Cable Tracer Software, p. 36 • Uncomplicated Embedded Wireless, p. 50 • Partition a Design, Minimize Processing, p. 62

www.circuitcellar.com

CIRCUIT CELLAR

THE MAGAZINE FOR COMPUTER APPLICATIONS

#230 September 2009

DATA ACQUISITION

Very Low-Frequency Datalogger Design

A Simple Circuit for Energy Conservation

Sophisticated Embedded App Development

Battery-Monitoring Circuitry

Design & Then Program a Stand-Alone Timer

$5.95 U.S. ($6.95 Canada)

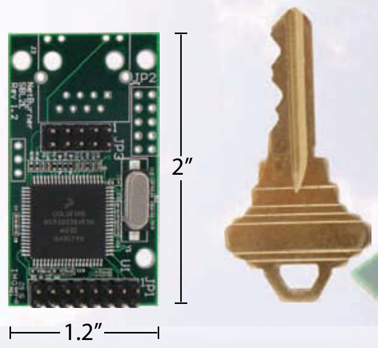
A Low Cost Solution for

Industrial Serial to Ethernet

with Digital I/O and Analog to Digital Converters

Hardware Features
- Two TTL serial ports
- 10/100 Mbps Ethernet
- Up to 10 digital I/O
- Four 12-bit A/D converters
- Operating temperature -40 to 85 C
- Dimenstions: 1.2" x 2"
- 32-bit performance

Software Features
- No programming required
- TCP / UDP / Telnet / HTTP modes
- DHCP / static IP support
- Web-based configuration



$19.95
Qty. 1000

The all-new

NetBurner SBL2e

INDUSTRIAL SERIAL TO ETHERNET SOLUTION

A NetBurner Complete Hardware & Software Solution

NetBurner
Networking in One Day!

Board Part Number | SBL2e-200IR
Information and Sales | sales@netburner.com
Web | www.netburner.com
Telephone | 1-800-695-6828

freescale™
Alliance Member

"We lost almost *two years* trying to get so-called 'off-the-shelf' solutions to work ...we could have reduced all of that effort to *just **six weeks*** with the Synapse development tools and wireless technology!"
***Mike Kranz**, Director of R&D, Stanley Associates*

Focus on Your Application Instead of The Network.

The Network is a SNAP!

· Auto-forming, 802.15.4 mesh network
· Mesh routing with maximum battery life
· Easily connects to TCP/IP
· Over-the-air programming

Synapse's technology is unprecedented in the industry

Called **SNAP**™, an auto-forming, self-healing mesh network (802.15.4 protocol) combined with an embedded Python interpreter (supporting XML-RPC) for running application code. SNAP is "Instant-On," meaning that communications begin within milliseconds of power-up, and all SNAP devices are peers that can bridge, route and sleep.

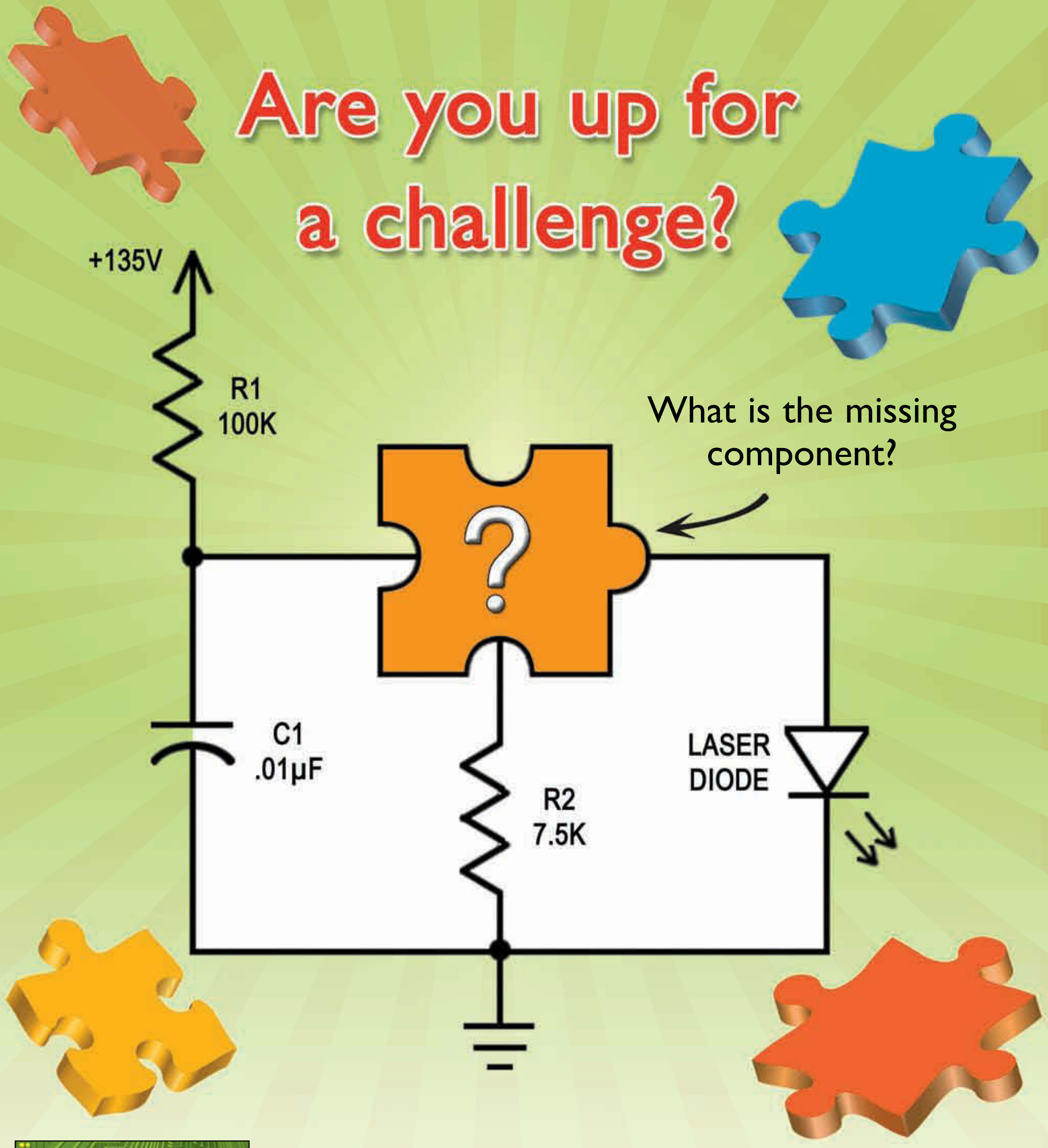
Forget hardware probes, compilers, assemblers, etc. With its companion Portal® software, SNAP supports interactive application development over-the-air. This allows you to focus on your application functionality, not the wireless infrastructure.

Evaluation kits start at $149 — begin developing your application today.
Call **1-877-982-7888** or visit us at **www.synapse-wireless.com**

Wireless Technology to Control and Monitor Anything from Anywhere™

SYNAPSE®

Are you up for a challenge?



Industry guru Forrest M. Mims III has created a stumper. Can you figure out what's missing? Go to www.Jameco.com/expose to see if you are correct and while you are there, sign-up for our free full color catalog. It's packed with components at prices below what you are used to paying.

JAMECO® ELECTRONICS

60 MHz - 1 GHz
Starting from $1,000

Experience the New LeCroy Oscilloscopes*

Debug, verify, characterize, troubleshoot, analyze — oscilloscopes do it all, but LeCroy's breakthrough oscilloscopes do more. More measurements, more math, and more time-saving tools to easily make measurements in even the most challenging situations.

LeCroy

*To learn more about LeCroy's full line of oscilloscopes, including the 30 GHz WaveMaster 8 Zi, the world's fastest real-time oscilloscope, visit www.lecroy.com or call 1-800-5-LeCroy

TASK MANAGER

Develop Code, Acquire Data

No code, no data. It's just that simple.

Whether you're gathering timing info, radio emissions data, or the locations of underground cables, a data acquisition system—no matter how well-designed the hardware—is useless without a well-written program running behind the scenes. This month we present articles about data acquisition systems that harness the power of reliable code to manage hardware and move data from point A to B in a fluid manner.

For instance, starting on page 16, Chuck Baird explains how he combined his programming and hardware design skills to develop a standalone timing system. With the right code, you can use the design as a standalone timer or a timepiece for a more complicated application.

Want to gather radio emissions data? Turn to page 24 to learn how Carlo Tauraso built and programmed a very low-frequency (VLF) datalogger. After describing the design, he details the firmware development process and presents an interesting real-world application.

On page 36 begins the second part of Kevin Gorga's two-part article series, "Cable Tracer Design." Here he presents the software portion of the design process and then describes the system's controls.

A successful embedded development application is built upon good code. Last month Dale Wheat familiarized you with ARM Cortex-M3 microcontrollers. This month he details the application development process and covers the coding procedures from start to finish (p. 44).

After covering C language in several articles, George Martin asks an important question (p. 62). What do you do once you have your embedded processor up and running? George describes how to compartmentalize your design work and minimize processing requirements.

Not every article in this issue is code-centric. For instance, in "Neural Networker," Tom Cantrell presents what he calls "a SNAP-shot" from Synapse Wireless, which now offers a unique solution that integrates multiple aspects of an embedded wireless project on a small piece of silicon (p. 50).

Starting on page 58, Brian Millier describes his "smart" power bar, which is a compact unit that can turn off several electronic devices at the same time, thus conserving energy. The interesting design's circuitry consists of a basic 60-Hz power supply, a microcontroller, a relay, and an IR module.

Columnist Jeff Bachiochi ends the issue with a presentation of how internal protection circuitry for Li-Ion battery cells can prevent dangerously high temperatures and failure (p. 66). Like Jeff, you can check a cell's state of charge, find problems, and then repair them.

Whether programming, developing hardware, or both, be sure to share your experiences with us. We're just an e-mail away.

cj@circuitcellar.com

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

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

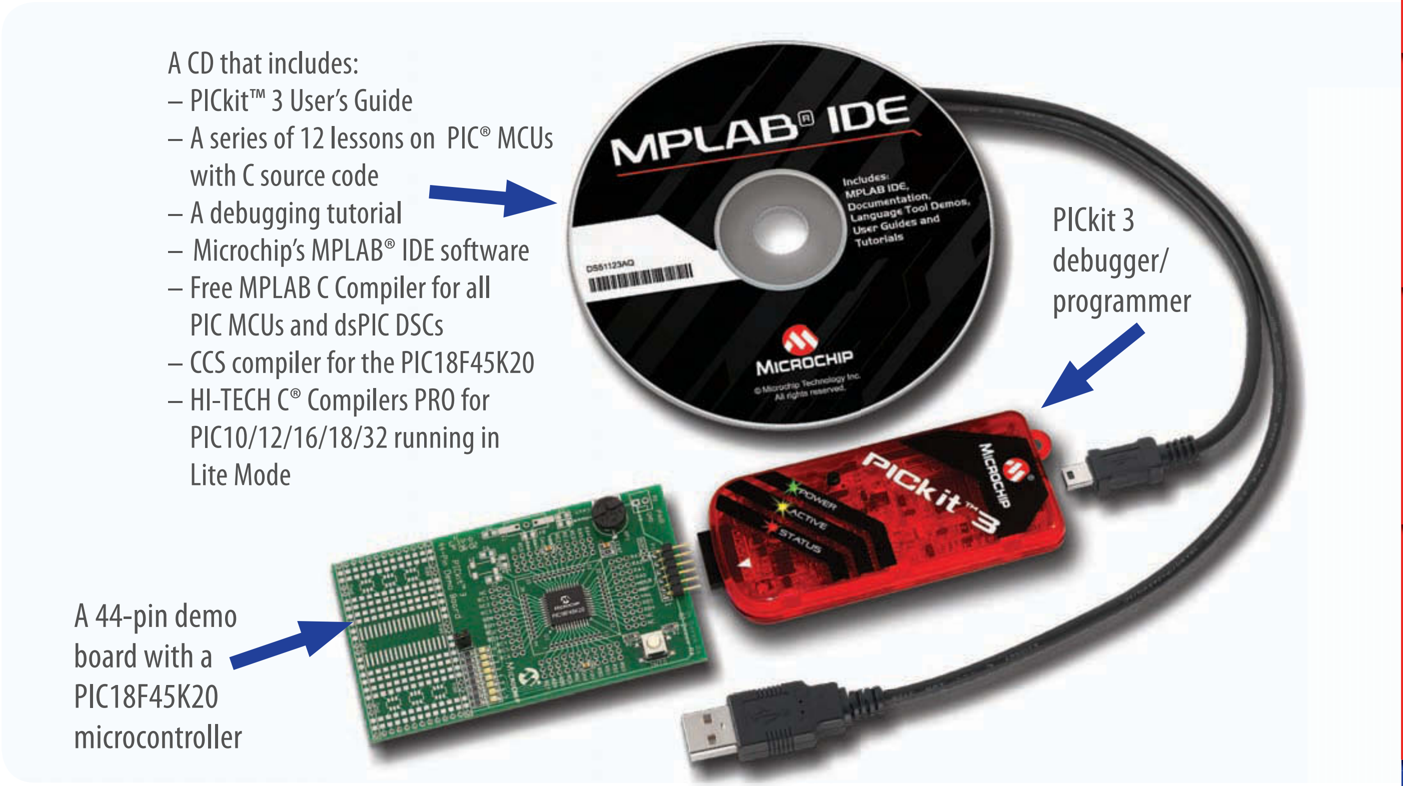
AUTHORIZED REPRINTS INFORMATION
860.875.2199, E-mail: reprints@circuitcellar.com

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.

One Debugger/ Programmer


A CD that includes:
– PICkit™ 3 User's Guide
– A series of 12 lessons on PIC® MCUs with C source code
– A debugging tutorial
– Microchip's MPLAB® IDE software
– Free MPLAB C Compiler for all PIC MCUs and dsPIC DSCs
– CCS compiler for the PIC18F45K20
– HI-TECH C® Compilers PRO for PIC10/12/16/18/32 running in Lite Mode
PICkit 3 debugger/ programmer
A 44-pin demo board with a PIC18F45K20 microcontroller
MPLAB® IDE
Includes: MPLAB IDE, Documentation, Language Tool Demos, User Guides and Tutorials
MICROCHIP
PICkit™ 3
POWER
ACTIVE
STATUS

One Low Price ... $69.99

Microchip's PICkit 3 Debug Express (part number DV164131) incorporates in-circuit debugging technology to provide an extremely affordable hardware debugger and programmer for the entire range of Microchip's Flash Microcontrollers (MCUs) – from the smallest 8-bit PIC10 MCU, through all 16-bit dsPIC® DSCs to the largest 32-bit PIC32 MCU.

PICkit 3 runs under the popular MPLAB IDE, the free graphical integrated development environment software, complete with programmer's editor, software simulator and project manager. MPLAB IDE's GUI promotes advanced debugging with a host of auxiliary features, such as a segmented LCD Designer, a graphics display designer and a data monitor and control interface. Easily connected to the PC with a USB interface, PICkit 3 is bundled with a demo board for fast learning and initial design prototyping. The two-wire interface easily connects to the final designs for application tuning and quick in-circuit programming.

With rich features, free compilers fully integrated with MPLAB IDE and a suite of tutorials, Microchip PICkit 3 Debug Express delivers substantial value at a remarkably low price.

For All PIC® Microcontrollers

www.microchip.com/PICkit3

MICROCHIP

INSIDE ISSUE 230

BONUS CONTENT
Verification and Simulation of FPGA Designs

September 2009 • Data Acquisition

Hammer Down Your Power Consumption with picoPower™!

THE Performance Choice of Lowest-Power Microcontrollers

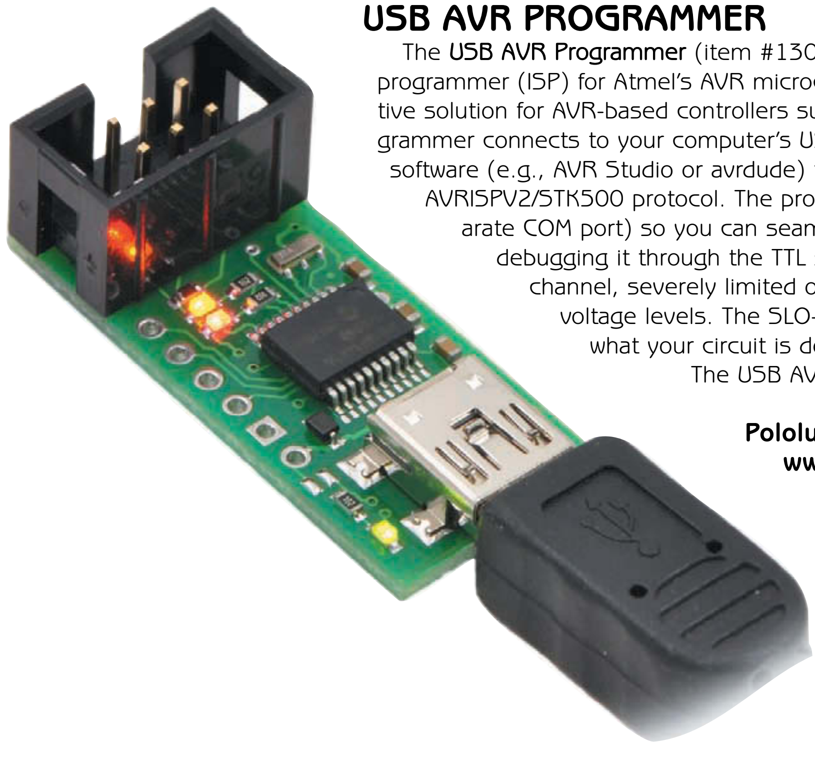
Performance and power consumption have always been key elements in the development of AVR® microcontrollers. Today's increasing use of battery and signal line powered applications makes power consumption criteria more important than ever. To meet the tough requirements of modern microcontrollers, Atmel®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



USB AVR PROGRAMMER

The **USB AVR Programmer** (item #1300) is an extremely compact, low-cost, in-system programmer (ISP) for Atmel's AVR microcontrollers. This makes the programmer an attractive solution for AVR-based controllers such as the Orangutan robot controllers. The programmer connects to your computer's USB port and communicates with your programming software (e.g., AVR Studio or avrdude) through a virtual COM port using the AVRISPV2/STK500 protocol. The programmer includes a TTL-level serial port (on a separate COM port) so you can seamlessly switch between programming an AVR and debugging it through the TTL serial port. The programmer also features a two-channel, severely limited oscilloscope (SLO-scope) for monitoring signals and voltage levels. The SLO-scope Client for Windows makes it easy to see what your circuit is doing in real time.

The USB AVR Programmer costs **$19.95**.

Pololu Corp.
www.pololu.com

32-Kb SERIAL F-RAM

The **FM24CL32** is a serial nonvolatile RAM that offers high-speed read/write performance, low-voltage operation, and superior data retention. The FM24CL32 features 32-Kb nonvolatile memory, 2.7- to 3.6-V operation in an eight-pin SOIC package that uses the two-wire I^{2}C protocol. The FM24CL32 features fast access, NoDelay writes, virtually unlimited read/write cycles (1E14), and low-power consumption. The FM24CL32 is a direct hardware replacement for serial EEPROM used in industrial controls, metering, medical, military, gaming, and computing applications, among others.

The FM24CL32 is a serial ferroelectric random access memory organized as a 4,096 × 8-bit memory array. Unlike an EEPROM, there are no write delays when using the FM24CL32, and the next bus cycle may commence immediately without the need for data polling. The device writes at a bus speed up to 1 MHz and supports legacy timing at 100 kHz and 400 kHz. The FM24CL32 provides data retention for up to 45 years while eliminating the complexities, overhead, and system-level reliability problems caused by EEPROM and other nonvolatile memories. The FM24CL32 operates over the industrial temperature range of –40° to 85°C at 2.7 to 3.6 V, with an active current of 70 µA (typical at 100 kHz) and 12-µA current in standby.

Unit pricing begins at **$0.99** in quantities of 10,000.

Ramtron International Corp.
www.ramtron.com

NEW PRODUCT NEWS

Edited by John Gorsky

TOUCH PANEL COMPUTER WITH PoE

The **TPC-43B** is a cost-effective computer with a color TFT LCD and touch panel plus power-over-Ethernet (PoE). Targeted at building automation and factory HMI applications, this standard, off-the-shelf, ARM-powered, wall-mount design features a WqVGA (480 × 272) 4.3", color TFT LCD panel in landscape format, with 65,000 colors, an adjustable LED backlight, and a touch panel.

The computer is powered by a standard Medallion CPU Module (an SA2410-650 is included). The ARM9 processor features 32 KB of cache, 64 MB of SDRAM, and 128 MB of NAND flash, plus a separate NOR flash for reliable bootloader and software update options. Power via PoE dramatically reduces installation costs and replaces power cables with a single CAT-5 cable for both power and Ethernet. The unit can also be powered from 10 to 30 VDC or 7 to 24 VAC.

The unit offers low-power consumption and passive cooling that enables fanless operation. To reduce power consumption, software can turn the backlight on or off and control the brightness. A low-power, RTC with battery backup is included.

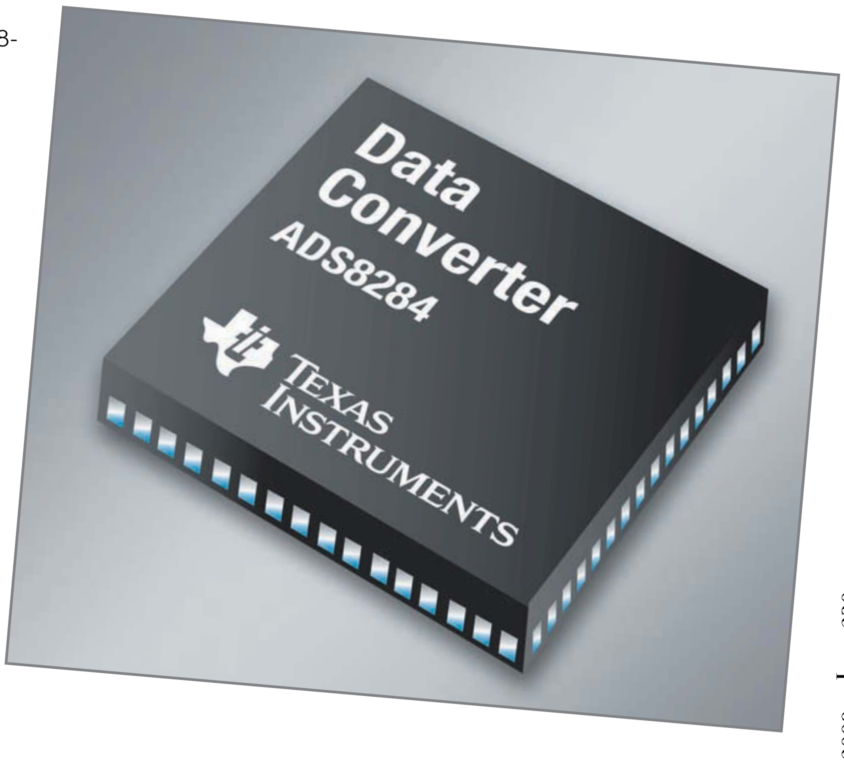
Communication is via a full-speed, USB host port (USB type A connector), and it supports wireless expansion, such as 802.11 or Bluetooth. Also included is a 10/100-BaseT Ethernet interface with an RJ-45 connector featuring integrated LED status indicators. An RS-232 serial interface with an optional DE9 adapter to accelerate software development is also available. The unit ships with the proven Medallion Linux 2.6 plus drivers preinstalled.

Pricing for units with all options except wireless is **$399** in 10-piece quantities. Single units of the TPC-43B are available as development kits with full software development tools and support. Call for pricing.

Techsol
www.medallionsystem.com

18-BIT SoC UP TO 1 MSPS FOR HIGH-SPEED DATA ACQUISITION

The **ADS8284** and **ADS8254** are two new system-on-chip (SoC) solutions that enable customers to easily develop ultra-high-performance ADC front ends for precision applications, including high-speed data acquisition, automated test equipment, and medical imaging. The 18-bit ADS8284 and 16-bit ADS8254 combine for the first time TI's latest generation successive approximation register ADCs with all the components necessary to optimize design around the ADC, often the most challenging part of system design.



The devices offer best-in-class AC and DC performance at 1 MSPS. This provides guaranteed performance characterized for the entire input channel from the multiplexer to the digital output, including the driving amplifier. They are fully integrated SoCs and include a four-channel multiplexer, an input op-amp, a 6 ppm/°C reference, and a reference buffer to reduce design complexity and cost while ensuring that real-world applications achieve datasheet performance levels. They also offer ultra-low-channel drift to maximize reliability and repeatability for critical measurements.

The ADS8284 and ADS8254 are available now in a 64-pin QFN package for **$20.50** and **$16.50**, respectively, in 1,000-unit quantities.

Texas Instruments
www.ti.com

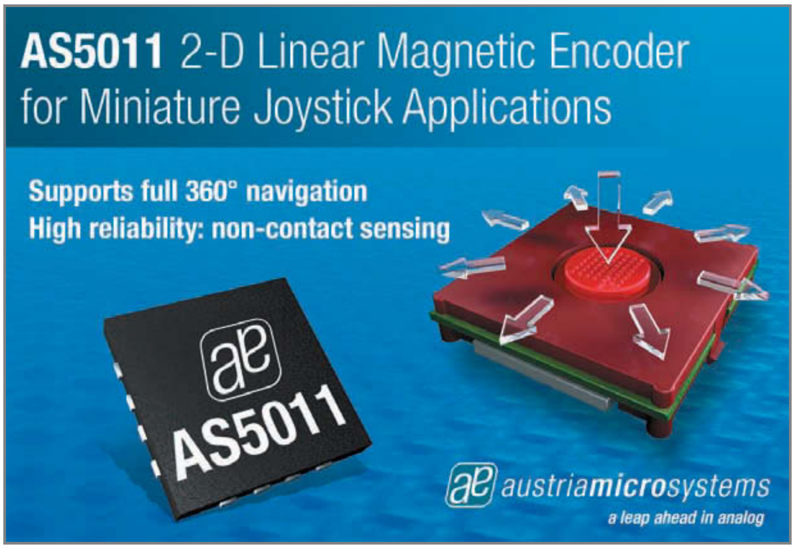
JOYSTICK MODULE BASED ON CONTACTLESS SENSING TECHNOLOGY

The **EasyPoint** is a new joystick module aimed at portable communication devices. EasyPoint consists of a mechanical stack incorporating a navigation knob with a magnet and the AS5011, a contactless sensing IC. Its simple construction, and the contactless-sensing technique implemented by the AS5011, means the module offers very high mechanical reliability. The EasyPoint joystick modules support true 360° range of movement, encompass a "select" function, and can be housed in a variety of form factors, with a module height as low as 1.8 mm.

At the core of the EasyPoint joystick module is the AS5011, a magnetic encoder IC that monitors knob displacement relative to its center position and provides position information via I^2C outputs. The AS5011 offers user-selectable, power-saving modes and is available in an ultra-thin QFN package (5 × 5 mm, 0.55 mm high).

EasyPoint joystick module demonstration units and the AS5011 IC are available for sampling. The modules come in two mechanical versions, either as a complete module with integrated AS5011 IC or as a mechanical module only. The sizes range from 12.5 mm × 12.6 mm to 18.6 mm × 22.6 mm and support 1-mm and 2-mm knob displacement, respectively.

The EasyPoint starts at **$2.30** and the AS5011 costs **$2.62** each in 1,000-unit quantities.


AS5011 2-D Linear Magnetic Encoder for Miniature Joystick Applications
Supports full 360° navigation
High reliability: non-contact sensing
AS5011
austriamicrosystems
a leap ahead in analog

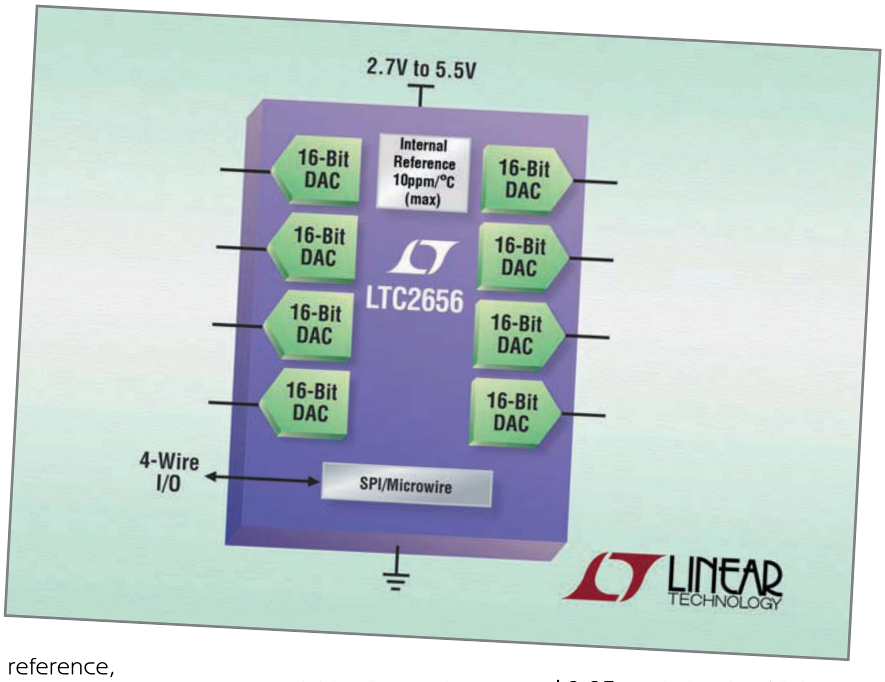
Austriamicrosystems
www.austriamicrosystems.com

NPN

16-BIT OCTAL SPI DAC

The **LTC2656** is a 16-bit octal DAC that offers ±4 LSB INL maximum over temperature, a factor of three times better than the nearest octal competitor. The combination of low 0.1% (max) gain error and low ±2 mV (max) offset error ensure that the LTC2656 remains accurate near the supply rails and provides the user with a wider effective output range.

The LTC2656 integrates a precision reference that achieves 2 ppm/°C typical and 10 ppm/°C maximum temperature coefficient. Operating from a single 2.7- to 5.5-V supply, supply current is a low 375 µA per DAC with the reference activated. Communication is via a four-wire SPI-compatible interface up to 50 MHz.

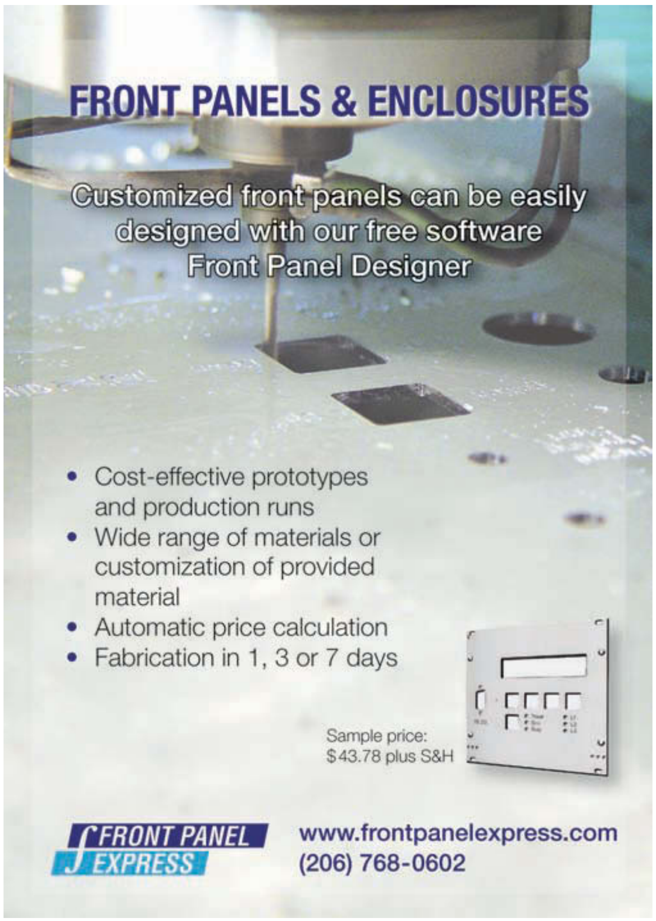


The device offers a wide range of options to meet application-specific requirements. Designers can choose between 16- or 12-bit resolution and an internal 1.25- or 2.048-V reference, which produce a full-scale output voltage of 2.5 or 4.096 V. Alternatively, an external reference of up to half the supply voltage can be for rail-to-rail operation. The LTC2656 also includes a hardware option to power up the DAC outputs at zero-scale or midscale.

LTC2656 16-bit and 12-bit DACs and demo boards are now available. Pricing begins at **$8.95** each for the 12-bit options and **$17.95** each for the 16-bit options in 1,000-piece quantities.

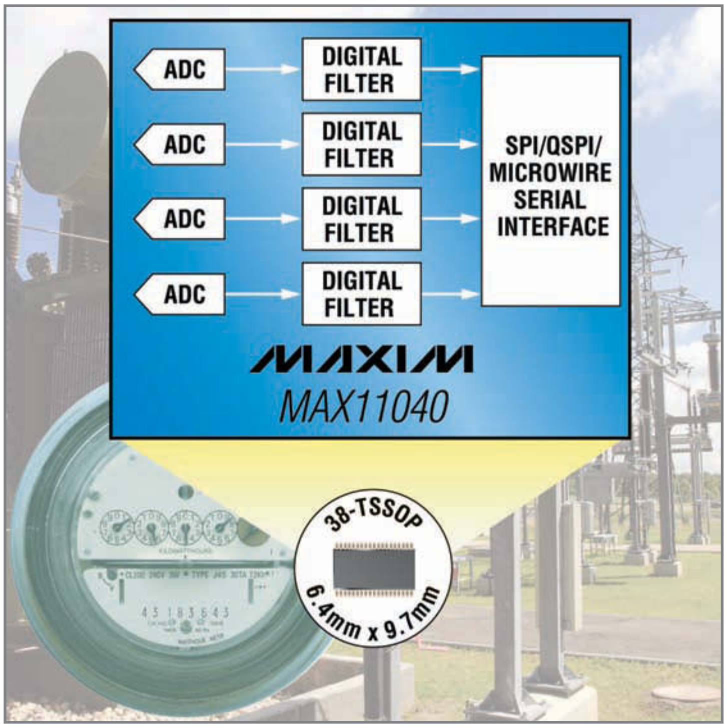
Linear Technology Corp.
www.linear.com

WIRELESS MADE SIMPLE®
BRING YOUR PRODUCT QUICKLY AND LEGALLY TO MARKET
RF Modules
Add INSTANT wireless analog / digital capability to your product.
Low-Cost TX, RX &TRX Modules
Multi-Channel Modules
Long-Range Modules
OEM Products
FCC PRE-CERTIFIED & ready to customize for your application.
Handheld TXs
Function Modules
Keyfob TXs
Feature Products
A closer look at Linx innovation
LOW-COST • LONG-RANGE TRANSCEIVER
• Direct serial interface
• Low power consumption
• PLL-synthesized architecture
• RSSI and power-down functions
• Compact surface-mount package
• No external RF components (except antenna)
REMOTE CONTROL TRANSCODER IC
• Up to 8 inputs
• Bi-directional control
• Transmitter ID output
• Automatic confirmation
• Secure 2^24 possible addresses
• Latched and/or momentary outputs
LINX® TECHNOLOGIES
WIRELESS MADE SIMPLE
800-736-6677
159 Ort Lane · Merlin, OR 97532
www.linxtechnologies.com


FRONT PANELS & ENCLOSURES
Customized front panels can be easily designed with our free software Front Panel Designer
• Cost-effective prototypes and production runs
• Wide range of materials or customization of provided material
• Automatic price calculation
• Fabrication in 1, 3 or 7 days
Sample price: $43.78 plus S&H
FRONT PANEL EXPRESS
www.frontpanelexpress.com
(206) 768-0602

FOUR-CHANNEL, SIMULTANEOUS-SAMPLING 24-BIT ADC

The **MAX11040** four-channel, 24-bit, simultaneous-sampling ADC was developed to enable unprecedented simplicity in system design—up to eight MAX11040 devices can be daisy-chained to provide a sampling capacity of up to 32 channels. The MAX11040 achieves this high capacity and scalability with a unique cascadable SPI/QSPI/MICROWIRE interface, which eliminates the complexity of using individual chip-select inputs to control each ADC. Because all eight MAX11040 devices can be accessed using only a single chip-select input, from the host processor's point of view, the control logic remains the same regardless if there are only four or as many as 32 channels.



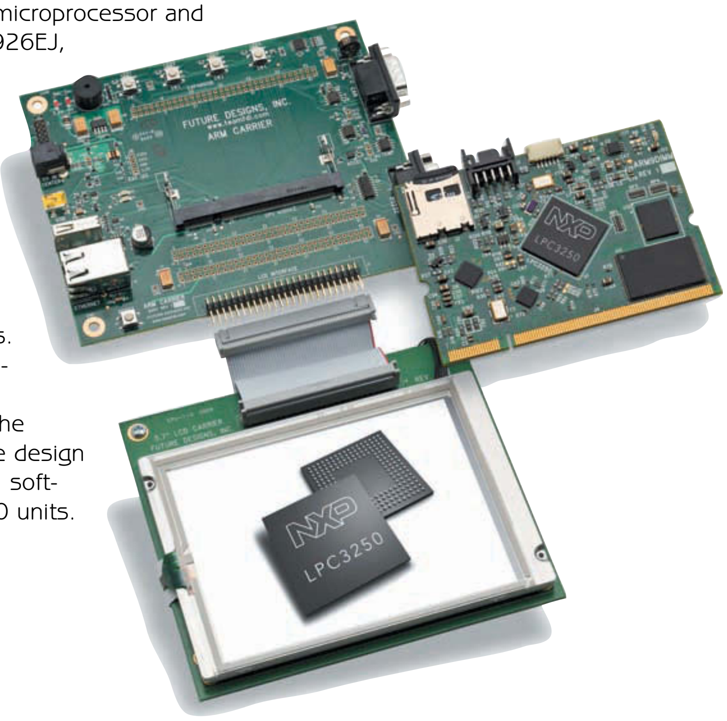
The MAX11040 uniquely provides compensation for phase shifts caused by resistor dividers, transformers, or filters at the analog inputs through a programmable delay of 0 to 333 µs. Together with a minimum of 90 dB of SINAD and 91 dB of SFDR, these features make the MAX11040 ideal for industrial power-grid-protection equipment, medical EKG/EEG equipment, and other applications that require accurate conversions of simultaneously sampled channels between 0.25 ksps and 64 ksps. Additionally, overvoltage protection on the analog inputs protects the MAX11040 from voltages as large as ±6 V when referenced to the device's analog ground, thus simplifying design and reducing cost.

The MAX11040 is available in a lead-free, 38-pin TSSOP package. Prices start at **$13.45** (1,000 up, FOB USA).

Maxim Integrated Products
www.maxim-ic.com

ARM9-BASED TOUCH SCREEN PLATFORM FOR HMI/GUI APPLICATIONS

The **ARM-57TSLPC3250** is a new addition to a modular development platform that supports various size touch screen LCD devices from multiple vendors. The design supports ARM microprocessor and microcontroller devices in three popular families: the ARM926EJ, ARM7TDMI-S, and Cortex-M3. The new ARM9-based platform complements the initial product, the ARM-57TS-LPC2478. The ARM-57TSLPC3250 includes a Toshiba 5.7" TFT LCD with integrated touchscreen and is based on the NXP LPC3250 266 MHz ARM926EJ-S microcontroller with integrated LCD driver. The "brain" of the system is a SOC module called the ARM9DIMM-LPC3250. This modular 200-pin SODIMM is 2.66" × 1.89" and contains the LPC3250 microcontroller, along with 256 MB of external NAND flash, 32 MB of SDRAM, and support circuits. The kit includes the Linux operating system with a full complement of software driver support.

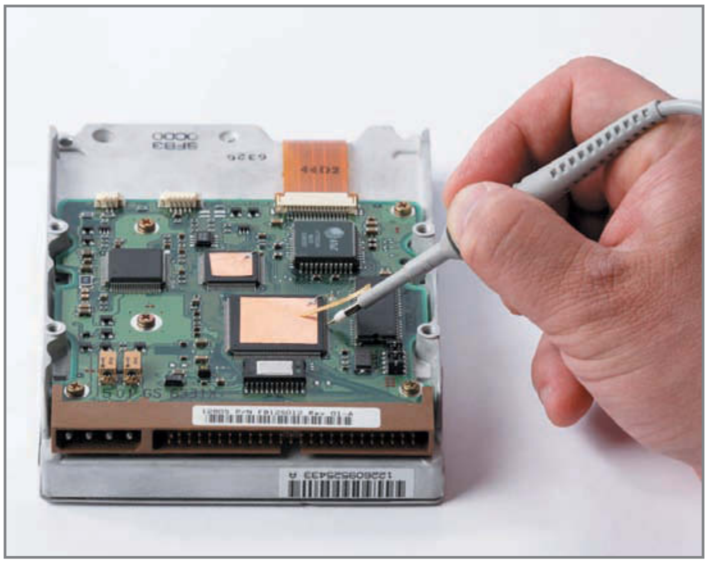


The ARM-57TS-LPC3250 development kit costs **$480**. The ARM9DIMMLPC3250 costs **$99.75**. FDI offers cost-effective design and customization services for customer-specific hardware, software, or repackaging applications at volumes as low as 500 units.

Future Designs, Inc.
www.teamfdi.com

FAMILY OF MINIATURE, PASSIVE OSCILLOSCOPE PROBES

The **N2870A Series** is a family of miniature, passive oscilloscope probes and accessories with bandwidths from DC to 1.5 GHz. Compact 2.5-mm probe head diameter, low-input capacitance, and various fine-pitch, probe-tip accessories make the passive probes ideal for probing densely populated IC components or surface-mount devices used in today's high-speed digital applications.



The sharp probe tip is spring loaded to help engineers keep the probe from slipping off the device under test. Insulating IC caps keep the small probe tip centered on the IC lead and keep it from shorting adjacent leads. Optional probe-tip accessories provide specialized capabilities for demanding applications. N2870A Series probes are available with attenuation ratios of 1:1, 10:1, 20:1, and 100:1, and probe bandwidths of DC to 35 MHz, 200 MHz, 350 MHz, 500 MHz, and 1.5 GHz. The new Infiniium 9000 Series oscilloscopes come equipped with one N2873A 500-MHz, 10:1 passive probe per channel.

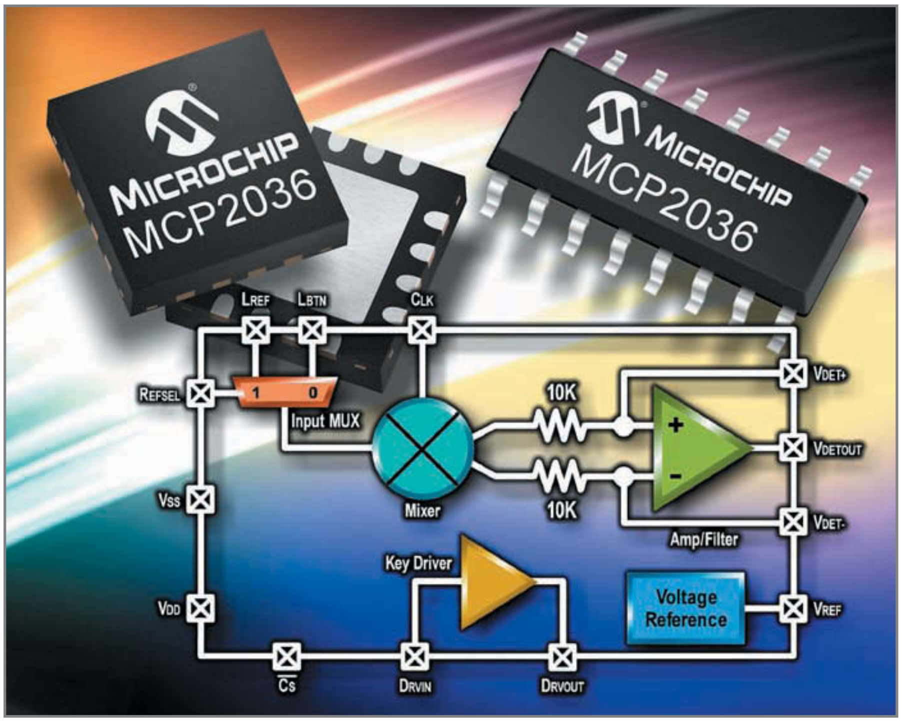
The N2870A Series passive probes are available now, with prices starting at **$220**.

Agilent Technologies, Inc.
www.agilent.com

INDUCTIVE TOUCH-SENSING ANALOG FRONT END

The **MCP2036** is an analog front end (AFE) for inductive touch-sensing applications. The fully-integrated MCP2036 AFE works with almost any 8-, 16-, or 32-bit PIC microcontroller or dsPIC digital signal controller, making it even easier and more cost effective for designers to enhance user interfaces with inductive touch-sensing technology. The inductive-touch AFE includes a multiplexer, a frequency mixer, an amplifier, a driver, and a voltage reference, which drastically lowers the component count, design size, and cost. Additionally, the AFE can be easily configured for a variety of applications in the appliance, industrial, and automotive markets, among others.

Inductive touch sensing's fundamental operating principles enable it to work through a front panel, such as plastic, stainless steel or aluminum. The technology also works through thick gloves and on surfaces where liquids are present. These characteristics make inductive touch sensing suitable for applications in the appliance market because of the possibility of a stainless steel front panel, the industrial market because of the technology's robustness, and the automotive market because of the technology's sleek aesthetics and ability to reduce accidental-touch triggers. The MCP2036 AFE is available in a 16-pin, 4 mm × 4 mm QFN package, as well as 14-pin PDIP and SOIC packages. The device starts at **$0.33** each in 10,000-unit quantities.



Microchip Technology, Inc.
www.microchip.com

NPN

COMPACT MODULES SIMPLIFY GPS

The **SG** and **SR Series** of GPS modules can be easily applied to blend high performance, low power and cost effectiveness into a single, compact, SMD package. The module's SiRFstar III low-power chipset minimizes power consumption and provides exceptional sensitivity, even in dense foliage and urban canyons. The receivers feature an on-board LNA and SAW filter, as well as an integrated antenna (SR Version) or external antenna (SG Version), which further lowers cost and reduces complexity. No other RF components are needed and the module's standard NMEA data output makes them easy to integrate, even by engineers without previous RF or GPS experience.

Also available is the MDEV-GPS Master Development System, which contains everything needed to rapidly evaluate the SG or SR Series GPS modules and implement them in record time. This all-inclusive Master Development System features a pre-assembled development board with an on-board OLED display for standalone testing. The system can also be attached to a PC via USB and operated using the supplied software. The software shows satellite positions, SNR, satellites in use, NMEA data, coordinates, and even the module's position on Google Maps! For development, a convenient prototyping area with breakout headers and regulated power supply allows for rapid testing and interface.

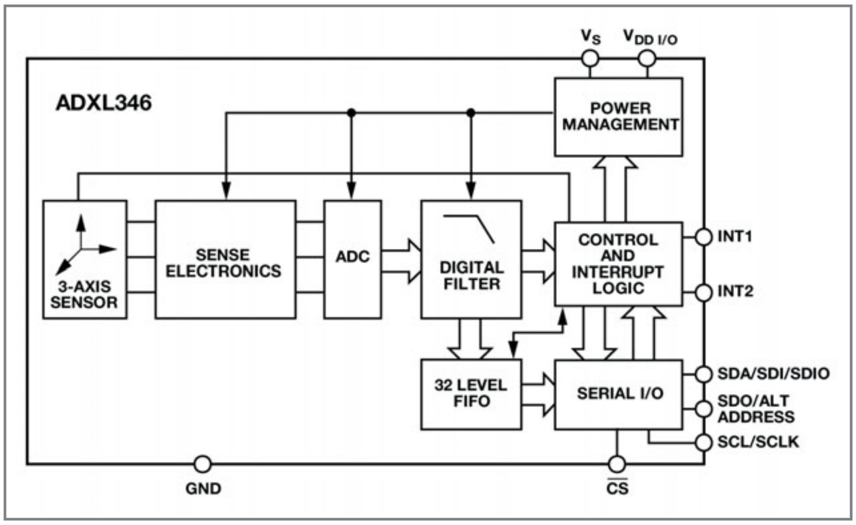
The RXM-GPS-SG and RXM-GPS-SR modules are priced at **$24.95** and **$34.95**, respectively, in 200-piece quantities. MDEV-GPS Master Development Systems are priced at **$249** for the SG version and **$289** for the SR version.

Linx Technologies
www.linxtechnologies.com

HIGH-RES MEMS MOTION SENSOR

The **ADXL346** digital three-axis iMEMS smart motion sensor is the latest offering in a new class of small, ultra-lower-power, high-resolution iMEMS accelerometers for measuring tilt, shock, and acceleration. Designed to operate at a primary supply voltage down to 1.8 V, this new motion sensor is capable of high resolution (4 mg/LSB).

The device automatically modulates its power consumption in proportion to its output data rate as well as saves additional power by automatically switching to a sleep mode during periods of inactivity. For even more power savings, a standby mode can also be used.



The ADXL346 has a wide, selectable bandwidth of 0.1 to 1,600 Hz. Power consumption ranges from less than 150 µA at 1600 Hz bandwidth down to 25 µA under 10 Hz. The device also measures dynamic acceleration with ±2g/4g/8g/16g user-selectable measurement ranges.

Several special sensing functions are also programmed on-chip along with user-programmable threshold levels. Activity and inactivity sensing detect the presence of or lack of motion or if the acceleration on any axis exceeds a user-set level. A tap-sensing function detects single and double taps. Free-fall sensing detects if the device is falling. These functions can be mapped to one of two interrupt output pins. In addition, the new accelerometer includes I^{2}C and three- and four-wire SPI digital interfaces.

The ADXL346 costs **$3.04** per unit in 1,000-unit quantities.

Analog Devices, Inc.
www.analog.com

CIRCUIT CELLAR

Test Your EQ

Edited by David Tweed

Problem 1—*Given the FIFO data structure defined below, can you write thread-safe functions to initialize the FIFO, add an item to it, remove an item from it, and find out how many items are in the FIFO? How many items can you put into this FIFO?*

```
#define FIFO_SIZE 100
typedef struct {
        unsigned int head;
        unsigned int tail;
        int data[FIFO_SIZE];
} FIFO;

void fifo_init (FIFO *f);
unsigned int fifo_count (FIFO *f);
void fifo_add (FIFO *f, int item);
int fifo_remove (FIFO *f);
```

Problem 2—*Suppose you add a "full" flag to the data structure, as shown below. How many items can you store in the FIFO now, and what do the thread-safe access functions look like?*

```
typedef struct {
    unsigned int head;
    unsigned int tail;
    bool full;
    int data[FIFO_SIZE];
} FIFO;
```

Problem 3—*To a first approximation, what is the voltage gain of a common-emitter or common-base transistor amplifier?*

Problem 4—*What is the voltage gain of a common-collector transistor amplifier?*

Contributed by David Tweed

What's your EQ?—The answers are posted at www.circuitcellar.com/eq/
You may contact the quizmasters at eq@circuitcellar.com

PCBCORE

China PCB Supplier
(Prototype thru Production)

✓ 1-layer up to 30-layer
✓ Cost and quality
✓ On time delivery
✓ Dedicated service
✓ Instant Online Quote & Order
..... Day and Night

No minimum quantity - 1 piece is welcome
Check our low price and save big $$$...

86(571)86795686 sales@pcbcore.com

www.pcbcore.com

RF Specialists

RF Modules
From Part 15 to Part 90 Compliant
Narrow Band FM, UHF Multi-Channel

GSM/GPRS
M2M Solutions
GSM/GPRS modules and modem series

Industrial Bluetooth
OEM, Modules, Wireless Device Servers, RS-232
Long range options, low cost

Data Loggers
Stand Alone and
Wireless Mesh Networking Logger

GPS
OEM Modules and USB ZigBee Sticks,
Mesh Networks

ZigBee Pro
OEM Modules and USB ZigBee Sticks,
Mesh Networks

Wi-Fi
RS232 / 422 / 485 to Wi-Fi Adapter
Connect Data Acquisition Equipment through Serial Port to Wi-Fi network

LEMOS INTERNATIONAL
www.lemosint.com
866.345.3667
sales@lemosint.com

FEATURE ARTICLE

by Chuck Baird

Timer Development

From Timing Cycles to System Programming

Need a stand-alone timer or a timepiece for a lar ge application? Read on to learn how to develop the right timer design to suit your needs. It' s time to put your hardware and programming skills to good use.

As every parent knows, a young child occasionally needs to take a break from terrorizing the world and sit quietly to settle down. This serves two purposes: the kid gets a cooling off period and the parents don't go to jail for acting on their impulses. Most experts agree that a timeout should be presented not as punishment but rather as quiet time. In fact, many recommend timeout periods of 1 minute per year of a child's age.

In this article, I'll describe a simple timeout timer with a user interface featuring one push button, eight LEDs, and audio capability for marking certain events (see Photo 1). If you don't need a design for timing timeouts, you can look at this project in a different way. You can use this timer design project to experiment with several software techniques and to work with a versatile yet inexpensive development system.

OPERATION

The timer design is easy to operate. You first turn on the power and then press and release the push button. This initiates a repeating sequence of slowly lighting one to eight LEDs. When the number of illuminated LEDs matches the number desired "timeout" minutes, you press the button again. When you release the button, the timing cycle for the chosen number of minutes begins. All eight LEDs then light up. The rightmost LED flashes, and it will do so faster and faster until it finally goes out. The process then repeats using seven LEDs, then six, and so on. When all the LEDs have turned off, the system plays a tune and the timeout period is over.

If the time chosen is 1 minute, each LED goes through its flashing sequence in 7.5 s for a total of 60 s. At the other extreme, if the time chosen is 8 minutes, each LED's flashing sequence is 1 minute long. (In theory, an errant child will be diverted by watching the changing rate of flashing LEDs. If you aren't using the design for

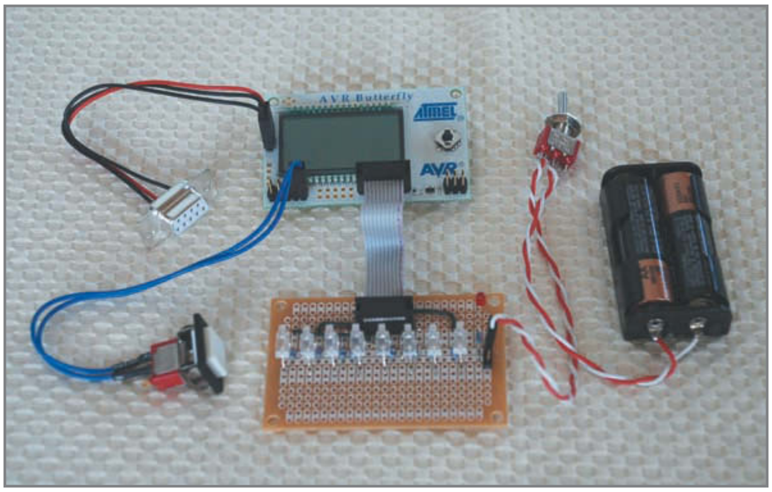


Photo 1—Here is the prototype using headers soldered onto the Butterfly. The LEDs and resistors are mounted on the perf board, with power routed from it to the Butterfly.

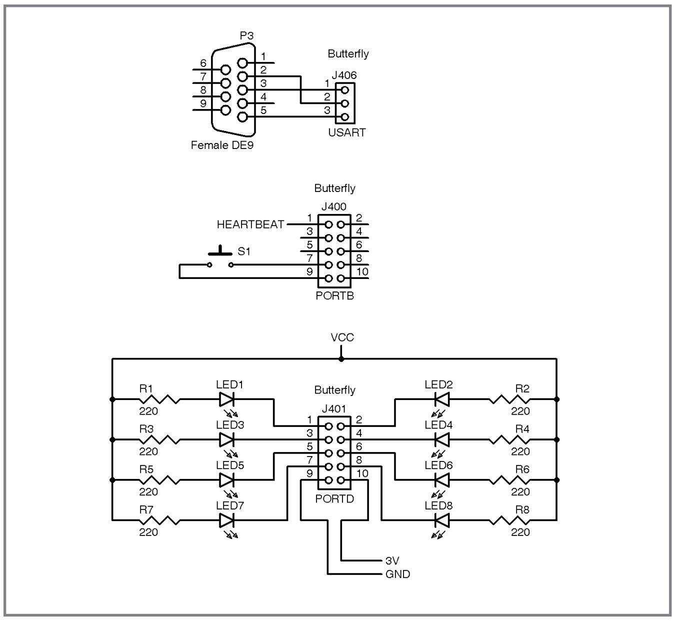


Figure 1—This schematic shows the connection of components external to the Butterfly. The Butterfly's schematic is available on the Atmel web site.

childcare purposes, perhaps your LEDs will be useful for something else, like interval indication.)

As a reminder to turn off the power, the system beeps four times every 15 s when the timer is not in use but left on. Although the design has a low-power Sleep mode, I assumed that this device might sit unused for weeks and that zero power consumption would be better than a Sleep mode. If you press the button during the timeout period, the MCU resets and the process repeats.

MCU OVERKILL?

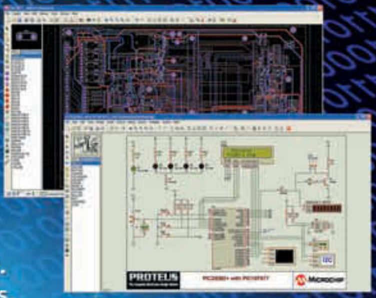
I built the timer around an Atmel AVR Butterfly demonstration and evaluation kit (see Figure 1). You might think the Butterfly is extreme overkill for this design, but it works for me. The kit has an 8-MHz ATmega169 processor, a six-character LCD, a piezo element for noise making, a 4-Mb dataflash memory, a joystick, a temperature sensor, 16-KB flash memory, 1-KB SRAM, and 512 bytes of EEPROM. It also has RS-232 level converters and is shipped with a bootloader installed that enables its flash memory to be programmed via a Windows serial port without additional hardware other than a DE9 connector. The free programming software includes an assembler and a full-featured C compiler (WinAVR).

With such an array of peripherals, most of the Butterfly's general I/O pins serve dual purposes (or more). One pin is needed for the push button input, eight pins are needed for the LEDs, and one pin is needed for the piezo element mounted on the Butterfly. In addition, I use an I/O output as an interrupt "heartbeat." Doing so enables me to use an oscilloscope to monitor and verify the

SIMULATE YOUR DESIGN AND SAVE TIME

PROTEUS: the *ONLY* EDA software to simulate micros from TI, Microchip, Atmel, Intel, ARM7 and More!

PROTEUS DESIGN SUITE FEATURES:



- -Easy to use, standard Windows interface.
- -Publication quality schematics.
- -Fully configurable bill of materials.
- -Large component libraries for schematic simulation and PCB layout.
- -Mixed mode SPICE circuit simulation.
- -Co-simulation of PIC, AVR, 8051, ARM7 & BasicStamp microcontroller firmware.
- -Integrates with MP-LAB and AVR Studio.
- -Automatic component placement and gateswap optimization.
- -Highly configurable design rules.
- -Interactive design rule checking.
- -Polygonal and split power planes.
- -World class shape based autorouter.
- -Automatic mitre/unmitre commands.
- -Industry standard CADCAM & ODB++ output.
- -Integrated 3D Viewer with 3DS and DXF export.

R4 Systems Inc.

All levels of the Proteus Design Suite include a world class, fully integrated shape-based autorouter at no additional cost - prices start from just $249.00!

www.r4systems.com * Toll-Free 866.499.8184 * info@r4systems.com

KEIL
An ARM® Company

Development Solutions for ARM, 8051 & XE166 Microcontrollers

Microcontroller Development Kits

- C and C++ Compilers
- Royalty-Free RTX Kernel
- µVision Device Database & IDE
- µVision Debugger
- Complete Device Simulation

Examples and Templates

Keil PK51, PK166, and MDK-ARM
support more than 1,700 microcontrollers
www.keil.com/dd

RTOS and Middleware Components

- RTX Kernel Source Code
- TCPnet Networking Suite
- Flash File System
- USB Device Interface
- CAN Interface

Examples and Templates

Keil RL-ARM and ARTX-166
highly optimised, royalty-free middleware suites
www.keil.com/rtos

www.keil.com
1-800-348-8051

timebase, although this can be omitted. I sacrifice some of the LCD lines to get these eleven I/O pins, but I'm not using them for this project so it isn't a problem. The lines shared with the LCD are only driven low or tristated, so there is no danger of damaging the LCD.

The Butterfly's bootloader enables it to be programmed (read, write, and verify flash memory) through its serial port. You can download the Atmel AVR Studio Windows programming software for free. Your PC will need a serial port or a USB-to-serial converter. Additional hardware isn't needed, so for the purpose of this article, assume that this is the programming method used. However, the Butterfly can be programmed in a variety of other ways with a separate hardware programmer. These alternate methods usually erase the bootloader, but both the bootloader and the application that ships with the Butterfly are available as hex files on the Atmel web site and may be reloaded. The bootloader can't modify itself, so it should remain intact unless you use one of the other programming methods.

This project involves wiring a DE9 to the Butterfly's RS-232 pins for use by the bootloader, connecting LEDs and resistors to eight of the output pins, and adding a push button to ground to another I/O pin. I used two AA batteries to supply the 3-V power because the Butterfly's coin battery isn't quite up to this job. There is a switch on the batteries, and you may want to add an LED and resistor for a visual reminder. We then burn the hex file from this program into the Butterfly's flash memory, typically by using the Butterfly's bootloader and AVR Studio.

I used ImageCraft's C (ICCAVR) to write the code for this project. Other C compilers will have slight syntactical differences, but the results are essentially the same. The ICCAVR convention that might be of concern to the casual reader is that `char` declarations are unsigned.

Besides the free WinAVR GCC port, the various commercial C compilers for the Atmel AVRs have free demo and trial versions for noncommercial use if you want to modify the source code. To create your own timer using the compiled version, the inexpensive Butterfly and a few common parts are all you need.

You may want to download the Butterfly's documentation and datasheet for the ATmega169 from Atmel's web site. As I mentioned, you'll need to download and install the free AVR Studio software to communicate with the bootloader, unless you plan to use other software tools.

BOOTLOADER

A few things complicate this project. I also have plenty of computing resources readily available, so I'll add to the complication for educational purposes by presenting interesting coding techniques.

The first consideration is that you may or may not be running code that was loaded via the Butterfly's bootloader. If the bootloader isn't used, the power-on reset jumps directly to your code. If the bootloader is used, the power-on reset jumps to the bootloader, which then waits to determine if it should accept input for burning flash, jump to the application (your program), or go to sleep.

The Up and In joystick switches wake the bootloader out of Sleep mode, and it jumps to the application if it sees the joystick pressed up. Therefore, you will wire your push button in parallel with the Up joystick button. That will cause the initial button push to make the bootloader run your code (in case the bootloader is used). In that case, pressing the button once will simply start your program. (It plays a start-up tune.) A second press starts the process of selecting the timeout minutes. If you use a programming method other than the Butterfly bootloader, the program (with its start-up tune) will run immediately after power-up or reset.

Another issue is that the bootloader calibrates the microcontroller's speed when it starts up. The Butterfly's ATmega169 runs off its internal R/C oscillator, which AVRs can fine-tune by adjusting a register. The bootloader tweaks the ATmega169's speed in comparison to the Butterfly's 32-kHz clock crystal, thus ensuring its RS-232 communications timing will be as accurate as possible. This adjustment won't be performed if you bypass the bootloader—although it does not matter for this application because exact timing isn't necessary, except for the bootloader's serial communications.

The bootloader leaves the ATmega169 running at 2 MHz, which you will bump to 8 MHz. It also leaves a few other little messes (at least in some versions of the bootloader), which you must clean up.

TIMING IS EVERYTHING

You are building a timer, so you need to have some sort of a timebase running. You can use one of the ATmega169's 8-bit timers to generate an interrupt once every millisecond and then derive your various timing needs from that. As I already mentioned, exact timing isn't necessary for this application. It doesn't matter if a 6-minute timeout actually lasts 6.12 or 5.94 minutes. Not that you will be far off, but you won't worry about running off an R/C clock or whether it has been calibrated. Most of the timing considerations are handled inside the 1-ms interrupt handler.

One thing you'll do is multiplex the LEDs (whether you need to or not). This will cause them to appear dimmer, but it will limit the current drawn. (The main reason for doing it in this case is to demonstrate one method of multiplexing outputs.) The program will view the LEDs as a single unsigned byte of 8 bits, and if a bit is set, its corresponding LED will illuminate. The mechanism behind the scenes will be a second mask byte with exactly 2 bits set. On each interrupt, the mask and the main byte are ANDed together to determine the outputs. The mask bits are then shifted circularly in preparation for the next interrupt. Thus, each LED is serviced every four interrupts.

The interrupt handler also handles push button debouncing. After a certain number of interrupts (currently two), the push button is read and its single bit value is shifted into an unsigned byte variable. It takes eight shifts to "fill" the variable, so checking that variable (at any given

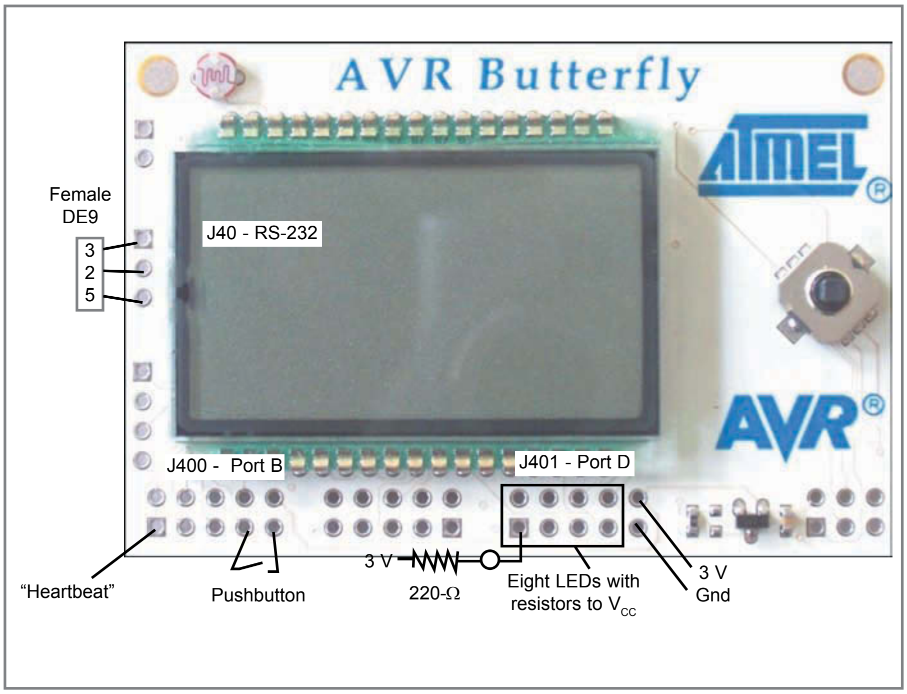
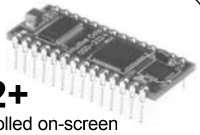
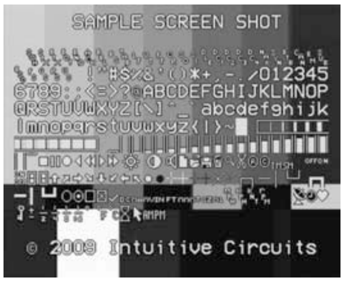


Photo 2—The components attach to the Butterfly's pads. They may be either directly soldered or plugged on using standard 0.1" headers.

New!



OSD-232+
RS-232/TTL controlled on-screen composite video character and graphic overlay in a small 28 pin dip package.



Intuitive Circuits
www.icircuits.com
(248) 588-4400

Celebrating Our 10th Year Anniversary

Electronic Kits 20% OFF For The Month of July!

Kits For: Complete with instructional manuals.
Your HOME
Your CAR
Your PHONE
Your MUSIC
SCHOOL PROJECTS

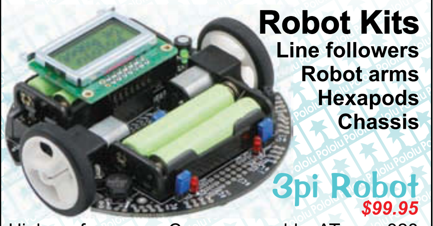
Give as GIFTS! Make for FUN!

DesignNotes.com
What Your Electronic Hobby Stores Used To Be
1-800-957-6867 www.DesignNotes.com

USB Oscilloscope for $169.50
Logic and Spectrum Analyzers, Generator.
www.HobbyLab.us

Pololu
Robotics & Electronics

Robot Kits
Line followers
Robot arms
Hexapods
Chassis

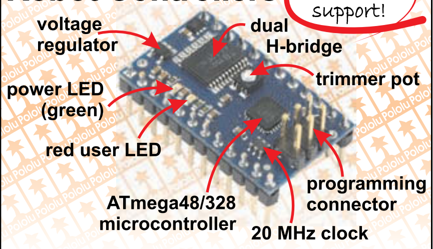


3pi Robot
$99.95
High-performance, C-programmable, ATmega328-based robot (with Arduino support)!

Mechanical Components
Motors, servos
Wheels, ball casters

Motion Control
Motor controllers
Servo controllers

Robot Controllers with Arduino support!



Solder Paste Stencils
Use our low-cost solder paste stencils to quickly assemble your surface-mount designs.
From $25

Custom Laser Cutting
From $25

Cut your own custom chassis, front panels, and more!
1-877-7-POLOLU
www.pololu.com
6000 S. Eastern Ave. 12D, Las Vegas, NV 89119

“The ATmega169’s 16-bit timer is used in one of its PWM modes to generate each note. Conveniently, its PWM output bit on the Butterfly is tied to the piezo element. All you have to do is feed the timer a PWM frequency count to start each note at the correct time. The PWM duty cycle determines the volume (more or less).”

point in time) gives the value of the last eight reads. With the shift happening after every two interrupts, you get a debounce time of 16 ms, which is probably considerably longer than necessary. If the variable equals zero, the push button is not being pushed and it is debounced. If the variable is 255 (0xff), the push button is pushed and debounced. Any other value means it is in transition.

The programming “gotcha” associated with the debouncing is that the push button’s variable must be declared volatile because its value changes in the interrupt handler. This is something the compiler won’t normally expect, so it needs to be instructed to reload the variable for each use. Otherwise, the optimizer’s tough love could kill you—or at least your code.

The heartbeat output isn’t really a necessary part of this project, but sometimes it’s useful to be able to verify that the interrupts are working, that the timer setup is correct, and that the clock rate is right. On every interrupt, you toggle an output bit, so an oscilloscope should show a square wave with a half period of 1 ms.

Finally, note that the interrupt handler handles sound generation. I’ll explain this in greater detail in an upcoming section of this article.

THE MAIN EVENT

An event list is implemented for general timing needs. An event in this program consists of a function that will be executed in the future; it’s measured from the present time. The function is called when the time for the event arrives.

Event times are specified in tenths of a second. Every 100 interrupts, the interrupt handler updates the active events’ pending times, eventually decrementing them to zero. When the main task sees an event with zero time remaining, it removes the event from the list and executes the event’s function. For example, when the program starts, it sets an event for 1 minute, or 600 time units. If this event executes, it starts the beeping as a power-down reminder—sort of the poor person’s watchdog timer interrupt. In the meantime, other events are loaded and executed that display the initial sequence of lights so you can select a timeout value. When and if the selection is made, the shutdown event is cancelled.

The event list is modified by both the interrupt handler and the main task, so a locking mechanism is necessary to prevent conflicts. The main program sets and later clears a variable to indicate when the event list is busy. If an interrupt needs to access the locked list, it sets a flag for itself and performs the action on the first interrupt following the unlocking of the list. Although there are numerous ways to implement the timing needs for this project, the event list is versatile, interesting, and effective. It also simplifies the interrupt handler’s ability to manage pending tasks.

SOUNDS & IMPLEMENTATION

Finally, there is the matter of sound generation. As written, the sound routines can play just about anything,

Photo 3—This is one possible mounting for the timer. I used a small plastic case from RadioShack. The Power button is on top. The Input button is on the front of case.

although making a few simple sounds suffices for this project. I implemented the code in a more complete form to allow reuse and to demonstrate the general principles.

The technique is fairly straightforward. A sound or song is encoded as a series of notes, each identified by common notation. The tempo, note duration, volume, and repeat count can be included with and within each song.

The ATmega169's 16-bit timer is used in one of its PWM modes to generate each note. Conveniently, its PWM output bit on the Butterfly is tied to the piezo element. All you have to do is feed the timer a PWM frequency count to start each note at the correct time. The PWM duty cycle determines the volume (more or less).

To generate a sound, the main task supplies the song definition's address and sets a flag. The interrupt handler does the rest. This involves determining the next note, starting the PWM output at the appropriate frequency, and sustaining the note for its duration across subsequent interrupts. After that, it plays the next note, repeats the song, or shuts down the sound mechanism as needed. In other words, once a song starts, it plays to completion with no further intervention outside the interrupt handler.

My application (written in ImageCraft C) is split into three source files and two header files. There are header and source files for the event list and header and source files for the music functions. Everything else is in the main source file. The processor-dependent definitions are in an ICCAVR include file, as are

FlashPro430
FlashPro-CC
FlashPro2000
GangPro430
GangPro-CC

USB Flash Programmers for Texas Instruments' MCUs
MSP430, Chipcon CCxx, C2000 DSPs

Reliable and the fastest programmer on the market.
Perfect for production usage.

* can assign unique serial number
* up to eight programmers can be connected to one PC and program target devices simultaneously

Elprotronic Incorporated
www.elprotronic.com

COMPACT EMBEDDED SERVER

- Fanless x86 1 GHz CPU
- 256MB DDR2 RAM On Board
- 128MB Internal Flash Disk
- 10/100 Base-T Ethernet
- Reliable (No CPU Fan or Disk Drive)
- Two RS-232 & Three USB 2.0 Ports
- On Board Audio
- Optional Wireless LAN & Hard Drive
- Dimensions: 4.5 x 4.5 x 1.375" (115 x 115 x 35mm)

2.6 KERNEL

Microsoft Windows CE

Compact SIB
(Server-In-a-Box)
Starting at $230.00
Quantity 1.

- Compact Flash & Micro SD Slots
- Analog SVGA Video
- Extended Temperature Range
- PS/2 Keyboard & Mouse Port
- EMAC Linux 2.6 Kernel & WinCE 6.0

www.emacinc.com/servers/compact_sib.htm

Since 1985
OVER
24
YEARS OF
SINGLE BOARD
SOLUTIONS

EMAC, inc.
EQUIPMENT MONITOR AND CONTROL

Phone: (618) 529-4525 • Fax: (618) 457-0110 • www.emacinc.com some macros.

I derived the frequency definitions for sound generation from the Butterfly's demo application, although all of this project's code is original. Atmel has published the source code for its demo. If you are interested, you can learn a lot by studying it.

WIRING THE BUTTERFLY

Photo 1 shows my prototype with the DE9 wired to a header so it can be removed after programming. The red LED shows that the power is on. Refer to Photo 2 to see the connections for the Butterfly. You can solder leads directly to the board or you can use standard 0.1″ headers. The 3-V supply is switched with an optional LED and resistor to ground, and the Butterfly's coin battery is removed. The actual port and pin assignments are shown in the comments listed in the code. The resistor sizes vary depending on the LEDs, although 220 Ω is typical.

As for your timer, anything goes. Photo 3 shows a rectangular plastic case. The square button at the top is for power. The round button is the input.

Because my design was originally intended to be a timeout timer, I

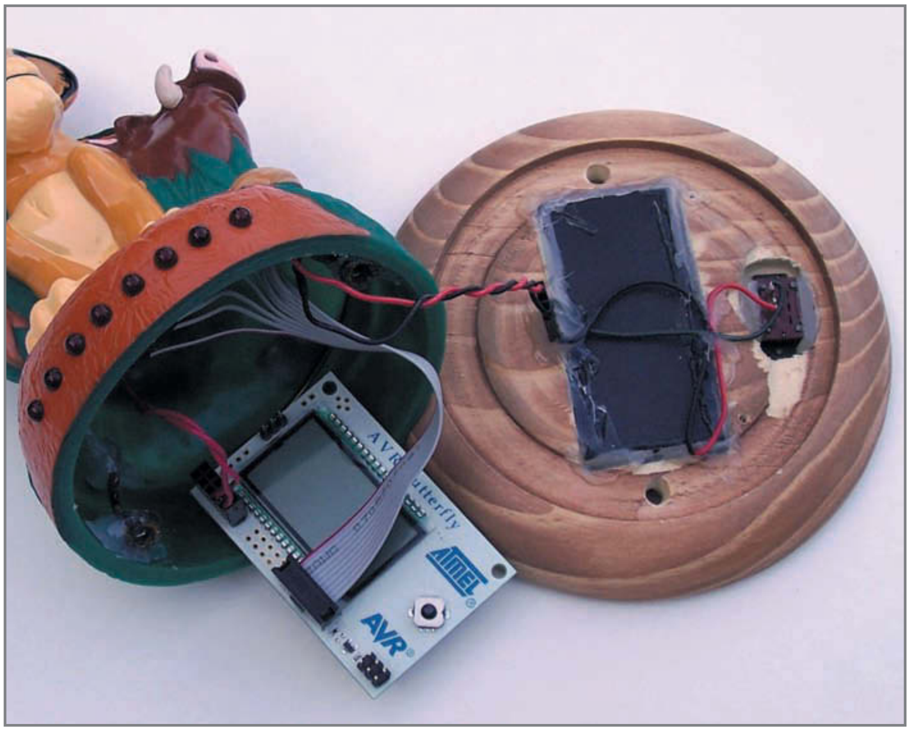
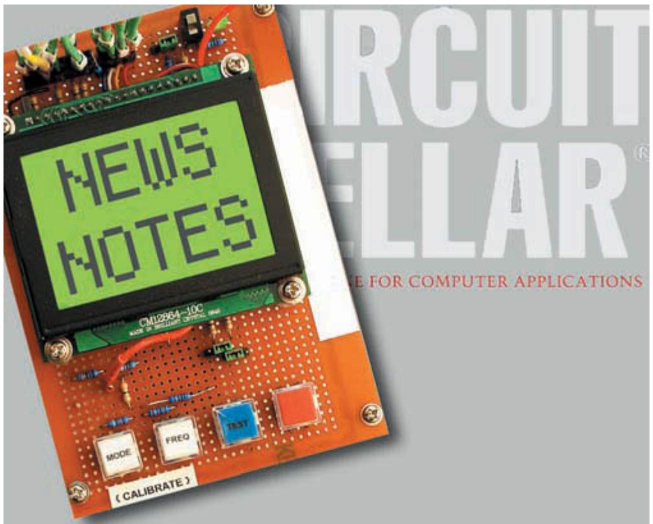


Photo 4—Take a look inside my timer. You may not need a "timeout" timer, but you can use the basic techniques I cover in this article to make a timer to suit your needs.

Robowaiter
Saturday, April 10
Trinity College
Hartford, CT USA
Robots helping humans gain independence
Cash Prizes and Awards
Additional $500 cash prize for best Connecticut Robot
Learn More. Volunteer. Register.
www.trincoll.edu/events/robot
Connecticut Council on Developmental Disabilities



made a case out of the soft plastic lid on a cartoon cup. The finished result is shown in Photo 4. The push button is mounted in the back in a hole where the cup's straw once protruded. The power switch is on the bottom.

In any mounting, make sure the piezo element's sounds are audible. One option is to relocate it from the back of the Butterfly to a more audible position; but otherwise, you may need to creatively locate the Butterfly within your case.

TIME TO BUILD

Sure, using a Butterfly for this application may seem like overkill, but there are some noticeable advantages to doing so: minimal hardware construction, no additional programming hardware, free development tools, and affordability. These benefits certainly make up for any wasted resources. You will find, as I did, that the Butterfly is an impressive board with great potential. I hope you enjoy building your own Butterfly-based timer.

Author's note: Special thanks to Jennifer Flory for helping with the photos and Carl Livingston for assisting with the schematic.

Chuck Baird started in programming by learning FORTRAN for a numerical analysis course in college. Forty-three years later, he still enjoys bit fiddling and problem solving. Chuck's current computing interests lie in the embedded world. He has authored two books about programming Atmel AVRs. You may contact Chuck with a private message at www.avrfreaks.net. His username is zbaird.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/230.

RESOURCES

AVR User forum, www.avrfreaks.net.

Butterfly information and support materials, Smiley Micros, www.smileymicros.com.

SOURCES

AVR Studio IDE and Butterfly development tool
Atmel Corp. | www.atmel.com

ICCAVR C Compiler
Imagecraft Creations, Inc. | www.imagecraft.com

Embedded & Network Computing Technologies
CALAO SYSTEMS
Imagine YOUR systems with OUR ready-to-use boards!
Embedded Computer TNY-A9G20 for data processing.
Daughter board DAB GPRS for data transmission. (Pass-through technology)
Daughter board DAB GPS for data acquisition.
Now, with pass-through technology, you can plug Daughter boards one on top of the other.
www.calao-systems.com

Surface Mount Probing Adapters
In-Circuit test Your IC
• BGA, QFN, QFP, PLCC, & SOIC
• SMT adapters solder to target board
• Modular probe boards, controlled Impedance Flex (plug & test)
• High quality ZIF sockets
• Gold plated test points
• 1000's of standard products
• Quick-Turn Custom Solutions
RoHS
Ironwood ELECTRONICS
1-800-404-0204
www.ironwoodelectronics.com

by Carlo Tauraso

Very Low-Frequency Datalogger

If you're interested in radio emissions, you can build a very low-fr equency (VLF) datalogger at your workbench. This PIC18LF4620-based design features a 16-bit datalogger, an SD card for data storage, a customizable hexagonal loop antenna, and circuit-contr olled battery power.

The topic of very low-frequency (VLF) radio emissions has fascinated me for some time now. In 1989, NASA launched a project called the Interdisciplinary National Science Program Incorporating Research and Education Experience (INSPIRE) with two purposes in mind: teach students about VLF radio emissions testing and create working relationships between professional and amateur researchers. After reading about the history and evolution of the project—which continues to involve thousands of researchers in the United States and elsewhere—I recently designed a Microchip Technology PIC18LF4620-based circuit capable of monitoring ultra-low frequency (ULF, 0 to 120 Hz) radio emissions (see Photo 1 and Figure 1). The design features a 16-bit datalogger with an SD card for data storage, a customizable hexagonal loop antenna, and circuit-controlled battery power.

In this article, I'll describe how I designed and built the system. Despite being an experimental design, I'm sure you will find it interesting and quite easy to implement. For instance, when used in an urban environment, you can easily analyze manmade radio emissions. When used in a rural environment, you can record natural radio emissions.

SCHUMANN RESONANCE

I am interested in one radio emission in particular, the Schumann resonance, which is an electromagnetic resonance associated with our planet that has become

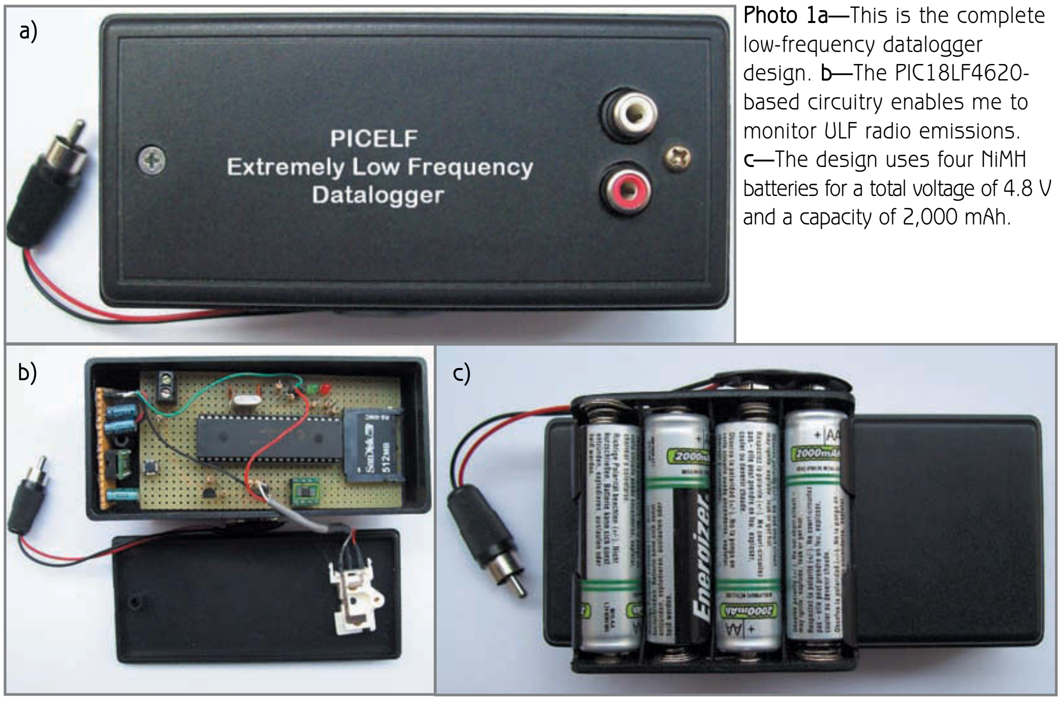


Photo 1a—This is the complete low-frequency datalogger design. b—The PIC18LF4620-based circuitry enables me to monitor ULF radio emissions. c—The design uses four NiMH batteries for a total voltage of 4.8 V and a capacity of 2,000 mAh.

Stellaris®

Connected. Versatile. Cost-effective.

Stellaris LM3S9B90 Ethernet Evaluation Kit

featuring USB, Ethernet and CAN

$99

Stellaris LM3S8962 Evaluation Kit

featuring integrated Ethernet and CAN

$89

Stellaris LM3S3748 Evaluation Kit

featuring USB Host/Device

$109

Stellaris® Means:

- Deterministic ARM® Cortex™-M3 cores
- Real-time multi-tasking MCUs
- Advanced interrupt-driven software with Stellarisware® software

Stellaris LM3S6965 Ethernet Evaluation Kit

demonstrating an embedded web server out-of-the-box

$69

Stellaris LM3S811 Evaluation Kit

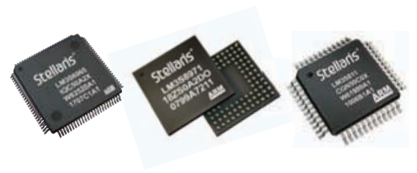
featuring basic embedded control

$49

Stellaris LM3S2965 CAN Evaluation Kit

demonstrating a CAN network out-of-the-box

$79



ARM POWERED

Cortex Intelligent Processors by ARM

Texas Instruments

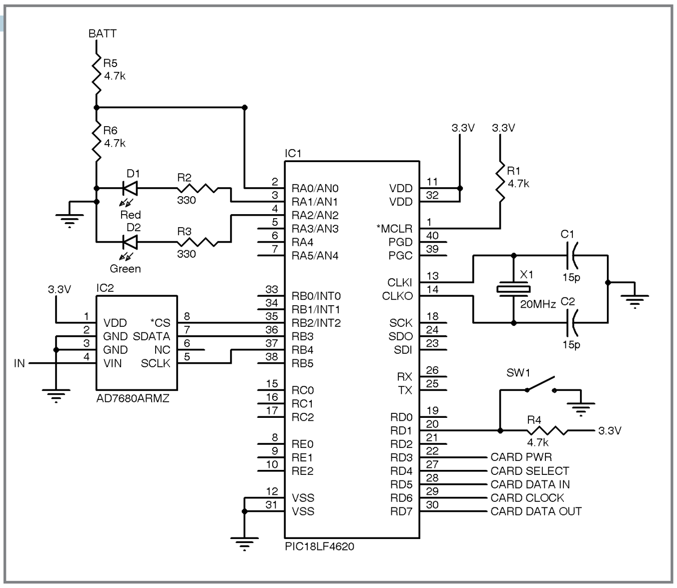


Figure 1—The circuit is simple. This is the core section. Be careful when soldering the Analog Devices AD7680 ADC.

increasingly difficult to hear because of the increase in electromagnetic pollution. In 1952, a German researcher, Winfried Otto Schumann, claimed that the zone between the ionosphere and the Earth's surface works as a waveguide and it contains a pulsed magnetic field that's continuously fed by electrostatic discharge from lightning. The planet is surrounded by a pulsed magnetic field whose frequency remains at about 7.8 Hz. There are also four fundamental harmonic frequencies: 14, 21, 26, and 33 Hz. These low-frequency signals, however, have been masked by manmade electromagnetic emissions. Electrical devices, engines, and power lines have "covered" the natural frequencies so much so that you have to leave populated areas in order to detect them. This can be difficult.

Nikola Tesla presented the first observations of resonance phenomena in his fabulous article, "The Transmission of Electrical Energy Without Wires as a Means of Furthering World Peace" (*Electrical World and Engineer*, January 7, 1905). The alternator inventor idealized a world in which energy could be transmitted freely—based on these phenomena—across the Earth's surface. More recently, in the 1960s the existence of Schumann resonance was demonstrated thanks to measurements taken by two researchers, M. Balser and C. A. Wagner. During some Cold War years, they also investigated the effects of high-altitude nuclear explosions on the ionosphere. The relevance of this research has led to a better understanding of meteorological occurrences, transient luminous events (TLEs), and global temperature changes. On the lighter side of things, the topic of Schumann resonance also plays a role in a PlayStation video game called "Serial Experiments Lain," which involves a young female character named Lain Iwakura and the evolution of the Internet called "Wired" that uses global communication protocols.

DESIGN & CONSTRUCTION

The heart of the circuit is a PIC18LF4620 with a 3.3-V power supply. This prevents signal-level conversion between microcontroller and SD card (see Figure 1). The PIC18LF4620 integrates a 10-bit ADC; but in order to not excessively restrict the recorded signal, I inserted an Analog Devices AD7680, which is a 16-bit, low-power, successive approximation ADC. The ADC can achieve the extraordinary sampling speed of 100 ksps with the SPI bus. Obviously, recording signals in the frequency range between 0 and 120 Hz does not require so much speed. For the Shannon theorem, a 256-sps sampling is more than enough. For communications purposes, I used the PIC18LF4620's PORTB and the SHIFTIN/SHIFTOUT PICBASIC instructions. The only problem was the AD7680's packaging: it's distributed only in SOT-23 or MSOP-8 packages. I chose the latter because it was easy to find an MSOP-8-to-eight-pin DIP adapter (www.cimarrontechnology.com).

For monitoring the battery state, I used the 10-bit ADC integrated in the PIC18LF4620. The battery voltage is applied to a voltage divider consisting of two resistors (R5 and R6). Here it's divided by two and transferred to the input line RA0. The firmware performs the `BATT` sub-procedure to verify that the halved voltage will not fall below the 2.05-V threshold, below which the four batteries are unable to ensure the proper writing on the SD card (see Figure 2). After the system detects a voltage

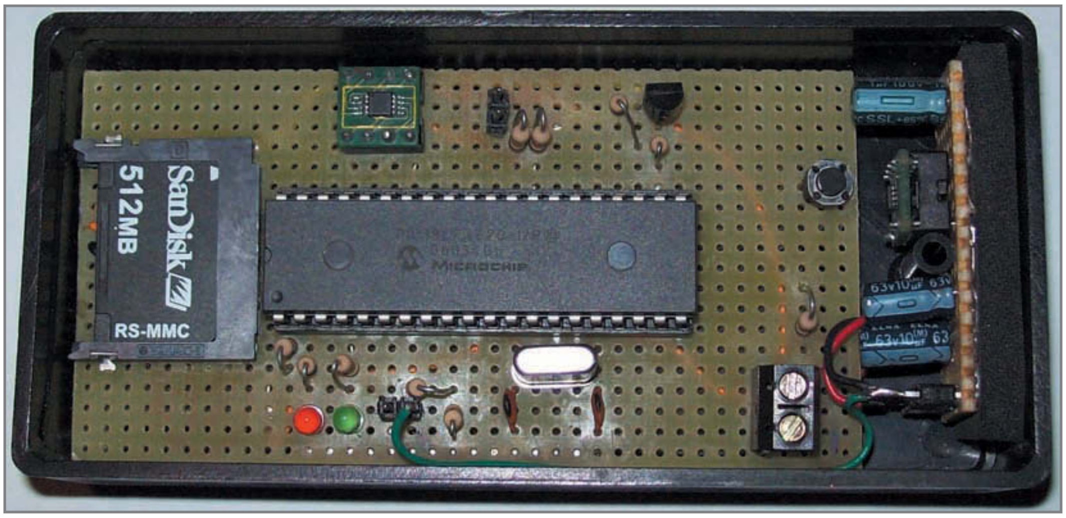


Photo 2—I created the circuit on two prototype boards. The core is on the left. The battery voltage regulator is on the right.

Saelig

UNIQUE PRODUCTS & SUPPORT

www.saelig.com

Testgear

Color LCD Scopes

2-ch + trigger standalone USB bench scope. $325 / $599

PDS5022S / PDS6062T

2-ch 1GSa/s Scopes

2-ch 1GSa/s (25GSa/s equiv.) 50/100 MHz scope. $595 / $795

DS1000E

Scope + Analyzer

25MHz 2-ch /16 logic scope and logic analyzer. $699

DS1022CD

August Offer while supplies last!

Buy any Rigol scope and get a FREE ATSC digital TV!

7" TFT LCD ATSC 16:9 Digital TV with NTSC in, A/V O/P, 480 x 234 pixel, 1W stereo audio O/P + jack, inc. DC12V supply.

You must ask for offer# CC8

AT-70T

FREE COFFEE

Call 1-888-772-3544 to get a free Starbucks Card with your >$50 order!

While supplies last - not available with any other offers

Mention offer# SBW

Handheld Scopes

20MHz / 60MHz rugged handheld USB 2-ch scope. $593 / $699

HDS1022MN / HDS2062M

Pen Scopes

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

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 20MS/sec 2MHz oscilloscope, 2-ch spectrum-analyzer, 3MHz 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. What's more – its' open-source software runs with Windows, Linux and Mac OS's! **Only $180!**

"I really like this scope adapter - it's meant for teaching electronic experiments but it's ideal for engineers too."

Alan Lowne
Saelig CEO

CGR-101

www.saelig.com

Low-Cost Scopes

2-ch 40/100/200MS/s 8-bit scope range with 5/10/25MHz. $297 +

PS2203/4/5

Scope/Logger

2-ch DSO 16-bit DSO, FFT, VM, logic analyzer, standalone + 24 I/O.

Mephisto

Mixed-Signal Scopes

100MHz Scope, + Spectrum/Logic Analyzer and Signal Generator. $1259+

CS328

USB Bus Analyzers

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

LAD-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.

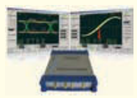
EMC RF & EMC Spectrum

EMC Spectrum Analyzer

Handheld Palm PC-based 2.7GHz Spectrum Analyzer.

PSA2701T

Worlds's Fastest



2-ch 12GHz sampling scope for high-speed electrical signals.

PS9200

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

CANminiBOX

Embedded controller series: 2 x CANbus, Ehternet, USB2.0, CF.

emPC-x133

RF Generator

High-res, extremely low-noise, portable 6GHz RF generator.

APSIN6000

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 Loggers

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

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 I^{2}C/SMBUS + 32 I/O lines. $89

USBI2CIO

FTDI USB ICs

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

USB-Serial

A complete CP2102 USB-serial converter in a DB9 shell. $26

CE-USB

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 $50 order! Check ***www.saelig.com*** *often for special offers, bargains, business hints, blog, etc.*

below 2.05 V, the PIC18LF4620 closes the file and disables the card. As a result, all data stored to that point is safe. Pressing switch SW1 for a few seconds stops the recording and forces the file to close. You can use the SW1 to stop recording at any time or the PIC18LF4620 can do it automatically when it detects a low battery. If you use a different battery, you can change the threshold for PIC18LF4620 intervention after the call to the BATT subprocedure. The limit for this project is 637. (Remember that the supply voltage is 3.3 V.)

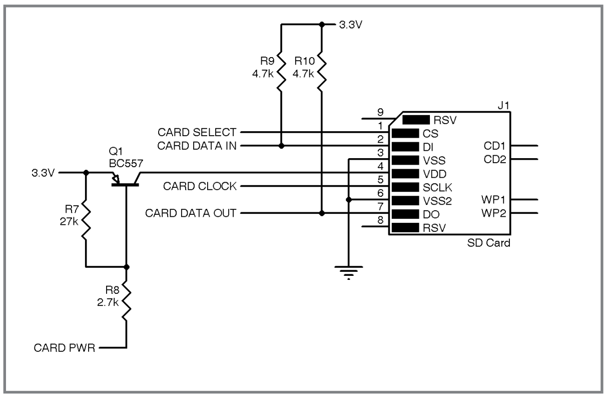


Figure 2—This is the SD card section. The Q1 transistor is essential for SD card supply control.

I used a 512-MB miniSD to store the samples. You can, of course, insert cards of different sizes, but the important thing is that they're formatted to the FAT16 format. This file system is useful. When done recording, you can remove the card, insert it in a PC card drive, and analyze the data directly through a spectrogram.

All the samples are written in a raw binary file (VLF-REC.dat). The transistor Q1 controls the card supply in order to reset it after a mistake or disable it for protection during extraction. When RD3 is low, Q1 conducts and the card is powered. Otherwise, when RD3 is high, Q1 is off and the card is disabled. Communication with the SD card is done through the PIC18LF4620's PORTD.

LEDs D1 and D2 indicate the circuit's state. When the red LED illuminates, there's an error or the card is in an initialization phase. The green LED blinks during recording. The microcontroller uses a crystal to generate a 20-MHz clock signal. It serves as a reference for the timing signal necessary to keep the sampling frequency constant. For details, refer to the Firmware section of this article. I built it with a prototype circuit board (5 cm × 10 cm) that I enclosed in a black plastic box (see Photo 2). I mounted two RCA connectors on its top side: a red one for battery power and a white one for the shielded cable coming from the antenna. The box is set at the base of the antenna to facilitate the transport.

HEXAGON LOOP ANTENNA

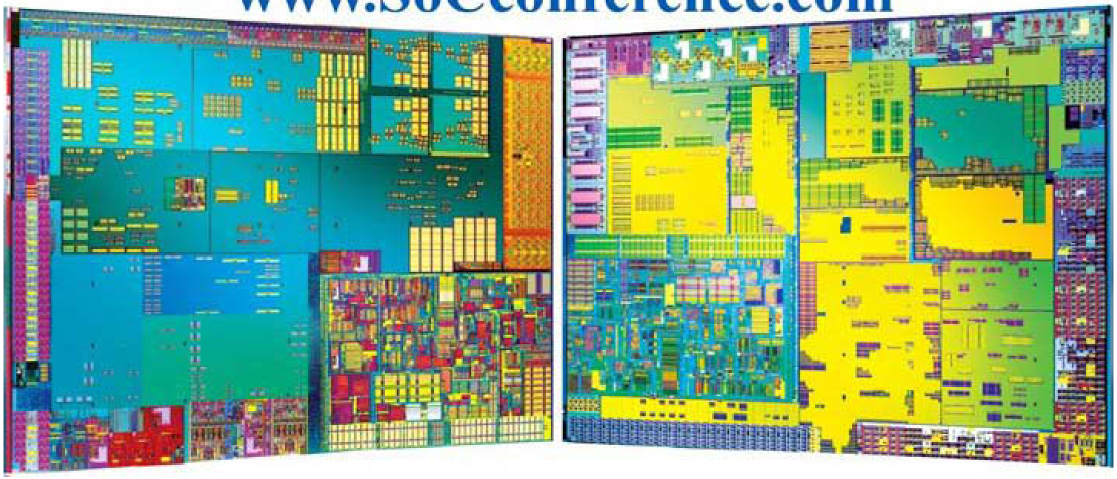
Making a sufficiently sensitive antenna in the ULF is not easy because of large wavelengths. To make a dipole, I needed linear

7th International System-on-Chip (SoC)

Conference, Exhibit & Workshops

November 4 & 5, 2009 - Radisson Hotel Newport Beach, California

www.SoCconference.com



SoC 7th
INTERNATIONAL SYSTEM-ON-CHIP CONFERENCE

For Information or Questions, Please Contact:
SoC@SavantCompany.com or (949) 851-1714

www.SavantCompany.com

Die images courtesy of Intel Corporation.

FORMERLY ARM DEVELOPERS' CONFERENCE:

ARM® techcon³

DESIGN TO THE POWER OF THREE

IS COMING **OCTOBER 21-23, 2009**

TO THE **SANTA CLARA CONVENTION CENTER**

Get 3X the Solutions for your ARM Powered Designs

Increase your knowledge exponentially and decrease your time-to-market by overcoming challenges in designs for everything from Motorcontrol to High-speed Communications that require energy conscious processors, tools and applications that are secure and connected.

Three Events in One!

Select from over a hundred classes in our intensive and value-packed Conference, and visit key industry experts on the exhibition floor for the latest in:

Energy Efficiency Leveraging energy efficient SoC strategies to minimize power requirements

MCU & Tools Enabling successful on-time product development, integration, testing and production

Internet Everywhere Developing applications for a connected world

REGISTER NOW for only $495*

Use Promo Code: ARM09EarlyBird

*A Discount of $200 off Std. Registration

WWW.ARMTECHCON3.COM

MEDIA SPONSORS:

CIRCUIT CELLAR · EDA Tech Forum · HEARST Electronic PRODUCTS · Extension MEDIA · IQ · OpenSystems Publishing · RTC

BROUGHT TO YOU BY:

Convergence PROMOTIONS · the RTC Group

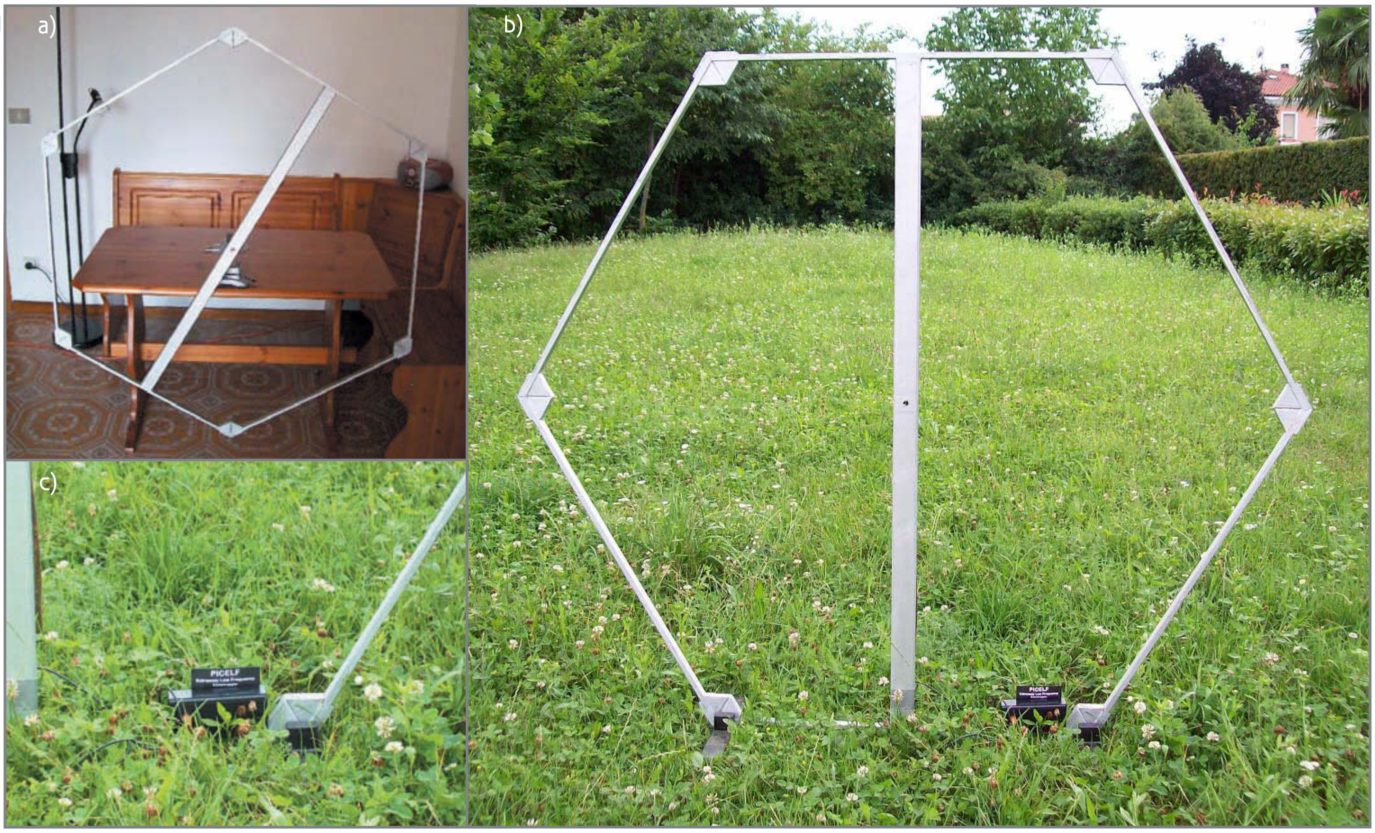


Photo 3a—Here you see the hexagonal loop antenna during wrapping. Notice the hole in the center. It's easy to use for hexagonal rotation. b—This is the datalogger and antenna working in the field—quite literally! c—Take a closer look at the design at the base of the antenna.

Touch Panel PC

CUPC-P80 $699 / Qty. 1

Windows XP based Touch Controller
- Touch panel & Touch controller
- AMD Geode LX800 (500MHz)
- 512M DDR RAM
- 4G FLASH HDD
- USB, ETHERNET, RS232, RS485 port
- SD card support
- Windows XP embedded,
  Windows 2000 / XP, Linux support

CUWIN3100
- ARM9 32bit 266MHz
- WinCE 5.0 Core
- RTC
- SD CARD support
- RS232 x 2 ports
- USB port
- Speaker & Stereo Jack

From $299 / Qty. 1

COMFILE TECHNOLOGY Toll-Free: 1.888.928.2562 http://www.cubloc.com

lengths over a few kilometers, which is certainly not easy at home. One way to resolve the issue is to detect only the magnetic component of the waves using an aerial loop. According to Faraday's law, it produces a voltage proportional to the signal frequency:

$$E = 4\pi \times 10^{-7} \times N \times A \times 2\pi F \times H$$

N is the number of turns. H is the magnetic field (A/m). A is the area of the loop (m^2). F is the measurement frequency (hertz). For the magnetic field, Tesla (T) is often used as the unit of measurement. The conversion factor is 1 µT = 0,796 A/m. I made a hexagonal antenna with a side of 70 cm. Its structure is less cumbersome than a square with equal area (see Photo 3). The form enables you to easily wrap the copper wire and avoid the stress arising from the 90° corners that could lead to failures. I made a hole in the center that I used to rotate the antenna during the winding operation. In this

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

way, it's possible to wrap a 0.1-mm copper wire to 150 laps with a total length of 630 m in 20 minutes, keeping tension in the wire without any break. The hexagon's area is 1.28 m^2 (i.e., 0.7 × 3 × 0.61). When you choose the antenna's form and size, you have to be aware of thermal noise. It is generated by the natural agitation of electrons in the conductor, and it is proportional to the loop's resistance. You must minimize the resistance to ensure that noise is less than the signal; otherwise, it will be covered by interferences. The generated thermal noise can be calculated quickly with a simple formula:

$$Q = 0.13 \times \sqrt{R}$$

R is the loop resistance in ohms. The noise Q is measured as nV$\sqrt{\text{Hz}}$. For my antenna, the loop resistance is equal to 820 Ω, so the noise is equal to 3.72 nV$\sqrt{\text{Hz}}$. The AD7680 ADC is sensitive, so I didn't include a preamplifier in order to avoid additional noise. The loop is connected directly to the ADC through a short piece of shielded cable. Of course, using the aforementioned formula, you can choose the antenna you want. The ADC can detect a voltage between 0 and 3.3 V with a depth of 16 bits; therefore, it can discern low values without problems. The important thing is to find a compromise between the loop resistance (proportional to the number of turns) and its dimensions. You also must stabilize it mechanically. Vibrations from wind or other events may cause the so-called microphone effect to generate vertical lines in spectrograms. For my hexagonal antenna, I added a central vertical axis that sticks in the ground. The structure is stabilized with two nylon bolts that prevent the torsion movements. I made two wooden supports to prop up the antenna.

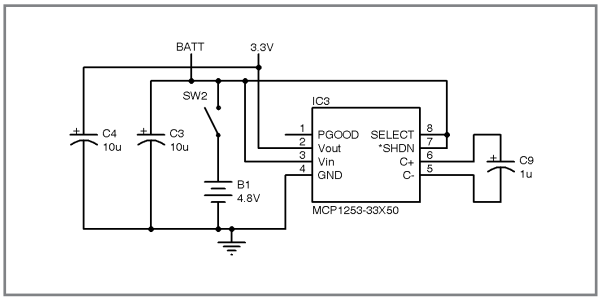


Figure 3—This is the battery voltage regulator. You need only three capacitors. It can't get much simpler than this.

BATTERY CIRCUIT

The datalogger power section consists of four NiMH batteries with a total voltage of 4.8 V and a capacity of 2,000 mAh. I kept it separate from the heart of the circuit (see Figure 3). The batteries are connected to a Microchip Technology MCP1253 chip. It is an inductorless, positive-regulated charge pump DC/DC converters (see Photo 4). The device allows the input voltage to be lower or higher than the output voltage by automatically switching between

Cellular and GPS capable

Data Mover

- *Flash file System*
- *4 Chan 12-Bit A/D*
- *1MB SRAM*
- *512KB FLASH*
- *4 Isolated Inputs*
- *4 Hi Current Outputs*
- *2 External RS-232*
- *2 Internal RS-232*
- *Bat Backed Clk/Cal*
- *Cell Modem Option*
- *Internal GPS Option*
- *Metal Case Option*
- *1ma Standby Option*

It's easy and cost-effective to do mobile or solar-powered data collection and asset monitoring with the JK*micro's* **Data Mover**.

With the ability to integrate a Cellular Modem, GPS and DOS-based embedded controller in a single rugged enclosure, you can capture and transmit your data quickly, easily and at low cost. Inexpensive development kits including Borland C/C++ and PowerBasic are available now. Call or email us for more details.

Call **530-297-6073** Email **sales@jkmicro.com**
On the web at **www.jkmicro.com**

JK *microsystems*

AP CIRCUITS
PCB Fabrication Since 1984

As low as...
$9.95
each!

Two Boards
Two Layers
Two Masks
One Legend

Unmasked boards ship next day!

www.apcircuits.com

VISA MasterCard PayPal IPC MEMBER ASSOCIATION CONNECTING ELECTRONICS INDUSTRIES BBB

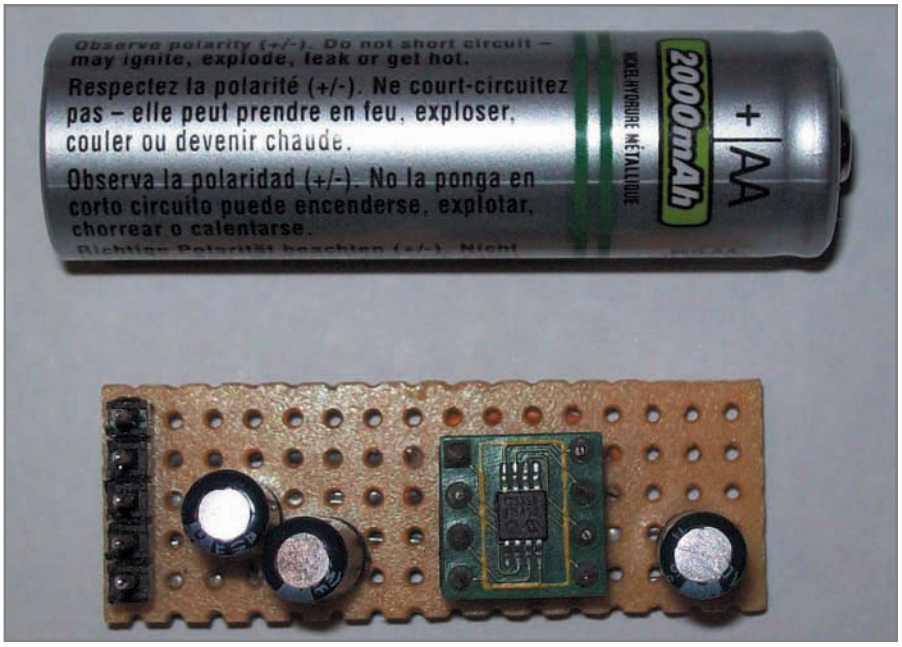


Photo 4—This is the battery voltage regulator circuit. Note the MCP1253's dimensions with respect to a single AA rechargeable battery.

buck and boost operation. In my design, it generates a regulated 3.3-V fixed voltage.

For the charge pump, I used a 1-µF electrolytic capacitor with optimal results. To filter the input and output, I added two 10-µF capacitors. I disabled the Shutdown function by connecting the SHDN pin to V_{IN}. Again, the chip package is an MSOP-8, so I used the usual converter for assembly.

To monitor the battery voltage, I used the 10-bit ADC integrated into the PIC18LF4620. I have not used the line PGOOD because its action threshold is rather low (7% below the nominal value). The circuit sinks current only during write operation, so it could be that the battery voltage is sufficient for sampling but not for the file closure involving the firmware stop and the loss of data stored so far. The recording system has a current absorption which stands at around 35 mA. If you use other types of batteries, be careful to calculate the limit of the PIC18LF4620's intervention so the battery can provide the energy required for the file closing. The autonomy with four NiMH batteries (2,000 mAH) quietly covers 48 hours of continuous recording at room temperature.

FIRMWARE

I wrote the code—which includes several interesting procedures—in PICBASIC. I can reuse it for other projects, especially those with an SD card. To speed up the writing process, I translated the PICBASIC instructions `SHIFTIN/SHIFTOUT` into two assembly sub-procedures: `SHIN` and `SHOUT`. They are also called by three basic procedures: `STARTSD`, which initializes the card; `WRITEBEG`, which initiates the writing of a 512-byte sector beginning at the 32-bit address specified by variables `IND1` and `IND0`; and `WRITEND`, which completes the write task.

The sampling operation is timed through the TIMER0 configured through the register T0CON. The TIMER0 is used in 16-bit mode with the clock source CLKO (internal instruction clock cycle) and a prescaler set at 1:4. The firmware starts the TIMER0 count up setting the TMR1ON bit to one. Then, it reads repeatedly the increasing 16 bit value until it reaches the limit according to the sampling frequency. So, the sampling cycle is not interrupt-driven.

See It & Solve It

with

USBee Test Pods

- Logic Analyzers
- Oscilloscopes
- Signal Generators
- Protocol Analyzers
  - I2C, SPI, ASYNC, CAN
  - 1-Wire, PS/2, USB
  - I2S, SMBus, Serial

- Configurable
- Programmable
- High Speed USB
- PC-Based
- And start at $139

USBee™

USB based Electrical Engineer

www.USBee.com

This prevents that interrupt signal from stopping the writing of each 512-byte sector.

To optimize the SD card storage, you must know the data structures used by the FAT16 file system. The most important information is stored in the boot sector during the formatting process. This information enables you to make the firmware independent of the card size. By reading some areas of the boot sector the firmware can independently calculate pointers to the three fundamental data structures in order to write a file: ROOT SECTOR, FAT SECTOR, and FIRST DATA SECTOR. In ROOT SECTOR, you save the file name, its attributes, the first cluster file, and the file size. In FAT SECTOR, you save the "FAT chains," cluster sequences containing the recorded values. In FIRST DATA SECTOR, you save the values in blocks of 512 bytes. Let's look at this calculation.

After card initialization, the firmware reads the BOOT SECTOR (the card's first sector). In particular, six essential values are read at the following offsets: 0BH BSET (16 bit), 0DH SCLU (8 bit), 0EH Sres (16 bit), 10h NFAT (8 bit), 11h NROT (16 bit), and 16h NSFAT (16 bit). BSET is the number of bytes per sector. SCLU is the number of sectors per cluster. SRES is the total number of reserved sectors. NFAT is the number of file allocation tables (FATs). NROT is the root directory number of entries. NSFAT is the number of sectors per FAT. With these six values it is possible to calculate the three pointers: the first FAT sector, the first ROOT sector, and the first DATA sector. The pointers in the SD card are always multiples of 512. The first FAT sector is calculated

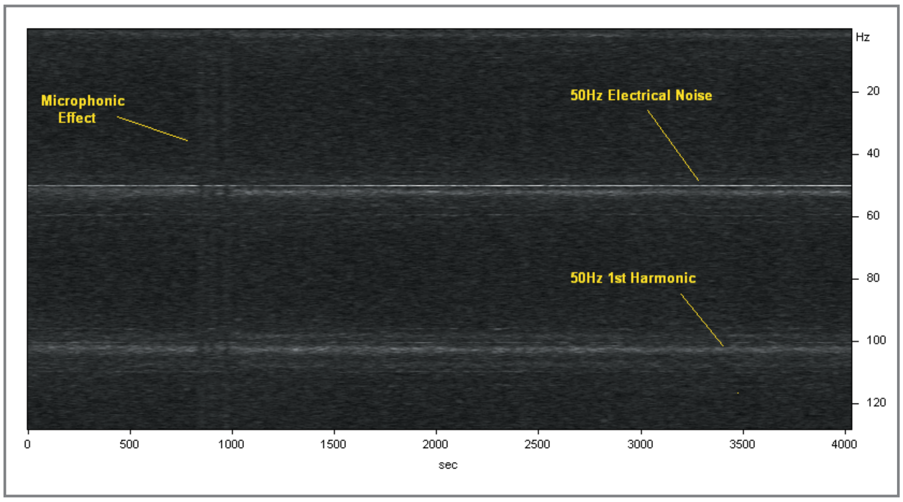
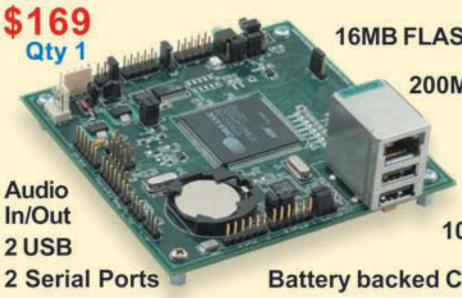


Photo 5—This screenshot shows six hours of recording in an urban environment.

Easy Embedded Linux

$169 Qty 1



16MB FLASH / 32MB RAM

200Mhz Arm9 CPU

16 Digital I/O

Watchdog

10/100 Ethernet

Audio In/Out

2 USB

2 Serial Ports

Battery backed Clock/Calendar

*We brought you the world's easiest to use DOS controllers and now we've done it again with Linux. The **OmniFlash** controller comes preloaded with Linux and our development kit includes all the tools you need to get your project up and running fast.*

*Out-of-the-box kernel support for USB mass storage and 802.11b wireless, along with a fully integrated Clock/Calendar puts the **OmniFlash** ahead of the competition.*

Call **530-297-6073** Email **sales@jkmicro.com**

On the web at **www.jkmicro.com**

JK microsystems

THE ORIGINAL SINCE 1994

PCB-POOL®

Beta LAYOUT

Servicing your complete PCB prototype needs :

- **Low Cost - High Quality** PCB Prototypes
- **Easy online Ordering**
- **Full DRC included**
- **Lead-times from 24 hrs** NEW !
- **Optional Chemical Tin finish** no extra cost NEW !

Watch "ur" PCB® Follow the production of your PCB in **REALTIME**

FREE LASER STENCIL WITH ALL PROTOTYPE PCB ORDERS

email : sales@pcb-pool.com
Toll Free USA : 1 877 390 8541
www.pcb-pool.com

Beta LAYOUT

simply by multiplying SRES by 512. The first ROOT sector is calculated by adding the previous value with NSFAT × NFAT. Finally, the first DATA sector is calculated by adding the last value with (NROT/512) × 32.

To avoid problems with 16-bit multiplication, I used a simple sub-procedure called `CALCIND`. It translates the 16-bit operation in a progressive summation of values equal to 512. This simple code lets you make much more dynamic the access to SD cards that can be of any size. The only care is that they must have been previously formatted in FAT16.

With respect to sampling, the AD7680 provides two modes of communication: a 24 clock cycle message and a 20 clock cycle message. I used the last one. The sampling is started by bringing the CS line to low logic level. The first four clock cycles are leading zeroes so they are discarded. The next 16 cycles allow the detection of the sample value bits from SDATA line at each falling edge of the SCLK line. After that, the CS line returns to high logic level. For more details, refer to the source code on the *Circuit Cellar* FTP site.

REAL APPLICATION

I performed several tests in both urban and natural environments. To facilitate the data analysis process, don't lengthen the recording times. You can generate useful spectrograms with 6, 12, or 24 hours of sampling. Excessively long recordings can hinder data processing and generate poor results.

You must first position the antenna correctly. The best direction is perpendicular to the North/South or East/West direction. Next, mechanically stabilize the antenna to avoid torsion and vertical/horizontal movements. You can then insert the SD card that you already formatted in FAT16. In Windows XP, you can use the following command line:

```
FORMAT [SD-volume]: / FS: FAT
```

Now you can start recording. Simply put the SW1 switch in the On position. The red LED will illuminate during initialization, and then the green LED will start flashing slowly (once per second). The red LED illuminates during the card initialization process or during a writing error. The firmware performs repeatedly the AVVIOSD sub-procedure until the SD Card initialization phase has completed without significant errors. The initialization procedure is repeated in 2-s intervals. You can restart the system by switching off SW1 and then turning it on a few seconds later. Do not worry about formatting. It's not necessary until you close the file.

The firmware considers any writing error as a serious error, so it saves all the data recorded until the card is turned off. The system monitors the battery voltage. When the voltage drops below 4.1 V, the system closes the file and stops the recording. Of course, you can stop recording at any time. Just hold the SW2 switch for a couple of seconds until the two LEDs are off. At that point, you can remove the SD card and read the raw data from the LF-REC.dat binary file on a PC.

I used SignalLab's SIGVIEW32 real-time signal analysis software for this project. It has the complete range of spectral analysis tools, statistical functions, and 2-D and 3-D graphical solutions. It can import data from raw binary files in 16-bit unsigned format. Photo 5 is a screenshot that shows the results of 6-hour recording in an urban environment. You can clearly see the interference from power lines at 50 and 60 Hz.

FUTURE DEVELOPMENTS

The datalogger is a handy design that should be easy to export and reuse in other projects, especially projects that require me to monitor signals with slow variations. For instance, I plan to add new antennas with different sensitivities in the near future. An interesting project would be to develop an instrument for measuring electromagnetic pollution. In the meantime, I will continue to use my design to analyze ULF radio emissions. This is a topic that never ceases to amaze me.

Carlo Tauraso (carlo@tauraso.eu) wrote his first assembly code in the 1980s for the Sinclair Research ZX Spectrum. Today he's a senior software engineer who performs firmware development work on network devices and various kinds of microinterfaces for a range of European companies. Several of Carlo's articles and programming courses have been published in Italy, France, and Spain. In his spare time, Carlo enjoys playing with radio scanners and his homemade metal detectors.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/230.

SOURCES

AD7680 ADC
Analog Devices | www.analog.com

MCP1253 DC/DC Converter and PIC18LF4620 microcontroller
Microchip Technology, Inc. | www.microchip.com

miniSD Card
SanDisk Corp. | www.sandisk.com

SIGVIEW Software
SignalLab | www.sigview.com

FEATURE ARTICLE

by Kevin Gorga

Cable Tracer Design (Part 2)

Software and System Control

In the first part of this series, Kevin pr esented his innovative Cable Tracer project, which is used to detect under ground cables. In this article he covers the software and explains how to operate the design.

As you learned in the first part of this article series, I designed a cable tracer for detecting underground cables. After the system injects a 125-kHz signal into an underground cable, a pick-up coil receives a sample of the field. The field's strength indicates a cable's location and direction.

In this article, I'll present the software portion of the design process. I'll finish up with information about the system's controls and a description of its operation.

SOFTWARE

The tracer's code is simple. The configurable port pins are selected and locked. This is one of the Microchip Technology PIC's nicest features. It enables you to choose functions and set them on the pins you want. This eliminates the problem of having two features that both need the same port pins (thereby forcing you to choose one or the other). It is also a useful feature for the PC board layout. You can select the pin that best wires up on your board. The various registers are then initialized. The Microchip Technology MCP2030 requires a block of data to initialize its registers along with column parity on the data. The data is sent out port pins by bit-banging the port pins for the part's SPI. The main reason is because the MCP2030 uses the same pin for SPI DATA IN programming and as a DATA OUT pin. When Chip

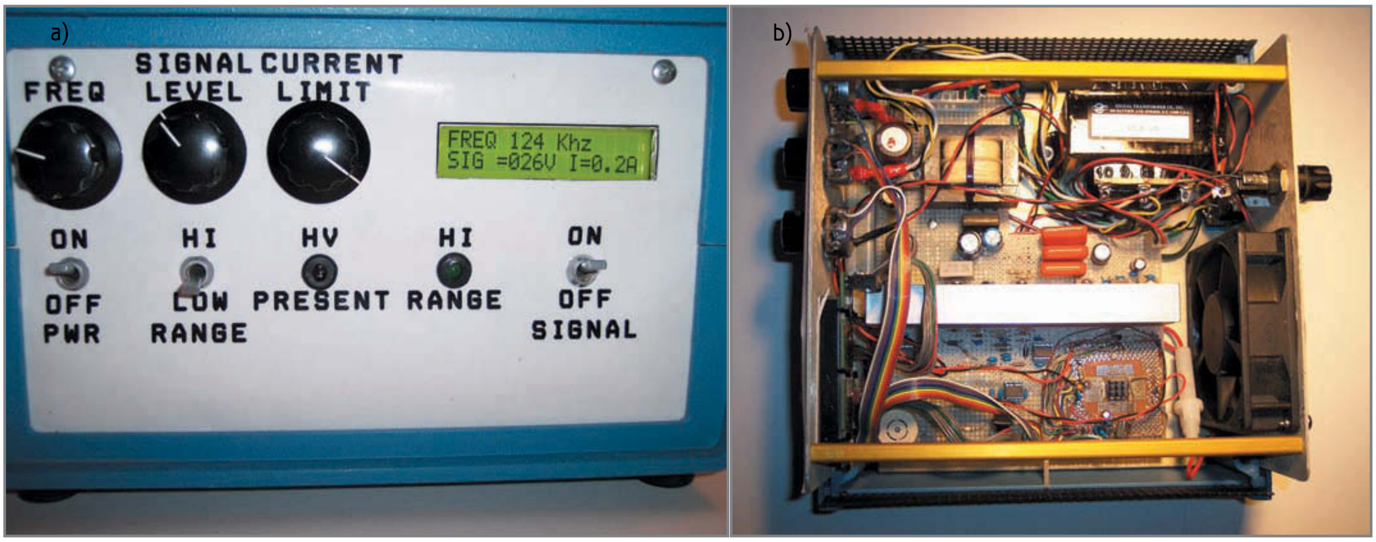


Photo 1a—Take a look at the sender. This is the front panel showing the various adjustment pots, switches, indicators, and the LCD. b—You can see the top of the circuit board and placement of the various components. All of the power semiconductors are mounted on the large heatsink in the center. It is oriented so that the fan can blow air along it. You can also see the heatsink on top of the dsPIC in the lower right corner.

Select is low, it is the SPI Data input pin. When Chip Select is high, it is the RFID Data Output pin. Chip Select has to be high for the MCP2030 to operate. This means it drives that pin and the micro can't.

Assigning the MCP2030 to the SPI controller and sharing it with the VGA—which is often updated—would be messy. The MCP2030 would have to be turned off in order to update the VGA. Instead, the VGA is exclusively connected to the SPI controller, which is needed to allow for high-speed updates. The main software loop gets the current signal strength (RSSI value), checks the push buttons, adjusts the gain if needed, and adjusts the audio tone. One of the timers is used to provide a software audio voltage-controlled oscillator. The oscillator's frequency is varied by the RSSI voltage level to provide an audio tone like a metal detector.

The sender's code is a bit more involved. The sender is more of a switch-mode power supply, so you must be careful to minimize the code executed during the regulation time. This application doesn't need a fast transient response time. This enables the microcontroller to do more work like outputting data to the LCD and monitoring the switches and potentiometers. The PWMs for the Microchip Technology dsPIC30F2023 are functionally very powerful. This enables many of the PWM's time-critical functions to be performed in hardware (e.g., the current limit function and fault shutdown).

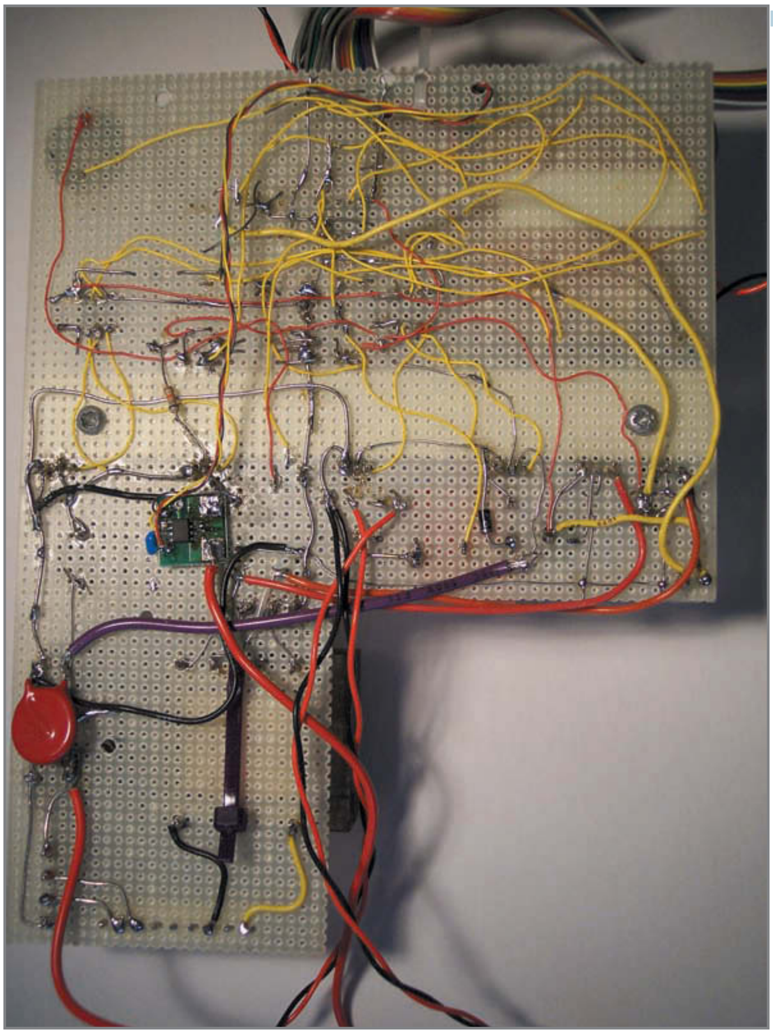


Photo 2—This is the wiring side of the Sender board. The power section is on the bottom. The low-level analog portion is on the top. The common point ground is in the center. The small PC board is the current sensor. Note the three twisted wires for the current sensor power, ground, and sensor output.

The code first initializes the various peripherals and then reads the Current Limit potentiometer to set the trip current. This is only done on power-up to keep it out of the main loop. Originally, I had it in the main loop and looked for a change in value of the Current Limit potentiometer to update the value. I had to do it only at power-up because switching noise was causing it to appear as though the value was changing all the time. The Current Limit comparator was set with a maximum value instead of the potentiometer value for noise reasons. This allowed for a high (2.5-A) short circuit value to immediately shut down the PWM in hardware. This value was high enough to prevent a false shutdown from noise on the current-sense input. The Adjustable Current shutdown is done by reading the Current Sense with the ADC and comparing it to the potentiometer value. This slower process tends to filter out the noise spikes. It may occasionally hit a noise spike and turn off for an instant, but it isn't a problem in this application.

The main code loop checks the various switches and sensor inputs. It also adjusts the PWM to regulate the output voltage. The Display update is performed after many program loop iterations because of the time it takes to update the display and carry out floating-point conversions. Doing this occasionally enables the processor to work on maintaining regulation. The beep that sounds when high voltage is present is triggered by an interrupt-driven timer for the same reason. The Frequency Adjust potentiometer sets the Output PWM's frequency. The Voltage adjustment potentiometer is compared in software against the output voltage and varies the PWM duty cycle to regulate the output voltage.

All the variables are global in my code. This is not efficient use of memory, but it makes debugging much easier. By doing this, you can "mouse over" any variable in main or a procedure and see its value.

As you can see in Photo 1, the Sender was constructed in an old case that had many of the pieces I needed (e.g., a fan, transformer, and power plug). Almost any case will do. I used a plastic case, but a metal case would have better shielding. I used just a perf board, but a good 2-oz copper PC board would make noise issues much easier to deal with. When the voltage level is turned up, noise can start to cause big problems. I had to do some software filtering to get rid of occasional A/D spike values. The

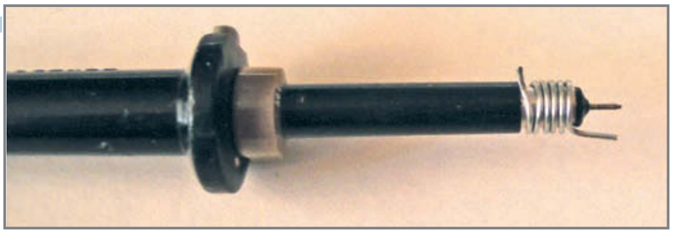


Photo 3—This is the scope probe. The small ground clip on the end of the scope probe will help reduce the ground lead "antenna effect" picking up magnetic field-coupled noise and other radiated signals.

PWM's Current Limit and Fault Current comparator features worked well at low-voltage levels, but the noise generated at higher voltages was a problem. I needed to disable this feature and do it in software with the A/D input. If I had time to do a ground-planed PC board, this would not have been a problem.

I added filter capacitors on the potentiometers. When the potentiometers were near the low-resistance end (at either end), the noise was shunted directly to ground or through a decoupling capacitor on the 5-V bus. But as the resistance was turned up in the center of the range, it became a problem. In general, any higher-impedance input will have a problem and will require a 0.1-µF bypass capacitor to reduce the noise spikes. I had noise-related false indications from the Signal On/Off switch, neon optoisolators, potentiometers, and current sensor before the capacitors were added. The Hall effect current sensor required special care. I put it on a small PC board for my prototype. Power and ground for the sensor were twisted together with the sensor output and routed back to the microcontroller. This is the small PC board in the center in Photo 2. The cable was routed away from the FET switches and related components to try to reduce switching noise pick-up. The power and ground points were at the microcontroller board.

Careful common point grounding is also a must. In the schematic shown in Part 1 of this series, I indicated three different ground regions that should come together at only one point. The covers over the back of the potentiometers should be soldered to the ground end. The FET bypass capacitors (C13 and C27) should be wired directly across the FETs.

Looking at signals in a high-noise environment can be challenging. If you place the scope tip to ground at the point of the ground clip and see something on it, the ground clip is picking it up. The small ground wire on the end of the scope probe can pick up a significant amount of noise. The arrangement in Photo 3 works well. You should remove the plastic scope probe end and wrap some bare wire around it to provide a ground point right next to the probe tip. Remove the coil from the tip and twist it together (a small amount) to make the coil tight enough to hold itself and make good contact with the grounding barrel. Use pliers on the small end of the coil to pull it back on the barrel. Any noise you see is real. Of course, this assumes you have a nearby ground point to which you can touch the ground side of the tip.

The neon bulb optoisolators are made in black heat-shrink tubing. The neon bulb and the CdS photocell are placed next to each other with the center of the bulb parallel with the photocell. This gets the maximum amount of neon light to the photocell. Placing the end tip of the bulb point at the cell doesn't get enough light to the photocell. Cover the combination with black heatshrink tubing and leave extra on the end to fold over. After it shrinks, you can fold over the end and hold it there until it hardens. It is big and ugly, but works well. The CdS I used had about 150-kΩ dark and 3-kΩ light resistance. The neon bulb is an NE2.

I used a SchmartBoard TQFP adapter to fan out the dsPIC. I used SMT decoupling capacitors across the pads on the adapter. I also mounted the crystal there to keep leads short. All of the power parts are mounted to one side of the heatsink for noise and safety reasons. The low-level components are grouped together on the other side. The power FETs and voltage regulators are mounted on a heatsink with thermal washers in the fan's air flow. The thermal washers also insulate the devices, and should be used to insulate all devices, even those with tabs that are grounded to reduce coupled noise. I also found that not grounding the heatsink itself reduced the noise in the grounds. In most cases, when the sender is just driving an open cable, the unit runs cool. I used a PC-style power connector with a built-in filter and fuse. I find that the removable PC power cord makes storing projects much neater.

The receiver is much less critical. It is housed in a plastic project box. Plastic boxes are easy to work with. A Unibit can easily cut through it and you don't need an entire set of drill bits. I also used perf board and a SchmartBoard adapter with the crystal and decoupling mounted on it.

The Search Coil needs to be on edge and not flat with the ground like a metal detector. At first, I thought about mounting it on the end of a wooden dowel and attaching the other end of the dowel to the Tracer box, but I found that holding it my hand made it easier to rotate to find the curves of underground cables. The downside of this technique is a sore back from bending down to the ground to follow the cable. If most cable runs that you are looking for are straight, the wooden dowel approach is the way to go.

CONTROLS

The frequency potentiometer is used to set the sender's output frequency to the resonant frequency of the search coil on the tracer. This can be done by adjusting the frequency for a peak on the tracer meter at about 125 kHz. The signal-level potentiometer is for adjusting the sender's output signal level to compensate for various search depths. The Current Limit potentiometer is for adjusting the sender power supply's overcurrent trip point.

The Hi/Low Range switch selects the 0- to 60-V or the

The Newest Embedded Tools

WiFi • USB • BLUETOOTH™ • ZIGBEE
WLAN • NETWORK • USB • RFID • WLAN
BLUETOOTH™ • EMBEDDED CONTROL • NETWORK
RFID • USB • WLAN • GPS • BLUETOOTH™
BLUETOOTH™ • ZIGBEE® • USB • NETWORK
RFID • USB • WLAN • BLUETOOTH™ • WiFi
CYMBET

TEXAS INSTRUMENTS
Authorized Distributor

Solar Energy Harvesting Development Tool:
eZ430-RF2500-SEH
www.mouser.com/ti_ez430_rf2500_seh/

www.mouser.com

Over A Million Products Online

New Products from:

MICROCHIP
MPLAB PICkit 3 Debug Express
www.mouser.com/
PICkit3debugexpress

LINX TECHNOLOGIES
Evaluation and Development Kits
www.mouser.com/linx/a

SILICON LABS
KEYBOARDEK Evaluation Kit
www.mouser.com/silabs/a

The ONLY New Catalog Every 90 Days

Experience Mouser's time-to-market advantage with no minimums and same-day shipping of the newest products from more than 390 leading suppliers.

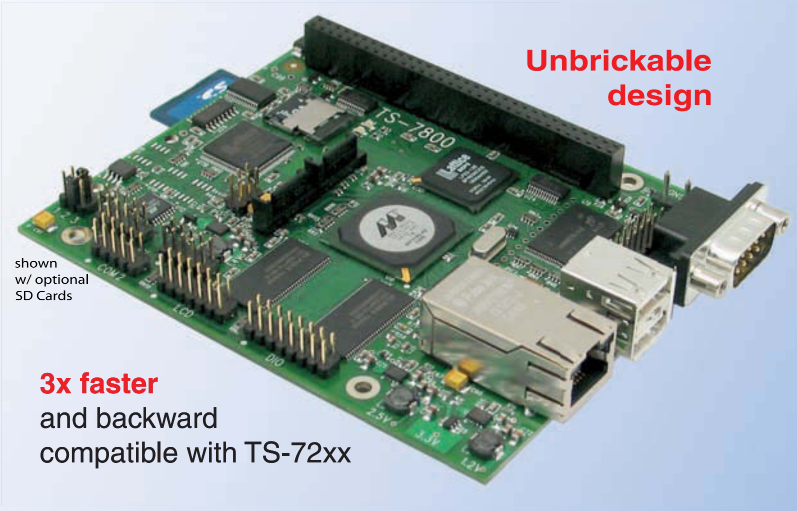
MOUSER ELECTRONICS®
a tti company

The Newest Products
For Your Newest Designs

(800) 346-6873

Embedded Single Board Computers

High-End Performance with Embedded Ruggedness


Unbrickable design
shown w/ optional SD Cards
3x faster
and backward compatible with TS-72xx

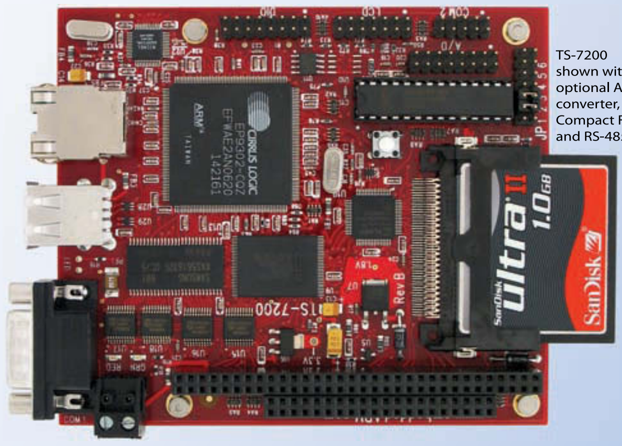
TS-7800
500 MHz ARM9

- Low power - 4W@5V
- 128MB DDR RAM
- 512MB high-speed (17MB/sec) onboard Flash
- 12K LUT customizable FPGA
- Internal PCI Bus, PC/104 connector
- 2 host USB 2.0 480 Mbps
- Gigabit ethernet
- 2 SD sockets
- 10 serial ports
- 110 GPIO
- 5 ADC (10-bit)
- 2 SATA ports
- Sleep mode uses 200 microamps
- Boots Linux 2.6 in .67 seconds
- Linux 2.6 and Debian by default

$229 qty 100

$269 qty 1

Low Price, Low Power, High Reliability using Linux development tools


TS-7200 shown with optional A/D converter, Compact Flash and RS-485

- options include:
onboard temperature sensor, A/D Converter 8 channel 12 bit, Extended Temperature, Battery Backed Real Time Clock, USB Flash, USB WiFi

200 MHz ARM9 Family
Power as low as 1/4 Watt

- 8 boards, over 2000 configurations
- Fanless, no heat sink
- SDRAM - up to 128MB
- Flash - up to 128MB onboard
- 10/100 Ethernet - up to 2
- DIO lines - up to 55
- 2 USB ports
- COM ports- up to 10
- Programmable FPGAs
- Linux, Real Time extension, Debian
- SD card option
- VGA video
- LCD ready

as low as $99 qty 100

$129 qty 1

- Over 20 years in business
- Open Source Vision
- Never discontinued a product
- Engineers on Tech Support
- Custom configurations and designs w/ excellent pricing and turn-around time
- Most products stocked and available for next day shipping

Design your solution with one of our engineers (480) 837-5200

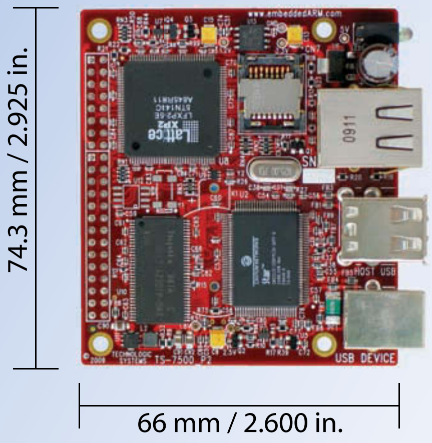
New Products

TS-7500
250 MHz ARM9

NEW!

Our Smallest Computer at Our Best Price Point

- Low power, fanless, < 2 watts
- 64MB DDR-RAM
- 4MB NOR Flash
- Micro-SD Card slot - SDHC
- USB 2.0 480Mbit/s host (2) slave (1)
- 10/100 Ethernet
- Boots Linux 2.6 in < 3 seconds
- Customizable FPGA - 5K LUT
- Power-over-Ethernet ready
- Optional battery backed RTC
- Watchdog Timer
- 8 TTL UART
- 33 DIO, SPI, I^{2}C



$84 qty 100

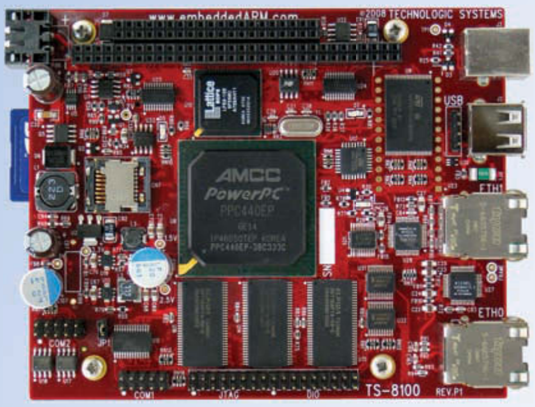
$99 qty 10

TS-8100
Ultra Reliable w/ 128MB ECC RAM

NEW!

400 MHz PowerPC with Floating Point Unit

- POE ready
- Dual-execution unit, double-precision FPU
- Multifunctional PC/104 connector
- 12K LUT customizable FPGA
- 512MB NAND Flash
- 1 USB Host, 1 USB Device (12 Mb/s)
- Boots Linux 2.6 in < 2 seconds
- Fanless < 4W, sleep mode < 1mW
- Regulated 5-28V power input
- 2 10/100 ethernet
- 4 COM ports
- SPI & DIO
- RS485/RS422
- 2 SDHC sockets
- 5 10 bit ADC
- RTC & WatchDog
- 2 DMX Channels



shown w/ optional SD Card

$229 qty 100

$269 qty 1

Technologic Systems

We use our stuff.

Visit our TS-7800 powered website at

www.embeddedARM.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

0- to 120-V range on the sender. This is used for depth adjustment along with the signal-level potentiometer. The Signal On/Off switch allows for the Output signal to be turned off. This is useful when connecting to a wire to see if high voltage is present. The HV Present neon bulb indicates high voltage on the cable. The Hi Range LED indicates the signal generator level is about 50 V and caution should be used. The LCD normally displays the operating frequency, signal level, and current. It also posts various other messages for shutdown conditions.

OPERATION

I don't recommend using the sender on a cable with a load. The high-voltage, 125-kHz signal could damage some devices like X10 controls. Most of today's electronic devices (e.g., PCs, VCRs, TVs, and CCFL lights) use switching power supplies. They have built-in filters to try to prevent their switching noise from going out on the line. These filters also try to absorb the 125-kHz signal. Now you know why your X10 devices don't work. Some outlet strips also have filters that absorb the 125 kHz.

The first thing to do is set the operating frequency. With either a signal generator or the sender, set the output frequency for 125 kHz. Place the search coil near the signal lead and increase the signal source's output level until a meter reading is obtained. Adjust the frequency for a peak on the meter.

When you power on the sender, it has a display to set the current limit. This display stays active until a few seconds after the last change of the Current Limit potentiometer. I normally set the current limit for just less than 2 A. The normal current drain is only a few hundred milliamps. Note that 2 A prevents noise from causing false trips, but it will still protect from short circuits.

After the Current Limit display times out, the normal operational display shows the Frequency, Signal Level, and Current. With the Signal switch on Off, the display indicates if line voltage is present and the HV Present neon lamp is on. The beeper will also pulse if high voltage is present. Remove the line voltage if this happens or proceed with extreme caution. The Line Plug test box also indicates with the neon lamps if there is line voltage present on any of the wires. The High/Low Range switch selects the 60- or 120-V power source. In most applications, the 60-V source is more than enough and you should start with it. Turn on the Signal switch and adjust the signal level for an indication on the tracer meter. This is usually around 10 to 20 V. Adjust the level for the minimum amount that still gives a reading on the tracer meter at gain 7 (maximum). If the gain meter on the tracer starts to drop, cut back the generator level. If the generator level rises above 50 V, the High Range LED turns on and the beeper sounds.

I've used the tracer to locate a buried cable (about 2′ to 3′ deep) with about a 20-V signal level. I was able to trace my underground gas line with about 50 V. The gas line is plastic, but it's buried with a single-conductor 12-gauge solid wire so that the gas company can find it. This gas line is probably around 4′ to 5′ deep. Tracing wires in walls requires only 5 to 10 V.

PROJECT SOURCE

Old CRT monitors and PC power supplies are excellent sources for many useful power supply-related parts. They have a wealth of expensive analog power components. I used them as my sources for this project's FETs, heatsinks, fan, degaussing coil, PC power cord filtered jack, high-voltage capacitor, and electrolytic capacitors. I caution you to test the ESR of the electrolytic capacitors. A monitor's or power supply's high heat can dry out a capacitor and raise its ESR to an unusable level. If you don't have an ESR meter, stay clear of any yellowed or old-looking electrolytics.

Now it's your turn to build a cable tracer. Hopefully you have easy access to all the essential components. If you learn anything new while designing your own system, be sure to share your findings with the rest of us! ■

Kevin Gorga (kgorga@stny.rr.com) has an MSEE and has been a design engineer with IBM in Endicott, NY, for the past 32 years. His technical interests include embedded system design, power systems, and motor controls.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/230.

RESOURCE

Microchip Technology, Inc., "MCP2030 Bidirectional Communication Demo Kit User's Guide," DS51637A, 2006.

SOURCES

ACS712-05 Current sensor
Allegro MicroSystems, Inc. | www.allegromicro.com

IR2101 FET Driver
International Rectifier | www.irf.com

MCP2030 RFID Receiver, MCP6S21 VGA, PIC24FJ64GA004 microcontroller, MCP1700 LDO, dsPIC30F2023, and MCP602 op-amp
Microchip Technology, Inc. | www.microchip.com

by Dale Wheat

Get Started With Embedded Development (Part 2)

Coding from Start to Finish

Now that you're familiar with ARM Cortex-M3 microcontrollers, it's time to take a close look at the application development process. This article covers the coding process from start to finish.

In the first article in this series, I introduced you to the STmicroelectronics STM32 ARM Cortex-M3 microcontroller. In doing so, I described some affordable development hardware and software. Now it's time to focus on application development using the new Raisonance CircleOS framework. As you'll see, you can leverage the power of the peripherals and extra hardware on the STM32 Primers to create some exciting new designs. Let's wrap up the "bare metal" example program, show those LEDs who's boss, and then move on to more sophisticated application development using the new CircleOS application environment.

PROJECT CODE

Now that you have some source code, you need to add it to the project. Select the "Project/Add Item..." menu item and select "Blink.c" from the list. This adds the source code to the project. Be sure to save your work periodically.

Since we've left the Raisonance reservation, we have to cobble up our own linker script. This tells the linker where to put the different sections of the program within the STM32 chip's memory map. For the original Primer, the linker script looks like the code in Listing 1.

Listing 2 is the linker script for the Primer2 with its correspondingly larger memory sizes. The linker script defines the two memory spaces in the chip: the nonvolatile flash memory for holding the program and the SRAM for stack and variables. Each has its starting address and length specified.

The C compiler separates most programs into three

Listing 1—This linker script is for the original Primer, Primer.ld

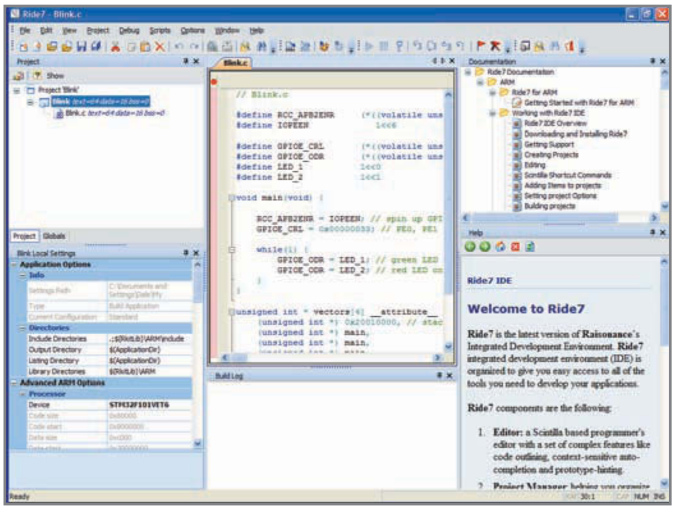
```
MEMORY {
   flash (rx) : ORIGIN = 0x08000000, LENGTH = 128k
   ram (rwx) : ORIGIN = 0x20000000, LENGTH = 20k
}

SECTIONS {
   vector : { *(vector*) } > flash
   .text : { *(.text*) } > flash
```

Listing 2—This linker script is for the Primer2, Primer2.ld

```
MEMORY {
   flash (rx) : ORIGIN = 0x08000000, LENGTH = 512k
   ram (rwx) : ORIGIN = 0x20000000, LENGTH = 64k
}

SECTIONS {
   vector : { *(vector*) } > flash
   .text : { *(.text*) } > flash
}
```



```
// Blink.c

#define RCC_APB2ENR      (*((volatile uns
#define IOPEEN              1<<6

#define GPIOE_CRL        (*((volatile uns
#define GPIOE_ODR        (*((volatile uns
#define LED_1            1<<0
#define LED_2            1<<1

void main(void) {

    RCC_APB2ENR = IOPEEN; // spin up GPI
    GPIOE_CRL = 0x00000033; // PE0, PE1

    while(1) {
        GPIOE_ODR = LED_1; // green LED
        GPIOE_ODR = LED_2; // red LED on
    }
}

unsigned int * vectors[4] __attribute__
    (unsigned int *) 0x20010000, // stac
    (unsigned int *) main,
    (unsigned int *) main,
```

Ride7 IDE

Welcome to Ride7

Ride7 is the latest version of **Raisonance**'s Integrated Development Environment. **Ride7** integrated development environment (IDE) is organized to give you easy access to all of the tools you need to develop your applications.

Ride7 components are the following:

1. **Editor:** a Scintilla based programmer's editor with a set of complex features like code outlining, context-sensitive auto-completion and prototype-hinting.

Photo 1—The Raisonance Ride7 IDE is in editing mode. As you can see, you have access to quite a lot of information in the default view, including the project outline, individual project options, multiple source code windows in a tabbed interface, and extensive, online help documentation. You can also customize the view to your liking by selecting, rearranging, and resizing the various panes.

distinct sections: program code, initialized data, and uninitialized data. These sections are historically referred to as ".text," ".data," and ".bss." We create a new section called "vector" to hold the minimum vector table for the Cortex-M3. We declare the section using the `__attribute__((section("vector")))` syntax, which is specific to the GNU GCC compiler. This defines a set of four addresses that the Cortex-M3 needs to wake up in the morning. The first entry is the initial value for the stack pointer. This is set to the end of physical SRAM on the chip. The next entry is the reset vector. Since we don't need any special code to be executed before starting our `main()` function, we use the address of the `main()` function itself here. The next two entries are supposed to be for exception handling, but in this overly simplistic example we just repeated the address of the `main()` function, so no matter what happens, `main()` executes. This would be replaced, of course, by actual exception handlers in a real application.

By specifying the section order in the `SECTIONS` area of the linker script, we are able to dictate to the linker the exact placement of the various sections of our program within the final binary image that we will download into the chip. The Cortex-M3 is the first ARM architecture where it is possible to write an entire application in C. Previous architectures required assembly language programming for start-up, exception, and interrupt handlers.

Before we can compile this project and load it into the Primer, we have to make a few tweaks to the Blink project. These are easily browsed and edited in the "Project Options" pane of the IDE, which is initially located in the lower-left corner of the screen (see Photo 1).

Change the "Optimization level" from "-O1" to "No optimization." This forces the compiler to emit executable code that looks a lot like your original code, which helps when debugging. You wouldn't think that the compiler would be able to cut much fat from our exceedingly skinny example, but you would be wrong. You're encouraged to play with the different optimization levels and see what cleverness ensues; but for the purpose of this exercise, turn off the optimizations for now.

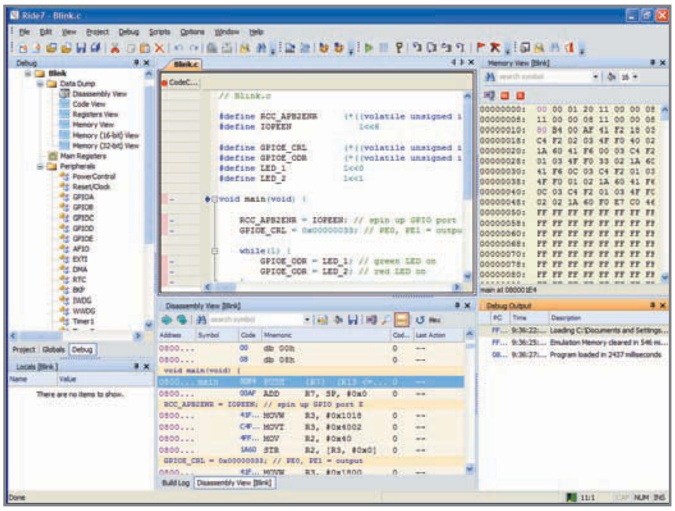
Under "More compiler options" add "-ffreestanding" (that's with two lowercase Fs). This tells the compiler that our code stands alone and neither requires the presence of an operating system nor returns to the operating system upon completion. This is typical for embedded systems that code to the "bare metal." It also enables us to declare the `main()` function as not returning a meaningful value (Return to whom?), which would otherwise trigger a warning message that the return type is not "int," as it was for our fathers and their fathers before them.

Below the "LD linker" section, change "Remove unused sections" from "Yes" to "No." I'm not really sure what it bases this decision on, as it considers all of our hand-crafted sections to be unused and our output file ends up being completely empty. Not good.

In the "Startup" section, change "Use Default Start-up" to "No" and clear the entry for "Startup File." We don't need no stinkin' start-up files.

Furthermore, change "Use Default Script File" to "No" as we wrote our own. Right below that, click on the Browse button (it has an "..." ellipsis on it) on the right side of the text box and navigate to the Primer.ld or Primer2.ld files. In reality, either one would work for our trivial example.

In the "Libraries" section, just say "No" to all the offered libraries. Our program has no need of them and



```
// Blink.c

#define RCC_APB2ENR      (*((volatile unsigned i
#define IOPEEN              1<<6

#define GPIOE_CRL        (*((volatile unsigned i
#define GPIOE_ODR        (*((volatile unsigned i
#define LED_1            1<<0
#define LED_2            1<<1

void main(void) {

    RCC_APB2ENR = IOPEEN; // spin up GPIO port
    GPIOE_CRL = 0x00000033; // PE0, PE1 = outpu

    while(1) {
        GPIOE_ODR = LED_1; // green LED on
        GPIOE_ODR = LED_2; // red LED on
```

```
Address   Symbol  Code   Mnemonic                  Cod...  Last Action
0800...           00     db 00h                    0       --
0800...           08     db 08h                    0       --
 void main(void) {
0800... main      B0F4   PUSH     {R7}  {R13 <-... 0       --
0800...           00AF   ADD      R7, SP, #0x0     0       --
 RCC_APB2ENR = IOPEEN; // spin up GPIO port E
0800...           41F... MOVW     R3, #0x1018      0       --
0800...           C4F... MOVT     R3, #0x4002      0       --
0800...           4FF... MOV      R2, #0x40        0       --
0800...           1A60   STR      R2, [R3, #0x0]   0       --
 GPIOE_CRL = 0x00000033; // PE0, PE1 = output
0800...           41F... MOVW     R3, #0x1800      0       --
```

```
00000000:  00 00 01 20 11 00 00 08
00000008:  11 00 00 08 11 00 00 08
00000010:  80 B4 00 AF 41 F2 18 03
00000018:  C4 F2 02 03 4F F0 40 02
00000020:  1A 60 41 F6 00 03 C4 F2
00000028:  01 03 4F F0 33 02 1A 60
00000030:  41 F6 0C 03 C4 F2 01 03
00000038:  4F F0 01 02 1A 60 41 F6
00000040:  0C 03 C4 F2 01 03 4F F0
00000048:  02 02 1A 60 F0 E7 C0 46
00000050:  FF FF FF FF FF FF FF FF
00000058:  FF FF FF FF FF FF FF FF
00000060:  FF FF FF FF FF FF FF FF
00000068:  FF FF FF FF FF FF FF FF
00000070:  FF FF FF FF FF FF FF FF
00000078:  FF FF FF FF FF FF FF FF
00000080:  FF FF FF FF FF FF FF FF
main at 080001E4
```

```
PC     Time        Description
FF...  9:36:22:... Loading C:\Documents and Settings...
FF...  9:36:25:... Emulation Memory cleared in 546 mi...
08...  9:36:27:... Program loaded in 2437 milliseconds
```

Photo 2—The Raisonance Ride7 IDE in debug mode. You have a wide array of debugging tools available here, including the ability to single step through the code, see the actual machine language instructions, examine memory locations and peripheral registers, as well as change them in real time. As in the editing mode, the selection and arrangement of the individual panes in the window can be customized to your liking.

they do not need to be linked into our executable image. Be sure to save your work so far. "Save early, save often," I always say.

OK, all the clicking, tweaking, and poking at our project is done. It is properly configured and ready to be built. Select "Project/Make" or press F9 and the project will be built. Hopefully, you'll see "Build succeeded" in the Build Log at the bottom of the screen. If not, scroll back up in the Build Log and look for any colored lines of text describing the compiler's complaint.

Now that the program has been compiled, linked, and converted into a downloadable format, select the "Debug/Start" menu item, click the "Start Debugging" toolbar icon (blue arrow), or press Ctrl+D. This writes our tiny program into the STM32's nonvolatile flash memory, assuming your Primer is plugged in to the USB DEBUG port and powered on. As before, the system switches over to Debug mode and waits in a state of readiness (see Photo 2).

The cursor should be lined up at the beginning of the `main()` function. There is an option to have the debugger start at the reset vector itself, which would be helpful for debugging any code that might need to be executed before `main()` takes control, but in our case there is none.

Let's single-step through our program. Select the "Debug/Step Into" menu item, click the "Step Into" toolbar icon or press F7. The cursor moves to the first of our two initialization statements. Now take a look at the "Disassembly View [Blink]" in the pane at the bottom of the screen. This shows a representation of the actual machine language codes that are about to be executed by the processor. Click "Step Into" again and this code is executed and the cursor advances to the next initialization statement. Nothing to see yet, move along.

Click "Step Into" again and we've reached the inside of the endless `while()` loop. Now take a look at your Primer as you click on "Step Into" again. The green LED lights up. This might be a little hard to see as the user-programmable green LED is immediately adjacent to the "Battery Charged" green LED indicator on the Primer2.

Click again and the green LED is extinguished and the red LED is illuminated. Click again to get to the end of the `while()` loop, which corresponds to a "branch" instruction that takes us back to the top of the endless loop. Click until the novelty wears off.

When you have exhausted the possibilities of this fine bit of firmware craft, select the "Debug/Terminate" menu item, click the "Terminate debugging" toolbar icon (blue square) or press Ctrl+Shift+D. This ends the debugging session and returns you to the code-editing mode.

As promised, I will detail the small changes that are required to run our example program on an original Primer instead of the Primer2. First, by looking at the schematic diagram of the Primer, we see that the two user-programmable LEDs are connected to Port B, pins PB8 and PB9. This means that we have to enable the clock for Port B (IOBEN instead of IOEEN in the RCC_APB2ENR register). We also need to write our magic number 0x00000033 to GPIOB_CRH instead of GPOIE_CLR to configure Port B I/O lines 8 and 9 as outputs. It would also be appropriate to use the address of the GPIOB_ODR (General-purpose input and output Port B/Output data register) instead of the address of GPIOE_ODR when writing the ones and zeros that toggle the LEDs.

Finally, we need to change the value of the initial stack pointer as described in the vector table from 0x00010000 to 0x00005000 so that it starts at the end of the 20-KB SRAM of the STM32F103RBT6 instead of the 64-KB SRAM of the STM32F103VET6 of the Primer2. Other than that, the programs are identical.

BEYOND THE BASICS

Now that we have a working skeleton of a program, it's time to take inventory of what we have available to us at this point. Obviously, we have the technology to

Listing 3—This is a simple exit strategy for a CircleOS application, embedded within the `Application_Handler()` function.

```
    if ( BUTTON_GetState () == BUTTON_PUSHED ) {
           BUTTON_WaitForRelease ();
           BUTTON_SetMode ( BUTTON_ONOFF_FORMAIN );
           return MENU_Quit ();
    }
```

Listing 4—The program needs some global variables.

```
const u16 white_dot[16] = {
       RGB_WHITE, RGB_WHITE, RGB_WHITE, RGB_WHITE,
       RGB_WHITE, RGB_WHITE, RGB_WHITE, RGB_WHITE,
       RGB_WHITE, RGB_WHITE, RGB_WHITE, RGB_WHITE,
       RGB_WHITE, RGB_WHITE, RGB_WHITE, RGB_WHITE
};

const u16 green_dot[16] = {
       RGB_WHITE, RGB_GREEN, RGB_GREEN, RGB_WHITE,
       RGB_GREEN, RGB_GREEN, RGB_GREEN, RGB_GREEN,
       RGB_GREEN, RGB_GREEN, RGB_GREEN, RGB_GREEN,
       RGB_WHITE, RGB_GREEN, RGB_GREEN, RGB_WHITE
};

const u16 red_dot[16] = {
       RGB_WHITE, RGB_RED, RGB_RED, RGB_WHITE,
       RGB_RED, RGB_RED, RGB_RED, RGB_RED,
       RGB_RED, RGB_RED, RGB_RED, RGB_RED,
       RGB_WHITE, RGB_RED, RGB_RED, RGB_WHITE
};
```

“The first part of the supplied initialization stub performs a check of the required revision level and either proceeds with program execution or displays a message to the user on the LCD and halts. Any additional one-time initialization that your application requires would then be executed.”

blink a couple of LEDs. By looking at the schematic, we can find other trivial circuits to manipulate, such as the LCD backlight. Remember, each GPIO port must be enabled before trying to access its associated registers, or an exception will be thrown by the hardware. This is a protection mechanism that prevents runaway code from wreaking havoc. Also, you must configure each I/O line for its proper operational mode. You will need to review the datasheet for the STM32 part in question to find out more about the specific capabilities of each I/O line. As you can see from our example, it will probably end up being just a couple of lines of executable code, with some optional `#define` statements if you want your code to make sense a few months from now.

CircleOS

If you like the challenge of writing all your own peripheral drivers from scratch, you have all the tools you need at this point. Get busy! If you would rather focus on application software and leverage a set of already-written routines to access Primer peripherals, such as the color LCD, MEMS accelerometer, push button, or buzzer, then allow me to introduce you to the world of CircleOS programming.

The CircleOS was developed specifically for the Primer to showcase its unique capabilities and help STM32 designers get a quick start on their application development. This open-source software project was launched by Raisonance in August 2007 to support user development of Primer applications. It provides a simple mechanism for allowing multiple applications to be loaded simultaneously within the flash memory of the unit and selected via a menu. This makes it much easier to carry around a selection of applications with you when you’re away from your development machine.

The CircleOS is evolving at a rapid pace. I got a sneak preview of the 3.4 beta version that was the first to differentiate automatically between the Primer and Primer2. Get a copy of the latest CircleOS sources from www.stm32circle.com. I would like to show you how easy it is to write your own CircleOS applications using the project templates provided with the Primer.

YOUR FIRST CircleOS APP

Start the Ride7 integrated development environment and select “File/New.../Project.” As in our previous exercises, select the target processor using the tree view, but this time look at the top of the list and select either “STM32_Primer1_CircleOS” or “STM32_Primer2_CircleOS,” think of a clever name (I used “Primer2 Test”) and a memorable location to save your project files, and then click the “Finish” button. This creates a complete CircleOS project definition and loads up all the pieces you need to create a CircleOS application, including a skeleton application file called “Application.c.”

If you compile and download the skeleton project as created, you will get

We Listen. Think. And Create.

Distributed I/O — Serial I/O — **Digital I/O** — Industrial Computing — HMI

SeaDAC USB modules are the fastest, most reliable way to connect I/O to any computer.

SeaDAC USB Modules Offer:

- Optically Isolated Inputs, Reed Relay Outputs, Form C Relay Outputs, Digital/Analog Combo
- Status Indicator LEDs for Communication, Fault, & Status
- Field Removable Terminal Block Connectors
- High-Retention USB Type B Connector
- Rugged Plastic Tabletop Enclosure
- Extended Temperature Option Available

FOCUS On Success
Call Today!

SEALEVEL

sealevel.com > sales@sealevel.com > 864.843.8067 a clean installation of the CircleOS on your Primer with a single application available, called "My App." It doesn't do much at the moment. In fact, there is no way to exit the program gracefully, short of holding down the button momentarily to power-off the unit. This is a little more civilized than our bare metal example. Before we add some razzle-dazzle to the program, let's review the structure of a CircleOS application a little bit and see which way we want to go with it.

CircleOS is basically very simple in structure. To create a CircleOS application, you need to write an initialization routine named `Application_Ini()` and a handler routine called `Application_Handler()`. Stubs for both of these routines were automatically generated when you created your CircleOS application and both contain "TODO:" comments to remind you where the necessary code goes.

The initialization function is called once when the application is selected from the menu. You can indicate in your code what the minimum revision of the CircleOS is required for proper operation, in case you want to use the latest and greatest bells and whistles. One of the design goals of the CircleOS was to maintain backward compatibility with previous versions so that existing applications would not need to be modified or even recompiled to work with new releases of the CircleOS.

The first part of the supplied initialization stub performs a check of the required revision level and either proceeds with program execution or displays a message to the user on the LCD and halts. Any additional one-time initialization that your application requires would then be executed.

The "application handler" routine then gets called repeatedly by the CircleOS. This allows the CircleOS to continue to react in real time to the peripherals such as the MEMS accelerometer and push button. The use of a state machine is indicated for anything but the most trivial of examples. Upon entry into the application handler, a state variable (or variables) would be checked to see where the application is in relation to its expected set of behaviors. Each state should be clearly defined in terms of what events lead to its execution, what is to be accomplished within that state, and whether or not it should remain in the same state or advance to another state. In any case, the program should execute quickly and then return to CircleOS with either the `MENU_CONTINUE` value or the `MENU_LEAVE` value to terminate the application and return to the CircleOS menu.

Add the lines in Listing 3 to the `Application_Handler()` function. Remember that the CircleOS repeatedly calls the application handler function. Do not set up any infinite loops or time-consuming processes such as waiting for user input. Check to see if something has changed, react to it, and then exit, leaving a note for future generations (i.e., update the state variable).

According to the code sample

Listing 5—This is code for the application handler function.

```
if ( BUTTON_GetState () == BUTTON_PUSHED ) {
       BUTTON_WaitForRelease ();
       BUTTON_SetMode ( BUTTON_ONOFF_FORMAIN );
       return MENU_Quit ();
}

prescaler++;
if ( prescaler > 10 ) {

       switch ( leds ) {

       case 0: // both LEDs off
          LED_Set ( LED_GREEN, LED_OFF );
            DRAW_SetImage ( white_dot, 96, 32, 4, 4 );
            LED_Set ( LED_RED, LED_OFF );
            DRAW_SetImage ( white_dot, 80, 16, 4, 4 );
            break;

       case 1: // green LED on
          LED_Set ( LED_GREEN, LED_ON );
          DRAW_SetImage ( green_dot, 96, 32, 4, 4 );
          LED_Set ( LED_RED, LED_OFF );
          DRAW_SetImage ( white_dot, 80, 16, 4, 4 );
          break;

       case 2: // red LED on
          LED_Set ( LED_GREEN, LED_OFF );
          DRAW_SetImage ( white_dot, 96, 32, 4, 4 );
          LED_Set ( LED_RED, LED_ON );
          DRAW_SetImage ( red_dot, 80, 16, 4, 4 );
          break;

       case 3: // both LEDs on
          LED_Set ( LED_GREEN, LED_ON );
          DRAW_SetImage ( green_dot, 96, 32, 4, 4 );
          LED_Set ( LED_RED, LED_ON );
          DRAW_SetImage ( red_dot, 80, 16, 4, 4 );
          break;

       default:    // ??? unknown/unexpected state
          break;
      }

       leds++;     // move to next state
       leds %= 4;  // constrain values 0-3

       prescaler = 0; // reset prescaler
}
```

given, the button status is checked. If the button is being pushed, the application terminates by calling the `MENU_Quit()` function, which returns the `MENU_LEAVE` value to the CircleOS, informing it that the application is finished. If the button is not being pressed, the application simply returns the value `MENU_CONTINUE` and exits, resulting in the CircleOS taking control again, performing any sort of duties that it needs to take care of and then executing the application handler again. Again note that the application handler gets in, gets out, and gets done, all quickly. This is the basis of CircleOS application development.

Let's add a little more code to this example before moving on, both to make the program more usable and also to give you a taste of the other library functions available. Let's add a short instructional message to the screen in the application initialization function:

```
DRAW_DisplayString( 8, 60,
"Press the Button", 16 );
```

This is program initialization code within the `Application_Ini()` function. The function call writes a message on the LCD using the default colors. The first two parameters are the X and Y coordinates on the screen, in the first quadrant of the Cartesian plane, followed by the text and then the length of the text in characters. This draws the message in the center of the screen.

Now let's look at a slightly more complex example that uses a simple state machine to keep up with what's going on. Create a new CircleOS project for your particular Primer flavor, then add this single line of code to the `Application_Ini()` function:

```
DRAW_DisplayString( 16, 60,
"The LEDs blink", 14 );
```

This is the `Application_Ini()` function for the CircleOS version of the LED blink example program.

Also add the following global variables just before the `Application_Handler()` function (see Listing 4).

Next add code to the `Application_Handler()` function (see Listing 5). Notice some "static" variables defined at the top of the application handler code. These are preserved between calls to the function and are used to store the program state (LEDs) and a prescaler value.

Next comes the same exit mechanism used in the previous example. This works great as long as you don't need to use the button for anything else in your application. I have also used input from the MEMS accelerometer to exit the application when the unit was turned upside down, reminiscent of the Etch-A-Sketch I had as, ahem, a small boy. OK, I still have it.

Within the body of the application handler, the prescaler value is incremented. Once it exceeds a preset value, the current value of the state variable is used to determine what action to take. This illustrates both the direct LED control functions as well as some of the bitmapped graphic functions that are available in CircleOS.

The function winds down with some housekeeping. The state variable is incremented but constrained to meaningful values within this application. The function returns the value `MENU_CONTINUE` to let CircleOS know that it wishes to continue executing. Again, let me emphasize that the code executes straight through with no looping or blocking and then returns to the operating system as quickly as possible. Take the opportunity to change "Application_Name" from "My App" to "Blink" or something equally clever, bearing in mind the eight-character size limit.

This should give you a good idea of what is involved with writing a proper CircleOS application. Raisonance has a habit of promoting coding competitions with large cash prizes so it might behoove you to take a peek at the past winners and see if you think your next great idea could impress the judges.

BEYOND THE PRIMERS

Once you've exhausted the potential of the STM32 Primers as development platforms, it's time for you to move on to the big guns, the STM32 evaluation boards from STMicroelectronics. Currently, there are two varieties, the STM3210B-EVAL and STM3210E-EVAL. Both offer more flexibility, peripherals, and, most importantly, connectors than the fun-loving Primers. These are the "gold standards" for STM32 development and fundamentally remove any excuse you might have for not inventing the next great gizmo that will change the world for the better. Get busy!

Dale Wheat (dale@dalewheat.com) is a full-time freelance writer working primarily with embedded systems and shiny things that blink or beep. Dale is married and the father of two adult children. He lives near Dallas, where he enjoys mowing two acres of grass in the summer and not mowing it in the winter. Find out what he's been up to at his personal web site www.dalewheat.com.

PROJECT FILES

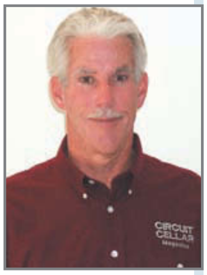
To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/230.

SOURCES

CircleOS
Raisonance
www.raisonance.com | www.stm32circle.com

STM32 Microcontrollers
STMicroelectronics | www.st.com

by Tom Cantrell



Neural Networker

A SNAP-Shot from Synapse Wireless

When it comes to embedded wireless, the good news is that there are plenty of alternatives to choose from. The bad news is that too many complicated options can lead to paralysis by analysis. This month Tom looks at a solution that puts all the answers—including the radio, protocol, and tools—on a little bit of silicon.

No doubt wireless gadgets (I call 'em "wadgets") are all the rage. Once again, we can thank Mr. Moore and the Silicon Wizards for delivering radio chips that slash cost and power consumption.

Indeed, maybe we're suffering from too much of a good thing. Recently, an on-line music store had a sale and I was thinking about picking up a guitar. But there were so many different models, finishes, and options that I got tired of window(s)-shopping and didn't buy anything.

Contemplate, if you dare, the ever-growing list of radios (e.g., Wi-Fi, Bluetooth, 802.15.4, and proprietary) and protocols (e.g., TCP/IP, ZigBee, 6LowPAN, and proprietary) that designers have to choose from. Grand concepts like "Internet of Things" and "Smart Dust" belie the complexity of the silicon and software under the hood.

Consider the saga of ZigBee. Its genesis nearly a decade ago was the concept of a simple, low-cost, low-power wireless network for embedded devices. Now, after nearly a decade of feature creep, the capabilities have grown quite a bit, but so has the complexity. The latest ZigBee Pro spec is 600 pages long, and that doesn't even include extras like the various "Profiles" (e.g., Metering, Home Automation, RF remotes, etc.), not to mention the documentation for the chips, modules, and boards that populate a particular ZigBee wadget.

Don't get me wrong. I'm not a ZigBee basher by any means. ZigBee is not only too big to fail, but already rolling out in real-world applications (notably metering) today. It's just that there never will be, nor does there need to be, a single radio and protocol standard. Presuming a modicum of decorum, there's no reason a variety of wireless solutions can't coexist. That's the beauty of radios compared to wired connections. We can all get along.

The good news is that all the choices and competition means there's something for everyone. The only problem is wading

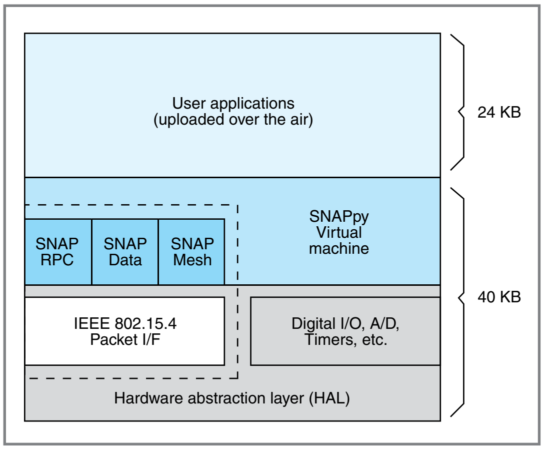

User applications (uploaded over the air)
24 KB
SNAP RPC
SNAP Data
SNAP Mesh
SNAPpy Virtual machine
40 KB
IEEE 802.15.4 Packet I/F
Digital I/O, A/D, Timers, etc.
Hardware abstraction layer (HAL)

Figure 1—By exploiting a virtual machine architecture, Synapse is able to fit everything, including the SNAP network stack and your application, into a low-cost 8-bit MCU.

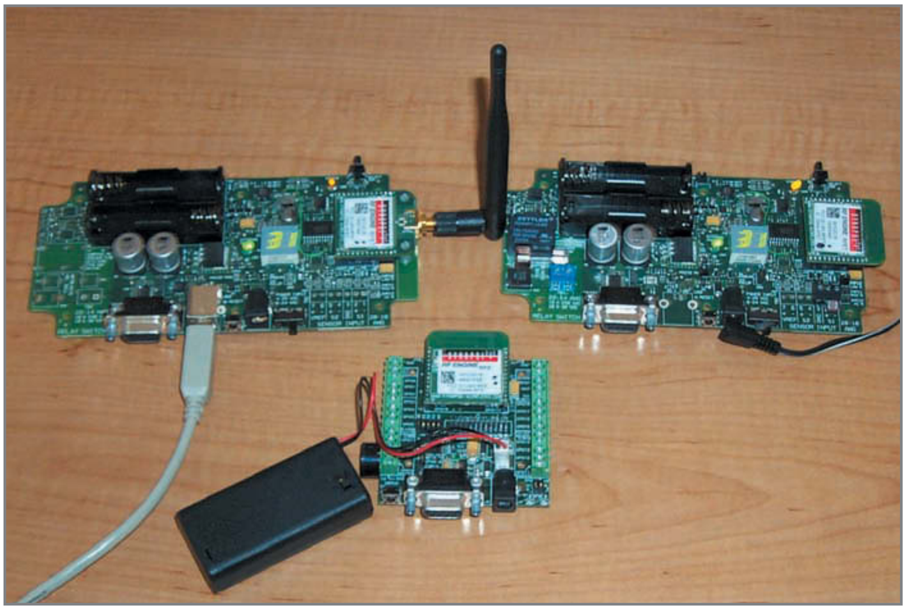


Photo 1—The Synapse EK2500 Evaluation Kit comes with three nodes, all based on their RF Engine radio module. Note the external antenna with transmit amp option on the "bridge" board, which also offers a USB connection. Just remember that with a SNAP network any node can act as a "bridge."

through all the options to find it.

SNAP TO IT

In the human brain, the synapse plays a critical role interconnecting the neuron processing elements. At this year's Embedded System Conference in San Jose, I discovered a company, aptly named Synapse Wireless, that hopes to do the same for silicon. Their secret sauce is software comprising the Synapse Network Appliance Protocol (aka "SNAP") and a unique "SNAPpy" programming environment.

Whimsical branding aside, there's some significant technology behind the scenes. And behind the scenes is where Synapse says it should stay. Their clearly stated mission is to keep it simple, and they do it pretty well. I found answers to a lot of questions and plenty of information to get started in Synapse's "SNAP Reference Manual" at just 127 pages.

SNAP manages to pack sophisticated "instant-on, self-forming, and self-healing" mesh networking into a small package. Going beyond just the network itself, SNAP also comes with a Python-based application development environment and "Portal," a PC/Mac/Linux-based network GUI.

At *Circuit Cellar*, as Steve Ciarcia has said, our favorite programming language is solder (i.e., hardware). Nevertheless, we all know every piece of silicon needs some (and inevitably some more) software to go with it.

Over the years, I've dabbled with many programming languages, but never Python. It's a relatively "new" language—first released some 20 years ago—that just now seems to be hitting its stride. Python is completely open source and supported on virtually every platform. (Now you know what to do with that Commodore Amiga in your closet.) You can find out everything you could possibly want to know about Python over at www.python.org.

I'm pretty agnostic when it comes to programming languages, having concluded it's possible to hack unreadable code in any of them. Some languages (you know who you are!) just encourage reckless behavior more than others. Maybe the "high" (somebody was) point was APL, with its literally "all-Greek-to-me" character set.

Having gotten my feet wet with some of the Synapse Python "SNAPpy Scripts," let me point out some of the language features that stand out. The syntax is pretty conventional with the usual operators and control flow. In some respects it reminds me of, dare I say, BASIC. Heck, you can even type in:

```
print 'hello world'
```

Also like BASIC, variables (Boolean, string, and 32-bit integers) don't need to be defined prior to use, and names are case sensitive, so watch out for typos. It can be hard to track down a misspelled variable name since there's no error message. I noticed there are no arrays per se, but it looks like string functions (e.g., CHR and STR) could be used to mimic a numeric array with a string.

One rather unique aspect of the language is that indentation—being used to define block structure—matters. This makes perfect sense conceptually because well-written code should use indentation for readability as a matter of course. In practice, it's also a feature that will cause some head scratching until you get used to it.

The biggest difference from other languages is the overall program structure. Rather than a sequential procedure (a la `main` in C), a SNAPpy script comprises a collection of functions that are "event-driven." Events include things like startup (i.e., power-up or software reboot), pin state change, 100-ms timer, data received on STDIN (e.g., UART), etc. Also notable, and most useful from a wireless-networking perspective, is built-in messaging with the "Remote Procedure Call" (RPC).

Although the Synapse SNAPpy subset of Python is somewhat downsized from full-feature computer implementations, it does seem capable in its own right. There are even some blue-collar additions for embedded apps with functions for parallel I/O, serial I/O (UART, I^2C, etc.), flash memory access, Sleep mode, and more. For the adventurous, PEEK and POKE statements provide direct access to the hardware.

As far as I'm concerned, the main feature isn't the language itself, but rather its interpreter-like virtual

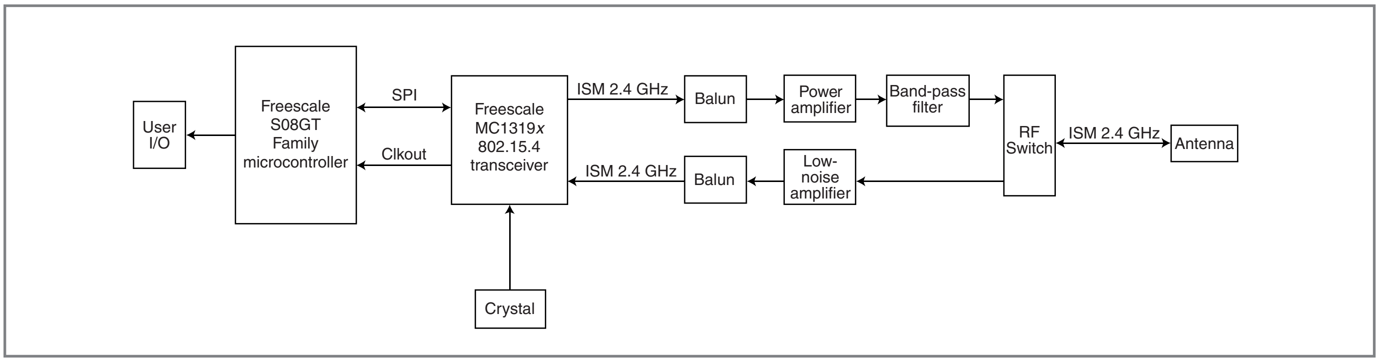


Figure 2—The Synapse RF Engine is a complete solution that's easy to use, certified, and ready to be deployed. It's also a bargain with prices starting at just $24.

machine implementation. Yes, execution speed is less than it is for a conventional compiled language, but there are some key benefits. Most notably, thanks to a small SNAPpy virtual machine and efficient tokenized code, the footprint is so tiny that everything (i.e., SNAP network stack, SNAPpy VM, and a meaningful application program) fits into a low-cost, 8-bit MCU (see Figure 1).

To be sure, there are some compromises, notably the limited MCU RAM available for data. Avoid the temptation to get overly verbose with string variables. Besides the fact that they quickly consume RAM, string functions such as concatenation and substring extraction (called "slicing" in Python-speak) must share limited-size buffers, which impose some constraints on your code. For instance:

```
String1 = "Use" + "it"
String2 = "or" + "lose it"
```

Once the second instruction executes, both `String1` and `String2` will equal "or lose it."

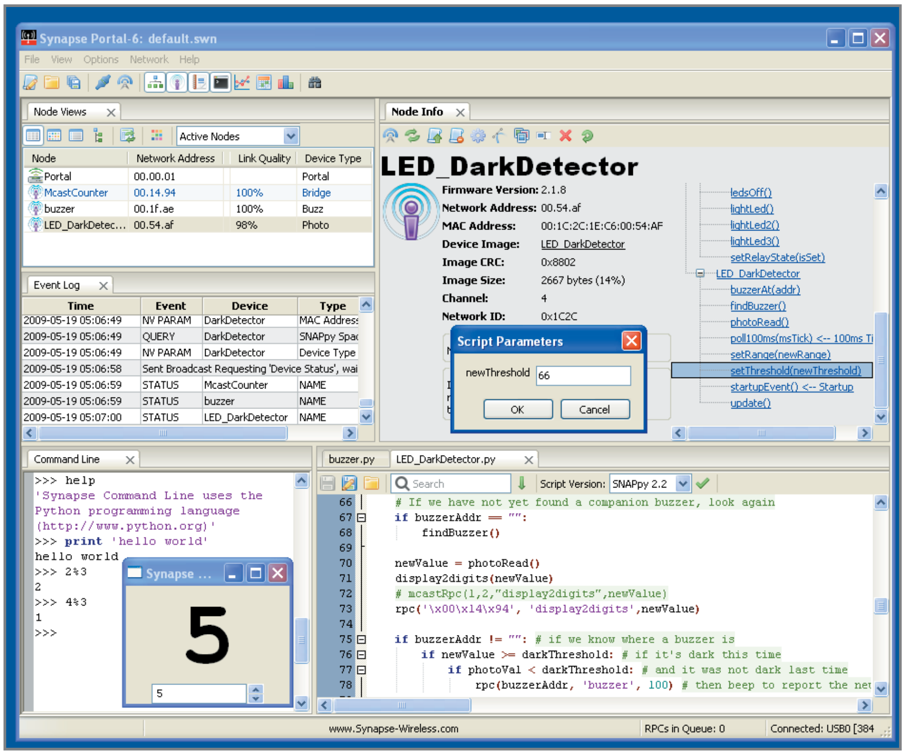
PORTAL POWER

Synapse was nice enough to send me an EK2500 evaluation kit, which comes with three nodes (see Photo 1). The heart of each is the Synapse RF Engine radio module (see Figure 2) featuring a pair of Freescale chips (8-bit MCU and 2.4-GHz 802.15.4 radio) under the hood. The module offers your application access to a decent mix of the MCU's I/O capability, 19 pins in total, including parallel and serial I/O, a timer, and an eight-channel, 10-bit ADC. One of the RF Engines in the kit comes with an external antenna (the others use PCB trace antennae) and an optional transmit power amplifier said to boost outdoor (i.e., line-of-sight) range to 3 miles.

The RF Engine gets decent gas mileage, running off a single 2.7- to 3.4-V supply, consuming 65 mA on average with the radio on and 20 mA with it off. When sizing your power supply, do note peak current (i.e., transmitting with the external power amp option) can surge to 110 mA. At the other extreme, in the deepest of Sleep modes, the module draws a mere 2.5 µA.

As you can see in the photo, the larger boards have I/O add-ons, including LEDs (seven-segment and discrete), switches, a relay, an optosensor, a buzzer, and the like. One of the boards, the so-called "bridge" (the one with the transmit amp and external antennae) has a USB interface that both powers the board and makes the connection between a PC (or Mac or Linux) and the SNAP network.

It's worth noting that there's nothing special (other than the convenience of



Node	Network Address	Link Quality	Device Type
Portal	00.00.01		Portal
McastCounter	00.14.94	100%	Bridge
buzzer	00.1f.ae	100%	Buzz
LED_DarkDetec…	00.54.af	98%	Photo

Event Log

Time	Event	Device	Type
2009-05-19 05:06:49	NV PARAM	DarkDetector	MAC Address
2009-05-19 05:06:49	QUERY	DarkDetector	SNAPpy Spac
2009-05-19 05:06:49	NV PARAM	DarkDetector	Device Type
2009-05-19 05:06:58	Sent Broadcast Requesting 'Device Status', wai		
2009-05-19 05:06:59	STATUS	McastCounter	NAME
2009-05-19 05:06:59	STATUS	buzzer	NAME
2009-05-19 05:07:00	STATUS	LED_DarkDetector	NAME

Node Info

LED_DarkDetector

Firmware Version: 2.1.8
Network Address: 00.54.af
MAC Address: 00:1C:2C:1E:C6:00:54:AF
Device Image: LED_DarkDetector
Image CRC: 0x8802
Image Size: 2667 bytes (14%)
Channel: 4
Network ID: 0x1C2C

ledsOff()
lightLed()
lightLed2()
lightLed3()
setRelayState(isSet)
LED_DarkDetector
buzzerAt(addr)
findBuzzer()
photoRead()
poll100ms(msTick) <-- 100ms Ti
setRange(newRange)
setThreshold(newThreshold)
startupEvent() <-- Startup
update()

Script Parameters
newThreshold 66
OK Cancel

Command Line

```
>>> help
'Synapse Command Line uses the
Python programming language
(http://www.python.org)'
>>> print 'hello world'
hello world
>>> 2%3
2
>>> 4%3
1
>>>
```

Synapse …
5

buzzer.py LED_DarkDetector.py
Script Version: SNAPpy 2.2

```
    # If we have not yet found a companion buzzer, look again
    if buzzerAddr == "":
        findBuzzer()

    newValue = photoRead()
    display2digits(newValue)
    # mcastRpc(1,2,"display2digits",newValue)
    rpc('\x00\x14\x94', 'display2digits',newValue)

    if buzzerAddr != "": # if we know where a buzzer is
        if newValue >= darkThreshold: # if it's dark this time
            if photoVal < darkThreshold: # and it was not dark last time
                rpc(buzzerAddr, 'buzzer', 100) # then beep to report the ne
```

www.Synapse-Wireless.com RPCs in Queue: 0 Connected: USB0 [384

Photo 2—"Portal" is the one-stop-software that handles every aspect of your wireless design, including network management, connecting with PC services (via Portal scripts), and developing your own "SNAPpy" applications.

the USB interface) about a SNAP "bridge." The SNAP network is completely peer-to-peer, and any node that can connect directly to the PC (e.g., via RS-232 port) can act as a "bridge." There are no "coordinators," "hubs," or other specialized controllers, nor the risk they pose of a single loco chip bringing the entire network down.

"Portal" is the one-stop-shop that gives you command and control of all aspects of your wireless application. I tried to craft a screenshot showing many of the features in Photo 2, so let's take a closer look.

Starting at the top left you see the "Node Views," which shows this network comprises the three nodes that come in the kit. Actually, if you count Portal running on the PC, there are four nodes, but more on that in a moment. Underneath the "Node Views" is the "Event Log" that keeps track of what's going on with the protocol and monitors message traffic. On the bottom left, there's a Portal (i.e., Python) "Command Line" window that's handy for trying things (e.g., here reminding myself how the "%" modulo operator works).

Clicking on a node, in this case "LED_DarkDetector," brings up the "Node Info" window on the top right showing a node's particulars, including the name of the script it is running with a list of every function together with built-ins and those defined by your application.

A very cool feature for testing and debugging is that you can simply click on a function name to run it. This demo has one of the nodes that is equipped with a photosensor acting as a 0 to 100% light-level detector. Here I've clicked on the `setThreshold` function and a pop-up window is asking for the `newThreshold` argument, which I'm entering as 66. Sure you could edit the script to add a `setThreshold(66)` statement and re-flash, but it's a nice convenience for testing and experimenting to be able to tweak things with a button click.

Just clicking on the node's image name (i.e., LED_DarkDetector) opens the script for editing. It's a simple matter to make a change, recompile, and upload the new version over the air. And it's fast, dare I say snappy, with a complete blow-and-go cycle taking just a few seconds. Admittedly, these are small demo programs, maybe 50 lines or so, but that's enough to do something useful.

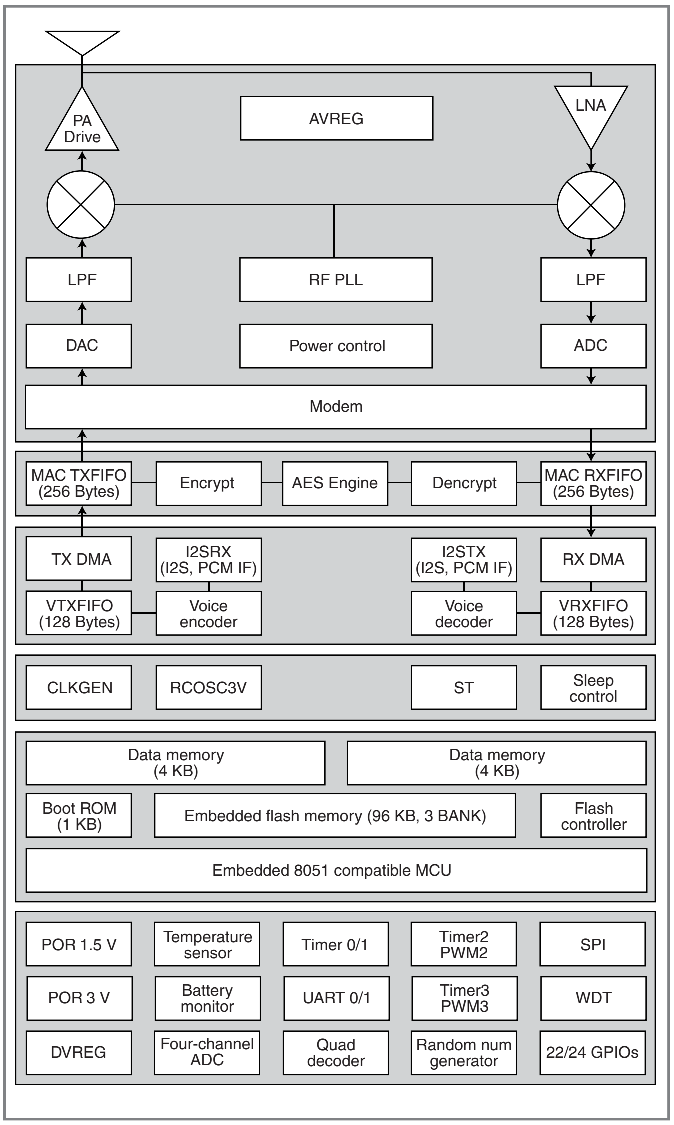


Figure 3—Using the ZIC2410 module from CEL, I was able to put SNAP interoperability claims to the test, which it passed with flying colors.

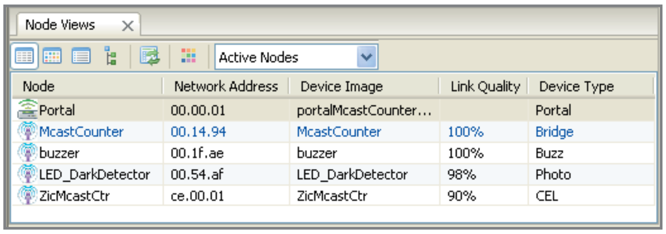
Indeed, noting the tiny code size, just a couple of thousand bytes, you can see a little flash will go a long way.

That brings us to the final window, the one showing "5" on the bottom left. I mentioned earlier that the PC itself, via the bridge, becomes a fourth node in this network. What's happening here is that the `McastCounter` script—running on the bridge (but it could be on any node)—is monitoring a push button and incrementing a counter. Each time the button is pushed, the new count is "multicast" across the network. In turn, the Portal itself (i.e., PC) can run a script—in this case, one that grabs the new count and displays it in a window on your PC screen.

The Portal script uses full computer-grade Python giving any node, even the simplest one, the ability to access a full range of PC services. Besides doing stuff on the screen as shown here, a Portal script could, for example, fire off an e-mail or update a PC file if a remote sensor signals an alarm condition. Portal scripting is made even more versatile with a script-scheduling capability. For instance, you could have a Portal script that polls the network on a regular schedule (every minute, hour, day, etc.) and sends a summary report in an e-mail.

CASTING CALLS

To be sure, the SNAPpy language has its share of eccentricities and the implementation on an 8-bit MCU is a bit constrained. Script execution speed is a modest 11.4K "instructions per second" according to the Synapse documentation. Synapse quotes 1.9 kHz as the maximum GPIO pin toggle rate, and 5 kHz for sampling the ADC, and that's optimistic (i.e., no



Node	Network Address	Device Image	Link Quality	Device Type
Portal	00.00.01	portalMcastCounter...		Portal
McastCounter	00.14.94	McastCounter	100%	Bridge
buzzer	00.1f.ae	buzzer	100%	Buzz
LED_DarkDetector	00.54.af	LED_DarkDetector	98%	Photo
ZicMcastCtr	ce.00.01	ZicMcastCtr	90%	CEL

Photo 3—Another benefit of a virtual machine approach is that it's easy to support multiple hardware platforms. Synapse and CEL nodes happily coexist on the SNAP network even though the hardware under the hood is completely different.

allowance for additional processing of the data). Minimum network latency is on the order of 5 ms, plus at least 20 ms for each extra hop. And as I mentioned earlier, the limited MCU RAM can crimp your style if you're not careful.

Nevertheless, as far as I'm concerned, any complaints you might have with the SNAPpy language itself are more than offset by the effectiveness and ease-of-use of the Portal development environment. Presuming SNAPpy can do what you want, it's a really easy way to do it.

Putting aside language ideology, it's the "Remote Procedure Call" (aka "RPC") capability of SNAPpy that is the fundamental basis for wireless networking applications, so let's turn our attention there.

Just like a regular procedure call, an RPC specifies a function name and arguments. The only difference is the "remote" part of the equation, namely the network address of the node where that function resides and will be executed. So the following statement will invoke the `display2digits` function with the argument `newValue` on the node at address 00.14.94:

```
rpc('‘\x00\x14\x94','display2digits',newValue)
```

A SNAP node address is the least significant 3 bytes of its MAC address, so I guess a network could conceivably encompass 16 million nodes (don't try this at home). It's all well and good for a script running on one node to ask another node to do something. But how do you know the requested function was actually performed, or for that matter whether the request even got through? According to the documentation, the SNAP protocol will retry sending the message up to eight times before giving up, but apparently doesn't report failures back to the sender. To be safe, the node being requested to perform a function should report back a completion status. To do that, it needs to know the node address of the requestor, which SNAPpy makes available with the `rpcSourceAddr` function. So continuing the above example, at the end of the `display2digits` function on node 00.14.94 (i.e., after the LEDs are updated with `newValue`), the following line closes the loop by calling the `display_is_updated` function on whichever node initiated the `display2digits` RPC:

```
rpc(rpcSourceAddr(),'display_is_updated')
```

There's a "callback" capability that streamlines the handshake process (i.e., the requesting node could execute):

```
rpc('‘\x00\x14\x94','callback','display_is_upd
ated','display2digits',newValue)
```

So far, we're talking about communication between two individual nodes. As mentioned earlier, SNAP networks also support "multicasting" (i.e., the ability for a node to send an RPC to many other nodes with a single SNAPpy `mcastRpc` statement). It's like the `rpc` statement in that it specifies the remote function name and arguments. But in addition, there are "group" and "time_to_live" parameters that specify which subset of nodes will see the RPC. SNAP networks are logically divided into 16 possible groups, and an individual node can direct an RPC to one or more of them, as well as be a member of more than one group. Group number 1 is used for "broadcasting" an RPC to every node on the network, regardless of its group affiliation, only limited by the "time_to_live," parameter, which specifies the allowed hop count. So, for example, the following would display `newValue` on the LEDs of every node within two hops of the requestor:

```
mcastRpc(1,2,'display2digits',newValue)
```

The documentation doesn't say much about the `time_to_live` parameter. I presume it's meant as kind of a network-wide jabber control, or maybe it's meant to serve as a range limiter should two SNAP networks happen to overlap. In any case, if the actual routing information is available, I couldn't find it in Portal or the documentation. All the demo examples just use 2 for the `time_to_live` parameter, which is fine for playing around on a lab bench where multihopping isn't necessary

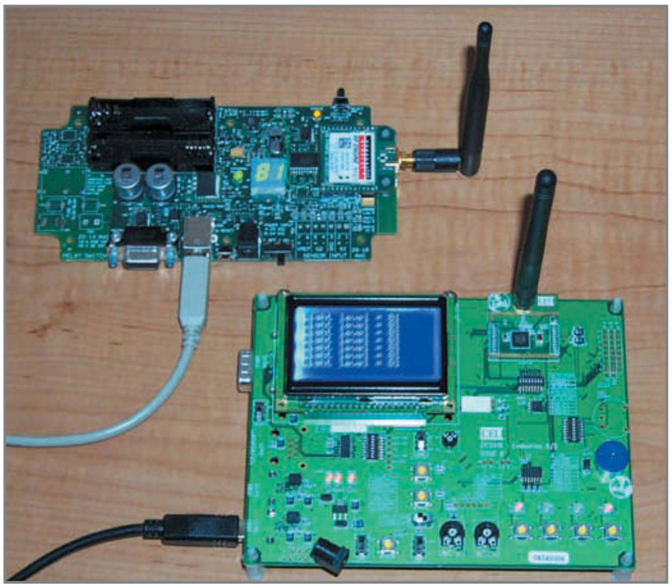


Photo 4—Interoperability in action. The fact SNAP can, and is, being ported to multiple vendors' hardware gives it an advantage over "sole source" solutions.

PCB WEST 2009
Conference & Exhibition

PCB West is the premier conference and exhibition for the PCB supply chain, including engineers, designers, fabricators, assemblers and managers

PCB West returns for its 18th year to the Silicon Valley with more reasons than ever to attend:

- Over 30 Professional Development and Technical Conferences courses
- Technical presentations on critical topics, including:
  - RF design
  - Component integrity
  - Libraries
  - HDI
  - Design basics
  - Via reliability
  - Routing
- Two-day exhibition featuring the industry's leading vendors
- Expanded free technical sessions on Tuesday and Wednesday
- Networking opportunities with coffee breaks, a complimentary lunch on the show floor on Tuesday and Wednesday, and an Opening Night Reception on Tuesday evening

Conference: September 14 – 18, 2009

Exhibition: September 15 – 16, 2009

Santa Clara Marriott • Santa Clara, CA

Visit www.pcbwest.com to request a 2009 Conference Brochure!

and presumably never happens.

Although keeping track of who's saying what to whom can be a little confusing, the RPC scheme can pull off some interesting tricks. For example, the `LED_DarkDetector` function doesn't just display the light level on another node's LEDs. It also beeps a buzzer on the third node when a programmable threshold is crossed. The cool thing is that the node with the light sensor doesn't know the address of the node with the buzzer, but rather it finds it using a `findBuzzer` multicast RPC. When the node with the buzzer (and hosting the `findBuzzer` function) sees the broadcast request, it uses the `rpcSourceAddr` function to ascertain which node it came from. Knowing who's asking, the buzzer node replies with a `buzzerAt` RPC to send back its own address. Now the node requesting buzzer service can use the address it "found" at runtime to communicate directly (i.e., unicast) with the buzzer.

It's an interesting model, kind of an embedded version of "cloud computing," that has the intelligence of the system comprising the entire collection of functions distributed across the network.

THE MORE THE MERRIER

A proprietary protocol like SNAP raises the specter of getting locked into a sole source for hardware. Not so with SNAP. Sure, Synapse will be glad to sell you some of their RF Engines to go with, but interestingly (and, I'd say, wisely), they're more than willing to get SNAP up and running on other vendors' hardware. Indeed, since it's relatively easy to port the SNAPpy virtual machine and hardware abstraction layer, Synapse will even help you get it running on your own hardware if that makes sense.

In my in-basket, I've got a full-blown ZigBee evaluation kit from CEL (California Eastern Labs) that's based on their own MCU+802.15.4 radio SoC. Their ZIC2410 radio module (see Figure 3) is similar in appearance and capabilities to the Synapse RF Engine, although it is completely different under the hood (e.g., CEL uses an 8051-type MCU versus the Synapse Freescale MCU). In addition to serving as the platform for their ZigDee solution, CEL also has SNAP up and running on their gear.

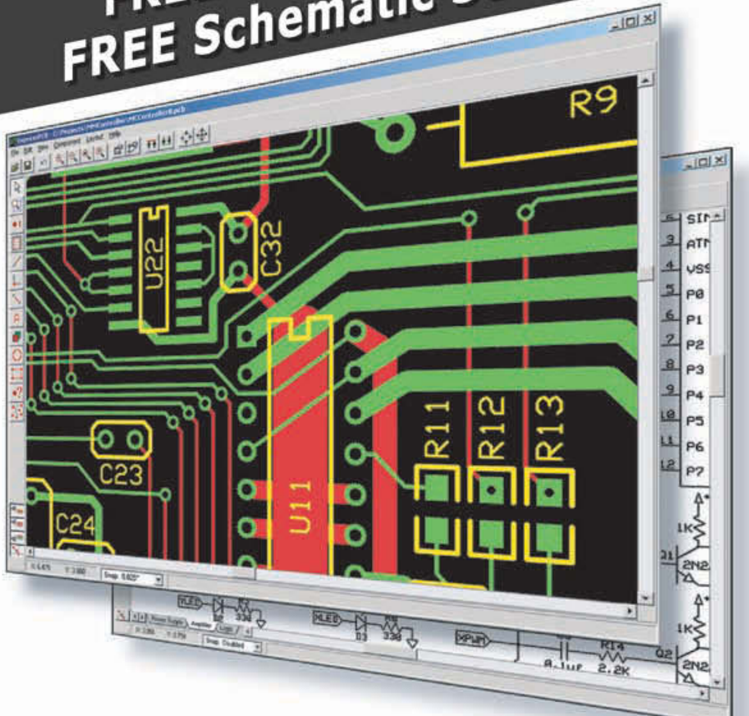
Time to run a little interoperability test and, ta-da, everything worked as advertised. Portal instantly recognized the CEL board and added it to the network (see Photo 3). I was even able to unplug the Synapse board from the PC and plug in the CEL board and it was able to take over the "bridge" role. It was easy to use built-in functions to bring the CEL boards specific I/O capabilities, such as the LCD, into the application mix (see Photo 4).

RANGE ROVER

To be sure, in the short time I spent with the Synapse gear, I wasn't able to probe the limits in terms of script complexity, node/hop count, radio range, adaptive routing, and so on. It wouldn't surprise me at all if

$51 For 3 PCBs

FREE Layout Software!
FREE Schematic Software!



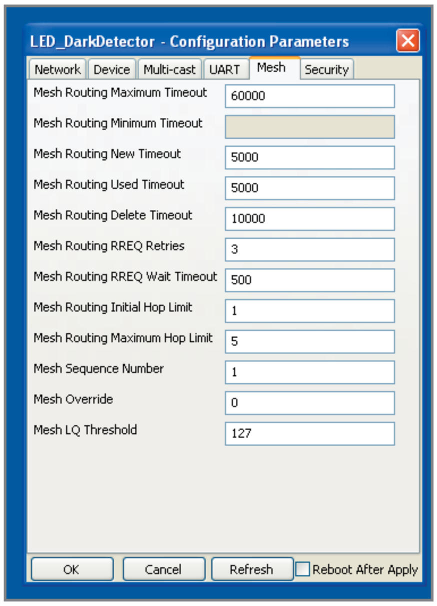
01 DOWNLOAD our free CAD software

02 DESIGN your two or four layer PC board

03 SEND us your design with just a click

04 RECEIVE top quality boards in just days

expresspcb.com



Mesh Routing Maximum Timeout	60000
Mesh Routing Minimum Timeout	
Mesh Routing New Timeout	5000
Mesh Routing Used Timeout	5000
Mesh Routing Delete Timeout	10000
Mesh Routing RREQ Retries	3
Mesh Routing RREQ Wait Timeout	500
Mesh Routing Initial Hop Limit	1
Mesh Routing Maximum Hop Limit	5
Mesh Sequence Number	1
Mesh Override	0
Mesh LQ Threshold	127

Photo 5—Sure there's complexity to be found if you dig deep enough. But Synapse handles it the right way (i.e., it's there if you need it, but not a bother if you don't).

some glitches emerged under stress testing. But that caveat is true for every wireless lashup. The real point is that within the short time spent, I was fully able to grasp the scope of SNAP and get pretty far along the learning curve.

That's something that can't be said for most of the wadgets I've come across, which seem to struggle under the combined weight of protocol complexity, network management hassles, and bloaty software tools. Yes, drill down through the menus of Portal and you'll find some complicated underpinnings (see Photo 5), but Synapse doesn't rub your nose in it.

The bottom line is that I'm impressed. Synapse deserves credit for thinking outside the box and delivering a unique solution that integrates every aspect of an embedded wireless project. All the better their willingness to promulgate SNAP onto other suppliers' hardware, a potential win-win-win for everyone involved (i.e., Synapse, other radio suppliers, and users). ■

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.

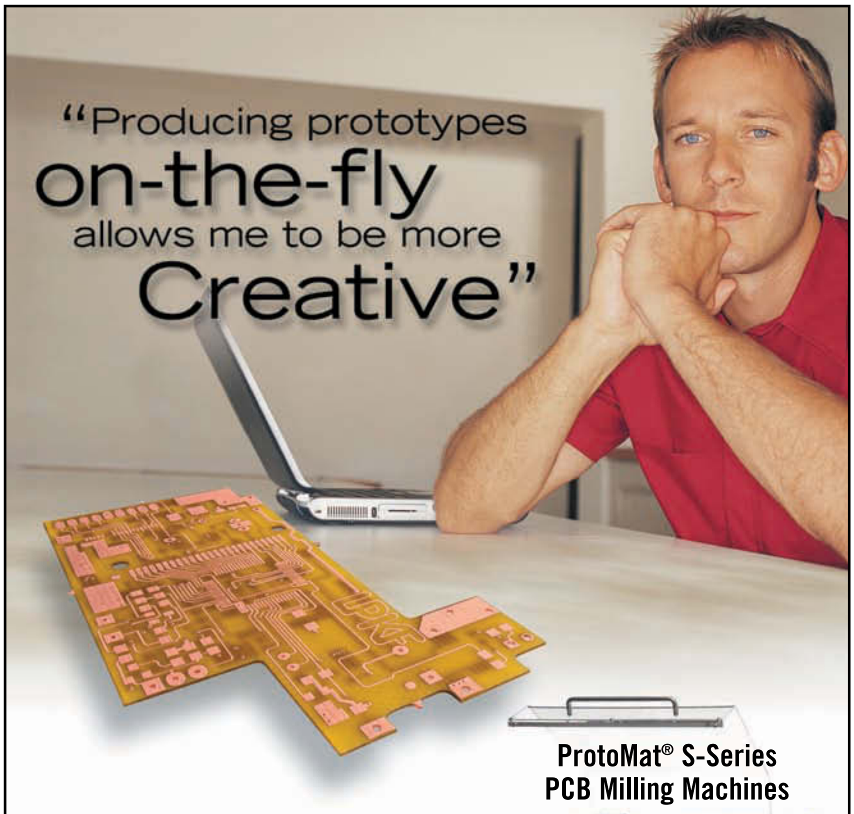
Resource

Synapse Wireless, Inc., "SNAP Reference Manual," 600-0007B, http://forums.synapsewireless.com/upload/SNAP%20Reference%20Manual.pdf.

Sources

ZIC2410 Radio module and ZigBee development kit
California Eastern Laboratories | www.cel.com

EK2500 Evaluation kit, SNAP Wireless network, and RF Engine radio module
Synapse Wireless, Inc. | www.synapse-wireless.com



ProtoMat® S-Series PCB Milling Machines

Electrical engineers agree: with a Protomat S-Series prototyping machine at your side, you'll arrive at the best solutions, fast. These highly accurate benchtop PCB milling machines eliminate bread-boarding and allow you to create real, repeatable test circuits—including plated vias—in minutes, not days.

- **Declare your independence from board houses**
- **Affordable, entry-level price tag**
- **The best milling speed, resolution, and accuracy in the industry**
- **Single-sided, double-sided, and multilayered machining without hazardous chemicals**
- **Optional vacuum table and autosearch camera for layer alignment**

LPKF® Laser & Electronics

For complete details visit:
www.lpkfusa.com
or call:
1-800-345-LPKF

by Brian Millier

Smart Power Bar

Simple Circuitry Enables Energy Conservation

In-home energy conservation is just a project away. Power bars are used to turn off several electronic devices at the same time. Brian recently designed his own to meet his needs. The compact design consists of a conventional 60-Hz power supply, an MCU, a relay, and an IR module.

Have you heard the song "Little Green Apples," which Bobby Russell wrote in 1968? I was a teenager back then, so I remember the song well—although I can't recall whether it was Roger Miller's or O.C. Smith's version that first drew my attention. Today, I have some "little green" things of my own: design projects. Recently, I've been working on "green" projects that all concern energy conservation in some way. I've always been a big fan of energy conservation, even before it became trendy. Why? It's a great way to save money, and I'm a big fan of that!

In this article, I'll present one of my most recent energy-conserving designs: a "smart" power bar (see Photo 1). This project proves that simple Atmel ATtiny45-based circuitry can be a real energy saver.

ENERGY CONSERVATION

In Nova Scotia, Canada, where I live, the government and utilities have just recently become interested in promoting alternative energy. Historically, we mined coal in the province and burned it to generate fairly cheap electricity, but current government initiatives are aimed at promoting large wind farm installations and convincing the electric utility (a monopoly) to pay a premium for such energy. In my area, if a private homeowner wants to install a modest grid-tied wind turbine or solar panel array, he must be prepared for a long pay-back period because there is little available in the way of tax incentives and the like. I can't say I disagree completely with that premise. There is certainly something to be said for the efficiencies gained when wind turbines are built on a utility scale, and I also wonder how difficult it will be for the average homeowner to maintain wind turbines and solar arrays after they start to show their ages.

With the aforementioned concepts in mind, I've always kept an eye out for ways to reduce the amount of electricity I use at home. The big change I made about seven years ago was to install electrical energy storage units to replace our in-ceiling electric radiant heating. These units draw all of their electricity needs over-night (and during weekends and holidays) and store all the heat in bricks inside the unit. A small fan in the unit then slowly distributes the heat throughout the day. It's all microprocessor-controlled,

Photo 1—The smart power bar unit is perched on top of a home theater speaker next to my TV. A commercial multi-outlet power bar plugs into the switched power cord, which exits from the back of the unit.

efficient, and comfortable. I pay only 50% of the standard electricity rate for this off-peak power. The control system also runs our hot-water heater, clothes washer, dishwasher, and so on by activating them at night and on weekends. While we are not saving any overall energy with this system, we draw about 75% of our electricity during the off-peak hours, which the power utility appreciates and as a result gives us half-price power as a reward.

I am also interested in "phantom power." This is electricity drawn by consumer appliances and entertainment units when they are ostensibly turned off. This is something that has been getting out of hand in recent years as more and more product manufacturers have done away with a "real" power switch and instead have begun incorporating circuitry that remains in a Standby mode when turned off. (Thus, it's able to respond to either an IR remote control or the signal from the tiny momentary contact switch now used for the "power switch.")

A number of electronic devices in the modern home are designed to remain running all of the time. For instance, I don't want to go down to my basement every time I need to turn on and off my cable modem and wireless router. In addition, I don't think I could do without Caller ID to screen out telemarketers, so my phone is powered all the time via a wall wart. And then there's the personal video recorder that automatically records all of our TV shows. It needs to remain running, at least in Standby mode, at all times. I think my wife would leave home if I were to remove it for the sake of energy conservation.

There are, however, some areas where phantom power can be minimized. My home office computer setup is quite substantial, with a powerful desktop computer running dual 20" LCD monitors, a black-and-white laser printer, and a color Inkjet printer. With everything turned off, this setup still draws about 16 W, which would cost about $14.85 per year at $0.106 per kilowatt-hour (local pricing). Therefore, I connect the entire system through a power bar. I turn off the power bar whenever I am not using my computer. In addition to saving electricity, this scheme also minimizes the threat of damage to the electronics, which often happens as a result of power surges that occur during storms and other power outages.

Like many people, I have a substantial home entertainment system. It consists of a satellite receiver/PVR, a widescreen CRT-type TV, a powerful home theater 5:1 surround sound receiver, a powered subwoofer, DVD player, and an old VCR. Factoring out the satellite receiver/PVR—which must remain powered all of the time—leaves five units that I can turn off to save power. I initially did this using a conventional power bar, but that was somewhat inconvenient and unsightly because it required the power bar to be accessible. Thus, I had an entire rat's nest of power cables to plain view. Such a setup was unacceptable. So, after a bit of thought, I came up with the smart power bar solution.

SMART SOLUTION

My idea for a smart power bar arose from the observation that IR remote controls can control most home entertainment devices. I asked myself: Why not let the presence of IR remote control codes be the trigger that turns on the smart power bar? This would mean that I would push my normal start-up button—but do so twice. The first tap would turn on the system's power via the power bar. By tapping the button again 1 s later, I would activate the device I want to turn on.

In each individual home entertainment device, the on-board microcontroller's program logic decodes only the specific IR codes that refer to a particular unit. Many companies make home-entertainment products, and most of them use either their own proprietary IR coding schemes or some variation on a few standard protocols. You've seen universal IR remotes—which are readily available for about $15—and it would seem like a reasonable assumption that there can't be that many different IR codes in use. You're wrong if you assume this. The only reason such universal IR remotes are so common is that companies like Zilog have exhaustively compiled the IR codes from the bulk of the products available and sell application-specific microcontrollers with large amounts of ROM, which contain the bulk of these codes. As a result, universal IR remote manufacturers customize generic microcontrollers and use them in products (or they buy the IR code library from companies like Zilog).

Given the diversity of IR codes, how does the smart power bar actually work? To begin with, I start with the premise that I don't really care what particular IR code I am receiving. The presence of any IR code is evidence in itself that I am trying to use at least one of the devices powered by the smart power bar. Because I don't care about the actual code, my main concern is that there is a repeated pattern of IR being received. The reason to look for a pattern—instead of merely the presence of IR—is that sunlight and normal room lighting contains IR. Even in the absence of a real signal from an IR remote, the IR receiver still produces spurious signals.

There are many encoding schemes used by different types of IR remotes. But while testing the remotes I use, I found in general that the codes were made up of pulses that were either all the same width or there were two or three discrete widths. Often the pattern would start with a start or sync pulse of a certain length, followed by a datastream using two different pulse widths to represent the ones and zeros in the code. Sometimes only one width was used for the datastream, with the time between pulses representing the bit values of the code. In any case, when capturing the IR codes for a short period of time and analyzing the distribution of pulse widths, I found that certain pulse widths generally show up regularly.

To handle this scenario, the microcontroller interprets the incoming IR datastream. Timer1 triggers an interrupt that sets a flag every 0.26 s. During this time, the IR receiver module's signal output is constantly monitored for any transitions. The time between the start and end of each IR pulse is measured, and up to 20 such time intervals are stored in a RAM array. If the 0.26-s interval passes without 20 pulses being detected, then basically nothing happens and the process restarts. However, once 20 pulses are detected, we enter a nested loop structure in which we compare each pulse width to that of all the other pulses. I use a timer with a 32-µs timing interval to measure the IR pulses. This resolution is small enough to differentiate the different pulse widths without being so finicky as to notice the slight width variations that occur naturally. Experimentally, I found that this algorithm worked well if I set a criterion of finding five or more equal width pulses amongst the 20 collected within the 0.26-s interval.

Once this criterion is met, the smart power bar activates the relay that supplies power to the attached devices. At the same time, it starts a 2-h countdown timer. As long as I occasionally hit any of the remote control buttons (e.g., the channel select or volume buttons), the microcontroller recognizes these codes—in the same way it did the initial one—and it reinitializes the countdown timer back to the 2-h point. (It also activates the relay. But because the relay is already activated at this point, nothing changes in that respect.) However, 2 h after the reception of the last IR code, the smart power bar turns off the relay, thus powering down everything. I started with a 1-h countdown interval, but my entire system shut down occasionally while I was watching long movies (during which times the remote remains untouched). That's why I upped the timer to 2 h.

I neglected to mention an important detail of IR remote control operation. To greatly improve the reliability of IR remote control code transmission, the IR light is not modulated directly by the data pulses. Instead, the IR light beam is modulated at a fixed ultrasonic rate (36 to 56 kHz depending on the model). The carrier frequency is pulsed on and off with the proper pattern required by the IR code. Built into the IR receiver is a demodulator (detector) followed by a band-pass filter, which filters out IR signals that are not modulated at the proper rate. This filtering drastically reduces the IR remote receiver's susceptibility to random light variations. IR receiver modules are sold in various models, with each model being specified for a certain carrier frequency. The band-pass filters in these receivers are reasonably broad. I chose a model based on a carrier frequency of 38 kHz that can respond to the different

Listing 1—This is the entire BASIC code for the smart power bar.

```
' Smart Power Bar
' C 2008 Brian Millier
$regfile = "ATtiny45.dat"
$crystal = 8000000
Dim Secondflag As Boolean
Dim Hourcount As Integer
Dim Transition(50) As Byte
Dim X1 As Byte
Dim X2 As Byte
Dim Index As Byte
Dim I As Byte
Dim J As Byte
Dim K As Byte
Dim Maxcount As Byte
Dim Count As Byte
Dim Temp As Byte
Const True = 1
Const False = 0
Config Pinb.0 = Output    'Relay drive - positive logic
' Use Timer 0 for IR transition timing
Tccr0a = 0
Tccr0b = &B00000100 ' use 8 mhz clock /256 = 32 us clock
' period use timer 1 for the 0.26 second clock interrupt
Tccr1 = &B00001110 ' Use 8 Mhz clock / 8192 (0.262 second timer overflow)
Timsk = &B00000100 ' enable Timer 1 overflow interrupt
Reset Portb.0
On Timer1 Tickisr:
Enable Interrupts
Wait 1
Secondflag = False
Hourcount = 0
Do
      St:
      Secondflag = False
      For Index = 1 To 20
         Do
           Temp = Pinb
           Temp = Temp And 8
           If Secondflag = True Then Exit Do
           If Temp = 0 Then
             X1 = Timer0
             If Index = 1 Then Secondflag = False
             Exit Do
           End If
         Loop
         If Secondflag = True Then Goto St:
         Do
           Temp = Pinb
           Temp = Temp And 8
           If Secondflag = True Then Exit Do
           If Temp = 8 Then
             X2 = Timer0
             Exit Do
           End If
         Loop
         X2 = X2 - X1
         Shift X2 , Right , 1 ' ignore bobble in LSB of time
         Transition(index) = X2
         Next
         Maxcount = 0
         For I = 1 To 19
           Count = 0
           K = I + 1
           For J = K To 20
             X1 = Transition(i)
             X2 = Transition(j)
             If X1 = X2 Then
                Incr Count
             End If
           If Count > Maxcount Then
                Maxcount = Count
           End If
         Next
 Next
         If Maxcount > 5 Then
                Set Portb.0
                Hourcount = 0
         End If
Loop
Tickisr:
         Secondflag = True
         Incr Hourcount
         If Hourcount > 28000 Then     ' 2 hour delay
           Reset Portb.0
           Hourcount = 0
         End If
 Return
```

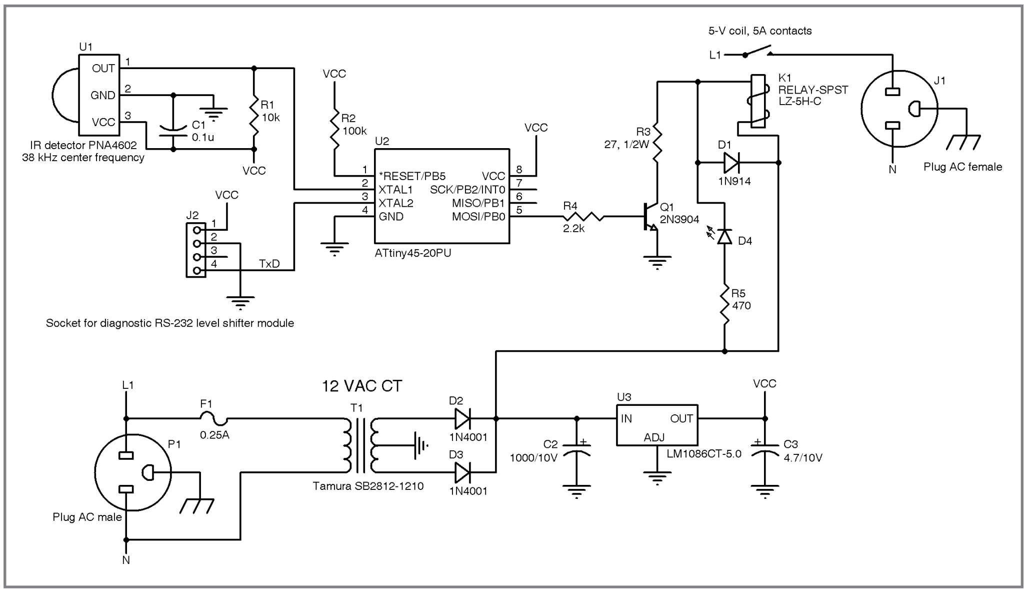


Figure 1—The circuit couldn't get much simpler than this: a conventional 60-Hz power supply, an ATtiny45 MCU, a relay, and an IR module.

IR remotes that I have. I don't believe the higher 56 kHz remotes are too common. But if your main remote uses that frequency, you might have to substitute an IR receiver that's based on that higher carrier frequency. Also, the particular IR receiver module I used presents a high output signal in the absence of a modulated IR signal—and drops low during the incoming IR carrier pulses. The microcontroller firmware is tailored to this active-low receiver signal. If you substitute a module with a different output sense, the firmware must be modified at two places (see Listing 1).

CIRCUITRY

The circuit is simple (see Figure 1). I used a Panasonic PNA4602 IR receiver, but availability of specific IR receivers changes quickly so you can use a different one if you want. However, most modules are open collector, so don't forget to include R1, the pull-up resistor.

For this application, I could've used the smallest available microcontroller. I routinely use Atmel microcontrollers, and its ATtiny family includes several members that fit the bill. The less powerful members of the family don't include SRAM—just the 32-byte register bank present in all AVR chips. I prefer to program using the BASCOM-AVR Basic compiler, which really needs SRAM to do anything meaningful. Therefore, I seldom use the less-capable members of the ATtiny family. So, I chose an ATtiny45-20PU with 4 KB of program flash memory and 256 bytes of SRAM—which was more than twice what I needed. An ATtiny25—containing half the RAM and flash memory—would have worked just as well, but it is only $0.50 cheaper and I don't bother to stock it.

I run the ATtiny45 at 8 MHz, using its internal RC clock. This eliminates the need for a crystal. At this speed, it draws about 5 mA, which could be reduced a bit with the selection of a lower clock frequency. However, there is little to be gained by lowering the MCU power consumption below the 5 mA unless you also incorporate a power supply with extremely low quiescent power consumption.

Power supply design depends a lot on your ultimate design goals. If you're manufacturing thousands of low-power devices and going for Energy Star efficiency, there's no doubt that the best bet is to use a high-frequency switching design. If you're building a prototype, there is something to be said for doing it the way that I did.

As you can see, I used a Tamura SB2812-1210 60-Hz power transformer along with a full-wave rectifier/capacitor. I used a low-dropout National Semiconductor LM1086CT-5.0 three-terminal regulator (which I had on hand), but the inexpensive MC33275 would have been a better choice. Using the latter regulator would have resulted in a total power supply cost of about $4.50, which is much less than any switching power supply I could have built. When the power bar is supplying power to its connected devices, relay K1 is energized and the unit draws about 1 W. But the unit draws only 0.5 W most of the time. This is much less than the total standby power drawn by all the home entertainment devices that I have plugged into it, giving the overall power saving.

The only other component is an NPN transistor to boost the ATtiny's port drive current up to the 100 mA needed to run the relay. The relay I used has 5-A contacts that are sufficient to handle the connected devices, but it might be too light when running a large plasma TV and using a home stereo receiver at high volumes.

Brian Millier (brian.millier@dal.ca) is an instrumentation engineer in the Chemistry department at Dalhousie University in Halifax, Canada. He also runs Computer Interface Consultants.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/230.

SOURCES

ATtiny45 Microcontroller
Atmel Corp. | www.atmel.com

LM1086CT Regulator
National Semiconductor Corp. | www.national.com

SB2812 Power transformer
Tamura Corp. | www.tamuracorp.com

by George Martin

Embedded Breakup

Divide a Design and Minimize Processing

Once you have your embedded processor up and running, what's next? Your design's success will have a lot to do with how you answer that question. George puts you on the right path with tips about partitioning your design work and minimizing processing requirements.

In my July 2009 article titled "C Start-Up," I presented how an embedded processor running C starts up (*Circuit Cellar* 228, 2009). This time I'll cover what to do after it is up and running.

EMBEDDED IN REAL TIME

If we are writing a payroll program or an inventory program to run on a PC, we can read the input data and commands, process the data, and produce the outputs. The program's success will be based on correctness, memory and disk usage, and ease of use. Correctness will always be first, ease of use will probably be second, and run time will be third. We could perform the calculations in almost any order as long as the correct results were produced. We could speed up run time if more memory and disk space are available.

Now let's consider an embedded program that has an additional requirement of real-time operation. To help you understand this presentation, let's keep it simple and define real-time operation in terms of the time it takes for the system to respond to an input. This must be fast enough to meet certain requirements. Different embedded systems have different real-time requirements. A controller for a machine tool needs to move the tool in a path that meets certain accuracy requirements. Because the tool is moving in many axes at the same time, small motion steps must be coordinated to maintain positional accuracy. It would do no good to have the x-axis calculation completed twice as fast as the y-axis calculations. A more basic example: when you press a button, a light should come on. And any noticeable delay between a button press and illumination is unacceptable. Let's call this the "operator response time," and let's give a number of no greater that 100 ms. You might argue that 100 ms is noticeable, but let's use 100 ms for this discussion.

PARTITIONING WORK

This real-time requirement is what makes an embedded system different from a payroll system. We need to manage all the system's resources in real time. Said differently, we just can't give a large amount of processing to one component of the system and wait around until it's completed. We will lose our "real-timeliness." How do we partition our work? Let's look at several examples.

In keeping with the button input operation, we need to debounce all the push button inputs. And that's typically done by reading the inputs periodically. If enough of the readings indicate a button is pressed, the program will also consider it pressed. This is typically

Listing 1—Add this to the start of the DebugProcessCmd() routine.

```
INT16 DebugProcessCmd(void) {

   if (InLineCnt == OutLineCnt) {   // And new line to process?
       return(0);                   // No. Nothing to process
   }
   // Yes we have at least one line of data to process.

}


--------------------------
Add this to the start of the DebugProcessCmd() routine
--------------------------
```

accomplished using the microprocessor's timer interrupts. Interrupt routines can be set up so that the processor stops what it's doing each time an interrupt occurs and then enters the interrupt routine. When that routine is completed, the processor continues where it left off. Check the online documentation and review the manuals for the device you're using.

TIMERS

Let's set up a timer interrupt so that it interrupts every 10 ms. In the interrupt routine associated with the timer, let's have a counter for each push button. Then, if the button is pressed, we'll increment the counter. If the button is not pressed, we'll decrement the counter. We're doing just the minimum inside the interrupt routine to keep it running as fast as possible.

Outside the interrupt routine, in our main operating loop, we'll look at the counters and see if they've increased beyond a limit. That limit number multiplied by the 10-ms interrupt rate represents our debounce time. Depending on the types of switches you use, you can change the debounce time to suit your application. If the counter has passed the limit, we need to reset its value to that limit so that it does not continue to increase and then set the switch's state as closed. If the counter is decremented below zero, we need to reset the counter to zero and set the switch's state as opened. Outside the interrupt routine, we will manipulate these counters and set the switch's state (open/closed). After that, we can then take the action that the switch status would require.

Look at what we just did. We are managing our system's CPU (time) resources. In this example, we're managing interrupt-processing time and program loop time. We divide the problem into the timing component and the processing component. The timing component (the interrupt routine) is used to perform the debouncing. The processing component (the switch counter management, switch state control, and actions) is accomplished in code running in the main loop. We divided the components of the solution to the problem and put them in areas of code that work to meet the real-time requirements.

You need to divide each of your system's requirements in this manner. Some requirements might have only an interrupt portion, while others might have only a main loop portion. Most will have a portion in each. As you'll see, that can get rather involved.

SERIAL

Let's consider a serial interface. Look at the files for my July 2007 article, "From 'Hello World' to Big Iron," (*Circuit Cellar* 204). In that project, I included fairly simple serial interface code. Let's explore that code once again and make it more resource-friendly (ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2007/204/).

Most modern embedded CPUs have several serial interfaces. Let's look at just a simple RS-232 serial interface as an example. Say that the host computer is a PC that's sending commands. If those commands are received correctly, they are to be acted upon and a response is sent back. Let's look at how I would divide up this problem.

As serial characters are received, an interrupt is generated. This is as straightforward as the timer interrupts. The serial interrupt code reads the new character and checks for errors. If reception errors aren't detected, the character is stored in a buffer. If possible, I try to design the protocol so that there is a unique start character and end character for each command. In the code as it was presented, we looked for commands in a brute-force manner. Let's think about how to make this more resource-friendly so we save CPU time.

Received characters are saved in a buffer. When an end character is received, a new variable named `UINT16 InLineCnt` is incremented. I've found that when an error is detected, a simple-as-possible approach is best. Just set the received character to 0xFF and continue. When the characters are processed, the 0xFF can be detected and the entire line will be thrown out. Some systems have an ACK/NAK response that alerts the host that all is well or a problem is detected.

In the main operating loop, we keep looking to process a new line of data. The call to `DebugProcessCmd()` returns the status. If the status is a one, a new line is received. There are several techniques for saving CPU time while looking for this new line. One method is to keep two variables, `UINT16 InLineCnt` and `UINT16 OutLineCnt`. The `InLineCnt` keeps count of lines received while the `OutLineCnt` variable keeps track of lines processed. If each line ends in, say, a carriage return character, then in the interrupt routine, the `InLineCnt` is incremented when a <CR> is received.

The `DebugProcessCommand()` routine checks the two counters for lines received as (see Listing 1). When they differ, it means there is a new line of data received. After that line is processed, the OutLineCnt is incremented. If we try to do this with one variable, there's a chance that the interrupt and the processing routine can get out of synch and corrupt the variable. By using separate variables and each part of the code changing only its variable, we avoid any possible data corruption. Also, as we're processing a line of data, another line of data can be received without confusion. The actual processing of the received line of code depends on your system. I've worked on systems with over 1,000 unique commands. The system I'm currently working on has only one command line.

Look at the time the CPU has to expend with and without this new concept of counting lines. Without the counters, the code must test what's received to determine a line is received. With just a simple compare added to each serial interrupt, these counters can eliminate all that testing and overhead.

EXTRA CREDIT

Now, for extra credit. When it's determined that we have a new line of data, we can process it immediately or flag that we have it and process it at a later time.

I generally process it immediately, but let's say that you are running a time-critical operation. Think about how you would set up to process the line at a less critical time.

Also, some commands are short and sweet and don't have a lot of overhead. But you could receive a command that says list all the key parameters out the serial port. And if the list is long, you can take several seconds to assemble the message and send it out the serial port. How would we break up this task and be less of a CPU hog? Hint: read on about state machines.

STATE MACHINES

Let's investigate a simpler resource-management problem. Assume we need to keep a running checksum on all the variables in memory. This checksum will be used to determine if something has changed and the variables need to be written into a more permanent type of memory. The process of calculating a checksum (or a CRC) takes too long just to do it all at once.

I'd first break up the checksum calculation into smaller steps. Perhaps calculating the checksum of the next 16 variable locations will be fast enough. In Listing 2, I broke up that calculating of the checksum on all the variables into calculating the checksum on 16 of the variables at a time. If 16 is not the right size, it can be changed. In addition, your variables won't be all the same size. So, you might use a pointer to an 8- or 16-bit object and increment that pointer as you go through the loop.

The point here is a technique for breaking a big operation into smaller operations and then performing these smaller operations repeatedly

Listing 2—This is a state machine that breaks up a lengthy task.

```
INT16   CalcState;  // State variable
#define CCS_INIT      1
#define CCS_RUN       2
#define CCS_DONE      3
INT16 i;

void CalcChkSum(void) {
INT16 j;

switch CalcState {
       case default: {
            CalcState = CCS_INIT;
       } break;

       case CCS_INIT: {
            i = 0;   // start with first variable
            ChkSum = 0;  // Init the CheckSum
            CalcState = CCS_RUN;
       } break;

       case CCS_RUN: {
            for (j = 0; j < 16; j++) {
               CheckSum = CheckSum + var[i+j];
               if (i+j)  > LastVariable) {
                 CalcState = CCS_DONE;
                 Break;
            }
            i = i + 16;
       } break;

       case CCS_DONE: {
            // Do whatever you need to here
            CalcState = CCS_INIT;    // start this all over again
       } break;
}  // end of switch CalcState {

}  // end of  void CalcChkSum(void)
```

Listing 3—This simple code sets and clears test point 1.

```
void SetTP1(void) {
       P4 = P4 | 0x01;     // Set bit 0 to a 1
}
void ClrTP1(void) {
       P4 = P4 & ~(0x01);  // Set bit 0 to a 0
}
```

Look at the time the CPU has to expend with and without this new concept of counting lines. Without the counters, the code must test what's received to determine a line is received. With just a simple compare added to each serial interrupt, these counters can eliminate all that testing and overhead.

until the work is completed. Do you have a task that is too big to do all at once? Here's a way to chop it up. If we have a real-time operating system (RTOS), it would hand out CPU time to each running task. Using an RTOS is another method to divide up tasks, but it's a lot more complicated and expensive.

Another key point is that we use a state machine in a manner that you don't usually see discussed. It's perfectly fine to use state machines in this manner.

INTERRUPTS

Interrupts are indeed a very powerful concept in the embedded world. It's tempting to overuse them. Let me give you an example.

Suppose we have an interrupt that comes in with the line frequency. That's 60 times per second for the United States and 50 times per second for European countries. It's tempting to do more work in the interrupt routine than should be done. For example, if the display needed to be updated, I've seen code that does this inside the line-frequency interrupt routine. The problem comes with the code that is changing the values of the displayed digits or characters. You now have two routines using the same variables and that's a recipe for a big problem.

A simple example: If you have test points that the software can manipulate and you do this both inside and outside an interrupt routine, problems usually occur. Say you have four test points and you have routines to Set, Clear, and Toggle each of the test points. Also, let's say that all four test points are on the same processor port (Port 4, just to give it a name). The code to set and clear test point 1 would look like Listing 3.

Now think about the assembly language that is generated for these lines of C code. The content of Port 4 will be loaded into a register, the contents of the register will be changed with the AND or OR operation. The result will then be saved in Port 4. This one C language statement probably translates into three assembly language statements. And test points 2, 3, and 4 will have much the same code.

What if we want to work with test point 1 inside the interrupt? What if we want to work with test point 2 outside the interrupt? Sometimes we'll be running the assembly code outside the interrupt that works with test point 2. We will load a copy of Port 4 into a register and then receive an interrupt. That interrupt routine will change Port 4's value.

When we return from the interrupt, the data in our register no longer matches what is in Port 4 since the interrupt routine just changed the contents of Port 4. These two routines are competing for one resource (Port 4) and will overwrite each others' work.

It's best to never share resources between interrupt and non-interrupt routines. Set flags, markers, or use other techniques to keep from corrupting each others' data. I've seen too much work done inside the interrupt routine, and this leads to difficult problems to find and fix.

SPREAD THE WORK

So, we just covered some basic concepts of how to break up the work in an embedded design and minimize the processing requirements in order to meet the real-time requirements. This is just the tip of the iceberg, and the discussion continues right into the topic of real-time operating systems.

Let me know if there is any aspect of C programming for embedded systems that I could cover in a future column.

George Martin (gmm50@att.net) began his career in the aerospace industry in 1969. After five years at a real job, he set out on his own and co-founded a design and manufacturing firm (www.embedded-designer.com). His designs typically include servo-motion control, graphical input and output, data acquisition, and remote control systems. George is a charter member of the Ciarcia Design Works Team. He is currently working on a mobile communications system that announces highway info. He is also a nationally ranked revolver shooter.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/230.

RESOURCE

Microprocessor Interrupt Routines, Wikibooks, http://en.wikibooks.org/wiki/Microprocessor_Design/Interrupts.

by Jeff Bachiochi

Smart Circuitry for Battery Monitoring

Well-designed internal protection circuitry is needed in Li-Ion battery cells to prevent dangerously high temperatures and failure. In this article, Jeff explains how such smart circuitry works. He covers how to check a cell's state of charge, find problems, and fix them.

Help! My laptop is on fire! I recall a comedy sketch I saw years ago on *Saturday Night Live* called "Happy Fun Ball." It was an advertisement for a child's toy that had every conceivable negative side effect from warts to hair loss. The funny (or sad) thing is that like many of the pharmaceutical advertisements we see today, the side effects often seem to outweigh the relief they claim to provide. So what gives with a battery technology responsible for arson?

Come on, get real. We're asking for lighter, longer-lasting (higher-power) batteries capable of running our portables longer on a charge. We got it. Along with this super power comes the possibility of using it for evil, even if by accident. Some early laptops had a design flaw that caused a short circuit to the lithium-ion battery pack. Without internal protection circuitry to prevent high temperatures from occurring, Li-Ion cells can fail. Thermal runaway is responsible for most high-visibility failures (as you can see in some YouTube videos). All Li-Ion battery packs now contain smart circuitry to open up the current path should temperatures exceed predetermined limits.

Thermal runaway occurs once a cell exceeds the unstable temperature of approximately 150°F (cobalt chemistries).[1] How the temperature ever reaches this level is the key. This can come from the lack of internal protection circuitry, the failure of such circuitry, or manufacturing defects. Should internal protection circuitry fail undetected, requesting high current levels will generate a rise in temperature that could lead to thermal runaway. Although battery manufacturers strive to minimize the presence of metallic particles in their manufacturing process, complex assembly techniques make the elimination of all metallic dust nearly impossible and can lead to internal shorts that (if sufficient) can elevate temperatures again and lead to thermal runaway. When thermal runaway occurs in a cell, other cells in the battery pack (those in close proximity) will be affected by the localized heat source.

A Li-Ion battery fire is considered a Class D (combustible metal) fire and must be dealt with accordingly. A fire extinguisher with a Class D rating will use dry powder agents and work by smothering and heat absorption. (Refer to the figure on the *Circuit Cellar* FTP site.)

BATTERY CHEMISTRY

Li-Ion batteries shouldn't be confused with lithium batteries, which are disposable (primary) batteries that use lithium metal or lithium compounds. Not only are lithium-ion batteries rechargeable, they also have one of the best energy-to-weight ratios, no memory effect, and a slow loss of charge when not in use.

The lithium-ion battery has a three-layer,

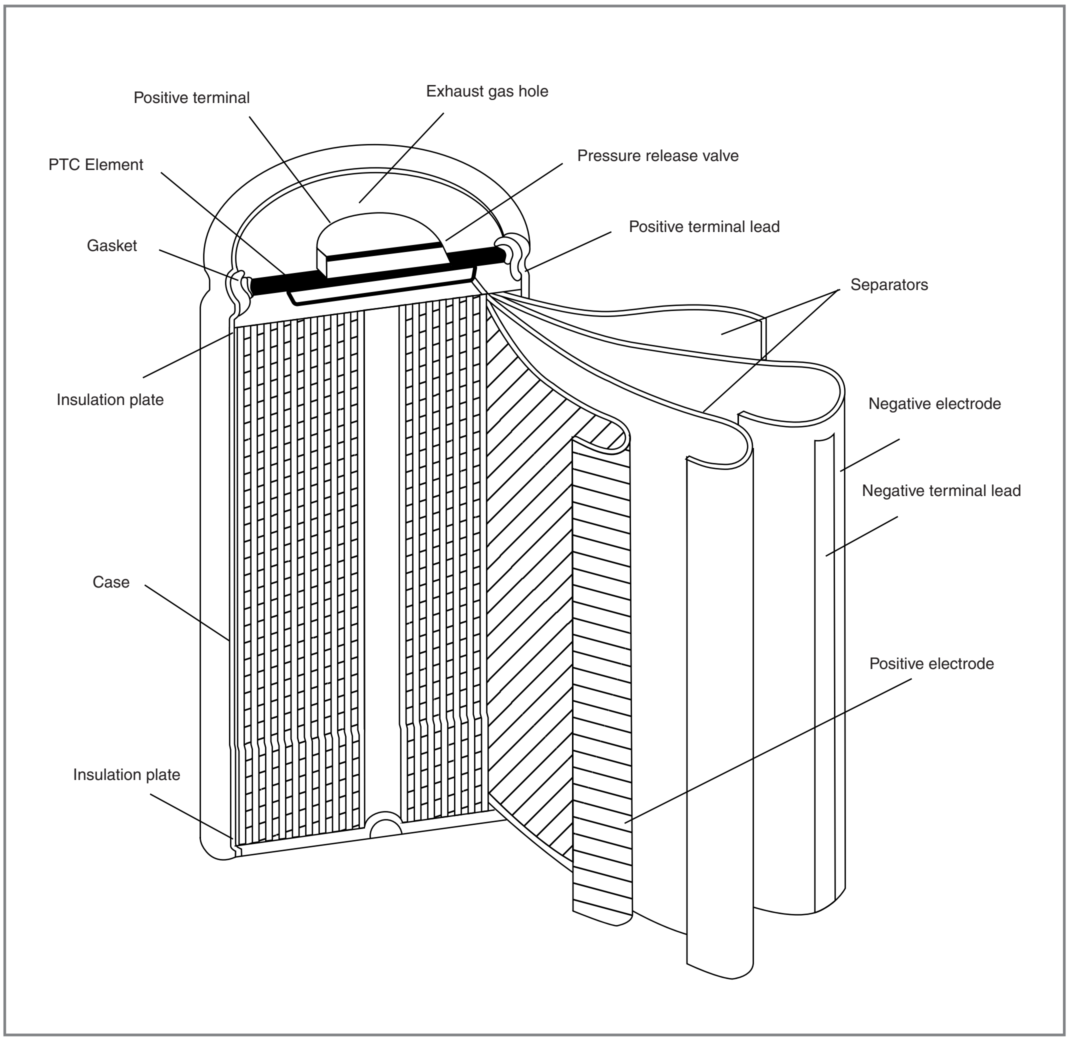


Figure 1—A lithium-ion cell is built much like a capacitor. Electrodes are separated by an insulator that allows lithium ions to pass through. While the most common shape is cylindrical, rectangular shapes pack a higher density into the allotted space.

coiled structure within its case much like a capacitor (see Figure 1). These three layers typically comprise a positive electrode plate of lithium cobalt oxide (as its chief active ingredient), a negative electrode plate of a specialty carbon (as its chief active ingredient), and a separator (insulating) layer. The battery is equipped with a variety of measures to ensure safety, along with a pressure release valve that releases gas if the internal pressure exceeds a specific value (high internal temperature), thereby preventing the battery from rupturing.

A cathode of lithium cobalt oxide provides a crystalline structure that easily gives up lithium ions during a charging cycle. Lithium ions pass easily through the separator, which contains a nonaqueous solution of a lithium salt to prevent electrolysis of the electrolyte. The carbon material used in the anode also has a crystalline structure that can accept lithium ions during a charging cycle. A charging potential that moves ions to the anode creates an abundance of electrons there (negatively charged) compared to the cathode that has given up the ions (positively charged).

Lithium-ion technology continues to improve by altering the chemical makeup of the electrodes. Through continual research, you can expect an increase in the cycle life, the capacity, and the safety of lithium-ion-style batteries. With improvements, we might find a change in cell potential and charging requirements. How can the public be expected to keep track of all this?

For the most part, manufacturers are avoiding problems by preventing Li-Ion cells from being purchased unless they are packaged in modules that contain smart circuitry to monitor their activity. This allows charging and recharging to be handled in a safe and efficient way.

SAFETY ISSUES

There are safety issues associated specifically with lithium ion. Many are obvious. Extremely low temperature will slow chemical reactions impeding nominal output. Elevated temperatures run the risk of damaging the electrolyte and producing increased internal gas pressure. Ambient temperatures during discharge should be –20° to 60°C. Each cell is vented to avoid uncontrolled bursting, so module packs must not be sealed units in order to allow for controlled release if damaged. While the largest cause of damage (both physical and operational) to a cell is heat, physical damage can be catastrophic and must be reduced by properly protecting each cell in the module's design.

We want everything: high current on demand, high energy density by mass and weight, quick recharging, and a long operational lifespan. As the capacity of batteries increases, it is important to keep in mind its potential to release this energy. Gasoline is an extremely hazardous energy source, yet we respect it and use it carefully. Figure 2 shows the energy density for selected items.[2] Items toward the right hold higher energy densities per kilogram (kg, weight). Items toward the top hold higher energy density per liter (volume).

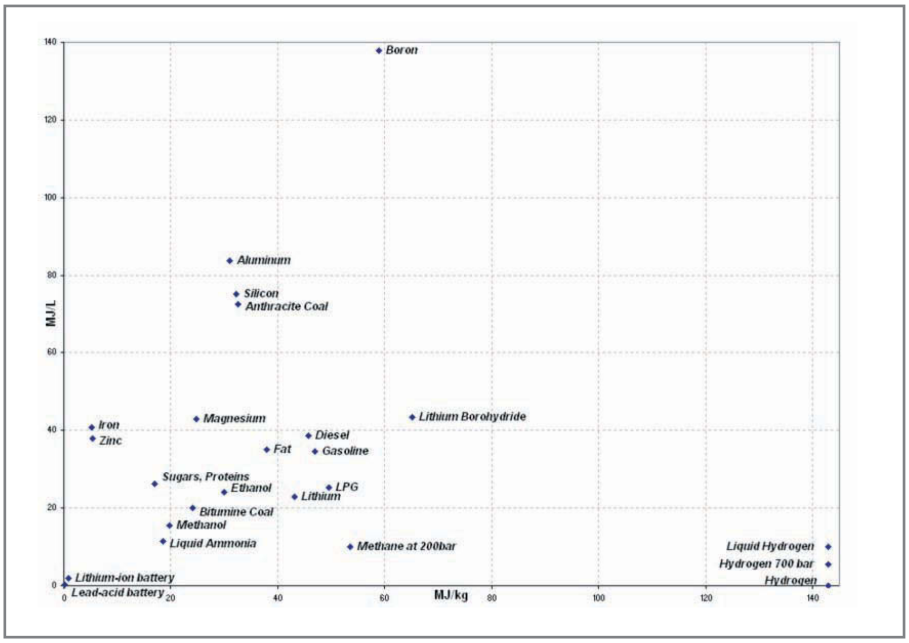


Figure 2—This graph shows how various elements compare in energy density based on weight (horizontal) versus volume (vertical). The ultimate material would be both light in weight and small in volume.

While Li-Ion batteries aren't challenging gasoline at this point by weight or volume, and today's hybrids prefer nickel metal hydride, most future plug-ins will use Li-Ion.

At approximately 3.6 V, the Lithium-Ion battery has a typical output voltage of more than two times that of alkaline batteries. Note that this does not easily substitute well in today's electronic equipment the way NiCd batteries have. To prevent a user from misusing Li-Ion batteries, either by discharging the cells too far or too fast, manufacturers require individual cells to be combined into modules that include some safety circuitry.

SMART CIRCUITRY

Safety is a priority with Lithium-Ion manufacturers. Each module contains circuitry to monitor the electrical charge of individual cells as well as their temperatures. Figure 3 shows a typical multiple-cell module. In some cases, the battery pack makes internal data, like temperature, available to external circuitry. But for the most part, the internal smart circuitry handles everything. Excess current draw (possibly from a short circuit) can be handled by opening up the current path via an internal FET. Note that two FETs in series are used for added protection.

Although it isn't a safety issue, the smart circuit also can prevent the module from being sucked dry. By preventing the output voltage from dropping below a minimum voltage (approximately 3 V per cell), the module remains within its recommended charge-discharge parameters providing optimum life. The smart circuitry will also stay alive with a minimum of current draw to keep protecting the module. On the charging side of the operation, the circuit will prevent damage to the module from overcharging. Once the module has reached a maximum charge voltage of approximately 4.3 V per cell, the smart circuit can again open the circuit.

A thermistor is often used to keep the module within temperature specs. This is a safety issue that is monitored closely during charging and discharging to prevent any chance of cell bursting. Over temperature may be due to the environment (ambient temperatures), charging-discharging (internal temperatures), or combination of both.

LONG LIVE LI-ION

The possibility of a cell mismatch within a module increases as the number of cells and load currents increase. Cell balancing can be employed to counteract cell mismatches. There are two kinds of mismatch in a module: state-of-charge (SOC) and capacity/energy (C/E) mismatch. Although the SOC mismatch is more common, both contribute to limit the capacity (milliamp-hours) of a module to the capacity of the weakest cell.

It is important to recognize that the cell mismatch results more from limitations in process control and inspection than from variations

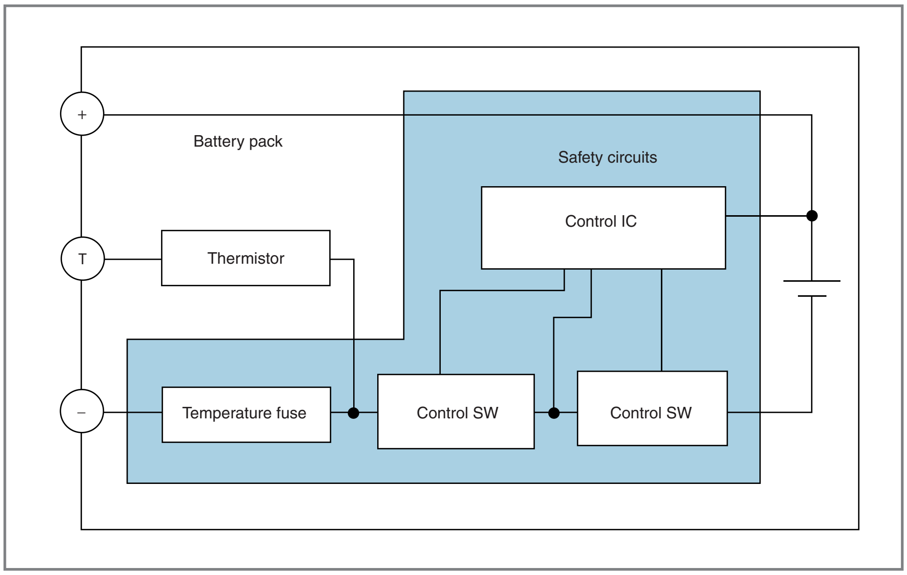


Figure 3—Today's Li-Ion battery packs have special safety circuitry that prevents high current and temperature runaway.

inherent in the Li-Ion chemistry. The use of cell balancing can improve the performance of series-connected Li-Ion cells by addressing both SOC and C/E issues. A SOC mismatch can be remedied by balancing the cell during an initial conditioning period and subsequently during only the charge phase. C/E mismatch remedies are more difficult to implement, harder to measure, and require balancing during both charge and discharge periods.

Cell balancing is defined as the application of differential currents to individual cells (or combinations of cells) in a series string. Normally, of course, cells in a series string receive identical currents. A battery pack requires additional components and circuitry to achieve cell balancing. However, the use of a fully integrated analog front end for cell balancing reduces the required external components to just balancing resistors. This type of solution eliminates the need for discrete capacitors, diodes, and most other resistors to achieve balance.

Battery pack cells are balanced when all the cells in the battery pack meet two conditions. One, if all cells have the same capacity, they are balanced when they have the same SOC. In this case, the open circuit voltage (OCV) is a good measure of the SOC. If, in an out-of-balance pack, all cells can be differentially charged to full capacity (balanced), they will subsequently cycle normally without any additional adjustments. This is mostly a one-shot fix. Two, if the cells have different capacities, they are also considered balanced when the SOC is the same. But, since SOC is a relative measure, the absolute amount of capacity for each cell is different. To keep the cells with different capacities at the same SOC, cell balancing must provide differential amounts of current to cells in the series string during both charge and discharge on every cycle.

As you can imagine, proper balancing depends on one's ability to monitor the state of each individual cell. Linear Technology is one of the many companies with a complete line of charging and monitoring devices for a number of different battery chemistries. A new family of multicell, high-voltage, battery-stack monitors includes 12-bit ADCs, precision voltage references, high-voltage input multiplexers, and a serial interface. Because the applications include electric and hybrid vehicles that can typically have battery stacks over 100 V, these devices can be stacked to monitor every cell.

LTC6802

It's no wonder hybrid and PEV manufacturers are using caution with Li-Ion technology. The high percentage of battery-to-total-vehicle cost makes these next-generation battery decisions make or break. So you won't be surprised in knowing that auto manufacturers are watching how battery manufacturer's products are operating in non-automotive applications. Some form of Li-Ion batteries is presently running in laptops, cell phones, portable DVD players, and power tools. Most of today's products require charging and discharging currents approximately 1 to 2 A. You

It takes more than a top program to win the

Product of the Year Award*

five times running

- For instance, reasonable prices without hidden costs.
- A fair update policy without mandatory maintenance fees or costly add-on contracts.
- A competent hotline available free of charge for every customer.
- A program development and optimization process based on an open discussion with the users.

EAGLE
Version 5
Schematic - Layout - Autorouter
for Windows® Linux® Mac®

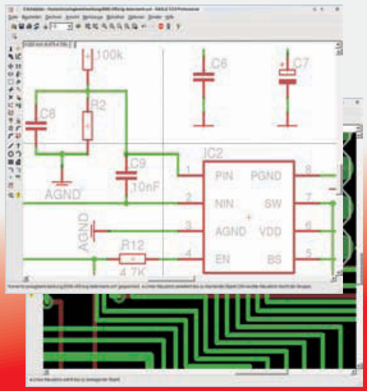
* Annual award of the leading German electronics magazine. The winners of eleven categories are determined by the reader's votes.

In other words: EAGLE is one of the world's highest rated schematic capture and PCB-board layout packages because for the last twenty years we have been treating our customers the way we would like to be treated ourselves.

An example:
In the course of product development we implemented a

Follow-me Router

which saves an enormous amount of time during the manual routing process. This new function is available free of charge to every customer as are all new features within a main EAGLE release. In this particular case you need to have the autorouter module of version 5.



Pick the level that is right for you —
pay only the difference for upgrades!

	Light	Standard	Professional
Max. number of schematic sheets	1	99	999
Max. board size	4x3.2 inch	6.4x4 inch	64x64 inch
Max. # of signal layers	2	4	16
Layout or Schematic Editor		$249	$498
Layout and Schematic Editor		$498	$996
Layout Editor and Autorouter		$498	$996
Layout Editor and Schematic Editor and Autorouter	$49	$747	$1494

Standard and Light Editions have full functionality except for the limitations mentioned in the table.

You can use EAGLE Light for evaluation and non-commercial applications without charge. Download it from our web site.

www.cadsoftusa.com
800-858-8355

CadSoft Computer, Inc., 19620 Pines Blvd., Suite 217, Pembroke Pines, FL 33029
Hotline (954) 237 0932, Fax (954) 237 0968, E-Mail: info@cadsoftusa.com

Windows / Linux / Mac are registered trademarks of Microsoft Corp. / Linus Torvalds / Apple Computer, Inc.

Add USB to Your Designs

Chips, code, protocols, embedded hosts, wireless options, debugging, **USB 3.0** and **SuperSpeed** too!

USB Complete
The Developer's Guide
Fourth Edition
Jan Axelson

ISBN 978-1-931448-08-6 $54.95
Lakeview Research LLC **www.Lvr.com**
By the author of *Serial Port Complete*

PIC-SERVO
MOTION CONTROL

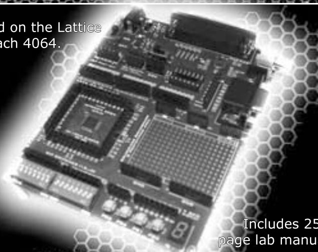
MOTION CONTROLLERS FOR BRUSH, BRUSHLESS AND STEPPER MOTORS.

- controller chips
- controller boards

www.picservo.com
JEFFREY KERR, LLC

MACH64
PROGRAMMABLE LOGIC STARTER KIT
NURVE NETWORKS LLC

Based on the Lattice ispMach 4064.



Includes 250 page lab manual.

Learn CPLDs the fun way with the MACH64! This complete kit comes with everything you need to take you from mystery to mastery with CPLDs and programmable logic.

Learn to turn *software* into *hardware!*

WWW.XGAMESTATION.COM

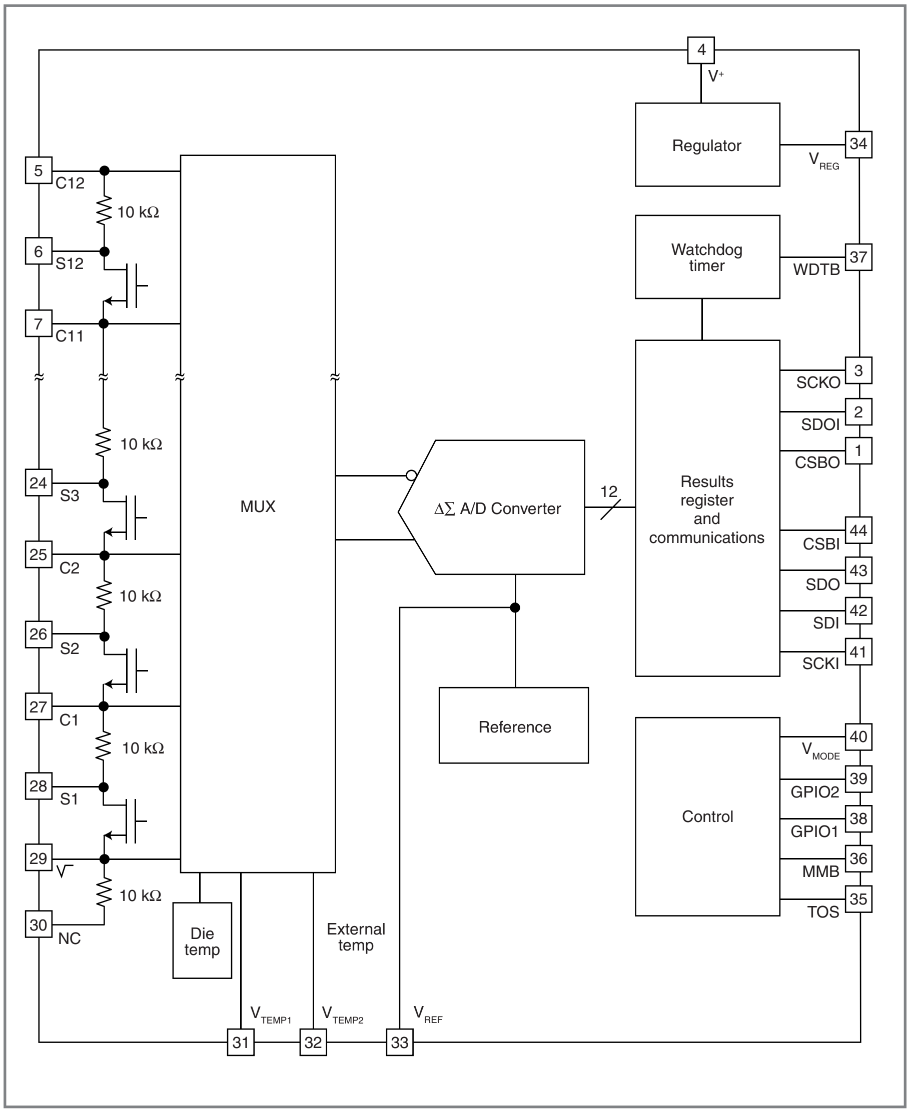


Figure 4—The Linear Technology LTC6802 contains all of the components to monitor and bypass current of up to 12 cells in series. Multiple devices can be daisy-chained to handle battery modules with longer chains of cells.

can expect vehicle currents to exceed this by a factor of 100, so the Li-Ion battery management system (BMS) will become even more important.

Good BMS designs will be a balance of performance, economics, and safety. Their requirements will include wide temperature fluctuations and electrically noisy environments. High common-mode voltages present tough challenges to analog electronics. Each cell needs to be measured differentially for an accurate SOC. The Linear Technology LTC6802 can measure and monitor a string of 12 Li-Ion cells (see Figure 4). A 13-to-2 multiplexer can connect any cell to a 12-bit delta-sigma ADC. Five groups of registers hold 6 bytes of RD/WR configuration data, 18 bytes of RD-only conversion data, 3 bytes of RD-only flag data, 5 bytes of RD-only temperature data, and a byte of RD-only packet error codes.

It is assumed that all cells in a serial chain will have the same capacity. In a perfect world, each of these would charge and discharge together, remaining in an identical SOC. However, slight imbalances due to manufacturing variables, aging, or differing SOCs when first assembled into modules can grow with each charge/discharge cycle. The module's potential is the sum of each OCV. The cells with less capacity will become fully charged first. They will then be overcharged while

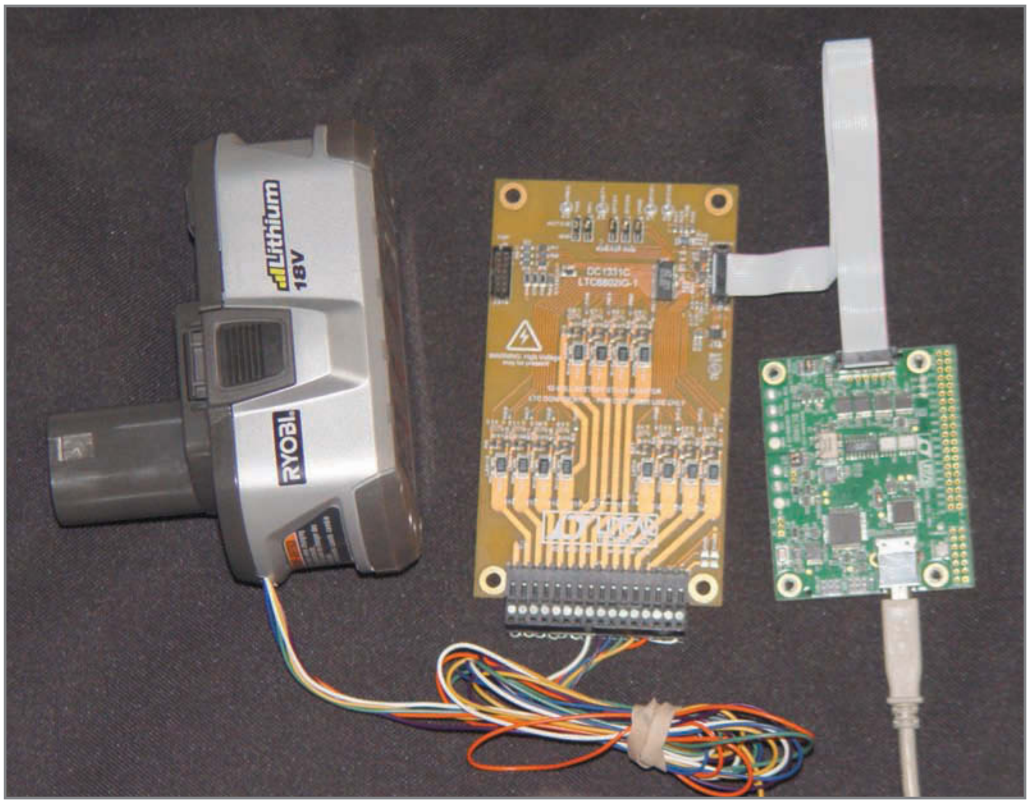


Photo 1—I monitored the five cells in this Ryobi 18-V Li-Ion battery pack with Linear's evaluation board for the LTC6802-1. This arrangement allowed me to balance the cells in this module and improve the poor performance I was getting from it.

the higher-capacity cells are still trying to get full. In fact, these may never get to a full charge because the module's SOC (sum of all the cells OCVs) has reached the maximum limit. Not only are cells within the module being damaged, but the total capacity drops faster than normal because it can only be as good as the least cell. In a serially connected stack, all cells receive the same charging current, so those cells with an imbalance will not be able to recover without outside assistance. This problem increases as the number of cells in a module's string increases.

Today, creating a balance in the cells of a module is most often handled during charging. This is

Got Serial, Need Network?

Volume Discounts Available

gridconnect™

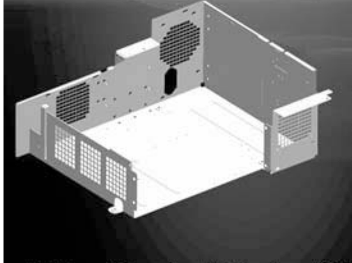
www.gridconnect.com
+1 800 975-4743

2B2C Factory on Your Fingertip

Factory on Your Fingertip

From Idea to Reality
-> Design
-> Prototyping
-> Production

See Our Differences in
-> Quality
-> Service
-> Price



*Milling *Turning *Grinding *CNC
*Wire Cutting *Laser *Plasma
*Water Jet *Plastic Injection
*Sheet Metal *Gear *SLA
*FDM *SLS *LOM

CLICK TO MAKE

MachinePIER

Tel: 408-421-9840 Email: sales@machinepier.com
Website: http://www.machinepier.com

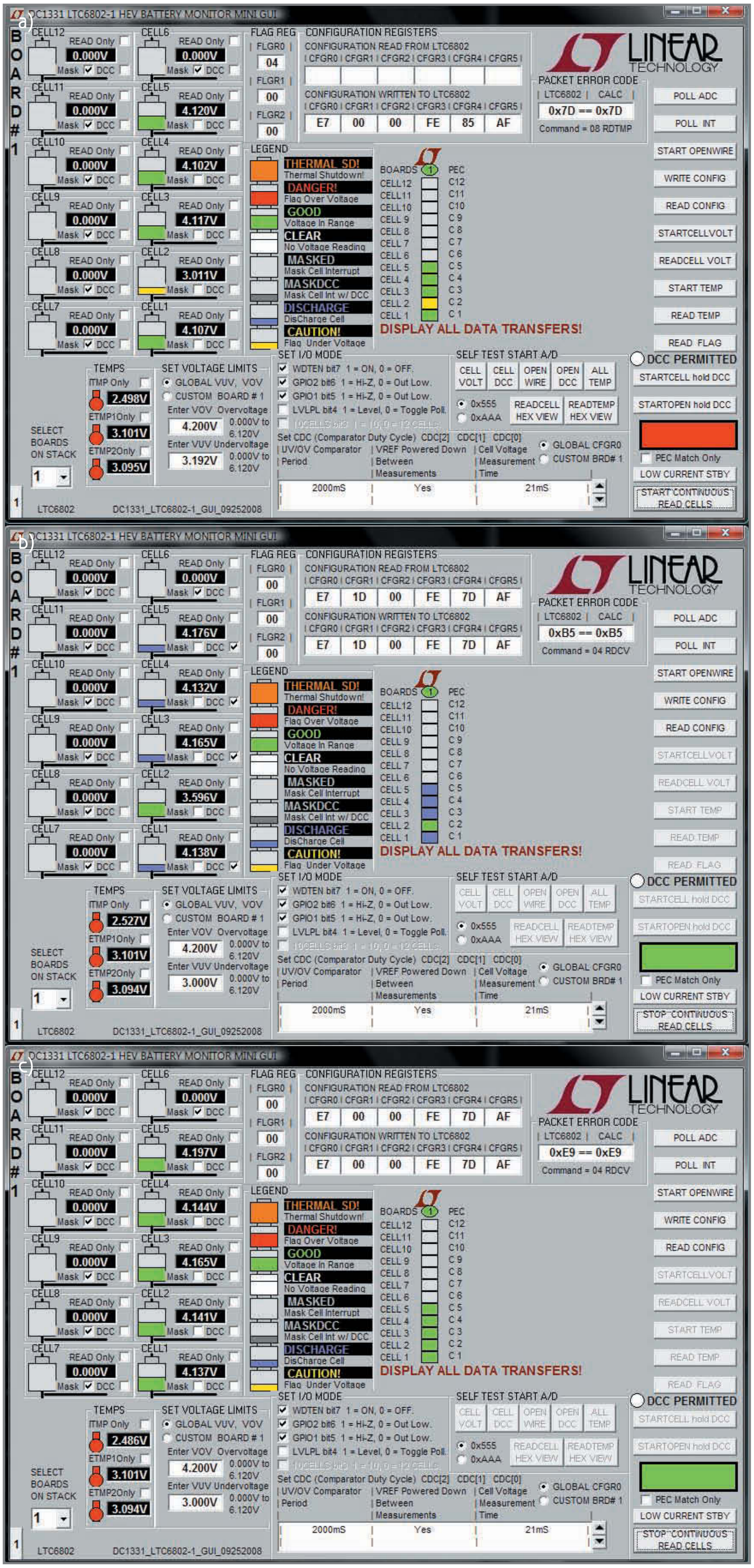


Photo 2a: CELL12 0.000V, CELL11 0.000V, CELL10 0.000V, CELL9 0.000V, CELL8 0.000V, CELL7 0.000V, CELL6 0.000V, CELL5 4.120V, CELL4 4.102V, CELL3 4.117V, CELL2 3.011V, CELL1 4.107V. FLAG REG: FLGR0 04, FLGR1 00, FLGR2 00. CONFIGURATION WRITTEN TO LTC6802: E7 00 00 FE 85 AF. PACKET ERROR CODE 0x7D == 0x7D, Command = 08 RDTMP. TEMPS: ITMP 2.498V, ETMP1 3.101V, ETMP2 3.095V. VOV 4.200V, VUV 3.192V. 2000mS, Yes, 21mS. START CONTINUOUS READ CELLS.

Photo 2b: CELL12 0.000V, CELL11 0.000V, CELL10 0.000V, CELL9 0.000V, CELL8 0.000V, CELL7 0.000V, CELL6 0.000V, CELL5 4.176V, CELL4 4.132V, CELL3 4.165V, CELL2 3.596V, CELL1 4.138V. FLAG REG: FLGR0 00, FLGR1 00, FLGR2 00. CONFIGURATION READ FROM LTC6802: E7 1D 00 FE 7D AF. CONFIGURATION WRITTEN TO LTC6802: E7 1D 00 FE 7D AF. PACKET ERROR CODE 0xB5 == 0xB5, Command = 04 RDCV. TEMPS: ITMP 2.527V, ETMP1 3.101V, ETMP2 3.094V. VOV 4.200V, VUV 3.000V. 2000mS, Yes, 21mS. STOP CONTINUOUS READ CELLS.

Photo 2c: CELL12 0.000V, CELL11 0.000V, CELL10 0.000V, CELL9 0.000V, CELL8 0.000V, CELL7 0.000V, CELL6 0.000V, CELL5 4.197V, CELL4 4.144V, CELL3 4.165V, CELL2 4.141V, CELL1 4.137V. FLAG REG: FLGR0 00, FLGR1 00, FLGR2 00. CONFIGURATION READ FROM LTC6802: E7 00 00 FE 7D AF. CONFIGURATION WRITTEN TO LTC6802: E7 00 00 FE 7D AF. PACKET ERROR CODE 0xE9 == 0xE9, Command = 04 RDCV. TEMPS: ITMP 2.486V, ETMP1 3.101V, ETMP2 3.094V. VOV 4.200V, VUV 3.000V. 2000mS, Yes, 21mS. STOP CONTINUOUS READ CELLS.

Photo 2—The application that supports Linear's Evaluation Board enabled me to check each cell's SOC (a), find a problem with cell 2 (b), and temporarily apply loads to discharge other cells to the same OCV. c—This allowed the charger to bring all cells to a full SOC.

accomplished by routing current around those cells that have reached optimum SOC allowing those with a lesser charge to continue gaining charge without exceeding the modules maximum voltage. Referring back to Figure 4, you can see a FET-controlled 10-kΩ resistor load that can be enabled across each cell. This is only approximately 36 µA and not enough to create any significant cell-balancing currents, but it can be used as a driver for an external FET and load. Or you can place a smaller external resistor across the 10 kΩ and use the internal FET. The catch here is that you must pick your components so that you are not creating a condition of dissipating too much heat internally (within the device or the module).

Firmware used to monitor each cell's SOC can determine which cells get bypassed, juggling the load so that the internal FET's dissipation won't raise the LTC6802's temperature above the prescribed limit. Using a higher load reduces the current (and heat) and allows multiple cells to be bypassed at the same time. A module that requires a high degree of cell balancing may take a few charge/discharge cycles for proper balancing to take place.

SPI

The SPI used to monitor the LTC6802-1's registers can be daisy-chained to multiple devices. This has special importance when multiple devices are used to monitor stacks larger than 12 cells. The second device is referenced to the top of the thirteenth cell, so it will have a different supply potential. Ordinarily, this potential creates a problem (no common ground) and the communications between devices must be isolated in some way. The LTC6802-1 uses high-side/low-side ports that allow daisy-chained devices separated by diodes to communicate using current levels rather than voltage levels.

When using stacked devices a command byte is clocked to all devices in parallel. When data needs to be passed, the devices are internally connected as a cascading shift register. Data bytes following a write command are sent to the device at the top of the stack first ending with the data for the device at the bottom of the stack. After a read command, data comes from the device at the bottom of the stack first with the data from the device at the top of the stack last.

Linear Technology has an evaluation board pair that can be used to evaluate the LTC6802-1 right on your PC. Photo 1 shows the demo DC1331C and 590B connected to my Ryobi 18-V lithium battery pack. I picked up a new portable drill expressly to investigate the lithium battery packs. The first thing I did was void the warranty by opening the high-capacity pack and tacking on some wires to the each cell so I could monitor each externally. What I found surprised the heck out of me.

BATTERY MONITOR MINI GUI

This battery pack internally monitors the SOC and turns off the current when the SOC falls below the minimum limit. So, I charged up the pack, plugged it into the drill, and ran it to the point of battery pack shutdown. Photo 2a shows what I found when I looked at the Ryobi battery pack using the Linear Technology battery monitor application. This application gives you complete control over the LTC6802-1. All but cell 2 had equal SOCs. Cell 2 was far below the others by more than 1 V! This meant the other cells had to be overcharged to compensate for cell 2's lower voltage.

I expected to find some kind of cell balancing when the pack was recharged. To my dismay, I find that this was not the case. Photo 2b shows the new charged state results. While the voltage of cell 2 has come up a higher percentage than the other cells, you would expect this since this cell is not operating in the horizontal region of a cell's discharge curve (4 to 3.5 V), but in the vertical region (less than 3.5 V). This pack needs to be cell balanced and gave me a good excuse to check out the features of Linear Technology's application a little more intimately. As you can see in Photo 2b, I've issued a command to the LTC6802 to apply loads to all the cells except cell 2. The evaluation board used external FETs and 150-Ω resistors as a load for each cell. The cells are discharged slowly. I shut off each load as it approached cell 2's voltage. When all the cells were at the same potential, I plugged the battery pack back into the charger. The charge cycle was able to do its job on all the cells, and the results were far more promising (see Photo 2c). Cell 2 now has a SOC that's comparable to the other four. Cell balancing has returned the battery pack to optimum performance.

TROUBLE IN PARADISE

I'm not getting warm fuzzies about my first experience with the Li-Ion technology in tools. There is nothing on the outside that tells me if everything inside is working correctly. Without digging into this, I would never have known there was a cell balance problem. This might be a fluke, but it has me questioning if we're ready to unleash the power of Li-Ion.

Our future is being built on lithium cell technology. Laptop fires have taught us that we must be cautious in working with any high-energy density material. Most people understand the potential of gasoline and treat it with respect. Smart modules may provide us with a safe way of dealing with the power of lithium, but it looks like we might have a way to go to make them foolproof.

Should a Li-Ion battery be able to withstand being pierced by a nail without exploding? It sounds to me like trying to make gasoline that will not burn if a lit match is accidentally dropped into the tank. ■

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.*

PROJECT FILES

To view an additional figure, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/230.

RESOURCES

[1] I. Buchmann, "Lithium-Ion Safety Concerns," Cadex Electronics, www.batteryuniversity.com/partone-5B.htm.

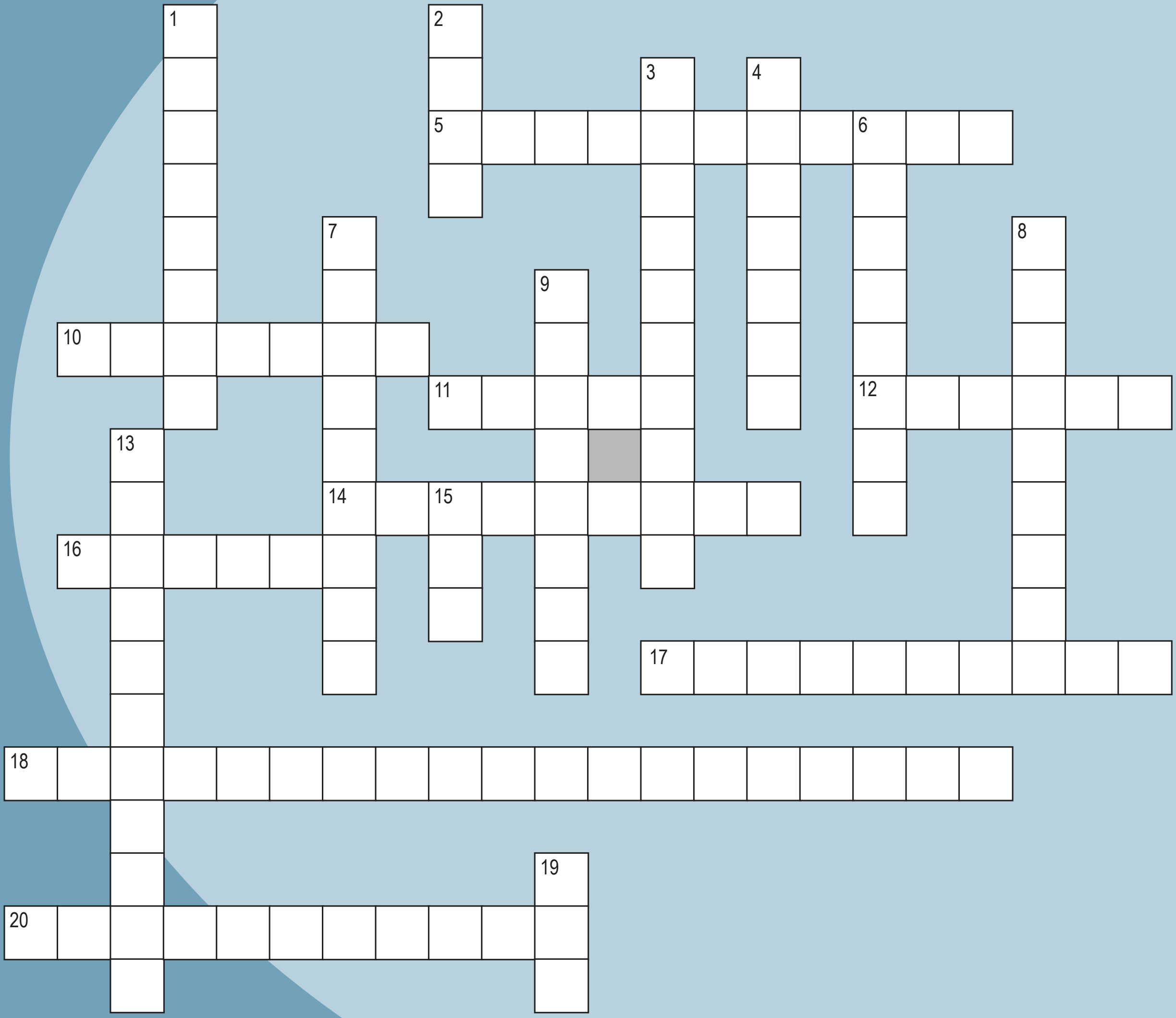
[2] Energy density, Wikipedia, http://en.wikipedia.org/wiki/Energy_density.

SOURCES

LTC6802-1 Battery stack monitor
Linear Technology | www.linear.com

18-V Lithium battery pack
Ryobi Power Tools | www.ryobitools.com

CROSSWORD



Down

1. Sa = b, b = a
2. Write data to a disk
3. Preset access; the opposite of random
4. Middle tonal range
6. IrDA is a standard for data transmission via what light?
7. The home of Station X, the main site of the UK's WWII decryption efforts
8. Charge holder
9. Protects a system from voltage surges
13. 3.463 candelas per square meter
15. Execute
19. No user configuration

Across

5. x = x
10. Oe
11. The "B" in BEDO DRAM
12. // Initialize the LED output subsystem
14. The "Curt" (1902–1988) behind the Curta
16. 10^{100}
17. "High-k" stands for "high ______ constant"
18. RPC (three words); Protocol for requesting services
20. Single-user PC

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 sh eet. **ALL TEXT AND OTHER ELEMENTS MUST FIT WITHIN A 2″ x 3″ FORMAT**. Call for current rate and deadline information. E-mail adcopy@cir cuitcellar.com with your file and digital submission or send it to IDEA BOX, Cir cuit Cellar , 4 Park Street, V ernon, 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 pr oducts and services.

It writes your Full Speed USB Code!

NEW! HIDmaker FS for Full Speed FLASH PIC18F4550

Creates complete PC and Peripheral programs that talk to each other over USB. Ready to compile and run!

- Large data Reports
- 64,000 bytes/sec per Interface
- Easily creates devices with multiple Interfaces, even multiple Identities!
- Automatically does MULTITASKING
- Makes standard or special USB HID devices

NEW! "Developers Guide for USB HID Peripherals" shows you how to make devices for special requirements.

Both PC and Peripheral programs understand your data items (even odd sized ones), and give you convenient variables to handle them.

PIC18F Compilers: PICBASIC Pro, MPASM, C18, Hi-Tech C.

PIC16C Compilers: PICBASIC Pro, MPASM, Hi-Tech C, CCS C.

PC Compilers: Delphi, C++ Builder, Visual Basic 6.

HIDmaker FS Combo: Only $599.95

DOWNLOAD the HIDmaker FS Test Drive today!

Trace SYSTEMS, Inc.

www.TraceSystemsInc.com

301-262-0300

USB

Add USB to your next project—it's easier than you might think!

- USB-FIFO up to 8 mbps
- USB-UART up to 3 mbps
- USB/Microcontroller boards pre-programmed with firmware
- **2.4GHz ZigBee™ & 802.15.4**
- **RFID Reader/Writer**

Absolutely NO driver software development required!

www.dlpdesign.com

DLP Design

ALL ELECTRONICS CORPORATION

Electronic and Electro-mechanical Devices, Parts and Supplies. Wall Transformers, Alarms, Fuses, Relays, Opto Electronics, Knobs, Video Accessories, Sirens, Solder Accessories, Motors, Heat Sinks, Terminal Strips, L.E.D.S., Displays, Fans, Solar Cells, Buzzers, Batteries, Magnets, Cameras, Panel Meters, Switches, Speakers, Peltier Devices, and much more....

www.allelectronics.com

Free 96 page catalog

1-800-826-5432

PRINTED CIRCUIT BOARDS

QUALITY PRODUCT
FAST DELIVERY • COMPETITIVE PRICING

- Aluminum-Backed PCB
- Single & Double Sided
- SMOBC, RoHS
- LPI Mask
- Through Hole or SMT
- Nickel & Gold Plating
- Routing or Scoring
- Electrical Testing
- Artwork or CAD Data
- Fast Quotes
- Flex Circuits

10 pcs. (3 Days)
1 or 2 Layers $249

10 pcs. (5 Days)
4 Layers $695
(up to 30 sq. in. each)
Includes Tooling, Artwork, LPI Mask & Legend

PROTOTYPE THROUGH PRODUCTION

PULSAR, INC.
9901 W. Pacific Ave. • Franklin Park, IL 60131
847-233-0012 • Fax: 847-233-0013
www.pulsar-inc.com • Email: sales@pulsar-inc.com

Flashlite 186

- 186 processor @ 33 MHz
- DOS w/ Flash File system
- 44 Digital I/O lines w/ CPLD
- Console / Debug Serial Port
- 7-34V DC or 5V DC power
- 2 Serial Ports
- Accepts 8MB DiskOnChip
- 2 16-bit Timers
- 512K DRAM & 512K Flash
- Watchdog Timer
- Expansion options with Peripheral Boards

$69 QTY 1

$99 Development System

Development kit includes:

- Flashlite 186 controller
- Borland C/C++ ver 4.52
- FREE Email Tech Support
- Serial Driver library
- AC Adapter and cable

Call 530-297-6073 Email sales@jkmicro.com
On the web at **www.jkmicro.com**

JK microsystems

NEW
RS232 to TCP/IP

- TCP/com™ v2.0, RS232 to TCP/IP software. Plus TCP/IP to RS232.
- WinWedge™, RS232 or TCP/IP data direct into any Windows app. - Excel, Access, etc.

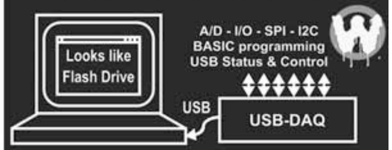
TAL tech
Free 30 day evals at
www.taltech.com

I2C SPI 1 Wire

3 Separate Buses
5V & 3V
Simple ASCII Interface
Cross Platform - All OS

www.i2cchip.com

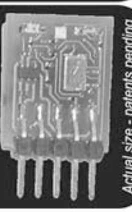
World Leading Driver-Free Win/Mac/Linux USB Chips



- USB-DAQ / FileSys sensing & logging
- USB-to-UART/I2C/SPI and I/O expanders
- No microcontroller programming required
- Add USB to your products in a day!

www.hexwax.com - Mouser - Farnell - Digikey

Amazing PIC programmer
Most devices supported
ICSP, SQTP, & copy limits
$32 at Digikey & Mouser
www.flexipanel.com



Prolific USA

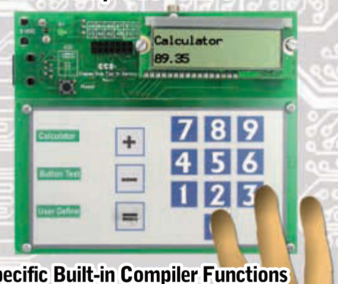
World Class Solutions

www.ProlificUSA.com

USB 3.0
USB 2.0
GPS SoC

Capacitive Touch Development Kit CCS

Create Human Touch Applications for Microchip PIC® MCU



- Specific Built-in Compiler Functions for PIC16LF727
- Includes 14 example programs

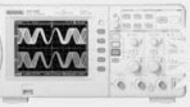
www.ccsinfo.com/ccap
262.522.6500 x35
sales@ccsinfo.com

$469 Includes PCW Compiler

PIC®MCU and dsPIC®DSC are registered trademarks of Microchip Technology, Inc.

RIGOL Beyond Measure | NKC ELECTRONICS

Electronic Design 2008 PRODUCT of the YEAR AWARD



From $595

Rigol Technologies DS1000E series
Up to 1GSa/s and 1Meg memory
50MHz and 100MHz models - TFT LCD - USB
Advanced triggering: Edge, Pulse, Video, Slope, Alt

Zeroplus LAP-16032U Logic Analyzer
$120
16-channel - 100MHz - USB 2.0
SPI - i2c - UART - 7-segment
2 free additional protocols

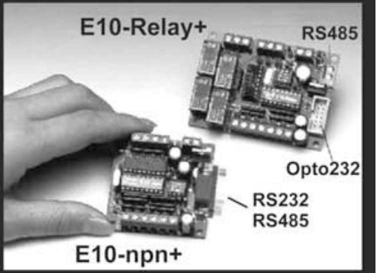
Digilent FPGA Boards
Xilinx Basys2, Nexys2, Spartan 3E Starter Board

Academic pricing for qualified buyers
Basys2 100K gates $79.00

WWW.NKCELECTRONICS.COM
OFFICIAL RIGOL DISTRIBUTOR | GLOBAL SHIPPING

The $69 PLC

Work as Stand-Alone Ladder Logic PLC.
Or as Smart Remote I/Os of PC/ PLCs.
RS485 allows 256 units to be networked.



Incredibly Easy to Program!
Our software is used by many colleges for teaching PLCs!

Get Free Ladder Logic Simulator:
www.tri-plc.com/cci.htm

TRi Triangle Research

Tel: 1-877-874-7527 - PLC specialist since 1993

PCB Fab & Assembly Since 1997

Experts in Quick Total Turnkey Assembly at Low Prices!

- PCB Design, Fab & Assembly
- Cable Assembly
- Box Integration
- Functional Testing
- BGA
- Product Development
- ISO Certified

CapTron Corporation
301-869-6100
sales@captroncorp.com

CUSTOM MEMBRANE KEYBOARDS / SWITCHES

- 1 TO 2 WEEKS TURNAROUND
- VERY COMPETITIVE PRICING
- Ex.: (5) 4-switch keyboards for $395.00
- PCB backed switches
- Custom metal backplates/assemblies
- Electronic assemblies/graphic overlays
- Electronic file transfer capabilities

Picofab Inc.
4780B Blvd. Henri-Bourassa
Charlesbourg, Quebec, Canada G1H 3A7
Tel: (418) 622-5298 • Fax: (418) 622-9996
Email: sales@picofab.net

Add a color touch interface to your embedded product!

- High level RS232 interface
- Easy to program
- Low cost
- In stock!

Add color graphics to any 8/16 bit embedded system. Easy, fast and flexible. Up and running in hours!

REACH TECHNOLOGY INC.

www.reachtech.com • 510-770-1417

842 Boggs Avenue • Fremont, CA 94539

Solve complex signal acquisition problems...

- positioning & control
- environmental
- acceleration
- transients
- pressure
- vibration
- sonar
- GPS
- Linux Driver
- Guaranteed in stock
- Many newly added features
- 16-bit analog inputs and outputs
- Million sample FIFO eliminates interrupts
- Wide analog input and output ranges
- -40°C to +85°C Standard
- Order 24/7, fast and easy.

Google Checkout — Fast checkout through Google

www.stx104.com

Apex Embedded Systems

help@stx104.com • 608-256-0767 x24

PHYTEC

••• Inside great products. Behind great ideas.

phyCORE® System on Modules:

- shorten time-to-market
- reduce development costs and avoid substantial design issues and risks
- Windows® Embedded CE and Linux BSPs (processor-dependent)
- $129/unit benchmark price at 1K for ARM9-based SOM
- Design Services available to assist with deployment into target applications

phyCORE-LPC3250

ARM11: i.MX35, i.MX31
ARM9: i.MX27, LPC3250, LPC3180
Cortex M3: STM32F103
ARM7: LPC2294
XScale: PXA270
x86: Z510, Z520, Z530 (Atom®)
Blackfin: ADSP-BF537
Coldfire: MCF5485
PowerPC: MPC5554, MPC5567, MPC5200B, MPC565, MPC555

phyCORE® Rapid Development Kits include SOM, Carrier Board, LCD (kit specific), schematics, software, free BSP for applicable kits and a start-up guarantee. The Carrier Board serves as a target reference design, allowing the SOM to easily port to the user's target hardware.

www.phytec.com | 800.278.9913 | www.phycore.com

www.can232.com

The original (Controller Area Network) CAN to RS232 serial converter.

Only $108 €89

CAN232 Features:
Free sample programs
8-15VDC supply via CAN
Timestamp in mS
Small size 2.7" by 1.2"
100% Bandwidth up to 125Kbit
Both 11 & 29 bit ID support
32 Message Receive FIFO
Works up to 1Mbit CAN
Simple ASCII protocol
Supports RTR Frames
Max 230Kbaud RS232
Firmware upgradable
No drivers needed
OS independant
CE Approved

CANUSB Features:
Free ActiveX component
PC, MAC & Linux support
Both 11 & 29 bit ID support
Simple CAN logger included
Free Threaded Windows DLL
Firmware upgradable via USB
Sample programs in C, C++, VB, Delphi, C#, PureBasic etc.
No need for external power
Works up to 1Mbit CAN
Supports RTR Frames
USB 2.0 Full Speed
Free USB drivers
CE Approved

Only $154 €129

www.canusb.com

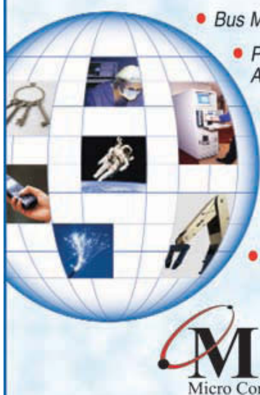
ezLCD - The "Smart Display" makes integrating a GUI ez!

*Versatile Programming LCD module
*USB, SPI, RS232 Interfaces
*Bright 250 Nit LED Display (800 x 600)
*Integrated Touch Screen
*LUA Scripting Language capable-
For stand alone embedded apps
*Memory 3.8 MB + SD to 2G
*ezLCD's are also available in 2.7", 3.5", 5.6", 6.4", 10.4"

www.earthlcd.com

Call for Custom Display Configurations

I²C™/SMBus



- Bus Monitors
- Protocol Analyzers
- Host Adapters
- Multiplexers
- Battery Applications
- Software Tools

MCC
Micro Computer Control

I²C is a trademark of Philips Corporation

www.mcc-us.com

T-Box™

Low cost Temperature Data Acquisition and Control

NEW!

- 3.0 x 4.0", 50 µA standby, 200 mA, 6-24V DC
- C/C++ programmable, Ready to use firmware
- 100+ Temperature IC-Sensors with 0.5°C accuracy
- Thermocouple with 24-bit ADC, 12-bit ADC
- CompactFlash with FAT file system support
- Solenoid drivers, LCD, RS232, ZigBee wireless
- 10/100-baseT Ethernet or USB
- Aluminum box with screw terminals

$89 OEM board

TERN INC. 1950 5th Street, Davis, CA 95616 USA
Tel: 530-758-0180 • Fax: 530-758-0181 www.tern.com • sales@tern.com

PC/104 PERIPHERALS

SCIDYNE Offers a Full Line of Innovative Modules for PC/104 Applications

ADIO-104 *This Module Does it all!*
•16 Analog Inputs •5V Power
•8 AnalogOutputs •Pulse Accumulator
•24 Digital I/O lines •Open-Drain Outputs

DIO96-104
•96 Bi-Directional Digital Channels
•Input, Output and Strobed I/O functions
•Uses Familiar 82C55A Chips

XIO-RO8

EMBEDDED-PC PC/104 MODULE

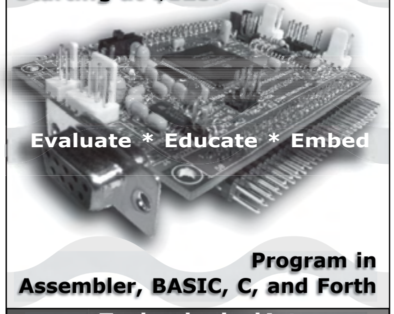
Add eight High-Power relay outputs to any digital port.

Call or Visit us on the web
SCIDYNE® 1-877-724-3963
www.scidyne.com

Adapt9S12XDP512
Modular Prototyping System

* Robotics and Mechatronics
* Electronic Fuel Injection
* Freescale 9S12XDP512
* RTOS-capable

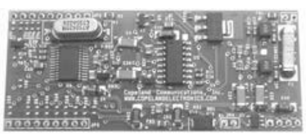
Starting at $125!



Program in Assembler, BASIC, C, and Forth

www.TechnologicalArts.com

Low Cost Embedded OEM
Modems & Device Servers



Low Cost 56K V.92 MNP4 Modems, Dialup Device Servers, Ethernet and WiFi—all in one form factor!

- 300 – 56K baud V.92
- V.42 - V.42bis - MNP4
- Global Compliance—5KV
- FCC, CE and TBR21 approvals
- AT command set
- SocketModem™ compatible
- Low Power—35mA
- Small Footprint 1.045" x 2.54"
- Wireless Modules—902-928MHz - 33Kb
- Auto Error Detection & Flow Control
- Modules are H/W & S/W compatible
- Custom antennas
- USB 2.0 Modules—Adapt legacy RS-232 DB9 to USB
- WiFi, Ethernet available soon!

www.copelandcommunications.com

Copeland Communications, Inc.
(614) 475-1690

TECHSOL
TECHNICAL SOLUTIONS INC

TPC-43A

Fanless Wall-mountable 32bit 200Mhz ARM9 CPU SDRAM/FLASH with Linux 2.6.2x with drivers & GUI

RTC w/battery
16bit Audio
10/100 Ethernet
RS232, IrDA
USB Host
USB Device
SD/MMC card
Optional: Java
- Bluetooth
- ZigBee
- WiFi

4.3" WQVGA (480x272) x 65K Color Touch Screen LCD w/backlight

From $339

www.medallionsystem.com

CROSSWARE®
Tools for Embedded Development

ARM7
ARM9
CORTEX-M3

A quality tool suite for ARM® microcontrollers

- ❑ *C/C++**
- ❑ *Code Wizards*
- ❑ *Debugging***
- ❑ *Simulation*
- ❑ *Support*

* Embedded C++
** Including Jaguar interface

Advanced software tools since 1984

www.crossware.com
360-812-2397

Weather Instruments for PCs

AAG electrónica

www.aagelectronica.com

Low Cost CAD Software for Windows XP, NT and Vista

- Circuit design package with schematic entry circuit-board layout with autorouting and simulation for only $499!
- Buy modules starting at $119 **(SuperCAD, SuperPCB, mentalSPICE & SuperSIM)**
- Order and download instantly
- Full up package allows up to 16 layers plus 4 power planes
- Manufacture circuit boards at <u>any</u> board house

Mental Automation, Inc.
253-858-8104
www.mentala.com

Order online at:
www.melabs.com

microEngineering Labs, Inc.
Development Tools for PIC® Microcontrollers

Phone: (719) 520-5323
Fax: (719) 520-1867
Box 60039
Colorado Springs, CO 80960

USB Programmer for PIC® MCUs **$89.95** (as shown)

RoHS Compliant

Programs PIC MCUs including low-voltage (3.3V) devices

Includes Software for Windows 98, Me, 2K, & XP

With Accessories for $119.95: Includes Programmer, Software, USB Cable, and Programming Adapter for 8 to 40-pin DIP.

Serial LCDs

2-line x 16	$39.95
4-line x 20	$49.95

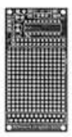
Quantity Discounts Available!

LAB-X Experimenter Boards

Pre-Assembled Boards
Available for 8, 14, 18, 28, and 40-pin PIC® MCUs
2-line, 20-char LCD Module
9-pin Serial Port
Sample Programs
Full Schematic Diagram

Pricing from $79.95 to $349.95

PICPROTO™ Prototyping Boards



Double-Sided with Plate-Thru Holes
Circuitry for Power Supply and Clock
Large Prototype Area
Boards Available for Most PIC® MCUs
Documentation and Schematic

Pricing from $8.95 to $19.95

BASIC Compilers for PICmicro®

Easy-To-Use BASIC Commands
Windows 9x/Me/2K/XP Interface

PICBASIC™ Compiler $99.95
BASIC Stamp 1 Compatible
Supports most 14-bit Core PICs
Built-In Serial Comm Commands

PICBASIC PRO™ Compiler $249.95
32-bit signed variables and math operations
Supports Microchip PIC10, PIC12, PIC14, PIC16, PIC17, and PIC18 microcontrollers
Direct Access to Internal Registers
Supports In-Line Assembly Language
Interrupts in PICBASIC and Assembly
Built-In USB, I2C, RS-232 and More
Source Level Debugging

See our full range of products, including books, accessories, and components at:
www.melabs.com

INDEX OF ADVERTISERS

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

Page		Page		Page		Page	
78	AAG Electronica, LLC	56	ExpressPCB	15	Lemos International Co. Inc.	75	Pulsar, Inc.
32	AP Circuits	10	ezPCB	11	Linx Technologies	17	R4 Systems Inc.
29	ARM techcon^3 Conference	76	FlexiPanel Ltd.	77	MCC (Micro Computer Control)	C3	Rabbit, A Digi International Brand
75	All Electronics Corp.	11	Front Panel Express LLC	71	MachinePIER	77	Reach Technology, Inc.
77	Apex Embedded Systems	71	Grid Connect, Inc.	78	Mental Automation, Inc.	28	SoC Conference
7	Atmel	19	HobbyLab, LLC	5	Microchip Technology, Inc.	27	Saelig Co.
33	CWAV	76	I2CChip	78	microEngineering Labs, Inc.	77	Scidyne
69	CadSoft Computer, Inc.	31. 42	ICbank Inc.	39	Mouser Electronics	47	Sealevel Systems
23	Calao Systems	19	Intuitive Circuits LLC	76	NKC Electronics	1	Synapse Wireless
76	CapTron Corp.	23	Ironwood Electronics	C2	NetBurner	76	TAL Technologies
30	Comfile Technology, Inc.	32, 34	JKmicrosystems, Inc.	70	Nurve Networks LLC	78	Technical Solutions, Inc.
78	Copeland Communications	75	JKmicrosystems, Inc.	15	PCBCore	40, 41	Technologic Systems
78	Crossware Products, Inc.	2	Jameco	55	PCB West Design Conf.	78	Technological Arts
76	Custom Computer Services, Inc.	70	Jeffrey Kerr, LLC	34	PCB-Pool	77	Tern, Inc.
75	DLP Design	18	Keil Software	C4	Parallax, Inc.	25	Texas Instruments
19	DesignNotes	57	LPKF Laser & Electronics, N. America	77	Phytec America LLC	75	Trace Systems, Inc.
21	EMAC, Inc.	70	Lakeview Research	76	Picofab Inc.	76	Triangle Research Int'l, Inc.
77	Earth Computer Technologies	77	Lawicel AB	20	Pololu Corp.	22	Trinity College Fire-Fighting Robot Contest
21	Elprotronic	3	Lecroy	76	ProlificUSA		

PREVIEW of October Issue 231

Theme: **Signal Processing**

INTELLIGENT ENERGY SOLUTIONS **Frequency Sensing Made Simple:** Power Grid Frequency Monitor Design

IR Signal Control

Digitally Controlled Amplifier

THE DARKER SIDE **Multirate Techniques and CIC Filters**

ABOVE THE GROUND PLANE **Capacitor ESR Measurement**

FROM THE BENCH **Air Flow Analysis**

SILICON UPDATE **Thumbs Up:** The ARM Saga Continues

ATTENTION ADVERTISERS

November Issue 232 Deadlines

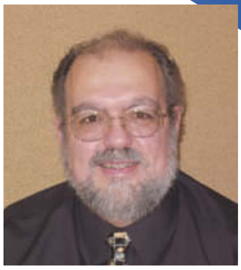
Space Close: Sept. 10
Material Close: Sept. 18

Theme

Analog Techniques

Call Shannon Barraclough now to reserve your space!
860.875.2199
e-mail: shannon@circuitcellar.com

PRIORITY INTERRUPT



by Steve Ciarcia, Founder and Editorial Director

The Critter Chronicles: The War Continues

Ordinarily, I wouldn't dwell on a nontechnical subject this long, but apparently I hit a nerve with readers when I first described my problems with moles (*Circuit Cellar* 228). It seems strange that discussing mole eradication in an engineering magazine should elicit the most reader feedback in a dozen years, but here we are again.

When I left you last time, I was experimenting with buried vibrators as the latest technical solution to the problem. Years ago, I had tried the commercial battery-operated vibrators with no success, but I wondered if it was merely an issue of the magnitude of the vibration and not the technique. This time I buried powerful 12-VDC motors with off-center shaft-mounted weights. The resulting vibration was so powerful, in fact, that it was virtually impossible to hold one of these motors in your hand. Even with the motors enclosed in plastic pipe and buried, I could feel the periodic low frequency rumble as I walked around the house. And the results?

I am very unhappy to report that it was an utter and absolute failure! Apparently, constructing rotating shaft vibrators with this much amplitude doesn't work. They just can't take the duty cycle (range 5% to 10%) without self-destructing. I didn't dig them all up, but the ones I did showed either bearing or commutating brush failure (either seized up or opened the electrical circuit) and all were dead in about three weeks. That much mechanical vibration must either necessitate a minuscule duty cycle or some super-duper bearings and robust mechanics. So much for that idea.

Now the moles have even started digging tunnels between the edge of the garage and the blacktop driveway. The good news is that this very irritating dig site afforded a perfect place to retest the method suggested by commercial exterminators and a few readers—castor oil. For this test, I bought a quart of MoleMax (a castor oil solution usually connected to a hose and enough to treat 10,000 square feet) and poured the entire bottle along the 20′ strip of soil between the concrete and blacktop where the mole keeps digging. About a week after soaking in the concentrated application of castor oil, the mole was back! Next …

My next strategy was: "If you can't beat 'em, crush 'em." Seriously, the biggest aggravation about moles is the ugly raised tunnels across the flat mulch. I had already determined that lots of vibration (from my big tractor) kept them away for a day or two, but the big agricultural tread tires tear up the yard more than the moles. Since the experiment with the buried motor vibrators was a dismal failure, the only way to still get substantial vibration was to drive around in a smaller tractor (with turf tires and a 625-lb water-filled lawn roller) that flattens all the mole hills too. While the jury is out on its ultimate success, there is a great deal of satisfaction in squashing the little suckers as I even out their dirty work.

One of the more popular eradication methods suggested by readers and frequently mentioned on the Internet is probably what I'll try next—pumping carbon monoxide into the tunnels. Typical methods involve attaching a hose to a motor vehicle exhaust, but there is no way I can get my pickup truck out in the back yard without doing more cosmetic damage than I'm trying to prevent. Disregarding chain saws and handheld gas-powered devices, the most portable exhaust generator I have appears to be either my two-cycle, 3-HP (30-lb) snow blower or my four-cycle, 10-HP (very very heavy) snow blower. I'll try simple and light first. And, yes, I know that this two-cycle engine is very inefficient for producing CO. Given the back pressure added by a hose, the exhaust is probably mostly unburned hydrocarbons, but I trust all that crap is just as un-breathable as CO.

Finally, the one method that might guarantee absolute success is something I don't dare try. Many readers proclaimed success using techniques involving stuffing road flares, smoke bombs, or gasoline down the holes and lighting them. The best commercial solution like that appears to be a $2,000 Rodenator (www.rodenator.com). It works by injecting a mixture of oxygen and propane into the tunnel system and literally exploding the entire tunnel network. (Make sure you watch the *Cadyshack* video on their site.) ;-)

Certainly, I've been known to spend thousands in pursuit of some obsessions, but this is a safety issue. Most Rodenator customers use the device on farms, lawns, or just dirt. The common denominator is good old "nonflammable" dirt. I've got 20-plus years of compacted "very flammable" bark mulch. My fear is that an underground oxy/propane explosion will either start the mulch equivalent of an underground coal mine fire or create a giant blown up cloud of fine bark dust particles that ignites like a grain silo explosion. Basically, while incinerating the entire yard down to the water table might be a viable solution, the neighbors would certainly frown on it.

So, I'm not quite back at square one yet, but options are becoming limited. Certainly, the uniqueness of having all that mulch makes abatement equally unique. I may yet have to resort to one reader's bizarre solution: put a 5-kW generator in a wagon along with a few salvaged Klystron tubes (from microwave ovens) aimed down at the tunnels and just cook a little old-fashioned road kill dinner. Sounds grizzly, but in desperate times …

For more mole war pics, check out www.circuitcellar.com/Newsletter/0709.html.

Steve

steve.ciarcia@circuitcellar.com

Sweet!

Introducing the MiniCore™ Series of Networking Modules

Smaller than a sugar packet, the Rabbit® MiniCore series of easy-to-use, ultra-compact, and low-cost networking modules come in several pin-compatible flavors. Optimized for real-time control, communications and networking applications such as energy management and intelligent building automation, MiniCore will surely add sweetness to your design.

- **Wireless and wired interfaces**
- **Ultra-compact form factor**
- **Low-profile for design flexibility**
- **Priced for volume applications**

Wi-Fi and Ethernet Versions

MiniCore Module Development Kits

From $99 Limited time offer.

Buy now at: **1.888.411.7228 • rabbitwirelesskits.com**

1.888.411.7228
rabbitwirelesskits.com
2900 Spafford Street, Davis, CA 95618

RABBIT™

09009

MICROCONTROLLER EDUCATION
WE WROTE THE BOOK!

What's a Microcontroller
Understanding Signals
Applied Sensors
Basic Analog and Digital
Robotics with the Boe-Bot
IR Remote for the Boe-Bot
Process Control
Applied Robotics with the SumoBot
Smart Sensors & Applications

SINCE 1997

12 years ago Parallax created the Stamps in Class program to address the need for affordable electronics and programming educational resources. We now offer nine comprehensive tutorials designed to introduce students and educators to electronics, programming, robotics, and sensor management with the easy-to-use BASIC Stamp microcontroller. Our interchangeable hardware platform and our free software, tutorials and sample code downloads make outfitting your classroom affordable. Our BASIC Stamp Educators Courses and broad support base help you get started and keep your program growing. To learn more about Stamps in Class, visit www.parallax.com/education.

BASIC STAMP PROGRAMMING TO THE PROPELLER CHIP, WE'VE GOT YOU COVERED.

In the spirit of Stamps in Class, we have created our Propeller Education Kit and Labs to provide an affordable platform for learning cutting-edge multi-processing microcontroller technology. With 8 processors that run simultaneously to perform independent or cooperative tasks, the Propeller chip is an ideal controller for integrated embedded system design projects. As always, our manual, labs, software and sample code are free downloads.

- Propeller Education Kit - 40 pin DIP Version (#32305; $99.99)
- Propeller Education Kit - PropStick USB Version (#32306; $99.99)

For more information visit www.parallax.com/education for details.

To learn more about Parallax microcontroller education visit www.parallax.com/education. Quantity discounts are available. Parallax Sales Department: 888-512-1024 (Mon-Fri, 7am-5pm, PDT).

PARALLAX INC
www.parallax.com
Friendly microcontrollers, legendary resources.

CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

BONUS ARTICLE

by Sharad Sinha

Verification and Simulation of FPGA Designs

Working with and testing FPGA designs requires you to develop, execute, and maintain a proper verification plan. Sharad covers simulation and verification from an engineer's perspective. He explains bug/defect tracking, revision control, and test documentation.

You're reading *Circuit Cellar*, so I assume you've worked with FPGAs at some point. If you've designed and tested FPGA-based projects, you're aware of the various processes that you must follow to develop, execute, and maintain a proper verification plan. Any verification plan will always include a functional and timing simulation of the FPGA design to achieve a finished design. However, at the same time, it is possible that there is another important goal to be achieved during simulation. This is to print appropriate messages on the console so you can obtain correct information. This kind of situation generally arises when design companies have to provide a client with the entire simulation and verification setup. Sometimes a client will request this. Other times the design house may prefer to have the simulation and test setup delivered—in addition to the FPGA image or the source files—to ensure that it does not run into problems if the client decides to test the source code. Another advantage is that it helps the person running the simulation and test. He doesn't have to review and sift through the simulation waveforms, which is a time-consuming and tiring task that's particularly difficult to do with large designs.

Figure 1 shows a typical setup for a design under test (DUT). The Driver module sends test vectors as input to the DUT. The Monitor module verifies the signal pattern coming out of DUT against the expected pattern. The Driver and Monitor modules are the two important portions of what is generally referred to as the test bench (TB). The complete setup is generally referred to as TB TOP, which includes the instantiated DUT. This entire setup works in a simulator environment, as shown by the gray box. Printing appropriate test success/failure messages is handled in the Monitor module.

To perform verification, designers and verification engineers generally opt for a script-based environment. A properly developed verification script makes design verification easy and structured, so let's review the basic objectives of a verification plan. One, test the functionality of the DUT to ensure that it functions as intended. Two, find the functional and timing-related bugs. Three, make the bugs visible to the verification engineer, designer, and the project manager. And four, help the designer initiate a design change by showing the connection between the bug and functionality.

How do you test functionality to catch the function- and timing-related bugs? This is where functional and timing simulations come into play.

FUNCTIONAL SIMULATION

You can test the functionality of the DUT with no information about the timing of signals in the actual target

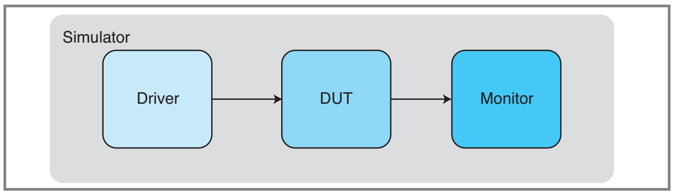


Figure 1—This is a typical setup for a functional and timing simulation of a DUT.

FPGA. Basically, a given input pattern to the DUT must give an output pattern expected for that input pattern. The routing and delay information related to the target device are not used in this simulation.

The register transfer level (RTL) description of the DUT is used in this simulation to check for RTL integrity. The driver sends different sets of signals as input to the DUT, and the monitor compares the DUT output with the expected output. What are these different sets of signals? Let's consider an example. Say the DUT is a logic design that's supposed to take Ethernet packets as input, strip all headers and control information, and send out only the payload on different FPGA ports, as per the packet type, to other modules connected to the FPGA (see Figure 2). This is the DUT's primary function. As you can see, this is basically the routing of different kinds of packets to different ports.

The Driver module in the test bench is then supposed to send different kinds of Ethernet packets as input to the DUT. These Ethernet packets will form distinct test cases. The IEEE 802.3 standard defines the structure of an Ethernet frame (see Table 1). There are 27 different values for the Ethertype field. Let's say that these 27 fields are allotted to the four ports, with ports 1, 2, and 3 getting seven fields and port 4 getting six fields. For the functional simulation, there should be a minimum of 27 test cases to ensure that the proper packets are routed to the proper ports. It is assumed that all other field content, except the payload, remains the same.

The Driver module in the test bench generates these packets and sends them to the DUT. It becomes very easy to conduct these 27 test cases if there are 27 text files, one for each test case, and the driver reads the file corresponding to the test case selected by the user and then sends its contents to DUT. The driver is also responsible for providing clock and reset signals to the DUT.

The Monitor module easily determines which payload was meant for which port. Hence, it can easily determine if the correct payload arrived on the correct port. This is because the payload information is present in the text files. Both the driver and the monitor can read these files and extract the needed information.

TIMING SIMULATION

The simulation setup remains the same as that in a functional simulation—except instead of using the RTL model of the DUT, the DUT model generated by the synthesis tool is used. Examples of synthesis tools are Altera's Quartus II and Xilinx's ISE. During the synthesis process, there will generally be an option to generate the post place and route model of the design. This model is generated after the placement of logic in the FPGA resources and the synthesis tool interconnects (routes) the resources. This model is a .vo or .vhd file in the Quartus II. It is a .v or .vhd file in the Xilinx ISE. This depends on whether the RTL top module is written in Verilog or VHDL. A standard delay format (SDF) file—extension .sdo in the Quartus and .sdf in the Xilinx ISE—will be generated too. The SDF file has information on the static delay characteristics associated with the mapping of logic into FPGA resources (cells, LUTs, routing, etc.) for the target FPGA. The simulator uses the SDF file to perform timing simulation for the design.

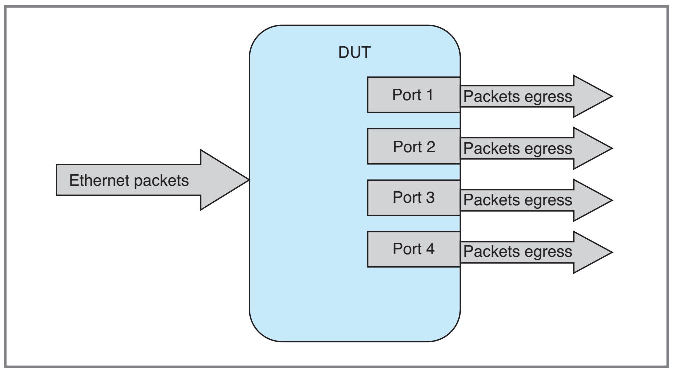


Figure 2—This is a DUT which receives Ethernet packets and sends them out on different ports based on the information in the Ethertype field of the packet.

The verification staff may decide to go for only the functional simulation post-place and route. In that case, one has to comment the reference to the SDF file in the .vo or .vhd model file. If this is not done, it may not make any difference if the simulator is instructed—through GUI preference or command line switch—to not use the SDF file, and the simulator will still use the SDF file. This is basically a result of how the model is generated by the synthesis tool and how it references to the SDF file. Tools require either manual commenting at the end or they will give an option to disable back annotation during model generation. For the .vo model (say, dut.vo generated by the Quartus II), the reference to the SDF file is the following statement in the .vo file:

```
initial $sdf_annotate("dut_v.sdo");
```

It is also important to account for the timing models generated by the synthesis tool. For instance, the Quartus II can generate two different types of timing models: fast corner and slow corner. The former timing model is a best-case analysis of the design. Best-case analysis

Preamble	Start of frame Delimiter	MAC Destination address	MAC Source address	Ethertype/length	Payload	CRC 32
Seven octets of 10101010	One octet of 10101011	Six octets	Six octets	Two octets	46-1500 Octets	Four octets

Table 1—The structure of Ethernet Frame/Packet. MAC address stands for Media Access Control address, which is an address assigned to a device which can communicate via Ethernet protocol. CRC 32 is the 32-bit Cyclic Redundancy Check field.

means the fastest device (in the particular FPGA family) at high voltage and low temperature. Slow corner, or worst-case analysis, refers to the slowest device (in the particular FPGA family) at low voltage and high temperature. For instance, for a commercial-grade device, if the standard V_{CCINT} is 1.2 V, then the fastest device at 1.15 V and 0°C is considered for fast corner (best) analysis. The slowest device at 1.25 V and 85°C is considered for slow corner (slow) analysis.

It is quite possible that timing closure may not be achieved with the slow corner model while it is achieved with the fast corner model. This should not be a concern, as long as it is known that the design is not meant for application under the worst-case conditions, or it is understood that the PCB—which has the FPGA—may itself fail under those conditions of process, voltage, and temperature (PVT).

The reason for the two timing models is that the internal timing characteristics of the same device may vary at different conditions. For instance, the output-register-to-output-pad delay may change at different temperatures. It may be, say, 0.4 ns, under worst-case conditions (low voltage and high temperature), while it may be 0.1 ns under best-case conditions (high voltage and low temperature).

When performing timing simulation, it is also important to ensure that the simulator does not optimize the netlist generated by the synthesis tool. If this happens, it can lead to unexpected problems in simulation. To ensure that the simulator does not do this optimization, you can use a `-novopt` switch with the `vsim` command in ModelSim.

VERIFICATION LANGUAGES

A DUT's RTL description is usually presented in either Verilog or VHDL. Simple to moderately complex test benches can be written in either language. In fact, you can find test benches generated in either Verilog or VHDL when you use IP cores provided by FPGA vendors like Altera and Xilinx along with their synthesis tools. However, these languages are primarily meant to describe hardware. For a complex design, hardware verification languages (HVLs) ease the task of writing a TB. HVLs provide features like a high level of data structures, object orientation with inheritance, and temporal assertions. This makes writing a TB easier for complex verification tasks.

There are also some prominent commercial verification language solutions like *e* from Verisity and OpenVera from Synopsys. System Verilog from Accelera is another language that's being widely adopted for the verification of designs.

Using HVLs requires simulation tools that support them. At this time, EDA tools that support HVLs are expensive, which means many small design houses can't afford them. Moreover, FPGA designs are generally used because of the lesser time to market and lower overall development costs compared to ASIC designs. Hence, design houses generally use a lot of discretion when investing in such tools. There is no doubt that processes like ASIC verification on FPGAs or the use of FPGAs in safety-critical and failsafe applications will definitely benefit from the use of HVLs. This is because their usage has demonstrated reduction in verification time for complex projects, as well as far better verification results compared to the usage of HDLs.

EFFECTIVE SCRIPTING

Effective scripting eases the workload for a verification engineer. You can automate most verification-related tasks with a properly written script. A well-written script is a simple and easy "user interface" for the verification engineer to carry out the tasks involved in verification. A single script file should be able to perform tasks in the following order: compile the source code and TB files; load the design; prompt you to select the type of simulation (functional, post-route functional, post-route timing); prompt you for the test case to be executed based on the type of simulation (if needed); and print messages related to the test cases in the simulator transcript window. The messages can be related to the passing or failure of the test case or any other descriptive and useful information related to the test case. An example of the descriptive information is the amount of time remaining to fill instantiated RAM in the design.

To compile the source code and load the design, the script file executes the simulator commands like those available in ModelSim. To print messages, it will set certain global variables to certain values (as you write the TB code you have the freedom to select the variables and their values), which will then be used by the Monitor and Driver modules to print messages on the transcript window. At the same time, the Monitor may open some other file to write information to it. If you take the aforementioned example of the preliminary router DUT, this information can be the payload, the port on which the payload was received, and whether it was received at the correct port, indicating the success or failure of the test case. Since the Driver supplies data to be written to the RAM in the DUT, the module can print messages in the transcript window related to the time remaining to fill the RAM completely. This kind of information is helpful when it takes a lot of time to execute a test case. Simulating a deep RAM fill becomes easier with these messages because you can be sure the simulation is progressing and not hanging.

Let's take the example of the script.do text file, which is written in Tool Command Language (Tcl) and is meant for the ModelSim simulator. It prompts you with three options. Based on the option you select, it executes the corresponding `vlog` (for compilation) and `vsim` (for simulation) ModelSim commands. Note that "work" is the name of the library needed by ModelSim for simulation.

Paths to all the RTL code, TB code, and any library code are indicated in the files.txt file. Note that logprint.do is another Tcl file that can have commands for making a log directory and storing the prints of the transcript window in

Test case no.	Test case description	Pass/Fail	Transcript message (Yes/No)	Waveform checked (Yes/No)

Figure 3—A format to record test case numbers, test case descriptions, the pass/fail status of test case, presence or absence of any associated transcript message, and whether any associated waveform was verified or not.

another file name. In the example logprint.do file, `simulation_print` is a Tcl procedure which when called by its name prints simulation start time and makes a log directory named "simlog." Directory "simlog" contains another directory which takes its name from the Tcl variable `directoryName` and stores the transcript window prints in another text file which takes its name from the Tcl variable `fileName`. Note that the variables `directoryName` and `fileName` take values based on time and hence one can refer to these files based on the time of simulation if multiple simulations are run. The verification engineer can add switches to these commands as per the need.

LANGUAGE OPTIONS

The more prominent scripting languages are Tcl, Perl, and Python. Tcl is widely used because it is supported by almost all EDA tools like the vendor-provided synthesis tools ModelSim, NCSim, and so on. An advantage associated with learning Tcl is that it helps in automating processes in the simulator environment.

Perl is a good language for performing many tasks related to project management. Perl programs cannot be executed from inside a simulator or synthesis tool. The same is the case with Python. However, these languages can be used outside the simulator environment to automate various tasks related to project management.

BUG TRACKING & REVISION CONTROL

It is important to report bugs so a design engineer may learn about it and make the necessary changes. There are various tools for this purpose (e.g., Mantis and IBM Rational ClearQuest). The different stakeholders in the verification plan (i.e., the verification engineer, the design engineer, and the manager) can be given access to all the information related to the bugs. This provides visibility to bugs and helps in their quick and easier tracking and resolution.

During the simulation and verification stage, it is important to maintain a proper timeline of source and TB code updates. Whenever source codes (DUT) are verified and bugs are reported (e.g., in Mantis) and resolved, it is important to move the updated source code into a database. This will help the verification team access the latest code for verification. At the same time, you can also add TB code and scripts to that database so users can always access the latest code. In this way, you can control revisions and ensure that staffers can access project-specific areas of the database (even if staffers are working from different locations).

A database is an important part of a revision control tool like Concurrent Versioning System (CVS), which is open source, and IBM's Rational ClearCase. Bug tracking and revision control tools are included in large software packages (e.g., Rational ClearQuest) that are aimed at large development organizations. Stand-alone revision control and bug tracking tools are available as well.

TEST DOCUMENTATION

Proper test documentation is necessary to finish a project. Not only does it provide information about the success and failure of tests, it also helps present the relevant information to a client. This helps if the client wants to run the test cases and check for simulation success. This is in general a requirement in cases where the client and the design house are in different locations. It is also useful when the design is divided into various large sub-modules and each sub-module is executed by a different team. Complex ASIC prototyping on FPGAs is a good candidate for the proper flow of test information between the different sub-module teams.

An interesting situation arises when a design engineer also works as a verification engineer. Design engineers perform individual module level testing. Most of the time, this is achieved by forcing signals and looking at the waveforms. This is because the individual logic modules are generally small and it's quicker to test them this way. If the design engineer is assigned the task of integration verification where all the modules are integrated and the aforementioned simulation environment set up, he may still look only for the top-level port signals for success. He may forget to check whether the appropriate messages are coming in the transcript window. Generally, a client will not devote too much time to simulation, even if it decides to simulate. To make things easier, it is important to print relevant messages in the simulator's transcript window or to a file. Hence, while preparing the test document, it is important to indicate the ways in which success and failure are recorded. You can use the format shown in Figure 3.

It should be noted that waveform-based verification is not necessary, but transcript messages are imperative. This is one way of presenting the information. The emphasis should be on proper information flow between the stakeholders.

The issues and techniques I described in this article are useful for all kinds of design houses, large or small. You can also use tools like ModelSim Questa that can provide an integrated platform for running test cases, maintaining proper databases, and so on. However, even with such tools, a certain amount of scripting is still needed. I dealt with verification flow and its management. In a future article, I may cover verification techniques like lint tools, assertion-based verification (ABV), formal verification, model checking, and verification methodologies like OVM and VMM. To maintain correspondence, lint tools are used before functional simulation. ABV is useful during functional simulation. Formal verification is used after synthesis and post place and route. OVM and VMM are basically verification management standards related to the primary task of setting up a proper verification environment. They can be considered as a superstructure built upon the verification flow. Again, the usage of all these tools depends on the complexity of the FPGA designs. The flow I described in this article is required irrespective of the complexity.

Sharad Sinha (sharad_sinha@ieee.org) holds a BTech in electronics and communication engineering from Cochin University of Science and Technology in Kerala, India. He worked as a design engineer at Processor Systems India and is now a PhD candidate at The Center for High Performance Embedded Systems at the Nanyang Technological University in Singapore. Sharad's technical interests include embedded systems, reconfigurable computing, FPGA/board design, and engineering project planning/management.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/230.

RESOURCES

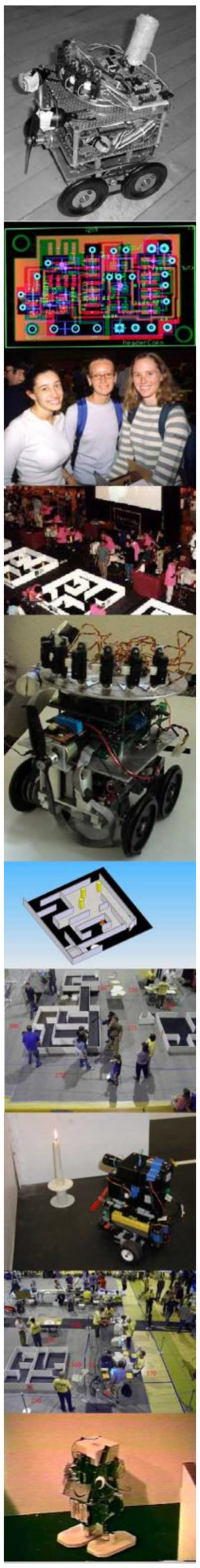
Defect tracking tools, http://testingfaqs.org/t-track.html.

Tck information, Tcl Developer Xchange, www.tcl.tk.

Trinity College Fire Fighting Home Robot Contest

April 10-11, 2010

www.trincoll.edu/events/robot



One of the nation's leading liberal arts colleges, Trinity College offers an ABET-accredited engineering program.

Mission: To inspire inventors of all ages and skill levels while advancing the fields of engineering and robotics.

The Contest: The Trinity College Fire Fighting Home Robot Contest (TCFFHRC) revolves around a simple task: extinguishing a candle flame in the fastest time. Robots are judged on their ability to navigate a floor maze representing a house, find a lit candle and extinguish its flame. Competing robots are truly autonomous—no joysticks in this competition! Fire-fighting robot designs are limited only by the rules of the competition and the imagination of contestants. The contest encourages the application of science and technology in an atmosphere of creativity, teamwork, and friendly competition.

Since 2008, Versa Valve, Inc has challenged robot teams to use its product line in the development of the robot's extinguishing method. Robots in all divisions are eligible to participate.

In 2009, a supplemental challenge sponsored by the Connecticut Council on Developmental Disabilities was introduced. In the RoboWaiter contest, specially-designed robots navigate a model kitchen, work to find a plate of food in the refrigerator and return it to the user.

Unique International Event: The TCFFHRC is a low-cost alternative to other robot competitions, ensuring that it is open and accessible to everyone. Many engineering programs in the United States and across the globe have modeled Trinity's fire-fighting robot theme and adapted it into their curricula. Since 1999, the Israel Ministry of Education has used fire-fighting robotics as the focus of graduation projects in their country's best high schools.

Participation: The contest welcomes 120 robots, and their more than 400 designers. Teams from across the US have participated, including large institutions such as the Massachusetts Institute of Technology, Yale University and Oklahoma State University, as well as many smaller schools such as Tufts University and Wellesley College. The contest has global appeal, welcoming international students from Denmark, United Arab Emirates, Singapore, India, South Korea, Portugal, Indonesia and Argentina. An average of 10 teams from both Israel and China has participated annually for nearly 10 years.

Contest Challenges: The contest has five skill divisions—junior, high school, senior, walking and expert. The new assistive robot division encourages design of autonomous robots to help persons with disabilities. Through the Spirit of an Inventor Prize, judges specifically recognize the entry that shows the greatest ingenuity and creativity regardless of how the robot places in the competition.

Symposium and Olympiad: The Symposium features well-known speakers from the field of robotics in an informal setting that encourages easy interaction. Drawn from government, academia, and the private sector, the speakers address the practical, theoretical, and philosophical issues that involve current robotic trends. The Robotics Olympiad is the first competitive theoretical exam held in conjunction with a leading robot contest and covers four fields central to autonomous robot design: mechanics, electronics, software, and sensors.

Publicity: The contest has enjoyed world-wide coverage in such media outlets as *The New York Times*, *The London Times*, *Scientific American*, *The Chronicle of Higher Education*, *Electronic Design*, *Circuit Cellar*, *Popular Mechanics*, *Forbes.com*, *Robot Magazine* and *Byte*, just to name a few. The contest has been featured on TV and radio, both locally and nationally.

Contact Information: David J. Ahlgren, Karl W. Hallden Professor of Engineering, Director and Host, dahlgren@trincoll.edu. For sponsorship opportunities: Amy Brough, Director of Institutional Support at Amy.Brough@trincoll.edu or (860) 297-5315.