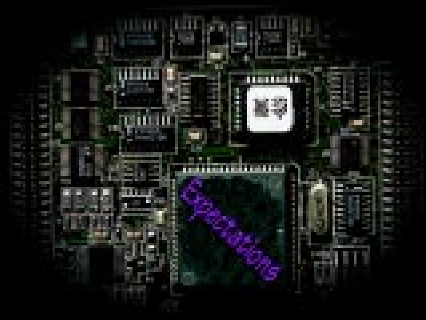
www.circuitcellar.com

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

#113 DECEMBER 1999

EMBEDDED INTERFACING



X-10 Temperature Sensor

Generating Thermoelectric Power

Network Data Collection

Near Space—
The Rediscovered Frontier

CIRCUIT CELLAR ONLINE

Double your technical pleasure each month. After you read *Circuit Cellar* magazine, get a second shot of engineering adrenaline with *Circuit Cellar Online*, hosted by ChipCenter.

WWW.CIRCUITCELLAR.COM/ONLINE
Table of Contents for November 1999

INTERNET PIC® 2000 CONTEST
www.circuitcellar.com/pic2000

INSIDE ISSUE 113

Managing Managing Editors

Elizabeth is moving to Boston this weekend. Her Doctoral thesis combined with the relocation make being a fulltime editor impossible. When Steve hired me to learn the ropes, we didn't think I'd be soloing so quickly. It's OK, though. There are a lot of experienced people ready to help. For example, Ken Davidson was here a few days ago dropping off editorial proofs on some of the articles for this issue. I've just been looking over some correspondence from Janice and Dave. It seems that editors never actually leave here - Elizabeth included. They just change their orbits slightly. I think I'm beginning to see why that is.

I started here as managing editor a few weeks ago, after seven years in academia editing engineering articles and proposals. The most refreshing difference I'm finding between those articles and the ones I've seen here is that Circuit Cellar articles are *real.* Each of them describe not only an actual device or system, but a step-by-step method for replicating it.

That sort of procedure is essential to the scientific method and is common to the types of experimentation and development found in both college laboratories and in basement workshops. In working at a University however, I sometimes felt that researchers frowned upon anything sufficiently developed as to be practical and immediately useful. It really pained some of them to have an idea simplified, let alone realized. Maybe they just don't like to share?

It's quite a contrast to be working here and seeing such a variety of authors sharing their experimental results so freely and with such enthusiasm. While some of the similarly-themed trade magazines boast of a large "readership", more of the qualities of a community apply here. The information flows in two directions, at the very least.

The articles reflect a diverse population of readers as well. It is particularly satisfying to see the projects that cross over from electronics into other areas of science. Paul Verhage's discussion of constructing and outfitting an instrumented weather balloon and Randy Heisch's re-engineered device for claiming electricity from heat fall into that category. If you include mathematics, Carl Huben's subversion of Poisson distributions to simplify sensor networks is another example. I look forward to those almost as much as the more straightforward articles and the in-depth discussions of emerging technologies and standards.

The theme of this issue is embedded interfacing, but by now you know that this is only a starting point. In FTB, Jeff makes some refinements to his 3-D accelerometer array and at last leaves the Circuit Cellar elevators alone. Tom weighs in from the West Coast with an early review of the first AI-64 chip in the latest Silicon Update, and Fred takes a look at the new, old idea of thin clients.

[illegible]

steve.meyst@circuitcellar.com

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

EDITORIAL DIRECTOR/PUBLISHER
Steve Ciarcia

MANAGING EDITOR
Steven Meyst

TECHNICAL EDITORS
Michael Palumbo   Rob Walker

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
Mike Baptiste   Ingo Cyliax   Fred Eady
George Martin   Bob Perrin

NEW PRODUCTS EDITOR
Harv Weiner

PROJECT EDITORS
Steve Bedford   Ken Davidson
Janice Hughes   Elizabeth Laurençot
David Tweed

EDITORIAL ADVISORY BOARD
Ingo Cyliax   Norman Jackson   David Prutchi

ASSOCIATE PUBLISHER
Sue Skolnick

CIRCULATION MANAGER
Rose Mansella

CHIEF FINANCIAL OFFICER
Jeannette Ciarcia

CUSTOMER SERVICE
Elaine Johnston

ART DIRECTOR
KC Zienka

GRAPHIC DESIGNER
Jessica Nutt

STAFF ENGINEERS
Jeff Bachiochi
John Gorsky

QUIZ MASTERS
Ray Payne
Benjamin Day
Bob Perrin

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

ADVERTISING

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

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

ADVERTISING CLERK   Sally Collins

CONTACTING CIRCUIT CELLAR

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, 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 Ciarcia (860) 875-2199 or e-mail jciarcia@circuitcellar.com.

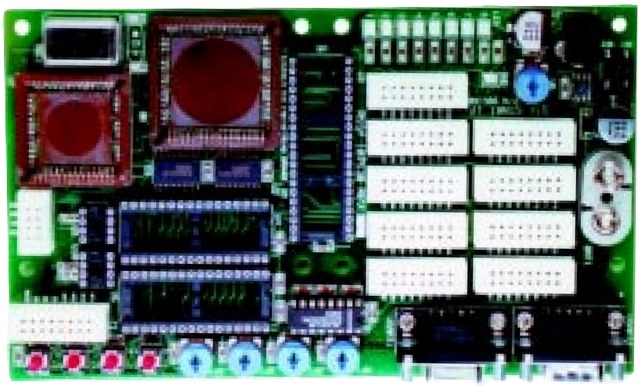
CIRCUIT CELLAR®, THE MAGAZINE FOR COMPUTER APPLICATIONS (ISSN 0896-8985) and Circuit Cellar Online are published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (860) 875-2751. Periodical rates paid at Vernon, CT and additional offices. **One-year (12 issues) subscription 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 Subscriptions, P.O. Box 698, Holmes, PA 19043-9613 or call (800) 269-6301.

Postmaster: Send address changes to Circuit Cellar, Circulation Dept., P.O. Box 698, Holmes, PA 19043-9613.

NEW PRODUCT NEWS

Edited by Harv Weiner



DEVELOPMENT MODULE

The **Qik Start 17** development module is a standalone prototyping and training tool that enables design engineers to develop hardware and software for PICmicro 17-series microcontroller projects. The module can be used as a stand-alone tool or installed in a benchtop design center. It allows an expanded prototyping area, interface components, signal generation, keypad, output displays and more, to be quickly and easily interfaced. It joins the family of current modules available for development, which include the **Qik Start 16** and **PICmicro Education Board**.

The module will accept DIP40, PLCC 68, and PLCC 84 packaged microcontrollers. All VDD, VCC, OSC, and MCLR are pre-wired, and the OSC signal is generated from an onboard 4-MHz plug-in clock oscillator or crystal. The 2–5-V adjustable power supply can be sourced from a 9-V battery or AC power pack. The MCLR can be initiated from the onboard momentary reset button or via an external reset source.

There are two dedicated sockets for EEPROM (either I^{2}C or SPI) and bus memory can be added by plugging in the appropriate ICs. Two serial ports are tied to the Tx and Rx lines via a MAX232 RS-232 interface chip to a male and female DE-9 connector. All port pins are brought out to connecting terminal blocks for easy access. It also has four potentiometers, three momentary push-button switches and eight LEDs complete with series resistors.

The Qik Start 17 Module sells for **$144**. A Power Pak is available for **$18** and a Wire Jumper Kit for **$21**.

Diversified Engineering and Manufacturing, Inc.
(203) 799-7875
Fax: (203) 799-7892
www.diversifiedengineering.net

KEYPAD INTERFACE MODULE

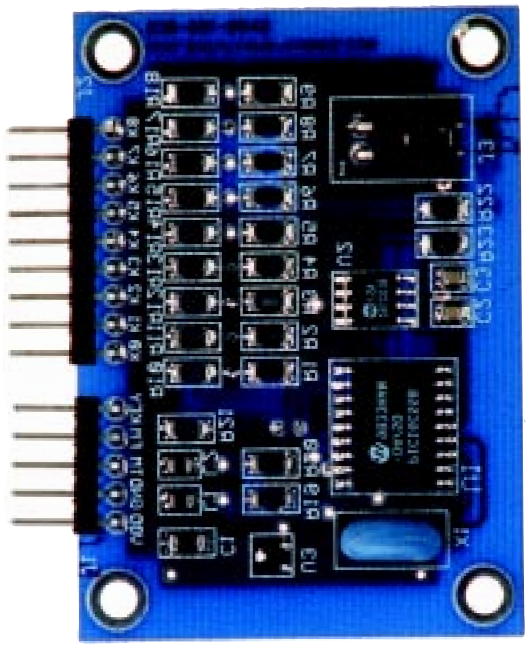
The **MEMKey** is a fully programmable keypad encoder that supports (via jumper) either a simple serial communication protocol or the standard PC/AT communication protocol. In either communication mode, it can decode key matrixes of up to four columns by five rows. The rows and columns can be programmed to match the row-column configuration of any off-the-shelf keypad.

The value returned by the MEMKey can be programmed to any standard value. In addition, the debounce time and typematic rate are fully programmable. All programmable values are stored in nonvolatile memory so they are saved when power is off. When operating with the serial protocol, the MEMKey communicates at 2400 bps, 8N1, LSB first, asynchronous. This can be either a one-wire or a two-wire interface. In addition, there are 64 bytes of EEPROM that are made available to the user as scratchpad space.

The MEMKey's small size and connection scheme allow the device to be inserted directly into circuit boards for production runs or into breadboards for easy prototyping. Complete datasheets and application notes are available online.

The MEMKey comes in a 1.6″ × 2.25″ SIP module, and sells for **$36.00**.

Solutions Cubed
(530) 891-8045
Fax: (530) 891-1643
www.solutions-cubed.com



NEW PRODUCT NEWS

STEPPER MOTOR CONTROL KIT

The **Stepper Motor Prototyping Kit** from Mosaic Industries facilitates prototyping motion-control applications. The kit provides two stepper motors and incorporates the functionality of indexers, motors, drivers, and high-level software control.

Running at 200 steps per revolution, the two 4-phase unipolar stepper motors with 1.8° resolution are mounted on a printed circuit board, which is interfaced via a ribbon cable to a QED digital I/O board. The digital I/O board provides the required high-current drivers for the unipolar stepper motors. Pre-coded library routines also make it easy to specify starting/jogging speeds, as well as acceleration and deceleration rates for each motor.

Simple functions can be called on to specify speed, change speed, and specify the number of steps to be taken. A QED board (not included in the kit) runs the high-level motor control and application program.

The price of the kit is **$279** and the QED Board is **$495**.

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

NEW PRODUCT NEWS

THREE-CHANNEL THERMISTOR CONVERTER

The **TH-03** Thermistor-to-PC-Converter is a highly accurate, low-cost adapter that connects to a computer serial port and allows up to three channels of temperature data to be recorded. It is ideal for applications where measurements close to the computer are needed.

The TH-03 measures temperatures from –55°C to 300°C with an accuracy of ±0.2°C. Optional sensors allow the user to check light levels and record the position of a door or any magnet or micro-switch response. The unit uses external precision thermistor sensors to give an accuracy that was previously only obtainable using expensive platinum PT100 sensors.

Supplied with PicoLog software for collecting data at rates from once a second to once an hour, TH-03 can be programmed to sound an alarm if a temperature goes out of range. PicoLog has a wide range of functions to assist in the analysis of the information that it collects, and to transfer the information easily to other applications for further processing.

Data can be displayed on a PC in graphical or text format, both during and after collection. TH-03 is supplied with DOS drivers with program examples in C and Pascal, and Windows 3.1/95/NT drivers with examples for C, Visual Basic, and LabView. It does not require a power supply.

The TH-03 sells for **$129** and comes complete with manual and PicoLog software.

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

READER I/O

GOT HELP?

I know my limits, and I know that I am too out of practice when it comes to programming. But, today's handheld calculators have as much power as early PCs. The TI line sports a 6-MHz Z-80 or a 68000 chip (depending on the model).

These devices are handheld computers with great potential. One of these units (perhaps a TI-86 or 92, or an HP-48) would be an ideal field terminal for programming, or even just checking the function of a PIC or a single-chip computer.

I'd do it myself if I had done any serious programming in the last 10 years. Anybody have any ideas?

Mike Walker
White Lakes, Wisconsin

THANK YOU

Just wanted to say thank you for putting the *Circuit Cellar* back issues on CDs. I used them for the first time while I was trying to get info on IrDA and they are good enough to allow me to free up some shelf space!

Phil Howson
p.howson@dial.pipex.com

PARTS BIN

Z-World had been trying to give away an SMT IR reflow oven for months and had been unable to drum up interest in it. After I posted it in the Parts Bin, we got a good number of inquiries. Before too long, a big truck was hauling the oven off to a *Circuit Cellar* reader. Thanks!

Bob Perrin
bob@mobots.com

LET ME COUNT THE WAYS...

Whether it's back-issue CDs or Parts Bin, Circuit Cellar *is always looking for new ways to provide a helpful resource of engineering information.*

Those of you who remember back to Steve's BYTE *days, may recall a column called "Ask* BYTE.*" Because* Circuit Cellar *readers have always been a great source for input, we'd like to hear from you if you think an "Ask Circuit Cellar" forum on our web site would be useful. Of course, we would need some engineers and programmers who'd be willing to make a few bucks doing research and providing answers for the questions.*

So, as always, we're listening. Drop us a note at answers@circuitcellar.com. If you're interested in answering questions, give us a brief bio and what topics or areas you would like to handle.

And here's just a few more of the things we've done on our web site:

- *Navigating through our new homepage is easy with our Site Navigation feature.*
- *You can now subscribe to* Circuit Cellar *or renew your current subscriptions from our web page.*
- *If you're missing an issue or two, or if you're just looking for a certain issue or article, you can view the issues we have available and order the individual back issues you need.*
- *We've included a keyword search for all of the articles from 1999. So, now you can spend less time searching for topics that we've covered this year.*
- *Ordering your CD-ROM of 1999* Circuit Cellar *back issues is quick and easy with our secure ordering form.*

Thanks to all of our readers for another great year of feedback and ideas. Keep up the good work.

Rob Walker, Editor
rob.walker@circuitcellar.com

STATEMENT REQUIRED BY THE ACT OF AUGUST 12, 1970, TITLE 39, UNITED STATES CODE SHOWING THE OWNERSHIP, MANAGEMENT AND CIRCULATION OF CIRCUIT CELLAR, THE MAGAZINE FOR COMPUTER APPLICATIONS, published monthly at 4 Park Street, Vernon, CT 06066. Annual subscription price is $21.95. The names and addresses of the Publisher, Editorial Director, and Editor-in-Chief are: Publisher, Steven Ciarcia, 4 Park Street, Vernon, CT 06066; Editorial Director, Steven Ciarcia, 4 Park Street, Vernon, CT 06066; Editor-in-Chief, Steven Ciarcia, 4 Park Street, Vernon, CT 06066. The owner is Circuit Cellar, Inc., Vernon, CT 06066. The names and addresses of stockholders holding one percent or more of the total amount of stock are: Steven Ciarcia, 4 Park Street, Vernon, CT 06066. The average number of copies of each issue during the preceding twelve months is: A) Total number of copies printed (net press run) 29,900; B) Paid Circulation (1) Sales through dealers and carriers, street vendors and counter sales: 4,063, (2) Mail subscriptions: 22,220; C) Total paid circulation: 26,283; D) Free distribution by mail (samples, complimentary and other free): 405; E) Free distribution outside the mail (carrier, or other means): 229; F) Total free distribution: 634; G) Total Distribution: 26,917; H) Copies not distributed: (1) Office use leftover, unaccounted, spoiled after printing: 368; (2) Returns from News Agents: 2,615; I) Total: 29,900. Percent paid and/or requested circulation: 97.6%. Actual number of copies of the single issue published nearest to filing date is (November 1999, Issue #112); A) Total number of copies printed (net press run) 28,600; B) Paid Circulation (1) Sales through dealers and carriers, street vendors and counter sales: 3,790, (2) Mail subscriptions: 21,720; C) Total paid circulation: 25,510; D) Free distribution by mail (samples, complimentary and other free): 50; E) Free distribution outside the mail (carrier, or other means): 0; F) Total free distribution: 50; G) Total Distribution: 25,560; H) Copies not distributed: (1) Office use leftover, unaccounted, spoiled after printing: 409; (2) Returns from News Agents: 2,631; I) Total: 28,600. Percent paid and/or requested circulation: 99.8%. I certify that the statements made by me above are correct and complete. Susan Skolnick, Associate Publisher.

The Poisson Network

FEATURE ARTICLE

Carl Huben

A One-Way Architecture for Data-Collection Applications

The technique described here could cut the cost, complexity, and power consumption of every sensor. An unusual approach to data collection? Perhaps, but with Carl as a guide, you can go from network basics to concept implementation in one sitting.

This article describes an unusual approach to data collection that may at first glance seem unworkable. To be fair, its usefulness is not exactly wide-ranging, for applications that do not require data on demand, this technique cuts the cost, complexity, and power consumption of every sensor in the network.

After explaining how this network ticks, I'll get into the nuts and bolts of designing it to meet specific performance criteria. In the final section, I'll trot out my own proof-of-concept implementation.

CONVENTIONAL VS. POISSON

The technique I describe here is best understood in relation to conventional data-collection networks, in which each sensor node is equipped with a transceiver (if only in the most general sense) and some external means of coordination is employed to make effective use of channel bandwidth. For example, sensors might be polled one at a time by a central hub—a fairly typical and efficient scheme.

In contrast, the network I propose here consists of a pool of completely autonomous nodes, each equipped with just a transmitter, and a central receiver with no power over the channel. Assuming it can be made to work, such an architecture offers a mixed bag of costs and benefits.

On the upside, we can axe the receiver, often the most unwieldy component in the whole design, from each node. This deletion cuts the cost of each sensor, enhances reliability, and, if the sensors are battery powered and disposable, stretches lifetime. The network as a whole becomes less complex and easier to maintain than a conventional one in which the central hub exerts control over the nodes.

Of course, there is a price to be paid—messages lost in contention are not recoverable, and transmissions from a particular node cannot be elicited on demand. The downside is serious, but not insuperable. There are many real-world applications in which these features are not strictly necessary.

The real problem, then, is this: without some facility for external control, how can we reliably obtain data from the nodes?

We need a triggering algorithm for the transmitters that can deliver, to a specified probability, a particular level of per-node throughput. Also, we want it to run the same way on all nodes, using identical hardware and firmware, and it should be simple to configure. But what kind of scheme fits the bill?

The solution is to take a standard tool of network analysis—the Poisson process—and stand it on its head (i.e., purposely engineer nodes to send their data at Poisson-random times).

As a result, the network takes on some surprising properties. First and foremost, its operation may be characterizeded mathematically, making it possible to design for specific performance criteria.

Nearly as important, all nodes operate the same way. They can use identical hardware and software, and each enjoys the same long-term throughput.

A formal derivation of the relevant design equations appears in the sidebar on page 18. There are two equivalent methods for specifying the desired performance.

One is to set the long-term average per-node reception rate. The other is to specify the minimum probability of receiving at least one valid packet per node in a given time frame. In either case, you are solving for the same quantity, λ_a, the Poisson-rate constant ultimately used by the nodes.

To demonstrate the feasibility of this technique, I designed a transmitter and receiver capable of Poisson triggering. The battery-powered transmitter sends data packets over a single channel radio link to the receiver, which verifies the integrity of incoming messages and passes valid packets to a PC over a serial link.

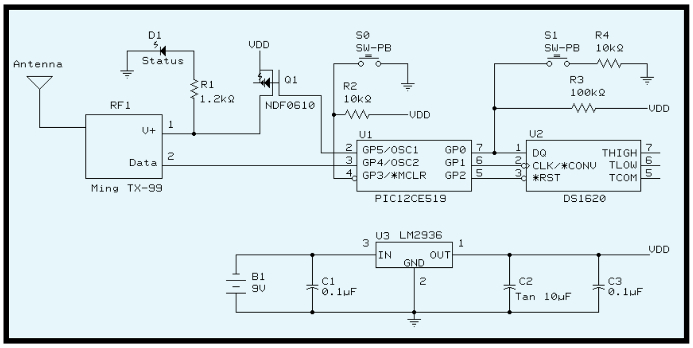


Figure 1—*This module transmits packets containing the transmitter ID and the current temperature at Poisson random times over a single channel radio link. The rate constant and other operational settings are programmed via a pusbutton and LED interface. This proof-of-concept design supports 256 unique IDs, rate constants from under a second to almost four hours, measures temperature to 0.1° F, and has a battery life of two years.*

TRANSMITTER

The transmitter (see Figure 1) features extremely low standby current (< 20 µA), and a low parts count, and it may be readily adapted to a variety of sensors.

It is built around the PIC12CE519. This quirky but capable microcontroller chip oversees the whole operation—data collection, transmission, power switching, as well as the Poisson trigger.

The Poisson-rate constant and an 8-bit node ID are stored in the PIC's 16-byte EEPROM area and may be set via a push-button interface. For details, see the sidebar on the page 15.

In the this design, data is taken from a DS1620 temperature sensor from Dallas Semiconductor. For simplicity and flexibility, off-the-shelf hardware is used for the radio link: Ming's TX-99 serial RF transmitter, and the matching RX-99 receiver used in the circuit in Figure 2.

All aspects of hardware design are keyed in to the idea of micropower operation. I used an LM2936 to regulate down to 5 V from a 9-V battery. This semiconductor consumes a paltry 10 µA at zero load.

A P-channel FET switches power to the RF module and the LED. In fact, you can omit this item from the sample design because the port pin itself has enough juice to cover these two components. Note, however, that you must tweak the firmware (set USINGFETPOWERSW to 0) and short the gate and drain lines if you ditch the FET.

Why is the FET there, then? Two reasons. First, the port pin does not deliver a true 5.0 V, and it has limited drive capability. This could be a problem if other, more powerful RF boards are substituted for the TX-99.

Second, the FET may come in handy when adapting this circuit to other sensors. The DS1620 just happens to have such a low quiescent draw (<1 µA) that it could be hooked right into the main power. But in general, sensors and their supporting circuitry require power switching.

ABOUT THE PIC12CE519

This is a chip only a hacker could love. Its architecture is quite convoluted. Check out these limitations:

- two-level stack
- no plain-Jane return opcode; RETLW must be used exclusively
- called functions must reside in low half of each program memory page

These restrictions make it difficult to write a program of any length without resorting to MPLINK, Microchip's tool for creating executables from relocatable blocks of code.

MPLINK is usually helpful, but for this application, I found myself wasting gobs of space on page and bank-switching macros. So, for the purposes of the transmitter firmware, I whipped up a more efficient and transparent scheme that relies on jump tables. Just make sure you understand what is going on before you touch the code at the head of each page.

Despite these idiosyncrasies, this chip is perfect for this application. It has plenty of horsepower, NVRAM, and a low-power sleep mode. And, it costs next to nothing.

SIMULATING POISSON TRIGGERING

An excellent approximation to a Poisson process is obtainable with no external hardware whatsoever. You should be able to get similar results on any modern embedded CPU.

Here, in a nutshell, is the basic idea. Poor-quality environmental noise is used to generate a seed value for a linear congruential pseudo-random number generator. After each packet transmission, the generator is tapped once or twice (a decision based on environmental noise) to produce a new uniform deviate.

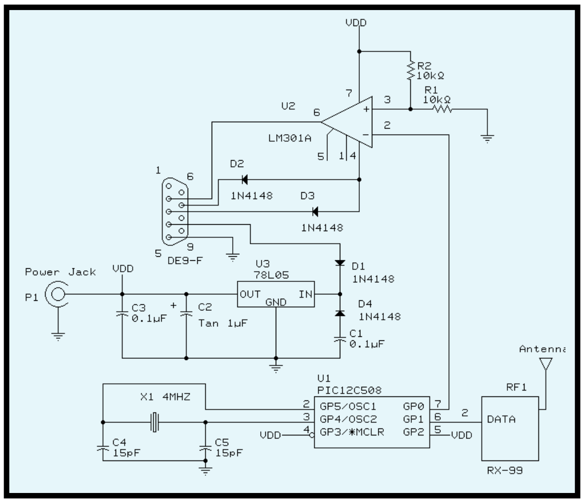


Figure 2—*This module receives packets containing 8-bit transmitter IDs, the current temperature, and CRC-16 checksums from up to 256 Poisson transmitters. The data passes to a PC through the serial port.*

Because intervals between successive Poisson-random events conform to the so-called exponential distribution, the new random number is transformed mathematically into an exponential deviate and scaled up by the current Poisson-rate constant to give the delay between one current transmission and the next.

I use the Park-Miller Minimal Standard 31-bit random number generator for uniform deviates [1]. Algorithms like this generate new pseudo-random numbers based on their last result, and so a four-byte area is given over to the generator and always holds the current deviate.

Using environmental noise to establish a seed value for the generator ensures that different transmitters generate novel sequences from the start. The seed is assembled bytewise from four iterations of a routine that measure the differential between the watchdog timer and the internal RC clock. Both timers are built into the PIC. The Δ*t* on the maximum watchdog delay provides some randomness.

Each iteration, the watchdog is set to the maximum (nominally 2.3 s), and TMR0 is set to increment every internal clock cycle and left to run until the watchdog resets the PIC. On reset, the value of TMR0 is read, it is XORed with the transmitter's 8-bit ID code (for good measure), and the result written to the seed register.

By the way, this produces good results even for identical ID codes. Another important note: we have to make sure the seed value is nonzero. Generators like the PM choke on naughts.

THE MAIN LOOP

OK, so on powerup we run through the seed generator, and from then on each call to PM results in a nice, uniform random deviate, with all values from 1 to 2^{31}—1 equally probable (an idealization; see the Park Miller paper).

After this and other initializations, the PIC enters an endless loop consisting of these steps: read the sensor, transmit the data, generate a new uniform random number, and use it along with the Poisson-rate constant to compute the next sleep period, sleep, repeat. Let's take a closer look at each step.

The sensor in this example is a DS1620. This chip painlessly delivers a 9-bit digital temperature over the range of –55°C to 125°C through a three-wire serial interface, with conversion times under a second.

My design uses the technique described in Dallas' app note 105 to wring out a few more bits of resolution. Right after each measurement, the raw high-resolution temperature is converted to IEEE 32-bit floating-point format and added to a transmit buffer.

Next, the RF module is flipped on and used to transmit the data. There are some issues here best covered in connection to the receiver design, so I'll just give the broad strokes.

You need to send the measurement, the box's 8-bit ID, and a CRC-16 based on the packet before going off-air. Packet transmissions are accompanied by a flash from the LED.

Configuring the Transmitter

Assuming you have done the necessary calculations and have the Poisson constant in hand, the next step is to configure the transmitters. Four byte-wide registers hold the relevant operational constants:

0x00—flags (bit 0 : Set=Use poisson trigger / Clr=Use periodic trigger)
0x01—least significant bit of rate constant
0x02—most significant bit of rate constant
0x03—transmitter ID

For a given Poisson rate of λ_e transmissions per second, you can compute the 16-bit rate constant from this relation:

$$\text{RateConst} = \frac{4.81}{\lambda_e}$$

The transmitter supports average rates from 4.81 transmissions/s down to 1 transmission every 3.7 h. To set the transmitter for an average rate of one transmission per minute, the appropriate rate constant equals

$$\frac{4.81}{\frac{1}{60}} = 289$$

Note that the actual interval may vary, depending on the accuracy of the onboard watchdog timer. In practice, the period is becomes slightly dilated by the time used to take and transmit measurement data.

Also, there is a provision for periodic triggering. In this mode the packets are transmitted at precise intervals using the same rate constant. To change the value of a register, disconnect the battery and reconnect it while holding down one pushbutton.

The third step is to release the button, and enter the desired register address as an 8-bit binary number, most significant bit first. Use S0 to toggle the current bit value and S1 to enter the current bit.

The LED shows the current bit value while S0 is down—the LED stays lit for a 1 and flashes to indicate 0. The bit value starts out as 0, and defaults to 0 after each press of S1. For example, to enter address 0x02, you would press S1 six times, then S0 once, followed by S1 twice.

After the last bit of the address has been entered, the register's contents will be displayed on the LED as an 8-bit binary number, using the coding scheme found in step three. The LED goes dark for a fraction of a second between successive bits.

Next, you can enter a new 8-bit value for the register. The transfer does not take place until the eighth bit is entered. The LED lights up and stays on to indicate success. To abort the process, disconnect the battery at any time up to that point.

To set the value of a different register, start the process over. To resume normal operation, interrupt the power for a few seconds and reconnect the battery with both pushbuttons up.

To restore factory settings, reconnect the battery while holding both pushbuttons. The LED will light up and remain on indicating success.

After that, you have to cook up a new sleep interval. Begin by calling the random number generator once or twice, depending on the CRC-16 computed in the last step. The checksum is a convenient pool of random bits, since small changes in the measurement have a big effect on its value.

I include this step because of its relevance to large Poisson networks. When you get into the thousands of nodes, there is a real possibility of two transmitters starting up on the same seed. This additional measure prevents them from remaining in lockstep.

It is counterintuitive, but there is a greater than even chance of this occurring in a system of just 55,000 nodes. Actually, the true number is much smaller because seeds are not perfectly random. So, it's a worthwhile precaution regardless of network size.

In any case, the next step is to convert the 31-bit uniform deviate to a 16-bit exponential deviate by taking its natural logarithm and negating this result. Then we multiply the result of that equation by the 16-bit Poisson-rate constant from EEPROM, the eight low bits are dropped and the 24 high bits become the next sleep interval.

The PIC has a flexible sleep timer, and propped up by a few lines of code, the 24-bit sleep is carried out with a minimum of wakeups, from 0 ms to the maximum of almost 84 h with a granularity of just 18 ms.

RECEIVER

The transmitter is where all the action takes place. In contrast, the receiver's job should be a cinch. All it has to do is recognize incoming packets and pass them to a PC through the serial port.

In my design, the receiver leeches power from the serial port lines, so you must assert DTR high and RTS low for the thing to work. Naturally, you should keep the serial cable as short as possible.

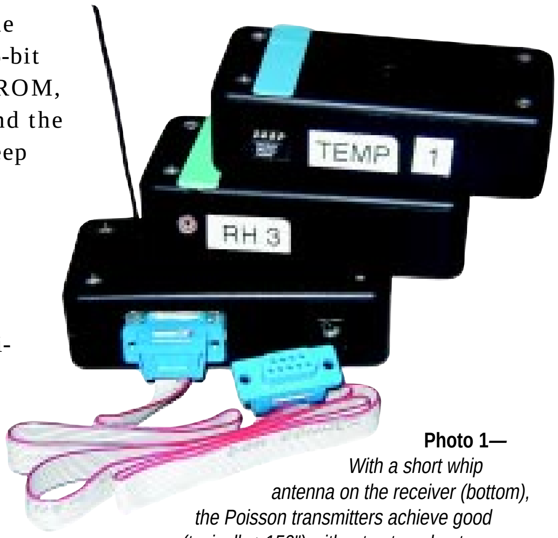


Photo 1— *With a short whip antenna on the receiver (bottom), the Poisson transmitters achieve good range (typically >150") without external antennas. On top is the transmitter design covered in this article.*

This arrangement works well on the majority of PC's. However, to ensure that the receiver can be used on as many machines as possible, I included an external power jack in the design. Through this jack, an ordinary wall transformer can supply the lion's share of the receiver's power needs, allowing it to run on virtually all PCs

Photo 1— *Here is the inside of a transmitter module. On the circuit board are is PIC that does the Poisson calculations, and the pushbuttons and LED that make up the (minimal) user interface.*

DESIGN CONSIDERATIONS

Packets as transmitted are compatible with the RS-232 protocol, so that the RF module's output could, in theory, be used to drive the port TX line. However, a design like this could get you fired, for several reasons.

For one, the output from RF modules is not always well behaved. Some modules output static between transmission—not something we want going out on the serial port line, because it can lead to spurious receptions and other nasties.

There is another problem, specific to this transmitter design. Packets are sent at a nominal rate of 1200 bps, but the actual rate is tied to the PIC's clock speed.

Because the design traded off crystal precision for extra port pins, clock speed is quite variable over temperature. The delta can be large enough to bollix up the PC's UART, which can only tolerate data-rate variations of a few percent.

To iron out these wrinkles, in the receiver design I interpose a PIC-12C508 between the RF receiver and the port driver. The PIC listens to the RF module for incoming messages, using a timing byte at the head of each packet to fix the data rate for the rest of the packet. Note that this PIC uses the crystal clock option and is therefore able to talk to the PC with no problem.

The receiver algorithm is finicky about valid signals. It resets on botched start and stop bits, successive bytes separated by more than two bit periods, and if the checksum at tail of the message does not agree with the locally computed value.

The hair-trigger reset is vital because it allows us to easily construct a transmission sequence guaranteed to reset the receiver.

Design Equations

The behavior of simple packet networks has been examined thoroughly in many papers, and design equations are well known. Relevant to this derivation is the equation for channel throughput, adapted from [2]. Assume N users send fixed-length packets, τ seconds in duration, to a central receiver over a common channel. Each user's transmission events follow a Poisson process of rate λ_a packets/sec. From the perspective of a particular user, the throughput is:

$$\lambda_e = \lambda_a e^{-2\lambda_a \tau (N-1)}$$

This equation can be used to compute λ, the Poisson rate constant used by the transmitters, from λ_e, the desired long-term per-node reception rate.

To use such a network as described in this article, it may be preferable to select λ_a so that, to a given probability (μ), at least one valid packet arrives within a given time period (T). To derive a relation between μ, T, and λ_a, recognize that for a Poisson process of rate λ_e, the probability distribution of the time until the next event, $F(t)$, is equal to:

$$F(t) = 1 - e^{-\lambda_e t}$$

Setting $F(t) = \mu$ and $t = T$, and solving for λ_e:

$$\lambda_e = -\frac{1n(1-\mu)}{T}$$

Combining with the first equation, we arrive at:

$$\lambda_a e^{-2\lambda_a \tau (N-1)} = -\frac{1n(1-\mu)}{T}$$

Why is this so important? Because just prior to a valid packet, the receiver may be occupied, running up a blind alley in the static. If we don't have some way of resetting the receiver just prior to a real packet, it may miss the first few bits. Therefore, the transmitter is designed to prepend a reset-inducing header to the packet, and everything is copacetic.

I wrote a simple data logger in QBASIC to help me test the network hardware in the article. This program is only intended as a demo, but it does support up to 16 transmitters, has a constantly updated screen display of the current and average temperature for each node, and may be configured to print a periodic summaries of the collected data. This program is available free, along with the rest of the source code for this article (see the software, section).

Currently, I am developing commercial versions of the network hardware. Among other improvements, the new design will run on lithium coins, eliminating the need for a regulator, and extending battery life considerably. Sensors for humidity and pressure are also in the works.

Although Poisson networks are of limited applicability due to their shortcomings, they do offer some unique advantages. For example, it might be good for collecting environmental data from an array of cheap sensors scattered over a large area. Or for monitoring the tire pressure of the tires on an 18-wheeler. For applications that require low power consumption, low weight, and low cost, this network architecture might be worth a thought. ■

Carl Huben is a embedded-systems specialist for his own firm, Huben Consulting. Since earning an engineering degree, he has been a programmer of various stripes. You may reach Carl at cchuben@ix.netcom.com.

SOFTWARE

The QBASIC data logger and firmware can be downloaded via the Circuit Cellar web site.

REFERENCES

[1] S. K. Park and K. W. Miller, "Random Number Generators: Good Ones Are Hard To Find," *Communications of the ACM*, **31**, 1192-1201, 1988.
[2] N. Abramson, "The Throughput of Packet Broadcasting Channels," *IEEE Transactions on Communications*, **25**, 117–135, 1977.

SOURCES

PIC12CE519, PIC12C508
Microchip Technology, Inc.
(888) 628-6247
(480) 786-7200
Fax: (480) 899-9210
www.microchip.com

DS1620
Dallas Semiconductor
(972) 371-4448
Fax: (972) 371-3715
www.dalsemi.com

TX-99, **RX-99**
Ming Microsystems
(818) 912-7756
Fax: (818) 912-9598

Being Cool is Easy

FEATURE ARTICLE

Donald Blake

A Temperature-Sensing Control Device

Skip the expensive sunglasses and red sports car. According to Donald, all you need is a microcontroller, an X-10 temperature sensor, and some interfacing software to control a fan to stay cool on a hot summer day.

My experiments with Microchip's PIC line of microcontrollers began with the '16C84. The PIC16C84 is a good starting device. Its 1-KB EEPROM is quickly reprogrammed and inexpensive programming devices are commercially available or easily built from readily available plans.

The elemental structure of any embedded architecture consists of input, processing, and output. My initial experiments started with the output component. I used a 2-line by 16-character LCD module.

The LCD module, which contains an onboard controller, has a relatively simple interface. I used the Optrex DMC16207, although there are a number of other manufacturers with many sources for new and surplus devices. The interface is documented in Optrex's databook and there's a good technical document available online.

Now that I had a working output device capable of presenting up to 32 characters, I needed an input device. I've always been fascinated with temperature measurement and that's where I focused.

My first temperature sensor was an analog device. An external ADC as well as a voltage reference was required because these functions aren't built into the '16C84. Another downside to the analog device is that a constant current source is required when the sensor is placed at any significant distance from the ADC.

I discovered the Dallas Semiconductor DS1820 1-Wire digital thermometer shortly after implementing the analog sensor. The DS1820 provides a digital interface, eliminates several external components, gives better accuracy, and permits the sensor to be located a considerable distance from the microcontroller.

Using the '16C84, LCD display and the DS1820 digital thermometer, my first complete PIC project was a simple indoor/outdoor thermometer that recorded minimum and maximum temperature, and displayed readings in Fahrenheit and Celsius.

THE X-10 INTERFACE

I've used the X-10 line of home automation (HA) products for many years. I started with simple one-way "dumb" controllers and later migrated to the programmable CP-290 X-10 Powerhouse. My current HA setup uses the Homebase intelligent controller by Home Controls, Inc. I decided on an X-10 interface for the next step in my PIC experiments.

Homebase uses the X-10 TW-523 two-way power line interface module to send and receive X-10 commands. My one and only TW-523 module failed a short time back and left me to rely on my old faithful CP-290 for several days. I ordered a replacement TW-523 along with a few spares.

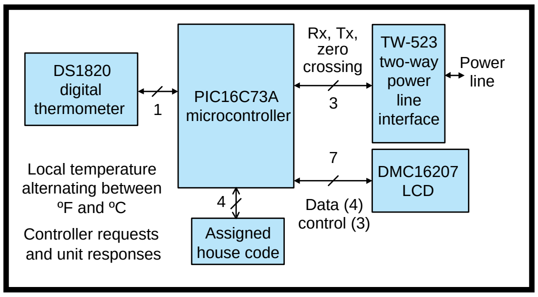

DS1820 digital thermometer
1
PIC16C73A microcontroller
Rx, Tx, zero crossing
3
TW-523 two-way power line interface
Power line
7
DMC16207 LCD
Data (4) control (3)
4
Assigned house code
Local temperature alternating between ºF and ºC
Controller requests and unit responses

Figure 1—*The PIC16C73A reads local temperature from the DS1820 and responds to controller requests via the TW-523. The optional LCD displays local temperature, controller requests, and sensor responses.*

These spare TW-523 modules enabled me to further my PIC experimentation.

PUTTING IT ALL TOGETHER

I now had to decide what to do with an X-10 interface. I certainly could make use of some sort of motion-sensing device. After some thought, I decided that a temperature sensing device would be a logical extension to my first project.

I also had a real need for such a device. My daughter's bedroom on the second floor tends to get hot during sunny summer days. I had been using the Homebase controller to turn on a fan on summer afternoons.

Most of the time this solution was adequate. On occasion, however, when we had a cool, rainy day (not uncommon in central New York), the bedroom would get too cool.

The obvious solution was to turn the fan on in the afternoon only if the temperature was above a certain level. However, I'd also like to be able to manually override the fan control when I'm at home.

The more I thought about the requirements for this fan controller, it became obvious that an autonomous device didn't make sense. What I needed was a "dumb" sensor device, which provided input to the Homebase controller.

My first thought was to implement a remote temperature-sensing device that would send an X-10 command to the Homebase controller whenever the temperature transitioned through a predefined threshold.

There were two problems with this approach. First, the threshold would have to be set in the remote device. Second, I already had enough X-10 transmission devices and adding one (or more) additional device(s) that transmitted on their own would only increase the probability of a collision.

What I finally decided on was a device that would speak only when spoken to. The Homebase controller sends a query to the remote temperature sensor. This query specifies the temperature threshold. The remote temperature sensor then, and only then, responds (transmits) to indicate if the current temperature is below, at, or above the specified threshold.

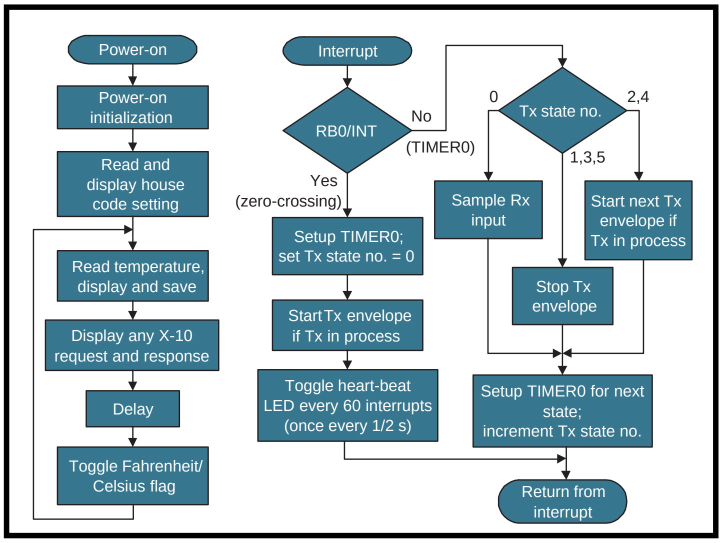
The sensor is programmed for a single house code and responds to queries from the Homebase controller. These queries consist of two consecutive X-10 unit On commands.

The sensor decodes these On commands to determine the temperature threshold value to use for the comparison.

The decoded value is compared to the actual sensed temperature. The sensor responds with a unit On command if the temperature is greater than or equal to the queried temperature value, or with an Off if not. The homebase controller then takes the appropriate action based on the response received from the sensor.

An example of the protocol is illustrated in Table 1. The controller sends a 75°F query and the temperature sensor responds with a unit On command to indicate that the current temperature is equal to or greater than 75°F.

Note that an On command consists of two X-10 transmissions. Each transmission contains a House Code and a Key or Function Code. The Key Code of the first transmission identifies the unit (1 through 16) and the Function Code of the second transmission identifies the Command Code (On in this case).



Power-on
Power-on initialization
Read and display house code setting
Read temperature, display and save
Display any X-10 request and response
Delay
Toggle Fahrenheit/ Celsius flag
Interrupt
RB0/INT
No
(TIMER0)
Yes (zero-crossing)
Setup TIMER0; set Tx state no. = 0
Start Tx envelope if Tx in process
Toggle heart-beat LED every 60 interrupts (once every 1/2 s)
Tx state no.
0
2,4
1,3,5
Sample Rx input
Start next Tx envelope if Tx in process
Stop Tx envelope
Setup TIMER0 for next state; increment Tx state no.
Return from interrupt

Figure 2—*These are the two main elements of the temperature sensor software. The foreground loop on the left manages the DS1820 and the LCD. The background interrupt-driven task on the right receives and transmits X-10 commands via the TW-523.*

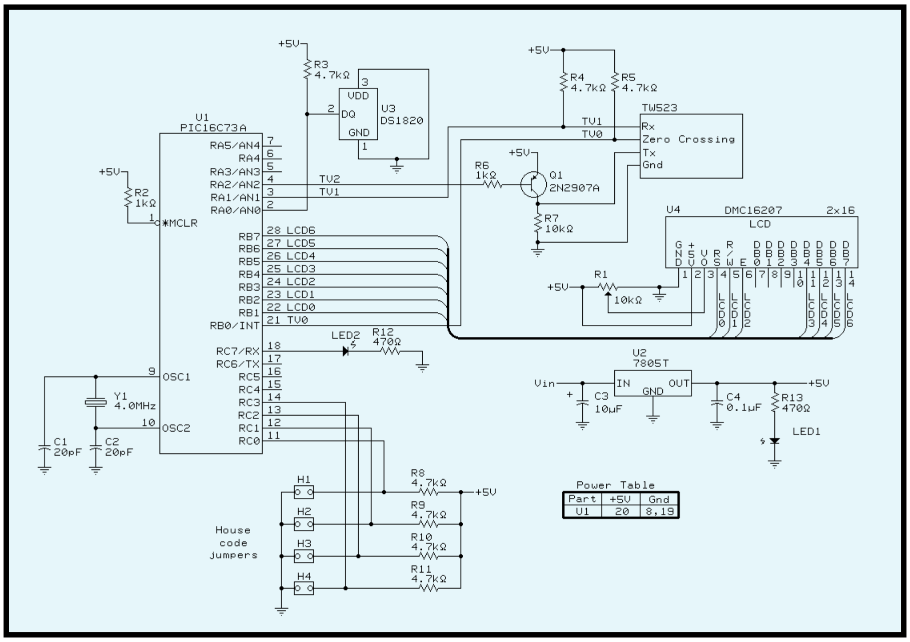


Figure 3—*The PIC16C73A is wired directly to the DS1820, LCD, and two of the three TW-523 signals. The TW-523 Tx input is controlled indirectly through Q1.*

OVERVIEW OF THE SENSOR

Refer to the block diagram of the X-10 temperature sensor in Figure 1. The PIC16C73A microcontroller is central to the sensor and controls its other elements. The PIC's on-chip program memory contains the X-10 temperature sensor software. There are two main elements of the software—a background component and a foreground component (see Figure 2).

The main foreground loop is entered on powerup after completing initialization. In this main loop, the temperature is read from the DS1820 digital thermometer once every 2 s and displayed on the LCD. The temperature display alternates between Fahrenheit and Celsius. The temperature is also saved for use by the background component.

The PIC's interrupt drives the background component, which controls the X-10 input and output interface via the TW-523 two-way power line interface module. There are two interrupt sources: an external interrupt generated from the TW-523 zero-crossing signal and an internal interrupt generated from PIC's Timer0.

The zero-crossing signal is used to synchronize the sampling of the TW-523 Rx output and control of the TW-523 Tx input. The software sets up the PIC's internal Timer0 to create a sequence of precise internal interrupts synchronized with the zero-crossing signal for this purpose.

The background component of the software monitors the TW-523 Rx output. X-10 transmissions are received bit-by-bit and reassembled. When a query request is recognized, it is compared to the temperature reading that was saved in the main foreground loop. The appropriate response is generated and subsequently transmitted bit-by-bit through control of the TW-523 Tx input (see Figure 5).

THE MICROCONTROLLER

When I started the X-10 temperature sensor, I had two different PIC microcontrollers on-hand—the '16C84 and the '16C73A (see Figure 3). The only choice was the '16C73A because the '16C84 had insufficient I/O.

The software for the X-10 temperature sensor was developed using Microchip's MPLAB Integrated Development Environment (IDE) in combination with Microchip's PICSTART Plus programmer. The MPLAB IDE software, information on the PICSTART Plus, and datasheets for the PIC microcontrollers are available from Microchip's web site.

THE LCD

The LCD is an Optrex DMC16207 2-line by 16-character display. The device operates with either a 4- or 8-bit bidirectional parallel data interface and three control lines. The X-10 temperature sensor initializes the LCD to its 4-bit mode. Only seven microcontroller I/O lines are required in 4-bit mode instead of 11, which would be required in 8-bit mode. An external 10 kΩ potentiometer controls LCD contrast.

The LCD accepts 8-bit ASCII data and control characters. In 4-bit mode, two consecutive output operations are necessary to transfer each ASCII or control character to the display. The four data lines are bidirectional and are also used to read busy status from the LCD.

The LCD is not necessary to the operation of the X-10 temperature sensor. It can be eliminated with no other modification to the hardware or software.

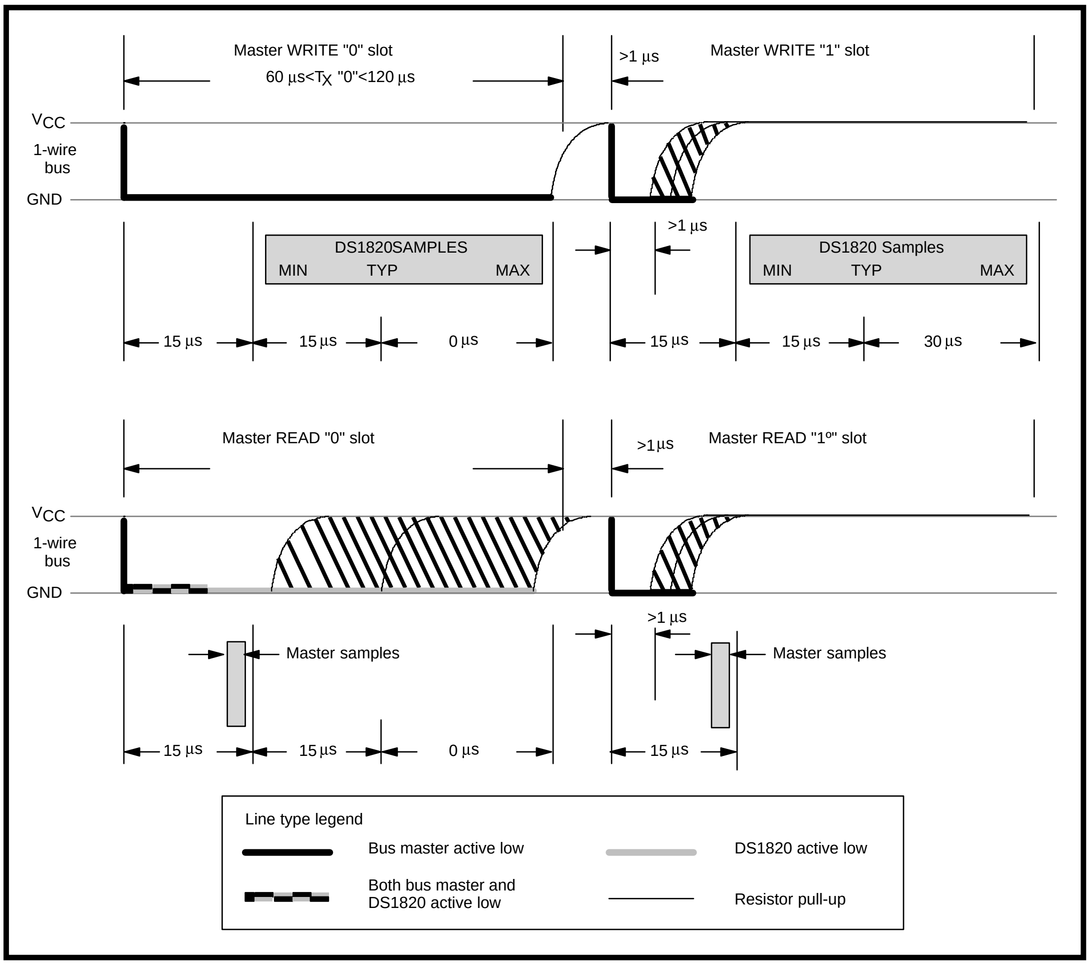


Figure 4— *Command bytes are written to and data bytes are read from the DS1820 a bit at a time. Each bit read or write slot takes 60 µs with at least 1 µs between slots.*

When present, the LCD displays the selected house code at power-on, current local temperature (alternating between Fahrenheit and Celsius), received requests, and transmitted responses.

The local temperature is displayed on the first line of the LCD and the second line is displayed on the selected House Code or the X-10 controller query and transmitted responses.

The selected House Code is displayed for about 60 s following power-on. A query received from the X-10 controller and the response transmitted by the X-10 temperature sensor are displayed for about 120 s. The second line is blanked after the specified time period has elapsed. The LCD is controlled by the foreground software component.

The LCD was also invaluable during debug of the X-10 temperature sensor software. I used the second line of the display to output information to help troubleshoot problems.

TEMPERATURE SENSOR

The DS1820 1-Wire digital thermometer from Dallas Semiconductor is available in both a PR35 (3-pin) and a 16-pin SSOP package. The DS1820 provides a 9-bit digital value that represents the device's temperature in 0.5°C increments over a –55°C to +125°C range.

I used the PR35 package for the X-10 temperature sensor. There are three connections with either package: power, ground, and data in/out. An unusual feature of the device is that the power connection is optional. The power and ground pins can be tied together to operate the device in its parasite power mode.

Controller
- Transmit 75°F query:
- Unit A, Key Code 7
- Unit A, ON Command
- Unit A, Key Code 5
- Unit A, ON Command

Temperature sensor
- Compare current temperature to 75°F
- Transmit response:
- Unit A, Key Code 16
- Unit A, ON Command

Controller
- Take appropriate action for temperature at or above 75°F.

Table 1— *In this command protocol example, the X-10 controller sends a 75° query to the temperature sensor. The sensor responds to indicate that the temperature is at or above this value.*

In this mode, which I use in the X-10 temperature sensor, the DS1820 steals power from the data in/out pin when it's high. The data in/out line is wired to the 16C73A Port A bit 0 (RA0) pin.

The DS1820 temperature conversion cycle requires 500 ms. During this time, up to 1 mA is required and the data in/out pin must be held high by the PIC. The 4.7-kΩ pull-up resistor, used when reading from the DS1820, will not supply sufficient current during the conversion. The X-10 temperature sensor software performs the DS1820 read temperature sequence shown in Table 2.

The sequence begins with a device reset performed by pulling the data in/out line low for 720 µs (minimum 480 µs, maximum 960 µs). The DS1820 responds with a "device present" indication 15 to 60 µs following the release of the data in/out line. The DS1820 pulls the data in/out line low for 60 to 240 µs.

The X-10 temperature sensor software checks for the "device present" response. If no device is detected, a software flag is set to indicate that a temperature reading is not available, and the remainder of the read temperature sequence is skipped.

The sensor uses three of the six DS1820 commands. Each command consists of 8-bits, which are written serially a bit at a time. The bit write timing is illustrated in Figure 4. To write a one, the PIC drives the data in/out line low for at least 1 µs and then drives the line high.

The DS1820 samples the data in/out line between 15 and 45 µs after the PIC first drives it low. To write a zero, the PIC drives the data in/out line low and keeps it low for at least 60 µs. The PIC drives the data in/out line high for at least 1 µs between bits.

Each DS1820 has a unique serial number in built-in ROM that enables several devices to be wired together on the same 1-wire bus. A `match ROM` command identifies the serial number of the selected device.

The X-10 temperature sensor uses the DS1820 in a single-drop configuration. The `skip ROM` command is used to select the device in a single-drop configuration. It works like the `match ROM` command without needing to provide the serial number.

The `convert T` command instructs the DS1820 to begin a temperature conversion cycle. The conversion requires a maximum of 500 ms during which the data in/out pin must be held high when using the parasite power mode.

The temperature conversion is read from the DS1820's scratchpad memory by the last four steps in the sequence. The DS1820 sends one CRC byte following the eight bytes of scratchpad memory. This byte can be used to validate the data transfer.

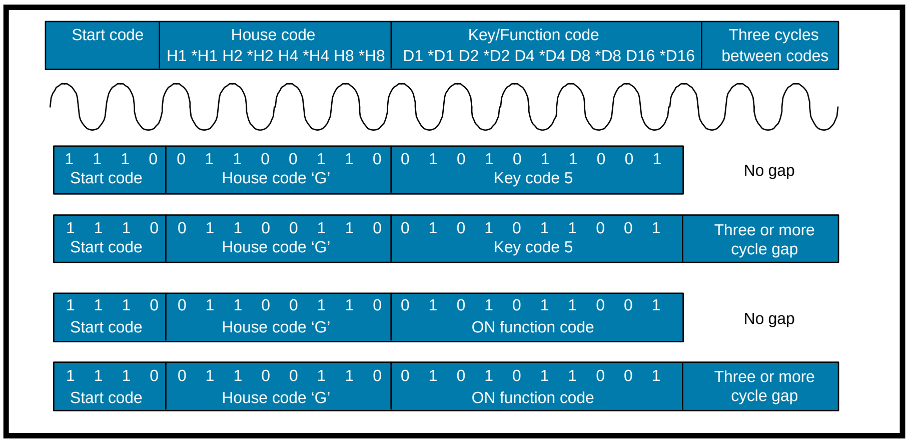


Figure 5—*Transmission of each X-10 command requires 11 power line cycles. Each command is transmitted twice with three or more cycles between pairs.*

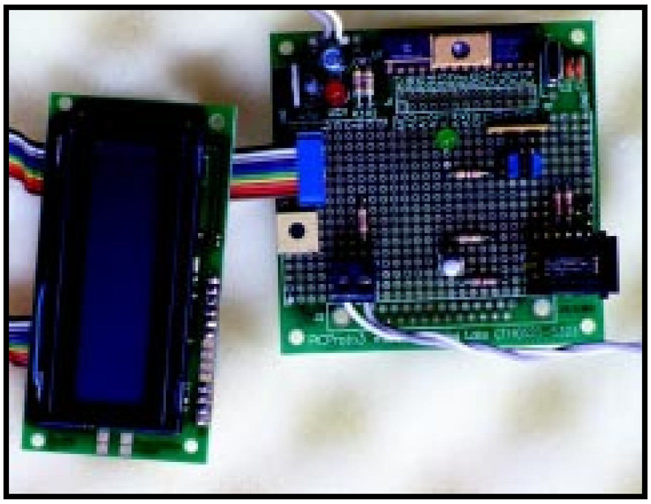


Photo 1—*All components except the LCD, DS1820 digital thermometer, and TW-523 two-way power line interface are mounted on the printed circuit board. The TW-523 cable plugs into the modular jack at lower right. The DS1820 is connected via the twisted pair running from the terminal block at the lower left.*

The sensor software reads these nine bytes, one bit at a time. The software saves the first two bytes, which contain the 9-bit temperature value. The remaining six bytes and the CRC value are discarded. The DS1820 datasheet further details the reset, read, and write cycles, and has information pertaining to additional features, including the content of the remaining scratchpad memory locations.

The bit-read timing is also illustrated in Figure 4. To read each bit, the PIC first drives the data in/out line low for at least 1 µs and then puts the RA0 pin in the high-impedance mode.

The DS1820 will drive the data in/out line low to output a zero or put it in a high-impedance mode to output a one. The pull-up resistor causes the data in/out line to go high when it's in the high-impedance mode. The PIC samples RA0 15 µs from the time it drives the line low.

POWER LINE INTERFACE

The TW-523 interface is used to send and receive X-10 codes via the AC power line. A technical note is available online from X-10. For those of you who are long-time subscribers to *Circuit Cellar*, the X-10 protocol and theTW-523 module were described by Ken Davidson in issues 3 and 5.

An X-10 command consists of a start code, a House Code, and a Key or Function Code. Transmission of each command requires 11 power line cycles: two for the Start Code, four for the House Code, and five for the Key or Function Code.

Each command is transmitted twice with at least three power line cycles between pairs. The four House Code bits and the five Key or Function Code bits are transmitted in true and complement form on alternating half cycles of the power line.

Here's an example. To turn on unit 5 of House Code G requires the following X-10 commands:

Start Code (1110), House Code G (0101), Key Code 5 (00010)
Start Code (1110), House Code G (0101), ON Function Code (00101)

Figure 5 illustrates the transmission of these two X-10 commands. The House Code, Key and Function Code encoding is defined in the X-10 technical note.

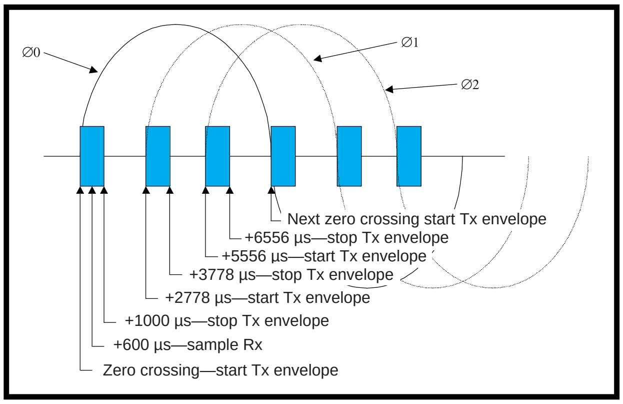
ZERO CROSSING DETECT

The X-10 transmission must be synchronized with zero crossing on each half cycle of the power line. This is the point when the AC voltage goes from positive to negative or from negative to positive. The TW-523 provides a zero-crossing output. This output is a 60-Hz square wave and is connected to the 16C73A RB0/INT pin.

The X-10 sensor software configures the RB0/INT pin to generate an interrupt. RB0/INT can be configured to select either the rising edge or the falling edge. An interrupt on both edges is desirable and is achieved by toggling the edge select (INTEDG bit in the '16C73A OPTION register) just after each zero-crossing interrupt.

The software uses the '16C73A Timer0 in combination with the zero-crossing interrupt to provide the precise timing necessary for sampling of incoming and gating of outgoing X-10 commands. This timing sequence is illustrated in Figure 6 and described in more detail in the following sections.

The zero-crossing interrupt is also used to blink the heartbeat LED (LED2) at a 1-Hz rate, which provides a warm fuzzy indication that the temperature sensor and TW-523 module are alive and well.



Ø0
Ø1
Ø2
Next zero crossing start Tx envelope
+6556 µs—stop Tx envelope
+5556 µs—start Tx envelope
+3778 µs—stop Tx envelope
+2778 µs—start Tx envelope
+1000 µs—stop Tx envelope
+600 µs—sample Rx
Zero crossing—start Tx envelope

Figure 6—*Transmission of each bit of an X-10 command starts at the zero crossing and lasts for 1000 µs. There are three transmissions of each bit, one for each phase of a three-phase setup. Received data is sampled at 600 µs after zero crossing.*

RECEIVING REQUESTS

Each X-10 command is transmitted twice. The TW-523 only provides the second of the two transmissions via its Rx output. The Rx output is valid between 500 and 700 µs after zero crossing. The software sets up Timer0 to generate six interrupts after each zero crossing. The first Timer0 interrupt occurs at 600 µs after zero crossing and the Rx output is sampled at this time.

Remember that it takes 11 power line cycles or 22 zero-crossing events for a complete X-10 transmission. Four samples are required for the Start Code, eight samples for the House Code, and 10 for the Key or Function Code.

The temperature sensor looks for a consistent X-10 transmission which consists of:

- valid start code: 1 1 1 0
- four House Code bits in true and complement form
- five Key or Function Code bits in true or complement form

The incoming X-10 command and any earlier received commands are discarded if any inconsistencies are detected and the software starts over looking for a valid start code.

PROCESSING REQUESTS

The X-10 sensor recognizes a request when it receives two consecutive On commands for the selected House Code. Remember from the example in Table 1 that one On command requires two X-10 transmissions. The first provides the Key Code and the second provides the Function Code (On in this case). Combining the two Key Codes forms the query value. In the example of Table 1, the Key Codes of 7 and 5 represent a query value of 75°F.

The query value is compared to the last local temperature read from the DS1820, if any. If the local temperature is greater than or equal to the query value, a response of Unit 16 On is created. Otherwise, a response of Unit 16 Off is created. An internal software flag is set indicating that a response is ready to be transmitted.

If the sensor detects an error during DS1820 communication, all queries are ignored and no response is generated or transmitted.

TRANSMITTING RESPONSES

As in receiving X-10 commands, transmission must be synchronized to the zero crossing event. The TW-523 Tx input controls when a high-frequency (120 KHz) carrier is superimposed onto the power line. A “1” is represented by a 120-KHz burst and a “0” is represented by no burst. The TW-523 Tx input is driven by the 16C73A RA2 output through Q1.

For single-phase residential power, this burst begins at zero crossing and lasts for 1 ms. A three-phase setup requires three 1-ms bursts during each half cycle. The temperature sensor is designed to work in a three-phase setup.

The first burst begins at zero crossing. Subsequent Timer0 interrupts control when that burst ends or when the remaining two bursts begin and end. See Figure 6 for the exact timing.

The TW-523 doesn’t handle the duplicate transmissions on outgoing commands as it does for incoming transmissions. The X-10 software must handle transmitting each command twice with a gap of at least three cycles between pairs.

I use 10 half cycles between groups so it takes a total of 108 half cycles (zero crossing interrupts) to transmit a complete response. Breaking down the 108 figure, you get 44 half cycles per group of two (11 cycles × 2 zero crossings/cycle × 2 transmissions), 10 half cycle gaps between codes, × 2 codes (Unit/Key Code and Unit/Function Code).

SYNCHRONIZATION OF TEMPERATURE MONITORING

Remember from the high-level software flow in Figure 2 that DS1820 communication takes place in the main foreground loop and the interrupt-driven background loop deals with the TW-523. The DS1820 communication requires precise timing. A zero crossing or Timer0 interrupt during DS1820 communication would disturb this timing.

One way to deal with this would be to disable interrupts during the periods of time-critical DS1820 communication. However, this technique disturbs the precise timing required for the TW-523 interface.

The solution is to synchronize the time-critical DS1820 code with the zero crossing and Timer0 interrupts. Note in Figure 6 that there are three intervals in each half cycle between transmission bursts where no interrupts occur. The intervals are 1.778 ms long—more than enough time to deal with time-critical DS1820 communication.

An interrupt synchronization subroutine is called just before all time-critical DS1820 code. This subroutine sets an internal flag then waits for that flag to be reset.

The ’16C73A interrupt handler resets this flag at the start of each 1.778-ms interval. The interrupt synchronization subroutine has a 3-ms timeout for protection against a situation where the zero crossing interrupt has been lost.

PROTOTYPE CONSTRUCTION

I built the X-10 temperature sensor on a microEngineering Labs PICProto3 prototyping board. The main components of the sensor are mounted on the PICProto3. This PCB provides traces and pads for the standard PIC components such as power supply circuitry, microcontroller and the oscillator circuitry.

The remaining temperature sensor components (with the exception of the LCD, digital thermometer, and TW-523 module) are mounted in the prototyping area of the PICProto3. Direct point-to-point wiring using 26-AWG solid wire was implemented.

The LCD is connected to the PICProto3 by a 10-wire ribbon cable and header connector. The digital thermometer is wired to a twisted pair cable connected via a terminal block. The TW-523 is connected by a modular cable, which plugs into a jack mounted on the PICProto3.

Reset device, check device present response
Send `skip ROM` command
Send `convert T` command
Delay 500 milliseconds (while holding data in/out high)
Reset device, check device present response
Send `skip ROM` command
Send `read scratchpad` command
Read scratch pad (8-bytes plus one CRC byte)

Table 2—*The temperature sensor uses 3 of the 6 DS1820 commands to read the temperature. The skip ROM command is used to select the device in a single-device configuration.*

APPLICATIONS

The X-10 temperature sensor was designed to aid in control of a house fan in a remote location. This device can also be used in other applications where it's desirable to determine temperature in relation to a threshold. Applications that come to mind are:

- monitoring freezer temperature to generate an alarm if the temperature is too high
- monitoring plumbing temperature to generate an alarm and/or turn on a heater if the temperature gets too low

For applications where it's desirable to obtain a remote temperature, not just the relationship to a given threshold, the software could be easily modified to transmit the local temperature upon request. The temperature could be encoded using standard X-10 Key and Function Codes. The X-10 controller could then, for example, record hourly temperatures and daily high and low temperatures, as desired.

Although the X-10 sensor provides feedback for a closed-loop environment, care should be exercised as with any X-10 application. Because X-10 control is not 100% reliable, it shouldn't be employed in a situation where safety might be compromised. For example, an electric heater should never be controlled by X-10 alone if it could cause a fire if left on too long.

COOLING DOWN

The temperature sensor provides remote temperature measurement without wires, using only standard X-10 Key and Function Codes. The Extended Code/Data capabilities of the X-10 standard are not required. In addition, the device uses only a single X-10 House Code.

Many other possibilities exist for this temperature sensor. The protocol could be expanded to share a single House Code among multiple sensors. The LCD could be left out to reduce the number of I/O required and permit the sensor to be used with an inexpensive microcontroller such as a PIC16C84. This technique could even be applied to motion detectors, light-level sensors, or other types of sensors.

Don Blake has 30 years experience in avionics embedded systems and diagnostic software development. He is a senior programmer at Lockheed Martin Federal Systems in Owego, NY. You may reach him at donblake@worldnet.att.net.

SOFTWARE

The complete source code for this project is available for download via the *Circuit Cellar* web site.

REFERENCES

David Tait's 16C84 programmer plans, www.man.ac.uk/~mbhstdj/files/pic84v05.zip

LCD FAQ, ftp.ee.ualberta.ca /pub/cookbook/faq/lcd.doc

TW-523 manual, www.x10.com/support/support_manuals.htm

K. Davidson, "Power-Line-Based Computer Control", *Circuit Cellar* 3, May/June 1998.

K. Davidson, "The X-10 TW523 Two-Way Power Line Interface" *Circuit Cellar* 5, Sept/Oct 1988.

SOURCES

PIC16C84
Microchip Technology, Inc.
(888) 628-6247
(480) 786-7200
Fax: (480) 899-9210
www.microchip.com

DMC16207
Optrex
(313) 471-6220
Fax: (313) 471-4767

DS-1820
Dallas Semiconductor
(972) 371-4448
Fax: (972) 371-3715
www.dalsemi.com

TW-253
X-10 USA
(800) 675-3044
(201) 784-9700
Fax: (201) 784-9464
www.x10.com

PICProto3
microEngineering Labs, Inc.
(719) 520-5323
Fax: (719) 520-1867
www.melabs.com

PIC development tools
Dontronics
(613) 9338-6286
Fax: (613) 9338-2935
www.dontronics.com

FEATURE ARTICLE

Randy Heisch

Thermoelectric Micropower Generation

An Alternative Power

In this project, Randy mixes new technology (a thermoelectric cooler) with some old technology (fire) to create an alternative low-power generator. It may not be the solution to today's increasing power demands, but if things get real dark on January 1, it might come in handy.

You've no doubt heard the term "candlepower" used to describe luminous intensity. In this article, however, you may discover that the term has a potentially new use. My project? An electrical power generator that uses an ordinary candle as the energy source.

THERMOELECTRICITY

Thomas Johann Seebeck (1770–1831) discovered in 1821 that if the junction between heterogeneous conductors is heated, an electromagnetic field (EMF) is developed across the junction and a current is caused to flow through the conductors. Twisting the ends of two dissimilar metals and heating this junction results in a typically millivolt potential difference across the junction, depending on the temperature. From Seebeck's work, the thermocouple or thermoelectric generator (TEG) was born.

Shortly thereafter, in 1834, Jean Charles Athanase Peltier (1785–1831) discovered the opposite of the Seebeck effect: a current passed through the junctions of dissimilar conductors results in heat being pumped from one junction to the other (i.e., one junction gets colder and the other gets hotter).

Using modern semiconductor materials, highly reliable and comparatively useful thermoelectric coolers (TECs) are possible, with heat-pumping capacities of up to 100 W or more.

There are several manufacturers as well as applications of TECs and TEGs. TECs are employed in cigarette lighter–powered ice chests but are probably more routinely used to cool electronic components.

TECs achieve an approximate maximum of 68°C temperature difference between the hot and cold sides, or heat loads of up to 100+ W. Perhaps most importantly, TECs have no moving parts (short of the muffin fan often used to cool the hot side).

TECs are constructed using soldered thermocouple junctions, which limits the maximum working junction temperature to somewhere below the melting point of solder. TEGs, by contrast, are designed to exploit the Seebeck effect for the purpose of measuring higher temperatures and to generate electrical power from a variety of heat sources.

To achieve higher levels of conversion efficiency and power output, TEG energy sources must provide higher temperature differentials. One manufacturer specs a 5% efficiency at a 200°C temperature differential.

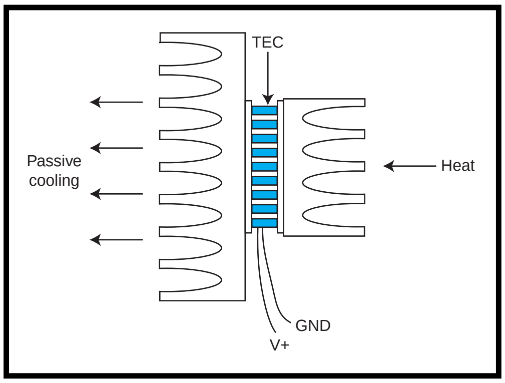


Figure 1—*A typical thermoelectric cooling module generates power when operated in reverse. This top-down view shows heat applied (arbitrary) to the hot side as the cool side passively dissipates heat through a larger finned heatsink, resulting in a micropower thermoelectric generator.*

Photo 1— *The candle-powered radio shown here uses a reverse-mode thermoelectric cooler to convert heat into electrical energy. Power is boosted to a usable level by a step-up switching regulator, which powers the AM/FM radio.*

These TEGs are constructed using a hot-press thermocouple junction attach method, which allows for these higher operating temperatures. But apparently, the TEG manufacturing process is substantially more complex: the modules cost upwards of five times more than their TEC counterparts.

Fortunately, as I discovered while implementing this prototype, the less-costly TEC can be operated in reverse as a TEG, at necessarily lower temperature deltas. The result is much lower efficiency and power output. However, if high power isn't required, the TEC will generate power from heat sources as simple and convenient as an ordinary candle.

CONVERSION EFFICIENCY

Power-conversion efficiency is not among the strengths of these devices (although compared to some solar cells with as little as 4% efficiency, one might not consider this a weakness). As indicated above, with a 200°C temperature differential maintained across the hot and cold sides, a typical TEG might effectively convert 5% of the input heat into electricity.

Under these conditions, a single module can produce 1.6 V at 8 A with a matched load, or almost 13 W, but it requires the significant temperature difference between the hot and cold sides. However, in situations where this heat is in the form of exhaust and lost anyway, harvesting even 5% of the waste is better than nothing, especially given the reliability and simplicity of these modules.

THE PROTOTYPE

The prototype thermal power generator, shown in Figure 1, uses a single 1.5 in. sq. TEC (because of its lower cost and because that's what I had lying around the bench). I fitted a heatsink to the hot side of the TEG and attached the cold side to a 4-in. sq. heatsink.

The cold side is passively cooled (i.e., no external fan) through the surface area of the finned heatsink. I fashioned a makeshift shroud from heavy aluminum foil for the hot side of the heatsink to confine the heat rising from the candle to the proximity of the heatsink.

Using this configuration and an ordinary household candle (see Photo 1), I measured an open-circuit voltage of 1.1 V. At near optimum load, the module produces 0.5 V at 125 mA or about 63 mW of electrical power. And that's with only a 20°F temperature delta across the TEC!

With an LM34 temperature sensor attached to the heatsink, I measured a maximum hot-side temperature of 150°F and 130°F on the cold side. It's reasonable, I think, to expect a lower conversion efficiency given this relatively small temperature differential. In fact, I expected even less efficiency, so I was amazed to find this much power available at such a low temperature difference.

Using a fairly crude power-dissipation test bed, I measured about 18 W of power liberated by the candle. Assuming this measurement is correct, this TEG configuration and the relatively small temperature differential provided by the candle and the passive cooling result in only a 0.35% energy-conversion efficiency. But, putting the efficiency factor aside, note that 0.5 V at 125 mA is enough power to breathe life into a variety of low-power devices.

HIGH-EFFICIENCY SWITCHING POWER SUPPLY

Because a 0.5-V supply voltage is usually inadequate for most devices, I added a low input, 3–5-V step-up switching regulator (see Figure 2). This regulator (the MAX756) specifies an efficiency of about 82% at 1.2-V input at a 10-mA load.

With the TEG providing less than 1.1 V loaded, the MAX756 is less efficient. I measured 4.99 V at 9 mA out of the switcher, or about a 71% efficiency.

An alternative would be to include several TEG modules connected in series to increase the loaded voltage, to either a usable range or a range where the switcher is more efficient. For example, 3.3 V at a 100-mA load for the MAX756 results in almost 90% efficiency converting to a regulated 5-V output.

But this design results in a larger configuration, and a single candle as the heat source would eventually prove inadequate. For this sample implementation, 5 V at 9 mA is sufficient to demonstrate at least a couple useful applications.

CANDLE POWER

I connected the MAX756 5-V output to the battery terminals of an AM/FM pocket radio. After a warm-up period of about 2–3 min., the reverse-mode TECs output voltage climbed enough to allow the switcher to kick in and the radio was brought to life.

Although the TEG described above doesn't produce enough power to run your average laptop (well, maybe your Palm Pilot, but I'll let someone else try it on theirs first), the module is capable of powering a variety of microcontroller circuits.

Figure 3 shows a PIC12C672 temperature-monitoring circuit powered from the TEG. The '12C672 requires less than 2 mA at 5 V (at 4 MHz) and features a four-channel 8-bit ADC and DIO (configurable across six I/O pins).

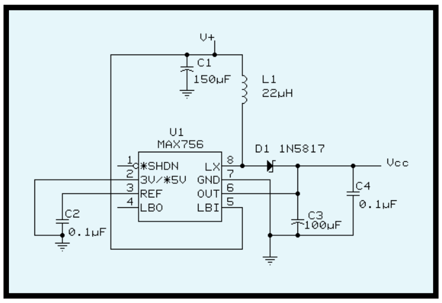


Figure 2—*A voltage of 0.5 V is insufficient for most electronic devices. This step-up switching power supply raises the voltage output from the TEG to a usable 5 V.*

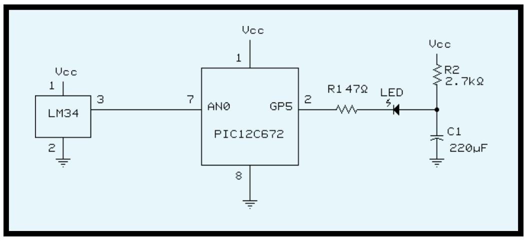


Figure 3—*This example low-power PIC microcontroller circuit requires less than half the power supplied by the TEG described in this article.*

I attached an LM34 (a 90-µA Fahrenheit temperature sensor) to analog input AN0 and general-purpose I/O pin GP5 to an LED circuit. For simplicity, I used the internal 5-V reference for the ADC.

The LED circuit is a simple RC network that slowly charges the capacitor (at less than 2 mA max.) and provides a quick +60-mA discharge (controlled by the PIC) to achieve a momentary bright pulse.

The LED pulses at about 1 Hz if the LM34 temperature rises above 86ºF. At a total maximum power requirement of ~4 mA, this circuit leaves ~5 mA to spare from the TEG.

FIRE IT UP

There probably aren't many circumstances where I'd opt for an open-flame power source for an electronic appliance (especially indoors). But, I can imagine that a campfire-driven power supply might come in handy someday.

Or, if Y2K has you concerned and you have enough combustible materials on hand, you won't have to depend on the power grid to tune in to your favorite doomsday radio station—assuming, of course, that the station has some form of alternative power to drive their transmitter.

One word of caution: the electric light bulb has several advantages over its predecessors, but one of the more important was the elimination of the open flame. Be sure to keep in mind the dangers of (re)introducing external combustion to the workbench. ■

Randy Heisch is a staff engineer in Sun Microsystems' Strategic Applications Engineering group.You may reach him at randy.heisch@ sun.com.

RESOURCES

Introduction to the Thermoelectric Module, http://server1.magi.net/inbthermoelectric/thermo-electric_cooler.htm.

SOURCES

MAX756
Maxim
(408) 737-7600
Fax: (408) 737-7194
www.maxim-ic.com

TEG/TECs
Hi-Z Technology, Inc.
(858) 695-6660
Fax: (858) 695-8870
www.hi-z.com

Marlow Industries, Inc.
(214) 340-4900
Fax: (214) 341-5212
www.marlow.com

PIC12C672
Microchip Technology, Inc.
(480) 786-7200
Fax: (480) 899-9210
www.microchip.com

FEATURE ARTICLE

Lloyd Paul Verhage

Asimov II

Getting a Near-Space Project Off the Ground

Though not as glorious as a space shuttle mission, each flight for the Kansas Near-Space Project is certainly exciting. Prepare for liftoff as Paul goes through the pre-flight checklist for a trip to the stratosphere via weather balloon.

Space may be the final frontier, but it's darn difficult for the amateur to get there. So I've settled for the next best thing.

Knowing that I would soon be teaching high-school electronics and science, I developed an educational project for my future school that involved sending experiments into the stratosphere. I refer to this as launching experiments into near space.

The first incarnation of this educational project was called the Kansas Near-Space Project (KNSP) and was active from November 1996 to July 1999. KNSP launched 19 near-space missions that performed experiments in meteorology, remote sensing, engineering, radio propagation, space science, and life science. In this article I describe my near-space project and tell you how I designed the electronics for our near-space capsules.

I hope that some of you will bring projects like this to other schools across the country. They can be used by K-12 students to enhance their studies, which is what I am attempting now in Idaho with the Idaho Near-Space Project (INSP).

First, a few words of explanation. In real life, I'm not a professional hardware or a software guy, however I do a little bit of both on the side. Professional software writers may cringe at some of my code and hardware folks may be shocked at some of my electronics, but both the code and hardware work well together. It's my opinion that the PCB is probably the better part of the project because it was designed to my specifications by Mr. Steve Kelly at the KSU Physics Electronics Shop (thanks Steve).

THE NEAR-SPACE STACK

The stack for a near-space flight begins with a helium-filled latex weather balloon, weighing either 1200 or 1500 grams. A 30′ length of nylon line (the load line) ties the nozzle of the balloon to the apex of a parachute.

The capsule itself is attached to the shroud lines of this parachute and hangs at the bottom of the stack. Because of limitations imposed by the FAA, our capsule actually consists of two, six-pound boxes. These boxes are constructed of polystyrene foam panels glued together with hot melt glue.

The boxes are covered in an abrasion bag made of spinnaker nylon to protect them when landing. The bag also provides attachment points (dacron loops and jump rings) for the shroud lines and the other box.

Experiments are mounted to square openings cut in three sides of each

Photo 1— *A dash across a field to catch a descending capsule after its 90-mile flight. (Inset) Here is a closer look at the payload during ascent.*

box. Because of these ports, we don't modify our near-space capsule for each flight. This feature makes a KNSP capsule similar to a space shuttle in that the same airframe can be reused for a wide range of experiments and activities.

Power switches for the capsule's various power busses (main, servo, and auxiliary) and a program header for the BS2 are mounted to a panel on the fourth side of each box. The switches and program header enable KNSP crew members to operate a capsule without opening its hatch.

Power for a near-space capsule is provided by military-surplus lithium cells. Each cell provides 3 V and has a capacity of six amp-hours. They have the form factor of a standard "D" cell but half the weight of an alkaline cell. They're also less sensitive to the low temperatures experienced during a flight (we've seen outside air temperature drop as low as –90°F).

Being able to easily mount a variety of experiments to a capsule airframe isn't enough though. It's the near-space controller I designed that gives a KNSP capsule its flexibility. I call these controllers the integrated housekeeping unit (IHU).

THE IHU

The IHU is microcontroller operated, which means necessary changes for any flight are simply a software change away. An IHU performs many functions during a flight, including:

- monitoring the various bus voltages
- monitoring internal temperatures
- commanding the capsule's GPS receiver
- determining current flight status (ascent or descent)
- performing mission experiments
- telemetering capsule status and experimental data to ground stations

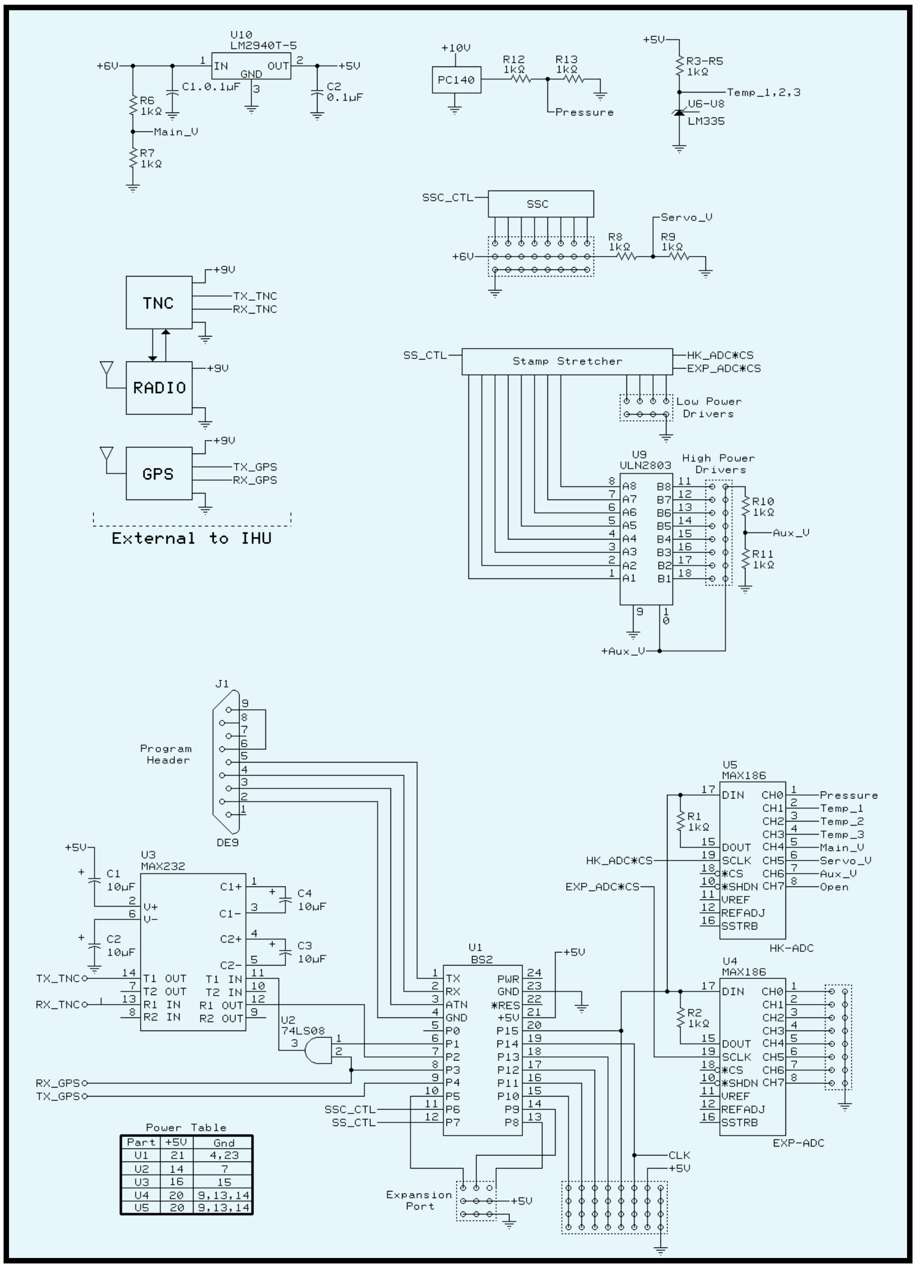
Designing an IHU was my first exposure to microcontrollers. I choose the Basic Stamp II (BS2) from Parallax because it used EEPROM (I didn't know about flash memory) to store code and is not an OTP microcontroller (an important factor when experiments change from flight to flight). Besides, like another columnist in this magazine, I'm a crash and burn kind of guy who makes lots of changes before I'm satisfied.

The BS2 appeared to be an easy-to-program microcontroller with a sufficiently powerful set of commands. I found plenty of app notes to get my design off the ground (see Figure 1).

Most importantly, the price for the BS2 and its programmer was right. For my first use of a microcontroller, I was quite happy with the results. I believe I could eventually design a simple satellite around the BS2.

Steve Kelly designed the IHU PCB using Tango, a PCB CAD program. The board itself was a presensitized, double-sided board purchased from Kepro.

After developing, the board was etched in 100 ml of ammonium persulfate. The board was left covered with resist at this point, protecting the board as I drilled mounting holes. The resist was then removed using a longer soak time. The whole process was clean and pleasant, unlike methods that use a ferric chloride etch. After tinning, it took about two hours to solder components on the board.



Power Table

Part	+5V	Gnd
U1	21	4,23
U2	14	7
U3	16	15
U4	20	9,13,14
U5	20	9,13,14

Figure 1— *Here is a look at all the components that go into making an IHU fit on a single 6" × 6" board. The set of male 0.1" headers below the ULN2803 are for the drivers and expansion ports.*

Power for the entire IHU board is provided by the LM2940T-5, which supplies 5 V at up to 1 A of current. The ’2940 is a low-dropout voltage regulator that I used in place of the more traditional LM7805 because the ’2940 can power the IHU with 6 V rather than with the 7–8 V the ’7805 requires. Capacitors between ground and the input/output leads on the ’2940 provide bypassing and improved voltage stability.

The Scott Edwards SSC allows the BS2 to control up to eight servos using a single serial line. Servos are used to operate experiments such as camera shutters and release mechanisms.

The Scott Edwards Stamp Stretcher doubles the number of I/O pins available to the BS2. Four Stretcher I/O pins are used to operate the IHU’s SPI devices (the ADCs and RTC). Eight of the Stretcher I/O pins are used to operate the ULN2803.

This arrangement enables the IHU to operate experiments for which the BS2 can’t source the necessary voltage or current, such as firing rocket ignitors. The last four pins of the ‘2803 are used as low-power drivers for experiments (they supply 5 V at up to 20 mA).

Many sensors we have flown require the IHU to digitize a signal voltage. The IHU uses two MAX186 ADCs for this purpose. With a maximum input voltage of 4.096 V and a resolution of 12 bits, the MAX186 has a precision of 1/1000 of a volt, though I suspect the board design will not allow this level of accuracy.

The first ADC on the IHU performs housekeeping functions like monitoring bus voltages and capsule temperatures. The voltage dividers on the IHU are used for this ADC. The second ADC is available for student experiments. The BS2 connections to the MAX186s are the CLK, D_{in}, and D_{out} pins. The D_{in} and D_{out} share the DATA_IO pin of the BS2, but the D_{out} pin has a 1-kΩ resistor to protect it.

The PC140 and one of the LM335s are used for meteorologic experiments, allowing us to measure both atmospheric pressure and temperature during a flight. The PC140 requires at least 8 V to operate. This voltage is provided by a voltage doubler on the IHU.

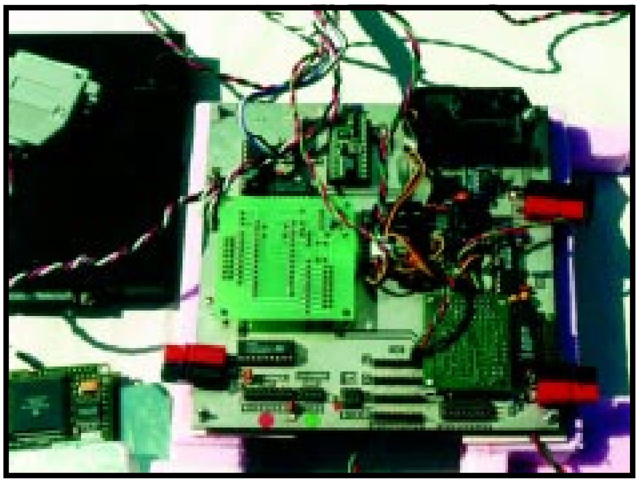


Photo 2—*In this IHU photo, the boards are the SSC and Stretcher. At the top right is the black PC140. The BS2 is at the top center.*

The meteorological ’335 is mounted outside the capsule body where it’s exposed to the air. Of the two other ’335s, one is soldered directly to the IHU and the other is mounted to the end of twisted wires and left inside the capsule. These ’335s enable us to monitor both the IHU and interior capsule temperatures as a part of housekeeping.

Eight pins of the BS2 are reserved as I/O for experiments. Four of them form expansion ports with access to only GND and +5V, and the other four form expansion ports with access to GND, +5V, CLK, and DATA_IO. Experiments using these expansion ports connect to them with .025″ crimp connectors with 0.1″ spacing between centers. These expansion ports act a lot like expansion slots in a computer.

Photo 2 shows the IHU. Half of its design comes from a series of app notes I downloaded from Parallax, and the other half consists of boards prebuilt by Scott Edwards Electronics.

INTERFACING EXTERNAL DEVICES

Without external devices the IHU would, for the most part, be blind and mute. Interfaced to the IHU are a

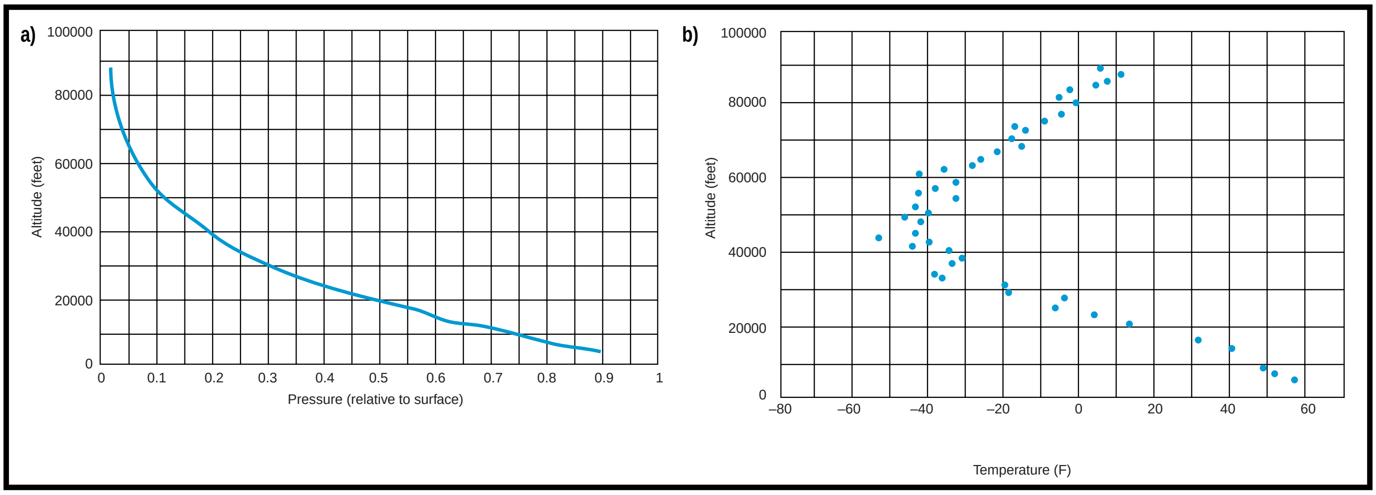


Figure 2a—*This flight reached 87,000 feet and returned pressure data during the flight. You can see that pressure drops approximately logrithmically as a function of altitude.* **b—***Air temperature decreases with altitude until you reach the stratosphere, which occured at about 50,000 feet on this flight. Once in the stratosphere, the air temperature begins to increase with altitude.*

GPS, Packet Radio, and APRS

The National Marine Electronics Association (NMEA) has developed a standard for marine electronics, including GPS receivers. This standard, 0183, Revision 2.0.1, defines how marine electronic devices communicate over a standard RS-232 serial port.

There are seven standard GPS sentences, of which amateur radio operators and KNSP typically use two. All NMEA sentences are sent at 4800 bps, N81 and begin with an ASCII $ (hex 24) and end with a ASCII <CR><LF> (hex 0D and 0A).

The $ is followed by a two-part, five-character address. For the GPS receivers, the first part is a two-letter designator (GP) indicating that GPS data is being sent. The second part is a three-letter designator indicating the type of GPS sentence being sent. The seven types of GPS sentences are:

GGA—GPS fixed data
GLL—geographical position, latitude and longitude
GSA—GPS dilution of precision and active satellites
GSV—GPS satellites in view
RMC—recommended minimum specific GPS/TRANSIT data
VTG—track made good and ground speed
ZDA—time and date

Amateur radio operators and KNSP use the GGA (for the time, latitude, longitude, and altitude data) and the RMC (for heading and speed data) sentences.

Here are two GPS sentences from one of our near-space flights:

```
$GPGGA,143906.00,3758.2473,N,09737.8307,W,1,08,1.0,
   20855.4,M,26.9,M,,*70
$GPRMC,143911.00,A,3758.2505,N,09737.8538,W,14.9,
   278.8,200699,6.0,E*79
```

The format for the GGA is:

```
$GPGGA,hhmmss.ss,ddmm.mmmm,n,dddmm.mmmm,e,q,s
   s,y.y,a.a,z,g.g,z,t.t,iiii*CC<CR><LF>
```

where:

hhmmss.ss is the UTC time of the GPS position fix
ddmm.mmmm,N is the latitude of the GPS position fix
dddmm.mmmm,W is the longitude of the GPS position fix
q is the quality of the GPS fix (1 means there is a fix, but no differential correction)
ss is the number of satellites being used
y.y is the horizontal dilution of precision
a.a,M is the GPS antenna altitude in meters
g.g,M is the geoidal separation in meters
t.t is the age of the deferrential correction data
iiii is the deferential station's ID
*CC is the checksum for the sentence

The format for the RMC is:

```
$GPRMC,hhmmss.ss,a,ddmm.mmmm,n,dddmm.mmmm,w,z.z,y.y,
   ddmmyy,d.d.v*CC<CR><LF>
```

where:

hhmmss.ss is the UTC time of the GPS position fix
a is the status of the GPS data, V means it's valid
ddmm.mmmm,N is the latitude of the GPS position fix
dddmm.mmmm,W is the longitude of the GPS position fix
z.z is the speed over ground in knots
y.y is the heading of the GPS receiver, in reference to true north
ddmmyy is the UTC time of the GPS fix
d.d is the magnetic variation in degrees
v is the sense of the variation, in either east or west
*CC is the checksum for the sentence

This information is from the Motorola OnCore User's Guide, 68P41117UOI, Revision 7.0, 1996.

It's easier for our capsules to use packet radio to telemeter digital data to ground stations than it is for them to use modulated analog signals that must be decoded. However, the cheaper analog system is currently used by the National Weather Service in their radiosondes.

In the early '80s, radio enthusiasts in Tucson, Arizona developed the TNC to handle the work of converting serial data into audio tones that could be sent over a radio. These individuals later became the Tucson Amateur Packet Radio (TAPR) and made these TNCs available to the amateur radio community.

The TNC they developed collects the serial data, breaks it into packets, calculates a CRC for each packet, converts the data and CRC into audio tones, keys the radio, then sends the tones over the radio. Another TNC at the destination performs these steps in reverse to convert the signal back into serial data, which is displayed on a PC or dumb terminal.

Most packet radio, and that used by near space capsules, is sent at 1200 bps over VHF (the 2-m amateur band) or UHF (the 70-cm amateur band). The transmission range for packet radio is limited to just a bit further than line of sight. However, during a balloon flight, line of sight can extend over 400 mi.

Because data is broken into packets, multiple users can use the same frequency, sharing the limited band efficiently by time sharing. Packet radio has a collision detection scheme built into it, so packets can be resent, if necessary.

(continued)

GPS, Packet Radio, and APRS (continued)

The communications protocol used in packet is Amateur X.25, or AX.25. As with other networking protocols, each packet of data contains the sender and receiver's identification, or call sign. In the case of the near-space capsule, we don't include a specific destination call sign, we just send packets to anyone listening.

When packet data contains information from a GPS receiver, the program Automatic Packet Reporting System (APRS) can display the GPS's location graphically. APRS was developed by Bob Bruninga (WB4APR) of the Naval Academy. Local amateurs can develop maps of areas of interest or use the maps available with programs like Delorme's Street Atlas.

Not only is the location of the transmitting GPS receiver displayed, but telemetry like weather conditions can be included. Currently Bob is working on a way to interface slow-scan television to APRS so that live images may be included as part of the telemetry.

Most of our chase vehicles have APRS running on a laptop. By carrying their own GPS and transmitting their location, drivers have a real-time map indicating their location along with the locations of the capsule and other chase teams. It also enables chase teams to send short messages to each other during a chase, keeping everyone up to date with the current status of the chase and recovery. With packet radio and APRS, chasing a balloon is like shooting goldfish in an aquarium—we just follow the balloon on the roads displayed.

terminal node controller (TNC) and GPS. A TNC is a modem that uses amateur radio instead of phone lines for its data link. The capsule's TNC is programmed before flight to operate in transparent mode (versus command mode). In transparent mode the TNC telemeters all data it receives, there's no attempt to send data to a specific recipient on the ground.

The TNC takes some of the workload off the BS2 by calculating the CRC for the data and determining the proper tones to send over the radio (if the BS2 performed these functions there would be no EEPROM left to fly the mission). Three seconds after the TNC finishes receiving data it keys its radio and sends out the proper tones. This particular TNC requires data to be sent at RS–232-level signals, hence the MAX233 converter on the IHU.

The MAX233 acts as a TTL/RS-232 transceiver. Its sole function is to interface the TNC to the rest of the board (only half the chip is used). I used the MAX233 in place of the MAX232 because the '233 doesn't need external capacitors.

Writing this article has made me double check the IHU design and the logic used to operate it. Signals to the AND gate are inverted logic. The MAX233 then sees a single inverted TTL level signal.

According to the datasheet, the MAX233 inverts the signals again, so the TNC will see a RS-232 level signal, but now in true, rather than inverted logic.

For the amateur radio operators out there, the TNC I am currently using is the Kantronics KPC3 with V.6 EEPROM. I chose this particular TNC because of its popularity, price, and because at the time I didn't know the difference between TTL and RS-232 levels (boy, did I learn fast).

The GPS receiver we use is a Motorola VP OnCore (it's just a GPS engine with no display). I chose this GPS because it will continue to output data once above 60,000′. Some commercially available GPS receivers will not provide location data above 60,000′ because of an overly stringent interpretation of DoD requirements.

To prevent commercial GPS receivers from being used to guide a terrorist missile, the DoD prohibits civilian GPS receivers from providing position data above 60,000′ when traveling at speeds in excess of 999 knots. Many GPS receivers treat this rule as an OR function, rather than the AND function that it really is.

Using software sent to me by Motorola, I programmed the OnCore to withhold transmitting data until specifically commanded. During a flight, the BS2 determines when the OnCore should send data and which sentences are to be sent. Luckily, the OnCore accepts TTL-level signals so there's no need to interface a MAX233 between the OnCore and the BS2.

Data from the OnCore is sent to two locations on the IHU. One PCB trace leads to one input of the AND gate (after the other input, TNC_Out, is set high) where OnCore and BS2 data are mixed and then fed into the MAX233. Because the BS2 controls the times the OnCore sends data, there are no signal collisions at the AND gate. OnCore data is also sent straight to the BS2 and parsed to determine the current altitude.

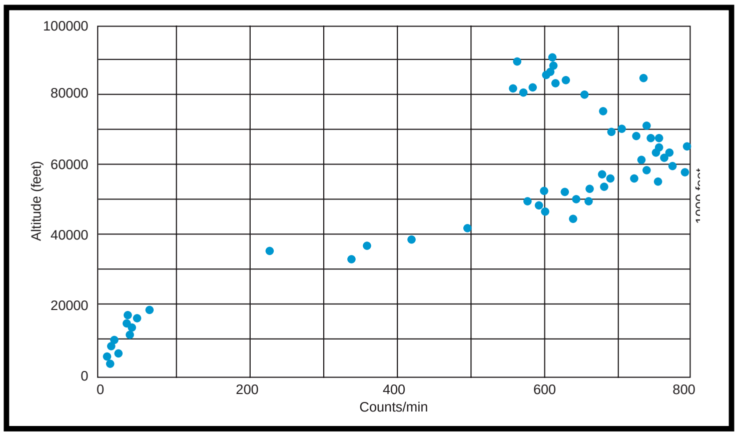


Figure 3— *Cosmic ray counts increase with altitude until we reach 62,000 feet. At this altitude we are getting above the tertiary cosmic rays and into primaries and secondaries.*

THE BS2'S POINT OF VIEW

The flight code I have developed treats a flight as consisting of three modes—preflight, flight, and descent modes. There are five, 1-min. loops in the preflight (wait while we fill the balloon) portion of the flight code.

Listing 1—*If the BS2 doesn't receive data from the GPS within 15 seconds, it jumps to the BadAlt subroutine.*

```
BadAlt:                             ;GPS did not respond in 15 sec
serout TNC_Out,Ext_Comm,
  [">NOGPS", cr]                    ;tell the world
pause 3000                          ;wait for the TNC
GPS_Lock = 0                        ;no GPS signal
End_GPS:
return
```

Listing 2—*This subroutine counts the number of 5-V pulses given off by the RM-60 for 10 s. Each pulse is a detection of a cosmic ray. We're counting atoms emitted by distant stars.*

```
TNC_Out                  con 1        ;data to TNC
Ext_Comm                 con 188      ;baud for TNC/GPS 4800 baud,
true
GM_Count                 var byte     ;number of RM-60 pulses
GM_Ch                    con 12       ;data from GM
'*********************
'* Geiger Counter *
'*********************
GM:
count GM_CH,10000,GM_Count            ;Count pulses for 10 seconds
serout TNC_Out,Ext_Comm,["GM:",
  dec GM_Count, cr]                   ;tell the world
pause 3000                            ;Wait 3 sec for tnc to transmit
return
```

Here, the BS2 performs housekeeping functions, like monitoring voltages and capsule temperatures. This data, along with current GPS data, is sent to the TNC enabling us to monitor the capsule's status before we commit to launch. We want to be sure the capsule is fully functional before launch, as they're too expensive to get lost in near space.

After we're satisfied the capsule is ready for flight, we attach the weather balloon and take the entire stack outside to launch. At the end of preflight mode, the BS2 goes to flight mode.

LET'S DO LAUNCH

Flight mode is divided into two segments, the housekeeping segment and the experiments segment. In Flight Mode, the BS2 continuously loops through these two segments until it has determined the capsule is descending. Each segment begins with a BS2 command to the OnCore via the BS2 GPS_Out pin. The BS2 commands the OnCore to send the GGA and the RMC sentences (with a 3-s delay between each command). I use the BS2 commands shown in Table 1 to talk to the OnCore module.

Data from the OnCore is received over the BS2's GPS_In pin. Currently, the BS2 only parses the GGA sentence for altitude data, but future flights may parse other fields as well. I've written the following code to get the altitude from the GGA sentence. In the real flight code this line is placed right after the command to the OnCore to send the GGA sentence.

```
curr_alt var word   ;current
  altitude
GPS_In  con 3       ;data from
  GPS
serinGPS_In,Ext_Comm,15000,
  badalt,[wait ("$GPGGA"),
  skip 45,dec curr_alt]
```

Command	Function
GPS_Out con 4	data to GPS
Ext_Comm con 188	baud for TNC/GPS 4800 baud, true
serout TNC_Out,Ext_Comm, ["GPS", cr]	tell world were doing gps
high TNC_Out	keep TNC line high
serout GPS_Out,Ext_Comm, ["$PMOTG,GGA,0",CR,10]	send GGA sentence, just once
pause 3000	wait for the tnc
serout GPS_Out,Ext_Comm, ["$PMOTG,RMC,0",CR,10]	send rmc sentence, just once
pause 3000	wait for the TNC
output TNC_Out	make an input

Table 1—*The last two serout commands tell the GPS to send the GGA and RMC sentences. The first just to tells us on the ground that GPS data is about to be sent.*

After getting the current altitude, the BS2 telemeters the altitude in clear text (without the surrounding GPS sentence) for the benefit of the chase crews who want to see a simple altitude reading. But, this is the U.S., so I also have a short routine that converts the current altitude to feet and telemeters that also.

Because the BS2 doesn't do floating-point math, I have a convoluted routine that is sufficiently accurate to satisfy our curiosity. When the capsule reaches 60,000′ (both on ascent and descent), we are required to notify the FAA. So, the BS2 telemeters a reminder at altitudes between 60,000′ and 70,000′.

KNSP uses the GGA sentence to track the capsule's location (latitude, longitude, and altitude) and to keep track of the mission elapsed time (MET). We use the RMC sentence to track wind speed and direction.

Typically, the OnCore responds within a second of being commanded. But, if the BS2 doesn't receive its data within 15 s, the BS2 will jump to the BADALT routine, which reports that the OnCore is not active.

It's important to place an error routine like this in the `serin` command. Without it, the BS2 will continue to wait for data if the GPS receiver fails to respond. At 50,000′ you can't get to the capsule to reset the BS2 should the GPS hang. Listing 1 shows the current code we use when the GPS locks up.

Future code modifications will probably cut the capsule loose at this point and begin emergency recovery procedures (like starting a tracking beacon). The sooner we terminate a failed mission, the closer the capsule will be to its last known position.

After GPS data is telemetered, housekeeping data is collected and sent to the ground. After performing housekeeping functions, the mission's experiments are performed (see Figures 2a and b). Because many experiments need to know the current altitude, the first order of business is to get updated GPS data. Some passive experimental data is collected by counting pulses from a sensor (such as a geiger counter) while other passive experimental data is collected via the ADC.

An example of a passive experiment we have flown is the Aware RM-60 geiger counter (see Figure 3). The RM-60 is plugged into an IHU expansion port where it uses the +5V, GND, and an I/O pin. The code for this experiment is shown in Listing 2.

The code I've written for the Experiments' ADC is in Listing 3. I change the limits of the loop according to the number of experiments on each particular flight. The subroutine in Listing 3 will report the ADC channel being read and its value.

Some of the experiments performed are active. These include operating shutters of onboard cameras, releasing gliders, or lowering tethered capsules (see Photo 3). To position servos I use the code in Listing 4, in which a servo pulls back a pin, releasing a glider.

The command to the SSC tells it to set servo channel 2 to position 70. The leading 255 is a start character for the SSC. The SSC is able to position servos with a precision of less than 1°.

Other active experiments include firing rocket engine ignitors and operating cutdown devices. For instance, the BS2 can command the Stamp Stretcher to set a pin high or low (see Listing 5). In this case, Stretcher pin 8 could connect to an input of a ULN2803 to fire a rocket ignitor.

After completing the experiment segment of the flight code, the BS2 returns back to the housekeeping segment and repeats the process over again. This setup enables us to collect about 100 data records throughout a flight, giving us a resolution of about 1000′ for each flight.

GOING DOWN

One function of the GPS subroutine is to compare the current altitude with the highest recorded altitude. This comparison lets the BS2 determine if the capsule is still ascending, or if the balloon has burst.

Selective Availability (SA) creates errors in the calculated GPS position and can produce jumps in altitude of 100 m while the capsule is on the ground (and I was sure Kansas had no earthquakes). During ascent, the capsule usually rises at a rate of 750 feet per minute. GPS data is collected about once a minute, so the ascent rate swamps any GPS altitude errors.

Just to be sure, I don't let the BS2 enter descent mode until there is a 500-m drop in altitude. Early descent speeds are greater than 100 mph, or more than 45 meters per second. So it only takes one or two loops during flight mode to determine the balloon has burst.

Currently I am experimenting with monitoring barometric pressure as a backup system to the GPS for determining balloon burst. I have found,

Listing 3—*The BS2 communicates with the MAX186 SPI ADC with this subroutine. The loop controls the number of channels we collect data from. Got a new experiment? Just increase the counter limit!*

```
Stretch                                 con 7        'data to
stretcher
CLK                                     con 14       'clock line
DATA_IO                                 con 15       ;data line
EXP_Value var word                      ;result of ADC conversion
ADC_Code var byte                       ;instructions to ADC
'********************
'* Experiments ADC *
'********************
ADC_Exp:
for loop = 0 to 2                       ;three experiments this
flight
lookup loop,[$8C,$CC,$9C,$DC,$AC,$EC,
  $BC,$FC],ADC_Code
serout Stretch,IHU_Comm,["L",0]         ;activate ADC
pause 100                               ;time to quiet down
shiftout DATA_IO,CLK,1,[ADC_Code]       ;shift in instructions
shiftin DATA_IO,CLK,2,[EXP_value\12]    ;shift out data
serout TNC_Out,Ext_Comm,[">R-", dec
  data_rep, "E-CH",dec loop+1,":",dec EXP_value]
pause 10
serout Stretch,IHU_Comm,["H",0]         ;shut her down
pause 100                               ;time to quiet down
next                                         ;do again until done
pause 3000                              ;tell the world
return
```

Listing 4—*Here we drop a glider by having the SSC retract a servo. In this case we're telling the SSC to position servo number 2 to position 70. After the glider is dropped, we set its status bit (Glide) to one.*

```
lide var bit
Servo con 6
IHU_Comm con 396+$4000              ;2400 bps, inverted
Release_Glider:
Glide = 1
serout Servo,IHU_Comm,[255,2,70]    ;retract servo
pause 1000
return
```

Listing 5— *Need to launch a rocket? Here the BS2 tells the Stamp Stretcher to set its pin 8 high. This sets an input pin on the ULN2803 high, firing the rocket ignitor.*

```
IHU_Comm    con 396+$4000 ;bps for servo and stretch 2400 bps, inverted
Stretch     con 7                  ;data to stretcher
serout Stretch,IHU_Comm,["H",8]    ;set pin 8 high
pause 1000                         ;wait a second
serout Stretch,IHU_Comm,["L",8]    ;set pin 8 low
```

Photo 3a—*The Earth's horizon at 85,000 feet. Distance to the horizon is over 300 miles away, so we're looking into other states. You can see the atmosphere as a thin blue band hugging the horizon.* ***b—****Looking down at farm fields from 79,000 feet.*

however, that above 90,000′, the pressure is lower than the PC140 can accurately measure, so we occasionally get bad readings. Future code changes will assign a minimum change in pressure to prevent PC140 readings from spoofing the BS2.

Ultimately, I'd like to use an accelerometer to make the determination of balloon burst. After the balloon bursts, speed picks up rapidly so it should be difficult to fool an accelerometer.

After the BS2 determines there has been a 500-m drop in altitude, the flight program jumps to the descent mode. There have been a few times when the GPS lost the satellite lock and the parse command generated bad altitude data. When this happens, the BS2 assumes the altitude has dropped sufficiently to start recovery procedures (we've never experienced an error that lead the BS2 to determine the capsule was still ascending).

After getting burned on one flight, I modified the GPS subroutine so the BS2 can jump back into flight mode should it determine that the capsule is really ascending. In this case, the BS2 sends a message that it was spoofed by the GPS before going back into flight mode. The code used to determine if the balloon has burst is given in Listing 6.

After returning from the GPS subroutine, `burst.bit1` is evaluated. If it is set to 0 the flight code will jump to flight mode, if it is set to 1, then the flight code jumps to descent mode.

During descent we typically close up dust samplers, release gliders (if they're still around), and stop collecting science data. With descent speeds in excess of 100 mph it's important that fragile items be retracted quickly to avoid damage from aerodynamic forces.

Occasionally we perform experiments during descent, but usually the capsule only telemeters its location. During the descent, the BS2 telemeters GPS data more frequently to help chase crews keep an accurate tab on the capsule's position.

At this time, no attempt is made by the BS2 to determine when it lands. Even if the flight code made the determination of touchdown, it would still only telemeter its location. Typically the chase crews are within a mile of the capsule as it lands. But, for the times we're not that close, it's useful to have the capsule continue to telemeter its location.

FUTURE PLANS

There are a few things about the current IHU that I'm unhappy with. One, we've never needed two-way communication with the capsule (in fact, we almost lost the first capsule when a radio operator tried to communicate with it). The BS2 has always been programmed before launch to be autonomous and has never relied on commands from the ground.

Also, it's a bit of a pain to need a TTL-to-RS–232 converter as a part of the IHU. Communications across the capsule should be easy and straightforward. Finally, the board is double-sided and therefore difficult to replicate in a home workshop.

Future designs will implement a few new features. First, the new IHU will have a transmit-only TNC built into the IHU. For this TNC, I'll use the MIM Module by Dr. Will Clement, which will remove about 6 oz from the total capsule weight and the

Listing 6—*Are we going up or down? This subroutine checks to see if the current GPS altitude is higher than the last recorded one, or is 500 meters lower. If lower, we jump to Descent Mode and tell the world.*

```
alt_curr var word                     ;current GPS alt
alt_max var word                      ;highest alt so far
alt_ft var word                       ;alt in feet
alt_temp var word                     ;temporary altitude holder

Check_4_Max:                          ;are we at a higher altitude?
if alt_curr > alt_max then New_Alt    ;set new max alt
if alt_max - alt_curr >
  500 then Descent                    ;we are dropping
serout TNC_Out,Ext_Comm,[">Lost
  GPS on Descent?", cr]               ;didn't drop enough?
goto End_GPS                          ;finished

Descent:                              ;detected a drop
Burst.bit1 = 1                        ;set burst status bit
if Phase = 1 then End_GPS             ;did we know about this?
serout TNC_Out,Ext_Comm,[">GPS
  BURST", cr]                         ;tell the world
pause 3000                            ;wait for tnc
goto End_GPS                          ;finish
```

MAX233 as well. Because I can think of a few times when two-way communication may be needed, I'll add a DTMF decoder to the board.

Second, I recently developed my own PCB workshop that works nicely with single-sided boards (it could be used to make double-sided boards with more practice on my part). So I'll simplify the IHU design to make it single-sided, which will enable me to make IHUs for groups interested in their own near-space project.

Also, making a few IHUs for myself will enable my near-space project to launch several identical capsules at the same time. Because each will be identical, there will be no need to build experiments to fit a specific capsule. For those interested, I'll post information about my new design when it becomes available, to the KNSP webpage.

You now have a brief introduction to what we do and the electronics and code required to do it. Please visit the KNSP web site, see the data we have collected in past flights, and hear the stories of our adventures. We're always glad to help others who are starting their own near-space project. As Pete Sias, my mentor said, it really is a poor-man's space program.

Lloyd P. Verhage, who prefers to be called Paul, is a former network administrator at Kansas State University and a current high school science and electronics teacher at Nampa Senior High School, Nampa, ID. His hobbies include astronomy and near-space exploration. He can be contacted at pverhage@sd131.k12.id.us.

SOFTWARE

The software used in the KNSP capsules can be downloaded from the *Circuit Cellar* web site.

RESOURCES

Kansas balloon projects, www.ksu.edu/humec/knsp/, jerryc.sound.net/habitat/, homepage. netspaceonline.com/~sias/
Nationwide balloon project, www.amsat.org/amsat/balloons/balloon.htm, www.geocities.com/CapeCanaveral/3161/hablic.htm
APRS, www.aprs.org
TAPR, www.tapr.org

SOURCES

Basic Stamp II
Parallax, Inc.
(916) 624-8333
Fax: (916) 624-8003
www.parallaxinc.com

Serial servo controller, **Stamp Stretcher**
Scott Edwards Electronics
(602) 459-4802
Fax: (602) 459-0623
www.seetron.com

Weather Balloon
Kaymont Consolidated
(800) 644-6459
Fax: (516) 549-3076
www.kaymont.com

MAX186, **MAX233**
Maxim Integrated Products
(408) 737-7600
Fax: (408) 737-7194
www.maxim-ic.com

RM-60
Aware Electronics Corp.
(800) 729-5397
(302) 655-3800
Fax: (302) 655-3800
www.aw-el.com

KPC3
Kantonics Co, Inc.
(785) 842-7745
Fax: (785) 842-2031
www.kantonics.com

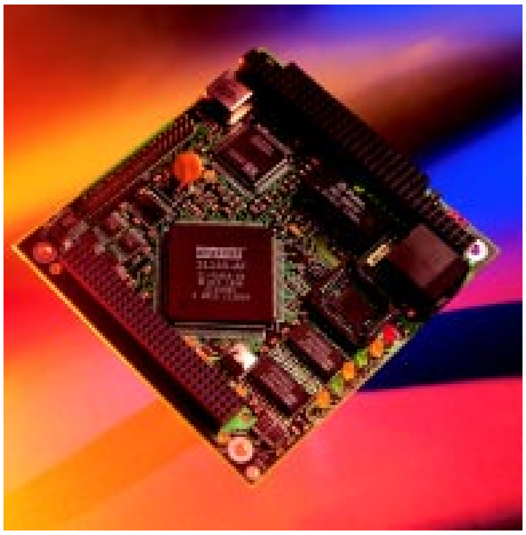
PCB
Kepro Circuit Systems, Inc.
(800) 325-3878
(636) 861-0364
Fax: (636) 861-9109
www.kepro.com

VP OnCore
Motorola
(512) 891-2030
Fax: (512) 891-3877
www.mot.com

TNCs
Tucson Amateur Packet Radio
(940) 383-0000
Fax: (940) 566-2544
www.tapr.org

PC/104 ETHERNET ADAPTER

The **COM-1450** is a high-performance PCI 100BaseT Ethernet card in a PC/104-*plus* form factor that provides flexibility and compatibility with existing PC applications. It is fully compatible with operating systems such as DOS, QNX, Windows 95/98/NT, and VxWorks. Its high integration makes this module ideal for rugged embedded applications where reliability is required.



The COM-1450 is a bus mastering PCI Ethernet card using the Digital Semiconductor 21140 Ethernet chip. The bus mastering architecture of the 21140 minimizes CPU involvement in packet transmission and reception. The large receive and transmit FIFOs permit efficient operation even with high system latencies. The 21140 chip is noted for its high-performance advanced feature set, and wide range of available drivers. The module supports full autonegotiation for automatic selection of all twisted media types. Full duplex operation is supported on 100BaseT and 10BaseT modes.

The COM1450 is a universal type PC/104-*plus* card-compatible with 3.3- and 5-V bus systems. The module requires 3.3- and 5-V power. The 100BaseT interface of the PHY is powered down when using 10BaseT. All options are software selectable except the PCI slot number. A MII compatible connector is available for use with other media types. A socket for a PLCC boot ROM is provided. Because of the high integration, the board consumes less than 2 W.

Eurotech SpA
+39-433-486258
Fax: +39-433-486263
www.eurotech.it

DIGITAL I/O CARD

The **Model 5660** is a versatile, 48-line digital I/O card that is fully ISA-bus compatible and operates in harsh industrial environments from –40° to 85°C. The card allows direct connection to many devices including opto-module racks, LEDs, relays, and other logic-level devices. It interfaces with TTL signal devices and industry-standard opto-modules like those from Opto 22 and Grayhill providing high voltage/current I/O with 4000 V of isolation.

The 5660 includes powerful programming features that simplify system design. Each port has individual 8-bit registers for data, masking, rising/falling edge sense, and change-of-state sense. Each of the 48 lines is programmable as an input or output and features programmable debounce from *x* to *y* to reject noise commonly found in industrial applications. The inputs can be programmed as interrupts, making software polling unnecessary. Since software-scanning routines are not used for either polling inputs or debounce, a system can have hundreds of I/O points with little intervention of the processor.

The card mounts in any desktop or industrial PC and will withstand 5 *g* of vibration and 40 *g* of shock. The I/O lines are protected against over-voltage and ESD by means of a 4.7-KΩ pull-up/down resistor and two clipping diodes on each line. Optical isolation also eliminates ground loops and reduces the chance of EMI events disturbing the control system.

The Model 5660 sells for **$195**.

Octagon Systems
(303) 430-1500
Fax: (303) 426-8126

Nouveau PC

edited by Harv Weiner

RISC-BASED SINGLE BOARD COMPUTER

The **Graphics Client** is a StrongARM RISC-based, single-board computer running Microsoft's Windows CE. The 4″ × 6″ card offers powerful RISC performance features along with sophisticated graphics capabilities and the Windows CE operating environment. The computer is suited to a variety of applications including hand-held, battery-operated systems, or systems requiring minimal embedded-computer space.

Significant features include the StrongARM SA1100 microprocessor operating at 220 MHz, support for three major development environments (Windows CE, MicroWare OS-9, and Windriver VxWorks), video support up to 1024 × 1024 (XGA), 10BaseT Ethernet interface, USB, PCMCIA, and IRDA communications protocols, and three serial ports. Because of its RISC environment and streamlined engineering platform, the Graphics Client produces minimal heat output and has a power requirement of less than 3 W.

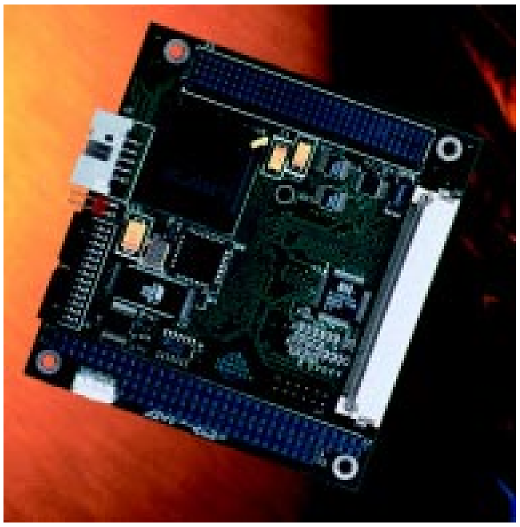
Other features include128-KB EPROM as a boot device 10 TTL digital, software-configurable I/O lines, four analog inputs with a range of 0–5 V, and manufacturer-configurable I/O for keypad, digital touchscreen and additional analog and digital ports. A real time clock, LED status indicators, and onboard Codec are also included.

The Graphics Client pricing starts around **$300**.

Applied Data Systems, Inc.
(301) 490-4007
Fax: (301) 490-4582
www.flatpanels.com

PC/104-*plus* DISPLAY MODULE

The **MiniModule/VFP-III** module delivers ultra high-speed, PCI-based CRT and flat-panel video performance on a compact, low-power PC/104-*plus* expansion module. Designed to stack directly with Ampro's family of PC/104-*plus*–compliant CoreModule CPU products, the module offers multiple video interface options to support a wide variety of display technologies.



Options include analog CRT, television (NTSC or PAL), and digital flat-panel (LCD, EL, and Plasma). An onboard PanelLink transmitter option is available for interfacing panels in systems requiring extreme noise immunity or remote panel mounting. In addition, a zoomed video (ZV) input port allows a live video image from an external frame grabber to be overlaid on a standard VGA graphics screen.

To support variable signal timing and interface requirements, a programmable VGA BIOS is located in an onboard flash memory. Preprogrammed flat-panel BIOS files for several displays are available, and OEMs can develop configuration files for additional displays using an optional flat-panel BIOS customization kit.

The MiniModule/VFP-III offers full software compatibility with all popular video standards. Resolutions of up to 1280 × 1024 and color depths to 24-bit true color are supported. In addition, the display controller includes GUI (graphical user interface) acceleration hardware that can dramatically boost the performance of Windows and other graphics-intensive applications.

The MiniModule/VFP-III is priced from **$267** in OEM quantities of 100.

Ampro Computers, Inc.
(408) 360-0200
Fax: (408) 360-0220
www.ampro.com

Nouveau PC

RPC

Real-Time PC

Ingo Cyliax

Parallel Port Interfacing

Join Ingo as he cruises the ins and outs of using the standard parallel port interface found on your PC. This tour gives you the basics on the different implementation modes as well as how to connect devices to the port.

Like the serial ports I wrote about last month, parallel ports are available on almost all PCs. In many embedded-PC applications, the parallel port is not used to drive printers. The basic parallel port on a PC is less structured than a serial port. It doesn't force you to use a specific protocol, unless of course, you're talking to a printer.

THE HARDWARE

The standard parallel port hardware is quite simple. There are three I/O registers—data, control, and status. Figure 1 gives you an idea of the basic register layout. The registers are in a continuous register address space and usually start at 0x3bc, 0x378, 0x278, or 0x2bc.

Figure 2 shows the hardware and the pin assignments on the individual input and output signals. The data register, which starts at the base address, is a simple 8-bit output latch. The 8-bit value you store in the data port register appears on the eight data lines (d0–7). When used with a printer, these signals carry the information (i.e., bytes) you want to send to the printer.

The control register contains various output signals most often used to control a printer. Although these signals are important to the printer controller, they are just output signals (with a few twists) as far as the printer port hardware is concerned.

First, most of the output signals in the control port are inverted, which means that when you set a 1 in the control register, the inverted signal will present a low voltage level on the output pin. Not all of the signals are inverted. In Figure 2, inverted signals are marked with a bubble (which signifies a voltage inversion).

The other twist on the control register is that these are actually bidirectional signals that are implemented as an open collector (open drain) signal. The hardware can only pull the signal low so a resistor is used to pull the signal high. Other open-collector signals can also pull this signal to a low state. By reading the control port, we can detect the true state of the signal.

If you want to read one of the control signals, you set the control register bits for that signal so that it sets a high. If another open collector driver pulls this signal low, it will read back as a low signal.

Of course, you have to account for the bubbles to find out what state the control register is in. Writing a 1 to the strobe signal will set it low. So,

Base + 0	Data	D7	D6	D5	D4	D3	D2	D1	D0
Base + 1	Status	*BSY	*ACK	PAP	OFON	*ERR	X	X	X
Base + 2	Control	X	X	X	IRQ	DSL	*INI	ALF	STR

Figure 1—*The standard parallel port uses only three registers. EPP/ECP parallel ports have additional registers that are used to enable and control them.*

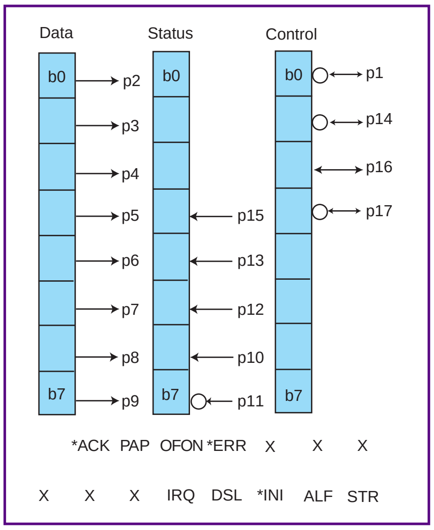


Figure 2—Here are the physical pin assignments of the SPP. A bubble indicates that there is a voltage inversion on that pin. When a signal is inverted like that, a 1 written in the register will generate a low voltage on the pin.

if you want to read the true state of the strobe signal, you have to write a 0 to the strobe bit. When you read back the strobe bit, a 0 means that the voltage is high and 1 means that the voltage is low.

Incidentally, on many parallel ports the data registers function similarly. By writing a 0 to a data bit, we can read the true signal state if another device overrides the low voltage with a high voltage. Although you can rely on the open-collector nature of the control port, you can't be sure that the parallel port on your PC will behave in this sort of bidirectional mode.

Finally, the control port has an interrupt request enable bit. This signal is not available outside the port, but controls the interrupt behavior of the parallel port itself.

The status register is a true read-only port. The signal can be read by the software. You have to account for the voltage inversion of these signals. These are also shown in Figure 2.

Finally, the ACK signal in the status port will generate an interrupt on the falling edge if the interrupt request enable bit in the control register is set to 1. When used with a printer, the handshake roughly follows this protocol (see Figure 3).

The CPU polls the busy signal in the status register to make sure the printer is available. The busy signal is asserted if the printer's buffer is full or if the printer is offline. Once the printer is not busy, the software will store the data to be sent in the data register and will pulse the strobe signal. The printer uses the strobe signal to store the data in a latch or buffer and then strobes the ACK line. If enabled, this signal causes the processor to interrupt and send the next byte. This process continues until the processor has no more data to send, or the printer asserts busy.

The general-purpose outputs (data register) and inputs (status register) as well as the open-collector I/O (control register) and the interrupt capability of the parallel port give you plenty of possibilities, even if you don't use a printer.

The PC parallel port is adapted from the Centronics standard serial port. The original Centronics parallel port used a 37-pin connector. The PC version shrunk this down to a DB-25 connector so it could fit on the original graphics adapter card. You're unlikely to find the old Centronics-style connector except on some older printers. Table 1 summarizes the signal types, the port they appear in and the pin numbers on the DB-25 connector. The directions are relative to the computer.

Up until now, I have been speaking only of the parallel port hardware. This hardware is the minimum subset you will see on a Windows-compatible computer and works well with all types of parallel printers. You may have noticed that the printer has several control signals it can use to signal error condition status information (e.g., out of paper).

As printers got smarter, there was a need to communicate more information from the printer to the computer. For example, in a laser printer you may also want to communicate that although the paper tray is not empty, it holds the wrong kind of paper. The parallel port also has the ability to transfer data faster than a serial port, which makes it useful for devices such as network adapters or

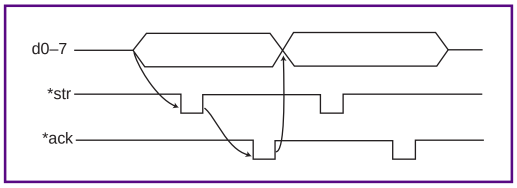


Figure 3—During the basic Centronics printer handshake, a strobe is used to indicate that a byte can be stored in the printer. An acknowledge is sent by the printer to indicate that the byte has been received.

external disk/tape drives, and for connecting to other computers.

The original parallel port was optimized to transfer data in only one direction—from the computer to the printer. For bidirectional communication you need to enhance the capabilities. Let's take a quick look at some ways to accomplish this enhancement.

One way to achieve bidirectional transfers is to use some of the five status bits as general-purpose data bits. Because one is already used to sense the printer ACK signal, we use the remaining four signals to transfer the data. Four bits makes up a nibble so this mode is called nibble mode over the standard parallel port (SPP).

The IBM PS/2 computer pioneered the use of a bidirectional parallel port. An extra bit in the control register would change the direction of the data port, thus enabling you to read the data back from the printer. This mode is called PS/2 mode, or bidirectional mode.

Transferring data one byte at a time and checking the ACK bit is time consuming, so you can add some FIFOs and handshake hardware. The enhanced parallel port (EPP) can automatically

Pin	Signal	Port	Direction	Description
1	STR	Ctrl/0	I/O	Data transfer strobe
2	D0	Data/0	O	LSB of printer data
3	D1	Data/1	O	Bit 1 of printer data
4	D2	Data/2	O	Bit 2 of printer data
5	D3	Data/3	O	Bit 3 of printer data
6	D4	Data/4	O	Bit 4 of printer data
7	D5	Data/5	O	Bit 5 of printer data
8	D6	Data/6	O	Bit 6 of printer data
9	D7	Data/7	O	MSB of printer data
10	ACK	Stat/6	I	Data acknowledge
11	BSY	Stat/7	I	Printer busy
12	PAP	Stat/5	I	Paper out
13	OFON	Stat/4	I	Off/on-line
14	ERR	Stat/3	I	Printer error
15	DSL	Ctrl/3	I/O	Printer select
16	INI	Ctrl/2	I/O	Reset printer
17	ALF	Ctrl/1	I/O	Auto line feed
18–25	GND	—	—	Ground

Table 1—For the standard parallel port signal-to-pin assignments, the pin numbers correspond to the DB-25 connector found on all PCs.

transfer 32-bit words over the 8-bit parallel port interface. You can never have enough speed, so the extended capability port (ECP) adds data compression by using run-length encoding. Both the EPP/ECP ports can be implemented using DMA on some motherboards to offload the CPU.

Now that you have SPP, PS/2, EPP and ECP ports, and various ways of implementing information flow from the printer (or other external device), things get kind of confusing. Many motherboards let you select the parallel port implementation (primarily to make it work with the right kind of peripherals). IEEE-1284 is a standard that describes all the modes and how to negotiate what mode/port types to use. This standard also implements standardized protocols to communicate with the external device. One feature is that you can now probe the printer and find out what's connected for plug-and-play (PNP) capability.

The details of IEEE-1284 are beyond the scope of this article since I'm focusing on nonprinter uses for parallel port hardware. As always, if enough of you let me know that you're interested in the details, I can explore the topic in a later article.

HOOKING THINGS UP (INS/OUTS)

Now that you know how the parallel port is wired, let's look at some ways to hook things up to it. Specifically, let's look at how to connect stuff to an SPP. In a nutshell, all input and output signals are TTL compatible and you can use any TTL-compatible logic family (HCT, etc.).

Hooking up an 8-bit DAC is easy. You only need to wire the eight data lines to the parallel input port of the chip (see Figure 4a). Just write the value to the data port, and use a software timer to control the output data rate of the DAC.

For this example, I used a DAC0808 from National; a fairly old chip that I found in my junk box. Wiring up newer chips shouldn't be a problem. If the chip you want to use needs a strobe to store the date and start the conversion, you can use the strobe line. Update rates of over 300 kHz are easily achieved in this way.

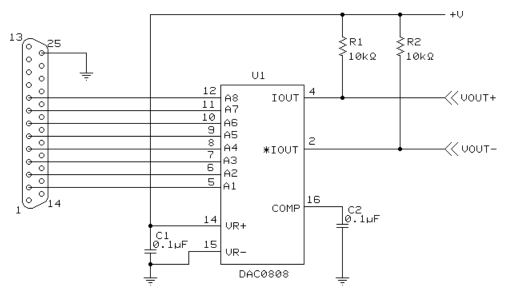
Figure 4b shows how to hook up an 8-bit input device to use for nibble mode operation. Here you interface a basic 8-bit ADC. The strobe signal (wired to the /WR of the ADC) starts the conversion. You can use the ACK line of the parallel port to sense when the conversion is done.

To multiplex the two nibbles, use the DSL and ERR signal. A low on these signals will enable one half of the 'HCT244. You have to be careful to only assert one at a time. One bit is inverted on the status port (p11) when we read the nibble back. Also, the order is jumbled. Listing 1 shows the code necessary to read a byte from the ADC.

Listing 1—Reading the 8-bit ADC using "nibble mode" isn't so straightforward. Because I wanted to make ACK line available for interrupts, the bits are kind of jumbled up. It's not really a problem, most '486-class machines will cut through this code much faster than the ADC on the port anyway. The bit manipulation doesn't present much overhead.

```
outb(0x278+2,0x01)            /* assert strobe to start */
while(inb(0x278+1) & 0x40)    /* wait for intr to go low */
        ;
outb(0x278+2,0x04)            /* high nibble */
high_nib = inb(0x278+1);

outb(0x278+2,0x08)            /* low nibble */
low_nib = inb(0x278+1);

high_nib = (high_nib ^ 0x80); /* invert bsy */
low_nib = (low_nib ^ 0x80);   /* invert bsy */

/* reassemble 8bit data */
data =  (high_nib & 0x80 >> 1) |
        (high_nib & 0x20 |
        (high_nib & 0x10 |
        (high_nib & 0x08 << 4) |
        (low_nib & 0x80 >> 5) |
        (low_nib & 0x20 >> 4) |
        (low_nib & 0x10 >> 4) |
        (low_nib & 0x08) ;
```

a)



b)

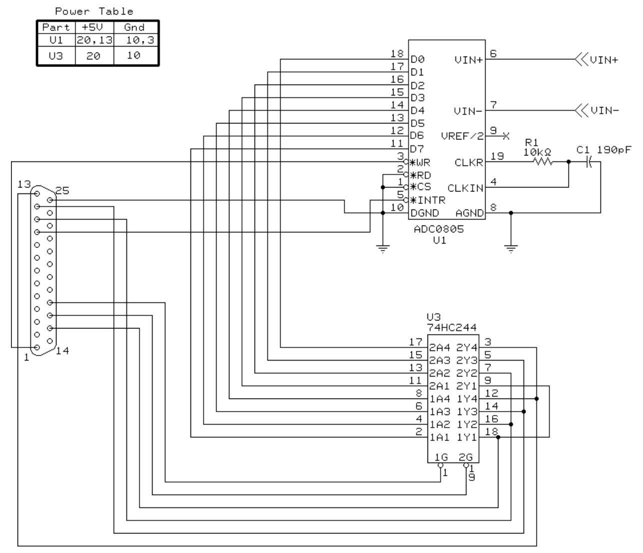


Figure 4a—The basic 8-bit D/A converter hookup is a no-brainer. Just wire up the data signals and pump data to it with software. Newer DACs might require a write strobe. We can use pin 1, the strobe line, for that. b—Here we are hooking up an 8-bit A/D converter in nibble-mode. The strobe line is used to initiate the conversion. The ACK line is used to sense when the converter is done converting and can be used to generate a interrupt request on the CPU.

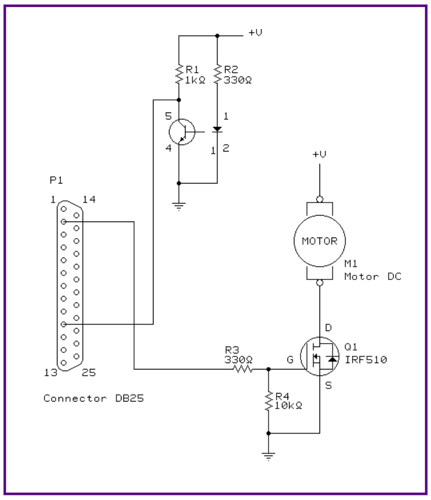


Figure 5—In this single-bit DAC motor controller with speed sensor feedback, you control the speed by modulating the pulse width of a fast pulse train to the MOSFET and sense the speed by counting pulses from the optical sensor that senses slots in a wheel attached to the motor's shaft.

I mentioned that the parallel port has interrupt capability. Because the INTR signal from the ADC is already wired to the ACK line, you can turn on the interrupts in the control register. Now, when the ADC is done converting, the ACK line will go low and cause an interrupt (IRQ 5 or 7). You can use almost the same code to read the data from the ADC, except you don't need a busy loop to poll the ACK bit. A timer routine can be used to fire the ADC at a regular interval.

Finally, how about a simple DC motor controller interface? In Figure 5, I hooked up an N channel power MOSFET to D0 of the parallel port. When I turn the line on, it sinks current through the motor to ground.

By modulating the MOSFET with a pulse train, I can vary the amount of power going to the motor. If the pulse train is fast enough, the motor won't respond to the individual pulses and will essentially average the power available. Thus, I can control the motor speed by varying the duty cycle of a pulse train on D0.

To measure the speed, I use an optical interrupter. This works like an optoisolator, except that the light path has an air gap. The motor has a slotted disk wheel attached to the shaft. When a slot passes through the light path of the sensor, a pulse occurs on the ACK line. The ACK can then cause an interrupt and the interrupt routine, with the aid of a real-time counter, measures the pulse rate, and thus the speed.

This is a fairly crude example, but illustrates the point. Some constraints to this implementation result from the software overhead necessary to process interrupts and manage processes and tasks. For example, in the RTOS I'm using (RT-Linux), the interrupt latency can be up to 10 µsec, on the particular machine I'm using. So, with a margin of 10:1, we could probably only deal with a 100-µs maximum interrupt rate. So, that would give me 100 µs = 10 kHz = 600,000 RPM, which is overkill for this application, but it's good to know the limits. Similar limits exist for the PWM code.

Of course, an 8-bit processor might be much better suited for this application since it's cheaper. But, if you already have a machine with a free parallel port, this is essentially free. The processing can be kept manageable by keeping the interrupt handler code compact. In this example, I was able to run a complete network-capable operating system and do other work, while the interrupt drive parallel port interface "did its thing."

BON VOYAGE

We have only scratched the surface on this topic. For a more extensive reference, check out Jan Axelson's *Parallel Port Complete*. Also, some of the other references have basic information about the PC parallel port interface.

So, the next time you need a couple of extra I/O bits, or perhaps an external interrupt source, maybe you can use a spare parallel port. RPC.EPC

Ingo Cyliax has written for Circuit Cellar *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. You may reach him at cyliax@derivation.com.*

REFERENCES

J. Axelson, *Parallel Port Complete*, Lakeview Research, Madison, WI, 1997.
W.L. Rosh, *Hardware Bible*, SAMS Publishing, Indianapolis, IN, 1997.
H. Messmer, *The Indispensable PC Hardware Book*, Addison-Wesley, Essex, England, 1997.

SOURCE
DAC0808
National Semiconductor
(408) 721-5000
Fax: (408) 739-9803
www.national.com

Fred Eady

Thin is In

Clients, Servers, and Systems

Anyone who's seen the latest palmtop systems can attest to the fact that computers are getting smaller. Fred looks at a new idea for making systems even tinier—running apps on a remote server via some slim software.

Back in days of leased hardware and software that filled rooms the size of football fields, a relatively obscure company called IBM was successfully implementing what we would know later as thin client architecture.

The user terminal of choice was called a 3270 workstation and consisted of a CRT and keyboard. The CRT/keyboard combination was the perfect picture of a dumb terminal.

All of the smarts were contained in a coax-attached external terminal controller called a 3274. The 3274 was a cluster controller capable of handling up to thirty-two 3270 devices.

Impact printers with 3*xxx* designations could also be attached to the 3274's BNC ports. Parallel channels using massive cables passed data between the central processor, the 3274, and its 3270 parasites.

At that time, the smallest of these processors was the size of a large doublewide refrigerator. Typically, though, the processor footprints were large enough to take a little bit of time to walk around.

Practically all of the application smarts were contained at the processor level. All of the programs were run on the remote processor that resided in the glass house.

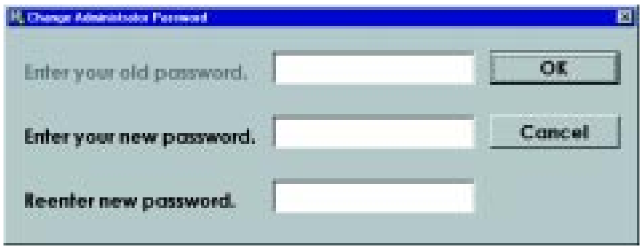
Photo 1—This box's got real hinges, too!

The 3274 was only there to convert the raw data and application interactions to and from the 3270 terminal protocol.

This processing architecture still exists today, with the 3274 and 3270 controller/terminal combination giving way to smarter devices at the terminal end. In fact, the 3270 green screen terminals gave way to the 3279 high-resolution color units later on. This was (and is) a large implementation of massive processing power capable of simultaneously serving thousands of users anywhere in the world.

To sum up the technology, relatively small packets of data from terminals were bounced between a processor (server) and terminal (client) with most of the heavy-duty application processing occurring at the server side. It has been debated for years as to the real value of having such a behemoth system when today's miniature desktops can do the same job. I really don't want to get in that fight.

It's funny that later on, the powers of the computing world decided that this was not the most efficient way to compute or do business. So, they decided to "distribute" the processing by offloading the processor (server) applications to the clients.



Change Administrator Password

Enter your old password. [ ] OK

Enter your new password. [ ] Cancel

Reenter new password. [ ]

Photo 2—Strawberries and cream to any administrator.

Thus, the clients (terminals) got smarter and the processors (servers) got smaller. IBM's first smart terminal answer was the 3270PC, which was no more than an IBM AT with some special BIOS hooks and a 3270 ISA adapter card.

At the time, the IBM AT ran at a mind-bending 6 MHz on top of a 30-MB hard drive! They even extended this PC/AT technology to run the large OSs like VM (Virtual Machine).

That was then, and this is now. Thin is in…again.

OLD FRIENDS, NEW TECHNOLOGY

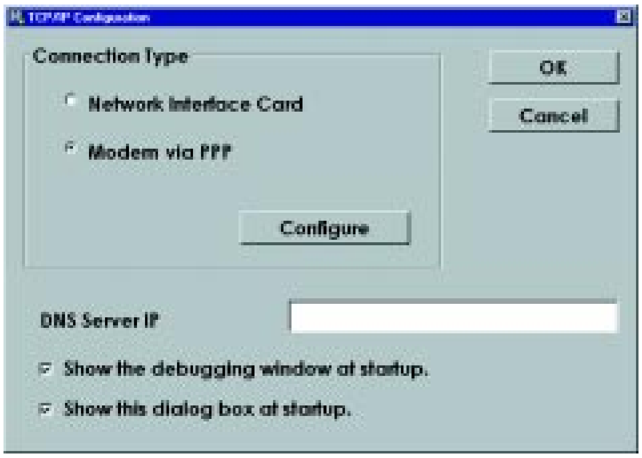
Datalight, known for ROM-DOS and WinLight, calls its Thin Client offering ThinSystem. ThinSystem is a software package that includes major components required to connect and operate a thin client system with a remote server. The end users can access and run applications on the remote server without having the burden of containing the application at the client.

I wasn't giving you a history lesson for nothing. Like the old IBM 3270 systems, Thin Client runs application programs like browsers and spreadsheets remotely on the server. The client system is responsible for passing keystroke and mouse data back to the server.

On the return stroke, and again just like their 3270 ancestors, Thin Clients receive raw data and video information that must be translated and passed to the eyes and ears of the user. In the IBM large computer environment, the 3270 protocol was the way data went from terminal to mainframe and vice versa.

For the ThinSystem, the client-side software (Citrix WinFrame or Microsoft Terminal Server Client) must have corresponding Citrix or Microsoft server-side components. My little piece of IBM history didn't go into what other companies were doing at this time, but we all know that another unknown corporation called Digital was in with the research and college crowds. They are known for the VT series of async terminals that connected to the then Digital VAX line of computers (and eventually to most everything else).

ThinSystem can emulate VT100, 5250, and 3270. 5250 is an IBM midrange protocol used on System32, System34, System36, System38, and AS/400 business computers. The 5250 protocol is still around and normally runs over what's called twin-ax cabling. The AS/400 is a formidable processing system and is IBM's premier midrange system.



TCP/IP Configuration

Connection Type

Network Interface Card

[✓ Modem via PPP]

Configure

OK

Cancel

DNS Server IP [ ]

[✓] Show the debugging window at startup.

[✓] Show this dialog box at startup.

Photo 3—Of course, I clicked on Modem via PPP.

All of the System3*x* stuff brings a story to mind. At that time, System36 was one of the most popular lines of small business computers IBM sold. I remember meeting a couple guys who wrote database applications for local businesses that had System36 installations.

In case you've never seen a System36, in those days, the smallest one was about the size of a large freezer. You know, the ones that open from the top? Down South, we keep fish in ours.

Anyway, I got to be friends with these bit-heads and one day I asked how they managed to write and debug code for this system since it was normally only found in a business environment. "Well," they said, "Get in the car."

They drove me to their house and, lo and behold, there was a full-blown System36 smack dab in the middle of their living room! My kind of guys!

Anyway, I not only have the Datalight ThinSystem software package, I have a neat Arcom embedded PC to run our ThinSystem on. Again, I'm going to take you through my experiences, beginning with the removal of the shrinkwrap.

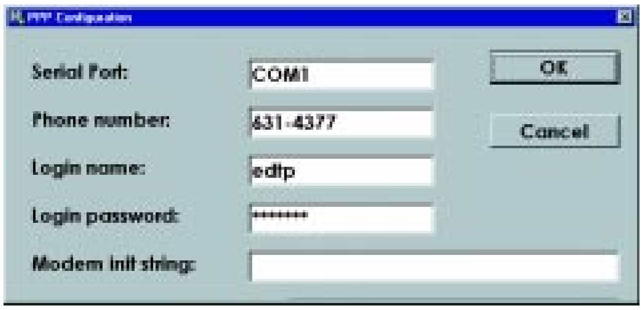
THE CHICKEN OR THE EGG

Unless I'm out of touch, I haven't seen any scientific evidence confirming the existence of a chicken laying the first egg. I'm gonna stick my neck out and proclaim that software followed hardware. (That will either prove me to be a genius or generate thousands of "Fred, you idiot" e-mail responses.) Regardless of the answer to either of the above mysteries, I'm still going to start by describing the hardware first.

Datalight ThinSystem hardware comes housed in a heavy duty, blue, padded carrying case (see Photo 1). Everything a ThinSystem developer needs is in the case somewhere. There's even an adapter for PS/2 keyboards that takes the smaller PS/2 connector to the larger legacy 5-pin DIN.

For the international readership, the modular power supply can be outfitted with three differing AC plug schemes. In essence, the ThinSystem hardware comes complete and ready to run right out of the case most anywhere in the world.

ThinSystem software is shipped as a CD but is also present within the bowels of the Arcom SBC-MediaGX I found in the blue case. Although designed specifically as a design reference board, the Arcom SBC-MediaGX is a whole bunch of embedded peripherals supported by a fast processor complex.



Serial Port: COM1
Phone number: 631-4377
Login name: edtp
Login password: ********
Modem init string:

Photo 4—*Had a little trouble being "smart" and toying with the Modem Init string field.*

The idea is to develop the application with this board and embed the same hardware/software configuration into an OEM product. The Arcom SBC-MediaGX has all the functionality of any PC/AT-compatible system with these additions:

- 16-bit SoundBlaster
- PC/104-*plus* expansion bus
- MMX-enhanced CPU
- high-performance flat-panel controller

The Arcom SBC-MediaGX clocks in at 233 MHz with a National/Cyrix processor, 32 MB of DRAM (128 MB max.), and 8 MB of Intel Strata flash memory. You can skinny the above down to any combination of DRAM and flash memory, but the SBC-MediaGX was specifically designed to support the National/Cyrix processor and it must remain the same.

On the communications side, the SBC-MediaGX is strong. Not two, but four RS-232 ports with a single RS-485-capable port are on the board. Realtek supplies my Ethernet interface at 10- or 100-BaseTX specifications.

There are also a couple of USB interfaces. With all that hardware, you may ask, what can't this thing do? Well, all of Bill's Windows can be washed, including his palmtop/embedded version, CE. You can go north of the border for QNX and you can pet the Linux penguin.

Although the SBC-MediaGX can accept the full complement of rotating media, I've decided to not attach any of that stuff and to run the board only on flash memory. If anything has to be downloaded or uploaded to the SBC-MediaGX, I plan to do it with either the serial or the Ethernet interfaces.

I'm not going into the CMOS setup detail because it is a standard process and differs little from any other PC-compatible embedded platform. However, I printed out the 0.397″ of doc that comes in Acrobat format on the hardware SBC-MediaGX support CD. It's thorough.

Basically, I took the brand spankin' new SBC-MediaGX embedded PC out of that beautifully engineered and padded antistatic case, and immediately dropped it!

I live in an older Florida home, and the floors are made of a mix of rock and concrete called terrazzo. The processor heatsink is glued on, and the first thing I thought I was going to have to do was chase the sink across the hard, slick Florida Room floor. I had installed the DRAM stick before I fumbled the SBC-MediaGX and I thought the DRAM stick wouldn't survive the meeting with the terrazzo either.

Well, what do I know? I plugged in the video ribbon and connected a standard mouse and keyboard. The power supply connector is supposedly keyed and contains only a +5-V and ground termination. Looks like you could put it on either way, so I took the path of least resistance.

I installed the standard AC plug set on the power brick and fired it up. No problems. I was surprised to find the ThinSystem software and ROM-DOS already installed in flash memory. I was expecting to have to tell you about doing that. It's OK—I just get more page space for important things like describing how to connect to Datalight's Citrix server with the ThinSystem-filled SBC-MediaGX.

THE SOFTWARE SKINNY

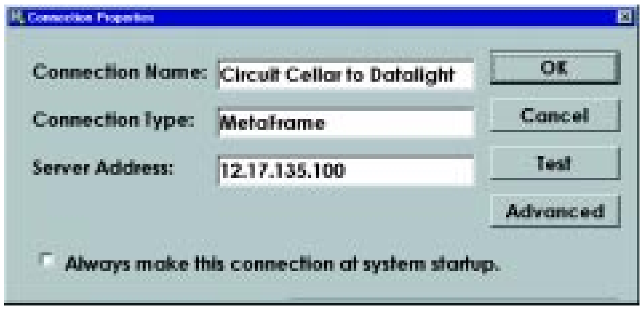
Datalight's ThinSystem includes Datalight Sockets, WinLight, Citrix WinFrame client software, a VT100 terminal emulator, and ThinSystem configuration utilities. Datalight Sockets is a Winsock 1.1–compliant TCP/IP stack. WinLight is Datalight's implementation of Bill's most famous work.

Our ThinSystem requires a fully compatible MS-DOS operating system. ROM-DOS fits the bill here and is included with the development kit. One extra that ROM-DOS brings to the table is the ability to interface with Datalight's FlashFX. FlashFX works with ROM-DOS to eliminate the spinning mechanical stuff.

Recall that data is sent and received between the client system and server in small packets. By design, there is no continuous datastream. If we want to transport these packets of data via an NIC, the SBC-MediaGX Ethernet interface requires the correct driver. Because the ThinSystem can be run on many different embedded platforms, it will obviously encounter many differing NICs. The SBC-MediaGX uses the Realtek RTL8139A Ethernet controller.

Realizing that design engineers have better things to do than run down obscure NIC drivers via the Internet, the Arcom/Datalight ThinSystem development kit comes loaded with the correct Realtek NIC driver. As well, standard drivers like the NE2000 driver are included on the CD. For those choosing to run ThinSystem on a desktop and other special cases, the CD documentation provides a URL for a good source of packet drivers.

ThinSystem configuration depends on who's doing the configuration and his or her goals and expectations. A designer, like you, would have a nuts-and-bolts approach to configuring a ThinSystem client, whereas an end user wouldn't be concerned with the hows and whys. They'd


Connection Properties

Connection Name: Circuit Cellar to Datalight

Connection Type: Metaframe

Server Address: 12.17.135.100

Photo 5—Piece of cake. Circuit Cellar to Datalight is the connection name, MetaFrame is the connection type, and 12.17.135.100 is the Datalight Citrix server address.

just want it to work. An administrator would be concerned with how the ThinSystem software could be configured to interact seamlessly with the ThinSystem designer's efforts and the user's goals. Let's take a look at this as a designer from the administrator's view.

ADMINISTERING THINSYSTEM

Most administrators are password-happy. So, the first thing the ThinSystem software does is put up the change administrator password screen (see Photo 2).

After a password is established, choosing the Admin Mode entry from the user menu exposes the system admin functions. After entering Admin Mode, the administrator can alter the administrative password, select the position of the WinLight taskbar, configure TCP/IP, define connections, and perform file transfer functions.

The taskbar display settings let you put the taskbar at the top or bottom of the WinLight screen. A mouse-over mode can also be set here. Ctrl+Esc activates the taskbar if no mouse is present.

ThinSystem can communicate with a remote server via the SBC-MediaGX Ethernet interface or standard RS-232 via a modem. Unfortunately, the SBC-MediaGX development kit didn't come with an external modem. If your design is dial-up based, be sure to take that into account.

Either way, Datalight Sockets must be configured for everything to work correctly. If the connection is to be made via an NIC and DHCP is not available, the system administrator or designer must supply the correct IP address, net mask, and gateway address. Photo 3 is where the IP work begins.

I don't have a direct LAN connection to the Internet so I dug out an old U.S. Robotics 28.8 Sportster external modem and attached it to COM2.

Since the Florida Room is without a Citrix server, the Datalight folks were kind enough to allow me to access theirs. All I had to do was enter the COM port, phone number of my ISP, my ISP login name, and my ISP login password. The PPP skeleton can be seen in Photo 4.

The next step was to set up a connection to the Citrix server out in Washington State. All the data I needed was supplied by the Datalight folks, with the exception of the connection name. Photo 5 shows the Connections Properties window.

OK. All set up and ready to connect to the Citrix server. I pushed the mouse pointer to the top of the screen, and the Start button appeared.

A click of the Start button, another click on the Connections menu item, and a final click on the Circuit Cellar to Datalight connection. Nothing, nothing at all. The phone line won't give me a dial tone and the client just keeps retrying with no luck.

I went to the nearest phone that was on the same line I was attempting to dial from and picked up the handset. Silence. Dang hurricanes! I'll bet the little splice job I did on the phone line outside the Florida Room is a tad wet from Floyd.

Well, not just wet—gone. I never claimed to be a professional "telephone guy." No problem. I fixed it and fired up the ThinSystem client again. Still nothing.

Two days later, I discovered that putting text into the Modem Init string field (Photo 4) under PPP Configuration was a mistake. (I do claim to be a telecommunications expert.)

Seems that just entering "AT&F" to reset the U.S. Robotics external modem didn't hack it with the ThinSystem software. The giveaway was that after the command was processed, it was echoed back to the terminal debug screen and the expected "OK" never appeared.

The key to entering text in the Modem Init field is to add everything as a character. That is, to send a carriage return, you must enter its Hayes command equivalent.

The simple thing to do was to not put anything there. Son of a gun! After applying that fix, I got a dial tone and connection but still no logical session with the Citrix server. To the phones....

I spoke with Datalight's Robert Krantz and his first suggestion was to hook up to the ISP and attempt to ping 12.17.135.100 (the Citrix server). Sounds reasonable. Did I say something about being a telecommunications expert? It helps to enter the correct IP address.

Again I fired up the SBC-MediaGX/ U.S. Robotics combo. The normal modem tones were exchanged, my ISP delivered an IP address to my SBC-MediaGX, and whap! A little man in a business suit holding a briefcase was flying on my MetaFrame screen that was being served from the Citrix server. Moments later, a familiar sight, the Windows NT desktop, appeared.

MY DESK(TOP) IN WASHINGTON STATE

Just for grins, I clicked up Microsoft Word to look at what NIC was installed on the server. I played around just like I'd never seen NT or any of its innards. It was pure joy. The reality of operating a Windows NT machine from an embedded PC all the way across the U.S. was awesome.

Although the SBC-MediaGX has plenty of spunk, I wasn't even moving its stress meter. And then it occurred to me that I was only connected at 26 kbps, not 56 kbps, through an ISP connection running heavy Microsoft apps as if I was sitting there with Mr. Krantz!

Once again, the computing world has gone full circle with Thin Client technology. And now, Datalight and Robert Krantz have proven that it doesn't have to be complicated (or big) 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.

SOURCES
ThinSystem
Datalight
(360) 435 8086
Fax: (360) 435-0253
www.datalight.com

SBC-MediaGX
Arcom Control Systems, Inc.
(888) 941-2224
(816) 941-7025
Fax: (816) 941-7807
ww.arcomcontrols.com

Design99: a world of opportunity for the development engineer.

Here are the top winners of *Circuit Cellar's* explore the 68HC908GPXX family of

Roboat
$5000 Prize

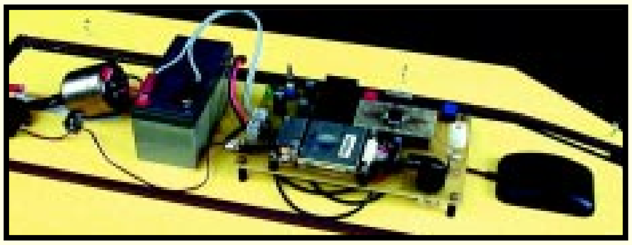
This project aims at testing the feasibility of driving an object using a GPS and a digital compass. In this case, the object is a model boat that can sail autonomously along a course planned in advance. The HC908GP20 stores the coordinates of the way-points to follow and, according to the data received from the GPS and from the digital compass, controls the electric motor for the propeller and the proportional servo for the rudder. Three switches enable the test modes and a potentiometer allows you to trim the rudder center position.

A MON08 interface allows the connection with the ICS08GP20 board to control Roboat directly from a PC for testing purposes and when new waypoint coordinates need to be stored.

The software for the HC908GP20 is written in assembler and controls all the functions of Roboat. The main actions range from starting the propeller and powering up the GPS, to acquiring data from the digital compass, to computing the course correction in order to head towards the next waypoint.

Because of all the computations with coordinates, plenty of effort was put into developing routines in assembler to deal with simplified mathematics and to handle arithmetic operation with four-byte numbers, azimuth angles, and the trigonometric function ArcTangent.

Riccardo Rocca
Piacenza, Italy
riccardo.rocca@iol.it


RoBoat

sCo*Pilot
$5000 Prize

sCo*Pilot

The practical functionality of modern digital oscilloscopes (as well as other test equipment) is unquestionable. Many modern digital scopes are eclipsing their analog counterparts in applications ranging from digital systems debug to RF design.

However, the advantages of many of these advanced capabilities can be somewhat negated by cumbersome user interfaces. Sometimes there are not enough buttons (function keys) on the scope, or quantities intuitively associated with knobs require multiple keystrokes to adjust.

As the name implies, the sCo*Pilot is a device intended to function as a "co-pilot" for operating an oscilloscope. Targeted at the Tektronix THS series of handheld digital oscilloscopes, the sCo*Pilot functions as a "copilot" between the user and the scope by providing human interface enhancement to the THS scope. The sCo*Pilot eliminates the frustration and confusion of using the scope's native "soft keys" to cycle through hierarchical menus. The single-button access to commonly used features, knobs (with acceleration) for adjusting analog quantities, and LED status indication have dramatically increased user efficiency and usability of the THS scopes (or other digitally controlled instruments).

Derek Matsunaga
Lousiville, Colorado
derek700@aol.com

sponsored by:

CIRCUIT CELLAR® THE MAGAZINE FOR COMPUTER APPLICATIONS

Digital DNA™ from Motorola

MOTOROLA

eleventh design contest, Design99. These projects flash memory microcontrollers from Motorola.

Neural Stamp
$5000 Prize

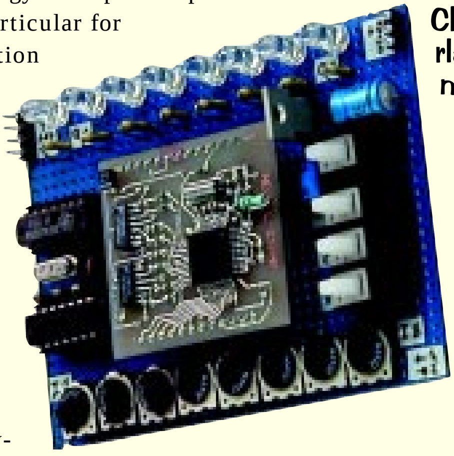
Considered a research topic for years, neural networks are now a mature technology with proven performance, in particular for pattern recognition and process control. However, the use of neural network techniques in embedded systems is still limited, mainly due to the poor offering of ready-made low-cost hardware platforms and dedicated development tools. To improve this situation, Robert designed a low-cost canned neural network implementation he calls the Neural Stamp.

Thanks to the NS'Drive, the Neural Stamp can be a quick and cost-effective solution for many medium complexity embedded-control applications. The speed of the HC908GP20 gives the Neural Stamp a response time of 50 ms, which is adequate for the majority of process control applications.

And if one Neural Stamp isn't enough for a given application, several Neural Stamp macro-chips can be easily chained together to build more complex networks.

Because the microcontroller still has a lot of free RAM and flash memory, there's room for improvements such as including learning algorithms directly on the target processor, which could open the door to adaptive in-field training.

Robert Lacoste
Chaville, France
rlacoste@nortelnetworks.com


The Neural Stamp

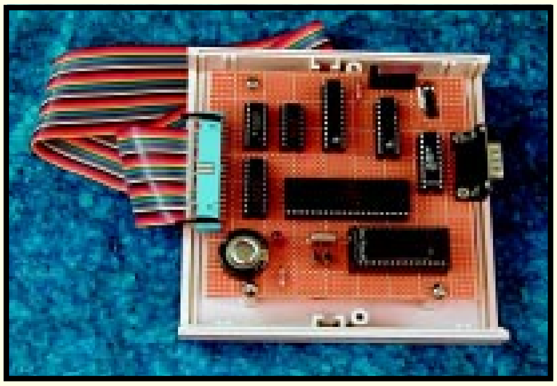
The Witness
$5000 Prize

Similar to the "black box" on airplanes, the Witness keeps track of valuable information that can be used as evidence or in accident reconstruction. The Witness stores the information as time-coded packets of data into a battery-backed SRAM. Each packet has a CRC for error checking. If the unfortunate happens and an accident occurs, the device would be retrieved and the data would be transferred to a PC software package. Using the information from the device, a computer-generated simulation of the accident can be created. Insurance companies, police departments, lawyers, and accident victims, could use the simulation as evidence to determine the cause of the accident.

When the car is turned on, the device begins checking the sensors every 100 ms. If the sensor data changes, a time-stamped packet is stored into SRAM. Sensor data would include accelerometers, headlights or blinkers, brakes, as well as side- or bumper-impact switches.

Nobody likes to think about getting in an accident, but if you do you'll be glad that you had the Witness with you (unless, of course, you're guilty!).

Travis Feirtag
Madison, Wisconsin
tfeirtag@etcconnect.com


The Witness

MICRO SERIES

Mark Balch

High-Definition TV

MPEG-2 Transport and ATSC Data Infrastructure

Part 2

MGT, PMT, VCT.... As Mark shows us, there's now much more to TV signals than just NTSC. In the second article of this series, he takes us through the details of HDTV's signal and its data structure.

Producing a high-definition television (HDTV) program in a studio is just the first step in the overall process of broadcasting those images into living rooms across the country. The first article in this series discussed the organization of the raw HDTV signal and its associated data structures.

For the last half-century, broadcasters have transmitted their TV images by modulating RF carriers assigned by the FCC with the analog NTSC signals they created in their studios. The FCC allocated a fixed 6-MHz portion of the RF spectrum to each local TV station for that specific purpose.

In the realm of digital television (DTV), the FCC decided to stick with that bandwidth allotment and is granting new 6-MHz slots to local TV stations as they transition to digital broadcast. (At the end of the transition period in 2006, stations will have to relinquish one of their two frequency assignments.)

As mentioned in Part 1, the term DTV does not directly equate with HDTV, but simply refers to the MPEG-2 compressed digital transport and formatting of television pictures and their associated data. DTV is extremely flexible because it gives broadcasters a fixed chunk of bandwidth to organize in almost any way they wish. There are stations today conducting multi-channel DTV broadcasts of both HD and standard definition (SD) material.

Given this digital content, RF engineers worked out a digital modulation scheme referred to as 8-VSB (vestigial side band) that allows 19.39 Mbps to be transmitted within the assigned 6-MHz channel bandwidth. Recall that the raw HDTV signal occupies nearly 1.5 Gbps of bandwidth—not quite a good match for a data channel with less than 20 Mbps of capacity! The requirement for some form of data compression was clear and it was decided that the MPEG-2 video compression standard was well suited for the task. Therefore, once an HDTV program has been produced and is ready for broadcast, the raw signal is fed into an MPEG-2 encoder where it is compressed by a factor of roughly 80:1. In his article a few years ago (*Circuit Cellar* #86), Do-While Jones provided us with a discussion of MPEG-2 compression.

To be properly displayed on a TV, the compressed bitstream requires additional data resources to enable the decoder to reconstruct the original timing of the video signal and to navigate and select the correct bits within the incoming 19.39 Mbps broadcast datastream. Although these topics are complex enough to fill volumes, this article will provide an overview of these data structures and

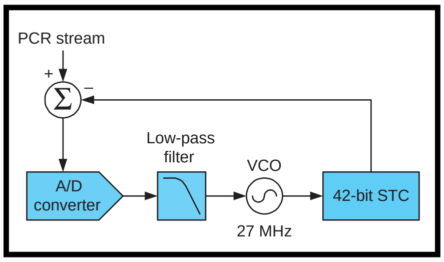


Figure 1—*The MPEG-2 STC is recovered at the receiver by means of a PLL. The phase reference and feedback terms are digital count values that are compared and converted into an analog signal that drives a voltage-controlled oscillator. As in many PLLs, a low-pass filter is useful to stabilize the loop's response.*

explain how they work together to enable a complete DTV broadcast.

TIMING AND SYNCHRONIZATION

Motion video is inherently real time in its nature. For a given video format, each pixel must be excited at a predetermined time, each line must have a fixed duration, and frames must be displayed at regular intervals. If these critical timing parameters differ from source to destination, the result will be a corrupted image.

One of the most critical functions of the MPEG-2 (ISO/IEC 13818) standard is to convey sufficient information to allow the decoder to accurately reconstruct the same clock that was used to encode the original video signal. With the decoder's internal clock running at the same frequency and phase as the encoder's clock, it can properly reverse the compression process and display a faithful representation of the original moving image.

MPEG-2 defines a system time clock (STC) that is 42 bits wide and comprised of two sections. The first component is the nine least significant bits that implement a modulo-300 counter clocked at 27 MHz, which is the fundamental MPEG-2 system clock rate and is generally derived from the video signal by means of a phase-locked loop (PLL).

Because of the stringent timing requirements of video, this 27-MHz clock is specified with an end-to-end system tolerance of ±30 ppm. To guarantee this level of system accuracy across multiple pieces of studio equipment and inexpensive consumer decoders, the accuracy of the master clock generated by an MPEG-2 encoder must be significantly tighter.

The result of this modulo-300 counter is a 90-kHz increment rate that drives the 33 most significant bits—the second component of the STC. These 33 bits are used as a reference for the presentation times of various components of an encoded program.

During the compression process, the MPEG-2 encoder generates groups of bits that translate into visible pictures. The encoder is able to control the display of these bits on the TV screen by attaching a presentation time stamp (PTS) to them. When the decoder finishes processing one of these collections of bits, it holds the uncompressed result in its memory until the associated 33-bit PTS matches the decoder's 90-kHz STC component.

The STC is used for more than simply keeping video in synchronization with itself. Have you ever watched an old movie on TV where the speech was not properly synchronized to the movement of a actor's lips? Sections of audio are also accompanied by their own PTS values so they pass through speakers at the correct time. The general term for this process is audio/video synchronization (A/V sync).

There is yet another use for the PTS—synchronized data. It is conceivable that a data-enhanced DTV program could be created where arbitrary events occurred at specific times during the program (perhaps graphical displays or other effects).

So you see how important it is for the decoder to be able to operate using a faithful reproduction of the encoder's STC. But how can a 42-bit, 30-ppm, 27-MHz clock be duplicated by a decoder that may be miles away from the compressed signal's origin?

It is done by a combination of a PLL within the decoder and regular phase reference markers sent by the encoder. Each of these markers is referred to as a program clock reference (PCR). The MPEG-2 encoder takes a snapshot of its STC at least 10 times per second and records this value (embeds a PCR) in the outgoing compressed bitstream.

The decoder contains a 27-MHz voltage controlled oscillator (VCO) that drives a local STC. As PCRs arrive at the receiver, their values are compared to the local STC. If they do not match, the VCO's frequency is adjusted slightly so that, over time, the frequency and phase of the STC agree with the encoder's master STC.

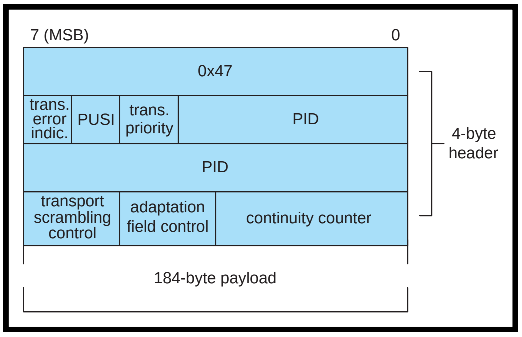


Figure 2—*MPEG-2 transport stream packets are 188 bytes in length and contain a four-byte header. A 13-bit PID uniquely identifies packets from different datastreams. The 0x47 sync byte allows a receiver to discover the beginning of packets by searching for 0x47 values spaced every 188 bytes.*

Figure 1 illustrates a generic STC recovery PLL within an MPEG-2 decoder.

TRANSPORT STREAM PACKETS

Once an MPEG-2 bitstream has been created within an encoder, it is broken up into chunks and inserted into 188-byte fixed-length transport stream packets. (The raw MPEG-2 bit stream is referred to as an elementary stream. Compressed audio and data are other examples of elementary streams.) Why not just send the elementary stream? Small, fixed length packets enable easier time division multiplexing of multiple datastreams onto a single medium.

As previously mentioned, more than just a single compressed bitstream comprises a DTV broadcast (additional video, compressed audio, and data structures are sent as well). At the same time, small, fixed-length packets can reduce the overall timing jitter that multiplexing causes (more on this later).

The structure of the MPEG-2 transport stream packet is shown in Figure 2. Notice that the header is four bytes long, leaving 184 bytes for payload—roughly 98% efficiency. The first byte is the sync byte and is assigned a fixed value of 0x47. The receiver uses this sync byte to lock to the incoming transport stream so it may begin to examine individual packets.

You can be sure that the value 0x47 will be present at other places in some packets because there are no restrictions on data values in the packet. Therefore, the receiver's locking algorithm must be sufficiently robust to avoid false detection of sync bytes. Clearly, if the receiver gets fooled into thinking a data byte is a sync byte, it will begin parsing packet headers that are actually payloads of other packets.

A robust locking algorithm can be implemented through the use of a counter to detect consecutive hits or misses of the sync value 0x47. Given the fixed 188-byte packet length, the algorithm is smart enough to know that 0x47 values spaced fewer than 188 bytes apart are data values. The receiver would begin looking for a 0x47 value and, when found, would wait 188 bytes. At this point, it would expect to see another 0x47 value. If so, the counter would be incremented. If not, the receiver would again begin looking for the sync byte.

When the counter reaches a certain threshold, the receiver would declare that it has achieved lock on the incoming transport stream. Using a method similar to this would allow the receiver to achieve lock relatively quickly on a real-world transport stream that contains many thousands of packets per second.

Following the sync byte are three flag bits. The first is the transport error indicator. This flag would commonly be set by a lower level data link over which the transport stream is actually being carried. A 1 value indicates that the current packet contains at least one uncorrectable error. Transport streams do not contain any header or payload error detection or correction information. This responsibility is left to the application-specific data link.

The second flag bit is the payload unit start indicator. The use of this bit is dependent on the specific type of payload carried within the packet. It is used to indicate the beginning of various data structures that are defined in the ISO/IEC 13818 standard. If the packet is carrying private data that is not defined under the standard, this bit is undefined.

Transport priority is the last flag bit. It was originally intended to serve as a means of communication with the data link in instances where the data link is overloaded and must drop a packet. Therefore, the idea is to label the most important packets with a 1 and all the rest with a 0.

In reality, this bit is largely ignored because data links that carry MPEG-2 transport streams almost always have a sufficient allocated bandwidth. In rare instances where packets are dropped, the drops are either not at the discretion of the transmitter (i.e., they result from physical signal problems), or the transmitter simply does not have the intelligence to store packets and provide special handling for those with the transport priority bit set.

The next 13 bits form the important packet identifier (PID) field. Each PID value denotes a separate collection of transport stream packets. For example, the packets that comprised a single compressed video stream would all be labeled with a unique PID. A different PID would be used to mark the associated audio for that video.

The receiver uses the PID field to filter different elementary streams and send them to the proper processing units. Therefore, PIDs for a given elementary stream must be unique within their transport stream. Otherwise, you might get video and

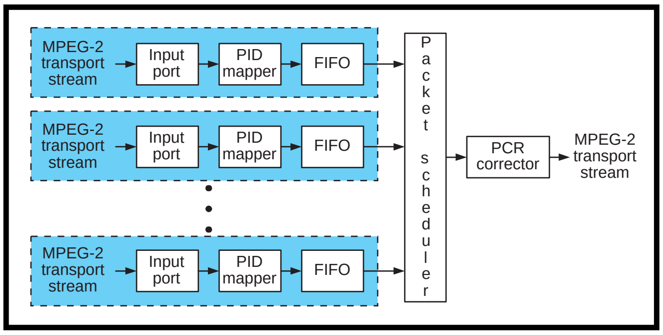


Figure 3—*An arbitrary number of transport streams can be multiplexed into one stream as long as their aggregate bandwidth does not exceed that of the final stream and there are sufficient unique PIDs to mark all of the incoming packets uniquely in the outgoing stream. PCR correction ensures that PCR-containing packets that are forced to wait in an input FIFO for their transmission timeslot are adjusted to account for this delay instead of introducing excessive jitter into the PCR streams.*

audio data mixed up and lose the ability to properly decode either. There are some reserved PIDs that may not be used by generic applications. These are 0x0000–0x000F. Some of these reserved values are used to convey necessary data structures that will be discussed later.

The PID value 0x1FFF is used to uniquely identify null packets. In cases where a packet must be transmitted, but there is no useful information to send, a null packet would be generated and marked with a PID value of 0x1FFF. This is not uncommon because data links that carry compressed video are often constant bit-rate interfaces (19.39 Mbps for U.S. terrestrial DTV broadcasters), which means packets must be sent regardless of the need for them. Various complexities and inefficiencies in real-world systems can result in the generation of null packets.

Three more data fields round out the transport stream packet header. Transport scrambling control indicates the scrambling mode of the packet payload. When set to 0x0, the payload is unscrambled. Other data values have application-specific definitions.

Adaptation field control indicates the contents of the packet payload. Adaptation fields convey special information such as the PCR. An adaptation field control value of 0x3 indicates that the payload contains an adaptation field followed by other data. A value of 0x2 indicates that the payload contains only an adaptation field. A value of 0x1 indicates that the payload contains only other data. The 0x0 value is reserved for future definition.

The remaining four bits of the header are defined as a continuity counter that is incremented for successive packets which contain the same PID. This counter allows the receiver to make a determination as to whether a packet has been lost or if a duplicate has been received. A lost packet would be indicated if, for example, two consecutive packets with the same PID were received with continuity count values 0x6 and 0x8. A duplicate packet would be indicated if the continuity count for two consecutive packets was the same value.

Of course, this protection is not perfect because if 15 or 16 consecutive packets containing the same PID were lost, the receiver would not properly detect these events based on the continuity counter state.

TRANSPORT STREAM MULTIPLEXING

All complete MPEG-2 transport streams are multiplexed in some fashion (i.e., they all contain packets from multiple elementary streams with multiple PIDs). The simplest program consisting of just video and audio would require a transport stream to contain multiple unique PIDs. Therefore, all MPEG-2 broadcast systems contain at least one multiplexing element.

The basic job of a transport stream multiplexer is to combine multiple transport stream packets from multiple sources into a single transport stream on the same outgoing transmission channel. These packets need to be time-division multiplexed because two pieces of data cannot coexist at the same instant in time in a single transport stream. Therefore, incoming packets need to be buffered until the next free transmission slot opens up for them.

Additionally, the restriction of unique PIDs within a single transport stream must be adhered to so unique elementary streams will not be corrupted. This requirement creates the need for a PID mapping function. Incoming packets may need to have their assigned PIDs mapped to new values prior to transmission.

The architecture of a basic MPEG-2 transport stream multiplexer is illustrated in Figure 3. Basic elements such as input FIFO buffers, PID mappers, and a central scheduling module are shown.

The scheduling module is responsible for deciding which input source feeds the output during any given transmission slot. Each slot lasts for 188 bytes—the length of all transport stream packets. Scheduling can be done in a variety of ways. It can be on a first-come-first-served basis or on an allocated-bandwidth basis where specific inputs have priorities based on their allocated percentage of the output data rate.

Scheduling implies that some incoming packets will have to wait before being transmitted. It doesn’t take long to realize that this wait will vary depending on the instantaneous traffic entering the multiplexer. Therefore, the multiplexer will inherently introduce a certain amount of jitter into the overall system.

Constant delays in an MPEG-2 system are not a problem because all data elements are offset by the same unit of time and therefore have a zero relative offset. However, variable delays can create problems, especially with regard to the sensitive PCRs.

Excessive PCR jitter can cause the receiver's clock to lose lock. In an extreme case, this would be visible to the TV viewer as a corrupted image. Therefore, a well-designed multiplexer will be able to compensate for this variable delay by correcting PCRs as they flow through the system.

Over small lengths of time, the STC increments continuously. Therefore, if the multiplexer keeps track of how long a PCR-containing packet has been stored in the system, it can add this time offset to the PCR value to nullify the delay.

When this operation is performed on all PCRs in a stream, the effect of the variable delay is negligible. Note that it is still desirable to minimize the variable delay through the multiplexer, and hence the PCR correction applied, to reduce exposure to discontinuities in the STC arising from splices or other events.

PROGRAM-SPECIFIC INFORMATION

So far, I have discussed the basic structure and functionality of the MPEG-2 transport stream with references to the data structures that allow sense to be made out of all the seemingly random bits.

The MPEG-2 standard defines several basic tables that provide the minimum necessary navigation of the transport stream. These tables are collectively referred to as program specific information (PSI). Other standards bodies geared to specific implementations of MPEG-2 have created their own ancillary data structures to suit their own purposes.

The Advanced Television Systems Committee is one notable example that will be discussed later because of its direct relevance to U.S. terrestrial DTV broadcast. Another is the Digital Video Broadcasting (DVB) consortium whose data structures and standards are in widespread use throughout the world.

It is important to realize that all such data structures are fairly complex in their nature and application, and cannot be covered in complete detail in this article. For complete information, consult the

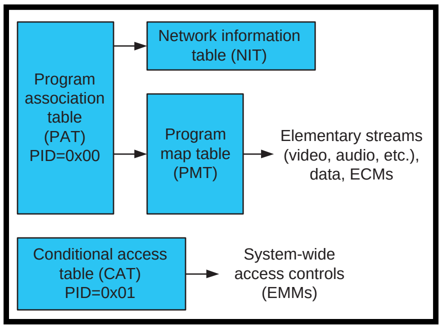

Program association table (PAT) PID=0x00
Network information table (NIT)
Program map table (PMT)
Elementary streams (video, audio, etc.), data, ECMs
Conditional access table (CAT) PID=0x01
System-wide access controls (EMMs)

Figure 4—*The four basic PSI tables enable a receiver to discover the contents of an MPEG-2 transport stream so separate datastreams (video, audio, etc.) can be extracted and processed. This information is hierarchical in its organization to allow for the distinction of separate programs, each with its own component datastreams.*

standards documents themselves.

MPEG-2 PSI consists of four tables in the original standard: the program association table (PAT), the program map table (PMT), the conditional access table (CAT), and the network information table (NIT).

The PAT is the central defining table that serves as the receiver's index to the incoming transport stream. As such, it is assigned the reserved PID of 0x0. When the receiver is first turned on, and in the absence of any descriptive information about how to parse the incoming transport stream, it knows that it can look at the payload of packets with PID=0x0 and obtain the necessary information to begin parsing the complete transport stream. As its name implies, the PAT associates program numbers with their PMT. Therefore, the PAT is a list of program number and PMT PID pairs along with certain overhead information.

The program number is a unique 16-bit field whose value is arbitrarily assigned by the broadcaster. For example, it may correspond to an indicated channel number on the TV.

Each program has an associated PMT section that may reside on its own unique PID or share a PID with other sections. The PMT provides the mappings between program numbers and the program elements that comprise them. For example, a common video program would contain separate video and audio elements whose unique PIDs and presence would be called out in the program's PMT. The PID associated with the program's PCR is also signified in the PMT.

Unlike the other three PSI tables, the NIT is largely undefined by the MPEG-2 standard, its contents are private (application specific) and its presence is completely optional. About the only standard provision for it is a reserved notation in the PAT (i.e., if present, its PID is called out). This seemingly odd decision was made because it was recognized that the carriage of basic information specific to the distribution medium would be useful, yet that information would be so network-specific that a common data structure for it would not be possible.

Figure 4 illustrates the relationships and common uses of the standard MPEG-2 PSI tables.

CONDITIONAL ACCESS

The CAT is the fourth PSI table and represents an important part of most subscription video broadcast systems. However, before discussing the function of the CAT, it is first necessary to discuss the concept and basic operation of conditional access (CA).

Many video broadcast systems are offered to viewers on a subscription or pay-per-view basis. Consider the monthly cable bill and the additional pay-per-view channels that are continuously available. Clearly, the broadcaster is trading the right to view a particular program for a monetary fee.

To guard against unauthorized viewing of a particular program, the broadcaster usually implements some form of security to prevent those who have not paid the required fee from watching that program. CA is the general process by which a broadcaster gives a specific receiver the rights to display a program and the mechanisms that enforce those rights.

In the context of DTV, conditional access is implemented by scrambling the MPEG-2 transport payloads and distributing keys to the appropriate receivers in a secure fashion. The general idea is to enable the broadcaster to send the correct descrambling keys to those customers who have paid for them. A customer's receiver would then use these keys to descramble the compressed bitstream and decode the program normally.

The heart of this methodology is obviously the secure distribution of the keys. These security and authentication mechanisms can be extremely complex and are proprietary systems. However, the means of communicating much of this proprietary information has been standardized to enable different vendors to operate using a common infrastructure with known interfaces.

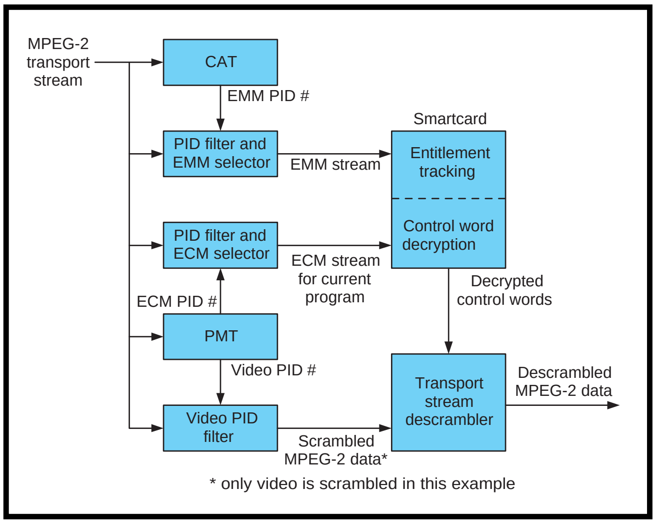


Figure 5—*The conditional-access system architecture in an MPEG-2 system enables the viewing of scrambled programs by distributing encrypted descrambling keys (control words) along with entitlement messages. A secure decryption engine at the receiver (generally implemented within a smartcard) uses this information to pass or reject control words, thereby controlling access to the scrambled program.*

Each receiver in most DTV broadcast systems that implement CA contains a small, secure microcontroller embedded into a card roughly the size of a credit card. This removable microcontroller is called a smartcard. Each smartcard has a unique identifier embedded within it containing proprietary decryption algorithms that operate using this unique ID.

The first step in allowing a particular receiver to display a scrambled program is for the broadcaster's CA system to tell the receiver's smartcard that it has rights to that program. Entitlement management messages (EMMs) are broadcast messages that do just that.

Each program has an EMM stream associated with it (indicated by the CAT). If present in a transport stream, the CAT is always found on PID 0x0001.

An EMM contains a proprietary message to each smartcard in the broadcaster's whole CA system. Therefore, as the number of subscribers grows, the size of the EMM stream increases. An individual receiver picks out the EMM that is specifically addressed to it and forwards the contents to its local smartcard. Once the smartcard receives an EMM, it updates its internal permissions if the message is valid.

Entitlement control messages (ECMs) contain the encrypted control words (keys) used to descramble transport payloads. ECMs begin streaming when their associated program begins. Just as the PMT associates video and audio packet PIDs with a program number, the PMT also indicates a program's ECM PID.

When the viewer tunes to the scrambled program, the associated ECMs are forwarded to the smartcard. Based on previous EMMs received, the smartcard attempts to decrypt the control words. If successful, it returns those control words to the receiver as descrambling keys. At this point, the receiver is able to use these keys to descramble the payloads of the incoming video and audio transport packets and then decode them to yield a viewable program. Figure 5 illustrates the basic CA system's components as they relate to one another.

Encryption and scrambling are two different processes. Encryption is more computationally demanding than scrambling and is suited for relatively small, discrete data. Scrambling can be performed on the fly more easily to streaming data such as an MPEG-2 bitstream. Encryption generally provides a higher level of security at the cost of increased processing power. This is why the individual control words or keys, are encrypted, while the MPEG-2 data is scrambled.

To strengthen the system against someone applying brute-force methods to determine the control words and thereby gaining unauthorized access to the scrambled program, some powerful CA systems update the control words as often as every few seconds. This means that, should you apply enough computing power to crack a descrambling key, it will only provide you with a second or so of video before having to crack a new key.

Given the frequent updates of control words, the tempting piece of the CA chain to try to break is the smartcard itself. Although no invention should claim to be perfectly tamper proof, it should be obvious that cracking these new digital CA systems is orders of magnitude more complex and expensive than the commonly available analog cable TV descrambling boxes.

ATSC PSIP

The structure of terrestrial DTV broadcast in the U.S. is largely defined by the Advanced Television Systems Committee (ATSC), an industry standards group whose membership includes both broadcasters and equipment manufacturers. Many of the ATSC's recommendations have been adopted by the FCC and are therefore mandatory in the U.S. One of these adopted recommendations is the Program and System Information Protocol (PSIP) called out in the ATSC standard document A/65.

PSIP builds on MPEG-2's PSI resources by providing additional detailed information about the programs contained in the transport stream. Therefore, the PAT, PMT, and CAT are present along with the new tables defined by the ATSC. The PSI provides detailed mappings of related datastreams and PIDs, but it's PSIP that enhances the program tuning process and defines a method for program schedule, rating, and description.

Six basic table types compose PSIP. Four of these are base tables that share the same PID, 0x1FFB. They are the system time table (STT), the master guide table (MGT), the rating region table (RRT), and the virtual channel table (VCT). Similar to the way in which the PAT directs the receiver to the PMT and NIT, the MGT provides the lengths and PIDs for all PSIP tables with the exception of the STT.

The two other table types have multiple instances depending on broadcast needs and are carried on arbitrary PIDs called out by the MGT. These two tables are the event information table (EIT) and the extended text table (ETT). Figure 6 illustrates the relationship of these PSIP tables.

As its name implies, the STT communicates the current date and time to the receiver. This time is expressed in global positioning system (GPS) terms—a 32-bit count of the number of seconds since 12 AM, January 6, 1980. The receiver converts this GPS time into local time after it determines the time zone in which it is located.

When the receiver is first turned on, its location would typically be programmed by the consumer. Additionally, daylight savings time change information is passed in the STT to ensure that all receivers change their internal clocks at the same time.

You may know that just as movies have ratings, TV shows have ratings as well. DTV offers the ability to convey detailed rating information for each program and enables consumer electronics manufacturers to add parental controls to DTV products.

The RRT exists to carry these program ratings. It associates a geographical region of the world with an arbitrary rating scale that is specific to that region. Descriptive text such as the name of the region and the names of the different rating levels are also provided. Once this information has been processed by the receiver, it may be referenced by content advisory descriptors present in the EIT (more on this later).

The VCT lists various attributes for programs carried within the MPEG-2 transport stream and enhances the basic program "tuning" capabilities of MPEG-2 PSI. Programs are referred to as virtual channels because of the broadcasters' desire to retain the familiar concept of a "channel" representing a single program. With NTSC, viewers are accustomed to tuning in to various channels and finding a series of individual programs on each of the channels.

Because DTV gives a broadcaster the ability to deliver multiple programs within the same physical channel (single carrier modulated over a 6-MHz bandwidth), there's the need to refer to each program as its own virtual channel. To this end, the concept of major and minor channel numbers was created.

The major channel number maps to the physical range of RF spectrum that is carrying the complete transport stream. The minor channel is a second level of hierarchy that is assigned arbitrarily by the broadcaster to represent individual programs within the major channel (transport stream).

The ATSC actually created two similar versions of the VCT. The Terrestrial VCT (TVCT) contains basic information fields relevant to over-the-air broadcast. The Cable VCT (CVCT) contains additional information fields relevant to the broadband cable medium.

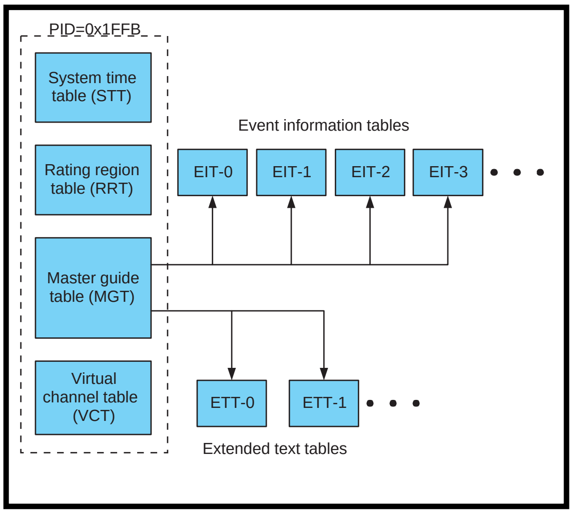


Figure 6—*The ATSC's PSIP tables complement MPEG-2's PSI by providing programming information, including text descriptions, rating information, and channel mapping. Each U.S. terrestrial broadcaster emits its own PSIP information as part of its MPEG-2 transport stream.*

DTV PROGRAM SCHEDULES

Like MPEG-2's PSI tables, the STT, MGT, RRT, and VCT provide information that is primarily used by the receiver to navigate the multiprogram transport stream that is sent from the broadcaster. The two remaining PSIP tables contain information that is primarily for the viewer. These tables carry both program schedule information and text descriptions of what the programs actually are.

The event information table (EIT) carries basic program information such as title, start time, and duration. A program is conventionally thought of as a TV picture with sound, but DTV programs don't have to conform to this mold. A program could be a data service that may not contain any video or audio. There are multiple instances of the EIT in ATSC broadcasts (a minimum of four and a maximum of 128 at any one time).

Each EIT instance is mandated to have a fixed duration of 3 h and starts on standard 3-h boundaries: 0:00 (midnight), 3:00, 6:00, 9:00, 12:00 (noon), 15:00, 18:00, and 21:00. The EIT instances are numbered upwards from 0 with EIT-0 being the event information for the current 3-h time segment. EIT-1, EIT-2, and EIT-3 are also required and provide information for the next three consecutive 3-h segments.

At the end of a 3-h segment, EIT-0 becomes obsolete, the other EITs move up in time and a new EIT is added to the transport stream to meet the minimum requirement of four EITs. Beyond the four mandated EITs, longer range event information is at the discretion of the broadcaster.

In the case where a broadcaster wishes to send a lengthier text description of a program, the extended text table (ETT) is utilized. The presence of ETTs is completely optional. They simply carry additional text information above and beyond those carried in other tables (e.g., program plot synopsis, actors' names, etc.).

All of this textual program information can be used in a variety of creative ways by DTV consumer electronics manufacturers. Whether in standalone set-top boxes (STBs) or in digital televisions, DTV receivers incorporate microprocessors and graphics hardware that far exceed the capabilities of most NTSC televisions and receivers today.

Electronic program guides (EPGs) have become the norm for digital broadcast satellite customers whose STBs already have this type of internal enhancement. With the proliferation of DTV broadcast, EPGs will become available to all television viewers and potentially change the norms of TV viewing. Applications range from simple EPGs that can be browsed with a remote control, to targeted ads depending on the program being browsed, to interactive guides that let the viewer search for specific types of programs.

THE DTV STATION

By now you can see that there's much more to a DTV broadcast than MPEG-2 compressed bits. Timing synchronization allows the remote decoder to work in lock step with the encoder. Transport multiplexing provides the flexible end-to-end carriage of multiple bitstreams. And the PSI and PSIP tables enable not only the viewing of the compressed program, but also a wide range of other applications. Many of these potential applications are still mysteries to both consumers and the industry itself.

The next, and final, article in this series will explore the system view of DTV terrestrial broadcast in the U.S. Many pieces of equipment are used to create and transmit the various components of a DTV broadcast. We'll discuss DTV station architecture, the operation of DTV equipment, and potential applications for this powerful new television infrastructure.

Mark Balch is a senior hardware design engineer and has participated in a variety of MPEG-2 product designs, including an HDTV MPEG-2 encoder. Mark actively attends meetings of the ATSC and SMPTE industry standards groups. You may reach him at mark_balch@hotmail.com.

REFERENCES

ISO/IEC 13818-1:1996(E), Information technology – Generic coding of moving pictures and associated audio information: Systems

ATSC A/65, Program and System Information Protocol for Terrestrial Broadcast and Cable, 1997.

www.atsc.org
www.fcc.gov
www.mpeg.org
www.smpte.org

Without Acceleration

FROM THE BENCH

Jeff Bachiochi

Part 2: Good (and Bad) Vibrations

In Part I, Jeff constructed his portable 3-D accelerometer. In Part 2, he adds a radio packet controller for data transmission and other hardware refinements that make the design more usable.

Remember last time when I discovered through charting my 3-D accelerometer that walking had a more exciting signature than a ride on the elevator here at *Circuit Cellar* World Headquarters? Well, I promised a couple of changes to my project's circuitry and this month I intend to keep that promise.

The part I like best about presenting these goofy projects is that they often become an exercise in multiple subjects. This project is a case in point.

No matter how good an idea looks on the surface, it isn't until you've actually used it that you find inadequacies and inefficiencies. This goes for tools, kitchen appliances, toys, you name it. If you're like me, every time I pick something up I'm thinking about ways it could be improved.

Well, two things come to mind with this project. I was forever twiddling the six pots in this design. Every time I changed gains, I twiddled. And every time I twiddled, I unintentionally twiddled the wrong one, leading to even more twiddling.

This month I want to twiddle them right out of the design. To accomplish this I'll use digital pots made by Dallas Semiconductor, and set these using only two outputs from the microprocessor.

The second faux pas was having to lug my laptop around, or drag a very long umbilical cord over which the data gushed. Moving data at this rate (19.2 kbps) required more than the everyday 2400-bps RF link.

Fortunately, by adding dollars you can easily move upscale. Although a transceiver is overkill for this project, I had this pair kicking around and use them often because they are so reliable and *almost* user-friendly.

DIGITAL POT

Dallas wasn't the first company I looked to for digital pots to use in this project. However, they make a single, dual, quad, and six-pack of pots (in a single package) controlled through an I²C interface (two I/Os).

I needed two different values, so I chose to use two quad packages. The pots do not have a nonvolatile memory so they do not remember where they were, but power up set to the midpoint of their range.

One package we'll use to set the gain of the external amplifier and the other we'll use to cancel the offset of each sensor's output. You may recognize the schematic in Figure 3 from last month. This time it has the digital pots instead of the manual trimmers.

And if you compare Photo 1 with the photo in Part 1 (*Circuit Cellar* 112), you'll also see why I put the pots on a separate PCB. The digital pots plug in, replacing the mechanical

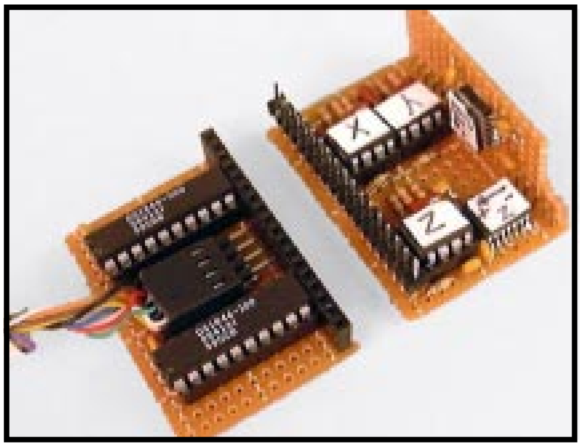


Photo 1—*The 3-axis accelerometer uses Dallas solid-state pots to set gain and offset trim and help to automate this project.*

ones, keeping the same exterior dimensions and avoiding a rewire of the sensor PCB.

By far the easiest implementation is with the *gain* pots. Setting these is based on the gain selected via two input pins. These two pins are sampled during the initialization of the PIC16C74 microcontroller. Each of the four combinations of inputs will load the appropriate wiper position value into a gain register.

The wiper position value (6 bits) is based on the tap resistance of the pot. For the 50-Ω pot, each tap is 1/64 of the total resistance or 781.25 Ω (50000/64).

Depending on the gain selected at the inputs, a data constant is sent over the I^{2}C interface to three of the four digital pots (the fourth isn't used). Although the 16C74 does have an I^{2}C port, it only supports slave operation in hardware.

Therefore, as a master, I must bit-bang the SCL and SDA lines through software. I can set all three gain pots with a single I^{2}C operation (back-to-back register writes).

Now that the gains have been set, something must be done about that nasty offset. As we saw last month, it is most important for the higher gains. Remember, there is always 1*g* pulling us toward the center of the earth.

If a sensor's element is aligned (even slightly) with the Earth's gravity, this *g*-force will create an output. This output will be easily confused with the natural (*0-g*) offset we are trying to eliminate, so sensor alignment is critical during offset compensation. Because the 3-D sensor is moveable, but the micro can't tell which sensor is on or off axis, the user will manage this important task.

To get me in sync with the microcontroller, I use LEDs and push buttons. The LEDs are used by the microcontroller to signal me, while the push buttons are used by me to signal the micro.

Once the gains have been set on all three amplifier channels, both LEDs turn on, signaling that the micro is ready for offset compensation calibration. The first compensation will be performed on the *x*-axis.

I must hold the 3-D module so that the *x*-axis is perpendicular to gravity. That way, gravity will have no effect on its output. When positioned correctly, I press the *x* button to let the micro know it may proceed with the compensation. The microcontroller turns off an LED and sends the *x*-offset digital pot a midpoint setting via the I^{2}C bus. This produces some sensor output, which is filtered and amplified creating a final output voltage.

As you can see from Figure 2, each sensor/signal conditioner channel has its own A/D input on the microprocessor. The micro now selects the appropriate A/D channel and begins a conversion.

When the conversion is complete, the 8-bit value is compared to 128 (which is what the output should be at 0 *g* without any offset), If the conversion is 128, then everything is perfect and the micro turns the LED back on, and this process is repeated for the remaining two axes (*y* and *z*).

However, if the conversion is less than 128, the micro sets the "less than" compensation flag and bumps the *x*-offset digital by one bit. Again, an A/D conversion is performed and the value is compared to 128.

This compensation adjustment loop continues until the conversion value is equal to or greater than 128. Bumping the offset by 1 bit and resampling the output allows the micro to automatically tweak the offset pot in an attempt to reach an output of ½ V_{cc} (128 conversion value) or as close as possible within the resolution of the digital pot.

Similarly, if the initial compensation conversion is greater than 128, the "greater than" flag is set and the compensation adjustment loop continues to reduce the output until it is equal to or less than 128.

If the I^{2}C transmissions are not acknowledged or the digital pot is adjusted to the end of either stop, the processor jumps into an error mode causing the LEDs to blink. Once all three sensors have been compensated, execution proceeds to the work phase where all three channels are sampled and the conversion values are reported every 10 ms. You can refer to Part 1 to get the lowdown on how the processor samples and converts the sensor outputs into ASCII data.

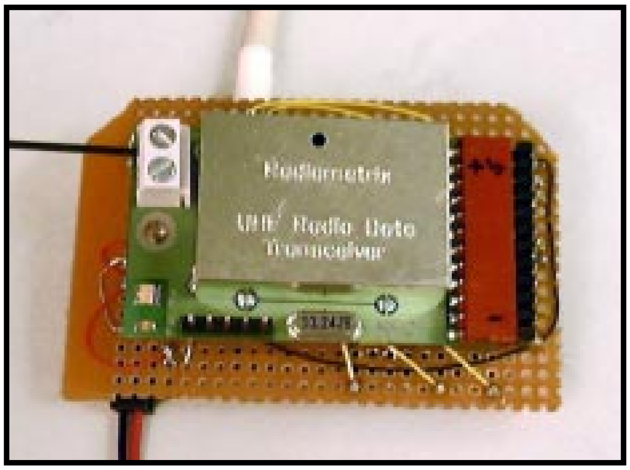


Photo 2—*The radio packet controller plugs on to the back of the 3-axis accelerometer controller circuit shown in Figure 2.*

RADIO PACKETIZING

As presented last month, the output from the project was 19.2 kbps ASCII serial data. I rewrote the output routine to send parallel nibble data, instead of the serial bit stream. This change allows the project to interface directly to an RPC (radio packet controller) as you can see in Photo 2.

The RPC is a complete RF transceiver. It can accept up to 28 bytes of data (including a length byte). The RPC accepts nibble transfers using four hardware handshaking control lines (two for transmitting and two for receiving). Its bidirectional nibble bus is quick.

Once the data has been transferred to the RPC, it modulates its RF output with a preamble string of alternating 1s and 0s followed by a frame sync byte. The actual data bytes are expanded into 12-bit symbols to ensure that each symbol has a 50-50 balance of ones and zeros and never more than four zeros or four ones in a row. Balanced data is required when pushing some receiver designs to the limits. The packet ends with a checksum byte (also expanded to 12 bits).

At the receiving end, if the packet is not 100% received with a correct checksum, it is thrown out. No data is preferred over bad data. The receiver in Figure 1 shows an RPC to RS-232 converter.

It does no good to send RF data if there is no way to retrieve the data. Remember, the RPC is a nibble parallel interface and while it may be interfaced a bit more easily to a PC's parallel port, there would be a lot of programming needed at the PC end to be able to use it. An RS-232 interface allows it to be used on any computer with the standard terminal emulation software.

RF 2 RS-232

For this end of the project, I thought double buffering would be a good idea, meaning that once the serial input buffer is full, the contents get moved to the parallel output buffer. And, vice versa, once the parallel input buffer fills, buffer contents are transferred to the serial output buffer. Transferring the data in this fashion will allow more data to continue to be received (presumably) without any need to wait.

Of course, this strategy also assumes there is no transmission error, no acknowledgement, and no retransmission. The host PC would have to handle packet acknowledgement if it is required .

Let's take a look at the RPC to micro interface. The RPC is normally in listen mode searching for a preamble signal. This signal allows the RPC to sync its recovery clock to the incoming data.

Once a frame sync is received, the RPC moves into the data decode state. Because the first data byte sent is the byte count, the RPC can determine the end of packet and test for proper checksum. If the packet was received correctly, the RPC will now signal the host for a data transfer by lowering the RXR control line (data request).

The host in this case is a PIC16C63. When the micro sees the RXR line low, it prepares to read data via the nibble bus by setting the four I/Os to inputs and lowering the RXA control line (request accepted). The RPC places a nibble on the bus and raises the RXR line (data ready).

And finally, the micro reads the data nibble and raises the RXA line (data read). This sequence is repeated until all the data has been transferred to the micro one nibble at a time.

As the nibbles come into the micro, they are assembled back into their respective bytes and are placed into the parallel input buffer.

In an attempt to keep things simple, the data is not passed directly to the UART for transmission because the RPC-to-micro transfer would be held up while waiting for each character to be transmitted (RXREG empty).

Instead, once the whole data packet has been received from the RPC, it is transferred to the serial output buffer, where the micro can feed characters to RXREG at the UART's pace without interfering with any other function.

The first data byte is the byte count and is not sent to the UART with the data. The 'C63 takes care of counting serial input bytes so that the attached PC does not have to pad or unpad the data.

Transfer of data to the RPC is similar, except the micro has control. The UART will receive data via the RS-232 interface. No byte count is sent. Instead, the micro counts data

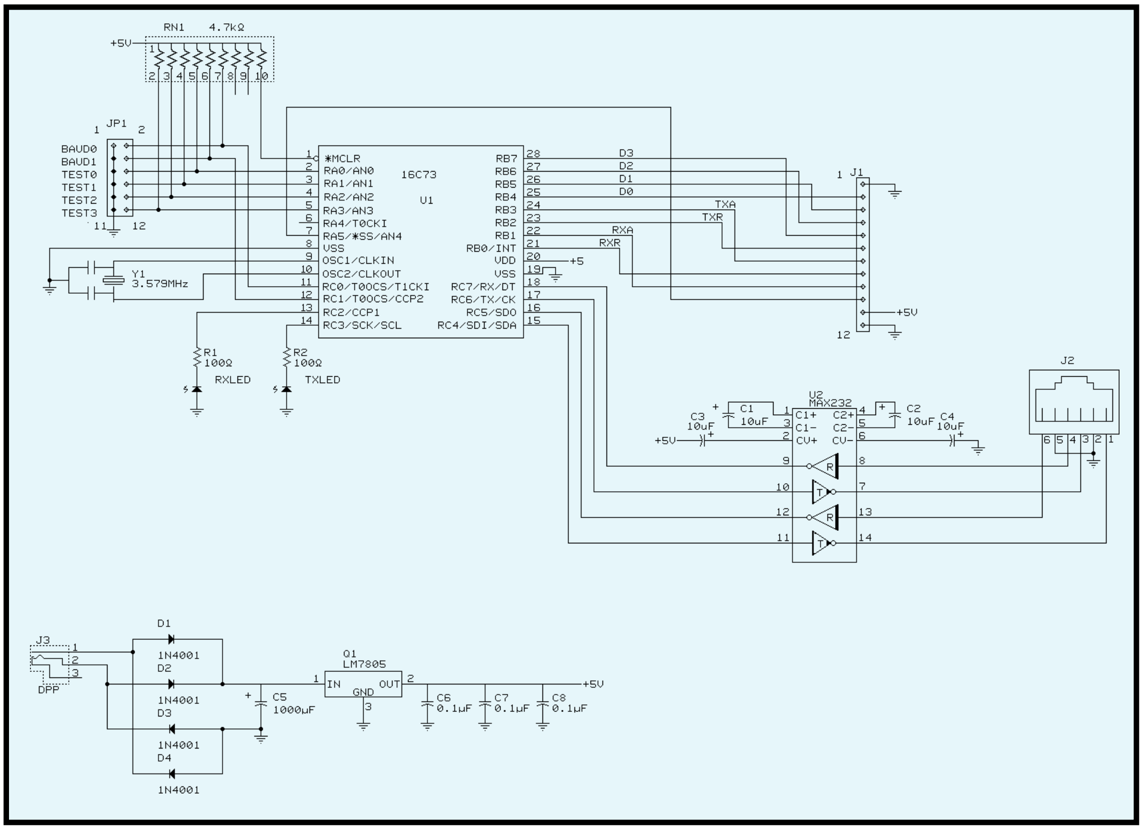


Figure 1— *This circuit creates an RS-232 interface to a Radiometrix RF packet controller. The PIC 16C63 double buffers data between a 19.2 kbps serial data stream and the RPC's nibble interface.*

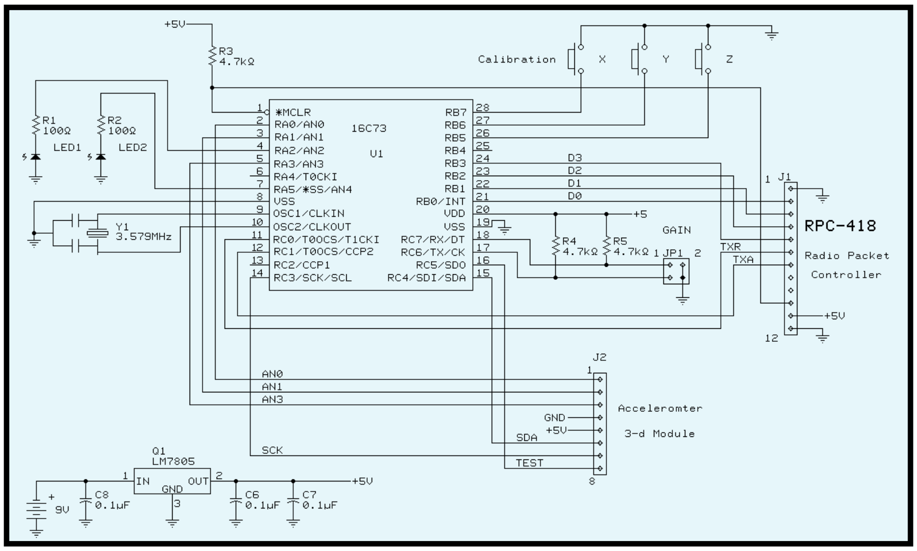


Figure 2—*Although Part 1 of this series used serial output, the 3-axis accelerometer controller in this RF version doesn't use the UART of the 16C73, so a 16C63 could be substituted in the design.*

bytes and when the serial input buffer is full (27 bytes or when a <CR> is received) the micro adds a byte count to the data and transfers it to the parallel output buffer.

Now the micro passes the data, a nibble at a time, to the RPC. The micro lowers TXR and waits for the RPC to reply by lowering TXA.

The micro sets the nibble bus's four I/Os to outputs, places a nibble on the bus, and raises the TXR. When the RPC has read the nibble, it raises the TXA line. This sequence is repeated until all the data has been transferred. The RPC then packetizes the data and sends it out if the receiver is not busy with a data reception.

RPC PARAMETERS

There are a number of parameters held in EEPROM on the RPC. These have to do with things like the preamble length, extended preamble, sleep, TX-to-RX delay, PWR-to-RX delay, TX backoff delay, slot delay, and reset state. See the manufacturer for more detail on these functions.

In addition to these parameters, the RPC has some built-in test modes. These were used for some stand-alone testing, which comes in handy when setting up your RF link.

The test modes are automatically entered on powerup if you have installed any of the TEST0–3 jumpers. The micro reads these jumpers on powerup and places the RPC into the associated test mode reflected by the jumper settings.

The most useful modes are Echo and Radar. As the name suggests, Echo mode retransmits any packet it receives. Radar mode periodically sends out packets and watches for their return. Although you can watch the TX and RX LEDs on the RPC modules, the 'C63 will signal a good reception by turning on its LED.

TIMING IS EVERYTHING

In this project, the data is 13 ASCII bytes (XXX_YYY_ZZZ<cr><lf>) from the three accelerometers sampled every 10 ms. For all of this to work, the samples need to be taken and the 8-bit digital conversion values need to be turned into ASCII data and shipped out every 10 ms. Certainly if the values were sent just as they are (three 8-bit values), the process would be much simpler, and the datastream much shorter.

Binary data (0–255) doesn't display as well as viewable ASCII data, so I decided to sacrifice simplicity for viewable data (much easier to debug). As we saw last month, the conversion and translation times were quick enough that the micro had plenty of time to ship out the 13 bytes at 19.2 kbps through its onboard UART.

This month's addition of the RPC modules replaced the UART with a

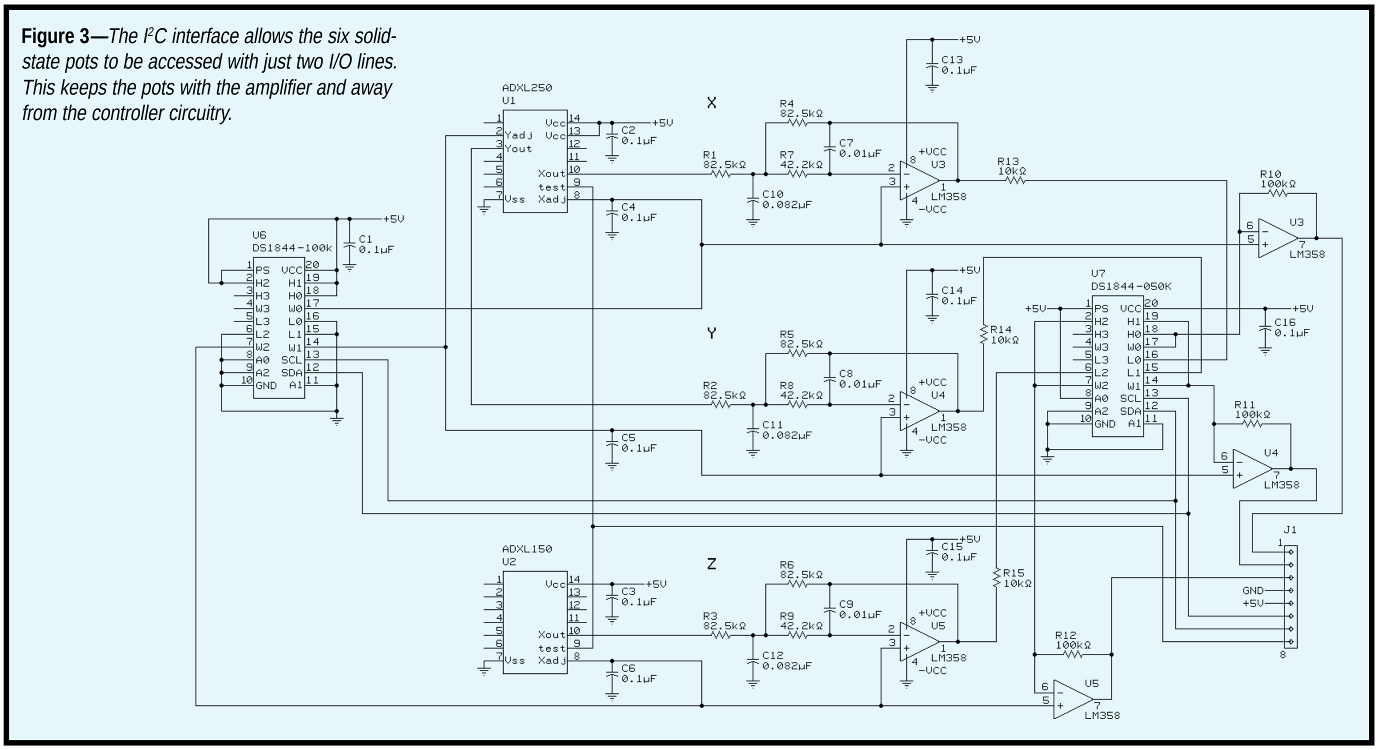


Figure 3—*The I²C interface allows the six solid-state pots to be accessed with just two I/O lines. This keeps the pots with the amplifier and away from the controller circuitry.*

parallel interface. The actual parallel transfer rate of this data from the micro to the RPC is about 25 µs/nibble (or 8µs/bit). This speed is about ten times faster than the 19.2-kbps rate it dealt with in the previous design (so there is no bottleneck here).

The RPC's RF transmission rate is 40 kbps. Transmission requires 200 bits of preamble, an 8-bit sync byte, 13 data bytes (each expanded to 12 bits), and one checksum byte (also expanded to 12 bits). That's a total of 368 bits (200 + [13 × 12] + [1 × 12]).

The 40-kbps rate yields a total transmission time of 9.2 ms (368 × 25 µs). Whew, that's less than 10% room to spare!

At the receiving end, things are a bit simpler. The RPC passes the data packet off through its parallel interface to the receiving micro at the same 8-µs/bit timing as its transmitter counterpart. The micro's UART dashes through the data in its buffer in about 7 ms (clearly not a bottleneck).

BAD VIBRATIONS

The road to Hell is paved with good intentions. The big wrap-up here was to be a trip to Riverside Amusement Park to give this project a real work-out. There was one tiny flaw in the day's schedule.

Because I would be placing the 3-D probe on my kids and storing the data with my laptop from afar, I didn't want to miss out on a full day of fun. So I decided to take a quick roller coaster ride. Riverside has two of the best wooden coasters in existence and I've ridden them for years.

But something different happened this time. My equilibrium was thrown for a loop (so to speak) and I emerged from the ride with beads of sweat running down my face. "Dad, you don't look so good," commented Kristafer, my youngest.

Needless to say, I was in no shape to work. For the rest of the day, I felt as though I had a whopping hangover and just wanted to curl up in a ball.

And so, I did not work. I did not play. I just tried not to spoil the day. What a shame, I love coasters.

I guess I'll be leaving the fun up to you. Even if you don't build this whole project, most of you will find some part of it useful. Be it accelerometers, real-time data collection, interfacing, serial/parallel conversion, or RF packet transmission, the whole is often not as intriguing as the value of its parts. ■

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

SOURCES

ADXL150/250
Analog Devices
1 Technology Way, PO Box 9106
Norwood, MA 02062
781-461-3060 • www.analog.com

PIC16C63/73
Microchip
2355 W. Chandler Blvd.
Chandler, AZ 85224
888-628-6247 • www.microchip.com

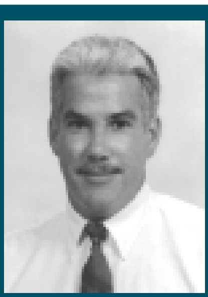
DS1844
Dallas Semiconductor
4401 S. Beltwood Parkway
Dallas, TX 75244
972-371-4000 • www.dalsemi.com

Radio Packet Controller
Radiometrics Limited
Hartcran House, Gibbs Couch
Carpenders Park, Watford
Hertfordshire, England WD1 5EZ
www.radiometrix.co.uk

SILICON UPDATE

Tom Cantrell

Test Driving a Merced with Pins



Ready or not, IA-64 architectures are headed down the pipeline. You won't find a Merced in your Timex this century, but Tom points out many new features that may be there in the next.

It's been cooler than normal here in Silicon Valley and the San Francisco Bay area isn't exactly tropical in the summer to begin with. All that fog and wind prompted Mark Twain to say something like, "The coldest winter I ever spent was the summer in San Francisco." Go to any of the SF tourist traps in the summer and be amused by the gaggle of visitors in their Bermuda shorts and tank tops freezing their buns off.

Fortunately, things warm up nicely as you head down the peninsula into Silicon Valley proper. In fact, if you hit Palo Alto and Stanford University in the middle of August, it gets downright hot because that's when the Hot Chips conference comes to town.

As usual, there was a lot of interesting technology on the table, stuff like SOC (System-On-Chip), CMP (Chip Multi-Processors), and embedded DRAM. However, instead of my usual roundup (and prompted by Intel's half-day tutorial on the subject), I'm going to focus on the chip on lots of people's minds—Merced, or "Itanium" in Intelese, which will be the first chip to incorporate the new IA-64 architecture.

Personally, I to stick with a PC until the march of silicon and bloatware renders it indisputably obsolete. Therefore, you won't find me lining up at the computer shop to be the first on my block to get a Merced-based PC.

So, why do hot chips like Merced and the others matter? *Circuit Cellar* isn't *PC Week* after all.

The point is, as I've said before (and feel I've been proven right), today's hot chip always yields insight into tomorrow's practical embedded chip, stripped of the gaudiest chrome and tail fins though it might be.

SMART COMPILER, DUMB PROCESSOR

For those of you not up on computer architecture, I strongly recommend *Computer Architecture: A Quantitative Approach* [1] by Hennessy and Patterson. Its 800+ pages shed light on just about everything 1s-and-0s, yet it is quite readable. I especially like the historical perspectives at the end of the chapters.

However, Hennessy and Patterson do give short shrift to the concept of long instructions (i.e., very long instruction word, or VLIW). Indeed, credit for the catchy heading above, which sums up its fundamental premise, goes to early '80s pioneers of the modern-day VLIW concept [2].

The 1985 doctoral dissertation written by Ellis is a good place to

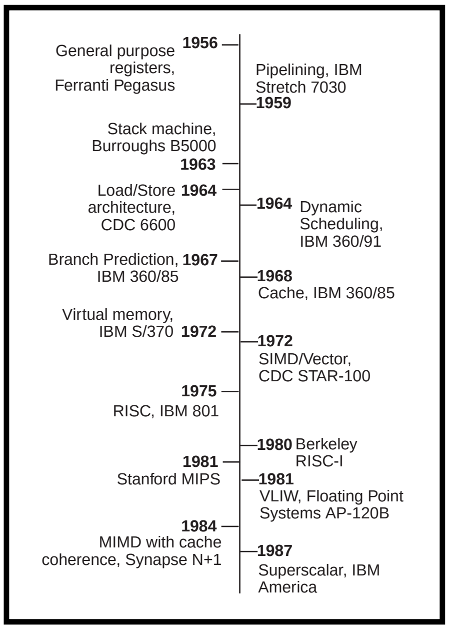

General purpose registers, Ferranti Pegasus 1956
Pipelining, IBM Stretch 7030 1959
Stack machine, Burroughs B5000 1963
Load/Store architecture, CDC 6600 1964
1964 Dynamic Scheduling, IBM 360/91
Branch Prediction, IBM 360/85 1967
1968 Cache, IBM 360/85
Virtual memory, IBM S/370 1972
1972 SIMD/Vector, CDC STAR-100
1975 RISC, IBM 801
1980 Berkeley RISC-I
1981 Stanford MIPS
1981 VLIW, Floating Point Systems AP-120B
1984 MIMD with cache coherence, Synapse N+1
1987 Superscalar, IBM America

Figure 1—*IA-64, like every computer back to the abacus, stands on the shoulders of its predecessors. However, relatively speaking, it represents a major shift in direction, and a major blessing for the VLIW concept.*

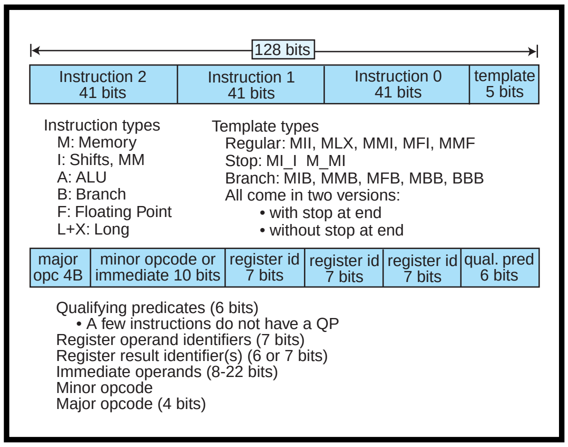


Figure 2—*Each IA-64 128-bit instruction bundle includes a "template" characterizing the type and parallel issue capabilities of the three 41-bit instructions. The instructions themselves are rather conventional with the notable exception that most feature conditional execution based on the state of one of 64 "predicate" bits.*

explore the roots of VLIW [3]. Though not as easy a read as Hennesy and Patterson, it offers 300+ pages devoted to the subject, and to smart compilers.

Actually, smarter compilers are a good thing no matter how the transistors are arranged. It's been said that RISC sometimes means Relegate the Impossible Stuff to the Compiler. And furthermore, the developments in VLIW hardware being bandied about today point to future processors that are by no means dumb.

Nevertheless, the essence of the concept remains: aren't programs run more often than they are compiled? Why not put the complexity in the compiler instead of the chip?

Yes, the software developers may gripe about compile times, as Ellis did [3] when hamstrung by a quaint DEC minicomputer with a (by modern standards) measly 1.5 MB of RAM that delivered compile times measured in hours. Perhaps they won't gripe at all (time for another long coffee break). Either way, are pokey compilers worse than making millions of chip users pay for extra silicon baggage?

As far as Intel is concerned, VLIW and IA-64 are quite different, and never the twain shall meet. I believe I heard the term "VLIW" mentioned only once in the Intel slide presentation—and even then it was in reference to work done by IA-64 development partner HP.

I hope as history is (re)written, the early pioneers won't be overlooked. Of course, few things in computing are brand new (see Figure 1) and much of the architecture underlying the VLIW concept goes back to, for example, the days of microcoding. Success in computer architecture has never been revolutionary but rather a process of incremental improvement, not to mention fortuitous marketing strategies.

VLIB?

Whatever Intel calls it, it looks like a VLIW to me (see Figure 2). Each 128-bit bundle is comprised of three 41-bit instructions and a 5-bit template. The template defines the parallel execution characteristics of the instructions within the bundle, most notably, if and at which point the CPU should stop and wait for instructions in progress to complete.

This is clever since it frees the compiler to find maximum parallelism in a program without being constrained by the limitations of a particular CPU implementation.

Suppose the compiler found six instructions that can execute in parallel. That turns into two 128-bit bundles, only the last with a stop. In principle, a version of IA-64 that's capable of executing three instructions at a time will take two clocks while a version that can execute six at a time will take only one, running exactly the same binary without the need to recompile it. That's cool, and a real boon for the shrink-wrapped software set.

The individual instructions look pretty much like a run-of-the mill three-operand RISC, with the addition of bits known as Qualifying Predicates. More on this topic in a moment. Do note the 7-bit register specifiers. IA-64 includes 128 65-bit general purpose, 128 82-bit floating point and dozens of other special purpose registers.

SPECULATIVE FEVER

We're not talking Internet IPOs here, though the concept of speculative execution also seems to fit in a world where the goal is peak performance at any price and something ventured always equals something gained.

The >1 IPC (instructions per clock) challenge is to find as many instructions that can be executed at the same time as possible. Unfortunately, doing so in hardware at runtime speeds (à la current superscalars) is difficult and leads to the silicon equivalent of bloatware (i.e., it takes a lot more transistors to get a little more IPC).

That makes chips costly, but even worse, ultimately impinges on the critical path and makes them slower as well. Boosting IPC by 10% at the expense of a 20% drop in the clock rate just won't do.

Even if hardware imposed no limits, the basic block bummer lurks in the shadows. Basic blocks are defined as a chunk of code delimited by single entry and exit points (in essence, the distance between branches).

The good news is it's relatively easy to identify instructions that can be executed in parallel within a basic block. The bad news is that it's hard to do so across basic block boundaries, and branches are all too frequent, which drastically limits the amount of instruction-level parallelism (ILP) available for mining.

IA-64 takes a two-pronged approach that boils down to making basic blocks larger by eliminating branches,

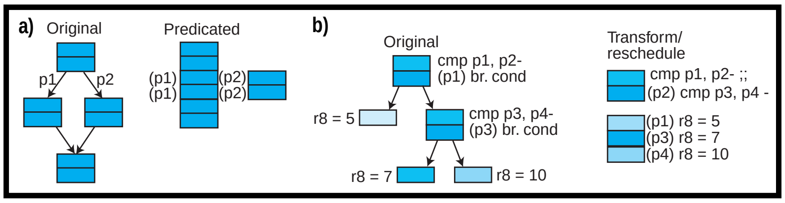
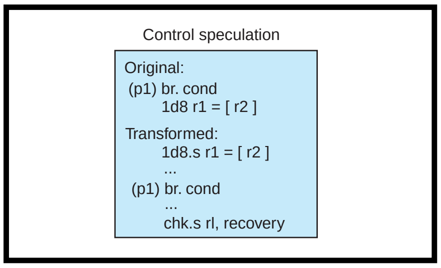


Figure 3—*Conditional execution (aka, predication) enables all sorts of compiler tricks. Traditionally, it's been considered a way to eliminate branches (a), but Intel takes it much further with, for example, multiple assignment. Notice in (b) the reduction in the worst-case path (i.e., from three bundles to two) even if the original version branches were predicted perfectly.*


Control speculation

```
Original:
 (p1) br. cond
       1d8 r1 = [ r2 ]
Transformed:
       1d8.s r1 = [ r2 ]
       ...
 (p1) br. cond
       ...
       chk.s rl, recovery
```

Figure 4—*Moving a load above a branch not only increases basic block length, but gives the memory hierarchy a head start. However, there is the risk the load will fault, if it never should have happened. The IA-64 speculative load (.s) defers, but keeps track of such a fault, for later inspection and, if necessary, recovery.*

and then moving code across basic block boundaries.

Eliminating branches brings us back to the Qualifying Predicate bits mentioned earlier. Traditionally, CPUs relied on condition codes or, more recently, combined compare and branch instructions. But for IA-64, most (but not all) instructions feature conditional execution based on the state of one of 64 predicate bits (the particular one specified by the 6-bit field in the instruction).

Predication not only simplifies control flow (i.e., eliminates branches thereby increasing basic block size), but also supports multiple selection, multiway branching (`case` statement) and multiple AND/OR compares (see Figure 3). Conditional execution is not a new concept (the ARM comes to mind), but Intel has taken it further.

Absent perfect branch prediction, moving code across basic block boundaries is an exercise in speculation. Without knowing with certainty which way a branch will go, you can't guarantee code moved from one basic block into another will be executed.

Of course, a smart compiler will move every nonspeculative instruction (one that will always be executed) it can, but beyond that, there's no choice but to execute instructions on spec and hope for the best.

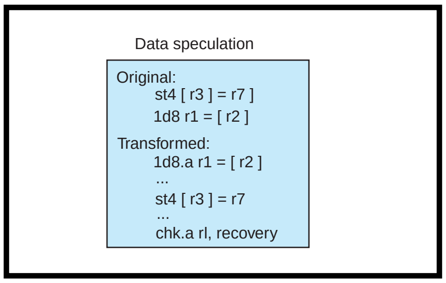
There are two kinds of speculation: control and data. Control speculation involves moving loads (and possibly instructions that use the loaded value) above a branch upon which their execution depends (see Figure 4).

Normally, the compiler wouldn't do that because if the branch went the other way, the speculated load could fault. Instead, IA-64 defers faults and simply sets an exception bit associated with the loaded register. This, the 65th bit in each register, is known as the NaT bit which stands for "Not a Thing", analogous to the '*x*87 NaN "Not a Number" bit, which performs a similar role tracking the validity of floating-point calculations. When it is time to use the loaded value, a check instruction verifies whether the load faulted (i.e., NaT bit set), in which case recovery code can be executed.

Similarly, data speculation moves loads (and possibly uses) above potentially overlapping stores (see Figure 5). Potentially overlapping stores refers to the infamous pointer alias problem. It's difficult to guarantee that two pointers don't reference the same address in memory, incurring a dependency. There are complicated analyses that can be applied to the problem of pointer disambiguation, but these tend to carry flaws, such as an inability to guarantee finite compile time!

Instead, IA-64 effectively does pointer disambiguation in hardware using a combination of advanced loads and checks. An advanced load is moved ahead of a possibly overlapping store and the load address is automatically saved into an advanced load address table (ALAT). From this point on, the hardware watches all memory write traffic for stores to the same address. Should one occur, the advanced load address is removed from the ALAT.

When the time comes to actually use the loaded value, a check instruction confirms whether or not it's safe


Data speculation

```
Original:
      st4 [ r3 ] = r7 ]
      1d8 r1 = [ r2 ]
Transformed:
      1d8.a r1 = [ r2 ]
      ...
      st4 [ r3 ] = r7
      ...
      chk.a rl, recovery
```

Figure 5—*Pointers (variables used as memory addresses) hinder optimization. How can the compiler know that r2 won't equal r3 at runtime? Hard to tell, but if so, they are aliases and the instructions cannot even be executed at the same time, much less in reverse order. By watching for subsequent writes to the same address (i.e., r3 does turn out to equal r2) at runtime, IA-64 solves the problem.*

to proceed (i.e., nothing else wrote to the same address in the meantime) by confirming the address is still in the ALAT. If a collision occurred and an address disappeared from the ALAT, a recovery routine would be executed, but otherwise, and most often, everything will be hunky-dory.

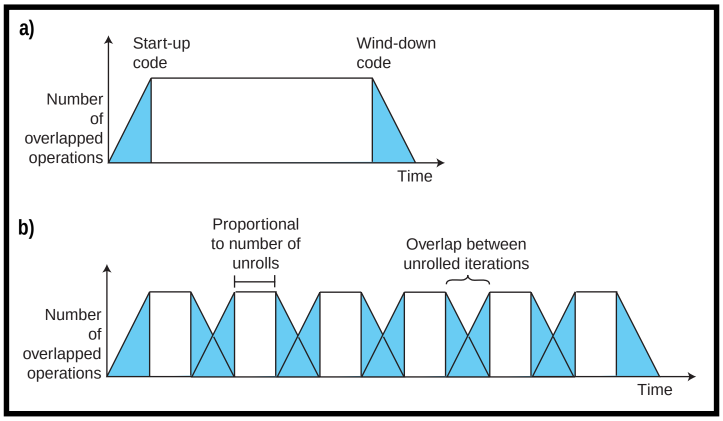

a)
Start-up code
Wind-down code
Number of overlapped operations
Time
b)
Proportional to number of unrolls
Overlap between unrolled iterations
Number of overlapped operations
Time

Figure 6—*As shown in Hennessey and Patterson [1] (a) software pipelining, (executing different iterations of a loop at the same time) goes beyond simple loop unrolling. IA-64 (b) brings all features, including rotating registers, predication and dedicated looping hardware, to bear on the subject, moving the technique out of the lab and into the mainstream.*

ROCK AND ROLL REGISTERS

All those registers are put to good use thanks to automatic stacking and rotation capability. The stacking feature shares the motivation of other register window schemes (à la SPARC), which is to minimize the push and pull overhead associated with subroutine calls. The rotation capability is more unique and works in concert with special branch instructions to facilitate software pipelining of loops.

Most of you have probably heard of the idea of loop unrolling, in which the body of the loop is replicated *n* times, reducing the amount of time spent on loop overhead and branching by 1/*n* and further increasing the basic block size to expose more parallelism. Software pipelining takes the concept even further by reorganizing the code such that different iterations of the loop execute in parallel. It's completely analogous to a hardware pipeline, including the need to fill and drain the pipeline (see Figure 6).

Although software pipelining, like its hardware counterpart, can significantly improve throughput, there is a cost—the unrolling (i.e., copies of the loop body) and the need for extra prologue (fill) and epilogue (drain) routines leads to bloaty code.

The Intel rotation scheme cleverly works in concert with special looping-oriented branches and counters to achieve the benefits of software pipelining with less code and complexity. In essence, it provides a hardware assist through a kind of register renaming in which reference to a single virtual register number maps to a physical register number that increments automatically with each iteration of the loop.

Notably, by eliminating the overhead and duplication of traditional software pipelining, the IA-64 scheme is suitable not only for rocket science DSP-loops and such, but also for the typical small loops that characterize meat and potato code.

All the registers are grand, but ultimately stuff has to get to and from memory. Here, IA-64 incorporates prefetch instructions and also hints to give the programmer/compiler a measure of explicit control over the memory hierarchy. Ideally, critical data ends up residing near the CPU, while rarely used, one-off data remains further adrift. In essence, the goal is to balance the best of both cache (automatic but statistical speed-up of data access in general) and RAM (deterministic access to hot items without cache thrash).

NOW COMES THE TRICKY PART

Based on what I've seen so far, IA-64 is quite impressive. I think Intel and HP have done an excellent job of extending the state-of-the-art. Most notably, it goes beyond the smart compiler, dumb chip genesis of VLIW by adding features to the hardware that don't displace the compiler's authority, but rather make it easier for the compiler to do a better job.

In a vacuum, "IA-64 Wins WunderChip Wars" would be the headline of this story. But there is the small matter of all those old '*x*86 binaries rattling around in the closet, a subject the tutorials said absolutely nothing about. I certainly presume the IA-64 chips will be able to run '*x*86 code, no doubt relying on runtime translation techniques similar to those Intel already uses to run old software on their current RISC-in-drag chips. Still, I wonder to what degree the native potential of the architecture will be lost in the translation.

Maybe it doesn't matter. At this point at least, Intel is positioning IA-64 and future IA-64 chips as targeting servers rather than PCs. Of course, traditionally, last year's server turns into next year's PC, but I'm not sure that IA-64 won't represent a bit more of a discontinuity.

Intel is caught between the rock of running decades-old binaries and the hard place of competition posed by newcomers with less baggage (the next generation Playstation comes to mind). Since completely casting off the yoke of compatibility isn't an option, I suspect that when push comes to shove, Intel will realize they must favor the future, rather than the past.

So what will an IA–64-based box look like? Will it run DOS and Win95/98? How about your archive of 1.44-MB floppies?

The "Developer's Interface Guide for IA-64 Servers" I downloaded from http://dig64.org is illuminating. The short answer is that most of these things are possible (though not all; it's curtains for ISA) but Intel really would rather everyone march to the beat of a legacy-free drummer with IA-64. I imagine they'll continue to offer new and improved IA-32 chips for the masses of plain PC users.

But when it comes to IA-64, I have to agree with Intel. It is time for a change, and what time could be better than the start of a new millennium? Hey, a 64-bit register can hold all 8 bytes of a calendar date, so at least there won't be a year 3000 problem.

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.

REFERENCES

[1] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, Chicago, IL, 1996.

[2] J. Fisher, J. Ellis, J. Ruttenberg, and A. Nicolau, "Parallel Processing: A smart compiler and a dumb processor," SIGPLAN conference on compiler construction proceedings, June, 1984.

[3] J. Ellis, *Bulldog: A Compiler for VLIW Architectures*, MIT Press, Cambridge, MA, 1986.

Hot Chips Conference, www.hot.org.

http://developer.intel.com/design/IA64

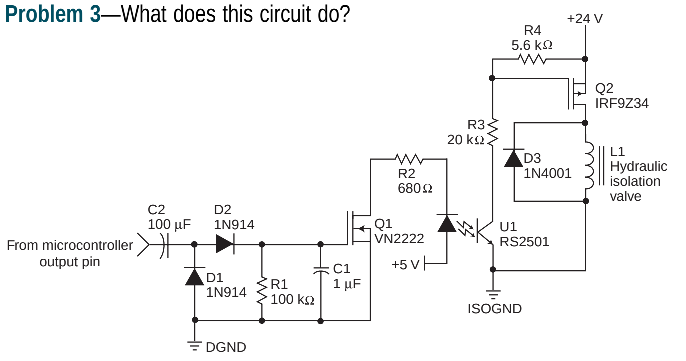
CIRCUIT CELLAR Test Your EQ

Problem 1—Does the hole in a metal flat washer get larger or smaller as the washer is heated?

Problem 2—The function shown below is used to generate a series of values for programming a DAC that will generate a sine wave output. The sine wave produced seems to have a slight glitch on every other zero crossing point. Can you identify the problem in the lookup table value-generating function?

```
#define TABSIZ 33
static unsigned short dac_sine_val[TABSIZ];

void dac_val_gen(void) {
 int i;
 unsigned short val;
 float f;

 for (i=0; i<TABSIZ; i++)
 {
 f=sin(2 * i * 3.1415 / (TABSIZ-1));
 printf("%f,", f);

 val = (short)((f+1.0) * 32767);

 dac_sine_val[i]= val;
 }
}
```

Problem 3—What does this circuit do?



Problem 4—The SMT-33 stepper motor has 15° full steps. An eight-tooth gear is attached to the motor shaft. The eight-tooth gear in return drives a 40-tooth gear attached to a wheel of 50-mm diameter. The wheel rests on the ground and drives the robot.

What is the minimum displacement the robot is capable of, assuming the robot can perform half-stepping?

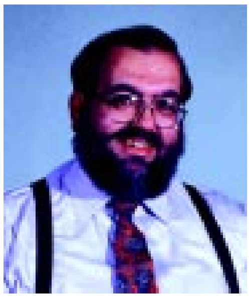
What's *your* EQ?—The answers and 4 additional questions and answers are posted at www.circuitcellar.com.

You may contact the quizmasters at eq@circuitcellar.com.

8 more EQ questions each month in Circuit Cellar Online see pg. 2

PRIORITY INTERRUPT

Reading Between the Lines



The index is the first thing I look at when I pick up one of the many magazines shipped to me gratis every week. If I find some interesting items in the index, I place the magazine in one pile, the rest go in the trash. It's just a fact of life for a working manager faced with diminishing time to indulge in optional reading.

The last page is where I go when I receive a brand new issue of Circuit Cellar. Why? Your thoughts are insightful, full of common sense, and sometimes outright inspirational. After reading your commentary, I know I am in good hands and can take the magazine home with me like a good bottle of wine. I place it next to the bed and savor it for a whole month.

Your comments in September's editorial prompted me to write this letter. Not because what you said was controversial, but to remind you about the statement you made, "By engineers, for engineers." This, in my opinion, excludes some of us who are not engineers by profession but who use your magazine to raise their knowledge bar.

So, you should be glad to know that your wisdom and magazine go beyond your profession.

– Anthony Cervone, VP of Manufacturing, Excell Mfg.

Well, Anthony, it's actually worse than that. I didn't just say it in my editorial. We also have that motto plastered all over our media kit and web site. I assure you, it's not meant to exclude anyone. It's just that I don't know a better phrase to describe what *Circuit Cellar* is all about, without going into a longer description. Let me explain.

I think the real issue here is what I mean by "engineer." What do you think of when I say engineer? Do you think back to college and the propeller-head geeks at the end of the hall, or the ones who spent so much time on the computer that you'd swear they had DSL implants? The picture that people get in their mind when you say "engineer" depends primarily on their experience and expertise. Somewhere in the overlap of impressions, however, everyone will agree that the word "engineer" denotes technical expertise. That's what I'm counting on.

A lot of us on the *Circuit Cellar* staff are engineers (I'd say "real" engineers, but that sounds corny). We don't let it go to our heads. All that means is that we stand a chance of understanding some of the stuff we publish. Perhaps one of the reasons you like *Circuit Cellar* is that we recognize that we don't have a corner on the market and this expertise is shared by all of you as well. I know that every issue is read by people who are a great deal smarter than the lot of us. It doesn't intimidate me. It's a healthy fear. It challenges us to check the facts, proof things over and over, and constantly review the accuracy of our content.

Regardless of whether you think my terminology is correct, you know from the author biographies that these people aren't all engineers. They are manufacturing VPs, educators, systems analysts, programmers, experimenters, and oh by the way, engineers. *Circuit Cellar*'s strength is that it isn't put together by authors all with the same expertise. And, unlike a trade journal, our readers are our authors (and vice versa). Every columnist was a reader first. Every project we publish comes to you because a reader wants to share that knowledge. Are they all engineers? Nope.

I really coined the "by engineers, for engineers" motto for everyone else. It's for the part of the public that isn't at our level and tends to group anything technical into easily defined categories. If something is "for engineers" they at least think about it before subscribing. Years ago, before we had this statement, I had people complaining that I wasn't teaching them electronics! Our content was too advanced and we hadn't warned them. What you have to realize, is that in the minds of most of the world, the people who understand *Circuit Cellar* are *all* engineers.

So, let's face it, the phrase we use is a factual overstatement. The good news is that you are the first person to take me to task for it in all the time I've said it. I can only hope that readers like you continue to value *Circuit Cellar* as a vehicle for raising the knowledge bar. No, *Circuit Cellar* isn't just by and for engineers. I suppose it's more accurate if we say, "by technically astute individuals with embedded-systems expertise for an equally well-informed readership of multi-disciplinary aptitude." Of course, saying "by engineers, for engineers" (and hoping you all understand) is a lot easier.

Steve

steve.ciarcia@circuitcellar.com