssec.honeywell.com

IR-linked data logger
Lascar Electronics, Inc.
(912) 234-2048
Fax: (912) 234-2049
www.lascarelectronics.com

MAX1458
Maxim Integrated Products
(408) 737-7600
Fax: (408) 737-7194

Shape Tape, Shape Sensor
Measurand, Inc.
(506) 462-9119
Fax: (506) 462-9095
www.measurand.com

LM80
National Semiconductor
(800) 737-7018
(408) 721-5000
www.national.com

RSC-164
Sensory, Inc.
(408) 744-9000
Fax: (408) 744-1299
www.sensoryinc.com

PRIORITY INTERRUPT

Which Numbers Count

When it gets too deep around here, I start threatening to turn this rag into a controlled-circulation trade journal! Well, not really. In truth, it's not where I want to go, but even I get to rant and rave once in a while.

Being a trade magazine would have some definite advantages. I wouldn't have to worry about editorial direction—PR/marketing-generated vaporware is readily available. I wouldn't have to worry about advertising—PR submissions typically include an advertising contract and at least one or two power lunches. I wouldn't have to worry about circulation—heck, when the magazine is free, what's the jeopardy in subscribing to a hundred of them? The worst that could happen is getting caught in a magazine avalanche.

All levity aside, the one ingredient that I didn't mention was readers. I think that if you focus a magazine on attracting loyal readers, it can't use the formula approach represented by traditional trade magazines. In an off-the-record conversation at a recent trade show, the editor-in-chief of a major trade magazine said that he envied our readership loyalty. He also lamented about the continuous solicitation process necessary to maintain high circulation numbers in support of advertising revenues. We both laughed about the typical pile of unread trade magazines on every engineer's desk.

When he volunteered that the real readership of a typical issue probably wasn't more than 20% of the distribution, I was floored. Of course, he didn't have to worry about his "stats." He was audited on how many they mailed, not on how many were read. Wow. I guess that wishing I was in his business meant it was time for a reality check.

When it comes to advertising, big-company PR departments always want solid stats. Certainly, the agencies have to justify their commissions, but I believe that the lack of these stats is frequently an excuse to avoid making decisions. If the editor I talked to is accurate, the value of circulation audits is not only questionable, it's pure advertising extravagance.

I think the only real measure of a magazine is the active readership. Traditional auditing methods monitor potential, not performance. When we get past the PR department obstacles, *INK* is always a proven performer. Want solid stats? Go ask your embedded system designers what they read.

Ultimately, the traditional commercial clique is in for a jolt. The Internet has proven to be a valuable resource for extending the reach of print magazines. But, like all things digital these days, Internet pathways are an accounting exposé. Virtually all servers include statistical packages that register things like user sessions, total hits, page views, and advertiser click-throughs.

Web stats offer a representative demonstration of real performance and perhaps a new undeniable truth. The readership of a small targeted magazine can easily equal the significant value of a larger controlled-circulation audience (in spite of its overwhelming potential) when real reader activity is the measure. The bombshell is in the numbers.

One electronic trade magazine recently went public with their Web stats. They claimed an astounding 350,000 hits per month. Ordinarily I wouldn't care, but I was on our Web stat page at the time—I was looking at 1000 user sessions and 30,000 hits just for the day! Admittedly, that day was above average, but it offers an interesting comparison. Overall, our documented Web traffic with 30,000 circulation easily equals and often exceeds that of the large trade magazine with 150,000 circulation. If you factor in the opinion that only 20% of their 150,000 distribution are active anyway, you coincidentally arrive at 30,000!

You'd think somebody would read the fine print by now. A Web-advertising executive recently told me that big companies think nothing of spending hundreds of dollars for each click-through. Obviously, that PR department isn't walking down to engineering again.

Truth be told, I wouldn't want to do this magazine any other way. Describe us however you want. My goal is to continue publishing a magazine that nurtures an active, involved readership. Occasionally, I have to rant and rave about the obvious insanity of the people and companies in the business. Controlled-circulation offers a wealth of potential, but I believe it comes up short in performance. Of course, when you get right down to it, there's a bit more incentive to read a magazine you're willing to pay for. We certainly appreciate your support.

steve.ciarcia@circuitcellar.com