Build a USB GPIO Pod p.16 • Robot Navigation Software p. 30 • Decode Signal Transmissions p. 40

www.circuitcellar.com

CIRCUIT CELLAR

THE MAGAZINE FOR COMPUTER APPLICATIONS

#225 April 2009

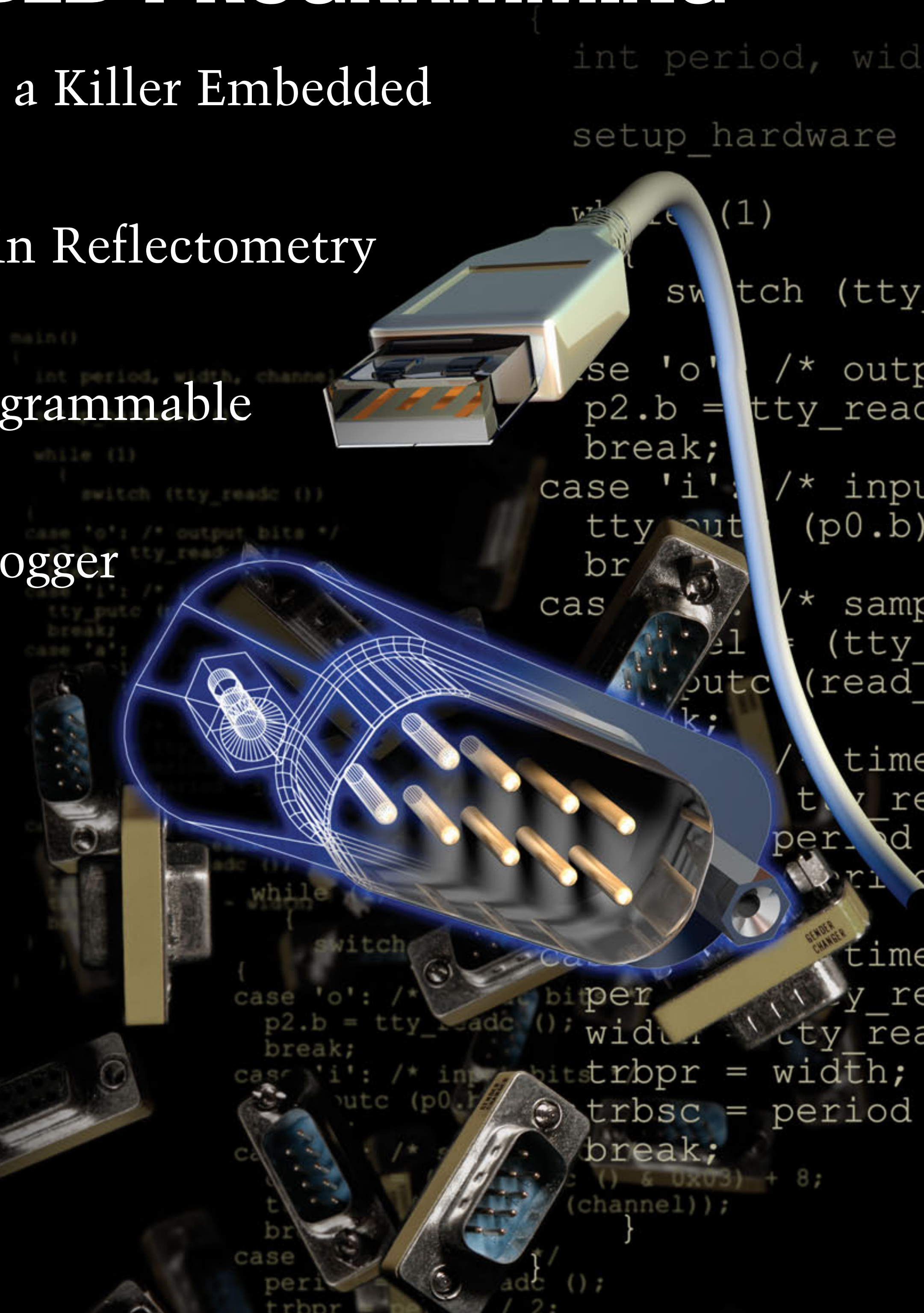
EMBEDDED PROGRAMMING

A Recipe for a Killer Embedded Application

Time Domain Reflectometry Explained

More on Programmable Robotics

Solar Data Logger Design



$5.95 U.S. ($6.95 Canada)

5 Competitive Advantages

Overseas Manufacturing

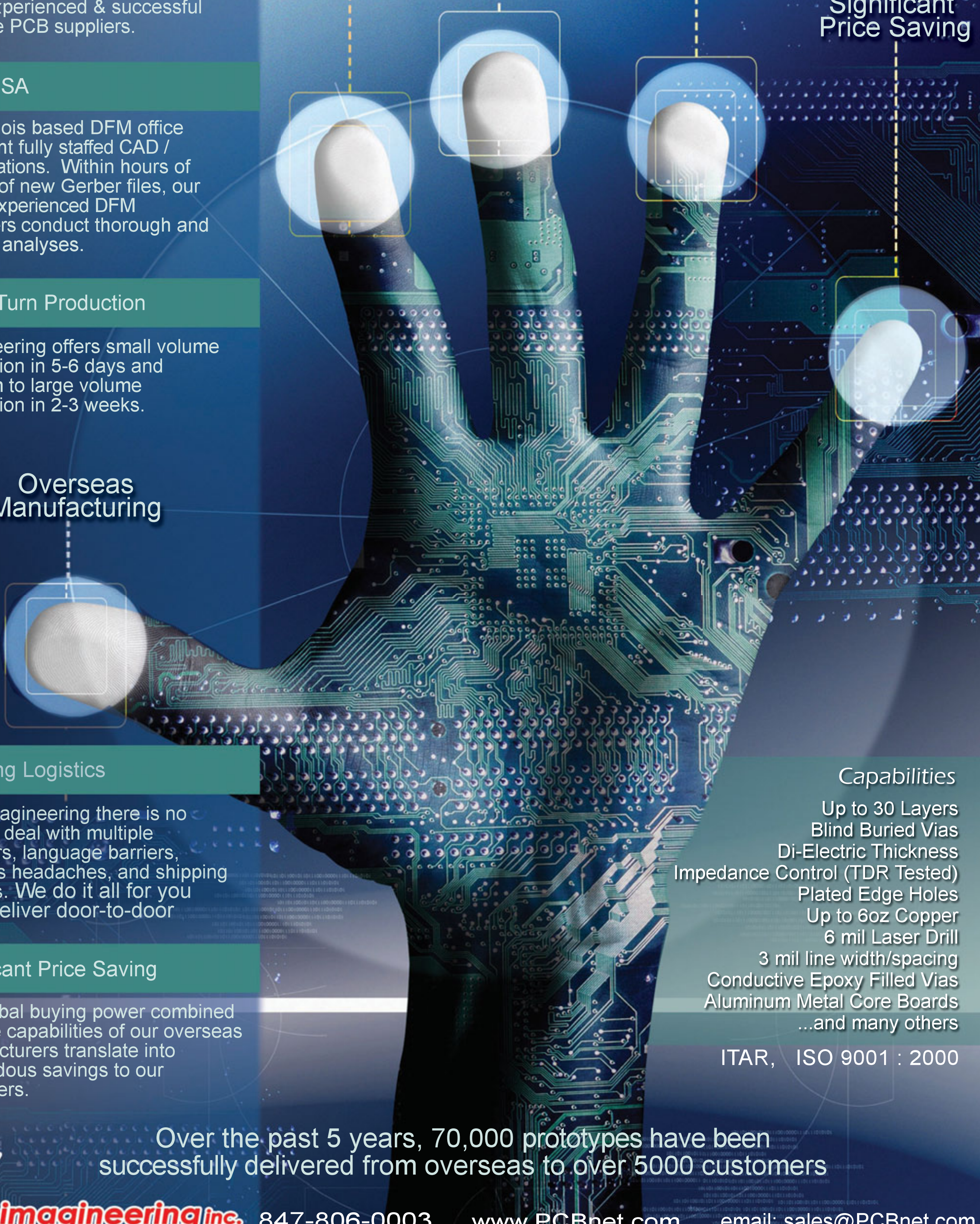
Imagineering, Inc. enjoys the reputation of being one of the most experienced & successful offshore PCB suppliers.

CAM USA

Our Illinois based DFM office has eight fully staffed CAD / CAM stations. Within hours of receipt of new Gerber files, our highly experienced DFM engineers conduct thorough and precise analyses.

Quick-Turn Production

Imagineering offers small volume production in 5-6 days and medium to large volume production in 2-3 weeks.



Shipping Logistics

With Imagineering there is no need to deal with multiple suppliers, language barriers, customs headaches, and shipping logistics. We do it all for you ..and deliver door-to-door

Significant Price Saving

Our global buying power combined with the capabilities of our overseas manufacturers translate into tremendous savings to our customers.

Capabilities

Up to 30 Layers
Blind Buried Vias
Di-Electric Thickness
Impedance Control (TDR Tested)
Plated Edge Holes
Up to 6oz Copper
6 mil Laser Drill
3 mil line width/spacing
Conductive Epoxy Filled Vias
Aluminum Metal Core Boards
...and many others

ITAR, ISO 9001 : 2000

Over the past 5 years, 70,000 prototypes have been successfully delivered from overseas to over 5000 customers

imagineering Inc. 847-806-0003 www.PCBnet.com email: sales@PCBnet.com

23 YEARS IN BUSINESS...AND STILL GOING STRONG

TASK MANAGER

Troubleshooters, Unite!

Sure, workarounds are great. It's clear that a nifty workaround can be helpful when you're in a tight bind, particularly when a client or project manager is breathing down your neck. But there's no "working around" the fact that a workaround is nothing more than a temporary fix—a fix that, if left unattended, can trigger a project-wide disaster. I know, you probably have a story about how you saved the day with a brilliant workaround on a day when failure was just over the horizon. I applaud you for that success, because we here at *Circuit Cellar* value ingenuity in all its forms. But at the end of the day, you must agree that a *solution* always trumps a workaround at the workbench.

The process of tailoring a solution to an embedded design problem requires a talented engineer to troubleshoot complex circuitry and code glitches of all sorts. And if a glitch lies around the corner, rather than presently before him, a good engineer must be able to troubleshoot that potential complication before it rears itself up.

As with most embedded-design-related skills, the science of troubleshooting both existing and potential problems takes time to master. It is typically developed over the course of dozens of projects and nurtured by adept mentors such as *Circuit Cellar* authors. Like our founder Steve Ciarcia, many *Circuit Cellar* authors have excelled at troubleshooting existing and potential design problems—as well as real-world problems fixable with embedded design applications—over the course of many years. In this issue, a group of stand-out authors unites to present useful articles that highlight their wide range of skills.

Lack a parallel port? No worries. In a series titled "Construct a USB GPIO Pod," DJ Delorie shows you how to address this problem with a general-purpose input/output module that plugs into a USB port. This month he presents the module (p. 16). Encountering trouble while building your first solar data logger? Columnist Ed Nisley describes how to assess your mistakes, regroup, and move forward (p. 24).

Wondering how to program that motionless robot sitting beside your workbench? Don't let software problems keep you from realizing your design goals. In the second part of his series, "Robot Navigation and Control," Guido Ottaviani explains how to write and debug software to get the job done (p. 30). Jeff Bachiochi's article on page 58 includes information about application development for a basic robotics system.

Having issues with the signal-processing aspect of a design or, more specifically, decoding a particular signal? You're in luck. Two authors focus on demystifying the topics of signal processing, signal reflection, and signal analysis. Danilo Consonni explains how he decodes hourly signal transmissions (p. 40). He built a digital decoder to analyze the Italian SRC-RAI time signal. If you're confused by the topics of signal reflection or impedance mismatching, turn to Robert Lacoste's article, "Time Domain Reflectometry" (p. 50). He describes how to detect and measure an impedance mismatch in a transmission line and more.

Tom Cantrell wraps up the issue by explaining why acquiring a "healthy mix" of MCUs, sensors, and wireless technologies to keep on hand can lead to the creation of exciting new "killer apps" (p. 66). With a nice variety of cutting-edge parts on tap, you can push the innovation envelope and quickly solve any number of menacing design problems.

cj@circuitcellar.com

C. Abate

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

MANAGING EDITOR
C. J. Abate

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
Jeff Bachiochi
Ingo Cyliax
Robert Lacoste
George Martin
Ed Nisley

NEW PRODUCTS EDITOR
John Gorsky

PROJECT EDITORS
Gary Bodley
Ken Davidson
David Tweed

ASSOCIATE EDITOR
Jesse Smolin

CHIEF FINANCIAL OFFICER
Jeannette Ciarcia

MEDIA CONSULTANT
Dan Rodrigues

CUSTOMER SERVICE
Debbie Lavoie

CONTROLLER
Jeff Yanco

ART DIRECTOR
KC Prescott

GRAPHIC DESIGNERS
Grace Chen
Carey Penney

STAFF ENGINEER
John Gorsky

ADVERTISING
860.875.2199 • Fax: 860.871.0411 • www.circuitcellar.com/advertise

PUBLISHER
Sean Donnelly
Direct: 860.872.3064, Cell: 860.930.4326, E-mail: sean@circuitcellar.com

ADVERTISING REPRESENTATIVE
Shannon Barraclough
Direct: 860.872.3064, E-mail: shannon@circuitcellar.com

ADVERTISING COORDINATOR
Valerie Luster
E-mail: val.luster@circuitcellar.com

Cover photography by Chris Rakoczy—Rakoczy Photography
www.rakoczyphoto.com

PRINTED IN THE UNITED STATES

CONTACTS

SUBSCRIPTIONS
Information: www.circuitcellar.com/subscribe, E-mail: subscribe@circuitcellar.com
Subscribe: 800.269.6301, www.circuitcellar.com/subscribe, Circuit Cellar Subscriptions, P.O. Box 5650, Hanover, NH 03755-5650
Address Changes/Problems: E-mail: subscribe@circuitcellar.com

GENERAL INFORMATION
860.875.2199, Fax: 860.871.0411, E-mail: info@circuitcellar.com
Editorial Office: Editor, Circuit Cellar, 4 Park St., Vernon, CT 06066, E-mail: editor@circuitcellar.com
New Products: New Products, Circuit Cellar, 4 Park St., Vernon, CT 06066, E-mail: newproducts@circuitcellar.com

AUTHORIZED REPRINTS INFORMATION
860.875.2199, E-mail: reprints@circuitcellar.com

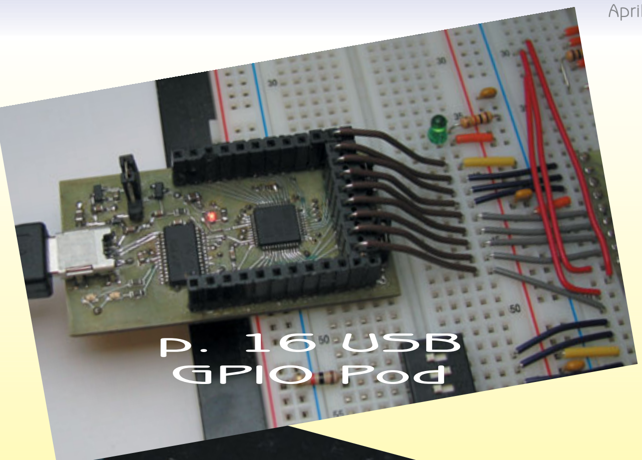
AUTHORS
Authors' e-mail addresses (when available) are included at the end of each article.

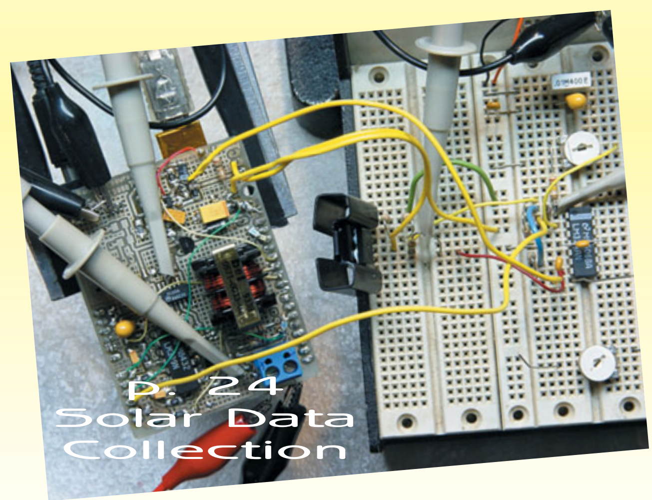
CIRCUIT CELLAR®, THE MAGAZINE FOR COMPUTER APPLICATIONS (ISSN 1528-0608) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Vernon, CT 06066. Periodical rates paid at Vernon, CT and additional offices. **One-year (12 issues) subscription rate USA and possessions $23.95, Canada/Mexico $34.95, all other countries $49.95. Two-year (24 issues) subscription rate USA and possessions $43.95, Canada/Mexico $59.95, 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 5650, Hanover, NH 03755-5650 or call 800.269.6301.**

Postmaster: Send address changes to Circuit Cellar, Circulation Dept., P.O. Box 5650, Hanover, NH 03755-5650.

INSIDE ISSUE 225

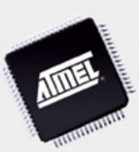
April 2009 • Embedded Programming





Hammer Down Your Power Consumption with picoPower™!

THE Performance Choice of Lowest-Power Microcontrollers



Performance and power consumption have always been key elements in the development of AVR® microcontrollers. Today's increasing use of battery and signal line powered applications makes power consumption criteria more important than ever. To meet the tough requirements of modern microcontrollers, Atmel® has combined more than ten years of low power research and development into picoPower technology.

picoPower enables tinyAVR®, megaAVR® and XMEGA™ microcontrollers to achieve the industry's lowest power consumption. Why be satisfied with microamps when you can have nanoamps? With Atmel MCUs today's embedded designers get systems using a mere 650 nA running a real-time clock (RTC) and only 100 nA in sleep mode. Combined with several other innovative techniques, picoPower microcontrollers help you reduce your applications power consumption without compromising system performance!

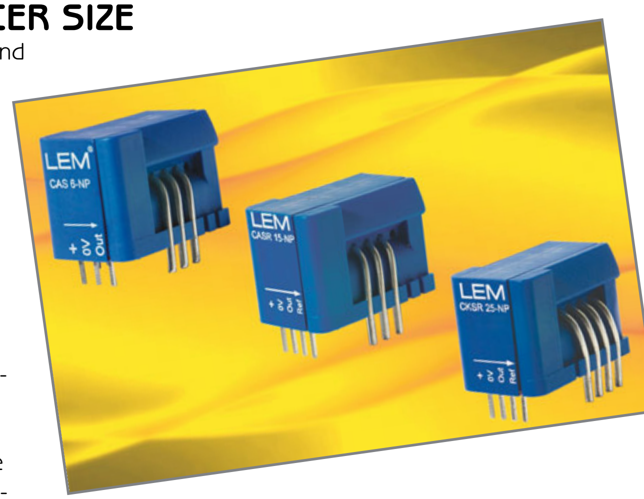
Visit our website to learn how picoPower can help you hammer down the power consumption of your next designs. PLUS, get a chance to apply for **a free AVR design kit!**

http://www.atmel.com/picopower/

Everywhere You Are®

FLUXGATE TECHNOLOGY REDUCES TRANSDUCER SIZE

The **CAS**, **CASR**, and **CKSR** family of transducers is intended for AC and DC isolated current measurement from 6 to 50 ARMS nominal, up to three times the nominal values for the peak measurement, and a bandwidth up to 300 kHz (±3 dB). These new transducers were designed to respond to technology advances that require better performance in areas such as common-mode influence, thermal drifts, response time, levels of insulation, and size. Using closed-loop fluxgate technology, the transducers offer high accuracy and affordability without compromising any of the advantages of its LTS family, such as size, dynamic performance and wide measuring range.



Although the new transducers are 30% smaller than the existing LTS family, their insulation performance can withstand use in industrial applications without a special layout of the PCB. The CKSR model has one more primary pin than the three pins of the CAS and CASR models and a different primary footprint that enables higher creepage and clearance distances of 8.2 mm. Moreover, this additional primary pin allows a configuration of the CKSR 6-NP model for a nominal current range of 1.5 ARMS.

All CAS, CASR and CKSR transducer models have been designed for direct mounting onto a PCB for primary and secondary connections and operate from a single 5-V supply. The CASR and CKSR models provide their internal reference voltage to a VREF pin. An external voltage reference between 0 and 4 V can also be applied to this pin.

Pricing for the transducers is $11.05 each in 500-piece quantities.

LEM S.A.
www.lem.com

TEMPERATURE/RH LOGGER WITH LCD

The new **USB-502-LCD** is a cost-effective solution for long- or short-term logging applications. It features an LCD window for viewing temperature and humidity information. The stand-alone data logger measures and stores up to 16,379 relative humidity and 16,379 temperature readings over the 0 to 100% RH and –35 to 80° C (–31 to 176° F) measurement ranges, and it calculates dew point.

The high-contrast LCD shows a variety of temperature and humidity measurements as well as logger status information. At the touch of a button, users can turn on the LCD, and cycle through the most recent, the maximum, and the minimum logged temperatures and relative humidity readings. Flashing LEDs indicate logging status. A user-replaceable long-life lithium battery is included, which typically allows up to one full year of logging.

The USB-502-LCD logger is supported by the USB-500 Series Data Logging Application software included with the package. This user-friendly intuitive software is used initially to configure the logger and program alarm thresholds, logging rates, start times, measurement units, and more. When the logger is retrieved from the field, the software provides simple one-click access to download, display, and export the data. The data can also be downloaded to a text or a .CSV file. The software and data loggers are compatible with Windows 2000/XP/Vista.

The USB-502-LCD sells for **$97**.

Measurement Computing Corp.
www.measurementcomputing.com

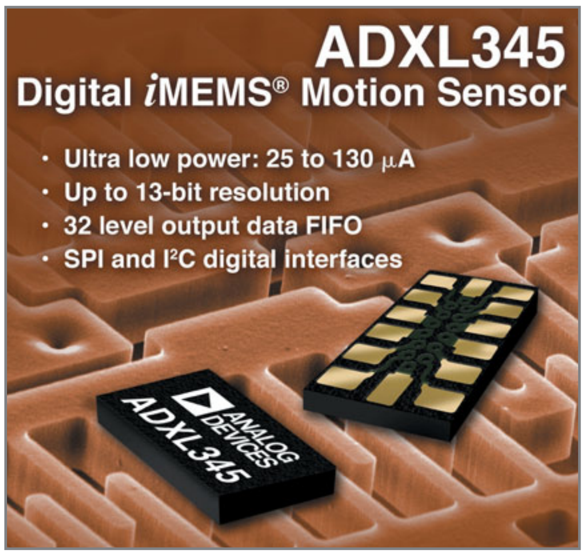
NEW PRODUCT NEWS

Edited by John Gorsky

POWER-EFFICIENT MEMS MOTION SENSOR

The **ADXL345** three-axis digital MEMS accelerometer is the lowest-power device in its class, achieving an 80% power savings compared to competing three-axis inertial sensors. The ADXL345 also incorporates an on-chip ADC that dramatically reduces power consumption requirements in wireless handsets, personal navigation devices, and other mobile applications.

The ADXL345 ultra-low-power digital accelerometer has an output data range that scales from 0.1 Hz to 3.2 kHz, unlike competing devices that have fixed 100-Hz, 400-Hz, or 1-kHz data rates. This enables portable system designers to better manage energy consumption by precisely allocating power for a given system function and reserving unused power for other uses. The ADXL345 also measures dynamic acceleration resulting from motion or shock and with a 10,000-g shock rating is well suited for applications such as hard-disk drive protection in personal computers. Featuring resolution of 4-mg/LSB across all g ranges, single tap and double tap detection, activity and inactivity detection, free fall detection, and user-programmable threshold levels, the new accelerometer also includes I²C and three- and four-wire SPI digital interfaces and a voltage range of 1.8 to 3.6 V.



The ADXL345 comes in a small, thin 14-lead LFCSP. It costs **$3.04** in 1,000-unit quantities.

Analog Devices, Inc.
www.analog.com

ALL-ENVIRONMENT ULTRASONIC SENSORS

The **MaxSonar-WR1** ultrasonic sensors are designed for either indoor or outdoor use while maintaining very low cost. The sensor will work in many industrial applications such as presence sensing, distance measurement, outdoor mounting, people and proximity detection, snow measurement, and tank level detection. The rugged packaging and narrow beam width of the WR1 means that these sensors can be used for mobile robotics applications such as room mapping where a precision beam is required.

The LV-MaxSonar-WR1 features a 3- to 5.5-V low power requirement and provides very short-to-long-range detection and ranging. It is designed to meet IP67 water intrusion criterion, and it operates over a wide temperature range from 40° to 85°C. The sensor is packaged in a robust PVC housing and offers a mounting system that matches standard electrical 0.750" pipe fittings (14 NPTS). The LV-MaxSonar-WR1 detects objects from 0 to 254" (6.45 m) and provides sonar range information from 12 out to 254" with a 1" resolution. The interface output formats included are pulse width, analog voltage, and serial digital.

The pricing structure of the MaxSonar-WR1 sensor makes it the lowest cost sensor in its class. The LV-MaxSonar-WR1 retails for **$99.95**.

MaxBotix, Inc.
www.maxbotix.com

NPN

TRIAC-DIMMABLE LED DRIVER

The **LM3445** is a constant-current controller that enables off-line, uniform, flicker-free dimming of high-brightness LEDs with a conventional TRIAC forward or reverse phase-control wall dimmer. The dimmable LED driver enables a full 100:1 range of dimming capability. It can maintain greater than 1 A of constant current to large strings of LEDs in a variety of residential, architectural, commercial, and industrial applications.

The LM3445 maximizes the light output for systems while maintaining ENERGY STAR power factor requirements in a typical application, positioning it among National's PowerWise family of energy-efficient products.

Today's TRIAC wall dimmers are designed to interface with a resistive load such as incandescent or halogen light bulbs. Since an LED bulb does not appear as a resistive load to the TRIAC wall dimmer, dimming an LED bulb using a conventional TRIAC wall dimmer does not yield optimal dimming performance. The LM3445 overcomes this challenge by translating the TRIAC-chopped waveform to a DIM signal and decoding it for a full-range of uniform, flicker-free dimming. The driver's patent-pending control architecture maintains constant ripple through the LEDs, which extends the life of the LEDs.

The LM3445 LED driver enables the direct LED bulb replacement of existing incandescent or halogen bulb systems connected to standard TRIAC wall dimmers. In addition, the driver allows master-slave operation, enabling control of multiple strings of LED bulbs. A complete LED system featuring the LM3445 can be created in minutes with the WEBENCH LED Designer.

The LM3445 costs **$1.75** in 1,000-unit quantities.

National Semiconductor
www.national.com

ETHERNET CARD FOR THE HYDRA SYSTEM

The **Hydra EtherX** is an Ethernet card designed to interface to the Hydra system via the expansion port. The card is built around Wiznet's W5100 hardwired TCP/IP Ethernet chip and interfaces to the Hydra using SPI. The Hydra allows designers to develop games, graphics, and media applications with the Propeller-powered game console.

The Hydra EtherX card comes with a complete eBook describing how the sample EtherX card driver works from the ground up. The manual covers SPI, Ethernet, and Internet protocols (IP, TCP, and UDP). With this knowledge, designers will be able to write their own drivers for the W5100 or expand on the previously written sample driver. The driver includes an easy-to-use API with source code, examples, tutorials, and detailed explanations. You can use the EtherX card to turn the Hydra into a web server, access a file server, or play games over the Internet.

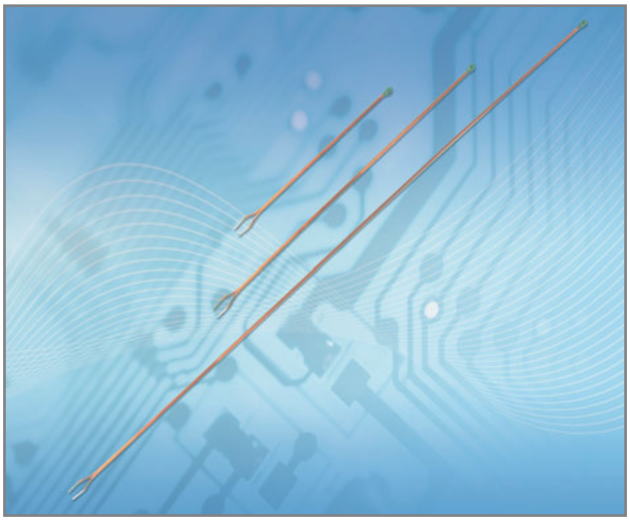
The Hydra EtherX Card costs **$59.99**. The Hydra Game Development kit costs **$199.99**.

Parallax, Inc.
www.parallax.com

LONG WIRE LEAD THERMISTOR

The **Thermo-String** temperature sensor is an enhanced thermistor fitted with long wire leads (up to 150 mm) so it can be placed close to hot spot areas. Utilizing the NCP series surface mount NTC thermistor line to provide accurate temperature measurement over wide temperature ranges, the Thermo-String NXF series proffers a convenient leaded package to reduce costs, because there is no need to design a new thermistor from scratch.

Aside from the benefits of long wire leads and cost savings, the Thermo-String offers proven performance in a small package. With a maximum head size of only 2 mm, the sensor provides operating temperatures in the range of –40° to 125°C. Suitable for lead-free soldering, the sturdy thermistor is resistant to mechanical vibrations and shock, and it meets ELV/UL requirements. Due to its numerous attributes, the Thermo-String is ideal for use in a variety of applications (e.g., battery cells, servers, PCs, power tools, measuring instruments, medical equipment, and heating, and ventilation and air conditioning (HVAC)).



The Thermo-String is available for **$0.16** to **$0.24** in volume.

Murata Electronics North America
www.murata.com

COMPACT FANLESS COMPUTER IN AN INDUSTRIAL-GRADE ENCLOSURE

The **PL-60590** is a compact fanless appliance in a rugged compact aluminum chassis (8.7″ W × 8.5″ D × 3.5″ H). The small device is designed for kiosk, digital signage, vehicle, gaming/entertainment, and industrial automation applications. The enclosure is sealed from environmental intrusion and is designed for antivibration for high reliability.

The scalable unit is powered by a choice of Intel Core 2 Duo/ Core Duo/ Celeron M processors supported by Intel 945GME and ICH7R chipsets. The Intel 965GME chipset supports dual independent displays with CRT, DVI, and 24-bit LVDS technology. An Intel 82573L Ethernet controller provides support for two GbE LAN ports.

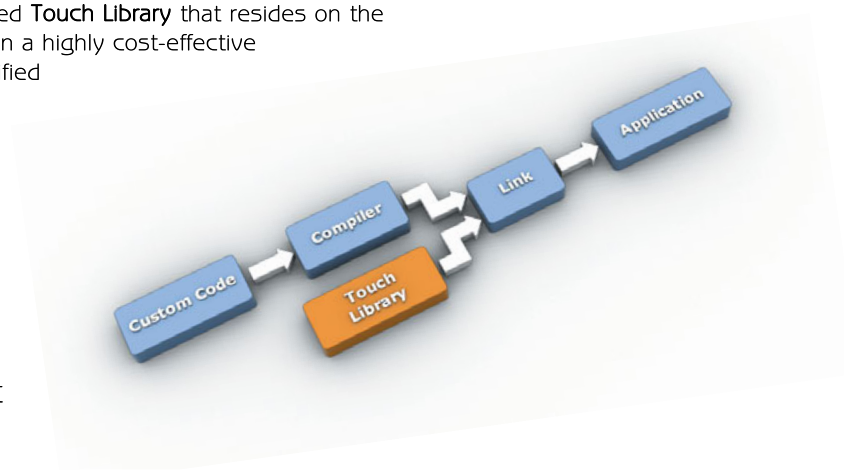
The robust unit features four COM ports, six USB 2.0 ports, dual GbE, VGA, a PS/2 port, and one MIO module that can be used for the expansion of I/O functions. The unit supports a 2.5″ hard disk for high-capacity storage and one DDRII 533/667 DIMM socket for up to 2-GB system memory. The RoHS-compliant product is currently available.

Unit pricing for the PL-60590 begins at **$448** in OEM quantities. Linux, Windows Embedded XP, and FreeBSD are supported.

WIN Enterprises, Inc.
www.win-ent.com

ADVANCED TOUCH LIBRARY FOR AVR MICROCONTROLLERS

Atmel has announced the availability of its advanced **Touch Library** that resides on the microcontroller instead of a separate chip, resulting in a highly cost-effective solution. The library consists of precompiled and verified binary files, which can be configured individually as discrete keys or combined at will (as groups) to form wheels and sliders.



A complete development environment makes it easy to develop a touch system based on the AVR microcontroller. The AVR core combined with QTouch technology provides the industry's most robust touch solution, particularly in demanding applications. This has multiple advantages for the designer, such as a high signal-to-noise ratio that improves the systems design margin, increases EMC performance, and provides high ESD tolerance. In addition, designers can address applications where reliability is required, such as stove tops, wall ovens, and automotive applications, as well as where high moisture levels are present.

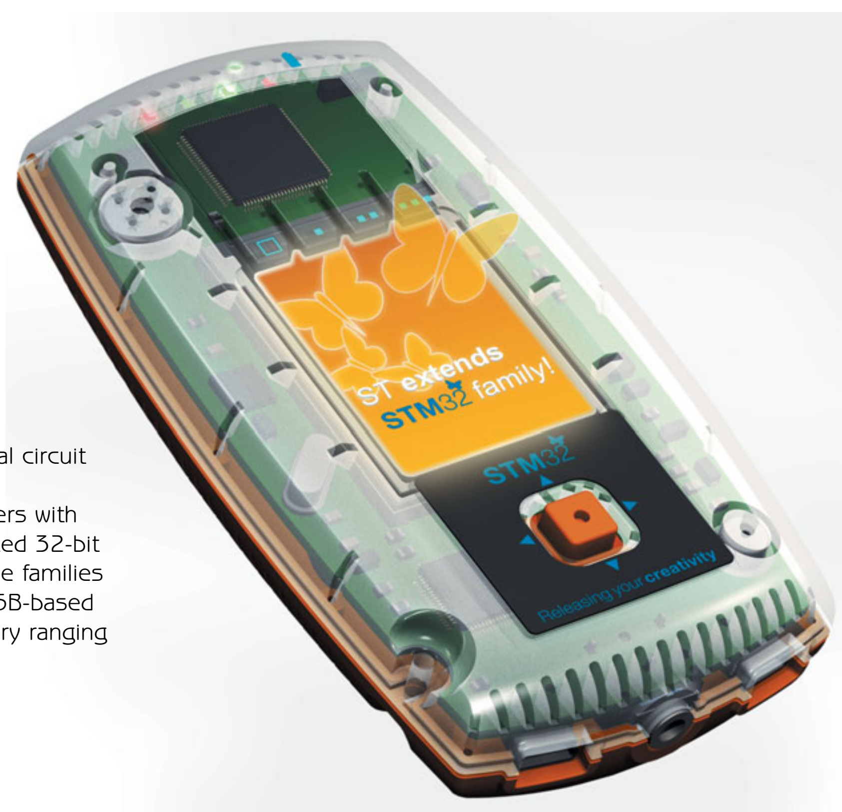
The Touch Library is currently available for download at the product web page together with the AVR QTouch Studio, royalty-free front end software supporting the two available demonstration boards TS2080A and TS2080B, supporting ATmega88, ATtiny88, ATmega88PA, ATmega168P and ATmega328P, respectively.

The Touch Library is available at no additional cost.

Atmel Corp.
www.atmel.com

HAND-HELD PROTOTYPING TOOL FOR SMALL COMPUTING DEVICES

The latest **STM32 Primer2** is about the size of a cell phone and comes with a 128 × 160-pixel color touchscreen, a joystick/push button, and an extension connector. It is preloaded with sample applications. There is also a built-in USB port, a MicroSD card slot, and other features (e.g., an accelerometer and an infrared transmitter/receiver) that enable users to quickly add numerous extra functions to their applications. All of these components are easy to manage through the built-in software framework, the open-source CircleOS.



Besides providing a turnkey solution that's ready to run sample applications, the device comes with Raisonance Ride7 application development software and the GNU C compiler. This enables embedded designers to use the platform for the complete development of complex end products. Necessary design resources—such as source code for the preloaded applications and software libraries for peripherals—are included. The STM32 Primer2 also has a 20-pin external connector that makes it easy to connect external circuit boards.

The STM32 Primer2 enables product developers with diverse skills to access the power of the advanced 32-bit STM32 microcontroller, which is available in three families targeting cost-effective, high-performance, or USB-based applications. Available devices have flash memory ranging from 16 to 512 KB.

The STM32 Primer2 is available for **$49**.

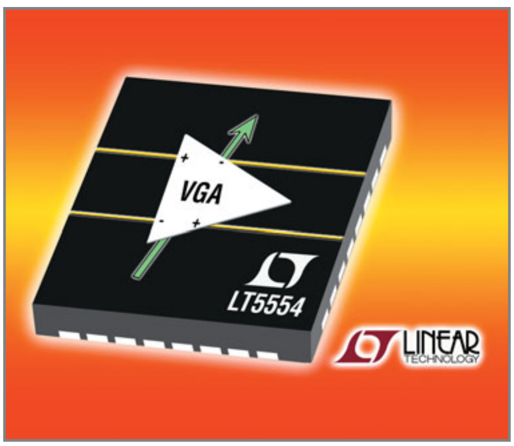
STMicroelectronics, Inc.
www.st.com

SUPER-LOW-DISTORTION DIGITALLY PROGRAMMABLE IF VGA

The **LT5554** is a broadband, digitally programmable gain IF amplifier, featuring a 48-dBm OIP3 (output third-order intercept) at 200 MHz. The amplifier has low noise, enabling high dynamic range performance in wireless communication receivers and signal processing systems. Its gain is digitally controlled from 2 to 18 dB by a 7-bit parallel word, producing the finest 0.125-dB steps gain control granularity of any amplifier available. The amplifier settles in less than 5 ns from a gain change, producing low glitch noise and supporting fast and accurate AGC performance. Its output stage has robust capability to drive into 50-Ω loads with low distortion and noise.

Unique to the LT5554 is its OIP3 performance, which remains consistently high over a wide varying signal level range. In contrast, the closest competing solution is limited to only a narrow ±1-dB signal level range. With today's new generation of wireless technology, such as LTE (Long-Term Evolution) and WiMAX, which use modulated signals with a 12-dB crest factor, the need to maintain linearity performance over wide operating levels is paramount for producing consistent base station performance.

The LT5554 is a full differential input and output amplifier. Its differential outputs can drive directly into 50-Ω loads. The amplifier may be overdriven and can recover quickly in less than 5 ns. Additionally, the amplifier has excellent reverse isolation. These characteristics make the LT5554 an ideal amplifier to drive high-speed, high-resolution ADCs, where the input sampling noise can couple back to the RF and IF circuits.



The LT5554 starts at **$4.40** each in 1,000-piece quantities.

Linear Technology Corp.
www.linear.com

QUICK-CONNECT WIRING SYSTEM

The new **M16 powerfast** wiring system is specifically designed for machine power distribution and motor control. The quick-connect M16 powerfast system provides a time- and cost-saving replacement for traditional hard-wiring installations and complies with NFPA 79: Electrical Standard for Industrial Machinery. These two-, three-, and four-pin connectors and tees provide up to 18 A in a compact form factor.

The cordsets are offered with Tray Rated, exposed run PVC flexlife cable. All connectors deliver IEC IP 67 protection and are rated for 600 V and up to 18 A. Tees are available with simple connectors or with branches.

Like most TURCK cordsets, the M16 powerfast line offers male or female options, straight connectors, standard and custom lengths, and pigtails or extensions. To complete the system, fully encapsulated mating receptacles with nickel-plated brass housing and 0.5" to 14 NPT, 0.375" to 18 NPT, M18, and M20 mounting threads are available.

Prices are dependent on the configuration, please contact TURCK directly.

TURCK, Inc.
www.turck-usa.com

NPN

JUMP-START 32-BIT REAL-TIME CONTROL

Two new USB-based tools for Piccolo 32-bit TMS320F2802x MCUs make it easier for designers to evaluate Piccolo MCUs and develop more energy-efficient, real-time control applications, such as solar power microinverters, LED lighting, white goods appliances, and hybrid automotive batteries.

The **Piccolo controlSTICK** helps you evaluate and learn about Piccolo MCUs. Packaged in a small, USB stick form factor; it features an onboard USB JTAG for quick and easy project evaluation. Access to all control peripherals is provided through header pins. A 32-KB limited Code Composer Studio IDE and example projects are available to help speed development.

Also available is the **Piccolo USB Experimenter's Kit**, which features a prototyping area and onboard USB JTAG emulation with connection for external emulator and power supply. The Experimenter's Kit offers the next step in development by enabling you to prototype and develop full Piccolo MCU-based projects quickly and easily. It also provides access to all the Piccolo pins.

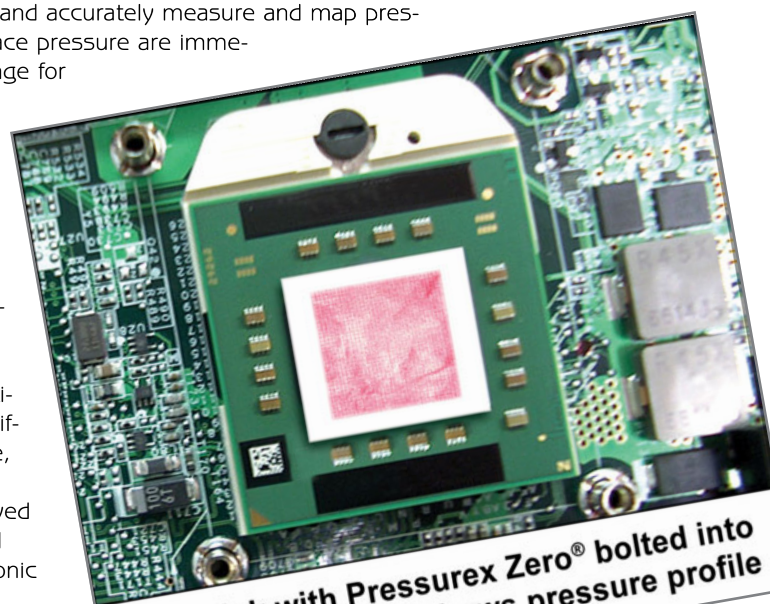
The Piccolo controlSTICK costs **$39**. The Piccolo USB Experimenter's Kit costs **$79** and includes the **$49** Piccolo controlCARD.

Texas Instruments, Inc.
www.ti.com

NPN

SENSOR FILM MEASURES LOW CONTACT PRESSURES

Pressurex Zero is an easy-to-use, tactile pressure-indicating sensor film. Pressurex Zero characterizes contact surface pressure down to an extremely low 7.2 psi (0.5 kg/cm^2). It is simply placed between contacting or mating surfaces to instantly and accurately measure and map pressure magnitude and distribution. Variations in contact surface pressure are immediately visible by the impression made on the film. The range for Pressurex Zero is 7.2 to 28 PSI (0.5 to 1.97 kg/cm^2).



Heat sink with Pressurex Zero® bolted into place to apply load shows pressure profile

Pressurex comes in the form of a thin plastic sheet (4 or 8 microns), physically similar in thickness to paper, and is available in eight different pressure ranges. When placed between contacting surfaces, it instantaneously and permanently changes color directly proportional to the actual pressure applied. The precise pressure magnitude (psi or kg/cm^2) is easily determined by comparing color variation results to a color correlation chart (conceptually similar to interpreting Litmus paper). Pressurex can also be scanned through one of Sensor Products's optical imaging systems. The film, which is available in eight different pressure ranges, is used in the design, manufacture, calibration, and quality control of many products.

Pressurex is flexible, which enables it to conform to curved spaces. It is ideal for invasive, intolerant environments and tight spaces that are not accessible to conventional electronic transducers.

A 270 mm × 3 m roll of Pressurex costs **$698**.

Sensor Products, Inc.
www.sensorprod.com

NPN

by DJ Delorie

Construct a USB GPIO Pod (Part 1)

No Parallel Port, No Problem

Unless you've been in a cave for the past decade, you know that the parallel port has been eclipsed by the USB port. In this article, DJ describes how to design a general-purpose input output (GPIO) pod that can plug into your USB port. Now you will have the flexibility to define the I/O you need.

I have been fiddling with electronics since before the arrival of the PC. In the early 1980s, it was easy to add custom electronics to a computer. The S100 and ISA busses were easy enough to interface with; but for ease of hook-up, nothing beat the parallel port with 13 output pins and five input pins, which was directly addressable in software and TTL-compatible. These days, it's getting hard to find parallel ports on computers, especially on laptops. Even when you can find one, it is not always compatible with today's 3.3-V logic. The USB port has taken the top spot in popularity.

In the first part of this two-part article series, I'll show you how to make a general-purpose input output (GPIO) pod that plugs into your USB port (see Photo 1). Next month, I'll describe how you can use it to program a CPLD.

USB OVERVIEW

There have been plenty of articles about USB, so I'll just cover the basics. USB is a high-speed serial port with a power feed and only four wires: two for data, one for 5 V, and one for ground. A packet protocol is used to communicate between the host and the devices, much like PPP or Ethernet is packet-based. In addition, the USB host can provide up to 500 mA on the 5-V line, as long as the device has requested it and the host has agreed to provide it. This makes USB a much more flexible connection, but more complex to interface with. Fortunately, there are interface chips that hide this complexity and offer a standardized interface.

The pod's first major component is a Future Technology Devices International FT232R USB-to-UART interface chip. On the USB side, it handles the physical and logical interface to the USB serial lines and manages power to the remainder of the device, if needed. On the UART side, it provides a standard asynchronous serial port, with full flow control and five additional user-assignable I/O pins (CBUS0 through CBUS4).

FTDI provides two libraries that can be used to interface to these chips. The first treats the chips strictly as standard UARTs. The second provides access to the chip's full capabilities. The libraries are available for both Windows and Linux, but the UART library usually is not needed. Both operating systems recognize the chips and automatically load UART drivers for them when a USB device is first plugged in.

R8C OVERVIEW

The second major component in the GPIO pod is a Renesas Technology R8C microcontroller. It is a 16-bit microcontroller that can run at either 3.3 or 5 V, at up to 20 MHz, with built-in flash memory, RAM, and a wide variety of built-in peripherals. While the FT232R chip has user-controllable I/O pins, using a microcontroller gives you much more

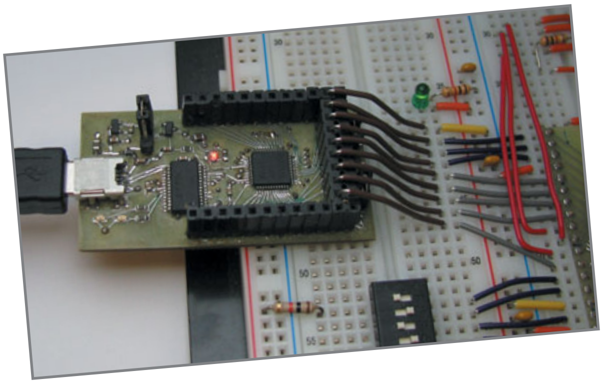


Photo 1—The pod can be used to interface your host PC to a project. Here it provides input data to a CPLD project.

speed and flexibility. For example, the R8C has an ADC, so you can use it for measuring analog signals. You can also download software into the R8C to turn it into a "smart" I/O pod, which will enable you to optimize the USB serial datastream. Next month, I'll explain how I downloaded an entire JTAG programming application into the pod and sent the compressed CPLD bitstream to it through the serial port.

There are a couple of reasons why I chose the R8C. You may remember the 2005 Renesas M16C Design Contest. The R8C is the M16C's little brother, available in packages with as few as 20 pins. It has two characteristics that make it ideal for this project. One, it can run off either 5 or 3.3 V, enabling it to interface to a wider range of projects. Two, you can program it using nothing more than a serial port and two GPIO signals, one for nRESET and one for MODE.[1]

I used DTR to control nRESET, CBUS2 for MODE, and feed the UART lines into the R8C's serial port. By doing this, you can use the USB connection to program the R8C and communicate with that program, so no additional programming connectors are needed! How does this work? Well, the R8C (and M16C and M32C, as well) has two flash memory blocks. One is for the user program, which is the flash memory that is described in the chip's hardware manual. The second is a small flash memory block that includes a simple bootstrapping program. When the chip comes out of reset (the nRESET line goes high), it samples the MODE pin to determine from which flash memory block to boot. Normally, MODE is pulled high, and resetting the chip runs your program. If MODE is low, the bootstrap program runs instead, enabling you to download a new program.

The specific R8C chip used here is the R8C/20 chip in a 48-pin TQFP. It has six byte-wide I/O ports. Four ports (Ports P0, P1, P2, and P6) expose all 8 bits to pins on the chip, while two (P3 and P4) expose only 6 of the 8 due to the limited number of pins. Port P6 is used for the seven UART signals (DTR goes to nRESET), leaving 1 bit to control an on-board LED. Ports P0, P1, P2, and P3 are brought out to standard 0.1" headers. Two of the P4 pins are used to make up for the missing P3 bits. Two of the remaining P4 pins are used for the ADC reference voltage and the clock.

I chose this chip because it has sufficient I/O pins to do nearly anything I can imagine. P0 has eight ADC inputs. P2 has full three-phase motor control capabilities. P1 offers six interrupt inputs, four more ADC, and a dual-mode (synchronous and asynchronous) serial port. P3 has an SPI/I^{2}C interface and two timer outputs. As you can see, there are a lot of peripherals for project-specific interfacing and intelligence.

THE USB GPIO POD

Given what I have covered so far, the pod almost designs itself. The FT232R chip lets the host talk to the R8C, either to program it or to communicate with it. The R8C controls

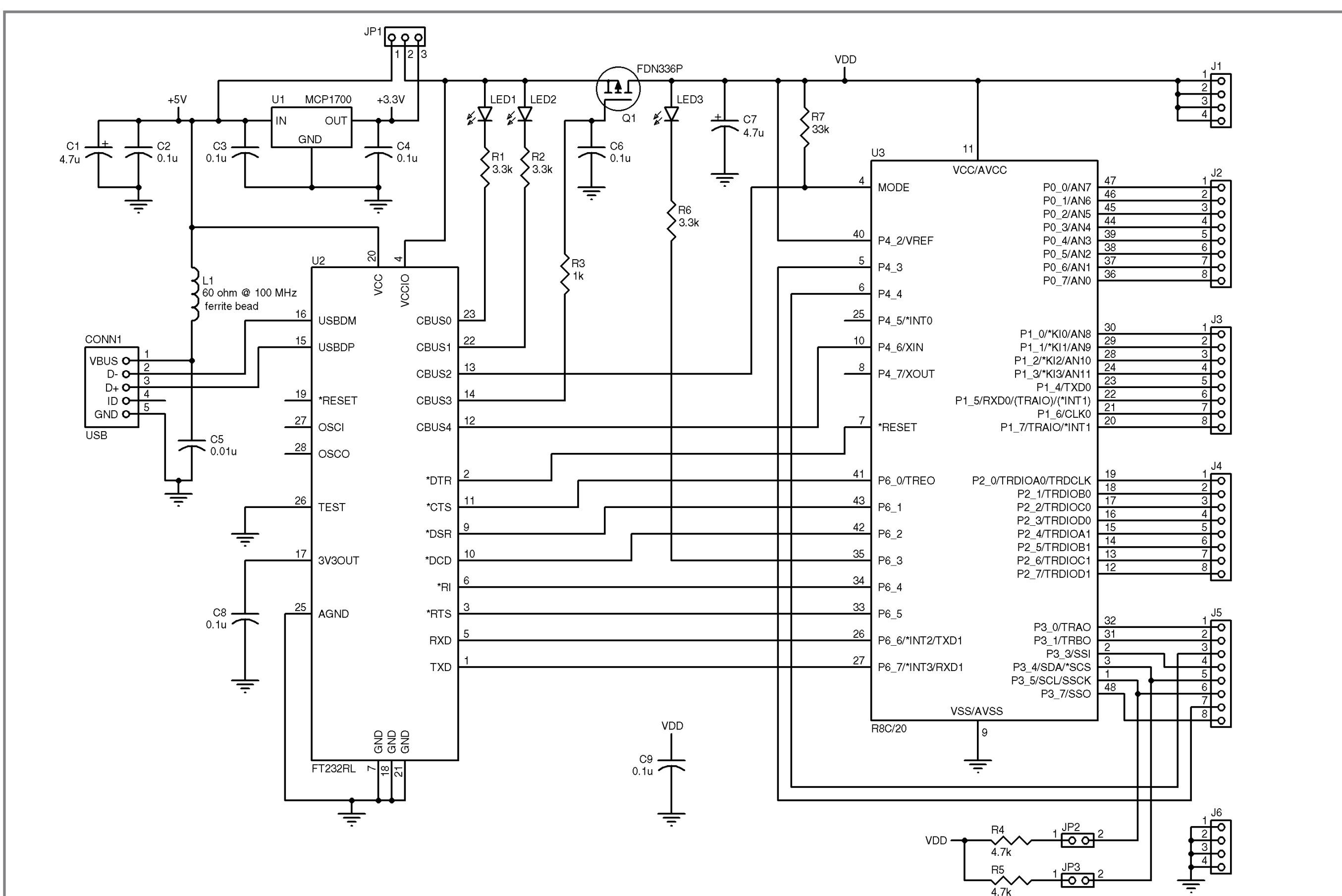


Figure 1—Note the simplicity of the circuit. Most connections go between the R8C (U3) and either the USB chip (U2) or the headers. The rest of the circuitry is mostly for power control. There are also LEDs and jumpers for I^{2}C pull-ups.

the various I/O ports on the headers according to its programming. The headers include power and ground, so the pod can power your project and talk to it. The headers accept standard 22-gauge solid wire, just like a solderless breadboard. Or you can make modules that plug onto the headers like a daughter board to add additional circuitry or change the connector pinouts depending on the project (see Photo 1).

I chose a small board—just over 2″ × 1″—so it could be used more like a "pod" and less like a "board." It's small enough to be considered part of the USB cable. One end of the board is the USB connector and power management circuit, followed by the FT232R chip. The other end is the R8C chip surrounded by headers. The wiring also follows this flow: the USB connector is wired to the FT232R and power circuits, the FT232R talks to the power circuits and the R8C, and the R8C connects to the headers (see Figure 1).

Listing 1—This is a simple pod application that toggles output pins.

```
#include "bsp.h"

main()
{
   unsigned char j;

   /* Configure port 1 for all GPIO output.  */
   pd1.b = 0xff;

   while (1)
   {
      j ++;
    /* Write the new value out to port 1.  */
      p1.b = j;
      wait_ms (100);
   }
}
```

The USB connector provides 5-V of power to the device. However, there are some rules about using this power. The device is limited to 500 µA in "suspend" mode and 100 mA during USB negotiation. If the device and host agree, the device may use up to 500 mA, although the pod is rated for only up to 250 mA. The pod has a 3.3-V regulator U1 and a jumper to choose between 3.3- or 5-V operation. The jumper controls power to the FT323R's UART pins, the R8C chip, and the headers. The jumper's output is switched with a P-MOSFET controlled by the FT232R, so the FT232R can keep the R8C powered off until it and the host agree on power requirements.

The FT232R chip manages the USB connection. When you plug in the USB cable, it negotiates the data rate (up to 12 Mbps) and power needs with the host. The FT232R has an internal EEPROM that contains configuration information, such as device identification and power needs. I programmed mine to be named "usb-r8c," and I asked for 250 mA, using utility software available at FTDI's web site. Once the power is negotiated, CBUS3 is pulled low to activate Q1, supplying power to the R8C chip and headers.

The connections between the FT232R and the R8C are fairly straightforward. The UART signals go to P6, except DTR, which is used for nRESET. I took care to match the Tx and Rx pins so the R8C's internal UART can be used to communicate with the FT232R. CBUS2 is used to control the MODE pin. CBUS4 provides a 12-MHz clock from the FT232R to the R8C. An external crystal is not required. Because the R8C runs at up to 20 MHz, you can use a separate crystal to increase its performance. Twelve megahertz is just the fastest clock available from the FT232R that doesn't exceed the R8C's limits. You could also use an 18.432-MHz crystal if you need accurate UART signals, because 18.432 MHz is 160× faster than 115,200 bps, and thus is the fastest clock under 20 MHz that results in an exact divisor in the UART clock control register. The UART needs a clock 16× faster than the data rate, and 115,200 × 16 × 11 is 20.275 MHz. However, running off the FT232R's clock enables you to coordinate data rate divisors between the FT232R and the R8C, allowing fast communication with the host. Finally, you can use the R8C/20's internal oscillator, which is the equivalent of a 20-MHz crystal but is not as precise.

On the other side of the R8C, I just connected each pin to one of the header pins. Because P3 is missing 2 bits, two of P4's bits fill in, providing 32 I/O pins in four groups of eight. Additionally, four ground and four power pins are provided in the headers. Because I use I^2C a lot, I included two I^2C pull-up resistors and jumpers to enable them. There is also an on-pod LED connected to pin P6.3, which can be used for diagnostics. The pod's host-side serial port driver in my software uses this LED to reflect the flow control status between the host and the pod.

PROGRAMMING & OPERATION

The pod can be used in one of two modes: Programming or Operation. In both modes, the R8C is reset by temporarily dropping DTR. Because the Linux built-in FTDI drivers do this automatically when the data rate is set to zero, it can be done easily whether you use the built-in drivers or the FTDI library. Note that the FT232R has programmable polarity for the serial port signals. I programmed the DTR line to be active high, so the chip is reset when DTR is dropped and runs when DTR is asserted. DTR is always driven, so no pull-up resistor is needed.

In Programming mode, the R8C MODE line is held low during reset by using the FTDI library to set CBUS2 low, putting the chip in Bootstrap mode. Normally, the CBUS2 pin is tristated, so a pull-up forces the chip into normal running mode when the FT232R is initially plugged in. Thus, only the programming software needs to know about the extra functionality. Once in Programming mode, the programming utility talks to the bootstrap firmware over the standard serial lines to download the new software into the R8C flash memory. It can then raise CBUS2 and reset the R8C to normal running mode.

Once the R8C is programmed, a host-side utility talks to the R8C using either the FTDI library or, more likely, a standard serial port emulation. In Linux, for example, plugging in the pod causes a device like /dev/ttyUSB2 to be created, which can be accessed like any

Listing 2—This is a more complex pod application, showing how to interact with the host PC and the on-chip peripherals.

```
main()
{
  int period, width, channel;

  setup_hardware ();

  while (1)
    {
      switch (tty_readc ())
    {
    case ‘o’: /* output bits */
       p2.b = tty_readc ();
       break;
    case ‘i’: /* input bits */
       tty_putc (p0.b);
       break;
    case ‘a’: /* sample ADC */
       channel = (tty_readc () & 0x03) + 8;
       tty_putc (read_adc (channel));
       break;
    case ‘t’: /* timer */
       period = tty_readc ();
       trbpr = period / 2;
       trbsc = (period + 1) / 2;
       break;
    case ‘T’: /* timer, with width control */
       period = tty_readc ();
       width = tty_readc ();
       trbpr = width;
       trbsc = period - width;
       break;
    }
    }
}
```

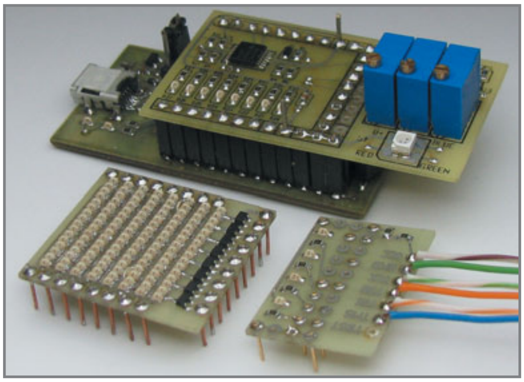


Photo 2—In addition to plugging wires into the headers, you can create a variety of modules that plug into them. Here you see the pod with my LED workbench module, which is used to characterize RGB LEDs. A raster display (lower left) and a JTAG adapter (lower right) are also shown.

other serial port. The actual protocol to use depends on the firmware programmed into the R8C, allowing great flexibility in customizing it for whatever purposes the pod is used for.

I did what everyone does first with circuits like this. I made a blinky light. But before I discuss my application, let me take a moment to discuss what goes on behind the scenes. Each piece of hardware needs a board support package (BSP), which includes all the ancillary support routines that are specific to your hardware (e.g., how to set up the stack, initialize memory, program timers, and more). All of the support files needed for this board are posted on the *Circuit Cellar* FTP site. I won't cover them further in this article, but assume that each program I cover here needs to be linked with the BSP files. My first pod program is in Listing 1.

First, I had to compile this from source format into a binary format that my programming tool would understand. I used the freely available GNU toolchain (specifically, GNU's GCC and Binutils, and Red Hat's Newlib) because it runs on Linux, my platform of choice. Instructions for obtaining and building the GNU toolchain for the M32C family posted on the *Circuit Cellar* FTP site. Each tool's name is prefixed with "m32c-elf-" to indicate that it's for cross-compiling to the M32C family and produces ELF format binaries, which is the most popular embedded file format these days. The Makefile posted on the *Circuit Cellar* FTP site uses m32c-elf-gcc to build the blinky.c into blinky.elf.

The programming tool is called uflash (for USB-based FLASHing tool). Running it downloads blinky.elf to the pod:

```
$ sudo ../uflash/uflash blinky.elf
```

USB devices are not normally writable by users, so the `sudo` command gives me permission to access it. The uflash tool talks to the bootloader in the R8C chip to program it. When finished, it resets the chip back into normal mode, and my blinky program runs. Plug a few LEDs and current-limiting resistors into port 1's headers and you've got blinky lights!

"Big deal," you say.

Well, I did something more interesting and made the pod interactive. I created a pod program that offers eight output signals on port 2, eight digital input signals on port 0, four analog inputs on port 1, and a clock output on port 3. All of these are controlled by the host PC. The more complex pod program is in Listing 2.

The call to `setup_hardware()` configures all the I/O ports and peripherals. It's handy to have a copy of the "R8C/20 Group, R8C 21 Group Hardware Manual," on hand, because there are many complex peripherals. For example, in this case, Timer B drives pin P3.2, but the pin can't be driven in standard timer mode, so configure it for function generator mode. That also lets you control both the period and the duty cycle. These details, along with the myriad control registers that configure them, are all detailed in the manual.

All pod programs need some sort of "loop forever" in `main()`, because there's nothing to return to. In this case, the peripherals are doing all the work, so my main loop is used to wait for data from the host. The protocol is simple. A command byte is followed by zero or more data bytes, and it may cause zero or more data bytes to be sent back to the host. The `O` command is followed by a byte, which is then sent

to port 2. The `I` command causes the pod to read port 0 and send the byte back to the host. The `a` command is followed by a byte that selects one of the four ADC inputs on port 1, reads the ADC value as an 8-bit value, and sends that back to the host. The `t` and `T` commands are followed by 1 or 2 bytes that get programmed into Timer B.

PODS AND MODULES

In addition to connecting wires directly to the headers, I have three modules I made for my pod (see Photo 2). One module is an "LED Workbench" that lets me experiment with red and RGB LEDs. One has an 8 × 8 grid of red/green cells. The third, which I'll cover in the second part of this article series, is a JTAG adapter. Each module's design starts with a schematic for the connectors and a PCB layout that places them to mate with the pod. Thus, I can quickly design a new pod module as needed.

The LED workbench includes eight red LEDs controlled by port 2 and an RGB LED controlled by port 1. The RGB LED has adjustable limiting resistors to help you determine the ideal resistors for a good white balance. The ADC inputs monitor the LED voltage drops as well as the voltage across the resistors. Three 10-Ω resistors and three op-amps let the pod monitor the current through each LED. This lets me test and characterize various RGB LED offerings. The pod software monitors the ADCs, disables the LEDs if excessive current is detected, and scales all the values to sensible values. It includes a command-line interface so any terminal emulator program that can talk over the serial port is sufficient to talk to the pod. I can also experiment with various LED control algorithms with the eight red LEDs.

Another module is a raster of red and green LEDs, creating an 8 × 8 grid of red/green pairs. (Yes, that's 128 LEDS, along with eight transistors and 16 resistors, in a 1″ square space.) This forms a 64-pixel micro display, with port 1 controlling the rows, port 2 controlling the green LEDs in each column, and port 0 controlling the red LEDs in each column. I use this as a scrolling marquee display for monitoring data from my PC. The pod software maintains an 8 × 8 array of what should be displayed, 1 byte per LED. The timer interrupt iterates through each row, enabling a single bit in port 1 and outputting appropriate data to ports 0 and 2, according to the desired intensity of each LED in that row. Meanwhile, data from the host PC is used to fill in new columns of data. When a column has been received, the data for the display is shifted one column over, providing a scrolling marquee style display, with data provided by the host but LED control handled by the pod.

In addition to these, you could use the pod to interface with SPI, I²C, 1-Wire, or serial devices. The R8C's internal ADC could be paired with a D/A chip or two. The pod has a fully functional and programmable computer chip on it, so the possibilities are endless!

YOUR TURN

If you need to turn a bit on and off, this isn't the fastest way to do it. But if you can offload some of the logic to the pod and do a lot more than just turning a bit on and off, this project gives you the flexibility to define the I/O functionality you need.

Next month, I'll show you how to offload an entire application to the pod and feed only its data files over the USB link. Stay tuned. ■

DJ Delorie (dj@delorie.com), who has been designing electronic circuits since high school, earned an ECE degree at Clarkson University. After holding jobs designing PC motherboards and network management software, he now writes embedded development tools for Red Hat. DJ is also the creator of DJGPP and one of the contributors to the gEDA project.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/225.

REFERENCE

[1] D. Delorie, "Easy R8C/M16C/M32C Flash Programming," 2008, http://people.redhat.com/dj/m32c/flash-guide.pdf.

RESOURCES

D. Delorie, "USB GPIO Project Page," www.delorie.com/electronics/usb-gpio.

GNU Operating System, Free Software Foundation, Inc., www.gnu.org.

Renesas Technology Corp., "R8C/20 Group, R8C 21 Group Hardware Manual," 2008, http://documentation.renesas.com/eng/products/mpumcu/rej09b0250_r8c2021hm.pdf.

SOURCES

GNU Binutils
Free Software Foundation, Inc. | http://sourceware.org/binutils

FT232R USB UART IC and MProg 3.5 EEPROM Programming utility
Future Technology Devices International | www.ftdichip.com/Products/FT232R.htm

Newlib
Jeff Johnston | http://sourceware.org/newlib

R8C Microcontroller
Renesas Technology Corp. | www.renesas.com/en/r8ctiny

GCC Operating system
The GNU Compiler Collection | http://gcc.gnu.org

Solar Data Logger (Part 1)

PCB Layout, Inductor Saturation, and Other Troubles

Every engineer knows projects don't always work out as planned. Ed's goal was to build a data collection board that could automatically characterize a solar panel, record the panel's output over the course of several days, and then compare the results to his manually collected data. Unfortunately, he experienced some problems. Read on to learn why you should always have a proper-sized board.

ABOVE THE GROUND PLANE

by Ed Nisley

Steve Ciarcia's fundamental requirement for my columns, right from the first issue, has always been that the projects must *work*. If I didn't build it and verify the results, I couldn't write about it.

With that in mind, the plan for this column started simply enough: build a data collection board that could automatically characterize a solar panel, record the panel's output over the course of several days, then compare the results to my manually collected data. Along the way, I would describe the analog circuit problems that come with low-level measurements near digital circuitry.

As Publius Syrus put it back in the day, "Homo semper aliud, fortuna aliud cogitat," which my atrophied Latin translates as "Man always has intentions, but Fortune intends otherwise." The Yiddish equivalent has more punch: "If you want to make God laugh, tell Him your plans."

Basically, the board I designed not only *didn't* work, it resisted some protracted debugging. Rather than bluff my way through, I'll begin by describing what I wanted to accomplish, then explore what went wrong. You'll certainly learn something along the way; perhaps what you shouldn't do in similar circumstances.

In my next column, you'll see how to get it right. Honest, Steve!

THE BIG PICTURE

As you saw in my February 2009 column, characterizing a solar panel requires measuring the panel's voltage while applying a known load ("Solar Measurements," *Circuit Cellar* 223). My manual measurements used voltmeters, ammeters, and a pair of old potentiometers, so automating the process requires putting those functions on a circuit board under computer control.

To that end, the schematic in Figure 1 has three main sections: a microcontroller interface, a constant-current load, and a boost-mode switching converter. Careful inspection of the small circuit board clamped in Photo 1 shows that the board, an early prototype, doesn't quite match the schematic.

The PC board plugs directly atop an Arduino Diecimila microcontroller board through a quartet of pin strips. The Diecimila features an Atmel ATmega168 microcontroller with digital and analog I/O, a regulated power supply, and a USB serial interface to a host PC. The open-source Arduino IDE runs on both Windows and Linux, with support for several different boards and chips.

The two op-amps on the right side of the schematic form a voltage-to-current converter that acts as a variable load for the solar panel. The current through Q2, a Darlington power transistor, matches the voltage produced by the ATmega168, with an overall gain of 100 mA per volt. R12A and R12B, which form a 0.5-Ω current-sensing resistor, set that gain along with the 20:1 voltage gain in U4. U3 forces Q2 to sink enough panel current to make the feedback voltage match the control input voltage.

My panels can produce, at most, a few hundred milliamps in full sunlight and, with a peak power output around 1 W, that

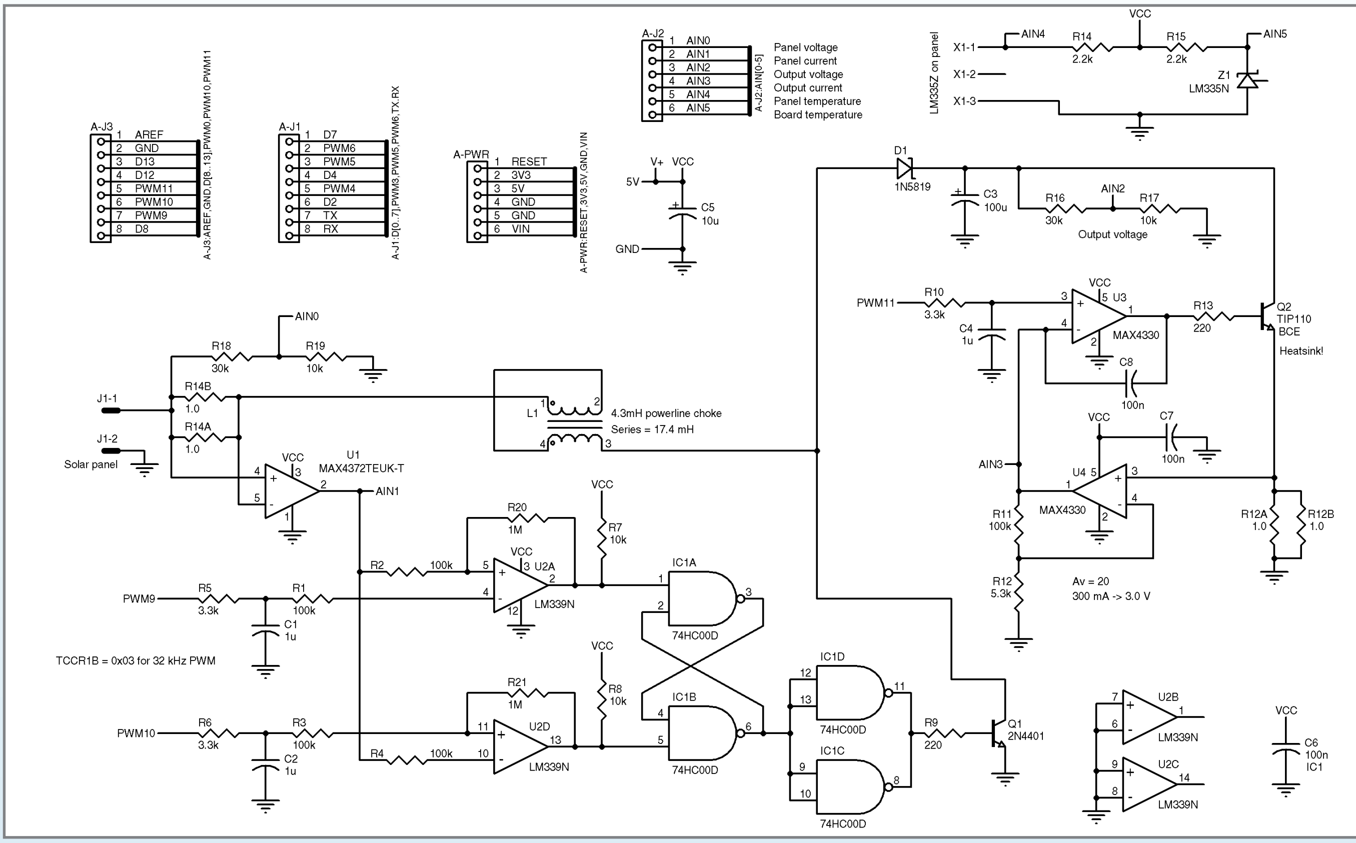


Figure 1—As with most analog circuits, getting anything done requires plenty of parts. The boost circuitry on the left runs the panel at its optimum current, while the current sink on the right dissipates the collected power as heat in the load transistor. The Arduino microcontroller, represented by the four connectors at the top, provides PWM voltage setpoints to control the booster and load, then reads analog data from the sensors. This circuitry does not match that shown in Photo 1 for reasons described in the text.

power transistor won't get very hot at all. Characterizing a larger panel would require a far more robust active load, which I'll leave as an exercise for the interested reader.

The Arduino board runs with V_{CC} = 5 V and its ADC inputs must not exceed that level, so the resistor pair R18-R19 presents 25% of the panel voltage to the microcontroller. The R16-R17 pair does the same thing to the transistor's collector voltage. The open-circuit voltage of both my panels is under 10 V, but the booster circuit described below can reach nearly 20 V, so some headroom is in order.

Characterizing the panel requires applying a series of load currents and measuring the resulting panel voltages. This circuit can't quite produce a true short-circuit load, because the voltage drop across D1, plus Q2's saturation voltage, plus the voltage across the current-sensing resistors, adds up to about a volt, but there's a way around that.

As you saw in December, the power drawn from a solar panel depends on the incident illumination and the load current. One specific current extracts the maximum power from the panel for a given illumination level: drawing a higher or lower current produces less power. Because solar power is so hard to come by, there's a strong motivation to always run the panel at its Maximum Power Point.

The standard solution requires a DC/DC converter drawing whatever current will produce the maximum power at whatever voltage the panel supports, while producing a more-or-less constant output voltage. That "raw DC" feeds the application circuit's power supply, much as line voltage feeds an ordinary PC.

For example, an 80-W solar panel might produce 4.5 A at 18 V in full sunlight. That current will decrease as the illumination changes, so the MPP controller must adjust the current going into the DC-to-DC converter, while maintaining a constant output voltage by tweaking the duty cycle. That requires monitoring the actual panel current and output voltage while adjusting the switching levels, a process known as MPP tracking.

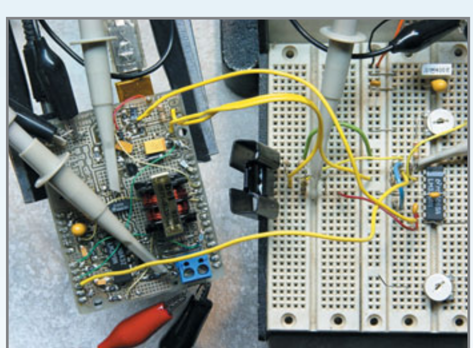


Photo 1—The solar data collector fits atop an Arduino Diecimila microcontroller board. The spillover on the solderless breadboard shows that not everything works correctly the first time.

The simple boost-mode converter circuitry on the left side of Figure 1 can do exactly that, albeit on a small scale and with relatively low efficiency. Transistor Q1 switches current through inductor L1 and then dumps it into C3 through D1. The Maxim Integrated Products MAX4372T high-side current amplifier has a gain of 20 to produce 1 V/100 mA of panel current, the pair of comparators in U2 set the high and low current limits, and a simple R-S flipflop built from the NAND gates in IC 1 turns Q1 on when the current falls

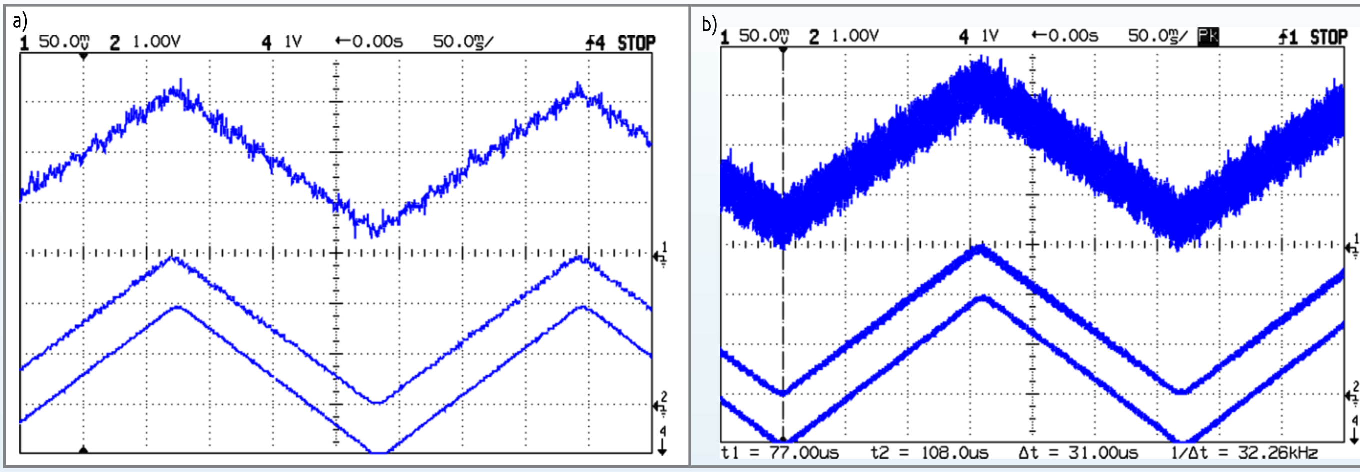


Figure 2—The upper trace shows the voltage at the top of R29/R30, the 0.5-Ω current-sense resistor, tracking the 0- to 3-V control input in the lower trace. Turning on peak-capture mode reveals the hash generated by the microcontroller through the power supply. The middle trace in each screen is the current feedback voltage from U4.

below the minimum and off when it exceeds the maximum.

The firmware can read the panel voltage and current through the microcontroller's analog inputs, compute the corresponding power, then set the high- and low-current limits to bracket the MPP and regulate the output voltage. As you might expect, there are all manner of patented MPP tracking schemes.

It's worth mentioning that MPP tracking can't produce something from nothing. A load that exceeds the solar input power will draw down the boosted DC supply from the panel, simply because there's not enough power available to support the output. The load must be smart enough to adapt itself to varying power inputs: full power in full sun, reduced power on overcast days, and sleep mode overnight.

Boost-mode converters have an obvious failure mode that's often not obvious until the smoke appears. Notice what happens should Q1 stay on for a protracted time: a high-power solar panel will roast R14A-B, L1, and probably Q1. A real-life implementation must include an interlock that holds Q1 off until the firmware gains control and sets the appropriate current limits, but I'll leave that as another exercise for the interested reader.

Pop Quiz: design such an interlock. Hint: a charge pump from a program-toggled bit may be helpful.

However, the firmware can turn Q1 on, measure the panel's short-circuit current, then turn Q1 off before the smoke appears, particularly for the low-power panels in my collection.

So that was the plan: characterize the panel using the current sink, warm up Q2 by running the panel at its MPP, and dump its measurements through the USB serial port to the host PC.

What could possibly go wrong?

CURRENT SINKHOLE

The single biggest mistake I made was attempting to squeeze everything into the same board footprint as the underlying Diecimila microcontroller. That size works perfectly well for digital projects, as shown by the many Arduino "shields," but poses problems for two-layer analog circuit boards. As you can see in Photo 1, the hulking inductor really takes over the board and forces the remaining parts away from their microcontroller connections.

The natural layout put the solar panel input (at the blue terminal block) on one end of the board and the current-sink transistor on the other, forcing return current from R14A-B to travel the length of the board past all of the op-amps and comparators. Although I poured a copper ground plane into all the unused

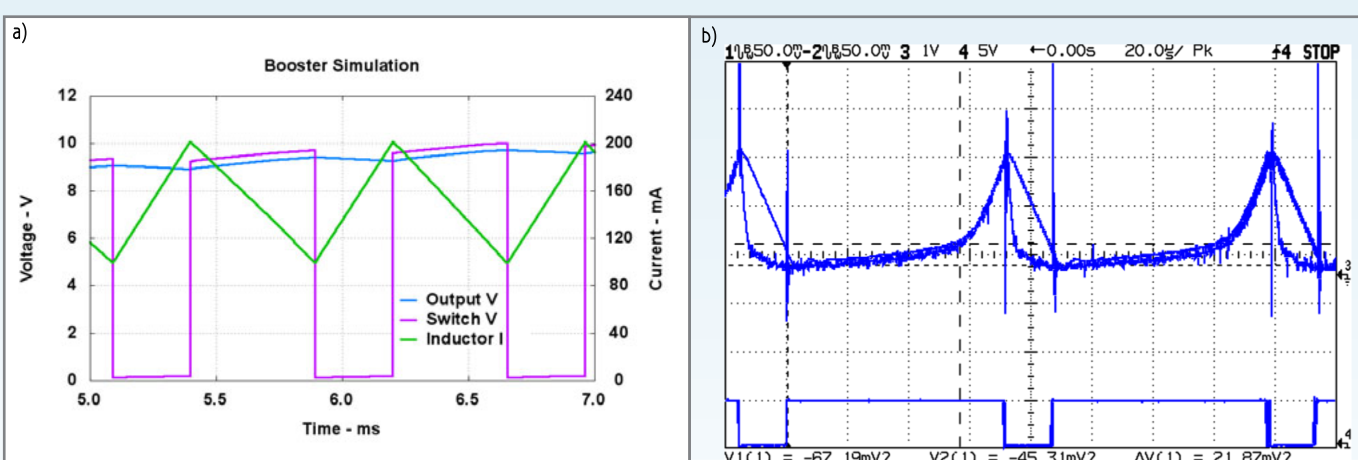


Figure 3a—The simulated booster circuit ramps between 100 and 200 mA at the di/dt rate set by the 17.4-mH inductor. b—The oscilloscope screenshot shows something's amiss: the actual inductor current has abrupt peaks due to core saturation. Trace 3 shows that the MAX4372 current amplifier can't keep up with the core flux collapse.


Saelig
UNIQUE PRODUCTS & SUPPORT
www.saelig.com
Testgear
Color LCD Scope
Best Seller
2-ch + trigger standalone USB bench scope. $325 / $599
PDS5022S / PDS6062T
2-ch 1GSa/s Scope
NEW! RIGOL
2-ch 1GSa/s (25GSa/s equiv.) 50/100 MHz scope. $595 / $795
DS1000E
Scope + Analyzer
NEW! RIGOL
25MHz 2-ch /16 logic scope and logic analyzer. $699
DS1022CD
Amazing 7 in 1 Scope! $180
CircuitGear CGR-101™ is a unique new, low-cost PC-based instrument which provides the features of seven devices in one USB-powered compact box: 2-ch 10-bit 20MSa/sec 2MHz oscilloscope, 2-ch spectrum-analyzer, 3 MHz 8-bit arbitrary-waveform/standard-function generator with 8 digital I/O lines. It also functions as a Network Analyzer, a Noise Generator and a PWM Output source – all for less than $180! What's more – its' open-source software runs with Windows, Linux and Mac OS's! Only $180
CGR-101
Handheld Scope
20MHz / 60MHz rugged handheld USB 2-ch scope. $593 / $699
HDS1022MN / HDS2062M
Pen Scope
10/25MHz USB powered scope-in-a-probe! Up to 100MS/s. $193 / $308
PS2104/PS2105
6 in 1 Scope
200kHz 2-ch 10-bit scope, 2-ch spectrum analyzer, 16-ch 8MHz logic analyzer, 5-ch sig gen, 8-ch pattern gen. $199
PoScope with Probes
Low-Cost Scope
2-ch 40/100/200MS/s 8-bit scope range with 5/10/25MHz. $297 +
PS2203/4/5
1/2GHz RF Generators
High accuracy/stability, wide range, low phase noise/leakage, serial control.
TGR1040 / TGR2050
Mixed-Signal Scope
100MHz Scope, + Spectrum/Logic Analyzer and Signal Generator. $1259+
CS328
USB Bus Analyzers
Best Value
Packet-Master™ - USB 1.1/2.0 analyzers and generators. $699 +
USB12 / 480+ / 500AG
16-Ch Logic Analyzer
Intuitive full-featured 16-ch 4MB 200MHz sampling memory. $299
LAP-16128U
SPI Bus Analyzer
Protocol exerciser/analyzer for standard SPI and non-standard 4-wire and 3-wire serial protocol interfaces up to 50 Mbps.
SPI Xpress
EMC Spectrum Analyzer
RF & EMF Spectrum Analyzer 1Hz to 7GHz for WiFi, mikes, etc.
EMC RF & EMC Spectrum
EMC Spectrum Analyzer
Handheld Palm PC-based 2.7GHz Spectrum Analyzer.
PSA2701T
"I really like this scope adapter - it's really meant for teaching electronic experiments but it's ideal for engineers too."
Alan Lowne
Saelig CEO
www.saelig.com
Waveform Generator
USB2.0 speed 16-bit digital pattern or arbitrary waveform generator.
Wave Xpress
I2C Xpress
Versatile USB 2.0 I2C protocol exerciser and analyzer.
I2C Xpress
Automotive Testing
Kits turn your PC into vehicle-electrics diagnostic tool.
PS3423 / KLARI-MOD
CAN Gateway
Janz - Full-featured standalone fanless industrial Linux PC.
emPC-x133
RF Generator
High-res, extremely low-noise, portable 3GHz RF generator.
APSIN3000
RF Testing/EMI Tents
Portable RF test enclosures & shielding tents with external frame.
RF Testing / EMI Tents
Wireless Data Loggers
Log and display temp, hum, volt, event-time or pulse-counting data
RTR-50
Multiparameter Loggers
Mini-logger with built-in temp/hum/pressure/3-axis accel sensors.
MSR145S
USB Logger
Standalone USB temp / hum / volt / current loop data logger. $49+
EL-USB-1/2/3/4
Electronic DC Load
Const. current, resistance, conductance, voltage & power modes
LD300
60/100/120MHz AWG
NEW! RIGOL
60/100/120MHz USB 14-bit ARB with USB RS-232, LAN/GPIB.
DG3061A/3101A/3121A
TorqSense
Configurable, patented USB-output non-contact SAW digital rotary torque transducers with integral electronics.
RWT320
Misc
USB to I2C
"Drop-in" solution connects PC to I2C/SMBUS + 32 I/O lines. $89
USBI2CIO
FTDI USB ICs
Lowest Prices
Popular UART and FIFO chips. Upgrade Legacy designs to USB.
FT232RL
CAN-USB
Intelligent CAN connection from PC's USB port. $299
CAN-USB
Serial-Ethernet Cable
Network serial product easily without a PC using this 28" cable. $89
eCOV-110-P
Lorlin Switches
Fantastic array of stock and custom switching devices.
Lorlin
PSoC Starter
Get going quickly with PSoC visual design environment.
PSoC Starter
Keyboard Simulator
USB board adds 55 I/O and 5 x 10-bit A/D inputs, 1 x 10-bit analog O/P.
PoKeys55T
Instant Ethernet
No OS needed. TCP/IP offload, ICs improve system performance.
WIZ110SR / W5100
Ethernet-IO
UDP/IP-controlled 24 digital I/O board 3 x 8-bit TTL ports.
Ether-IO 24
FPGA Systems
Ready-to-go out-of-the-box FPGA/DSP designs for beginners and experts!
RTG005
Wireless Solutions
Analog input, bluetooth wireless modules 433/868/915MHz.
EmbedRF / Adeunis
Temp/RH Sensors
Novel ambient sensors & modules accurately measure temp/RH.
UPSICAP / DLP-TH1
.NET Board
Small (2.2" x 2.2") lowest cost .NET Micro Framework dev system.
USBizi
Easy OLED Display
Compact, economical smart OLED with graphics drive from USB or RS232.
uOLED-96-G1
RF Modules
Simultaneously transmit composite video and stereo audio signals.
AWM6XX TX/RX
RS232 to 422/485
9p-9p or 25p-25p self-pwrd, isolated RS232-RS422/485
KK Systems
1/2/4/8/16 x RS232
Add 1-16 COMports via your PC's USB Port easily.
USB-COM
Quantum
Quickly add capacitive touch on / off & X / Y - sensing ICs.
Quantum
Saelig .com
unique electronics
888-7SAELIG info@saelig.com
Above are some of our best-selling, unique, time-saving products - see our website for 100s more: WiFi/910MHz antennas, wireless boards, LCD display kits, Ethernet/IO, USB/RS232/485, USB-OTG, instant Ethernet-serial, CAN/LINbus, USB cables/extenders, line testers, logic analyzers, color sensors, motion controllers, eng. software, wireless boards, SMD adapters, I2C adapters, GPS loggers, automotive testing, security dongles, video motion detectors, crystals/oscillators, custom switches, barcode scanners, DSP filters, PLCs, Remote MP3 players, etc. FREE Starbucks card with your $250 order! Check www.saelig.com often for special offers, bargains, business hints, blog, etc.

areas and stitched the top and bottom planes together, the effective resistance was high enough to induce offset and feedback voltages where they did the most damage.

I added a husky 16-AWG shunt wire directly from the resistors to the terminal block that somewhat reduced the problem. Despite that, I simply couldn't stabilize the current-sink op-amps, which continued to oscillate regardless of their compensation. I finally switched to LM324 op-amps, which have a much lower gain-bandwidth product, and moved the whole assembly to a solderless breadboard. Surprisingly, it worked fine, regardless of my usual dislike of that method of construction.

The oscilloscope screenshot in Figure 2a shows the current sink in operation. The low-pass filtered PWM signal in the bottom trace ramps from 0 to 3 V, which is precisely matched by the current-feedback signal from U4 in the middle trace. The actual voltage across R12A-B in the top trace, however, wears some fur throughout its entire range.

Switching to peak-detect mode reveals over 50 mV_{pp} of noise on the signal. That noise turns out to be high-frequency digital hash from the microcontroller that's entirely unrelated to its PWM outputs. Much the same noise appears on the other two traces, where it's barely visible as a slight thickening at 1 V/div.

Isolating digital and analog circuitry on a single board requires careful attention to power distribution and grounding. The Arduino Diecimila +5-V regulator supplies power to both the ATmega168 chip and the analog circuitry, with ground return paths through only three pins in the pin header strips. Although the Diecimila board has plenty of gridded areas, its compact layout chops up the ground connections.

My layout didn't fare much better. While I managed a broader connection between the ground pins on each side of the board, that lower impedance simply routed more digital hash from the Arduino board through my circuitry.

My revised layout will use a somewhat larger board that puts the relatively high-current areas outside the Diecimila footprint and the relatively delicate circuitry in a separate ground area. I'll also regulate 9- and 5-V supplies from a wall-wart transformer, with a single-point ground connection to the Arduino. That should reduce much of the hash and stabilize the op-amps.

MPP LETDOWN

The boost-mode DC-to-DC converter on the left side of Figure 1 stores energy in inductor L1 when transistor Q1 is on, then dumps that energy into C3 when Q1 switches off. The microcontroller can set the minimum and maximum inductor current limits to bracket the Maximum Power Point current, so that the average current is reasonably close to the ideal value.

The voltage across an inductor is proportional to the derivative of the current through it, according to the familiar equation:

$$v = L\frac{di}{dt} \quad [1]$$

When Q1 is on, it applies essentially the full panel voltage across L1, with the DC current limited only by the panel's power output and the current-sensing resistors. The current should therefore increase at a rate set by the equation:

$$\frac{di}{dt} = \frac{V}{L} \quad [2]$$

The Spice simulation in Figure 3a shows the results for a 17.4-mH inductor. The inductor current rises from 100 to 200 mA in about 300 µs, showing that the applied voltage is:

$$5.8\ V = \frac{100\ mA}{300\ \mu s} \quad [3]$$

When Q1 opens, the inductor current charges C3 as it declines toward the lower current setpoint. Because the output voltage changes only slightly, the current decreases nearly linearly.

I picked 17.4 mH because that matched the inductance of a common-mode power choke I found in my parts heap. In normal use, a common-mode choke has two windings connected to the AC line input so that the circuit's load current produces opposing magnetic fields; with no net field, the normal-mode current sees essentially zero inductance. Common-mode signals, usually caused by electrical noise, produce fields that don't cancel and therefore encounter an inductance that tends to filter out the noise.

The two windings each had 4.2 mH of inductance, so their series connection was 17.4 mH; remember that inductance varies as the square of the number of turns. The windings appeared to be 20-AWG copper, suited for currents of several amps. The ferrite core was a pair of C-shapes with a small gap, so I assumed saturation wouldn't be a problem with a few hundred milliamps of DC current.

The scope shot in Figure 3b shows the catastrophic error in that assumption. The upper trace actually consists of two overlaid traces: the differential voltage measured across R14A-B and the output of U1. The lower trace is the base drive signal from IC1C.

I set the current limits to 50 mA and 300 mA, but the current obviously isn't changing linearly. The current rises more-or-less linearly to about 60 mA, skyrockets to the 300-mA peak, and collapses almost instantly when Q1 switches off.

Eyeballing the flattest part of the curve, roughly the first half of the lower part marked by the cursors, where the slope seems to be:

$$800\frac{A}{s} = \frac{10\ mV / 0.5\ \Omega}{25\ \mu s} \quad [4]$$

Applied through a 17.4-mH inductor, that gives a voltage of about 14 V, more than a factor of three higher than the 4 V I was carefully using. Obviously, the inductor isn't working nearly as well as I expected.

Oops!

My February 2008 column described how to measure transformer core characteristics and that technique works equally well for chokes and inductors ("Transformers," *Circuit Cellar* 211). I set that circuit up again and found that the choke core began saturating at about 8 kG, roughly what you'd expect for ungapped ferrite. Evidently that gap was much smaller than the visible line led me to believe.

The test setup showed that a mere 50 mA through both windings in series pushed the core into saturation, which roughly agrees with Figure 3 and shows that the choke is completely useless for this application.

In retrospect, none of this is surprising. In its original application the choke windings must support the expected load current, but equal current in the opposed windings produces no net magnetic flux in the core. The only common-mode current comes from line noise, but most applications shouldn't see that much CM signal in the first place. Finally, a ferrite core provides useful inductance at much higher frequencies than laminated silicon steel and doesn't need an explicit gap for this application.

As an interesting sidelight, the MAX4372T high-side current amplifier has a –3 dB bandwidth of about 275 kHz. That would suffice for the original design with its relatively slow ramps, but falls behind the core flux collapse when Q1 switches off, as shown by that perfect straight-line declining at about 12 kA/s from the peak.

Although the booster circuit did work, sort of, it's certainly not operating according to the original design.

CONTACT RELEASE

You can generally recover from one blunder and a few minor goofs on a prototype circuit layout, but two big ones means it's time for another revision. With better power supplies, a different layout, and a beefier inductor, I'll be able to talk about what works, rather than what didn't.

All that sounds remarkably like a plan, though. We shall see!

Ed Nisley is an EE and author in Poughkeepsie, NY. Contact him at ed.nisley@ieee.org with "Circuit Cellar" in the subject to avoid spam filters.

PROJECT FILES

To download the additional files, go to ftp://ftp.circuitcellar.com /pub/Circuit_Cellar/2009/225.

RESOURCES

Arduino project, http://arduino.cc.

E. Nisley, "Transformers," *Circuit Cellar* 211, 2008.

SOURCES

Gnuplot Plotting utility
Gnuplot | www.gnuplot.info

LTspice IV Design simulator
Linear Technology Corp. | www.linear.com/designtools/software /ltspice.jsp

Arduino Diecimila Microcontroller
SparkFun Electronics | www.sparkfun.com/commerce/product _ info.php?products_id=666

by Guido Ottaviani

Robot Navigation and Control (Part 2)

Software Development

Guido built a navigation and control subsystem for an autonomous differential steering explorer robot. Here he describes the software development phase of the project.

In the first part of this article series, I described how to build a robotic platform with Microchip Technology dsPIC controllers. Now I will describe the software loaded on the board that manages wheel speed, closed-loop control with a PID algorithm, dead-reckoning by odometry (in both theoretical and practical forms), field mapping, navigation, motor control (MC), and more. The software is modular, so all the pieces can be examined as stand-alone black boxes. I'll focus on the Microchip dsPIC30F board so you can better understand every block. You'll find the detailed comments in the code to be extremely helpful.

FIRMWARE

The philosophies of MC and supervisor programs are similar. Both involve the recycling of numerous portions of the code. The programs are described step by step in the code. The name of the MC's DSC program is dsPID. The program in the supervisor is dsODO.

The source code, MPLAB project, and detailed flowcharts are posted on the *Circuit Cellar* FTP site. Both programs (dsPID and dsODO) are fully interrupt-driven. At start-up, after the initialization of the supervisor and MCs, the programs enter a simple main loop, acting as a state machine. In the main loop, the program checks flags enabled by external events and enters in the relative state (see Figure 1). Because it's a kind of simple cooperative real-time operative system (RTOS), each routine has to be executed in the shortest possible amount of time to free up the system to handle frequent tasks. There are no delays in the code. Interrupts are used whenever possible, particularly for slow operations like the transmission or reception of strings of characters.

MCs use the C30's PID library to control the speed and position of the wheels. The feedback from the encoders on the motors' axes

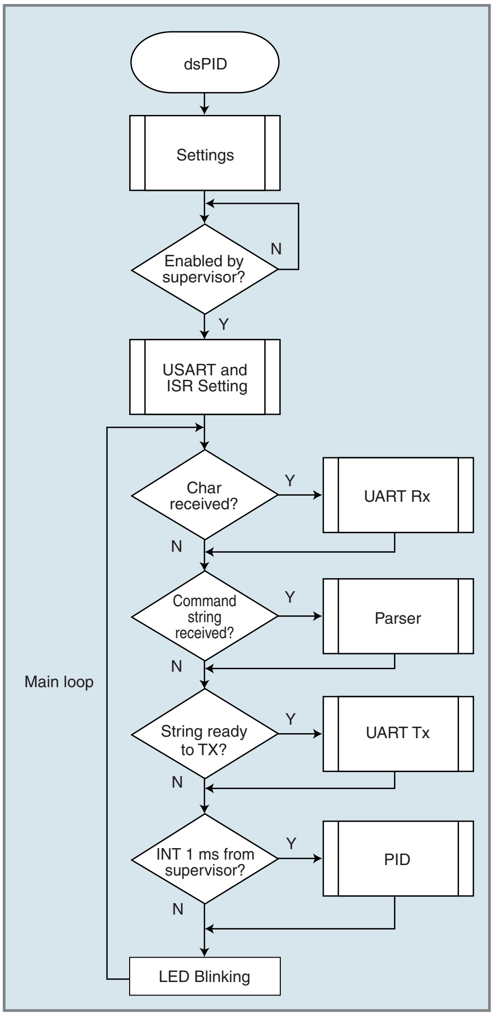


Figure 1—A dsPID main loop is simple because most of the program is interrupt-driven.

Usage	Pin name	Pin number		Pin name	Usage
	MCLR	1	15	INT1	TX Enable
ADC Reference	VREF+	2	16	INT0	Timer 1 ms from supervisor
Motor current reading	AN1	3	17	PGD/EMUD	
		4	18	PGC/EMUC	
Chip select from supervisor	CN5	5	19	VSS	
Quadrature encoder	QEA	6	20	VDD	
Quadrature encoder	QEB	7	21		
	VSS	8	22	RE4	LED 2
	OSC1	9	23	RE3	LED 1
		10	24	RE2	H-bridge Enable
Serial TX	U1ATX	11	25	PWM1H	PWM
Serial RX	U1ARX	12	26	PWM1L	PWM
	VDD	13	27	AVSS	
Velocity measurement	IC2	14	28	AVDD	

Table 1—These are the pins used on the Microchip Technology dsPIC30F4012.

enables this (see Table 1). Peripherals on the MCs include QEI to calculate the covered space, input capture (IC2) to calculate speed, an ADC to read motor current, enhanced PWM to drive the motors, and a UART to communicate with the supervisor.

dsPID

The same program (dsPID) is loaded in both of the MCs, and the supervisor assigns them a different ID at initialization (to address each one later). Speed and position measurements are executed simultaneously by both MCs when an external interrupt occurs from the general timing signal provided by the supervisor.

A QEI module determines the wheels' distance and direction. This value is algebraically cumulated in a variable every 1 ms and sent to the supervisor at its request. After the value is sent, the variable is reset.

Speed is measured at every encoder's pulse. Every 1 ms, it calculates the mean speed by averaging samples, executes a PID algorithm, and corrects the motor speed according to its result, changing the PWM duty cycle (see Photo 1). For a detailed description of the C30 PID library application, refer to the following Microchip code example: "CE019: Proportional Integral Derivative (PID) Controllers & Closed-Loop Control."[1] A link is provided in the References section at the end of this article.

Speed variations of the motors are executed smoothly, accelerating or decelerating with a rising or falling slanted ramp to avoid heavy mechanical strain and wheel slippage that could cause errors in odometry. Deceleration is faster than acceleration to avoid bumps with obstacles during braking (see Photo 2).

IC2, input capture, is used to measure the time elapsed between two pulses generated by the encoder (i.e., when the wheel moves a fixed distance). Connected in parallel to QEA, it captures elapsed time on the rising edge of the encoder's signal. TIMER2 is used in free-running mode. At each IC2 interrupt, TMR2's current value is stored and its previous value is subtracted from it. This is the pulse period. The current value then becomes the previous value, awaiting the next interrupt. TMR2's flag must be checked to determine if an overflow occurred in the 16-bit register. If one occurred, the difference between 0xFFFF and the previous sample has to be added to the current value. Samples are algebraically added in the `IcPeriod` variable. The `_UPDN` bit of the QEI register is set or reset if the wheel is rotating forward or backward. The value of each sample is algebraically cumulated, so it's added if the bit is set, or subtracted if reset, to measure the actual space covered. This is one of the suggested methods in Microchip's application note AN545.[2]

The ADC continuously measures motor current, storing values in its 16-position ADCBUF buffer. When the buffer is full, an interrupt occurs and a mean value is calculated. This happens approximately every 1 ms.

The UART receives commands from the supervisor and sends it the results of the measurements. The communication portion of the program runs as a state machine. Status variables are used to execute actions in sequence. Simple and fast interrupt service routines (ISRs) get or put every single byte from or to a buffer and set the right flags to let the proper function be executed.

TX I/O is disabled at initialization. If an I/O pin is set as an input pin, it enters into a "three-state" mode, meaning a high-impedance mode, which enables you to use parallel pins. This is the default configuration. This setup enables you to connect both MCs' TX ports together. They will be enabled one at a time by the supervisor with INT1.

The same program is in both MCs. Each MC is identified by an ID code to enable the supervisor to send commands to the proper motor. At start-up, the program loops before the "main" idle loop, waiting for a supervisor's enable signal through CN5 I/O port. After that, the correct ID is assigned. The start-up ID is 9 for both MCs.

Photo 1—This test set verifies H-bridge and PID parameters. The motor under test is mechanically joined with a similar motor. This one is loaded on a power variable resistor to easily simulate a variation in mechanical load for the first motor.

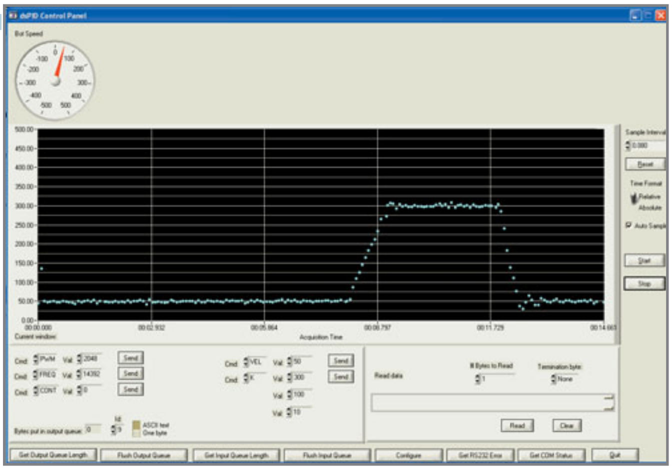


Photo 2—This is one of the first tests during the calibration of PID parameters. It shows the measured speed after a remote request to switch from 50 to 300 cm/s and back to 50 cm/s. Note the rising ramp with less slope than the falling one.

The supervisor will assign the definitive ID, subsequently enabling each MC. In normal operation, both MCs simultaneously receive data sent by the supervisor, but only the addressed one (with the correct ID) decodes the message. A message with ID = 0 (broadcast) is decoded by both MCs. If an error occurs during reception (i.e., UART, checksum, parsing errors), the status variable is set to a negative number and the red LED illuminates to indicate the fault condition.

The supervisor drives both MCs through the UART1 communication port, sending commands and reading information (space, speed, and motor current). It estimates the robot's position using that information (dead-reckoning by odometry) and creates a map of the path, obstacles, and so on. This is done with the help of the dsPIC30F's trigonometric capabilities programmed with the C30 compiler.

The peripherals used on the supervisor include UART1 to communicate with the MCs, UART2 for telemetry with the remote PC, I²C to communicate with the main board, and OC simple PWM to generate the clock for both MCs (see Table 2).

dsODO

The peripherals UART1 through UART2 are used to communicate with the MCs and for telemetry with a remote PC, respectively. They are used the same way as the MCs: similar ISRs, similar functions. The protocol used for the handshake is also the same. The physical-layer-independent protocol is used with the I²C bus, as well as to communicate with the main board.

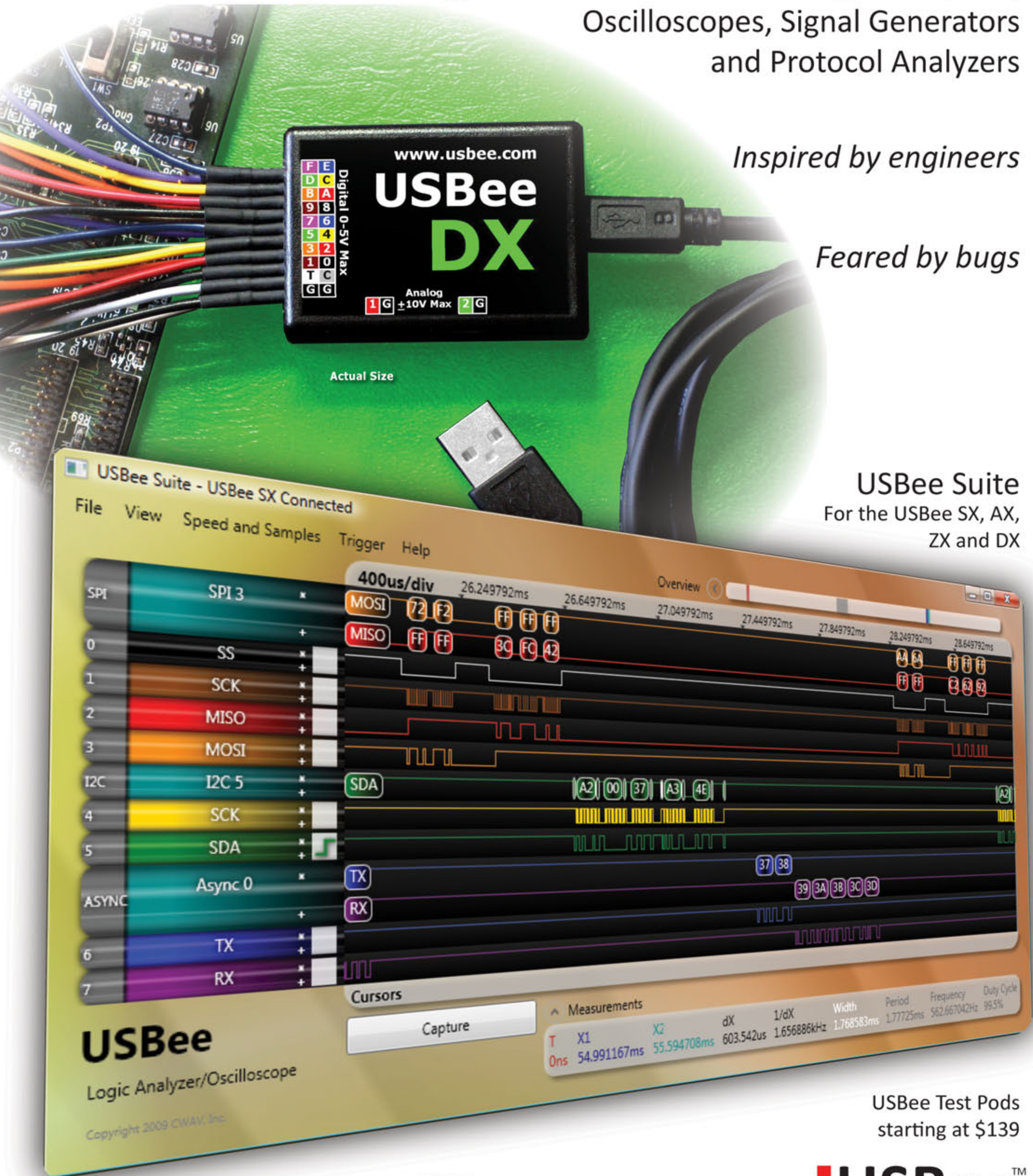
The dsPIC peripheral interface controls the first layer. Frame, or overrun errors (UART), or collisions (I²C) are detected by hardware, setting the appropriate flag. ISR routines handle the second layer. They fill the RX buffer with the bytes received from the interfaces. They also detect buffer overflow and command overrun. UartRx or UartRx2 functions manage the third layer. These routines act as a

USBee Test Pods

Programmable Multifunction Logic Analyzers,
Oscilloscopes, Signal Generators
and Protocol Analyzers

Inspired by engineers

Feared by bugs



USBee Suite
For the USBee SX, AX, ZX and DX

USBee Test Pods starting at $139

www.USBee.com
Powerful Debugging - Small and Portable

USBee™
USB based Electrical Engineer

Usage	Pin name	Pin number		Pin name	Usage
	MCLR	1	15	EMUC2	
		2	16		
Generic chip select 1	RB1	3	17	PGD/SCL	PGD also I²C clk
TX enable 1	RB2	4	18	PGC/SDA	PGC also I²C dat
Generic chip select 2	RB3	5	19	VSS	
TX enable 2	RB4	6	20	VDD	
1-ms Heartbeat	RB5	7	21	U2TX	Serial 2 TX
	VSS	8	22	U2RX	Serial 2 RX
	OSC1	9	23	RB9	LED 1
	OSC2	10	24	OC1	Clock out for Motor controllers
Serial 1 TX	U1ATX	11	25	EMUD2	EMUD2
Serial 1 RX	U1ARX	12	26	RB6	LED 2
	VDD	13	27	AVSS	
		14	28	AVDD	

Table 2—These are the pins used on the Microchip dsPIC30F3013.

state machine, getting bytes from the buffer and decoding the command string (see Table 3).

This layer controls timeout and checksum errors, as well as packet consistency (correct header, correct length). If everything is fine, it allows the Parser routine (fourth layer) to decode the message and to execute the required action. This routine sets the appropriate error flag if the message code received is unknown.

TMR1 generates a 1,000-Hz timing clock (the program's heartbeat). On each TMR1's interrupt, internal timers are updated, the watchdog is cleared, and a flag is set to enable the function that requests the MC's distance. Every 10 ms, an "All_Parameters_Ask" function (speed, position, and current) is enabled. The same clock is used, through a pulse on RB5, to synchronize MCs for PID and position elaboration.

PWM (output Compare 1) is used to obtain the oscillator frequency for the MCs. The OC simple PWM I/O peripheral is set to have a PWM at 50% duty cycle with a 7.3728-MHz frequency (the same as the supervisor crystal):

```
PWM_period=(PRx+1)•4•TOSC•(TMRx_prescale_value)
```

With `Prx` = 3, `prescale` = 1 7.3728 MHz is obtained again at output. With this output, both MCs can be driven in EC 16xPLL mode. This way, all three DSCs have exactly the same clock and some components are saved on the board.

With data coming from the MCs, the supervisor performs field-mapping. For more information about the topic of dead-reckoning by odometry, refer to the following works: "Where Am I?: Sensors And Methods For Mobile Robot Positioning," by Johann Borenstein[3]; "Implementing Dead Reckoning by Odometry on a Robot with R/C Servo Differential Drive," by Dafydd Walters[4]; and "A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators," by G. W. Lucas.[5] Simplified algorithms are also in these

Sequence	Name	Range	Note
1	Header	@	
2	ID	0–9	ASCII
3	Cmd	A–Z	ASCII
4	CmdLen	1-MAX_RX_BUFF	Number of bytes following (checksum included)
5	Data	...	
..		...	
n	Checksum	0–255	Obtained by simply adding up in an 8-bit variable, all bytes composing the message (checksum itself excluded).

Table 3—This is the structure for the command packets. Each one contains all the bytes shown.

documents. You can find the correct compromise between precision and computing speed by using the trigonometric capability of the dsPIC30F series.

Every few milliseconds, after the current position elaboration,

Listing 1—To create a 50 × 50 nibble matrix, you need to define a struct.

```
typedef struct
{
        unsigned char Low :4;
        unsigned char High :4;
}_Coordinate;

_Coordinate MapXY [25][50];
```

field mapping divides the unknown field in a 10 cm × 10 cm cell grid. Defining a maximum field dimension of 5 m × 5 m, you obtain a 2,500-cell matrix (50 × 50). Each cell is

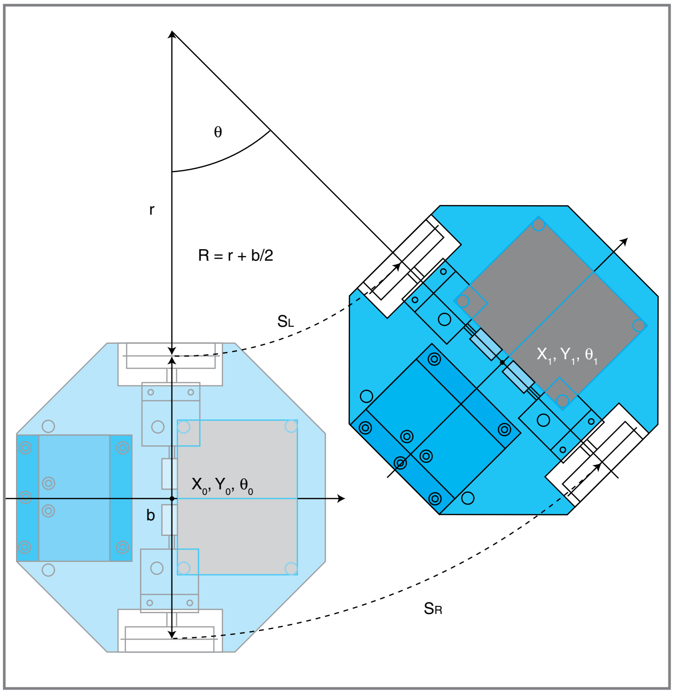


Figure 2—This is a definition of the terms used in the formulas for a turning platform.

defined with a nibble, with a total memory occupation of 1,250 bytes. Sixteen different values can be assigned to each cell (e.g., n = 00 unknown cell, n = 01 – 10 cell visited n times, n = 11 obstacle found, n = 12 target of type A found, n = 13 target of type B found, and n = 14 target of type C found).

The robot can start from any position in the field. Note that (0, 0) is the reference coordinate in its reference system. To translate robot reference system coordinates to a 50 × 50 matrix index pair, the values must be "normalized" in a 0 to 49 range: Xnorm = (Xrel + 50) mod 50 and Ynorm = (Yrel + 50) mod 50. Index is the remainder of division in a range of 0 to 49. A range check must be performed in advance to avoid overflow if the field is greater than 5 m × 5 m.

To create a 50 × 50 nibble matrix, you need to define a `struct` (see Listing 1). It fills up 1,250 bytes. Eliminating heap space (not needed if dynamic memory allocation or file I/O library functions are not used) leaves enough RAM to work with.

Field-mapping is useful for finding the best exploring strategy in an unknown field. The robot can direct itself to the less explored portion of the field (lower "n" value); it can save time by not stopping twice in an already discovered target; and it can find the best path to reach a given coordinate, and more.

DEAD RECKONING BY ODOMETRY

Let's consider the general dead-reckoning algorithm needed for a DSC- or microcontroller-based system. Once you have the information about the distance traveled by each wheel in a discrete time update (odometry), you can estimate the robot's position coordinates with the same periodicity without any external reference (dead reckoning). Refer to G.W. Lucas's aforementioned paper for information about the mathematics.[5] In the following equations, I used Lucas's symbols and terms:

$$\vartheta(t) = \frac{(v_R - v_L)t}{b} + \vartheta_0$$

$$x(t) = x_0 + \frac{b(v_R + v_L)}{2(v_R - v_L)}\left[\sin\left(\frac{(v_R - v_L)t}{b} + \vartheta_0\right) - \sin(\vartheta_0)\right]$$

$$y(t) = y_0 - \frac{b(v_R + v_L)}{2(v_R - v_L)}\left[\cos\left(\frac{(v_R - v_L)t}{b} + \vartheta_0\right) - \cos(\vartheta_0)\right]$$

Figure 2 shows the terms used in the formulas for a turning platform.

For each discrete time interval, the system measures the number of pulses generated by the encoders. Knowing the distance represented by a single encoder tick, you can calculate the distance traveled by the wheels (S_R, S_L) in time t. Note that velocity = distance/time:

$$v_R = \frac{S_R}{t}$$

$$v_L = \frac{S_L}{t}$$

According to Lucas:

$$R = \frac{b(\nu_R + \nu_L)}{2(\nu_R - \nu_L)} = \frac{b(S_R + S_L)}{2(S_R - S_L)}$$

You can calculate:

$$x(t) = x_0 + R\left[\sin\left(\frac{(S_R - S_L)}{b} + \vartheta_0\right) - \sin(\vartheta_0)\right]$$

$$y(t) = y_0 - R\left[\cos\left(\frac{(S_R - S_L)}{b} + \vartheta_0\right) - \cos(\vartheta_0)\right]$$

Note that at time t_i, the differences with the coordinates at t_{i-1} are:

$$\Delta\vartheta = \frac{(S_R - S_L)}{b}$$

$$\Delta x = R\left[\sin(\Delta\vartheta + \vartheta_{i-1}) - \sin(\vartheta_{i-1})\right]$$

$$\Delta y = R\left[\cos(\vartheta_{i-1}) - \cos(\Delta\vartheta + \vartheta_{i-1})\right]$$

By performing a summation of each delta x in x variable and each delta y in y variable, you know the current coordinates (position and orientation) of the platform.

To avoid computational errors (divide by zero) and wasted

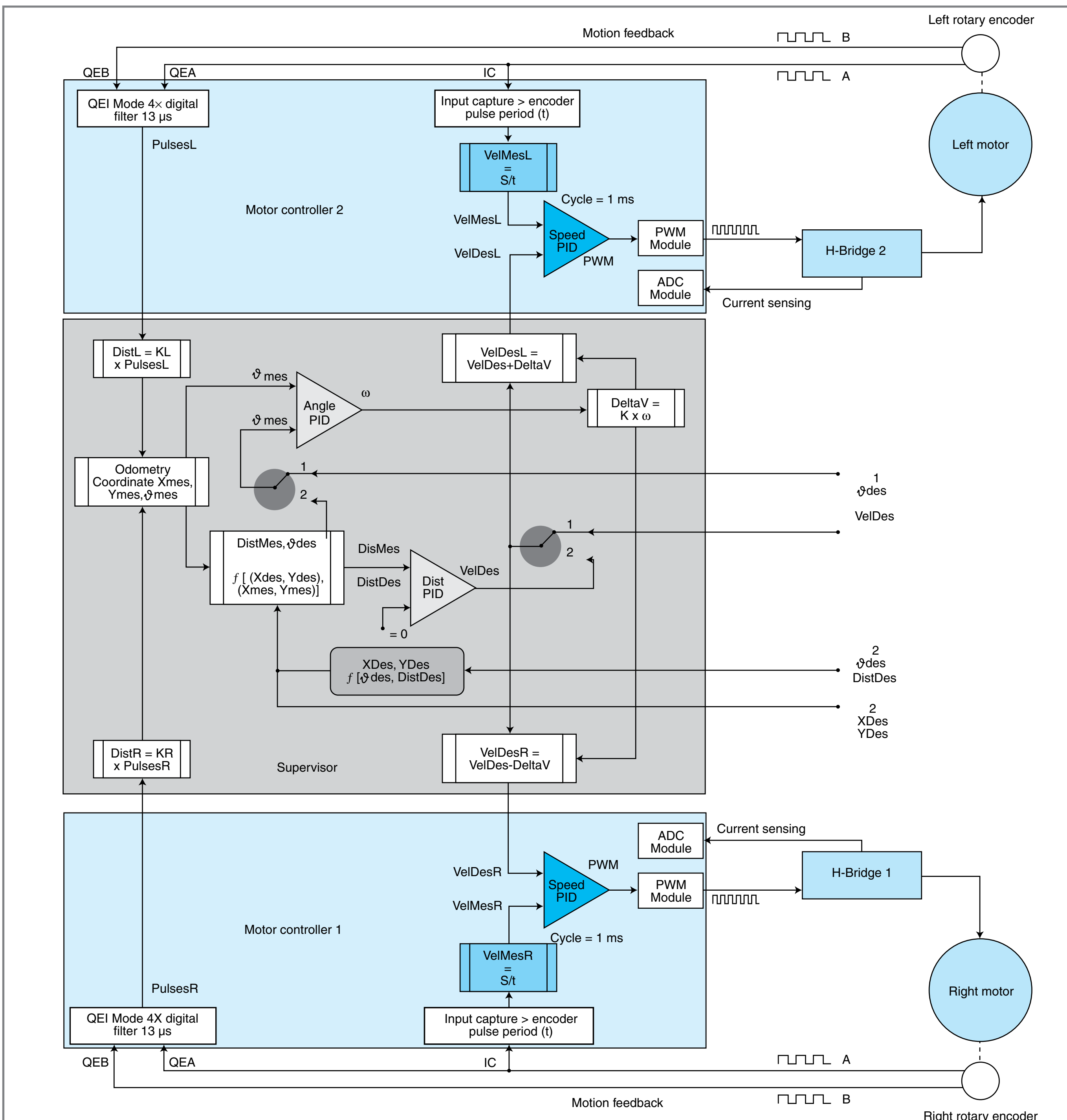


Figure 3—These software logical blocks govern the robot's navigation.

controller time, both the S_R and S_L variables must be checked in advance. Defining a quasi-zero value S_{min}, which takes care of minimal mechanical and computational approximations, you get the following. If $|S_R - S_L| < S_{min}$, the platform is traveling in a nearly straight line and you can use the method from Lucas's paper without approximations:[5]

$$\Delta S = \frac{(S_R + S_L)}{2} \cong S_R \cong S_L$$
$$\Delta \vartheta = \frac{(S_R - S_L)}{b} = 0$$
$$\Delta x = \Delta S \cos(\vartheta_{i-1})$$
$$\Delta y = \Delta S \sin(\vartheta_{i-1})$$

If $|S_R + S_L| < S_{min}$, the platform is pivoting around its own vertical axis without moving. Thus:

$$\Delta \vartheta = \frac{(S_R - S_L)}{b}$$
$$\Delta x = \Delta y = 0$$

SOFTWARE ARCHITECTURE

Figure 3 shows the overall software architecture for the dsNavCon board's control procedures and navigation strategies. The most important logical blocks are the four PID controls. They are shown in a three-level nested control loop. Starting from the top level, the Distance PID controls the robot's mean speed toward the target every 50 ms. The Angle PID corrects the orientation to point the target every 10 ms by adding or subtracting a DeltaV to the mean speed to make the vehicle spin around its vertical axis. By combining the outputs of the Angle and Distance PIDs, you can determine the setpoint for the most internal level, the Speed PIDs. Each of the PIDs controls the speed of its wheel every 1 ms to maintain the value set by outer loops (see Figure 3). By combining the output of Angle and Distance PIDs, you can obtain the setpoint for the Speed PIDs (see Figure 3). The three levels are nested. But, fortunately, the different PIDs (speed, orientation, and distance) are independent of each other, simplifying the K parameter's calibration procedure. They can be set one at a time starting from the bottom.

The motor controllers appear as dark boxes that take care of the wheels' speed. The supervisor sends them the reference speed (VelDesX: desired velocity) and the input capture modules of the microcontrollers get pulses from the encoders connected to the motor axis and derive the rotational speed of the motors (VelMesX: measured velocity). By combining the values in the PID control Speed PID every 1 ms, you can obtain the necessary PWM value in that condition to keep the desired speed of each wheel. In PID terminology, VelDesX is usually called the setpoint or control reference. VelMesX is the measured output or process variable. PWM is the control output, manipulated variable, or simply output.[6]

The Quadrature Encoder Interface (QEI) modules get both the A and B pulses from the encoders. They receive the traveling direction and the number of pulses in 4× mode (counting the rising and falling edges of signal A and signal B: 2 × 2 = 4) to the supervisor.

Multiplying the number of pulses by K—which indicates the distance traveled for each encoder pulse—you can determine the distances traveled by right and left wheels every 10 ms. The supervisor combines this traveling information and applies the dead-reckoning procedure to determine the robot's position coordinates: Xmes, Ymes, and θMes (orientation angle).

The supervisor receives an external navigation command via the serial interface (telemetry) or via the I²C interface (main board). Different strategies can be applied: A—Free running is movement at a given speed in a given direction (VelDes, θDes). B—Cartesian is movement toward a given coordinate (XDes, YDes). C—Polar is movement for a given distance in a set direction (DistDes, θDes).

In mode A with the logical control switches in position 1, only the PID control (Angle PID) is used on the supervisor (see Figure 3). This combines the desired angle θDes with the measured angle θMes computed by the odometry procedure—to obtain the value of the rotation angular speed ω of the vehicle around its vertical axis—needed to correct the orientation error.

The DeltaV value is proportional to ω. It's added to VelDes to obtain the left wheel's speed and subtracted from

VelDes—to obtain the right wheel's speed—in order to keep the heading corresponding to the θDes value, while the center of the robot is still moving at the VelDes speed.

In mode B, with the logical control switches in position 2, the desired speed VelDes is calculated by the PID control Dist PID, and it is used as in mode A. This means the mean speed decreases proportionally to the distance from the target. It becomes zero when the target is reached. The measured input for this PID (DistMes) is computed as a function of the current coordinates and the destination coordinates. The desired orientation angle θDes also comes from the same procedure and it's used as reference input for Angle PID. The reference input for Dist PID is 0, meaning that the destination is reached. With ω and VelDes available, the wheels' speed control runs as it does in the first mode.

In mode C, with the logical control switches in position 2, the destination coordinates (Xdes, Ydes) are computed once at the beginning as a function of input parameters (DistDes, θDes). After that, everything operates as it does in mode B.

A sequencer is also available to perform some specific paths for UMBmark (or something like the RTC competition I mentioned in the first part of this article series).[7] Like a washing machine timer, it schedules the robot's behavior by executing a series of primitives. The sequence is written in some arrays, and it is synchronized by external events. Some higher-priority events (e.g., obstacles found by external sensors) can override scheduling.

TIME TO GO ROBO

There are plenty of affordable robots on the market. Plus, the MPLAB development environment is free. You can design the schematic and PCB with the freeware version of CadSoft Computer's Eagle. These tools are versatile enough for a wide variety of applications. Affordable electronic and mechanical components are also widely available on the Internet. Do some reseach before shelling out a lot of cash for an expensive kit.

No more excuses. You are now ready to design, build, program, and test your own robot. ■

Guido Ottaviani (guido@guiott.com) has worked with electronics and ham radios for years. After working as an analog and digital developer for an Italian communications company for several years, Guido became a system integrator and then a technical manager for a company that develops and manages graphic, prepress, and press systems and technologies for a large Italian sports newspaper and magazine publisher. A few years ago, he dusted off his scope and soldering iron and started making autonomous robots. Guido is currently an active member in a few Italian robotics groups, where he shares his experiences with other electronics addicts and evangelizes amateur robotics.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/225.

REFERENCES

[1] Microchip Technology, Inc., Microchip code examples, "CE019: Proportional Integral Derivative (PID) Controllers & Closed-Loop Control," 2005, www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2620.

[2] M. Palmer, "AN545: Using the Capture Module," DS00545D, Microchip Technology, Inc., 1997.

[3] J. Borenstein, H. R. Everett, and L. Feng, "Where Am I?: Sensors and Methods for Mobile Robot Positioning," University of Michigan, 1996, www-personal.umich.edu/~johannb/position.htm.

[4] D. Walters, "Implementing Dead Reckoning by Odometry on a Robot with R/C Servo Differential Drive," *Encoder*, 2000, www.seattlerobotics.org/encoder/200010/dead_reckoning_article.html.

[5] G. W. Lucas, "A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators," SourceForge, 2001, http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html.

[6] Wikipedia, "PID Controller," http://en.wikipedia.org/wiki/PID_controller.

[7] J. Borenstein and L. Feng, "UMBmark: A Method for Measuring, Comparing, and Correcting Odometry Errors in Mobile Robots," 1994, www-personal.umich.edu/~johannb/umbmark.htm.

RESOURCES

G. Ottaviani, www.guiott.com/Rino/index.html.

Roboteck Discussion Group, http://it.groups.yahoo.com/group/roboteck/ (Italian) or http://groups.yahoo.com/group/roboteck_int/ (English)

Robot Italy, www.robot-italy.com

SOURCES

Eagle Software
CadSoft Computer, Inc. | www.cadsoftusa.com

dsPIC30F3013 Digital signal controller, dsPIC30F4012 motor controller, and dsPIC33FJ64MC802 microcontroller
Microchip Technology, Inc. | www.microchip.com


Let your geek shine.
Meet Michael Cooper, adrenaline junkie and SparkFun enthusiast. With SparkFun's WiTilt, and a bit of ingenuity, Michael was able to hack the product's firmware and measure the force his body endures during a BASE jump.
Whether your goal is to record the data from a 400-foot free-fall, or simply to make an LED blink, SparkFun products and services are here to help. Take the leap and let your geek shine too.
sparkfun ELECTRONICS
Sharing Ingenuity
WWW.SPARKFUN.COM
©2009 SparkFun Electronics, Inc. All rights reserved.
Be safe out there, Michael!

by Danilo Consonni

Digital Decoding

Decode Periodic Signal Transmissions

Danilo's digital decoder decodes the SRC-RAI signal, the Italian official hourly signal. With this device, you can learn how one of the oldest operating signals of sample time is made and how it works.

Men have been interested in the topic of time and how to measure it for centuries. Famous philosophers and scientists such as Albert Einstein have studied every aspect of this peculiar physical concept. Through his theory of relativity, Einstein introduced essential concepts about the regular flow of time. These matters continue to puzzle us today. For instance, it is impossible to synchronize two watches that are not in the same "condition."

Think about the GPS satellites orbiting the Earth at an approximate altitude of 20,000 km. Believe it or not, between the two atomic clocks, one on the satellite and one at sea level, there is a daily difference of about 38.9 µs. It's a difference we cannot ignore that's due to different gravitational pulls acting on the orbiting clock and its displacement velocity, which is 12 times greater than the one on Earth. Therefore, GPS satellite systems are usually adjusted before launch in an effort to minimize such effects.

When it comes to time, a temporal scale of reference is important. The old definition of a "second"—as a division of the 86,400 parts of the mean standard day—is inadequate. In 1967, the scientific community acknowledged the new definition of "second." Back in 1955, English physicians Louis Essen and J.V.L. Parry created the first cesium-beam atomic clock at the National Physical Laboratory in England. In the international system, the "second" then became SI, the duration of 9,192,631,770 periods of the radiation corresponding to the transition between two hyperfine levels of the ground state of the cesium-133 atom.

Photo 1—This is the IEN SCR-RAI decoder.

Accelerate your Design Time with EmbeddedDeveloper.com

Speed up your design time during the critical evaluation phase of your project by avoiding locating and comparing devices from different manufacturers. We make it easy: just one visit to **EmbeddedDeveloper.com** will shave hours or days off your schedule!

Embedded Developer's simple navigation and intuitive search engines help you quickly locate, compare, and evaluate thousands of different microcontrollers from different manufacturers--without knowing part numbers.

Compare all devices by their performance and features in seconds, download a data sheet, find and buy development tools, or link to a distributor for an RFQ or chip sample--without ever leaving the site!

Embedded Developer is the only site in the world where you're only a few clicks away from buying the right device or tool for your next job. So fasten your seat belts and log-in today--we guarantee this site will put you in pole position in the time-to-market race!

HEARST *business media* ELECTRONICS GROUP

EMBEDDEDDEVELOPER.COM
FIND. COMPARE. BUY.

Convergence PROMOTIONS

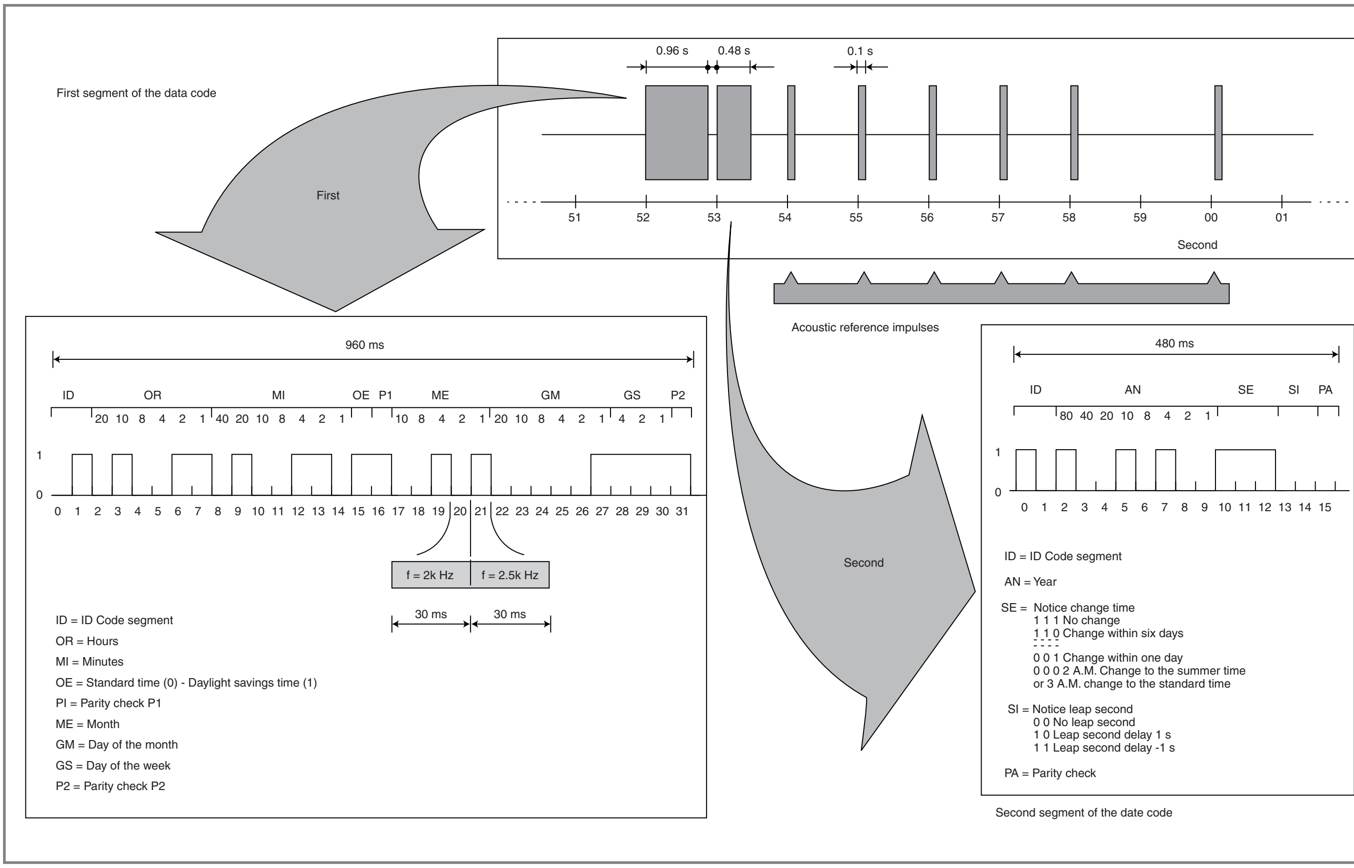


Figure 1—This is the complete IEN SCR-RAI signal.

After that, numerous metrological laboratories adapted to the new technological concept. For instance, since 1970, The National Electrical Institute (IEN) in Turin, Italy, has had its own coordinated universal time (UTC) that is based on cesium-beam atomic clocks. With the introduction of the "cesium fountain" atomic clock in 2003, the Institute, now called The National Institute of Metrological Research (INRIM), has been at the forefront of time measurement. This type of clock has a precision of 10^{-15}, which means a 1-s error in 30 million years, provided that it would work for such a long time.

To coordinate the ever-increasing amount of data produced by the atomic-clock-equipped labs throughout the world, we now refer to the International Bureau of Weights and Measures (BIPM) in Paris, which has calculated the time since 1988 using the UTC times of accredited centers, such as the so-called International Atomic Time (TAI), which has become the scientific community's official point of reference for time measurements.

In this globalized, computerized world, we cannot ignore the importance of having a universal temporal scale of reference. Think about the financial and stock transactions that are quickly transferred through the telecommunication networks and may change their values in only seconds. Time synchronization is important in everyday life as well. If you think I am exaggerating, let's go back for a moment to New Year's Day 2000 when a simple time change was supposed to trigger a worldwide catastrophe. Fortunately, no big problems occurred. But to prevent a potential catastrophe, numerous tests were conducted on devices for thousands of hours.

Today, many systems are automatically synchronized through Network Time Protocol (NTP) servers on the Internet. To supplement its divulgation activities, the INRIM has two NTP free access servers, which are set with the institute's UTC-IEN time. These servers are found at the following addresses: ntp1.ien.it (193.204.114.232) and ntp2.ien.it (193.204.114.233).

THE SRC-RAI SIGNAL (IEN)

In an international community, transmitting the time signal via a radio signal is common. The German DCF77 signal, which works on 77.5 kHz, covers a large part of Europe, including all of England and ranging from part of the Scandinavian countries down to near the North African coasts (2,000 km). The similar Swiss Prangins HBG signal is compatible with DCF77 signal coding. The only difference is that the emissions frequency for the HBG signal is 77 kHz. Like the German signal, it also has a considerable range of 1,500 km. Finally, in England, since 2007, the National Physical Laboratory (NPL) has transmitted the English official signal MSF from the Anthorn station through 17 kW of power working on 60 kHz.

Unlike the aforementioned signals, the SRC-RAI signal is not generated and directly transmitted on air. It is

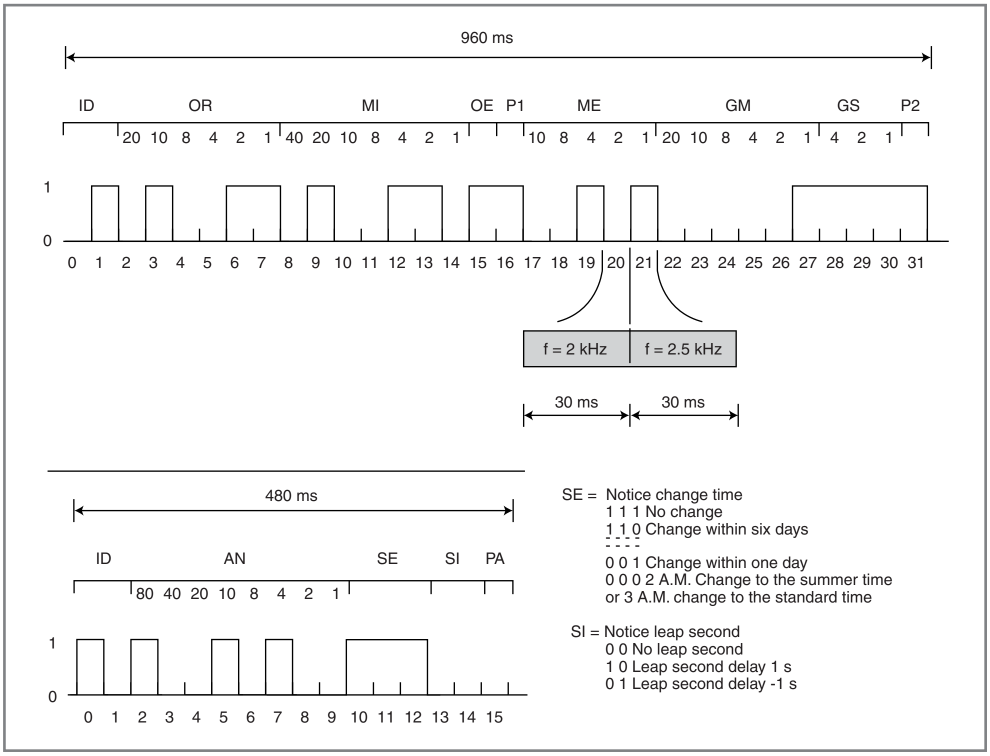


Figure 2—These are details about the two blocks of the IEN SRC-RAI signal.

coded and sent to the Radio Televisione Italiana (RAI) station, which broadcasts through its own radio network.

My IEN-SRC project cannot compete with the progress of the hourly signals or with Internet and the NTP service. But the purpose of this project is to study one of the oldest operating signals of sample time (see Photo 1). Even if the signal's mode of transmission is obsolete, the IEN in Torino is still at the forefront in terms of precision and technological research. Thus, this project is an excellent starting point for studying a signal's time and frequency. If you're in Europe, you can receive the radio RAI signal via satellite. Try Hotbird 8 (13.0°E, 11,804.00 MHz, vertical polarization) or Astra 1L (19.2°E, 11,567.50 MHz, vertical polarization).

Let's examine how the SRC-RAI time signal is made. First, note that it is a modulated frequency shift keying (FSK) signal in an audio frequency band. This means that it is an "audible" signal. As you can see in Figure 1, it consists of two groups of bits that are transmitted on seconds 52 (32 bits) and 53 (16 bits) of each minute. Every bit has a fixed duration of 30 ms. The bit is 2,000-Hz if it is a 0, or a 2,500-Hz tone if it is a 1. A third tone of 1,000 Hz is used for the last six audio synchronization signals.

At the head of the group of bits is the ID field (ID-1 and ID-2), which identifies the bits that follow. Bits 16 and 31 of the 32-bit group (P1 and P2) and bit 15 of the 16-bit group (PA) are parity bits that allow you to verify the preliminary accuracy of the data received through the no-parity method.

The first segment of code transmits the time, the entire date, and the day of the week. The second segment transmits the year and two interesting technical parameters. Data is transmitted through the binary-coded decimal (BCD) coding and not with a binary sequence, as you might think.

As you can see in Figure 2, the bit values are placed according to the positions in the transmitted code. For the remaining technical information (SE – SI), refer to the tables in

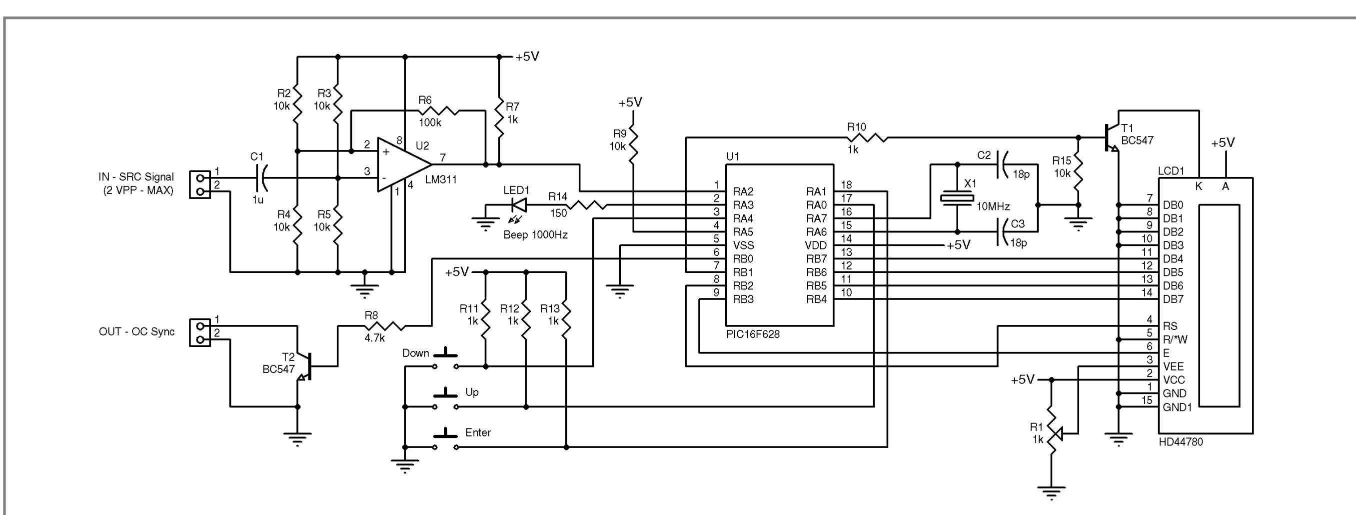


Figure 3—This is the decoder circuit. I used a PIC16F628.

Figure 2. A detailed analysis of the two aforementioned parameters shows that the first (SE) warns when time changes from standard to daylight-savings time and vice versa. The second (SI) sees to the warning for the so-called "leap second." In brief, the leap second represents the power to add or take away 1 s to compensate for variations in the speed of the Earth's rotation. If necessary, this compensation is inserted at the end of June or December after 23:59:59 UTC.

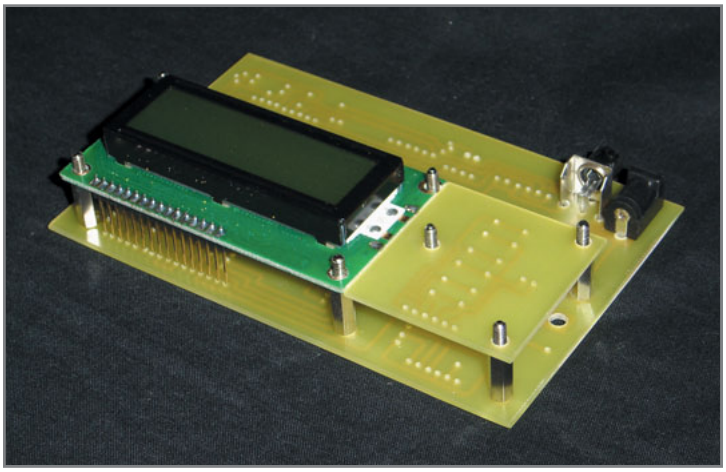


Photo 2—This is the PCB.

From the introduction of the leap second in 1972, up to the last update on January 1, 2009, a total of 34 s have been added to the time scale (TAI). The decision to use (or not use) the leap second is made by the International Earth Rotation Service (www.iers.org). It does so according to the measurements made by the metrological laboratories that it refers to.

To a casual listener, the SRC-RAI hourly signal sounds like a shrill, modulated, 1.5-s sound that's followed immediately by five short synchronizing signals. A final signal is transmitted that corresponds to second 00. As I mentioned, the INRIM's SRC-RAI signal is directly transmitted on a dedicated frequency to the RAI's headquarters and then broadcast in the normal radio-RAI program. This is why it can be received at different times and places on all three RAI stations in both amplitude (AM) and frequency (FM) modulation. The signal's broadcast usually takes place 15 times per day (sometimes 20 or more times) in different time bands. Signals at midday, midnight, and generally before the evening news cannot be eliminated.

If the transmission can't be received directly from Italy, the same radio-RAI signal can be received throughout Europe with a standard digital satellite TV receiver. In that case, you can can obtain the signal via the audio output on the back of the receiver. For more information about the SRC-RAI time signal, refer to the INRIM IEN web site (www.ien.it/tf/time/index.html).

SYSTEM DESIGN

This project was created for educational purposes. I developed the circuit to study the SRC-RAI signal generated by the IEN in Turin (see Figure 3). I don't pretend the circuit is highly accurate or perfectly synchronized. Bear in mind that the signal is transmitted via radio or TV satellite. In addition to being an old technology, it produces a delay in the propagation that cannot always be defined and that is often similar to other systems transmitting the hourly signal (see Photo 2). However, the sources of information contained in the SRC-RAI coded and received signal are exact. Therefore, you can use them as a point of reference.

Let's move on to the electric diagram. It is separated into two different sections. The first part hinges on the analog integrated circuit LM311, a classic single-supply comparator, which, surrounded by a few passive components, starts processing the input analog signal. This entire first phase basically supplies a TTL squaring of the signal to send it directly to the microcontroller, which handles the entire decoding process.

The core is a Microchip Technology's PIC16F628A microcontroller. Compared to the well-known PIC16F84, this microcontroller has 2 KB of program memory and more independent timer registers, which can be used for programming. The PIC16F628A handles numerous tasks: it reads the header, decodes the input TTL signal, manages the display (including the display light), and drives the open-collector synchronization output.

Many previous FSK signal decoder designs used a selective circuit as an A/D interface—that is, an integrated circuit (e.g., LM567) based on a PLL sample frequency, which literally "captured" the allocated frequency. Once the sample frequency was found in the signal, these devices simply produced a TTL output signal. There are a few obvious disadvantages associated with these circuits: the need to have different selective stages (one for every frequency to be found), the need to use laboratory instruments (oscilloscope or frequency counters) for the calibration phase, and a proven instability due to the noises often present in the input signal. Noises in the AM frequency (vanishing noises) and the less invasive FM frequency (over modulation) are the major causes for unsuccessful decoding with the old decoder circuits.

My project features a digital circuit where the task of selecting the FSK tone, as well as its quality, is delegated to the only microcontroller. In terms of functionality, you have notable advantages: calibration isn't required, you have stable selectivity, and you have an inexpensive, compact project.

The PIC16F628A microcontroller's software was developed with pure assembly code. High-level languages weren't used. This refined approach enabled me to use the 2 KB of memory and maximize the part's full potential. I used an alphanumeric integrated display that contains a Hitachi HD44780 with a reduced pin (nibble)

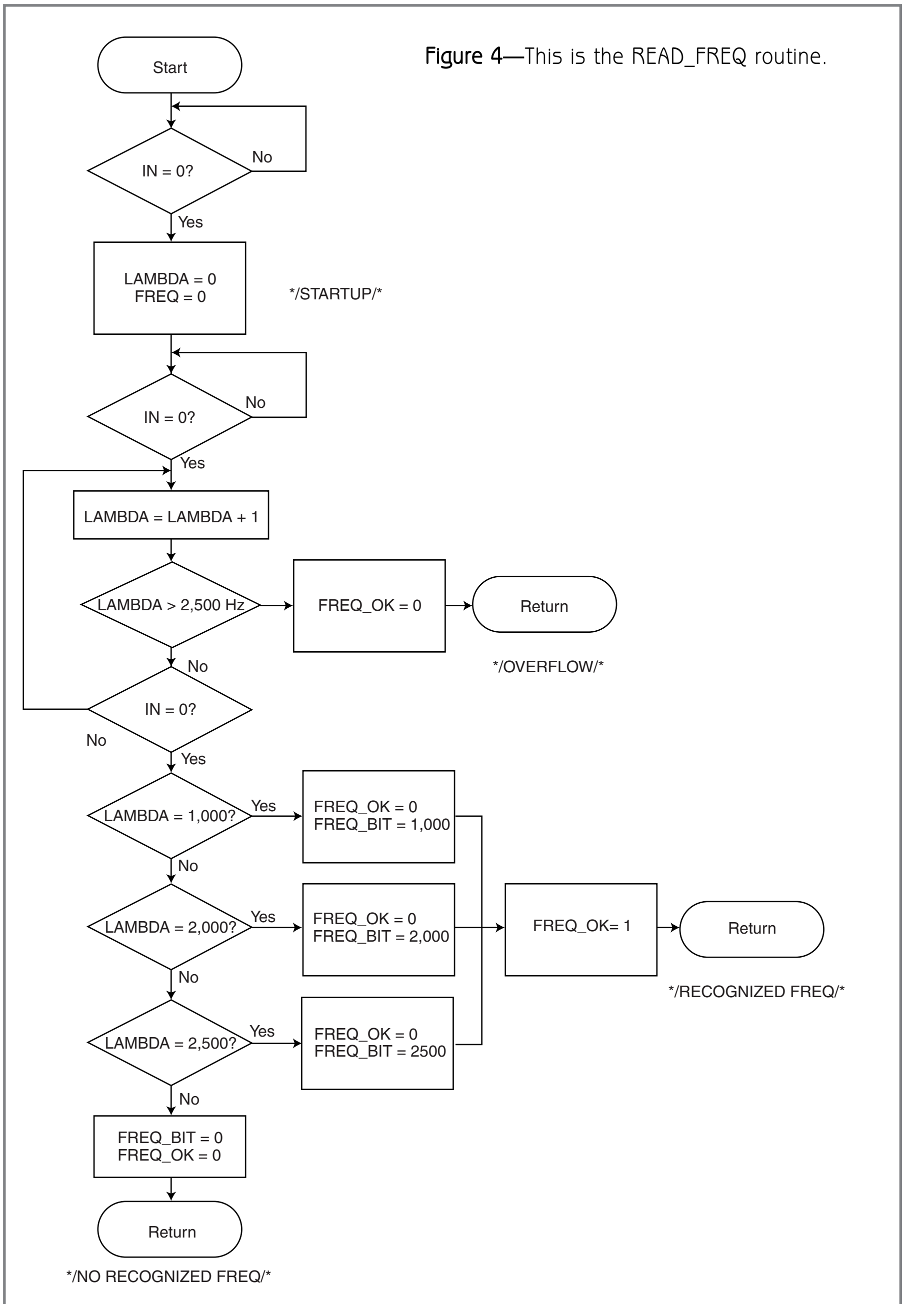


Figure 4—This is the READ_FREQ routine.

configuration. The microcontroller manages the display (and backlighting). The PIC16F628's software is posted on the *Circuit Cellar* FTP site.

I must point out one thing about the only output signal connected to the open collector phase managing the TR2, which is responsible for transmitting a 3-s pulse dead on second 00 of every hour. This signal pulse may occasionally help when connecting an external device to be synchronized with the aid of your SRC decoder.

Finally, note that the circuit includes a voltage regulator. A tested LM7805 voltage regulator supplies an adequate level of stabilization.

FIRMWARE & DECODING

I consider the software to be the most innovative part of this project. As you can see in Figure 4 and Figure 5, the decoding section of the three sample frequencies (1,000, 2,000, and 2,500 Hz) represents the most interesting portion of the firmware. The READ_FREQ routine is responsible for this activity. It can synchronize on the leading edge of the input signal, establishing its period (lambda). If the lambda value is included in an expected frequency range (1,000, 2,000, or 2,500 Hz), the routine exits and advises which incoming frequency has been recognized. The range of lambda values that define the frequencies to be recognized has been given a 3% margin to avoid errors in the recognition mainly due to a frequency slippage, which is typical with AM receiving.

In the same routine, several checks are made to verify the correct lambda reading, as well as to reject frequencies that are too high (caused by spikes) or frequencies that are too low (no incoming signal). The first routine using READ_FREQ is the one that reads the principal header, made up of one bit 0 followed by one bit 1. The correct reading of this part of the code is fundamental for all reading all of the data. The routine takes into consideration the exact length of the signals and then estimates the quality through a noise parameter rather than reading errors.

In fact, if the decoding activity is stopped as soon as a "simple" defection in reading occurs, it would become impossible. Bearing in mind the nature of the incoming signal source, you cannot expect to have a signal that is totally accurate.

The noise value then becomes an important parameter to value the signal's quality. Higher parameters mean more signal interferene. Beyond a certain cluster point, an error in reading occurs and an ID1-ERROR message appears. The wide margin-based reading technique and the quality of the read data are also made possible by the fact that there are only three frequencies to be read and their spacing. When designing the signal standard, it is likely that the IEN had already considered the most adequate frequencies to get a data transmission with the lowest number of potential errors.

Immediately after reading the main header HD-1, the following bits are decoded, which constitute the first segment in a quick succession, one after the other. The generic READ BIT routine carries out this activity depending on the preceding READ FREQ for the frequency identification.

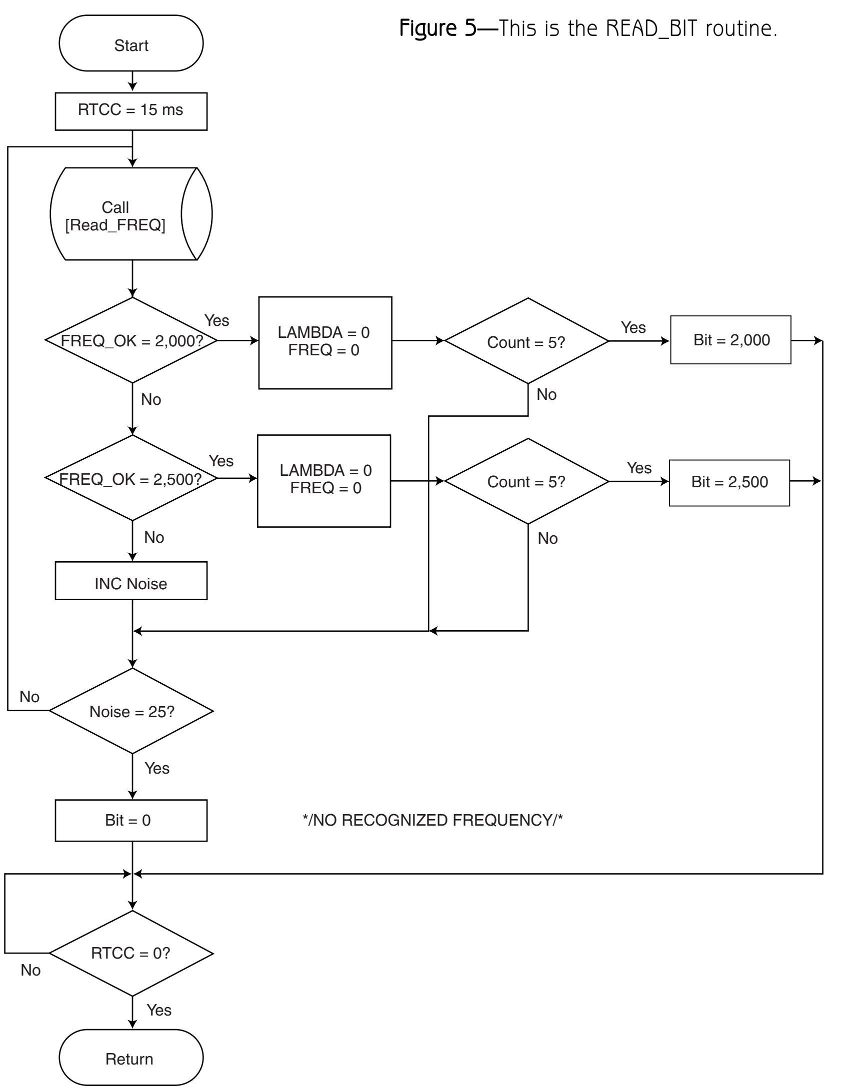


Figure 5—This is the READ_BIT routine.

All of the data, which are read in the first segment, will be stored and considered all at once after reading the second segment. When the 32 bits constituting the segment are available, there is a final check, which consists of testing the control of the odd parity residing in the code every 16 transmitted bits. It is useful for auditing the received data.

After the last check on the first segment, the second segment is read with the same procedures and software routine. The second segment is made up of 16 bits, which is half of the previous one. The use of specific routines and the technique based on orderly and recursive calls represents a typical example of a productive way of programming. The remaining software routines don't show any particularities apart from the calendar, which autonomously looks after the correct display of the time and the punctual date change for leap years included between 2000 and 2035. After 2035, this device will probably be considered only as a piece of history. However, an undoubtedly interesting feature is the display of the second intercalary "leap second," which will be able to display the virtual 60 s as soon as the leap second happens. Its display will happen at 1 A.M., not at midnight, because the Italian time zone is 1 GMT.

USER MANUAL

By now you should be persuaded that using the decoder is easy to use and highly reliable. You just need to decide if you want to use a common AM-FM radio or a satellite TV receiver. The former is easier and readily available, but imperfect frequency tuning may blanket the reception. Don't worry. A lot of work has been made on the software in the microcontroller in an effort to minimize mistakes as much as possible. A special routine calculates and judges the signal's quality to identify data that is also in the presence of reception noises. On the other hand, a TV satellite decoder offers excellent output signal quality but limits portability.

When switching on the device, a welcome message appears on the display. You're then asked to choose the source of the signal: <RADIO> or <SAT>. By selecting the satellite, you tell the device that the source of the signal will be taken from a satellite receiver with a calculated average delay of about 240 ms, which will be automatically adjusted by the software in the microcontroller. This enables perfect synchronization with absolutely no problem.

Once you connect the source of the input signal and feed the circuit, check the signal's volume. To access the test routine, hold the Enter key for 2 s immediately after you switch on the device. A graphic slider moves each of the three frequency samples (1,000, 2,000, and 2,500 Hz) to be recognized. Then, adjust the input volume. The bar moves a bit toward the right or left every time one of the three frequencies used for recognizing the SRC-RAI signal (i.e., on 1,000, 2,000 or 2,500 Hz) is identified.

If the volume is too low, the bar on the display won't move and will have to be turned on. Once you have terminated the check phase, press Enter again. The device, with the display off, will wait to receive a good SRC-RAI signal. As soon as the first two segments of information are received, they will be decoded and displayed. A bar placed in the lower part of the display will move forward, synchronized by six 1,000-Hz beeps.

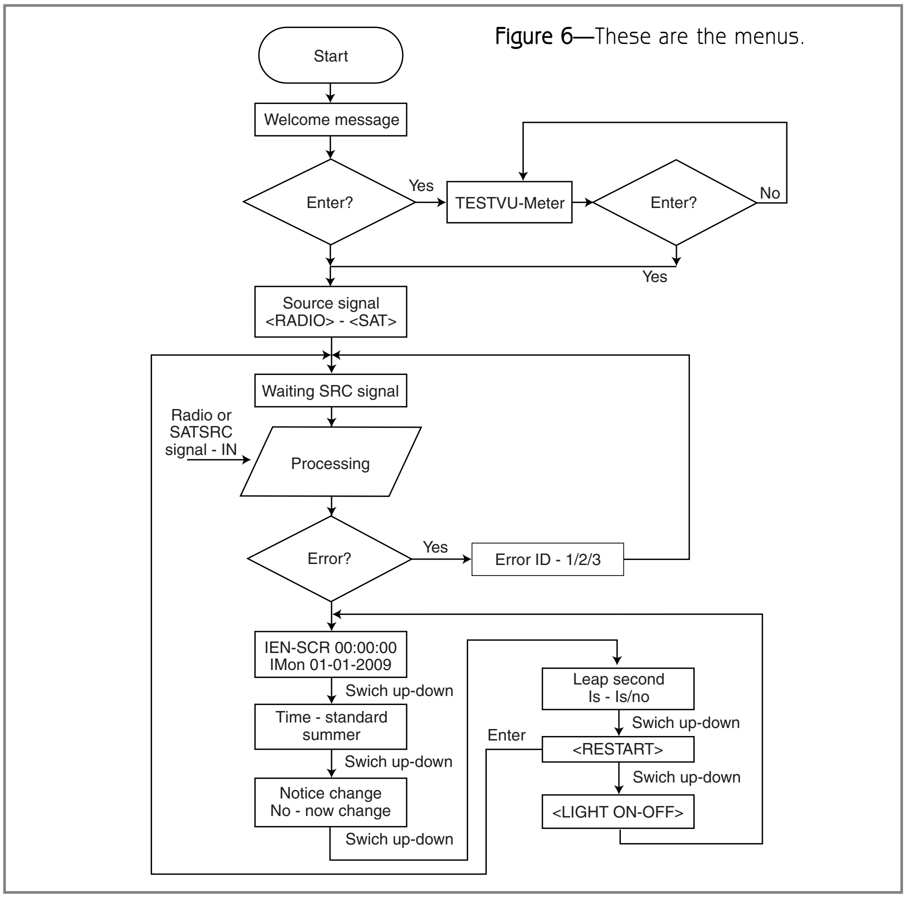


Figure 6—These are the menus.

If problems occur during reception, a warning message will appear. The most common errors reported on the display will have to do with the initial header of the first or second group (HD-1 and HD-2), rather than the noncorrespondence of the control parities of the byte of the information received (P1, P2, and PA). To be honest, during my frequent tests, I found that the reception problems had been limited.

I must point out that the coded SRC hourly signals, which are broadcast in the middle of the day, must sometimes be broken or incomplete for several technical reasons causing the abortion of the reception. Usually there are no problems with signals that are broadcast just before the news or at midnight because an adequate space for a corrected transmission is allotted.

After a positive reception, the decoder's display shows a clock perfectly synchronized to a millisecond. Because the clock's functionality is based on the PIC16F628A's internal oscillator, don't forget that it won't be precise 24 h after the synchronization.

The use of an external 10-MHz quartz doesn't ensure that all of the prototypes we make have the same precision. For example, some inexpensive quartz may have a margin (variance in the frequency) of ±50 ppm (0.005%). This is why programming an EEPROM memory location (38 Hex) with an "adjustment" value has been allocated. By changing this value, when developing the PIC, it is possible to adjust, in advance or later, the internal routine, which is in charge of the clock's adjustment.

Your internal clock is also fully equipped: it updates the complete date, even for leap years until 2035; it automatically updates the time when passing from standard time to daylight saving time and vice versa; and it is a perfect display of the sixtieth second in case of a leap second at 1 A.M. sharp. Central European Time (CET) in Italy is 1 h ahead of Greenwich Mean Time (GMT).

All the other parameters can be displayed using the two special keys on a keyboard. You can page through

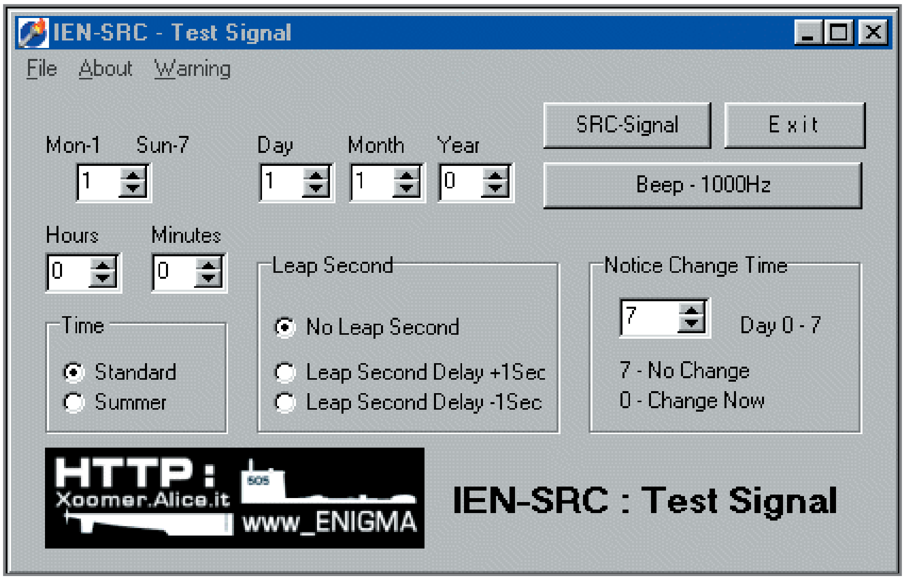


Photo 3—This is a principal panel in the test software.

to the next submenu. You will then know if it is a standard or a daylight saving time. If in the following seven days there will be some changes (and at the end of the month), a leap second will take place. As you can see in Figure 6, the menu enables you to control the display light and perform a reset.

TEST SOFTWARE (DELPHI)

The decoder for the SRC-RAI time signal is equipped with some of the most refined programming technology that can be squeezed out of one small PIC16F628 microcontroller. To thoroughly test your project without having to wait for the signal to be broadcast, refer to the Test-IEN-SRC.exe file on the *Circuit Cellar* FTP site. To use the software, you need a PC running Windows XP and a sound card. The program's main screen is shown in Photo 3.

After you set all the necessary parameters, the signal is ready to be transmitted in an entire block. You can then manually enter the beeps at 1,000 Hz for the transmission.

The software includes a wave format recording of a broadcast time signal. You can use it to test your circuit as well. Please consider this software as a mere test program with no control procedure. Good luck hunting for signals and then decoding and validating the received data! ◼

Author's note: You can enter GPS coordinates at www.gpscoordinates.eu/show-gps-coordinates.php to see images of the time signal transmission facilities via satellite. Enter 45.015037, 7.639634 for the IEN-IRITI in Torino, Italy. Enter 50.016000, 09.007600DCF77 for the Mainflingen signal in Hesse, Germany. Enter 54.911227, –3.279997 for the National Physical Laboratory, which transmits the Anthorn station in London, England. Enter 46.408300, 6.252500 for the HBG transmitter in Prangins, Switzerland.

Danilo Consonni (www_enigma@lycos.it) is an electronic engineer who lives in Italy, where he studied electronics at the Politecnico di Milano. He develops circuits and writes for several publications based in Europe. Danilo has been active in the computer security field for more than 15 years. You can contact him on the web at http://xoomer.virgilio.it/www_enigma.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/225.

SOURCE

PIC16F628A Microcontroller
Microchip Technology, Inc. | www.microchip.com

by Robert Lacoste

Time Domain Reflectometry

Detect and Measure Impedance Mismatches

Robert demystifies the topic of time domain reflectometry, which enables you to detect, measure, and locate any impedance mismatch in a transmission line. To do so, he explains the theory behind it and presents some practical experiments.

Welcome back to the Darker Side. We all have favorite topics. One of mine is impedance matching. I know I have already brought it up in previous columns while discussing antennas, filters, or microstrips, but this month, I promise I will present it from another angle: time domain. You may have read sentences such as, "When matching is not perfect, a part of the signal is reflected back to the source." This may seem strange for engineers not used to high-frequency effects. Imagine the worst case of an impedance mismatch: a wire grounded at one of its ends. Do you think there could be any signal, reflected or not, in such a wire? Of course, and I will show it to you!

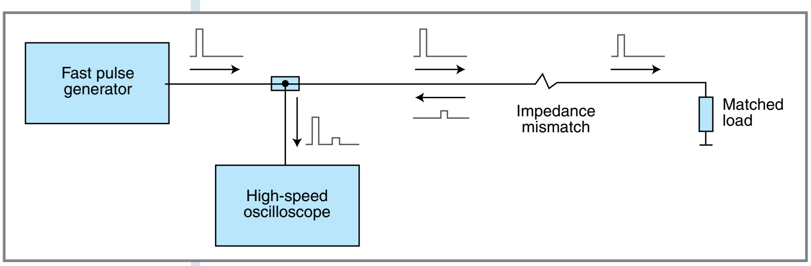


Figure 1—A time-domain reflectometer (TDR) includes a fast pulse generator and a way to display the reflected pulses, usually a high-speed oscilloscope. Thanks to a signal coupler, the oscilloscope enables you to display both the initial pulse and any reflected signals.

Signal reflection is in fact at the heart of an old but interesting measurement technique: time domain reflectometry (TDR). TDR enables you to detect, measure, and locate any impedance mismatch in a transmission line. In this article, I'll explain the theory. But more importantly, I'll present some practical experiments to demystify these techniques. You will just need a good oscilloscope.

TDR BASICS

Nothing can go quicker than c = 3×10^8 m/s, the speed of light in free space (except guys

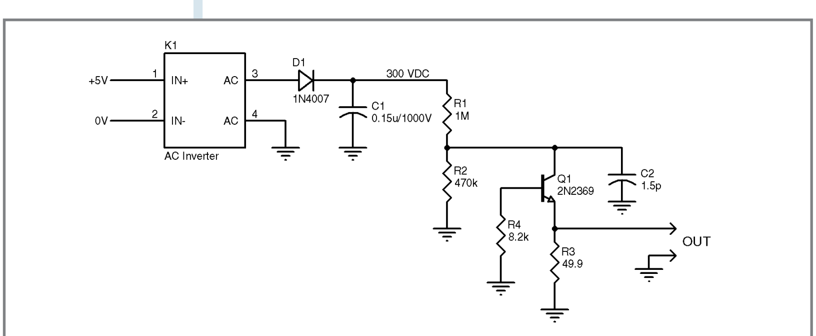


Figure 2—You can build a sub-nanosecond pulse generator for $5 or less using an avalanche mode generator. A high-voltage generator, here built using a CFL backlight power supply, drives an NPN bipolar transistor in its avalanche region, which generates a fast pulse on the output.

Photo 1—The flying SMT technique required some patience, but it enabled me to build a compact pulse generator without any specific PCB (thereby minimizing any parasitic inductance or capacitance).

jumping from black hole to black hole, if you trust some science fiction authors). The speed of an electrical signal going through a wire is a little lower than c, due to the velocity factor of the transmission line, which is always slightly below unity.

Imagine that you have an infinite wire or a sufficiently long wire terminated in its proper impedance-matching load, which is equivalent. Any signal will flow through the wire and will be absorbed by its matched load. No problem, no reflection. Now imagine you have a long, perfect cable that is grounded at its far end. On the other end of the cable, connect a voltmeter and a current-limited 10-V power supply, and switch on the power supply. What will happen? If you don't consider the cable length, then of course the power supply will be short-circuited to ground through the cable and the voltmeter will simply read 0 V. But there is no way to immediately know that the other end is grounded. The electrical signal will need to propagate through the cable up to the end to "see" that it is grounded. Then some information will need to return to give 0 V on the voltmeter. Practically speaking, if you replace the voltmeter with a fast oscilloscope, you will effectively see that the line voltage will at first be 10 V. It will drop down to 0 V only 2T later, with T being the time needed for the electricity to travel through the wire!

You can also interpret this phenomenon as if the 10-V input signal was reflected back from the grounded end as a –10-V signal, giving 0 V as soon as both signals are summed up, and this is effectively the case. In more complex applications, there may be several impedance changes through the wire, and each will reflect back a signal. The shape of the reflected signal will be characteristic of the mismatch. Its time position, relative to the initial pulse, will be directly proportional to the distance from the source. This is TDR, which is an invaluable technique for locating faults (e.g., in underwater communication lines and similar applications) and pinpointing impedance-matching issues (e.g., on high-speed PCB tracks).

TDR can be performed with either a step signal as an excitation, as in my previous example, or with a quick pulse. I will use the latter in this article because the interpretation of the signals is a little simpler. The basic setup for a pulse-based TDR system is shown in Figure 1. A generator provides a sharp and short pulse, which is sent to the transmission line to be tested through a signal splitter, enabling you to connect a high-speed oscilloscope while not perturbing the impedance of the wire. The oscilloscope will then display both the initial pulse and any pulses reflected by the wire. Note that the length of the cable between the splitter and the oscilloscope doesn't matter because both the initial pulse and the reflected pulses have to support the same delay through this cable.

Let me write a few words about 50-Ω signal splitters. Such a splitter can be built with three 17-Ω resistors in a star configuration. The 17-Ω value enables you to keep a 50-Ω impedance on all branches. Why? Because each of the two output branches are supposed to be connected to a 50-Ω load, so each will have a 67-Ω impedance (i.e., 17 + 50) thanks to the 17-Ω serial resistance. This gives 33.5 Ω as both branches are in parallel. Just add the last 17-Ω resistor in series and you are back to 50 Ω. Magical, isn't it? So you could build a 50-Ω splitter just with three resistors, but it is far easier to achieve good performances with an off-the-shelf splitter, especially when manipulating sub-nanosecond signals. The only disadvantage of such a resistive splitter is that a 6-dB loss is incurred in each of the two branches, but that's life.

1-NS PULSE GENERATOR

Unfortunately, there is a problem with TDR techniques. If you need a good distance resolution, then you must generate and detect quick pulses. Consider a standard transmission line with a velocity factor of, say, 0.8. The speed of light is 30 cm/ns in free space, so it is 24 cm/ns (i.e., 0.8 × 30)

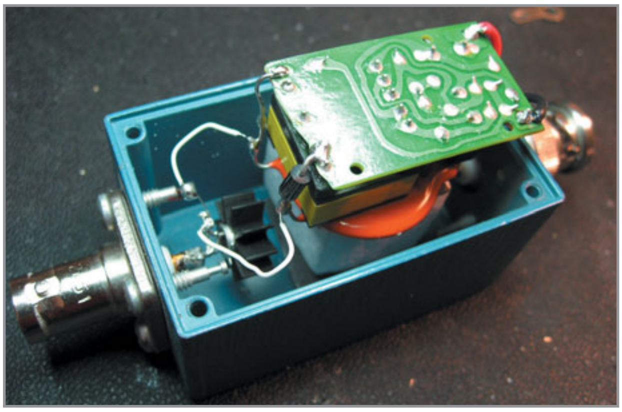


Photo 2—I soldered the pulse generator transistor directly on the output connector and added a reused CFL backlight DC/AC converter with a 1N4007 rectifying diode and a small 1,000-V ballast capacitor to provide a 300-VDC supply. In fact, 100 V would be enough. I added a small heatsink on the transistor just in case, but it seems useless.

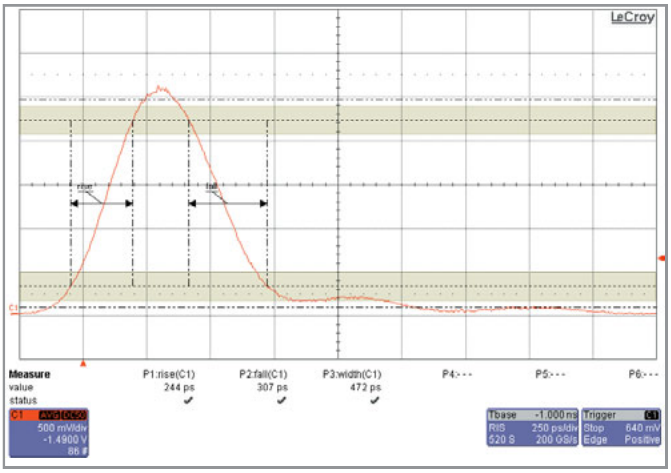


Photo 3—This is the output of the avalanche generator, grabbed on a 1-GHz digital oscilloscope. The pulse rise time is measured at 244 ps, far below the oscilloscope's specified rise time. The pulse width is less than 0.5 ns.

in this wire. Because the signal must go back and forth, you will need to be able to manage 1-ns signals to get a 12-cm distance resolution.

There are two issues: oscilloscope and generator. As for the oscilloscope, I can't help you. Of course, if you just need to locate a problem within tens of meters, a low-cost 50-MHz oscilloscope will be fine. But, if you need to work with tens of centimeters, you will need a high-end oscilloscope (500 MHz or even 1 GHz or more). If you have a tight budget, look for an old Tektronix 7000 series on the Internet.

As for the generator, I can help you build a high-speed, sub-nanosecond pulse generator for less than \$5. As you can see in Figure 2, it can't be simpler, right? Well, I must admit that I had to read it twice when I first saw this concept in an old National Semiconductor application note. It is quite unusual to see a transistor with a grounded base generating anything—and ultra-high-speed pulses in particular. K1 is simply a DC/AC high-voltage converter, the kind of converter used for CFL display backlights. With D1 and C1, this is a convenient, inexpensive way to generate a 300-VDC voltage. Yes, 300 V. This voltage is then used to charge the small C2 capacitor (1.5 pF) through R1. And this is where the magic happens. At a given point in time, the voltage on C2 exceeds the avalanche breakdown voltage of Q1, usually around 60 to 80 V. Q1 then briefly conducts and discharges C2 through R3. This generates a pulse on the output. The pulse's duration will be roughly proportional to C2. But more importantly, the pulse's rise time will be short, because the avalanche phenomenon is fast due to the underlying physics. Intuitively, with such a high voltage, the electrons will have a lot of energy and will be able to jump over the transistor's barrier very quickly. Some transistors are better than others for this application. The old 2N2369s are fine, so I have one. If you need longer pulses, just increase C2.

WHERE ARE YOUR MAGNIFYING GLASSES?

Figure 2 is simple, but you need to be careful as you build it. Its performance will depend on the parasitic component values. Any useless wire in the critical section of the design (i.e., between C2, Q1, R3, and the output connector) will inevitably introduce parasitic inductances, which will drastically degrade the pulse generator's performance. You need to build it as small as possible. Surface-mount versions for C2 and R3 will at least yield better results than classic packages; however, I don't know if there are good SMT equivalents for the 2N2369. You may design a custom SMT PCB for the generator. But on my side, I used an unusual assembly technique, which I don't recommend for more complex designs, or for trembling engineers. Let's call it the flying SMT technique, or FST (see Photo 1). The idea is to use SMT components for all passives and to solder them directly to the 2N2369 leads. It works well, but it is a little annoying to build because these nasty SMTs don't want to stay where you've soldered them. It worked for

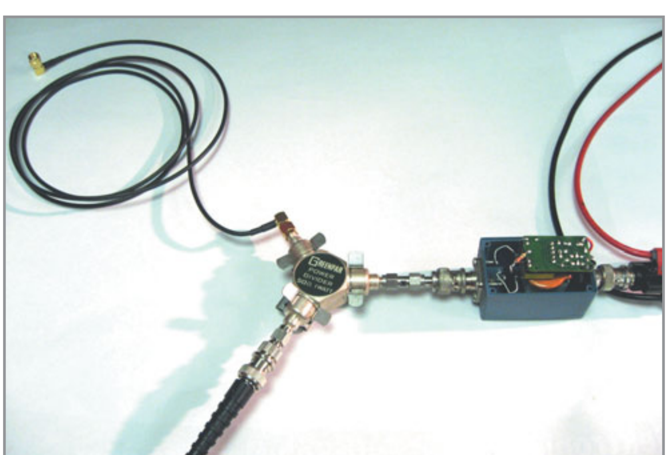


Photo 4—Check out my TDR setup. The custom pulse generator drives a three-way 50-Ω splitter (an old Greenpar model in this case). One output of the splitter (on the bottom) is connected to the oscilloscope through a 50-Ω coaxial cable. The other drives the system under test, which is a simple 1.5-m unterminated SMA cable.

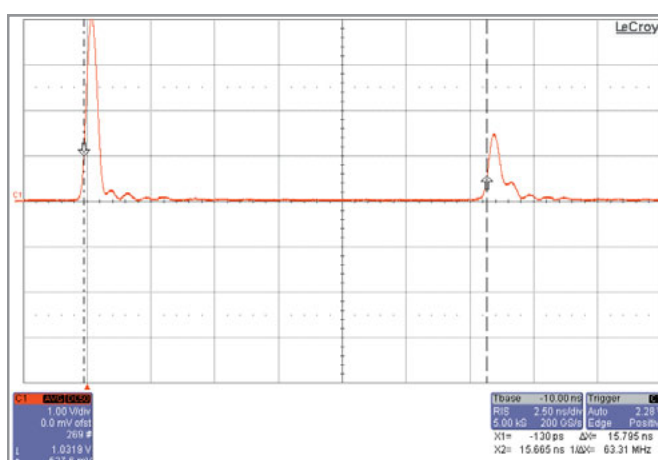


Photo 5—When a TDR is connected to an open-ended transmission line, there is a positive reflected pulse. The amplitude of the reflected pulse is equal to the incident pulse, but here the resistive three-way splitter induces a 6-dB loss. Thus, the voltage is theoretically divided by two (here a little more due to additional losses).

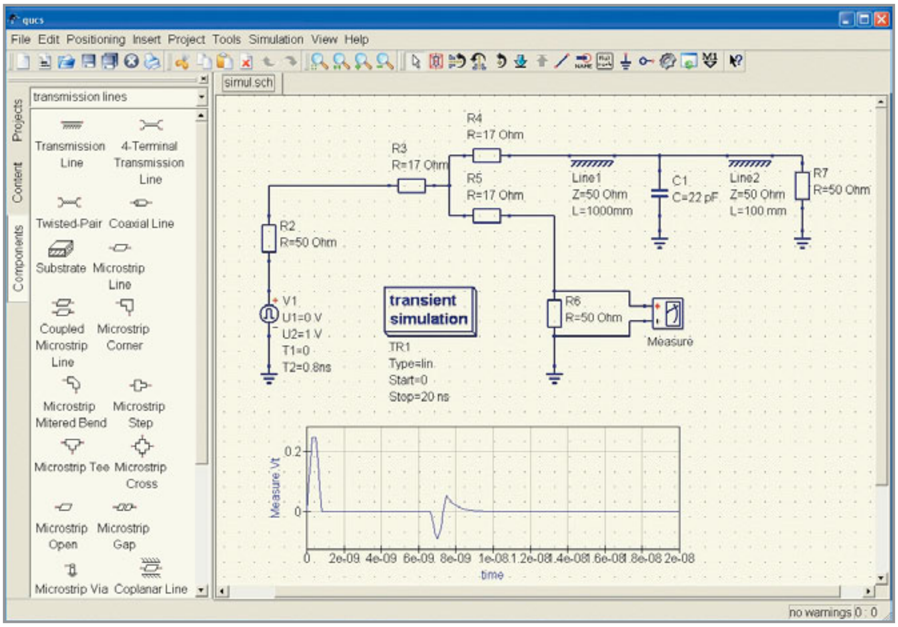


Photo 6—QUCS makes it easy to simulate a TDR experiment. A pulse generator drives an ideal three-way resistive signal splitter made with three 17-Ω resistors. One output drives a transmission line (here two 1-m lines with a parasitic parallel capacitor in the middle). The other drives a virtual voltage probe.

me after working patiently for 30 minutes, so it should work for you too.

The power supply section of the design is not critical. The entire assembly can fit in a small shielded box (see Photo 2). Just make sure the output wire from Q1 to the output connector is as short as possible.

FIRST EXPERIMENTS

I am sure you want to know about the actual performance of this $5 avalanche generator. Photo 3 was taken with a high-end 1-GHz Lecroy WaveRunner 6100 digital oscilloscope (which provides no less than 10-Gsps single-shot and 200-Gsps equivalent sampling speed for repetitive signals), using 50-Ω input impedance. The pulse rise time was measured at 244 ps, including the rise time of the oscilloscope itself, which is specified at 400 ps, so the pulse generator may be far quicker! The pulse width is around 0.5 ns, which is not bad. Such a pulse has frequency components up to 1 GHz or so, so it could be a helpful generator for numerous experiments. Keep it on your bench just in case.

It is time to show you my first actual TDR measurement. The test setup is in Photo 4. The pulse generator is connected to an off-the-shelf Greenpar three-way 50-Ω resistive splitter (obsolete). One of the outputs of the splitter is connected to the oscilloscope through a 50-Ω cable. The other is connected to a 1.5-m open-ended SMA cable (see Photo 4).

Switch on the oscilloscope, set the trigger voltage high enough to synchronize only on the initial pulse, and you get Photo 5. As expected, there is a reflected pulse 15.66 ns later than the original signal. Assuming a velocity factor of 0.7, and keeping in mind the factor of two for back and forth directions, this means the discontinuity was 1.64 m away (i.e., $0.7 \times 3 \times 10^{8} \times 15.66.10^{-9}/2$ m). Not too far from the actual 1.5-m cable length, the difference is probably due to the delays in the splitter itself or to a slightly different velocity factor. Moreover, the shape of the reflection signal is useful. Remember my first example of a shorted wire, which gave a negative reflected signal? The line is open-ended, and in such a case, the reflected signal is positive. If you have a step generator rather than a pulse generator, the reflected signal will add to the incident signal and will double its amplitude, as both have the same sign. This is normal because the voltage on an open-ended 50-Ω generator is twice its voltage when loaded with a matched 50-Ω load.

LET'S SIMULATE IT

Before going on the test bench or even in the field with your basic TDR system, it is nice to have a list of reflected pulse shapes for the different "usual" impedance mismatches: increase or decrease of the resistive impedance, parallel or series parasitic capacitor or inductance, and more. I could have built a dozen different test benches and measured the actual behavior, but using simulation is a wonderful time-saving tool. The only issue is that a classic analog linear simulator like Spice can't easily handle line-length effects, so it isn't appropriate to simulate TDR effects. Fortunately, you can use the free Quite Universal Circuit Simulator (QUCS), which I used in a previous column. The QUCS simulation of a parasitic parallel capacitor in the middle of a transmission line is shown in Photo 6. When a pulse is applied to a capacitor, this component first behaves as a short circuit, giving a negative reflected pulse similar to a short-circuited line. Then the capacitor slowly loads and the reflected

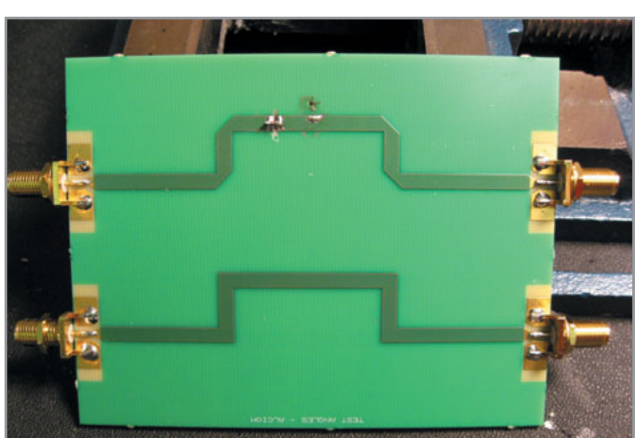


Photo 7—I reused the microstrip test board I presented in my *Circuit Cellar* 223 article. I added a parasitic serial or parallel 22-pF SMT 0805 capacitor in the middle of the line to compare the theoretical TDR behavior and an actual one.

pulse is positive and exponentially decreasing, with a time constant proportional to the capacitance.

Based on this simulation, it is easy to simulate all the other classic disturbances. The results are provided in Figure 3. All corresponding QUCS simulation files are posted on the *Circuit Cellar* FTP site if you want to play with them. Figure 3 shows that a parallel resistor gives a reflected pulse shape similar to a short-circuited line: a negative pulse, but with a smaller amplitude than a full short circuit. Similarly, a series resistor is a small open circuit with a small positive reflected pulse. You can also see that a series inductor gives a shape similar to a parallel capacitor but with an opposite polarity. Parallel inductors and series capacitors also have a dual behavior.

REAL LIFE VS. SIMULATION

Let's compare simulation with real life. I'll begin with a parallel capacitor. Note that I reused the small S-shaped 50-Ω microstrip PCB that was built for my February 2009 article, "Microstrip Techniques" (*Circuit Cellar* 223). I soldered a 22-pF 0805 SMT capacitor in the middle of the microstrip line, with its other end grounded (see Photo 7). I connected the microstrip board at the end of the SMA cable used in Photo 4, connected another 1.5-m cable at the other end of the microstrip test board, and finally used a 50-Ω SMA load to provide proper matching. So, the test setup is a 3-m, 50-Ω line with a 50-Ω load at its end, but with a parasitic 22-pF parallel capacitor to ground at the middle. I switched on the oscilloscope and pulse generator and, voila, I got what you see in Photo 8a. Comparing it with the theoretical shape for parallel capacitors, Photo 6 shows that the overall shape is similar, with a first negative pulse, then a smaller positive one. There are other small pops probably due to other impedance mismatches (i.e., far from perfect ground connection of the capacitor, nonideal capacitor, and nonideal connectors, and more). But once again, the overall shape is similar.

Do you want another test? This time, I hooked the same capacitor in series with the line, just by cutting 1 mm or so out of the microstrip and soldering the 22-pF capacitor across the gap. The result is Photo 8b. Once again, it is similar to the theoretical shape for a series capacitor, but with additional bumps in particular at the beginning of the pulse.

Another interesting use of TDR techniques is the evaluation of the performances of connectors, in particular at high frequencies. TDR will easily show you defective or any less-than-ideal connectors.

I performed a simple test by hooking a good 50-Ω load at the end of a test 50-Ω cable. Theoretically, a 50-Ω load shouldn't reflect anything. But because the load and the SMA connectors were not 100% perfect, there was a small reflected pulse. I increased

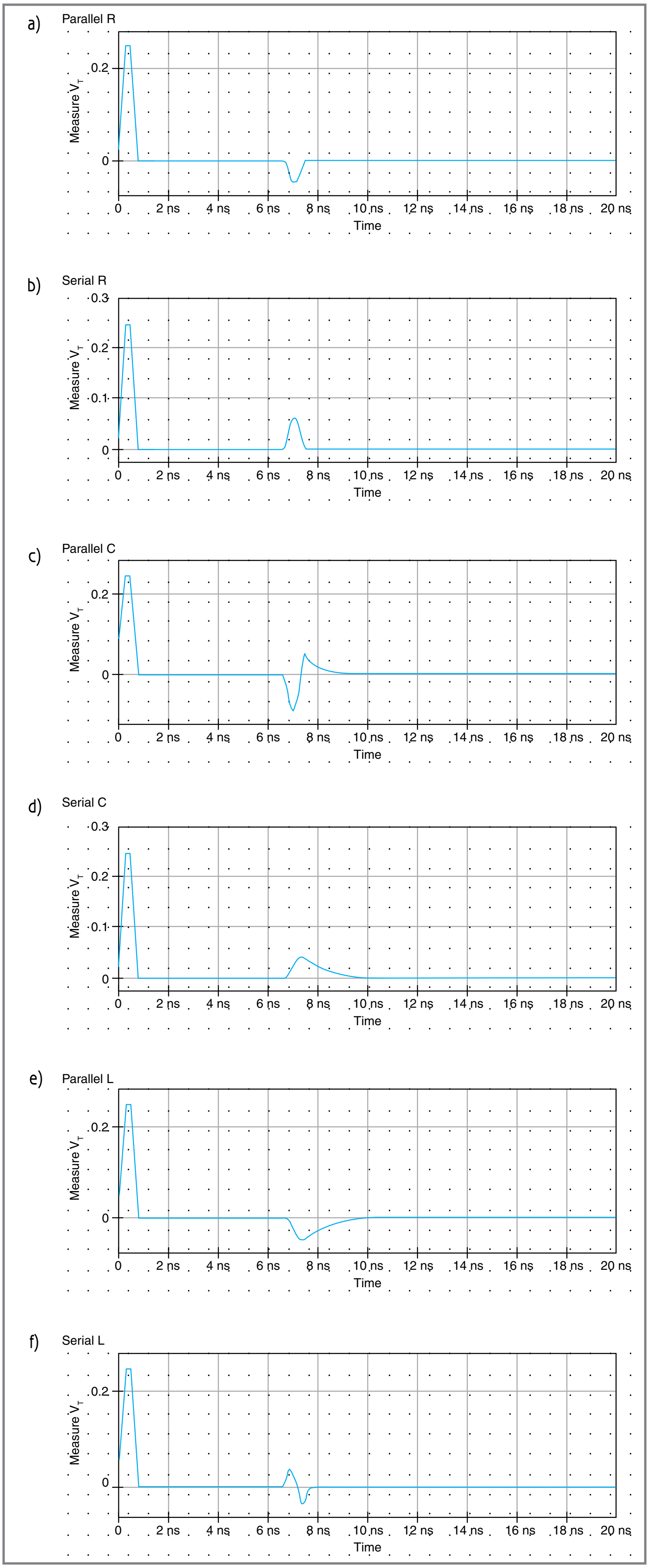


Figure 3—These are the results of the QUCS simulation of the reflected shapes for the six elementary impedance mismatches: parallel and serial R, C, and L.

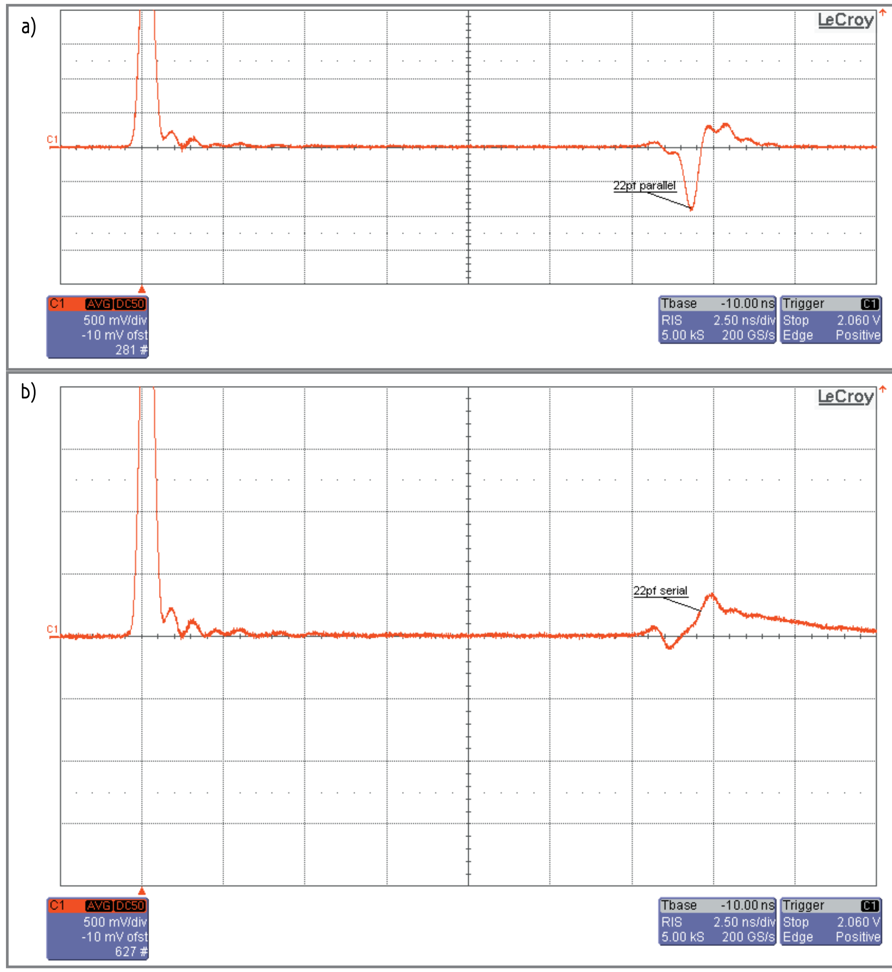


Photo 8—Experimental TDR waveforms with (a) a 22-pF parallel capacitor or (b) a 22-pF series capacitor in the middle on a microstrip transmission line. Just compare these shapes with the corresponding theoretical shapes provided in Figure 3.

the vertical sensitivity of the oscilloscope and was easily able to see it. Refer to the top curve in Photo 9. Next, I inserted an SMA to BNC and BNC to SMA adapters pair between the SMA cable and the same SMA load. The result is the bottom curve in Photo 9, with the same vertical settings. Do you see a difference? I conclude that you shouldn't use BNC connectors for high-frequency designs if you're looking for good impedance matching.

WRAPPING UP

I covered some of the potential applications for TDR. Even with a poor-man's pulse generator and a good oscilloscope, you can easily pinpoint impedance-matching problems on cables and transmission lines. Moreover, a quick look at the shape of the reflected pulse will enable you to qualitatively get a good idea of the kind of defect.

I read that TDR engineers working on the maintenance of submarine lines can easily guess if a problem is related to water ingress, corroded contacts, or something similar just by looking at the TDR shapes. You now know why.

I haven't discussed the mathematical aspects of TDR. This may be the subject of another interesting article, particularly because a simple fast Fourier transform (FFT) of the TDR signal can bring you back in the frequency domain. You can then deduce the line's band-pass simply by looking at its TDR shape, at least if there aren't any losses. As a TDR setup works only by looking at reflections, it may not detect a signal absorbed in a line and not reflected back.

I hope this journey into TDR has been enjoyable. Don't hesitate to test it, play with it, and send me the results of your experiments. TDR is no longer on the darker side for you. ■

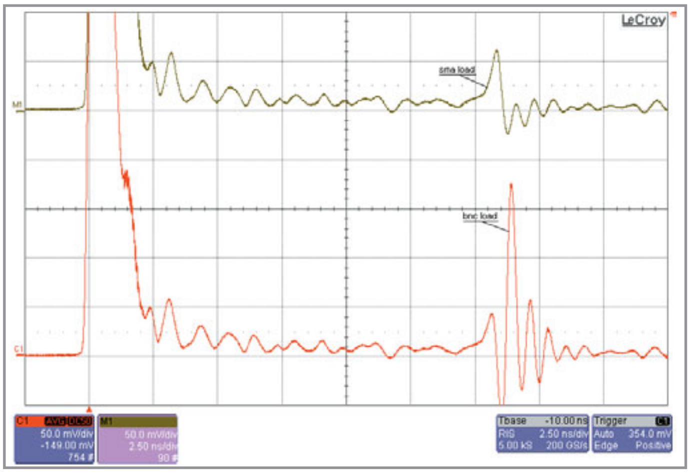


Photo 9—The vertical scale is increased to show the difference between the parasitic reflections with an SMA 50-Ω load (top) and with the same load connected through two SMA/BNC adapters (bottom). Conclusion: I don't like BNC.

Author's note: Caution! If you build the pulse generator described in this article, remember that there is a capacitor inside charged at 300 VDC, even if the output signal is a low-power 50-Ω signal. The 300 VDC could be significantly harmful, or even lethal, so please take care.

Robert Lacoste lives near Paris, France. He has 18 years of experience working on embedded systems, analog designs, and wireless telecommunications. He has won prizes in more than 15 international design contests. In 2003, Robert started a consulting company, ALCIOM, to share his passion for innovative mixed-signal designs. You can reach him at rlacoste@alciom. com. Don't forget to write "Darker Side" in the subject line to bypass his spam filters.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/225.

RESOURCES

T. Engdahl, "Time Domain Reflectometer," 2007, www.epanorama.net/circuits/tdr.html.

Hewlett-Packard, "Application note 1304-2: Time Domain Reflectometry Theory," 1988, http://materias.fi.uba.ar/6209/download/HP-AN1304.pdf.

F. Sischka, "TDR Measurements and Calibration Techniques," Agilent Technologies, 2002, http://eesof.tm.agilent.com/docs/iccap2002/MDLGBOOK/1MEASUREMENTS/71TDR/1TDRCalibration.pdf.

J. Williams, "Application Note 47: High Speed Amplifier Techniques: A Designer's Companion for Wideband Circuitry," Linear Technology Corp., 1991, www.mit.edu/~6.331/an47fa.pdf.

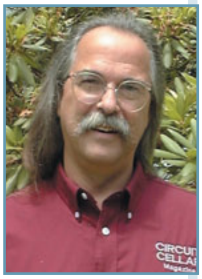
SOURCES

WaveRunner 6100 Oscilloscope
Lecroy Corp. | www.lecroy.com

2N2369 Transistor
Multicomp | www.farnell.com (Distributor)

Qucs project
Qucs | http://qucs.sourceforge.net

by Jeff Bachiochi



Programmable Robotics (Part 2)

Application Development

When Jeff couldn't add a Bluetooth adapter module to iRobot's Create (because it used the same cargo bay connector as the Command Module), he did what any good designer would do: he built his own. This month, he explains how to replace the Command Module with a two-serial-port microcontroller.

Last month, I introduced you to iRobot's Create robotic platform, which is based on the successful series of cleaning robots. I discussed the Create's complement of sensors and how they can be accessed using the Open Interface (OI) with the Command Module (CM) accessory. The CM contains an Atmel microcontroller that you can program in C or C++ using your PC. The resultant code is downloaded onto the CM with a USB interface. I demonstrated how you can use Atmel's AVR Studio to write your application in assembly language. Yup, some of us still like to get our hands deep into the bits and bytes. I also examined how the USB tether used in downloading could provide real-time debugging feedback. Obviously, you can't have a robot running around the floor with a long tether. Another accessory for the Create is the Bluetooth Adapter Module (BAM). The BAM gives you a wireless link to your PC. The only catch here is that it uses the same cargo bay connector that the CM requires. This means you must choose between either wireless communication or an on-board programmable microcontroller. Something's wrong with this picture.

This month, I will discuss how to replace the CM with a Microchip Technology PIC microcontroller that has two full serial ports so the BAM can be used along with the PIC. But first, I want to explain why I took this approach. If you remove the CM's plastic top, you will find an Atmel ATmega168 microcontroller. This flash memory microcontroller has a UART that's used for two functions: sending OI commands to and receiving data from the Create through the cargo bay connector and programming the microcontroller through a USB port. The paths to and from the UART are multiplexed externally and controlled by the microcontroller. I looked for a way to cleanly interrupt the data to the serial-to-USB chip (FT232) so I could replace the USB port with an alternative cargo bay connector (onto which the BAM could be connected). However, this could not be accomplished without wrecking the CM. I decided to prototype the idea. But when I started thinking about how I might use this, I realized that at some point my application would surely exceed the flash memory available, no matter the size.

At that point, I sat down and thought about the features that interested me most. I came up with three items: limitless computing power, expandability, and wireless connectivity. It looked like even with successful surgery, I would accomplish only the wireless feature I was looking for by using the CM. My ideal design would not depend on the CM for program storage and execution, but merely use the CM as a way to expand the capabilities of the Create (i.e., providing additional sensor capability via SPI, I²C, or other interfaces—something the Create's analog and digital I/O was not capable of).

In reality, I needed two UARTs. With the BAM on one port and the Create on the other port, the microcontroller could pass OI commands from one UART to the other. Additional commands (OI extensions) could be created for any sensor handled by the microcontroller. The microcontroller would play traffic cop and choose to keep an OIE command, but pass through all original OI commands.

BIG PICTURE

I assumed my future application would require a lot of computing power, not to mention a potentially large database. Therefore, I would write and execute it on a PC. The BAM wireless interface feeds the Create platform with commands and receives sensor data. Although the Create is autonomous, it does not have to carry around its brain. I can write and run an application from my recliner outfitted with the appropriate beverage.

The Create has a built-in application for finding its charging dock and replenishing its power pack, so it should never be found dead somewhere. The application can monitor battery capacity and send the Create home, when necessary.

Because I was redefining the CM, there was no particular reason to stay with the ATmega168. Not that there is anything wrong with it if you want to stick with an Atmel part. I chose a Microchip Technology dsPIC30F4012, a part that I hadn't used before. It operates on 5 V, comes in a DIP-40 package, has two hardware UARTs, and executes at close to 30 MIPs.

dsPIC

While the flash memory microcontroller has an internal 7.37-MHz RC oscillator that has been trimmed to 2%, I added an external crystal to give spot-on data rate generation. The (4×/8×/16×) PLL can give the source frequency a real boost, putting it far above the speeds that I've become used to. Refer to Figure 1 to see how the dual UARTs are used. All communication signals are TTL-level (uninverted). Between the microcontroller, the Create, and the BAM, no RS-232 level shifting is necessary. I added connections for the ICD2 emulator so execution can be easily debugged. Although it has not been implemented, both SPI and I²C interfaces have been included for whatever sensors I might add (i.e., compass and ultrasonic ranger).

The SPI and I²C interfaces share some of the same signals with the in-circuit programmer (ICP), so I implemented some buffering that should enable these to coexist. The SPI and I²C clocks have separate enables so the bus can be reconfigured and appropriately enabled, when necessary. The ICP needs control to program the part. One aspect of programming is the ability to reroute some signals to and from alternate pins. I chose to use alternate pins for the in-circuit debugger (ICD). By using alternate pins

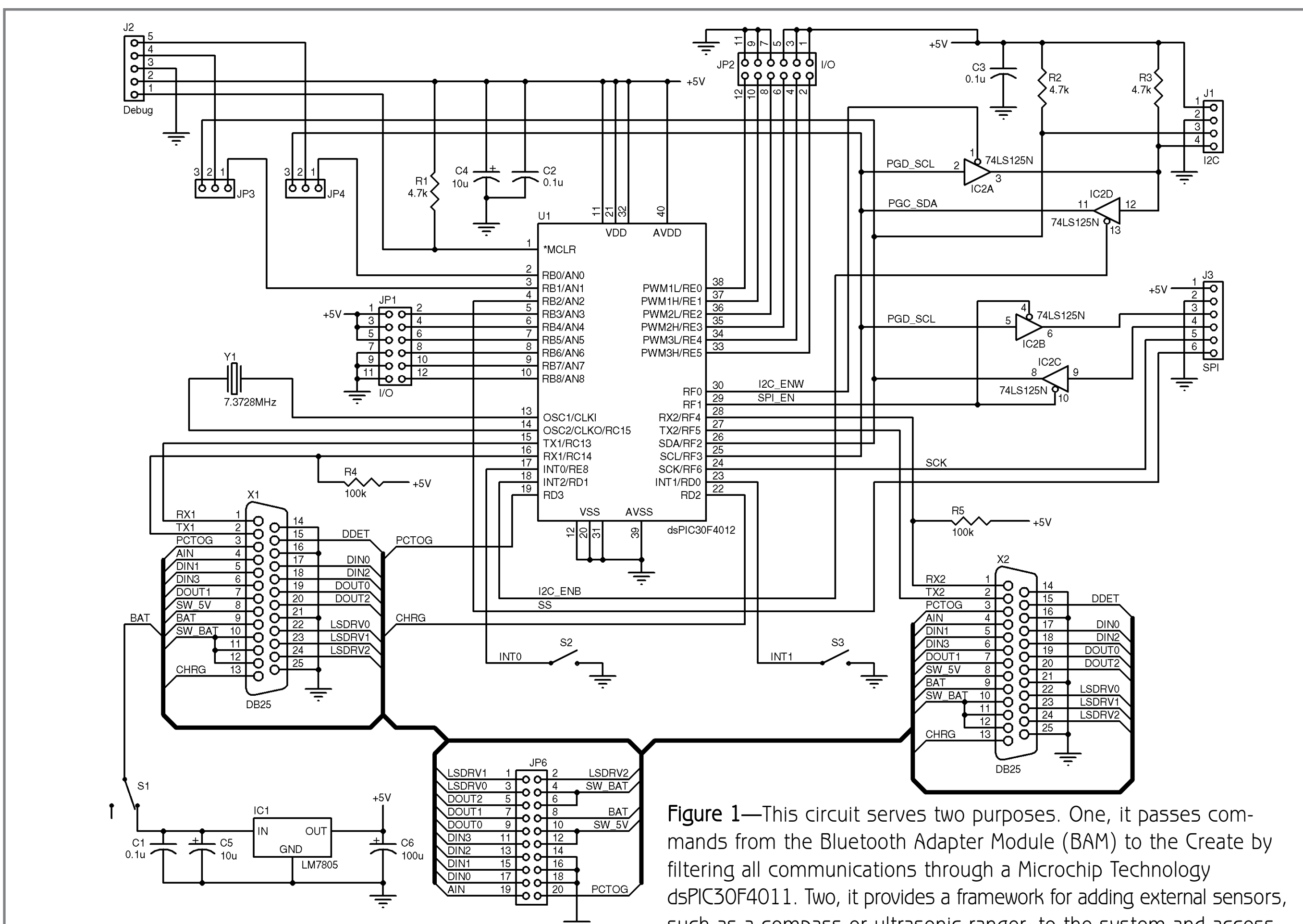


Figure 1—This circuit serves two purposes. One, it passes commands from the Bluetooth Adapter Module (BAM) to the Create by filtering all communications through a Microchip Technology dsPIC30F4011. Two, it provides a framework for adding external sensors, such as a compass or ultrasonic ranger, to the system and accessing them with Open Interface Extension (OIE) commands.

for debugging, the I²C and SPI lines are free to operate normally.

The microcontroller's basic function is two-fold: to handle the serial traffic between the BAM (UART2) and the Create (UART1) and to interface with any sensors added to the system. The microcontroller (I call it Me) and each UART (Create and BAM) will have two ring buffers associated with them—one for TX and one for RX—for a total of six. Both of the UART's RX buffers are automatically filled by RX interrupt routines when data is received. Anything placed in either UART's TX buffer is automatically transmitted by TX interrupt routines.

Any data coming from the Create (in the UART1RX buffer), as well as any data coming from the microcontroller (in the MeRX buffer), must be melded together into the BAM (the UART2TX buffer). Most OI data does not have any kind of wrapper; therefore, I had to interleave data from the microcontroller into a response from the Create.

To keep the Create's data packets as a single entity, I needed to know how many bytes to expect. For most commands, this is a fixed number of data bytes. When a command byte comes from BAM, its value is used as an offset into a look-up table. Each potential command (128–255) has two entries: the number of data bytes that follow the command and the number of data bytes returned in response to the command. The microcontroller uses the second entry ResponseLength to keep data from the Create in one piece (or packet). Figure 2 depicts how the upstream responses flow.

The first table entry DataLength helps the microcontroller know how many data bytes to expect after a command. This keeps the communication in-sync and unmingled. There are, however, a few commands that can have a variable number of data bytes following the command. These exceptions are treated a bit differently. When an exception is encountered, the DataLength (from the table) indicates where a length byte can be found—usually, it's the next byte following a command. When the byte is transferred, it is also reloaded into

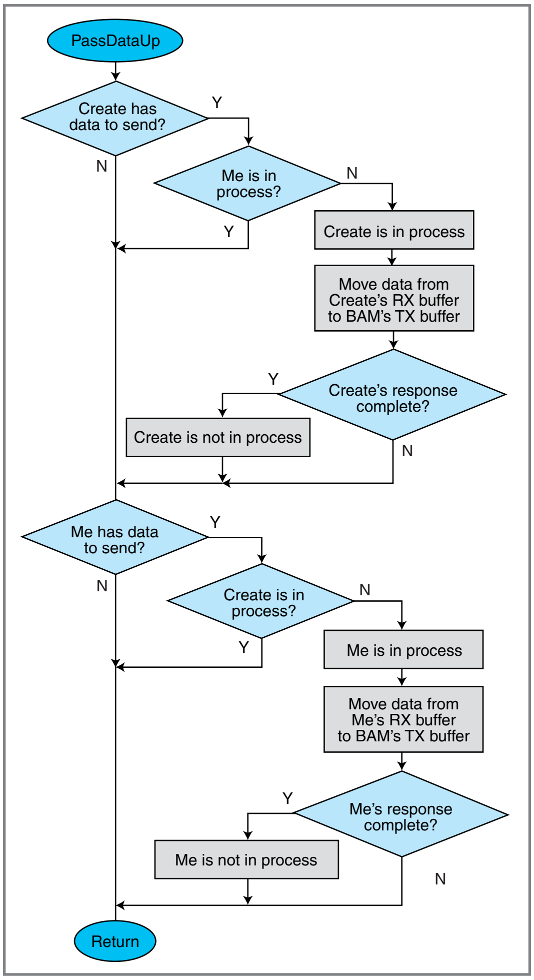


Figure 2—The PassDataUP flowchart shows how responses from the Create and Me (dsP) are sent to the BAM TX buffer without being intermingled.

DataLength. Thus, DataLength is corrected on the fly, enabling the correct number of data bytes to be transferred. Figure 3 shows the flow of the downstream command and data.

Extended OI commands can be developed, which when received from the BAM, will be redirected to the microcontroller where the appropriate function will be executed to control or gather new sensor data and reply, if necessary. While no commands have been written at this point, the OI leaves plenty of room, as it uses only command bytes 128 to 158.

BAM

Element Direct manufactures a number of accessories for iRobot products, including the Roomba and the Create. That's right. If you own a Roomba sweeper, you can take control of its internal microcontroller and experiment a bit with some of Element Direct's offerings (www.elementdirect.com). The BAM gives the Create a roaming radius of up to 100 m, which is plenty for most buildings. Element Direct also offers a Bluetooth dongle, the other end of a Bluetooth connection. I already have a few of these hanging around. I can use a USB Bluetooth dongle on a PC or laptop. Any application that has access to a serial port can take advantage of this connection.

Using the RealTerm terminal emulator, I connected to the BAM and sent and received OI commands. Although I couldn't request and analyze commands and responses fast enough, I used RealTerm to check the connection and test its range. For application programming, I used one of my favorite programs, Liberty BASIC. This let me program in BASIC and enabled me to get off to a running start.

The BAM was used as a serial link. Photo 1 shows my prototyped circuit plugged onto the Create's cargo bay connector with BAM going along for the ride. Like all Bluetooth devices, it first must be linked to the host before it can be used. To do this, make sure it's powered and allow the host to search for (discover) Bluetooth devices. Next, select BAM, which will have the name Element Serial. Pair the devices using passcode 0000. When you refresh its services, you should see a Serial Port Profile (SPP). Check this service to enable it. (Don't forget to make note of the COMport#. You need to use the port number to communicate with the Bluetooth wireless link from the application program in Liberty Basic.) When I plug the Bluetooth dongle in one of the two USB ports located on the front of my desktop PC, the service is available at COM21 on one USB port and COM23 on the other. This is a far cry from the COM1-4 that we all used years ago with just hardware serial ports!

LB APP

Before getting into any application, there are a few routines that will aid in simplifying any future code. All of the Create's constants are defined so the description names can be used instead of

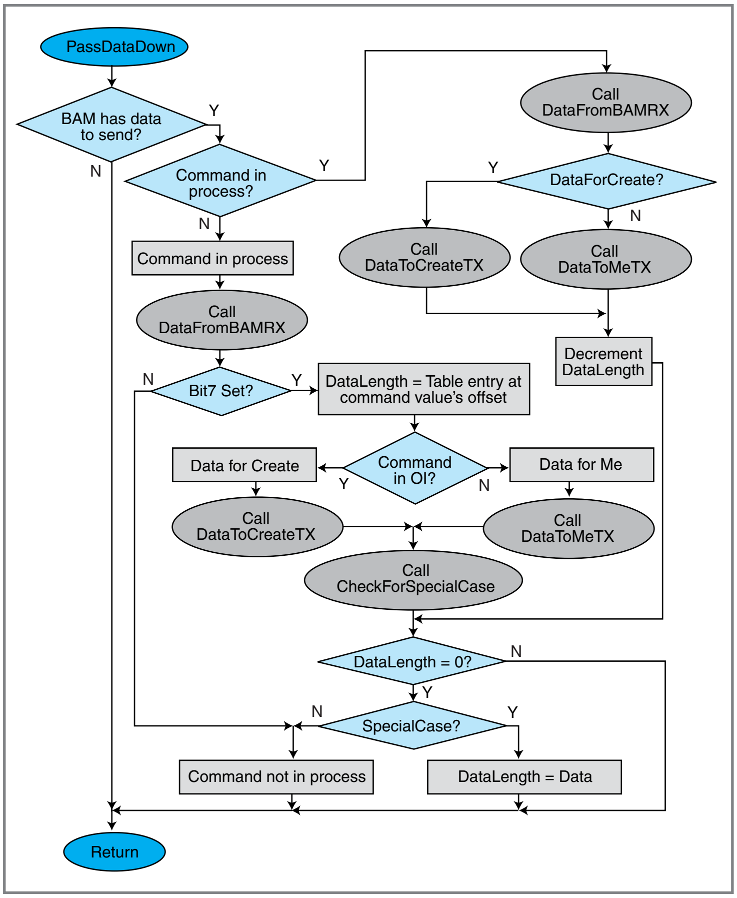


Figure 3—The PassDataDown flowchart shows how a command and any associated data is routed either to the Create—if it's an OI command—or to Me (dsP)—if it's an OIE command.

trying to remember values, such as right bumper sensor (BumpRight)=bit1 or the radius value to drive straight (RadStraight)=32768. I've been refining a COM select routine for some time now. This routine attempts to open each COM port (from COM1 to COMmax) and look for a comm error. Ports with errors are flagged as unavailable. Then a pop-up window displays all of the COM ports as radio buttons with the COM ports flagged as unavailable disabled. You can choose any of the enabled ports. (This is where you must select the service port that you made note of earlier.) Assuming that you've chosen the proper port, you now have a wireless connection between this application and the BAM, the PIC interface, and the Create.

To support the present OI commands, there are four routines that will be used in all applications: writing to the serial COM port, reading from the serial COM port, interpreting each command, and interpreting each response. The simplest OI command is a single-byte command like `CmdStart`. To send OI CmdStart using LB do a '`gosub [CmdStart]`'. This routine simply sets d = 128 and does a '`gosub [WriteData]`' and then returns. It has no response.

Photo 1—I built the Create Commander circuit to the same dimensions as the CM. It fits on the cargo bay connector and provides I/O expansion and Bluetooth wireless connectivity.

A more complicated command like CmdDrive must not only send the command byte (d = 137), but also 16-bit velocity and 16-bit radius values (as byte values with the MSB first). Not only does the velocity value need to be broken down into two high- and low-byte components, but it is a signed value that uses direction to determine whether the value should be positive (forward) or negative (backward). The radius value is also a signed value. In this case, the 16-bit values are predefined as RadCW, RadCCW, and RadStraight. It has no response.

At some point, we need feedback to determine when to stop, either because we've driven far enough, bumped into something, or are about to topple down the stairs. CmdSensors would be appropriate. This command requires one additional byte beyond the command byte (d = 142). We need to ask for a sensor response using a Packet ID. The sensor Distance (PacketID = 19) would appropriately respond with a 16-bit signed number indicating the distance traveled (forward or backward) since the last request. You will need to keep a running tally of the total distance traveled to determine if it's time to stop or not. If you remember from last month, the sensor command is also capable of returning the status of groups of sensors. Packet IDs 0–6 return the status of multiple sensors. Every sensor will return at least 1 byte, with some sensors returning 2 bytes, as in the aforementioned distance sensor.

Responses are handled by the command routine that makes the request. Any routine that expects a response does a gosub [GetPacket] to handle single- and multiple-byte responses. The [GetPacket] routine routes Packet IDs of 7–42 to the appropriate [SaveSensorData] routines that capture the response data using gosub [ReadData] and then saves it into the local variable, which the application will use. With Packet IDs of 0–6, the [GetPacket] routine uses for-next loops to capture the response data from multiple sensors in the proper order.

APP WITHIN THE APP

Although all of these routines are certainly part of the application as a whole, the real app is in having the Create follow some commands. My future application's task will be to go exploring and try to map what it thinks is out there. This sounds easy enough, but remember that right now all we have is the ability to bump an object (and respond) and to dead-reckon position (where the error between your position and where you think you are grows larger the farther you move). If you have experience in mapping or terrain learning, I would like to hear from you.

I went for something simple. I backed the robot out of its charging base a short distance, turned it around, and started exploring with the cover demo. While the Create explored, I continually monitored the battery and prepared to change to the cover and dock demo, if the battery's charge fell to 50%, so it would return it to the charging base.

Listing 1 is the mini app (within the application) to command this behavior. Up to this point, I haven't mentioned a couple of the routines in Listing 1. The [Move] and [Turn] routines are where the robot's intelligence begins to take shape. These are variations using the CmdDrive command. Notice the setup prior to these commands: Distance (for move) and Angle (for turn). These provide limits to the movement. CmdDrive can actually do both at the same time (i.e., it can combine forward or backward movement while turning at some radius). To simplify dead-reckoning calculations, turns

Listing 1—These are the Basic commands for the mini-app to leave the charging base, roam around until it needs a charge, and then go get recharged.

```
rem - Create is in it's charging cradle
  gosub [CmdStart]
rem - Wait for Create to be fully charged
[CheckForFull]
  d=ChargingState
  gosub [CmdSensors]
  if ChargingState<>NotCharging then goto [CheckForFull]
rem - Back up 100mm
[BackOutOfCharger]
  Direction=Backward
  Velocity=200
  Distance=100
  Radius=RadStraight
  Gosub [Move]
rem - Rotate 180 degrees
[TurnAround]
  Direction=Forward
  Velocity=200
  Angle=180
  Gosub [Turn]
rem - Go explore
[Explore]
  Gosub [Cover]
rem - Wait until we need gas
[CheckFor10V]
  d=Voltage
  gosub [CmdSensors]
  if Voltage>10000 then goto [CheckFor10V]
rem - Find Home
[Home]
Gosub [CoverAndDock]
rem - Wait until we're Home
[CheckForHome]
  d=ChargingSourcesAvailable
  gosub [CmdSensors]
  if ChargingSourcesAvailable<>HomeBase then goto
   [CheckForHome]
  goto [CheckForFull]
```

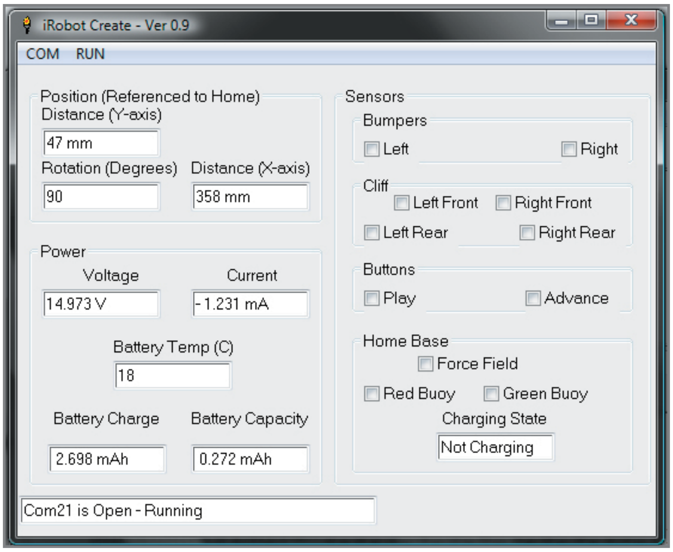


Photo 2—This is the dashboard. It is a real-time display of many sensor values.

will be made in two steps, rotating in place and moving in a straight line. These routines do more than just determine when the robot has reached its destination. They also check sensors for additional (and potentially dangerous) input. These can include bumper contact, cliff sensors, excessive wheel currents, button pushes, battery status, and digital/analog inputs. Each sensor input might have a routine associated with it that would be executed, if necessary. For instance, if there is contact with the bumper, you should provide a routine to back away from the collision and try a different route.

VIRTUAL DASHBOARD

Debugging feedback was one of the topics covered in the first part of this series. With the wireless link and control coming from the Liberty BASIC application running on the PC, the issue of tethering and access to the Create's sensor data is no longer an issue. The BAM eliminates the need for a tether, and because all variables are local, it is easy to display anything.

Photo 2 shows the dashboard I used with this application. I added some computed items that show the Create's dead-reckoning X-Y position in relation to the charging base. This provides an indication of dead reckoning, and you get to see how it relates to the Create's actual position over time. The Power items monitor power consumption. The Sensors checkboxes show when a sensor changes states. For debugging purposes, I use a separate debugging window in which I can print the receipt of sensor data as received. This window can also hold any computational information that would help with the application program's debugging process. Liberty BASIC has its own debugging resource, which is a useful feature.

RDS

Microsoft Robotics Developer Studio (RDS) 2008 is a Windows-based environment for hobbyists, academics, and commercial developers for creating robotics applications with a variety of hardware platforms. It includes a lightweight, REST-style, service-oriented runtime. In addition, it has a set of visual authoring and simulation tools, as well as tutorials and sample code, to help you get started. I mention this because the Create is one of the presently supported robotic platforms.

The idea behind RDS is to enable developers and manufacturers to come together. Your robot can use any manufacturer's device. With RDS, you can model and simulate things before handing your application over to your robot. If I get enough reader interest in the RDS, I'll devote a future column to it.

MAPPING?

As I previously mentioned, I am looking forward to experimenting with a mapping application. While our ultimate input sensor is our eyes, our other senses certainly add to the way we perceive the world. I am amazed by how those without the gift of sight use their other senses to fill in what most of us would consider a huge void. It's obvious that the absence of this ultimate input isn't a showstopper.

So, this is really about learning. Is there a limit to what a robot can learn? Is learning based on its sensing abilities? I believe that if robots are to become more useful to us, they must be capable of learning. Otherwise, we won't put up with them. That's why robotics has yet to live up to its hype. We seem to be constantly pushing for more, bigger, faster solutions. Maybe we need to step back and look at it from a more simplistic point of view. ◼

Jeff Bachiochi (pronounced BAH-key-AH-key) has been writing for Circuit Cellar *since 1988. His background includes product design and manufacturing. You can reach him at jeff.bachiochi@imaginethatnow.com or at www.imaginethatnow.com.*

RESOURCE

iRobot, "iRobot Create Open Interface Specification," 2006, www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface_v2.pdf.

SOURCES

ATmega168 Microcontroller
Atmel Corp. | www.atmel.com

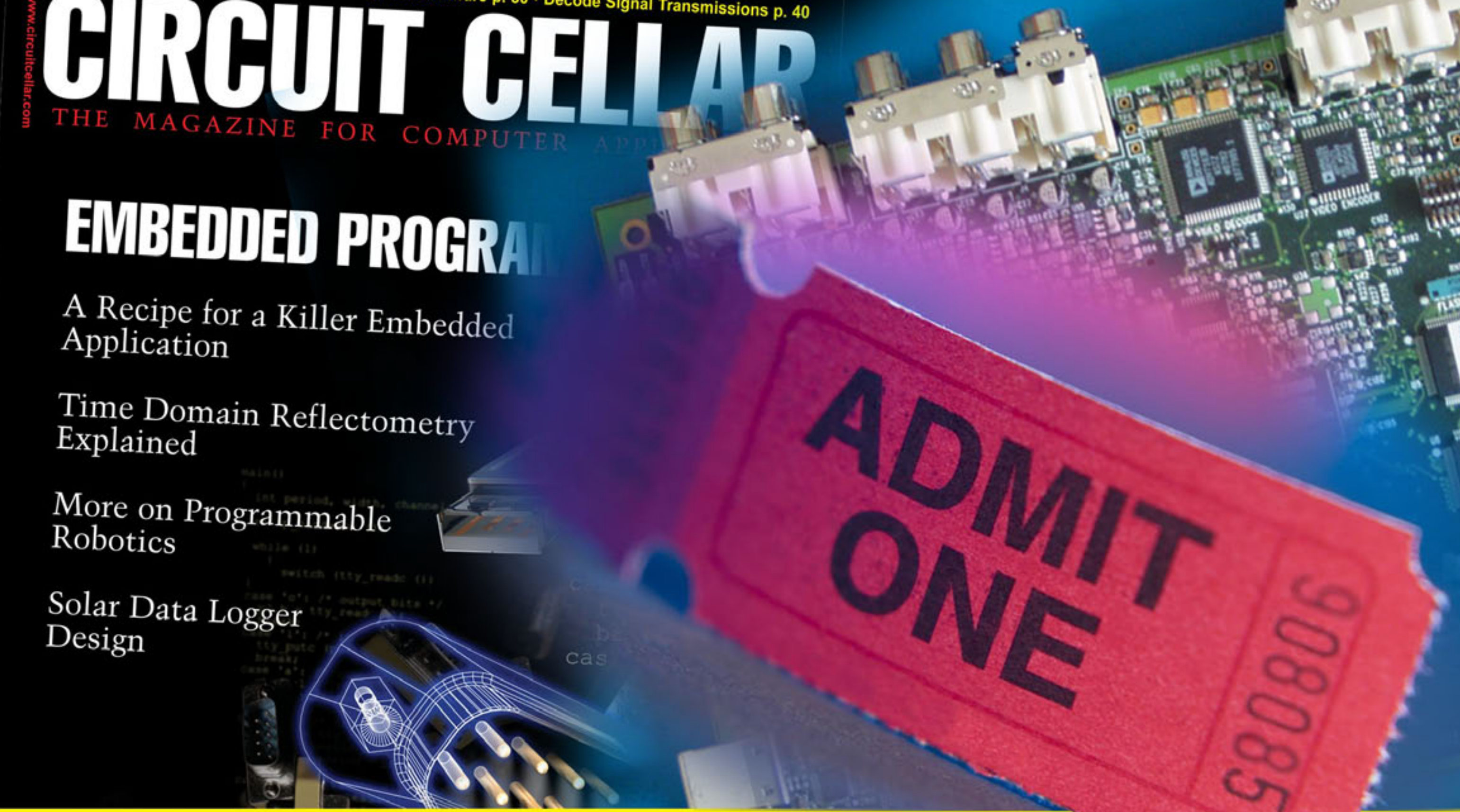
Create Mobile programmable robot
iRobot | www.irobot.com

dsPIC30F4011 DSC
Microchip Technology, Inc. | www.microchip.com

Robotics Developer Studio
Microsoft Corp. | http://msdn.microsoft.com/en-us/robotics/default.aspx

RealTerm Terminal emulation software
RealTerm | http://realterm.sourceforge.net

Liberty BASIC
Shoptalk Systems | www.libertybasic.com



Event: Circuit Cellar "Digital Plus" launch

Date: March 23, 2009 – April 30, 2009

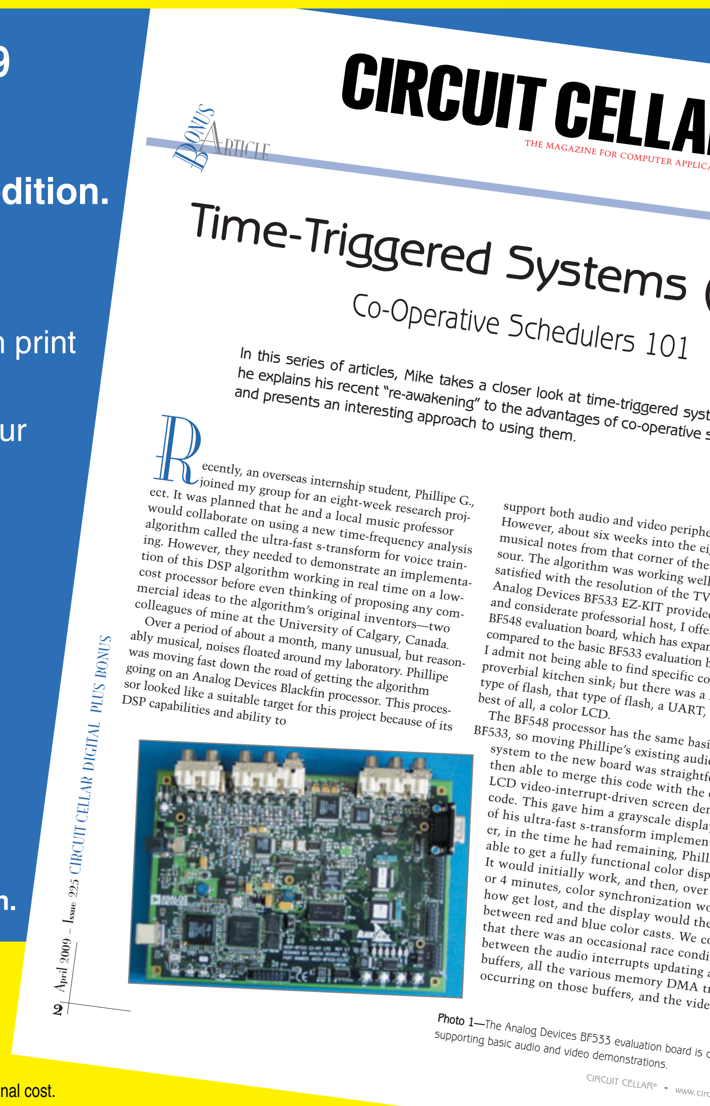
You hold in your hands a ticket to access Circuit Cellar magazine's new "Digital Plus" edition.

"Digital Plus" is your chance to:

- Read Circuit Cellar BONUS ARTICLES not available in print
- Enjoy audio/video-enhanced project articles
- Archive Circuit Cellar's print magazine as a PDF for your design library

Bring your friends to this limited-time public preview of Circuit Cellar's new venue. "Digital Plus" means more content, super-fast delivery of your favorite publication, and many additional features. As a current reader, you're entitled to special introductory pricing on "Digital Plus" subscription services.*

Act now to receive the April 2009 preview bonus material, which launches with "Time-Triggered Systems: Co-Operative Schedulers 101," by Circuit Cellar feature author Michael Smith.


CIRCUIT CELLA
THE MAGAZINE FOR COMPUTER APPLIC
BONUS ARTICLE

Time-Triggered Systems (

Co-Operative Schedulers 101

In this series of articles, Mike takes a closer look at time-triggered syst he explains his recent "re-awakening" to the advantages of co-operative s and presents an interesting approach to using them.

Recently, an overseas internship student, Phillipe G., joined my group for an eight-week research project. It was planned that he and a local music professor would collaborate on using a new time-frequency analysis algorithm called the ultra-fast s-transform for voice training. However, they needed to demonstrate an implementation of this DSP algorithm working in real time on a low-cost processor before even thinking of proposing any commercial ideas to the algorithm's original inventors—two colleagues of mine at the University of Calgary, Canada.

Over a period of about a month, many unusual, but reasonably musical, noises floated around my laboratory. Phillipe was moving fast down the road of getting the algorithm going on an Analog Devices Blackfin processor. This processor looked like a suitable target for this project because of its DSP capabilities and ability to

support both audio and video periphe However, about six weeks into the ei musical notes from that corner of the sour. The algorithm was working well satisfied with the resolution of the TV Analog Devices BF533 EZ-KIT provide and considerate professorial host, I offe BF548 evaluation board, which has expa compared to the basic BF533 evaluation I admit not being able to find specific co proverbial kitchen sink; but there was a type of flash, that type of flash, a UART, best of all, a color LCD.

The BF548 processor has the same basi BF533, so moving Phillipe's existing audi system to the new board was straightfo then able to merge this code with the LCD video-interrupt-driven screen de code. This gave him a grayscale displa of his ultra-fast s-transform implemen er, in the time he had remaining, Phill able to get a fully functional color disp It would initially work, and then, over or 4 minutes, color synchronization wo how get lost, and the display would the between red and blue color casts. We c that there was an occasional race condi between the audio interrupts updating buffers, all the various memory DMA t occurring on those buffers, and the vide

Photo 1—The Analog Devices BF533 evaluation board is c supporting basic audio and video demonstrations.

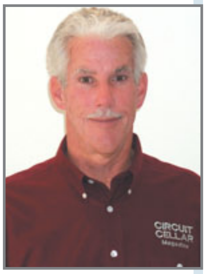
2 April 2009 – Issue 225 CIRCUIT CELLAR DIGITAL PLUS BONUS

CIRCUIT CELLAR® • www.circ

To access the preview edition, visit www.circuitcellar.com and click on the April issue icon on our home page.

*Active Electronic Edition subscriptions have been automatically replaced with Digital Plus subscriptions at no additional cost.

by Tom Cantrell



ZSTAR Trek

A Healthy Mix of MCUs, Sensors, and Wireless Technology

With a combination of MCUs, sensors, and wireless capabilities, the ZSTAR3 evaluation kit could be a great starting point for your next project. In addition to a wireless accelerometer, it includes a USB plug-in wireless hub and handy PC-based utility software.

What are three of the tastiest ingredients in an embedded designer's pantry?

The first is that modern-age miracle-worker, the MCU, packing the intelligence needed to give formerly ho-hum applications a silicon enhancement. Thanks to the fact Moore's law continues to deliver on the promise of more for less, ever better MCUs are finding their way into practically any gadget with moving electrons.

On another shelf we find ever-smarter sensors that give those fancy MCUs some real-world data to chew on. After all, it's the sensors that single-handedly enable, or not, potential applications. It's game over if you can't sense, because then you can't control (at least very well).

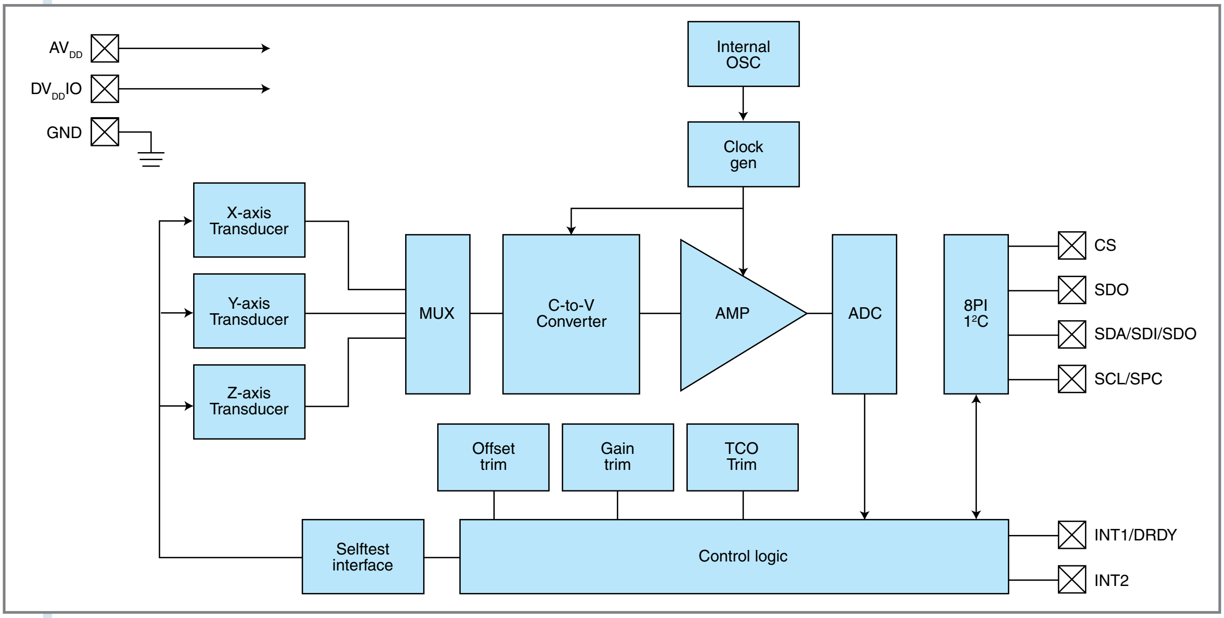


Figure 1—Add some e-motion to your designs with a Freescale Semiconductor MMA7456L smart, three-axis, low-g MEMS accelerometer.

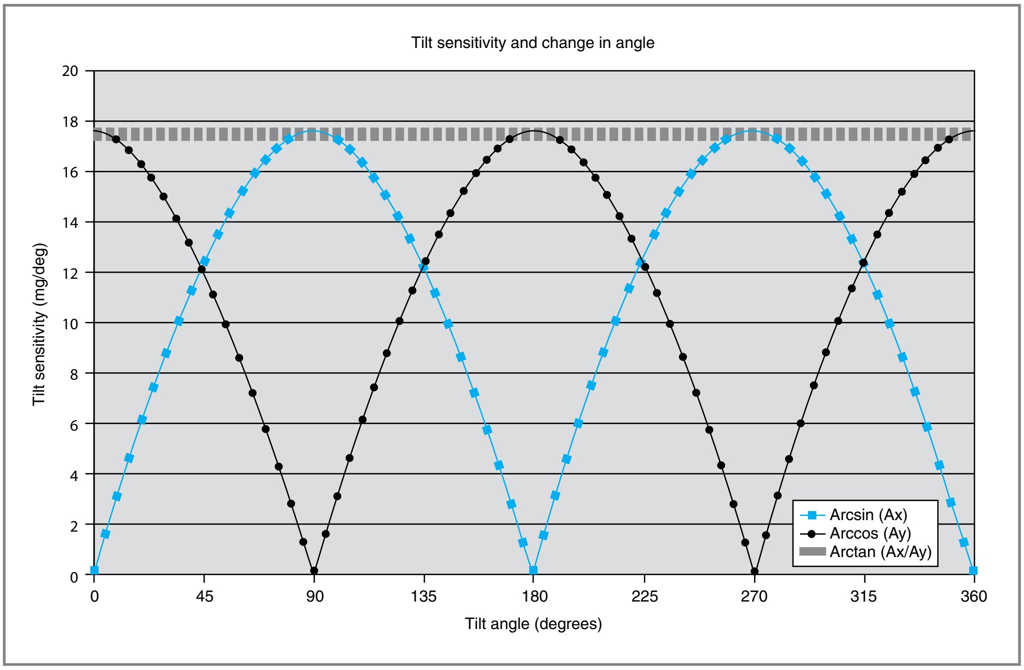


Figure 2—Measuring tilt with a single-axis accelerometer is complicated by the fact that sensitivity varies with the tilt angle, and is poor at the extreme. Using a second axis, and a bit of trigonometry, sensitivity is boosted and linearized across the tilt angle range.[1]

Thanks to these high-integration, easy-to-use MCUs and sensors, wiring up a new design is simple. It's even simpler if you skip the wiring part altogether by taking advantage of low-cost radio chips that are proliferating like bunnies.

Combine all three ingredients and you can cook up an endless variety of innovative embedded applications. This month, let's take a look at an excellent example of the trend courtesy of Freescale Semiconductor. Read on and I think you'll agree that mixing MCUs, sensors, and wireless together, seasoned with a healthy dash of designer creativity, is the recipe for success.

XYZ

"Just ask the Axis, He knows everything..."
—The Jimi Hendrix Experience, "Bold as Love," *Axis: Bold as Love*, MCA Records, 1968.

There is no better example of "It's the sensors, stupid" than the MEMS accelerometer. From their humble high-g airbag roots, low-g accelerometers have emerged to single-handedly enable vast new applications (i.e., Nintendo Wii), as well as, pardon the pun, shaken up existing ones (i.e., Apple iPod Nano "shake to shuffle"). Indeed, low-g accelerometers have come full circle, now back on the road in active suspension systems reunited with their high-g airbag ancestors.

I got my feet wet with a first-generation low-g MEMS accelerometer way back in 1995 ("A Saab Story: A Tale of Speed and Acceleration," *Circuit Cellar* 57). Since then, it's been the usual silicon story with the newest-generation parts, such as the Freescale MMA7456L (see Figure 1), offering better specs, more features, and, above all, a much lower price (just \$2.87 in 1,000-unit quantities).

In the beginning, accelerometers measured a single axis (X), followed later with dual-axis (XY) versions. It is not surprising that the latest and greatest like the MMA7456L have upped the ante with full three-axis (XYZ) capability. Remember that even applications that can get by with a single- or dual-axis part can take advantage of an extra axis to enhance features, reliability, and ease of use.

For instance, in "A Saab Story," I described how I used the accelerometer as the basis for a time/speed/distance display (i.e., using time and acceleration to calculate speed and distance). But the gadget would work only on a reasonably level road, lest a change in the Z-axis orientation be falsely interpreted as acceleration in the X-axis. Traveling at a fixed speed, going up hill would appear faster, and downhill slower, than on a level road. With a Z-axis, I could have dynamically compensated the speed calculations based on the pitch (i.e., heading

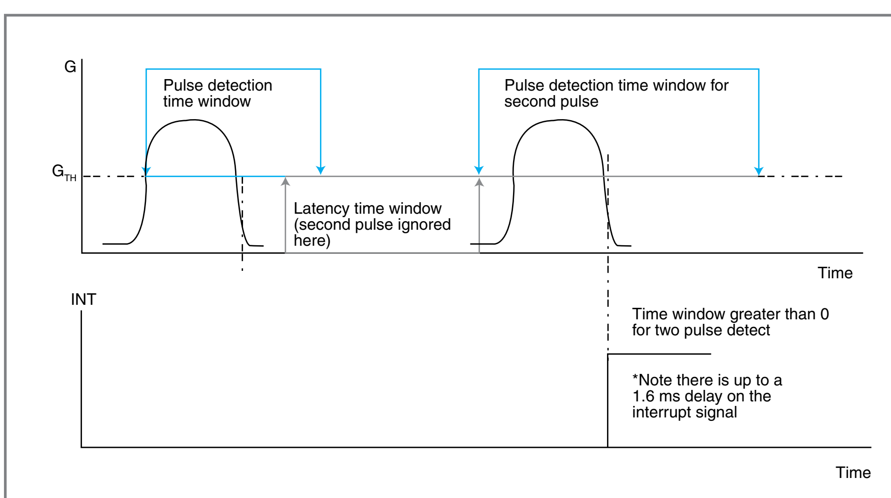


Figure 3—Configuring the MMA7456L single- and double-"shake" (i.e., pulse) detection feature is simply a matter of setting up registers that define the threshold and timing.

up or down hill) information.

Another more-is-better situation is using an accelerometer as an inclinometer to measure tilt, as in the "electronic levels" you'll find in the tool department these days. Sure, you can measure tilt with a single-axis accelerometer, but the problem is that the g-output is quite nonlinear over the range of 0° to 90° (see Figure 2). Getting a feel for this phenomenon is as easy as dropping and giving me 10 push-ups. (Consult your doctor first.) Now stand up and lean against a wall at a slight angle and give me 10 more. A lot easier, huh? By adding an extra axis, you can use whichever sensor is in its sweet spot for better resolution across the full range and especially near the extremes (i.e., 0° and 90°).

The accelerometer I used back in 1995 had an analog output, albeit thankfully signal-conditioned to provide a decently high-level signal. By contrast, the MMA7456L is fully digital with on-board 10-bit ADCs and a SPI. In my book, digital is generally preferred because it minimizes susceptibility to noise and enables "smart" features. However, there may be scenarios where an analog version still makes sense. An obvious one is when you're connecting to an MCU that has unused ADC inputs. Some applications may require higher resolution and are willing to trade-off bandwidth and signal processing MIPS to get it. Also, an analog part can work stand-alone in hardwired applications that don't need an MCU. Letting you have it your way, Freescale also offers the MMA7361L, a part that's similar to the MMA7456L except it has analog outputs with a healthy 800-mV/g swing.

But it's the digital smarts (offset correction and programmable threshold, motion, freefall, and single-/dual-pulse detection) that set the MMA7456L apart. The pulse detection feature is particularly useful for user interface applications as the shaken-not-stirred equivalent of single- and double-click on a mouse. There is a set of registers to adjust the pulse threshold and timing like you would with the mouse control panel on your PC (see Figure 3). In applications that need it, pulse detection is a big time (and power) saver for the MCU, which would otherwise be burdened having to constantly be on the lookout for properly formed pulses.

Low-power is all the rage and the MMA7456L obliges by sipping a mere 0.5 mA during normal operation. The chip has separate digital and analog power rails, but you can run both from a single supply anywhere between 2.4 and 3.6 V. Thanks to the low power and wide voltage range, powering the MMA7456L from an MCU output pin is a viable option. There's also a standby mode that slashes power consumption to

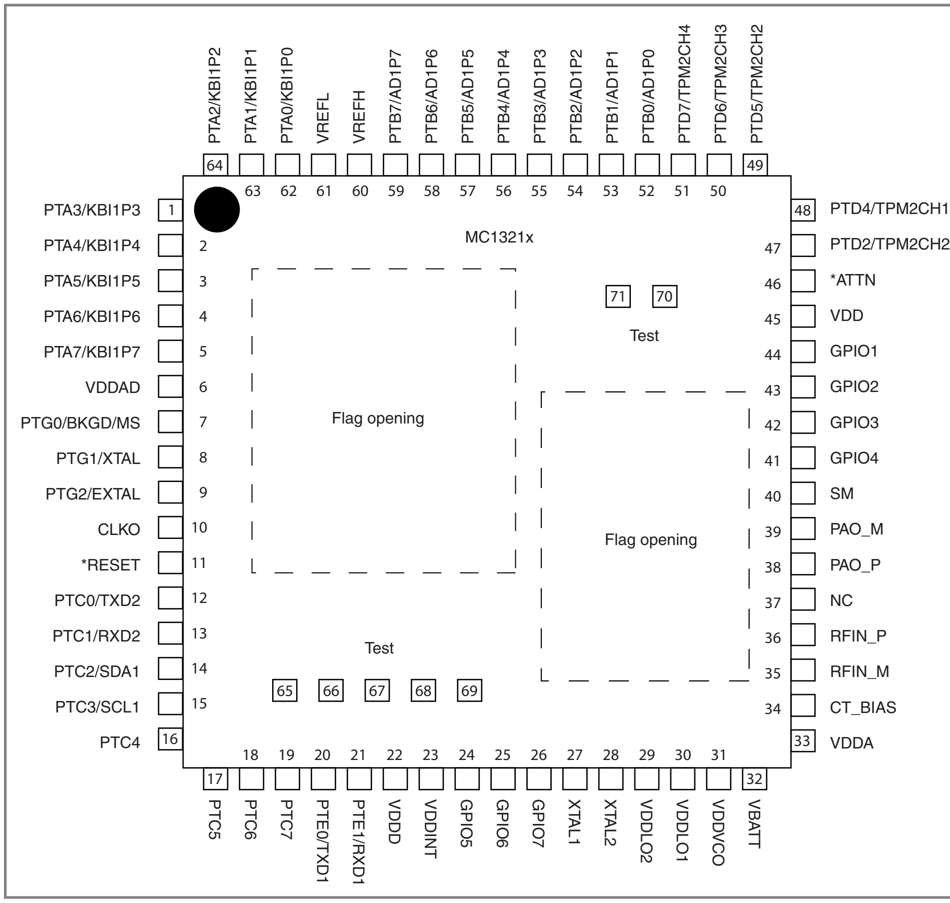


Figure 4—The two-die "System-in-Package" MC13213 looks like a typical '508 MCU, just one that happens to have a complete 2.4-GHz radio built-in.

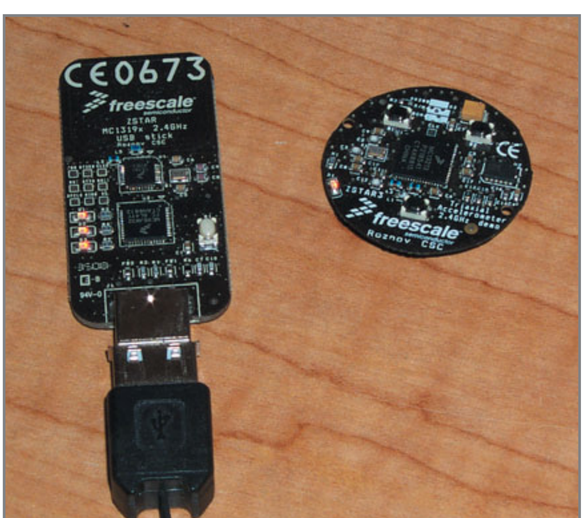


Photo 1—The ZSTAR3 evaluation kit provides a quick and easy way to taste-test Freescale's wireless sensor recipe, and the price is right at just $99. The network supports up to 16 sensors. Additional sensor boards (using the digital MMA7456L or analog MMA7361L) are available for $59.

just a few microamps. Do keep in mind there's a bit of latency entering and exiting standby (up to 20 ms each way).

ON THE AIR

With a sensor in hand, now we need an MCU and a radio to pull it all together. Freescale makes that especially easy with their MC1321*x*, which combines an 'S08 flash memory MCU with their 802.15.4-compatible 2.4-GHz radio on a single chip (see Figure 4).

The MC1321*x* line-up comprises three parts that differ only in the MCU's flash memory/RAM capacity with the MC13211 offering 16/1 KB, the MC13212 32/2 KB, and the MC13213 at 60/4 KB. Besides addressing applications of different scope and complexity, each part is semi-tailored to fit three different wireless protocols.

At the low end, Freescale offers their own "Simple MAC" (SMAC) solution. Supporting basic point-to-point and star networks, SMAC easily fits in an MC13211. Stepping up a notch is a full IEEE 802.15.4 MAC that supports all that standard's features, such as fancy topologies (e.g., mesh and tree) and bulletproof security (128-bit AES encryption). Finally, at the top of the stack so to speak, is Freescale's "BeeStack" fully ZigBee-2006-compliant platform. Note that the SMAC source code is available for adding your own proprietary protocol tweaks while 802.15.4 and ZigBee are delivered only as object-code.

The MCU and radio work well together. For example, because the radio needs a precise clock, it uses a 16-MHz crystal and a trimmable oscillator. In turn, the radio's clock can be output to a pin for use as the MCU clock, so only a single crystal is required.

Besides flash memory and RAM, the protocol software consumes some of the MCU I/O resources (e.g., using some of the timer channels for scheduling radio activity). That leaves plenty of MCU I/O resources (e.g., SPI, I²C, UART, 8 × 10-bit ADC, GPIO, and more) free for your application.

Just add a 16-MHz crystal, a single voltage (thanks to on-board regulators) 2- to 3.4-V power supply, an antenna, and a few discretes, and you've got a complete wireless gadget (aka "Wadget"). What could be easier?

WELL ROUNDED

What could be easier indeed? How about just heading over to the Freescale web site and ordering the ZSTAR3 evaluation kit (see Photo 1). In addition to the wireless accelerometer (versions are available with the digital MMA7456L or the analog MMA7361L), the kit includes a USB plug-in wireless hub and some cool PC-based utility software.

The ZSTAR3 wireless scheme supports up to 16 sensors simultaneously, each delivering data at 30 Hz. The protocol is based on the aforementioned SMAC upgraded with some new timing-related features for efficient scheduling (see Figure 5).

I experimented with the original

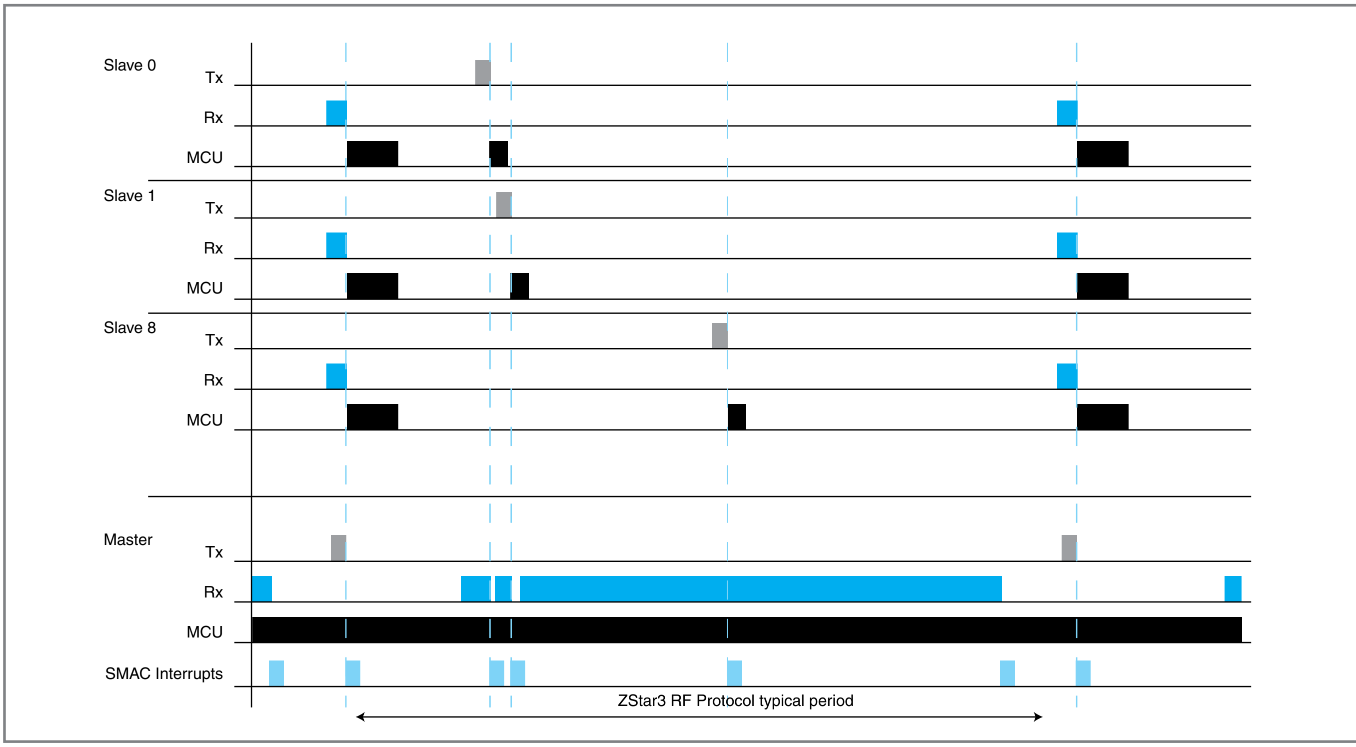


Figure 5—One advantage of centralized (i.e., hub) network control is the ability to schedule activity efficiently. Each ZSTAR3 node schedules radio activity (i.e., transmit/receive "windows"), which avoids interference and steadies data throughput. Scheduling also minimizes superfluous radio activity and allows the node to sleep most of the time, extending battery life.

version of the ZSTAR software back in 2006 ("Three-Axis Foursome," *Circuit Cellar* 191). It was fine for getting up and running and playing around a bit, but certainly nothing to write home about. But it's apparent that the Freescale folks have been hacking their hearts out in the interim because the ZSTAR3 software has a lot of new features to exercise the hardware and show off applications ideas.

Not that the version "0.2.3.0" of the software didn't come with some quirks and head scratching. I had to poke at it a bit to get the sensor linked in and bumped over some dubious error-message potholes from time to time, but nothing that was a showstopper.

The splash screen aggregates the 16-sensor display with tabs at the top to exercise various features (see Photo 2). The "RF Overview and Control" tab enables you to manually or automatically select the RF channel frequency and shows the energy in each band. A dynamic RSSI display is useful for evaluating signal strength and range, which are well-known challenges for 2.4-GHz radios with teensy PCB antennas (see Photo 3).

"General Sensor Tasks" start with calibration (i.e., zeroing out XYZ offsets while the sensor is motionless) to compensate for variations, such as mechanical misalignment and temperature drift. Naturally, there's a "Scope" display that traces acceleration in real time, and also offers the

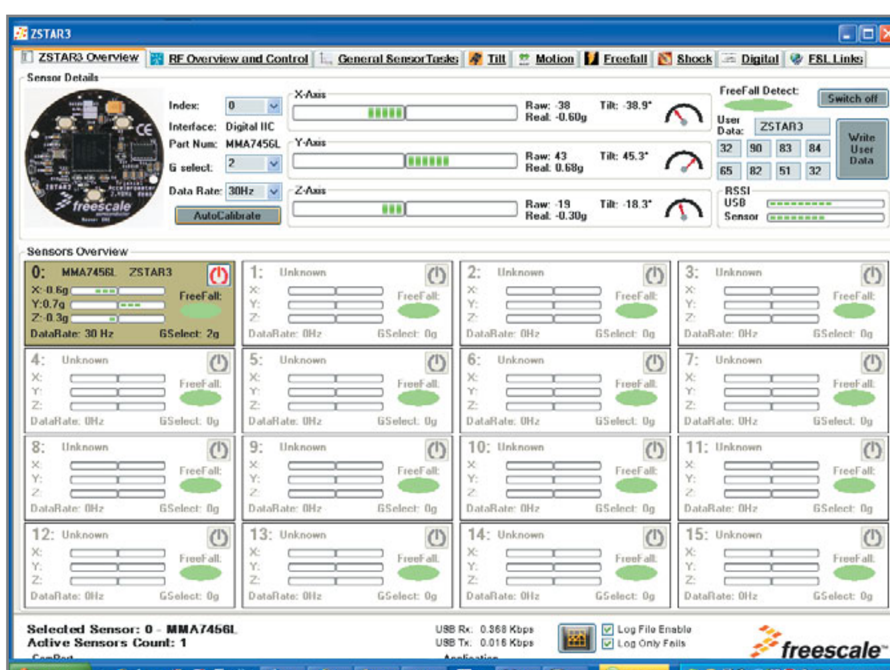


Photo 2—The ZSTAR3 evaluation kit's PC GUI software provides a complete set of MMA7456L hardware utilities and demo applications.

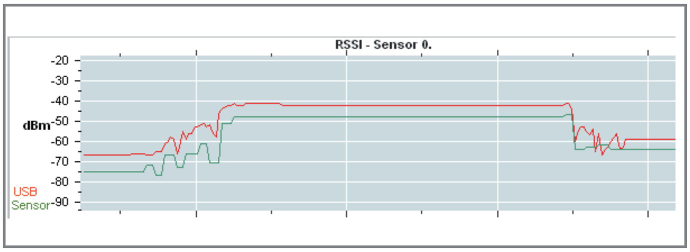


Photo 3—Range for low-cost 2.4-GHz radios using passive PCB trace antennas can be limited. Even the device's packaging (e.g., batteries) can block the signal. The ZSTAR3 RSSI utility makes it easy to monitor signal strength in real time and experiment with installation-specific alternatives.

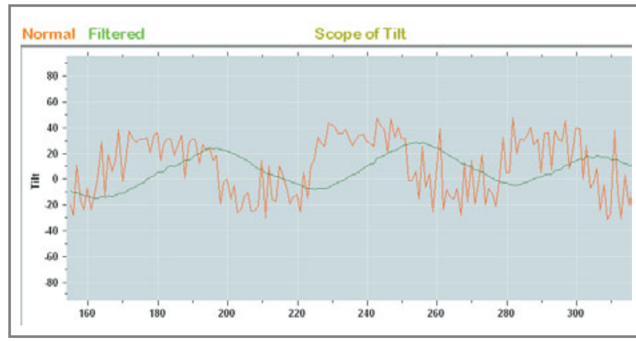


Photo 4—A little signal processing goes a long way to filter noise, as demonstrated by this "electronic level" demo.

ability to log data to an Excel file.

The "Tilt" tab demonstrates the technology at work in demo applications, such as PDA scrolling or switching a display automatically between portrait and landscape modes. There's a "Filtered Tilt" demo that highlights the ability of some simple signal processing to boost accuracy (see Photo 4). Using raw accelerometer data there's a noticeable jitter in the tilt reading on the order of 2° to 3°. But all it takes is a little bit of filtering (i.e., moving average) to cut the jitter to a fraction of a degree.

Likewise, the "Motion" tab demonstrates how you can use the threshold feature to give applications longer battery life (i.e., using motion as an on-off switch) and antitheft security. For instance, office equipment could be designed to sound an alarm if it's moved in a suspicious way or at an unusual time.

The "Freefall" tab exercises a feature that could prove useful in a variety of reliability, safety, and health applications. A neat feature is the software takes a decent stab at determining the distance of the fall based on the elapsed time (see Photo 5). Before you get carried away, keep in mind the 5,000-g maximum spec, which is less than it sounds (the "Drop Test" spec is 1.8 m beyond which permanent damage may occur). The demo works fine if you drop the gadget onto something soft, all the better to be safe than sorry.

Similarly, the "Shock" tab has demos of interest in blue-collar applications such as shipping and handling. For example, how many packages have you received with a "This Side Up" label? Wouldn't it be interesting to actually know if (and when, and where) it got tipped upside down along the way? Finally, the "Digital" tab pops the hood on the chip so you can probe and configure all of the accelerometers, registers, and options.

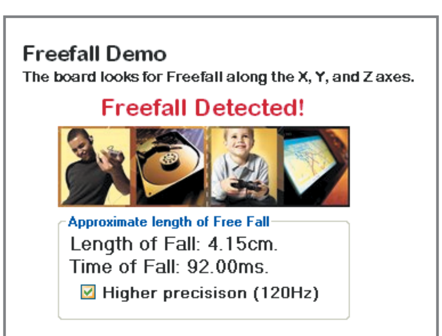


Photo 5—Today, the "I've fallen, and I can't get up" lady has to tell someone she's in trouble verbally. Err, kind of hard to do if you're unconscious. Tomorrow, an electronic fall detector will make the 911 call for her.

ONE FROM COLUMN A …

So there you have it, the recipe for your next killer app. Like the menu in a Chinese restaurant, all you have to do is choose one item from column A, one from column B, and one from column C.

Column A is an MCU. There are so many delicious choices here you can't go wrong, everything from 50-cent, 8-bit appetizers to MIPS-laden 32-bit entrees.

Column B is the radio. Here you have a choice of standard favorites like ZigBee, Wi-Fi, and IEEE 802.15.4, along with a variety of boutique alternatives. Options cover the spectrum from low-speed point-to-point "wire-replacement" to emerging "smart dust" mesh networks with hundreds or thousands of nodes. You can spice up your wireless offering with cellular broadband, RFID, or GPS for a new and improved taste.

Finish off with dessert, always my favorite part of the meal, by choosing a sensor from Column C. Or, for that matter, why not indulge with two or three of them. Thanks to the march of silicon, your designs can be both less filling and taste great.

Tom Cantrell has been working on chip, board, and systems design and marketing for several years. You may reach him by e-mail at tom.cantrell@circuitcellar.com.

REFERENCE

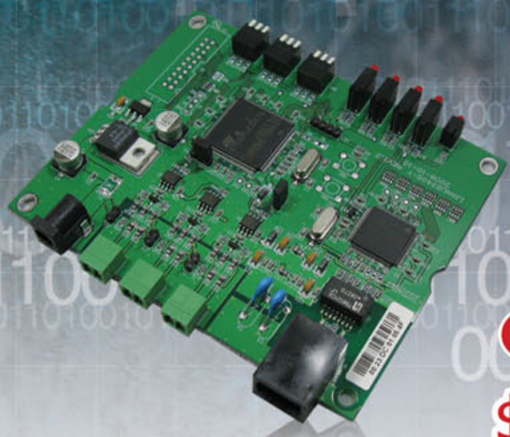
[1] K. Tuck, "AN3461: Tilt Sensing Using Linear Accelerometers," Freescale Semiconductor, Inc., 2007.

SOURCE

MC13213 MCU and IEEE 802.15.4 Radio, MMA7361L accelerometer, MMA7456L accelerometer, and ZSTAR3 accelerometer evaluation kit
Freescale Semiconductor, Inc. | www.freescale.com

ICbank
Gateway to Electronics

3 PORT INTERFACE
RS-485 to Ethernet Converter



Only
$170

RS-485 to Ethernet Converter

Powerful feature

- Protocol converter RS485 between Ethernet
- Offer TCP/IP Communication to Devices with RS485 I/F

Specification

Network	: TCP, UDP, DHCP, ICMP, IPv4, ARP, IGMP, PPPoE, Ethernet, Auto MDI/MDIX , 10/100 Base-TX Auto negotiation (Full/half Duplex)
Serial	: RS485 3 Ports, 1,200~115,200 bps, Terminal block I/F Type
Control program	: IP Address & port setting, serial condition configuration, Data transmit Monitoring
Accessory	: Power adapter 9V 1500mA, LAN cable
Etc	: - DIP Switch(485 Baud Rate setting) - LED: Power, Network, 485 Port transmission signal

ICbank
Gateway to Electronics

www.icbank.com

ICbank
Gateway to Electronics

MP3P DIY KIT, Do it yourself

(Include Firmware Full source Code, Schematic)

• myPIC

Only
$160 qty 100 **$220** qty 1

Powerful feature

- MP3 Encoding, Real time decoding (320Kbps)
- Free charge MPLAB C-Compiler student-edition apply
- Spectrum Analyzer
- Application: Focusing for evaluation based on PIC
- Offer full source code, schematic

Specification
Microchip dsPIC33FJ256GP710 / 16-bit, 40MIPs DSC
VLSI Solution VS1033 MP3 CODEC
NXP UDA1330 Stereo Audio DAC
Texas Instrument TPA6110A2 Headphone Amp(150mW)
320x240 TFT LCD
Touch screen
SD/SDHC/MMC Card
External extension port (UART, SPI, I2C, I2S)

• myWave (MP3 DIY KIT SD card Interface)

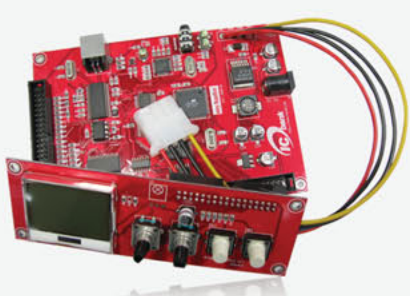
Only
$150 qty 100
$200 qty 1

Powerful feature

- Play, MP3 Information, Reward, forward, Vol+/-
- Focusing for MP3 Player
- SD Card interface
- Power: battery
- offer full source code, schematic

Item	Specification
MCU	Atmel ATmega128L
MP3 Decoder	VS1002 / VS1003(WMA)
IDE Interface	Standard IDE type HDD(2.5", 3.5")
Power	12V, 1.5A
LCD	128 x 64 Graphic LCD
Etc	Firmware download/update with AVR ISP connector

• myAudio (MP3 DIY KIT IDE)



Only
$180 qty 100
$220 qty 1

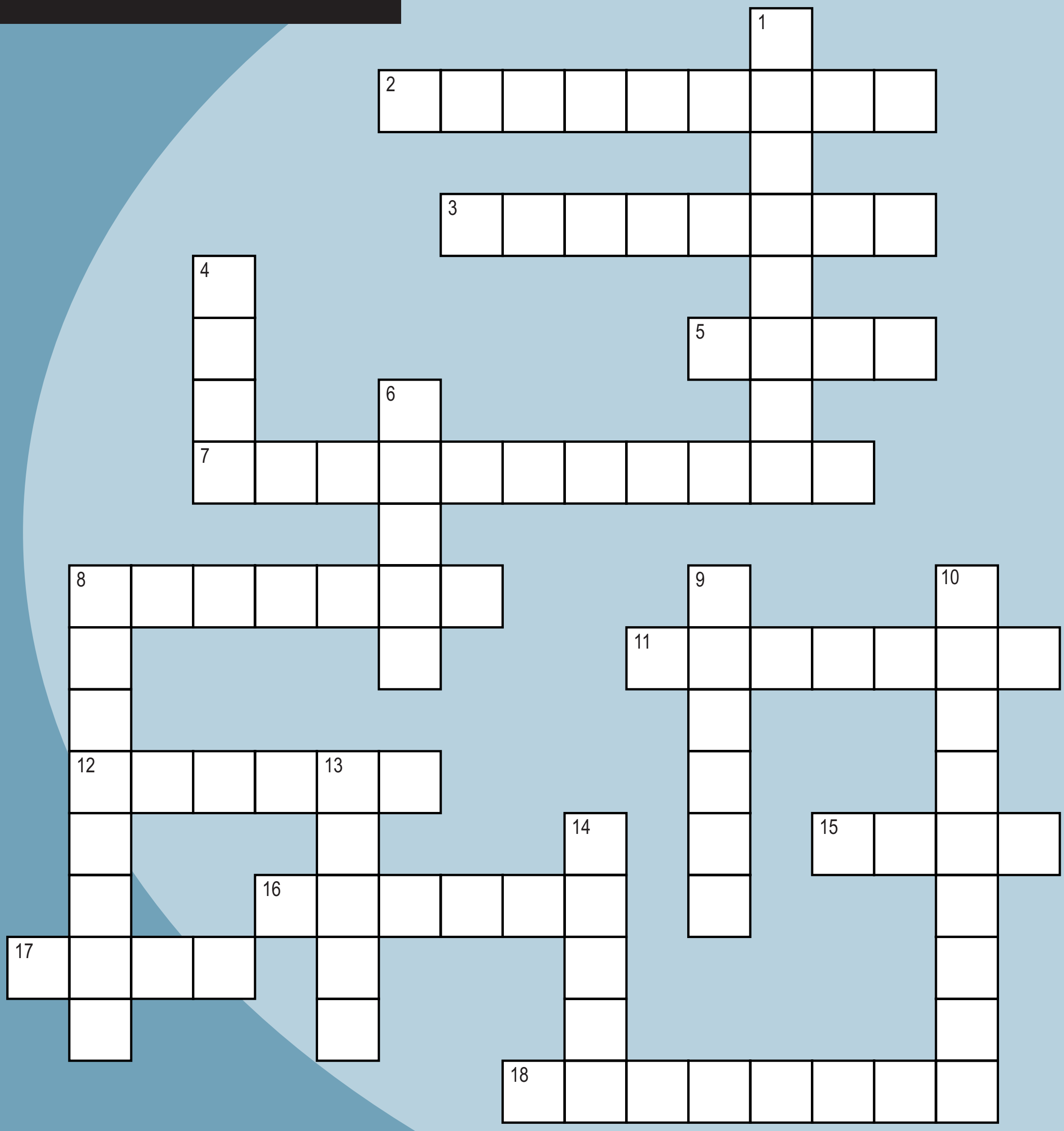
Powerful feature

- Play, MP3 Information, Reward, forward, Vol+/-
- Focusing for full MP3 Player (Without case)
- IDE Interface
- Power: Adapter
- Offer full source code, schematic

ICbank
Gateway to Electronics

www.icbank.com

CROSSWORD



Across

2. Bypass system security
3. Oil-drop experiment
5. Program of repeated commands
7. Sends electromagnetic signals
8. yy++ (notation)
11. Switch that protects damage
12. “S” in SETI
15. 10^{-18}
16. Dangles off for protection
17. 0.01 kg
18. p = mv

Down

1. Network’s nerve
4. One Joule per second
6. x + y (notation)
8. dF/dA
9. ++yy (notation)
10. 1,000,000 BTU
13. CLK
14. 10^{-15}

The answers are available at www.circuitcellar.com/crossword.

IDEA BOX

THE DIRECTORY OF PRODUCTS AND SERVICES

AD FORMAT: Advertisers must furnish digital submission sheet and digital files that meet the specifications on the digital submission sheet. **ALL TEXT AND OTHER ELEMENTS MUST FIT WITHIN A 2″ x 3″ FORMAT**. Call for current rate and deadline information. E-mail adcopy@circuitcellar.com with your file and digital submission or send it to IDEA BOX, Circuit Cellar, 4 Park Street, Vernon, CT 06066. For more information call Shannon Barraclough at (860) 875-2199.

The Vendor Directory at www.circuitcellar.com/vendor/
is your guide to a variety of engineering products and services.

INDEX OF ADVERTISERS

The Index of Advertisers with links to their web sites is located at www.circuitcellar.com under the current issue.

PREVIEW of May Issue 226

Theme: **Measurement & Sensors**

Battery Charge Measurement: A State-of-Charge Lead-Acid Battery Meter

Construct a USB GPIO Pod (Part 2): USB JTAG Module

Teach an Old DOS New Tricks: A Flash Drive Reader for MCUs Works for DOS

Transformerless Power Supply

LESSONS FROM THE TRENCHS **FAT File System Review (Part 2):** C Code for the File System

FROM THE BENCH **A World Without the NTSC:** NTSC-to-VGA Conversion Made Simple

SILICON UPDATE **Whistle While You Work:** A Look at a Modern DSP

PRIORITY INTERRUPT

by Steve Ciarcia, Founder and Editorial Director

Print Is Dead—Long Live Print

I'm one of those good guys who goes shopping with his wife and doesn't complain. Our trips include both mutual and independent destinations. She rarely accompanies me into RadioShack, and I wouldn't be caught dead in Joann Fabrics or Michael's. On those occasions, I don't pout or honk the horn after 10 minutes. Instead I tell her to take her time and I read magazines. The bad news is that the pile in the car keeps getting smaller.

Aside from 32-page trade journals and half-sized *Wired* and *Fortune* magazines, two biggies on my parking lot reading list, *US News* and *PC Magazine*, have gone digital. Should I take this as a sign that it is a brave new world or that I'm just the last of a dying breed?

A lot has been written lately about the reasons. Dan Costa, a columnist for *PC Magazine*, wrote an editorial titled "Print is Dead. Long Live Print" (12/11/08). He explained that ceasing the print magazine after 27 years was inevitable and necessary. Besides the economic pressures, he suggested that, because anyone can publish these days (true, but that doesn't mean it's quality article content), printed magazines are an environmental nightmare (and all these computers aren't, Dan?), that print-delayed news and product reviews can't compete with a real-time web (true, if that's what you are looking for), that traditional print publishing is dead.

The real epiphany in the editorial was his conclusion about the other side of the coin: "Print businesses aren't dead. They just need to change. Printing should be reserved for archival information—artifacts you'll hold onto for years instead of hours or days." Thank you, Dan Costa, and welcome to our world.

Unlike newspapers and trade journals, *Circuit Cellar* content is not time-critical and our readers aren't conventional. Rather than read *Circuit Cellar* and then give it to someone else, our readers hoard every issue and archive them for years. The chips may have evolved since an application was published in January 2001, but the tutorial example of the engineering thought process detailed in the article hasn't. Archiving past issues of *Circuit Cellar* into an engineering reference library is the norm, not the exception, for our readers.

I think of *Circuit Cellar* as a community, not just a business enterprise. While I'm completely aware that there is a significant cost savings in distributing a virtual magazine versus printed paper, I also know that *Circuit Cellar*'s status and credibility was built on print and it will be *over my cold dead* ... Sorry, I get carried away. Basically, we're caught between a rock and a hard place when it comes to content expansion. While other pubs have been biting the dust, *Circuit Cellar*'s overall circulation and popularity has increased. There has been a demand for more *Circuit Cellar* articles, not less.

Under the heading, "teaching an old dog new tricks," the undeniable path to cost-effective content expansion is a new and better digital presence that we call Circuit Cellar Digital Plus. Digital Plus replaces our Electronic Edition (EE) and adds a lot more. Like our former EE, Digital Plus is an exact online replica of the print magazine with additional full-length *Circuit Cellar*-quality content along with project "shorts" in a special bonus section. Our objective with Digital Plus is to provide more space for good authors and great applications without raising our subscription prices into the stratosphere or breaking the bank with other methods.

Digital Plus enhances the print magazine experience with special features that we can only do online. Articles in Digital Plus may include video demos and downloadable supplements. Even the ads can include "see before you buy" video demonstrations. The best part is that Digital Plus will be less complicated to view than most other online magazines. Of course, if you are still an EE loyalist, you won't even have to download a special program like Zinio to view it offline. Like our former electronic magazine, you can still simply download a PDF of the whole issue.

So what's the damage for all this? Not as bad as you think. For us, it's about serving the community and increasing our online presence without sacrificing print to do it. Other electronic magazines might immediately raise prices while giving less, but we plan to do it incrementally, if at all. For starters, the April issue of Circuit Cellar Digital Plus will be free to everyone—including print, EE subscribers, and lurkers—so you can see what I'm talking about. (We'll bump all paid digital subscriptions to make up the difference for any free months we offer.) The one-year price of the electronic edition subscription was $15. If you are a print subscriber, you could get it for $5 per year. Right now, you can extend either subscription or start a new Digital Plus subscription at those same prices until May 31, 2009. After that, the Digital Plus prices will be higher. And, before you e-mail me asking if print-only subscribers are now second-class citizens (i.e., no access to the bonus section), please note that we've fixed that too. Print subscribers can go to our web site and see all of the bonus sections posted for free, although on a delayed basis.

So, I admit to being an old dog learning some new tricks, but that doesn't mean I have to be radical about it. We're not dumping the old. We're just trying to create a happy medium by implementing something new. Go take a look at Circuit Cellar Digital Plus (available from our home page) and join our latest venture into tomorrow.

steve.ciarcia@circuitcellar.com

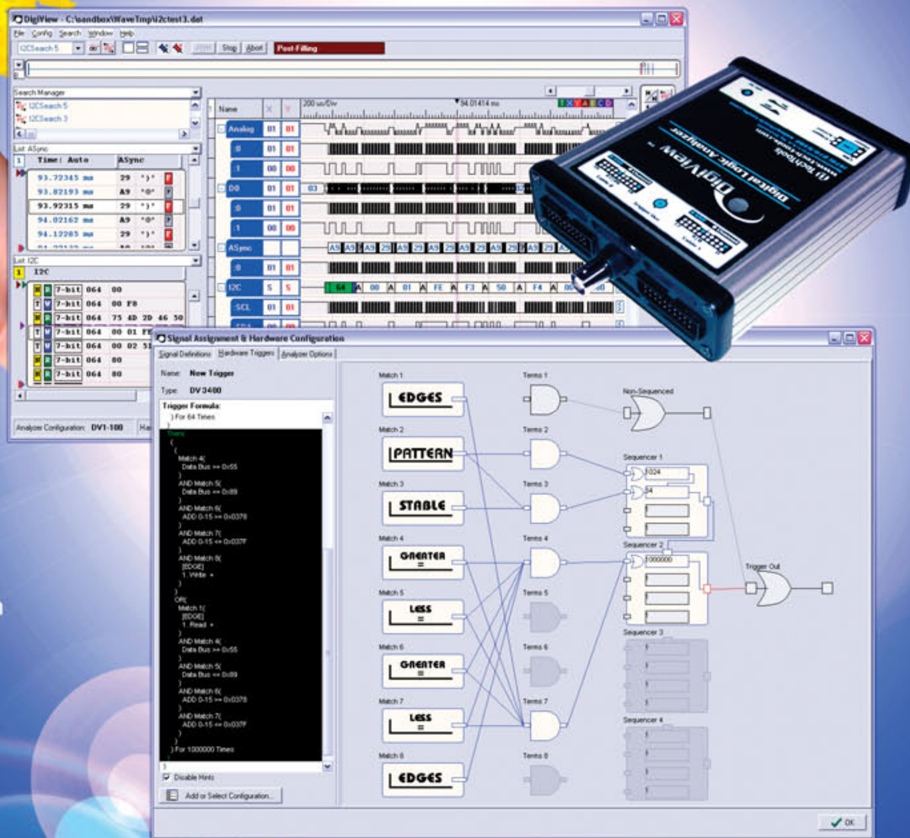
PC Based Logic Analyzers

NEW Model DV3400

- Faster Sampling (200/400 Msps)
- More Channels (36)
- More Memory (4x)
- Advanced Match circuits (8)
- ±6V Adjustable Threshold (x2)
- Cascadable Sequencers (4)
- Enhanced Compression

from $499.00

DigiView



Very flexible, "easy to configure" Advanced triggering.

Professional Hardware Capture + Software Analysis

- Automatic Real-time Hardware Compression eliminates the need to reduce resolution. Our newest version makes "dead time" insignificant.
- Edge and Pattern Triggers on all models. Advanced Model also includes Range (>, <, =, !=, >=, <=) and Stable Matches with Duration, and 4 flexible cascadable sequencers with pass-counters.
- Pattern Searches with Match & Duration.
- Specialized Sequential Searches for Serial and State Mode Signals.
- Display I²C, Synchronous (SPI), Asynchronous (RS-232), State, Boolean, Bus and Analog Data.
- Single or Dual Waveform Views.
- Resolution Zoom in Wave Views.
- Time based Link Groups for all views.
- Specialized Exports from Data Tables and List Views.
- Multi-Signal Data Tables.
- Drag & Snap Markers.
- "Click to Center" function.
- "Snap Previous/Next" function.
- Print or Save Images with comments.
- USB 2.0 (480 Mbps).
- USB 1.1 compatible (12 Mbps).

PICmicro® MCU Programmer

Multi-Function, In-Circuit & Gang Operation

only $199.00

QuickWriter™

See our Memory Emulators from $179.00

TechTools

www.tech-tools.com

(972) 272-9392 • sales@tech-tools.com

An entire community under one hat.

With eight 32-bit processors in one chip and deterministic control over the entire system, the Propeller microcontroller is just plain inspiring. ***With our free online community support and resources, progress is only a click away.***

http://obex.parallax.com

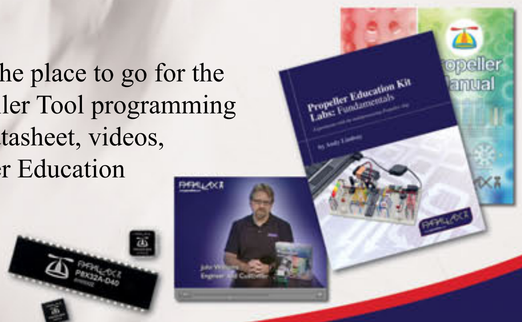
A virtual toy box for programmers, the **Propeller Object Exchange** contains source code objects in high-level Spin, Assembly, and C. Find objects created by both Parallax engineers and customers, for data storage, displays, motor control, communciation protocols, math functions, and more.

http://forums.parallax.com

For great conversations, visit the **Propeller Discussion Forum**. With 15,000 registered users, you may browse threads anonymously or sign up to join in the talk. It's a great place to get your questions answered by Parallax staff and other knowlegeable users, share your projects, and find inspiration.

http://www.parallax.com/propeller

Our main web site, **Parallax.com**, is the place to go for the latest downloads, including the Propeller Tool programming software, the Propeller Manual and datasheet, videos, and our latest education text "Propeller Education Kit Labs: Fundamentals."



Propeller web site: http://www.parallax.com/propeller
Discussion Forums: http://forums.parallax.com
Propeller Object Exchange: http://obex.parallax.com

PARALLAX INC

www.parallax.com

CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

BONUS ARTICLE

by Michael Smith

Time-Triggered Systems (Part 1)

Co-Operative Schedulers 101

In this series of articles, Mike takes a closer look at time-triggered systems. Here he explains his recent "re-awakening" to the advantages of co-operative schedulers and presents an interesting approach to using them.

Recently, an overseas internship student, Phillipe G., joined my group for an eight-week research project. It was planned that he and a local music professor would collaborate on using a new time-frequency analysis algorithm called the ultra-fast s-transform for voice training. However, they needed to demonstrate an implementation of this DSP algorithm working in real time on a low-cost processor before even thinking of proposing any commercial ideas to the algorithm's original inventors—two colleagues of mine at the University of Calgary, Canada.

Over a period of about a month, many unusual, but reasonably musical, noises floated around my laboratory. Phillipe was moving fast down the road of getting the algorithm going on an Analog Devices Blackfin processor. This processor looked like a suitable target for this project because of its DSP capabilities and ability to support both audio and video peripherals (see Photo 1). However, about six weeks into the eight-week project, the musical notes from that corner of the room started to turn sour. The algorithm was working well, but Phillipe was not satisfied with the resolution of the TV display that the Analog Devices BF533 EZ-KIT provided. Always the kind and considerate professorial host, I offered my visitor a BF548 evaluation board, which has expanded capabilities compared to the basic BF533 evaluation board (see Photo 2). I admit not being able to find specific connections for the proverbial kitchen sink; but there was a hard-drive, this type of flash, that type of flash, a UART, Ethernet, and, best of all, a color LCD.

The BF548 processor has the same basic core as the BF533, so moving Phillipe's existing audio analysis subsystem to the new board was straightforward. He was then able to merge this code with the existing BF548 LCD video-interrupt-driven screen demonstration code. This gave him a grayscale display of the results of his ultra-fast s-transform implementation. However, in the time he had remaining, Phillipe was never able to get a fully functional color display running. It would initially work, and then, over a period of 3 or 4 minutes, color synchronization would somehow get lost, and the display would then rotate between red and blue color casts. We concluded that there was an occasional race condition between the audio interrupts updating a number of buffers, all the various memory DMA transfers occurring on those buffers, and the video driver's

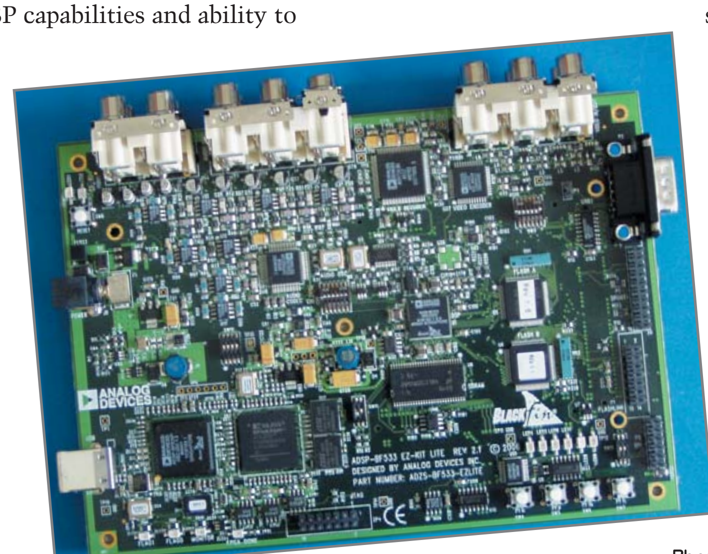


Photo 1—The Analog Devices BF533 evaluation board is capable of supporting basic audio and video demonstrations.

interrupts updating the screen based on the results from the transform calculations.

When Phillipe went back to Switzerland to complete his degree at the College of Engineering and Architecture of Fribourg, I was left with a problem. Is there a reliable and systematic way of merging together two complex audio and video algorithms into a single project when they both use code sequences involving preemptive interrupts?

This is a standard problem in an embedded environment found many times in industry. Airplane control systems successfully merge many DSP algorithms and associated external signals. After merging, all the subsystems reliably interact with each other, and remain responding to the external signals in a fast and reliable manner that ensures that the plane does not fall from the sky. Surely, there must be a systematic way to do get the same reliability with a far less complex system, right?

WHAT ARE THE CHOICES?

When merging two working subsystems together into a single system, the developer has many choices: the super-loop; the preemptive scheduler, which is what was effectively used in the DSP project I described in the introduction; and the co-operative scheduler. To see the differences between these three ideas, imagine the merging of two simple subsystem examples. One subsystem is responsible for input using the processor's general-purpose input/output (GPIO) pins to read four switches. The other subsystem outputs signals via a series of six LEDs connected to a parallel port that is part of the system's flash memory chip.

With a super-loop (see Listing 1), the two GPIO and LED interfaces are first initialized using `GPIO_Init()` (Line 13) and `LED_Init()` (Line 14). Then, in a forever loop, the GPIO interface pins are read (`GPIO_Read()`, Line 17) and transferred to the LED output (`LED_Write()`, Line 18). This sort of coding pattern is totally adequate in many embedded applications where the system requires only basic functionality, and no timing issues are present.

Imagine that we are now requesting something more from the system (e.g., the ability to add the audio and video capability of the system in the introduction while doing other basic tasks). However, rather than trying to tackle an example that complex, let's replace the audio and video tasks by subsystems with simpler timing requirements. Let's have second subsystem flash LED 4 ten times per second and LED 5 at 20 times per second, while at the same time the first subsystem continues to echo the switch input to the other LEDs. Even with this simpler example, the super loop (see Listing 2), starts getting messy to code and become more difficult to maintain as the complexity of the system functionality increases.

Photo 2—The BF548 evaluation board has significantly more capabilities than the BF533 evaluation board with many peripheral devices, including a colored touch screen.

In addition, if this code was running on a hand-held device, we are wasting a lot of battery power. There simply is no need to spend processor cycles to read the GPIO switches every time around the loop. We know that it is not humanly impossible to make changes to those switches at the speed at which the processor operates.

The power problem can be solved by sending the processor to sleep with an `idle()` instruction when it is not doing anything useful. Refer to lines 82 to 84 of `main()` in Listing 3. For this to happen, I needed to code up a system where a timer generates interrupts at 0.05-s intervals to cause the LEDs to flash (Lines 99 to 107) and enable the GPIO interface to generate interrupts whenever a switch is changed (Line 89–94)).

Such an "interrupt-driven" system works well, but there is one proviso for totally predictable operation in a safety-critical environment: the developer must ensure that no new interrupt can ever arrive during another interrupt's service routine. In reality, this condition can't be met even in this simple case we are looking at here! There is no guarantee that a switch will not be pressed just as the timer interrupt occurs. In such a preemptive system, there is always the possibility for one task to be "paused" while a higher-priority task is allowed to complete or to start. I agree that the time jitter associated with the delay of when the LEDs will be flashed is going to be small, and probably totally

Listing 1—If all the embedded system has to do is something basic—such as echo switch input values to an LED interface (Lines 17–18)—then a basic super-loop programming format is all you need.

```
12  int main(void) {
13    GPIO_Init( );            // Initialize the GPIO input sub-system
14     LED_Init( );            // Initialize the LED output sub-system
15
16    while (1) {              // Enter the 'Superloop'
17      int value = GPIO_Read( ); // Read the input (GPIO) switch values
18      LED_Write(value);      // Echo the switches to the LEDs
19    }
20
21    return 0;                // Keep the compiler from complaining
22  }
```

Listing 2—With each task that needs to occur at a predetermined time, the super-loop code gets messier to develop and maintain. In addition, the processor is always "on," a condition that would waste battery power in a hand-held application.

```
int main(void) {
  GPIO_Init( );                                   // Initialize the GPIO input sub-system
  LED_Init( );                                    // Initialize the LED output sub-system
  Timer_Init( );                                  // Provide ability to measure 'elapsed time'

  int timeLED4Flash = ElapsedTime( );             // Start measuring time intervals
  int timeLED5Flash = ElapsedTime( );             // used to control the flashing of LED 4 and 5

  while (1) {                                     // Enter the 'Superloop'
     int value = GPIO_Read( );                    // Read the input (GPIO) switch values
     LED_Write(value);                            // Echo the switches to the LEDs

        // Has sufficient time elapsed to cause LED #4 to be flashed
     if ( (ElapsedTime( ) - timeLED4Flash) >= ONE_TENTH_SECOND) {
        timeLED4Flash = ElapsedTime( );           // Update timing
        FlashLED4( );                             // and flash the LED
     }

        // Has sufficient time elapsed to cause LED #5 to be flashed
     if ( (ElapsedTime( ) - timeLED5Flash) >= ONE_TWENTIETH_SECOND) {
        timeLED5Flash = ElapsedTime( );           // Update the timing
        FlashLED5( );                             // and flash the LED
     }
  }

  return 0;                                       // Keep the compiler from complaining
}

```

Listing 3—With an interrupt-driven system, it now becomes possible to send the processor to sleep (Line 83), only responding when there is a specific task to do. However, the timing of critical events can no longer be guaranteed because one (human-driven) interrupt can arrive at the same time as an existing (timer-driven) interrupt service routine is being processed. One task is either blocked by the other task or must preempt it so that system predictability is lost.

```
72  int main(void) {
73      GPIO_Init( );                            // Initialize the GPIO input sub-system
74      LED_Init( );                             // Initialize the LED output sub-system
75      Timer_Init( );                           // Provide ability to measure 'elapsed time'
76
77      Activate_GPIO_Interrupts( );             // Activate GPIO interrupts to monitor switch operations
78
79                                               //Timer Interrupts to occur every twentieth of a second
80      Activate_TimerInterrupts(ONE_TWENTIETH_SECOND);
81
82      while (1)  {
83        idle( );                               // Send processor to sleep when not in ISR
84      }
85
86      return 0;                                // Keep the compiler from complaining
87  }
88
89  EX_INTERRUPT_HANDLER(GPIO_Interrupt) {       // ISR for when switch press occurs
90      Acknowledge_GPIOInterrupt( );
91
92      int value = GPIO_Read( );                // Read the input (GPIO) switch values
93      LED_Write(value);                        // Echo the switches to the LEDs
94  }
95
96
97  static volatile int numberInterrupts = 0;
98
99  EX_INTERRUPT_HANDLER(Timer_Interrupt) {      //  ISR to handle timer interrupts
100     Acknowledge_TimerInterrupt( );
101     numberInterrupts++;
102
103     FlashLED5( );                            // Flash LED 5 every 1/20th s interrupt
104     if ((numberInterrupts & 1) == 0) {
105        FlashLED4( );                         // LED 4 flashes every second interrupt
106     }
107 }
```

Listing 4—There are relatively few commands that need to be used to control a co-operative scheduler. The execution time of each event is controlled by delay and period parameters in the `TTCOS_AddTask()` call.

```
120 void TTCOS_AddTask(void (*TaskName)(void), int taskDelay, int taskPeriod);
121
122 int main(void) {
123     TTCOS_Init( );         // Initialize the co-operative scheduler 'to-do' list and timer
124
125         // Add  Tasks
126         // TTCOS_AddTask(FunctionPointer, taskDelay, taskPeriod);
127
128         //  The sub-system initialization tasks run with NO_DELAY and RUN_ONCE only
129     TTCOS_AddTask(GPIO_Init, NO_DELAY, RUN_ONCE);     // Initialize the GPIO input sub-system
130     TTCOS_AddTask(LED_Init,  NO_DELAY, RUN_ONCE);     // Initialize the LED output sub-system
131
132         // Specify delay and period of other tasks
133     TTCOS_AddTask(GPIO_LED_Echo,  NO_DELAY, ONE_TWENTIETH_SECOND);    // Task to echo switch input to LED
134     TTCOS_AddTask(FlashLED5, NO_DELAY, ONE_TWENTIETH_SECOND);  // Task to flash LED #5 every 1/ 20 s
135     TTCOS_AddTask(FlashLED4, NO_DELAY, ONE_TENTH_SECOND);        // Task to flash LED #4 every 1/ 10 s
136
137     TTCOS_Start( );        // Activate the co-operative scheduler's timer interrupt
138
139     while (1) {
140         TTCOS_DispatchTasks( );            // Run any tasks with a Run-Me-Now variable that is not zero
141         TTCOS_Sleep( );                    // and send the processor back to sleep until next timer interrupt
142     }
143
144     return 0;                              // Keep the compiler from complaining
145 }
146
147
148 EX_INTERRUPT_HANDLER(TTCOS_Interrupt) {    // Only ISR operative in the co-operative scheduler
149         Acknowledge_TTCOSInterrupt( );
150         TTCOS_Update( );                   // Update the Run-Me-Now variables for each task
151 }
```

irrelevant in this example. However, these simple tasks are intended as analogs of more complex audio and video algorithms that are triggered by interrupts, and we already suspect that the color de-synchronization issue discussed earlier is being caused by such an interrupt priority issue.

ENTER THE CO-OP SCHEDULER

At the November 2008 Embedded Systems Show in Birmingham, UK, I listened to a talk about time-triggered technology by Michael Pont of TTE Systems. In 2001, Pont published *Patterns for Time-Triggered Systems*, which is now a free download (www.tte-systems.com/books). I will admit that I stopped reading pretty early in that book because it was focused on the old 8051 processor—and it was 1,024 pages long. However, after listening to Pont's talk, I realized that I missed all the juicy bits after page 400, particularly the discussion about the advantages of the building and using a co-operative scheduler.

Refer to Listing 4. I reused the switches and flashing LED light example to demonstrate the basic manner to code using such a scheduler.

First, you initialize the system using `TTCOS_Init()` (Line 123). This function sets up an array of structures to use as the co-operative scheduler's "to-do list." It also prepares the TTCOS timer, which will be used to generate the only interrupt present in the system. Next, you add each new task to this "run-list" using a series of `TTCOS_AddTask()` calls (Lines 129 to 135).

The `TTCOS_AddTask()` function requires three parameters: a pointer to the task (function) you want to have run; the initial task delay describing when you want the task to run for the first time; and the task period detailing how often you want the task to run. Setting the period of the subsystem initialization tasks to `RUN_ONCE` (Lines 129, 130) causes those tasks to be deleted from the to-do list once they have been run.

The `TTCOS_Start()` function in Line 137 activates the TTCOS timer. Whenever the time interrupt service routine (ISR) (Lines 148 to 151) is activated, the co-operative scheduler's `TTCOS_Update()` code is run. This code checks through the to-do list and—based on the current time and the delay and period of each task—determines whether a given task is to be run or left asleep. A key point here is that the "Run-Me-Now" variables for each task are incremented rather than just being set to one. This guarantees that each occurrence of every task will be eventually be run even if the co-operative scheduler gets sidetracked handling an error condition.

The scheduler now falls into an infinite loop consisting of the `TTCOS_DispatchTask()` (Line 140) and `TTCOS_Sleep()` (Line 141) functions. The `TTCOS_DispatchTask()` function sequentially searches through the TTCOS task list for all tasks with a nonzero Run-Me-Now variable. It runs that task and decrements the Run-Me-Now variable. Finally, the processor is sent into a low power mode (`TTCOS_Sleep()`).

When the next timer interrupt reactivates the system, the Run-Me-Now flags are updated, the system awakens from the `TTCOS_Sleep()` mode, and the

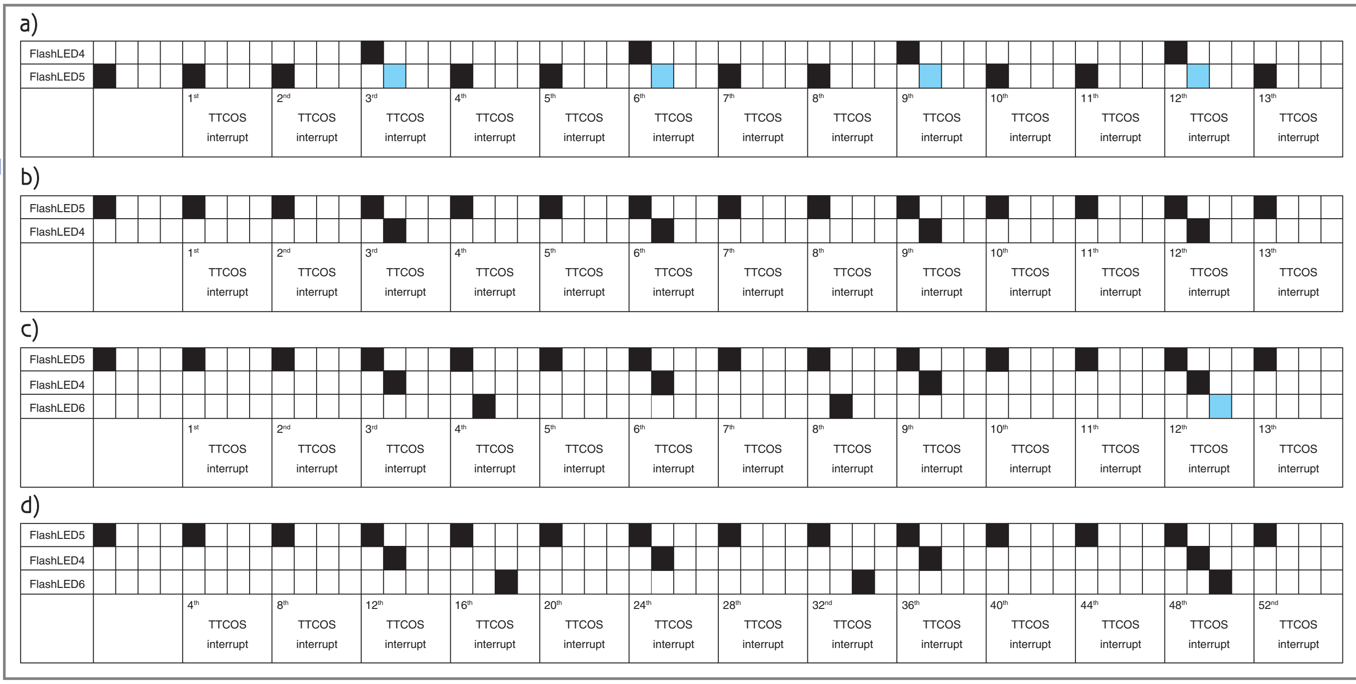


Figure 1a—Adding the tasks `TTCOS_AddTask(FlashLED4,3,3)` and `TTCOS_AddTask(FlashLED5,0,1)` to the co-operative scheduler task list leaves two tasks ready to run every third second. The second task is forced to wait until the processor completes the first task. The second task is said to suffer from "time jitter" (shown in blue). **b**—Adding the tasks to the scheduler list in a different order `TTCOS_AddTask(FlashLED5,0,1)` and `TTCOS_AddTask(FlashLED4,3,3)` still leaves two tasks ready to run every third second. As the second task is consistently forced to wait until the processor completes the first task, the system acts in the required manner, with LED 5 flashing at one 1-s intervals and LED 4 flashing every third second with no time jitter present. **c**—No reordering of the tasks will avoid the time jitter (shown in blue) when a third task `TTCOS_AddTask(FlashLED6,0,4)` is added to the system. This task flashes LED 6 every fourth second. **d**—By speeding up the scheduler interrupts four-fold, and predetermining appropriate initial delays for each task, no task is forced to wait until another task completes. All tasks complete at their designated time intervals without time jitter.

`TTCOS_DispatchTask()` is called again.

BUILDING A SCHEDULER

Coding a co-operative scheduler is straightforward. You need an array or linked to-do list containing information on required tasks and information about when to run them (the delays and periods set by `TTCOS_AddTask()` in Listing 4). To identify tasks ready to run, a timer-driven ISR is used to search through the list and compare the current system time with timing information stored with each task in the to-do list. Finally, there is a routine that activates all tasks that have been identified as ready-to-run.

Given the fact that I'm a little rusty when it comes to setting up prototypes of subroutines that manipulate pointers to functions (Line 120), I took the easy route. I used ideas from the example code on Pont's web site rather than completely develop my own.

Using a co-operative scheduler for the aforementioned tasks seems fairly intuitive. Each task that must be run is simply added to the scheduler's to-do list. With each task independent of the other tasks, it is not complicated to test each task in terms of functionality and execution time.

One of the stated advantages of a co-operative scheduler is that, because no tasks can preempt another, there are never any race conditions with two tasks fighting to access shared memory. That's not important with the simple GPIO/LED tasks in Listing 4. However, with the audio/video code I described earlier, that may be a significant advantage in providing the route to avoid the data race issues that may be causing the problems within the LCD code.

There is a well-known saying: *The devil is in the details*. Is using a co-operative scheduler just that simple? Yes, if you just want the co-operative scheduler as an organized form of "super-loop." In that case, you will probably be fine with a little bit of uncertainty about when a task will run (time jitter). To understand time jitter, consider Figure 1, which shows a simple example of a co-operative scheduler handling two tasks. Using the `TTCOS_AddTask(taskName, delay, period)` syntax, one task is set to flash LED 4 for 0.25 s every 3 s after an initial delay of 3 s:

```
TTCOS_AddTask(FlashLED4, 3, 3)
```

The other task flashes LED 5 every second after a zero initial delay:

```
TTCOS_AddTask(FlashLED5, 0, 1);
```

For simplicity, you can code both tasks using the following form:

```
void FlashLED_N(void) {
  Turn on LED N;
  Wait for ¼ second;
  Turn off LED N.
}
```

As a result, it should be easy for the processor to handle both tasks in the time available on the system. You can minimize the amount of expended battery power by setting the scheduler TTCOS timer to interrupt every second rather than more frequently.

You can test each task by making it the only task present in the scheduler's to-do list. It is easy to determine that each code section flashes the right LED as required. However, upon activating both tasks within the scheduler, you find that LED 4 is flashed every 3 s as required and LED 5 starts off flashing at 1-s intervals. However, after just three flashes, the time that LED 5 turns on starts fluctuating (time jitter). Sometimes LED 5 flashes 1 s after the last time is turned on, but sometimes after only 0.75 s; and at other times, the task is delayed to flash after 1.25 s.

It is easy to see why this is happening when you look at the timing graph (see Figure 1a). The `TTCOS_Update` code, driven by the TTCOS timer interrupt, will cause both the Run-Me-Now variables of the `FlashLED4` and `FlashLED5` tasks to be incremented every 3 s and every 1 s, respectively.

There are no problems until the timer interrupt at time = 3 s when both tasks have been prepared to run. The processor can handle only one task at any one time, so the first task in the to-do list runs (`FlashLED4( )`) and the second task (`FlashLED5( )`) is delayed.

For this example, there is a simple solution. Change the order in which the scheduler runs the tasks by changing the order in which they are added to the scheduler's to-do list:

```
TTCOS_AddTask(FlashLED5, 0, 1);
TTCOS_AddTask(FlashLED4, 3, 3);
```

As you can see in Figure 1b, there are now predictable operations with LED 5 flashing in 1-s intervals with LED 4 flashing reliably in 3-s intervals. However, this recoding is not a real fix, as adding a third task to flash LED 6 every 4 s will show.

```
TTCOS_AddTask(FlashLED5, 0, 1);
TTCOS_AddTask(FlashLED4, 3, 3);
TTCOS_AddTask(FlashLED6, 4, 4);
```

The time jitter reappears, now at 12-s intervals (see Figure 1c).

SOLVING THE TIME-JITTER PROBLEM

Rearranging tasks within the scheduler list will not ensure that tasks run with the exact period specified in the system requirements. Currently, the way I've described using the co-operative scheduler ensures a few things: LED 5 will flash during each 1-s period; LED 4 will flash during every 3-s period; LED 6 will flash during each 4-s period; and so on. However, there is no guarantee that the LED 5 task will run in exactly 1-s intervals, or that the LED 6 task will run in exactly 4-s intervals. For that to occur, you need to prearrange the time when every task runs rather than rearrange when each task runs.

This "unpredictability" issue can be solved in the following manner. By design, no interrupts other than the scheduler interrupt is permitted, so you can accurately predetermine the length of time that each task will run. You can then arrange that the initial delay parameter used when adding the task to the scheduler's to-do via `TTCOS_AddTask` is "just enough" to ensure that no task is ready to run until after a previous task has completed. For the three-LED example, you would need to add the tasks to the scheduler with the following initial delays and periods:

```
TTCOS_AddTask(FlashLED5, 0, 1);
TTCOS_AddTask(FlashLED4, 3.25, 3);
TTCOS_AddTask(FlashLED6, 4.5, 4);
```

However, just changing the task delay is not sufficient to stop the time jitter. With a 1-s timer interrupt, the `TTCOS_Update()` code will prepare the LED 4 task to run after a 4-s delay, not after the desired 3.25-s delay. Similarly, the LED 6 task will be prepared to run after 5 s and not after the 4.5 s needed to solve the time jitter problem. Thus, the time jitter will simply be reintroduced elsewhere in the project.

This problem can be solved if you speed up the timer interrupts so that the scheduler is capable of handling "a quarter second delay," as shown in Figure 1d. The final time-triggered schedule code for the three LEDs flashing at predictable intervals is shown in Listing 5. In this listing, the `TTCOS_Init` function prototype (Line 168) has been changed so that the optimum interrupt time for maximum power saving can be set for each co-operative scheduler project: 0.25-s interrupts for this project. The tasks delays (Lines 157 to 159) and task periods (Lines 161 to 164) are not defined as specific time intervals but as multiples of the timer interrupt period. Describing the delay or period times using a float variable would not lead to any greater precision as the Run-Me-Now variables for each task are only updated by the `TTCOS_Update` routine each time the timer interrupt occurs.

WHERE TO NEXT?

In this article, I explained my own reawakening to the advantages of co-operative schedulers. According to reports, such schedulers have the potential to offer an advantage that preemptive schedulers don't offer—predictability of performance. After detailing with some syntax issues, the simplicity and code maintainability of using a co-operative scheduler for handling multiple tasks was demonstrated. However, it was then shown that this "bull-at-the-gate" approach of using co-operative schedulers is acceptable only when the embedded project requirements can be satisfied with a predictability of the form—this task must run sometime during a period of x seconds. Using this predictability definition is often acceptable when the project functionality will not be compromised by a certain level of "time jitter" (random delay) in the execution time of the tasks. However, it is different than meeting the stricter requirement often

Listing 5—By analyzing the system ahead of time, it is possible to schedule all the tasks so that no task is ever waiting for another task to complete before being run by the scheduler. With this approach, each task operates with complete predictability and reliability. However, if the `GPIO_LED_Echo()` task to echo the switch input to the LED output was activated (Line 184), this task would suffer from "time jitter" as it is continually blocked and then unblocked by the other tasks.

```
155 #define QUARTER_SECOND_INTERRUPTS (250 * 500 * 1000)      // Processor clock cycles per 1/4 second
156
157 #define NO_DELAY                  (0)                      // 0 interrupt delay
158 #define QUARTER_SECOND_DELAY      (1)                      // 1 interrupt delay
159 #define HALF_SECOND_DELAY         (2 * QUARTER_SECOND_DELAY)   // 2 interrupt delay
160
161 #define RUN_ONCE                  (0)                      // Delete task if not periodic
162 #define EVERY_SECOND              (4)                      // 1 second task period = Every 4th interrupt
163 #define EVERY_THREE_SECONDS       (3 * EVERY_SECOND)       // 3 second task period
164 #define EVERY_FOUR_SECONDS        (4 * EVERY_SECOND)       // 4 second task period
165
166
167 int main(void) {
168     TTCOS_Init(QUARTER_SECOND_INTERRUPTS);  // Initialize the co-operative scheduler 'to-do' list and timer
169
170             // Add  Tasks
171             // TTCOS_AddTask(FunctionPointer, taskDelay, taskPeriod);
172
173             // The sub-system initialization tasks run with NO_DELAY and RUN_ONCE only
174     TTCOS_AddTask(GPIO_Init, NO_DELAY, RUN_ONCE);    // Initialize the GPIO input sub-system
175     TTCOS_AddTask(LED_Init,   NO_DELAY, RUN_ONCE);   // Initialize the LED output sub-system
176
177             // Specify delay and period of other tasks
178     TTCOS_AddTask(FlashLED5, NO_DELAY,       EVERY_SECOND);                 // Every second for 1/4 s
179     TTCOS_AddTask(FlashLED4, QUARTER_SECOND_DELAY, EVERY_THREE_SECONDS);    // Every 3 seconds for 1/4 s
180     TTCOS_AddTask(FlashLED6,  HALF_SECOND_DELAY, EVERY_FOUR_SECONDS);       // Every 4 seconds for 1s / 4 s
181
182     // Task to echo switch input to LED output realistically needs to be run every 1 / 4 s
183     // ****** If activated, this task would have "considerable" time jitter as it is 'blocked' by other tasks
184     //******* TTCOS_AddTask(GPIO_LED_Echo, NO_DELAY, RUN_ALWAYS);
185
186     TTCOS_Start( );             // Activate the co-operative scheduler's timer interrupt
187
188     while (1) {
189       TTCOS_DispatchTasks( );   // Run any tasks with a Run-Me-Now variable that is not zero
190       TTCOS_Sleep( );           // and send the processor back to sleep until next timer interrupt
191   }
192
193   return 0;                     // Keep the compiler from complaining
194 }
```

required in safety-critical systems that the task must run precisely every x seconds.

A simple approach was introduced to overcome the time-jitter problem. This involved speeding up the interrupts and precalculating task delays to ensure that no two tasks were ever prepared by the co-operative scheduler to be ready to run at the same time.

However, many practical embedded issues remain unresolved in the use of a co-operative scheduler. For example, suppose there are tasks that take 3 s to complete (e.g., output of a buffer) while other tasks must be executed every 0.25 (the immediate echo of a value in an input channel to an output channel, as shown in Listing 5, Lines 182 to 184). Direct application of the co-operative scheduler appears to suggest that the reliable operation of the shorter-running task will be compromised (blocked) by the longer-running task. It actually sounds as if the project would be better handled by a scheduler where the faster task preempts the slower task. I plan to look at these issues in future articles as I re-examine the use of time-triggered systems. There I will look at these issues, and others, and show that they can be overcome to recover the predictability stated as being available using co-operative schedulers.

Michael Smith (mike.smith@ucalgary.ca) is a long-time Circuit Cellar *contributor, who disappears into the woodwork for a couple of years until he finds something "new" that really sparks his curiosity! He is a professor of computer engineering at the University of Calgary, Canada. Michael's main interests are developing new biomedical engineering algorithms and moving them onto multicore and multiple-processor embedded systems in a systematic and reliable fashion. He recently became a convert to the application of Agile Methodologies in the embedded environment. Mention "test-driven development" and his eyes light up. In 2008, Michael had his Analog Devices University Ambassadorship renewed for the eighth straight year.*

SOURCES

BF533 EZ-KIT and BF548 evaluation board
Analog Devices, Inc. | www.analog.com