#197 December 2006

CIRCUIT CELLAR®

www.circuitcellar.com

THE MAGAZINE FOR COMPUTER APPLICATIONS

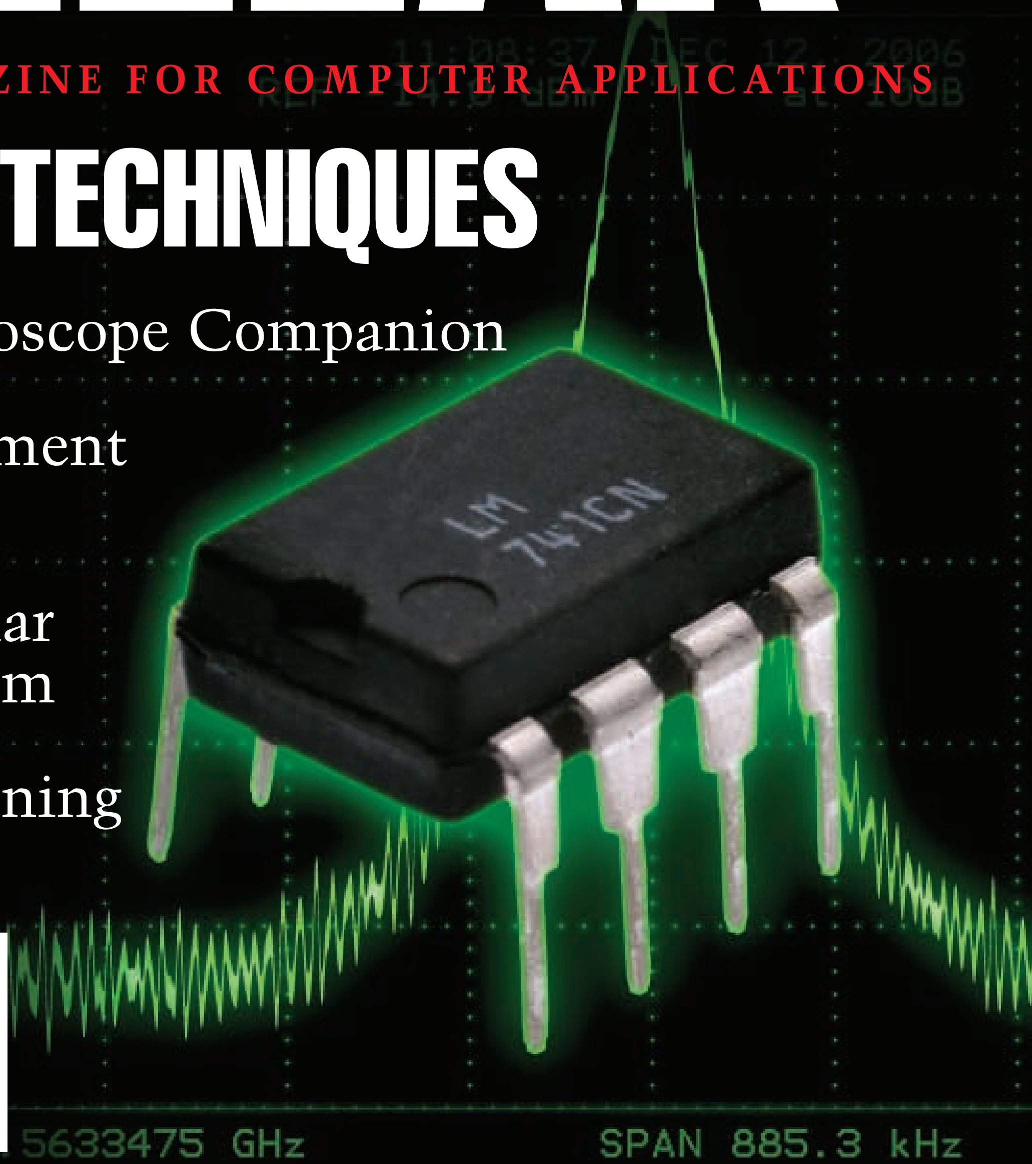
ANALOG TECHNIQUES

Analog Oscilloscope Companion

Digital Instrument Panel Design

Build a Modular Security System

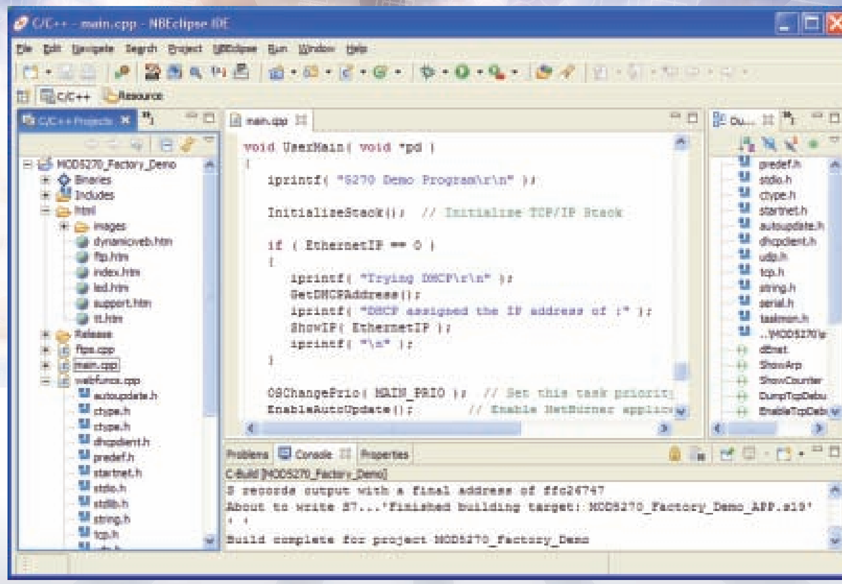
Portable Lightning Detection



$4.95 U.S. ($5.95 Canada)

The NetBurner Eclipse Ethernet Development Kit

Only **$99**



Featuring NetBurner's Eclipse IDE!

One Click Compile and Load | Intelligent Code Completion | Integrated Debugger

The Complete Hardware and Software Solution

Includes NetBurner MOD5270 Ethernet Core Module

Performance and Memory | 32-Bit CPU | Freescale ColdFire MCF5270 147Mhz | 512K Flash Memory | 2MB SDRAM

Device Connectivity | 10/100 Ethernet | 3 UARTs | I^2C | SPI | 47 Digital I/O | SD/MMC Flash Card Support

Communication Software

| TCP/IP Stack | HTTP Web Server | FTP | E-Mail | PPP | Flash File System

Development Software

| NB Eclipse IDE | Graphical Debugger | Deployment Tools | Examples

System Software

| µC/OS RTOS | ANSI C/C++ Compiler and Linker

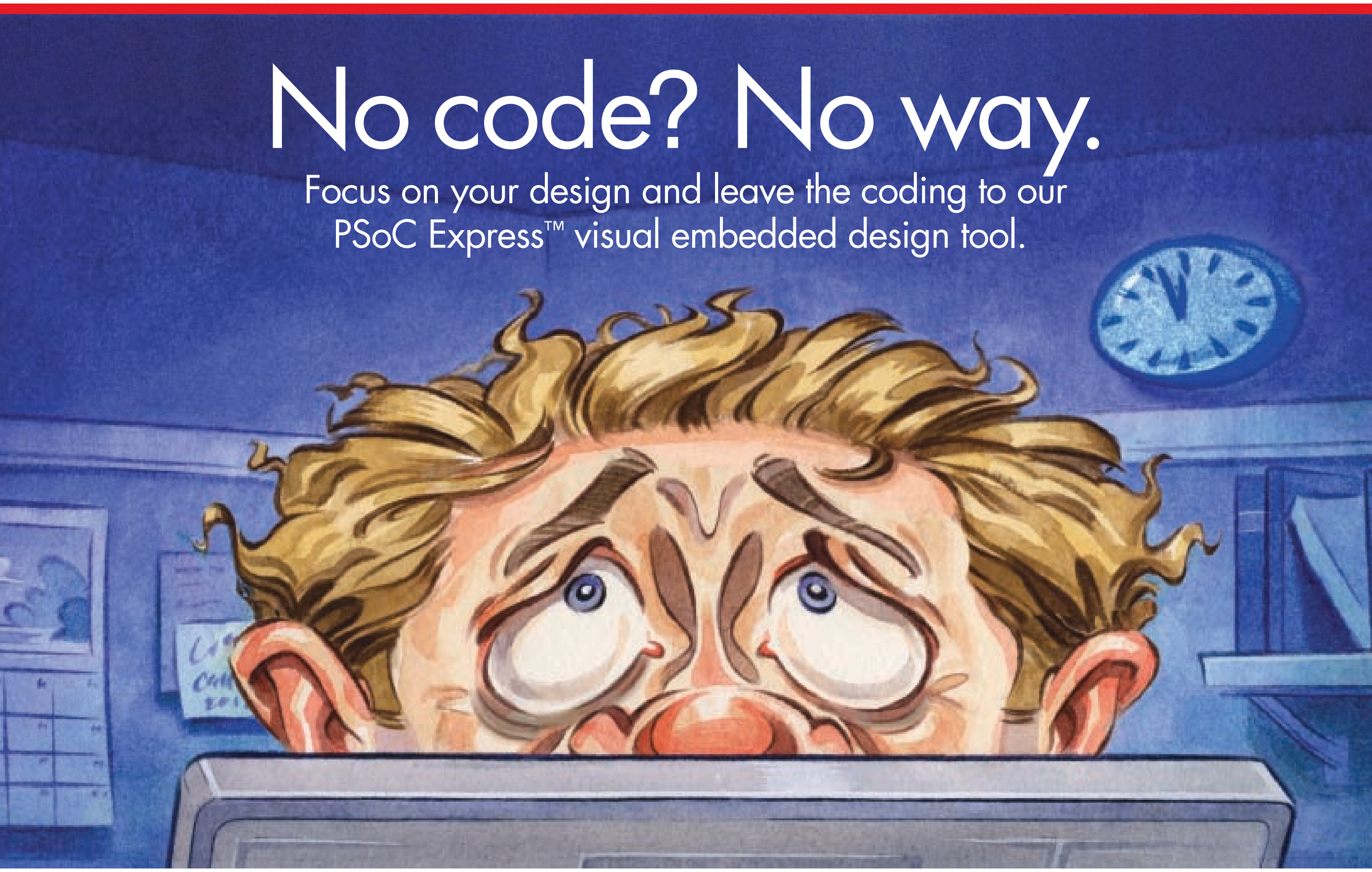
NetBurner.com

freescale™ semiconductor

Product No. | NNDK-MOD5270LC-KIT

Information and Sales | sales@netburner.com

Telephone | 1-800-695-6828



Without writing a single line of assembly or "C" code, generate a complete custom design with PSoC Express. Here's what you get with Version 2.1 of our visual embedded software:

- *Support for our family of Programmable System-on-Chip™ (PSoC®) mixed-signal arrays—powerful, programmable digital and analog blocks with integrated MCU and flash memory*
- *Rich visual environment with simulation enables you to see your design and evaluate its performance instantly*
- *Built-in support for interdevice communication; seamlessly divide design problems into smaller pieces using multiple PSoC devices*
- *Retarget to any PSoC mixed-signal array at any time; design first and select device later*

TRY IT OUT NOW.

Whether you have a minute or an hour to invest, we have a way to get you working with PSoC Express:

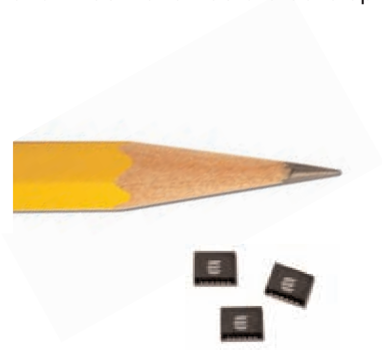
- *View our online PSoC Express demo:* **www.cypress.com/expressdemo**
- *Request a FREE PSoC Express evaluation kit:* **www.cypress.com/expresskit**
- *Test drive PSoC Express at a live seminar near you:* **www.cypress.com/expresstour**
- *Download FREE PSoC Express software:* **www.cypress.com/psocexpress**

Make all your embedded designs fast and easy.
www.cypress.com/getexpress

CYPRESS
PERFORM

CONTROL MADE EASY

Need to monitor or control the small sensors and actuators in your control network? Or make your machine more reliable, easy to assemble and remotely accessed? Use the Pyxos™ platform, the embedded control networking solution that installs with push-button simplicity. Pyxos chips are small enough for nearly any sensor or actuator, and includes a digital I/O and SPI port to use with existing microcontrollers. It extends the power of LonWorks® control networks to the smallest of devices, plays well with others—host processors and PLCs—and has the development tools and reference designs to reduce your development time.



- ☑ Deterministic TDMA protocol
- ☑ Bit rate: 312.5kbps
- ☑ Scan time for 32 devices: ≤25ms
- ☑ Wire type: 16AWG (1.3mm) to 24AWG cat5 (0.5mm)
- ☑ 24VAC or DC power and data on two wires
- ☑ Application power delivered to each device: 100mA@3.3VDC
- ☑ Built-in unique MAC ID
- ☑ Plug-and-play or preconfigured self-installation
- ☑ 20MLF (5mm x 5mm) RoHS-compliant IC package

Discover the technology that millions of devices worldwide already use.
Call 1 - 408 - 938 - 5200 or visit us online at www.echelon.com/easycontrol.

ECHELON®

TASK MANAGER

The 21st-Century Designer

He's a 29-year-old hardware designer tweaking a snippet of code as he waits for a bus in downtown Bangalore. She's a 38-year-old conservationist sitting at a Tokyo sushi bar and editing a hardcopy of a low-power lighting system. He's a 30-year-old, London-based professor listening to Thom Yorke on his MP3 player while soldering a new circuit. He's a 65-year-old retiree fingering through the part bins at a Radio Shack in Palo Alto while his grandson toys with a new satellite radio receiver in the front of the store.

What do all of these seemingly different people have in common? They're all contemporary embedded designers, and they're all part of the *Circuit Cellar* community.

Since the initial development of the silicon chip, the face, habits, and work ethic of the embedded engineer have changed. The community is no longer populated by a select group of middle-class, culturally homogenous, university-trained, predominantly male, electrical engineers. Today, young coders are branching out into the field of embedded design, and retired hardware designers are starting second careers as software gurus who blog about code languages in their spare time. This trend knows no borders. It's happening in North America, Europe, and beyond.

This excites us. Not only does it make for a more diverse magazine that features compelling articles written by an interesting group of designers, but it also bodes well for the future of our increasingly interconnected global community. Can anyone deny that we're on the verge of some major breakthroughs in the fields of microcomputing, wireless communication, and robotics design? Picture thousands of enthusiastic designers from culturally diverse backgrounds developing the groundbreaking technologies of tomorrow. If that's not exciting, I don't know what is.

Nowhere are the skills of contemporary embedded designers displayed more prominently than in our magazine and on our web site. Read through our articles and study the hundreds of designs posted on our various contest sites. If after doing so you come to only one conclusion, it should be this: there's a heck of a lot of high-level designing going on.

Circuit Cellar magazine is always a great place to start your search for information about new projects. This month, Greg Cloutier describes how he a built a handy analog oscilloscope companion (p. 14). The LPC2138-based system ensures the precise calibration of his oscilloscope.

On page 22, Samir Lohani walks you through the process of designing a digital instrument panel for a diesel engine. The system enables you to view essential engine-related data on your PC or any portable Bluetooth-enabled system.

Do you want to know what it takes to build a modular security system? Take a look at the PIC18F4580-based multitiered design developed by the team of Camosun College students on page 32. It's an innovative upgrade for any basic home or office monitoring system.

I know that after reading this issue you'll still be hungry for more, so you should also check out the amazing systems developed by the winners of the Atmel AVR 2006 design contest (www.circuitcellar.com/avr2006/). It's wonderful to see the wide variety of cutting-edge systems from such a diverse group of designers.

Do you have what it takes to design, build, and run a system? If you're looking to put your skills to the test, check out our newest contest sponsored by Luminary Micro (www.circuitcellar.com/designstellaris2006/).

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
Fred Eady
George Martin
Ed Nisley

NEW PRODUCTS EDITOR
John Gorsky

PROJECT EDITORS
Steve Bedford
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 DESIGNER
Mary Turek

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

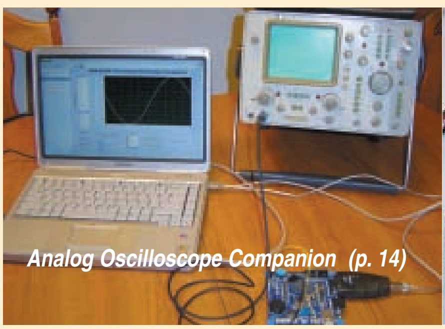
AUTHORS
Authors' e-mail addresses (when available) are included at the end of each article.

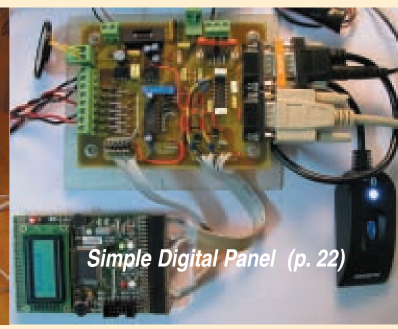
CIRCUIT CELLAR®, THE MAGAZINE FOR COMPUTER APPLICATIONS (ISSN 1528-0608) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Vernon, CT 06066. Periodical rates paid at Vernon, CT and additional offices. **One-year (12 issues) subscription rate USA and possessions $23.95, Canada/Mexico $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.

December 2006: Analog Techniques

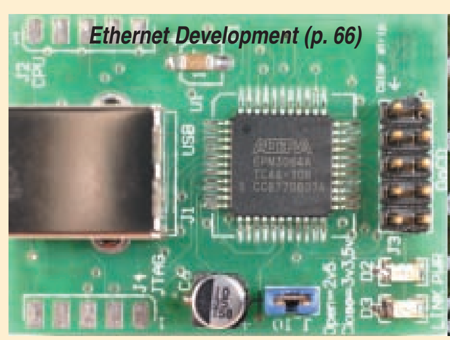
FEATURES

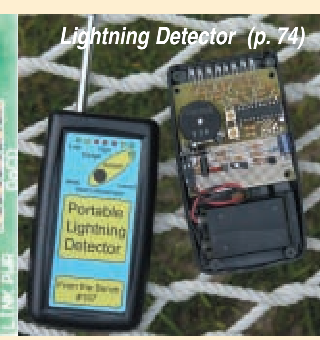

Analog Oscilloscope Companion (p. 14)


Simple Digital Panel (p. 22)


Multitiered Security (p. 32)

COLUMNS


Ethernet Development (p. 66)



Lightning Detector (p. 74)

DEPARTMENTS

AVR picoPower Microcontrollers

To meet the tough requirements to modern microcontrollers Atmel® has now combined ten years of low power research and development into picoPower™ technology for AVR® microcontrollers. picoPower enables AVR to achieve the industry's lowest power consumption with 650 nA with a real time counter running and 100 nA in deep sleep.

What can AVR picoPower do for your design?

- True 1.8V supply voltage enabling operation of all features and core down to 1.8V
- Minimized leakage current enabling 100 nA Power Down sleep consumption
- Sleeping brown-out detector enabling full protection with no power penalty
- Ultra low power 32 kHz crystal oscillator enabling operation at only 650 nA

For more information, check out **www.atmel.com/ad/picopower**

ATMEL®

Everywhere You Are®

NEW PRODUCT NEWS

Edited by John Gorsky

LOW-COST DEVELOPMENT TOOL FOR AVR MCUs

The **AVR Dragon** is a new low-cost development tool for AVR microcontrollers. By providing a complete on-chip debugging and programming tool for more than 30 different AVR flash memory microcontrollers, the AVR Dragon is the industry's most comprehensive low-cost development tool.

The AVR Dragon supports all programming modes for all AVR microcontrollers and on-chip debugging for AVR microcontrollers with up to 32 KB of flash program memory. At just a fraction of the price usually required for such a full-featured tool, the AVR Dragon will fulfill all of your programming and debugging needs. The AVR Dragon can be used with an external target board as well as an onboard prototype area, allowing programming and debugging without any additional hardware.

The AVR Dragon is powered by a USB cable and can also source an external target with up to 300 mA for programming or debugging. If the target is already powered by an external power source, the AVR Dragon will adapt and level convert all signals between the target and the AVR Dragon.

The AVR Dragon uses the free Atmel AVR Studio IDE. AVR Studio—which includes a simulator, assembler, and C compiler—is the software front end for all Atmel AVR debugging and programming tools. A flexible and secure firmware upgrade feature allows AVR Studio to easily upgrade the AVR Dragon to support new devices for free.

The AVR Dragon is available from all of Atmel's franchised distributors at a suggested retail price of **$49**.

Atmel Corp.
www.atmel.com

OP-AMP OFFERS LOW NOISE OVER A WIDE VOLTAGE RANGE

The **AD8599** dual op-amp is designed for medical, instrumentation, automated test equipment, and other industrial applications that require low noise without sacrificing precision. The AD8599 is the industry's only dual op-amp to specify 1-nV/rootHz voltage noise at 1 kHz over a wide operating voltage range up to 36 V. It lowers voltage noise by 80% as compared to standard amplifiers.

The low noise and small footprint make the AD8599 easy to use. It is ideal for closed loop gain, error amplifier configurations and transimpedance applications, such as microphone preamps, buffers for precision DACs, and photodiode amplifiers. The AD8599 comes in an eight-lead SOIC package, reducing board space requirements. It also reduces the cost of external components by including a compensation capacitor for unity gain stability.

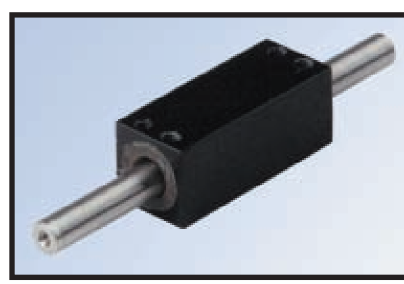
The op-amp is designed using the proprietary iPolar high-voltage process technology, which provides best-in-class precision and gain stability. The device also offers 3.5-pA/rootHz current noise and less than 0.001% distortion, both of which are required to maintain the highest levels of precision for circuits that require high closed-loop gain and stability.

The AD8599 costs **$4.25** in 1,000-piece quantities.

Analog Devices, Inc.
www.analog.com

4-mm LINEAR SERVOMOTOR

The **4-mm Linear Shaft Motor** is Nippon Pulse America's smallest linear servomotor. The 4-mm shaft diameter, a small forcer size (10 mm × 10 mm), a total weight of 9 g, and strokes as long as 40 mm make the motor a suitable replacement in piezo-type applications.



The 4-mm motor is quiet due to the absence of friction because the only mechanical contact section is the linear guide. Its coreless construction totally eliminates cogging. The motor's high motor stiffness allows it to be used in high-precision positioning applications where a resolution of 0.09 nm is achievable. With the Linear Shaft Motor, you will find that you have virtually no fluctuation in speed. Durable construction makes it possible to operate the Linear Shaft Motor in harsh conditions, including a vacuum situation and underwater. Compared to other linear motors on the market, it is compact and lightweight. Due to its design, the Linear Shaft Motor has no backlash.

The S040D-20, which features double coils and a 2-mm stroke, costs **$236.98**.

Nippon Pulse America
www.nipponpulse.com

NEW PRODUCT NEWS

PRECISION SINGLE-CHANNEL DIGITAL FILTER

The new **QF1D512** simple and versatile FIR engine (SavFIRe) precision digital filter enables system designers to quickly and easily add precision digital filtering to an application. The chip can be easily added between an existing ADC and the host controller (microcontroller, microprocessor, digital signal processor, or field programmable gate array), or it can be connected as a coprocessor device for controllers with embedded ADCs. Whereas the previously introduced four-channel QF4A512 implements a complete analog interface, the QF1D512 offers a simple upgrade path for existing systems.

The companion development kit (QF1D512-DK) allows the device to be configured to work with almost any ADC up to 24 bits, and a variety of precision filters can be quickly designed without the need to write code. The development kit includes the latest version of the Quickfilter Pro software. Developed specifically for the QF1D512, this software version uses the same filter design engine as earlier versions and adds a more intuitive user interface for configuring the chip itself. The development hardware allows this configuration to be downloaded into the chip to verify filter performance. The board features an on-board ADC to allow operation "out of the box." It also provides the option for users to add their own preferred ADCs to match their end application. This type of flexibility is further extended to allow the development board to be connected to the user's own host microcontroller, thereby allowing system development to proceed without the need for custom hardware.

The QF1D512 is packaged in a 3 mm × 3 mm QFN package and is characterized over the industrial temperature range. It's priced at **$2.98** in 1,000-piece quantities.

Quickfilter Technologies, Inc.
www.quickfiltertech.com

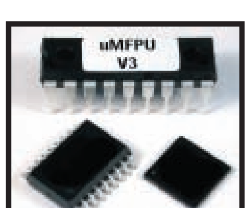
FLOATING-POINT COPROCESSOR

The **uM-FPU V3** floating-point coprocessor chip interfaces to virtually any microcontroller using a SPI or I^{2}C interface, making it ideal for microcontroller applications requiring floating-point math, including sensor readings, robotic control, GPS, data transformations, and other embedded control applications.

The uM-FPU V3 chip supports 32-bit IEEE 754-compatible floating-point and 32-bit integer operations. The new chip is 10 to 20 times faster than previous versions for all instructions and up to 70 times faster for advanced instructions. New instructions provide support for faster data transfer, matrix operations, multiply and accumulate, Fast Fourier Transform operation, unit conversions, and string handling. Two 12-bit A/D channels are provided that can be triggered manually by external input or from a built-in timer. A/D values can be read as raw values or automatically scaled to floating-point values. Local data storage has been expanded to include 128 general-purpose registers, eight temporary registers, 256 EEPROM registers, and a 256-byte instruction pipeline.

An IDE makes it easy to create, debug, and test floating-point code. The IDE code generator takes traditional math expressions and automatically produces uM-FPU V3 code targeted for any one of the many microcontrollers and compilers supported. The IDE also supports code debugging and programming user-defined functions.

The chip is available in an 18-pin DIP, SOIC-18, or QFN-44 package. The single unit price is **$19.95** with volume discounts available.



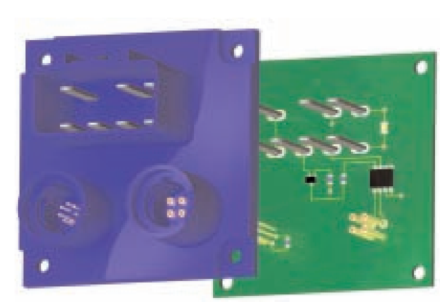
Micromega Corp.
www.micromegacorp.com

CONNECTOR COMBINES PINS, CONTACTS, BLADES, AND COMPONENTS

The **MultiWise Connector** is an industry-advancing connector technology that combines pins, contacts, and blades on a single platform, giving engineers the power to design greater conductivity and functionality into small-footprint devices. As the first significant alternative to discrete pin or blade configurations, the MultiWise Connector provides a new dynamic to satisfy the need for increased power density in small spaces. Configurations using both pins, contacts, and blades allow greater simplicity and flexibility in designs by offering engineers more options for lowering contact resistance while maintaining the necessary power/signal mix.

OEM design engineers once restricted to packaging multiple connectors and circuitry together in an assembly can now combine the required connector features and circuitry together on one MultiWise Connector. With the introduction of blade pins, designers can now take advantage of greater conductivity available in more rugged copper and brass alloy blades. Because these materials carry more current, the MultiWise Connector frees up precious space for additional components. Moreover, the manufacturing processes can place SMT devices like decoupling capacitors, resistors, and ICs directly on the connector.

Prices begin at **$3** in quantities of 1,000 or more for the configuration as shown.



Onanon, Inc.
www.onanon.com

NEW PRODUCT NEWS

FANLESS 1-GHz ULTRA-LOW-POWER INDUSTRIAL SBC

The **EBC-855-G** is an EBX-compatible Intel 1-GHz ultra-low-power Celeron (Dothan core) SBC that operates over an industrial temperature range of –40° to 70°C. Target applications include MIL/COTS, industrial automation, medical/diagnostic equipment, communications, security, test and measurement, and outdoor transaction terminals.

The EBC-855-G is a full-featured SBC with a variety of onboard peripherals that eliminate the need for additional standard I/O peripheral cards. It's based on the Intel's popular 855GME chipset with the ICH4 communications controller and integrated Extreme Graphics 2 video 3-D controller. Both the CPU and system controller are supplied from Intel's Embedded Architecture Division for long-term availability.

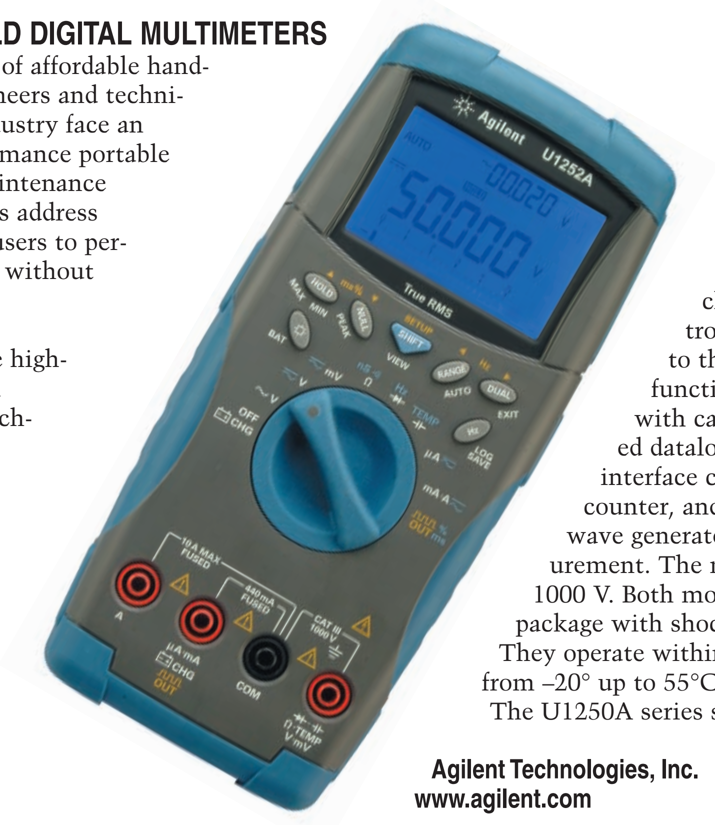
It supports a maximum of 1 GB of industry-standard PC2700 SDRAM, up to 8 GB of CompactFlash, and support for hard and floppy disk drives. The EBC-855-G offers a full set of I/O interfaces including a 10/100BaseT Ethernet port (with remote boot capability). It also offers a VGA and dual-channel LVDS flat panel video, a miniPCI connector that supports an optional 802.11 wireless networking module, four USB 2.0 ports, four serial COM ports, AC97 audio (5.1 codec), LPT, and a PS/2 port for a keyboard and mouse. An onboard, software-programmable, 48-line digital I/O controller provides input, output, or output with read back for each I/O line. The board also is populated with both PC/104 and PC/104-Plus interface connectors for support of additional off-the-shelf or user-designed specialty I/O modules.

The EBC-855-G board costs **$595**.

WinSystems
www.winsystems.com

NEW PRODUCT NEWS

FEATURE-RICH, HAND-HELD DIGITAL MULTIMETERS



The **U1250A** is a new series of affordable hand-held digital multimeters. Engineers and technicians in today's electronics industry face an increasing need for high-performance portable instruments in service and maintenance applications. These new meters address these market needs, allowing users to perform tests in-plant and off-site without compromising performance.

Built to ease electronic troubleshooting and validation, the high-performance, portable U1250A series enables engineers and technicians to complete their day-to-day tasks without being confined to the bench. The initial entry in this series comprises two feature-packed models, the U1251A and the U1252A, which are designed for the cost-sensitive test and measurement market.

The meters provide 4.5-digit resolution with 50,000 count full-scale on dual display and basic accuracy of up to 0.025%, offering simultaneous and accurate measurements and providing the flexibility to perform either quick validation or tolerance checks and marginal failure troubleshooting. In addition to the basic measurement functions, these models come with capabilities such as automated datalogging with the optional PC interface cable, a 20-MHz frequency counter, and a programmable square wave generator and temperature measurement. The meters are rated at CAT III 1000 V. Both models come in a robust package with shock-absorbing over-mold. They operate within their rated specification from –20° up to 55°C.

The U1250A series starts at **$399**.

Agilent Technologies, Inc.
www.agilent.com

NEW PRODUCT NEWS

Visit www.circuitcellar.com/npn for more New Product News.

8-BIT MCUs WITH INTEGRATED ETHERNET

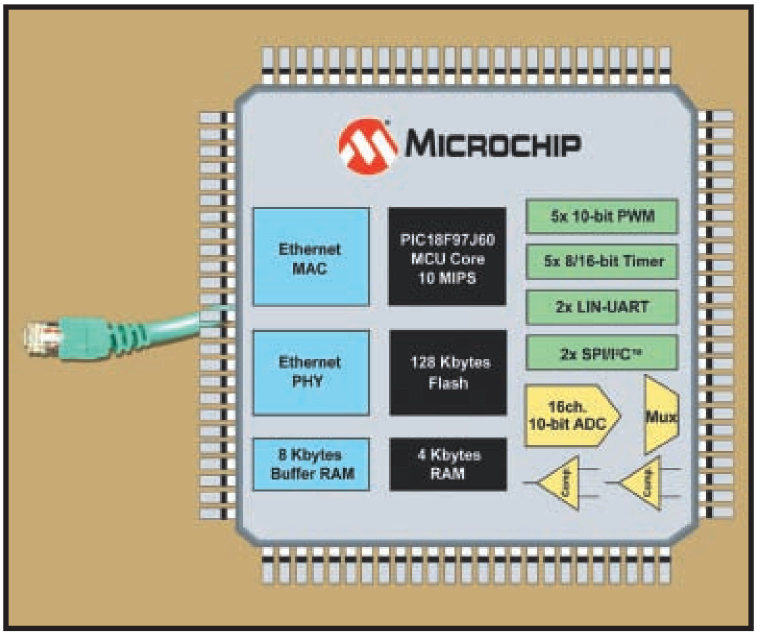
A new family of the world's smallest eight-bit microcontrollers with an integrated IEEE 802.3-compliant Ethernet communications peripheral is now available. The **PIC18F97J60 family** is optimized for embedded applications. It has an on-chip medium access controller (MAC) and physical layer device (PHY). By integrating a 10Base-T Ethernet controller onto a 10-MIPS PIC18 microcontroller with up to 128 KB of flash program memory, Microchip is providing embedded systems designers with a single-chip remote-communication solution for a wide range of applications.

Embedded engineers have long wanted to add remote monitoring and communication to their embedded designs, but previous solutions were expensive and complex. Due to the integration of the PIC18 8-bit microcontroller with a complete Ethernet controller, designers can now have network connectivity in 64- to 100-pin packages that is one of the most cost effective and easy-to-use Ethernet solutions. A free TCP/IP software stack to reduce development time is also available.

Key features for all nine members of the family include seamless migration from previous PIC18 designs, IEEE 802.3 compliance, and a dedicated 8-KB Ethernet buffer. The family also features 128 KB of flash memory and 4 KB of SRAM. This memory accommodates the TCP/IP stack with web server and leaves ample space for user application code.

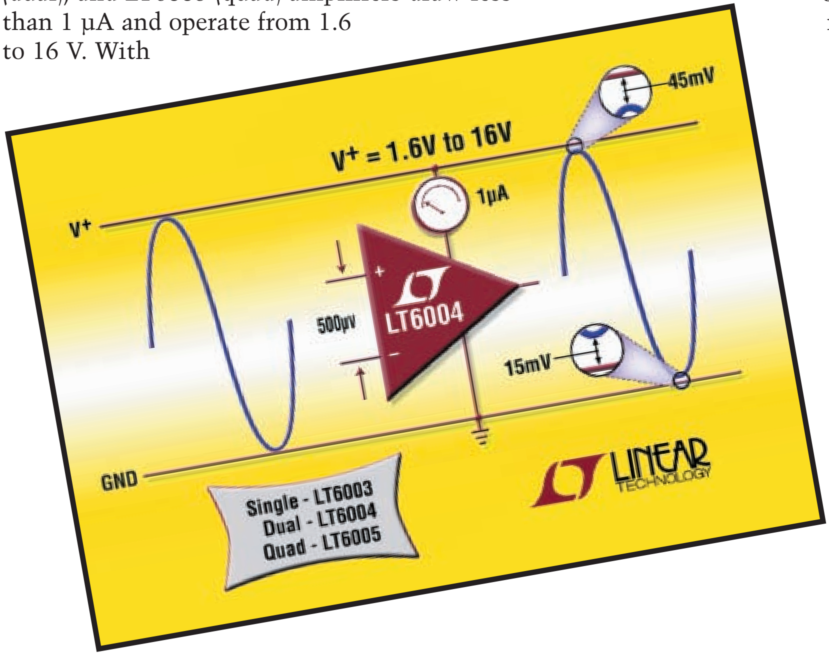
All nine members of the PIC18F97J60 microcontroller family are available starting at **$4.24** each in 10,000-unit quantities.

Microchip Technology, Inc.
www.microchip.com



1.6-V HIGH-PRECISION OP-AMPS

The new **LT6003** family of ultra-low-power op-amps that set a new standard for precision low-voltage operation and tiny footprint is now available. The LT6003 (single), LT6004 (dual), and LT6005 (quad) amplifiers draw less than 1 µA and operate from 1.6 to 16 V. With input offset voltage of 500 µV maximum at 25°C and a maximum drift of 5 µV/°C, the LT6003 family is the most precise low-voltage, ultra-low op-amp. A tiny 2 mm × 2 mm DFN package makes the footprint of the LT6003 the smallest in this class and inputs and outputs feature rail-to-rail operation. The LT6003 family is ideal for battery-operated, hand-held instrumentation due to the combination of low supply current, a 1.6- to 16-V operating range, and tiny packages.



Both leaded and leadless DFN packages are available with fully guaranteed specifications over commercial (0 to 70°C), industrial (–40° to 85°C), and automotive (–40 to 125°C) temperature ranges. The LT6003 family is available now with 1000-piece prices of **$0.72**, **$1.10**, and **$1.75** for the single LT6003, dual LT6004, and quad LT6005, respectively.

Linear Technology Corp.
www.linear.com

How far will your design take you?

Challenge yourself against other top embedded engineers around the world in **DesignStellaris2006**, proudly sponsored by Luminary Micro, Keil, and Circuit Cellar.

Use any microcontroller in Luminary Micro's Stellaris™ family of ARM® Cortex™- M3 controllers with the ARM RealView® Microcontroller Development Kit (MDK-ARM) to create your design contest entry, and see how far your design will take you!

- No purchase necessary to enter.
- $10,000 in cash prizes!
- Entry deadline is February 7, 2007.
- Winners will be announced at the Embedded Systems show... Silicon Valley 2007.
- Submit your design today!

FOR COMPLETE DETAILS, VISIT
www.LuminaryMicro.com/ DesignStellaris2006

DESIGN STELLARIS 2006

LUMINARY MICRO™

KEIL™ An ARM® Company

Cortex Intelligent Processors by ARM

CIRCUIT CELLAR®

www.LuminaryMicro.com/DesignStellaris2006

LUMINARY MICRO LM3S811 EVALUATION KIT

Visit our website for more information!

THE LM3S811 EVALUATION KIT includes the Stellaris LM3S811 Evaluation Board, an evaluation copy of MDK-ARM, USB cable, documentation, and programming examples.

 by Greg Cloutier

ARM Scope

Build an Analog Oscilloscope Companion

Greg's ARM Scope improves upon ordinary analog oscilloscopes that don't provide suitable results at low speeds. The LPC2138-based system features a timed PWM output for the more precise calibration of the oscilloscope as well as an analog input and an analog output with a graphical display. As a result, the displayed signal is more accurate at high or low speeds.

At work, I have access to a host of high-end instrumentation. Having access to such equipment spoiled me after a while, and I felt like something was missing when I was working on projects at my house. Although my digital multimeter worked well for most projects, I really wanted to be able to see signals as well. I couldn't justify spending a lot of money on new test equipment, so it looked like I would need to settle for an older analog oscilloscope. After spending some time searching the Internet for an affordable system, I ended up with the winning bid on an auction site for a Hewlett-Packard HP1742A oscilloscope. Spending $40 on a 100-MHz oscilloscope was worth it for me. The oscilloscope worked out great and has been a welcome addition to the small lab in my house.

Once I had the oscilloscope hooked up, I thought that the void was filled. But a problem with an old analog oscilloscope is that it can't provide much useful information at low speed. The display is not retained, and you are left guessing at what the signal looked like. Another problem is that I am never too sure about the calibration throughout the timebase range. The on-board calibration pin is for voltage, while the frequency is not specified. Even if the frequency were specified, it wouldn't necessarily be useful throughout the full timebase range.

Recognizing these issues, I designed an analog oscilloscope companion to help the system achieve complete functionality (see Photo 1). The ARM Scope does not duplicate what the old analog oscilloscope does well. It simply adds some functionality where it's needed most.

The companion requires three functions to make a nice addition. The first function is a pulse-width modulator (PWM) source that can be used to calibrate or verify the oscilloscope's timebase throughout its entire range. The second function is a low-speed capture and display that enables me to visualize the slower signals that the oscilloscope can't retain. While I was at it, I decided that it would be nice to be able to source an arbitrary analog signal. This can be used to test a filter or drive a sensor. Signal generation is also useful for test and measurement applications.

HARDWARE OVERVIEW

I have played around with many different microcontrollers and development environments. Until I started this project, all my experience with microcontrollers had been with either 8- or 16-bit varieties, and none of the microcontrollers I had used could run fast enough for my desired PWM rate. Most of them didn't have a DAC or

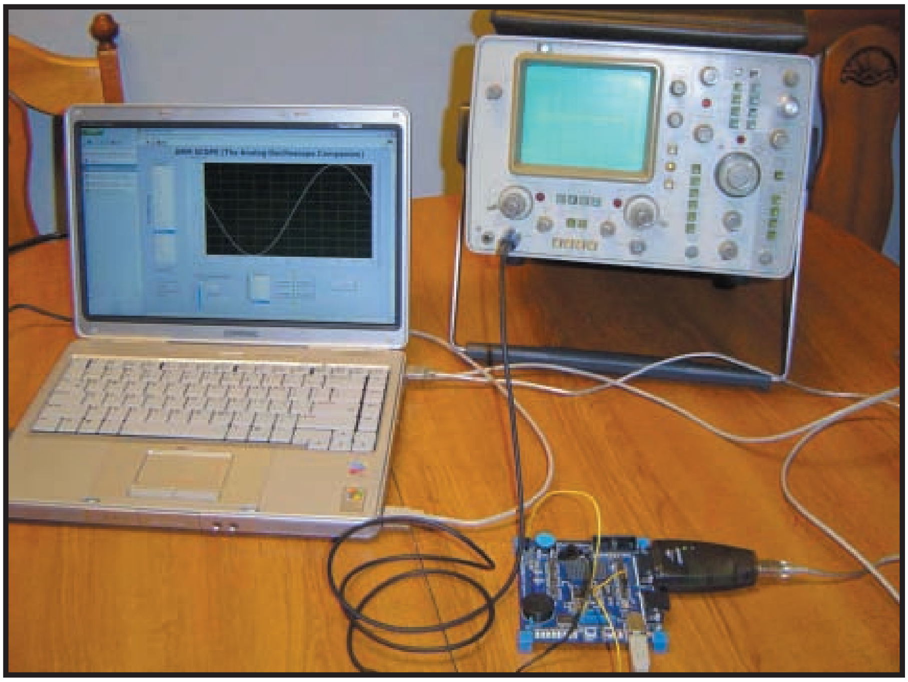


Photo 1—*Based on the user interface's setting, the processor generates a signal that is being read back in and displayed. At the same time, the PWM signal is generated and used to verify my analog oscilloscope timebase.*

enough RAM. I hadn't ever used an ARM processor before, but I tried one in this design.

The companion features a Keil MCB2130 evaluation board, which made it easy to experiment with and explore the capabilities of the NXP (founded by Philips) LPC2138, which is a superset of the LPC213*x* Tiny 32-bit ARM7TDMI-S processor family. This series of processors appears to be extremely popular. Without providing hard data, the family's cost is surprisingly competitive with most 8- or 16-bit offerings. The processor's peripherals matched what I needed for my oscilloscope companion.

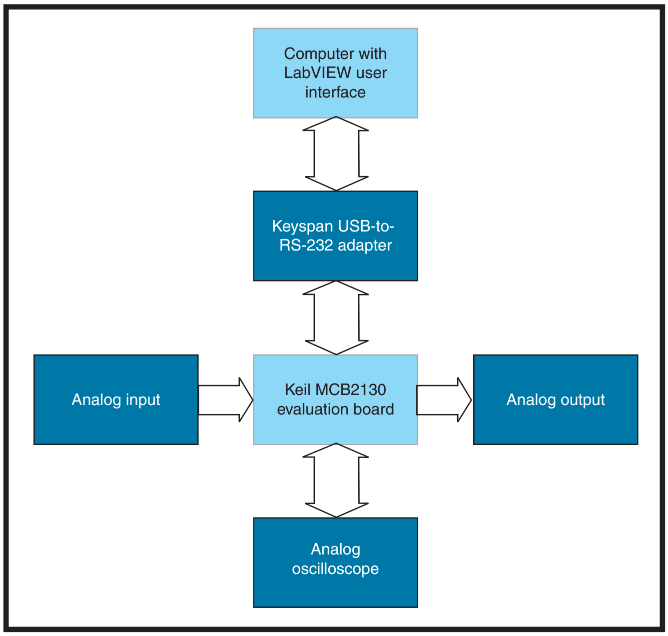


Figure 1—*Take a look at how the equipment is linked together. The Keil evaluation board is at the heart of the system. All of the data and signals flow through it.*

The MCB2130 evaluation board enabled easy access to route signals in and out. I didn't need to create a custom circuit board to accomplish my goals. Many of the evaluation board's features (e.g., push buttons, LEDs, a potentiometer, and a speaker) came in handy while I was learning to program the ARM processor. It also includes a JTAG port, which is useful if you happen to have a JTAG debugger. (I don't have one, so I didn't use the port.) Power is supplied through a USB plug so that your PC sources 5 V to the board. Best of all, there are two fully functional RS-232 ports complete with level converters. One of the RS-232 ports is used to download your program into the ARM processor's flash memory. Both can be used for communicating with a PC. After learning about all of these features, I knew the project would become the ARM Scope (see Figure 1).

Being a superset for the series, 512-KB of flash memory was nice to see, but well beyond what I needed for this application. The included 32 KB of RAM was also more than I required. That being said, have you ever heard of anyone complaining about having too much RAM? Most of the microcontrollers I've used had less total flash memory than this ARM contains for RAM. The evaluation board is clocked by a 12-MHz crystal, which the processor then multiplies up to 60 MHz via its PLL function. This is how the project achieved the speed I desired.

I used the PWM peripheral to output the timebase calibration source, which was my first functional requirement. The PWM is its own peripheral and does not link to (but is based on) the system counter/timers. A programmable 32-bit prescaler can slow down the PWM, which contains its own 32-bit timer/counter. The PWM can drive up to six pins for various uses. I decided that two PWM signals would be a nice fit. One pin is configured to output a pulse that is one division wide on the oscilloscope display. The other pin is configured to output a pulse that is eight divisions wide. This provided enough flexibility to verify my oscilloscope's calibration all the way from 2 s per division to 0.05 µs per division. The period and duty cycle are set on the fly by command. Refer to Table 1 to see how I associated the PWM timing with the oscilloscope timebases.

The Timer0 peripheral is set up to pace the A/D input and the D/A output. As with the PWM, this also contains a 32-bit timer/counter and can be slowed down by a programmable 32-bit prescaler. In this case, however, it's used to time or count events. The events that it counts in my application are clock cycles. I use it to create time delays. Given that a clock rate of 60 MHz will count from zero to 60 million in 1 s, you can calculate what you need to count to for a desired delay. When it matches a value that you provide, an interrupt is triggered. This interrupt is used to pace the system. The Timer0 interrupt can reset the counter to zero and start counting all over again. This way repetitive tasks (e.g., collecting data at a certain rate) occur on their own. Even running at 60 MHz with no prescaler, a 32-bit counter can range from a count of one (about 17 ns) to a count of 4,294,967,296 (about 71.5 s). The time delay is to be set on the fly by command.

The ADC peripheral is used to capture analog voltage signals and convert them into a binary number. All but the lowest-end LPC213x ARM processors contain two 10-bit ADCs. The lower end devices contain just one. This isn't as limiting as it sounds. Each ADC is multiplexed among up to eight input pins. The ADC conversion takes place in 2.44 µs. This means you could gather full 10-bit readings from a pin at a rate of just higher than 400,000 samples per second. You could even go a bit faster if you are willing to accept a lower resolution. This 10-bit result represents a range of 0 to 3.3 V. A voltage of zero will be given a value of zero. A voltage of 3.3 will be given a value of 1,023. Every thing in between is scaled accordingly.

Timer0 is used to pace the system, so each ADC conversion happens within the timer interrupt. I chose to use only the top 8 bits of the result (256 possible values rather than 1,024). I didn't do this because I wanted to increase my collection rate. What I really wanted was to increase the rate at which I could transfer data to my PC. To send an 8-bit result, I can do it in 1 byte. Sending a 10-bit result would require 2 bytes that would slow down the display rate by a factor of two. An 8-bit result looks just fine on my PC's display, especially because

I'm most interested in the shape or appearance of the signal.

The DAC peripheral is used to output an analog voltage by converting a binary number to the desired voltage. Unlike the ADC, the DAC is not multiplexed, and it outputs only a voltage through one pin. All but the lowest-end LPC213*x* ARM processors contain a single 10-bit DAC. The lower-end device doesn't contain a DAC. Given the opposite functionality of the ADC, you supply a value and are rewarded with an analog voltage in 1 µs. You could update this voltage up to 1,000,000 times per second. To output 0 V, you would provide a value of zero. To output 3.3 V, you would provide a value of 1,023.

I am not passing this data continuously across from the PC, so the resolution will remain at its full 10 bits. The analog output signal is contained in a 1,000-byte array (500 data points). Within the Timer0 interrupt, I increment the data array index and send the value to the DAC. Because both the ADC and DAC's functionality are paced by the same timer interrupt, these signals are synchronized. The data buffer, which is filled by command, can contain any arbitrary signal or pattern.

I used the two UART peripherals. UART0 was used to program the on-board flash memory with my program. The LPC213*x* ARM processor series contains an on-board bootloader. At reset, the bootloader looks to UART0 to see if a new program needs to be loaded. This is great because you don't need to buy or build a special device programmer.

On the PC side, once you have created a program, you can use the NXP application dedicated to programming the ARM. The LPC2000 Flash Utility communicates with the bootloader to get things done with no intervention from the developer. The other UART is used for communicating with the PC over an RS-232 interface. This port is used for sending and receiving the analog data. It is also used for receiving and responding to commands from the user interface. The port's speed is configured to send or receive data at 57,600 bps. This data rate is the true limiting factor in my design. Because I send the analog data as I capture it (in the Timer0 interrupt), I don't capture data any faster than it can be sent to the PC. Of course, this is fast enough to meet my functional goals. The UART's input and output both contain a 16-byte hardware buffer. When a byte arrives at the UART, a polling loop calls a function to handle the data.

EMBEDDED CODE

Along with the MCB2130 evaluation board, the kit included Keil's DK-ARM software package. The software package included Keil's µVISION 3 IDE and Keil's ARM compiler. The IDE handles everything including project management, code editing, calling up the flash memory programming application, simulation, and handling the compiler. If I had a JTAG debugger, it would handle that as well.

You have a choice of using Keil's ARM compiler or the included GNU compiler. The Keil compiler is limited to producing 16 KB or less of code (as is the simulator/debugger), while the GNU compiler can produce code of any size (although the simulator/debugger is still limited). I knew my code would fit under the 16 KB limit, so I used the

			Time tick 16.67 ns	Maximum ticks (no prescale) 4,294,967,296	Maximum int (signed) 2,147,483,648	
	PWM timing at 60 MHz					
	Scale (s)	Divisor	Seconds per division	Seconds per sweep	60-MHz Counts per sweep	Refresh rate (Hz)
1	2	1	2	20	1,200,000,000	0.05
2	1	1	1	10	600,000,000	0.1
3	0.5	1	0.5	5	300,000,000	0.2
4	0.2	1	0.2	2	120,000,000	0.5
5	0.1	1	0.1	1	60,000,000	1
6	0.05	1	0.05	0.5	30,000,000	2
7	0.02	1	0.02	0.2	12,000,000	5
8	0.01	1	0.01	0.1	6,000,000	10
	Scale (ms)					
9	5	1,000	0.005	0.05	3,000,000	20
10	2	1,000	0.002	0.02	1,200,000	50
11	1	1,000	0.001	0.01	600,000	100
12	0.5	1,000	0.0005	0.005	300,000	200
13	0.2	1,000	0.0002	0.002	120,000	500
14	0.1	1,000	0.0001	0.001	60,000	1,000
15	0.05	1,000	0.00005	0.0005	30,000	2,000
16	0.02	1,000	0.00002	0.0002	12,000	5,000
17	0.01	1,000	0.00001	0.0001	6,000	10,000
	Scale (µs)					
18	5	1,000,000	0.000005	0.00005	3,000	20,000
19	2	1,000,000	0.000002	0.00002	1,200	50,000
20	1	1,000,000	0.000001	0.00001	600	100,000
21	0.5	1,000,000	0.0000005	0.000005	300	200,000
22	0.2	1,000,000	0.0000002	0.000002	120	500,000
23	0.1	1,000,000	0.0000001	0.000001	60	1,000,000
24	0.05	1,000,000	0.00000005	0.0000005	30	2,000,000

Table 1—*For each timebase on my oscilloscope, I was able to calculate an appropriate PWM value that would be useful for timebase verification.*

Keil compiler. I assumed that it was fully integrated into the IDE and that it would get me up and running quickly. The user-friendly environment worked without fail. It is also nice to know that it can be set up to not only develop code for the LPC213*x* series, but also for nearly every brand and model of ARM processor in production today.

I knew that communicating with my PC would be essential, so I immediately began studying the examples included with the evaluation kit. Samples of UART functionality were found in two of the three examples. Along with the datasheet and NXP's "UART/SPI/I²C Code Examples," I was able to create my routines to communicate with the PC.

The straightforward output routine typically just sends the ADC data to the PC via the UART as the data is acquired within the Timer0 interrupt. The `printf()` command sends the data. I send the 8-bit piece of data and a `\n` character. The application on the PC adds each piece of data it receives to the display. The data sent from the PC and received by the ARM processor is handled in a slightly different way. A typical transfer in this direction usually includes a command and some data or parameters. This was implemented in such a way that the processor doesn't stop while waiting for data to come in.

The main loop is set up to poll the UART for data in a continuous manner. Not much goes on in the main loop, and there is a 16-byte buffer, so it's unlikely that any data will be missed. An alternative would be to set up a UART receive interrupt. Within the polling routine, the received data is transferred into a receive buffer array. This is done because it is not necessary to act on a command until all the data or parameters are received as well. It is determined that the reception is complete when a `\r` character is received. Once the reception is complete, a function is called to process the command. This function determines which command has been sent and then acts accordingly on the data or parameters.

I left the command structure fairly simple because there were not too many commands to implement. Commands must be agreed upon between the processor code and the application running on the PC. In this case, there are eight command represented by the uppercase letters A through H.

Command A sets the PWM period. The output duty cycles are calculated and updated based on this value as well. Command B stops the PWM output. Command C updates the Timer0 counter match limit, which updates the rate at which the ADC data is read in and the DAC data is output. Command D sets a flag that enables the

Listing 1—*This is a sample of what happens in the command handler. First, the command is determined and then the appropriate case handles the data or parameters.*

```
void processCommand(void)
{
  int tempInt;
  char tempChar;
  int readBufferIndex;
  //int aoBufferIndex;
  tempChar = *readBuffer; //get fist char from global read buffer
  tempChar = toupper(tempChar); //ensure upper case for switch statement
  tempInt = atoi(readBuffer + 1); //read integer value if command has one
  switch(tempChar)
  {
     case 'A': //set up pwm values and run the pwm
  {
       PWMTCR = 0x0000002; //counter reset
       PWMMR0 = tempInt; //set period
       PWMMR2 = (tempInt / 10); //set PWM2 (P0.7) duty to 10%
       PWMMR5 = ((tempInt / 10) * 8); //set PWM5 (P0.21) duty to 80%
       PWMTCR = 0x00000009; //Enable counter and PWM
       IOSET1 |= 0x00020000; //P1.17 LED ON (PWM enabled)
       break;
     }
     case 'B': //disable pwm
     {
       PWMTCR = 0x0000002; //counter reset (disable PWM outputs)
       IOCLR1 |= 0x00020000; //P1.17 LED OFF (PWM disabled)
       break;
     }
```

Listing 2—*This is the entire interrupt routine. It's really the heartbeat of the program. It performs all the ADC and DAC operations while pacing the program.*

```
void tc0 (void) __irq
{
  unsigned int val;

  if(aiFlag == AI_ENABLE) //capture and send analog input value if
                          //flag is set
  {
    val = ai();
    printf("%c\n",val);
  }

  if(aoFlag == AI_ENABLE) //output and increment analog output
                          //buffer if flag is set
  {
     DACR = aoBuffer[aoBufferIndex];
     aoBufferIndex++;
     if(aoBufferIndex >= 500)
       aoBufferIndex = 0;
  }

    T0IR        = 1; // Clear interrupt flag
    VICVectAddr = 0;
}
```

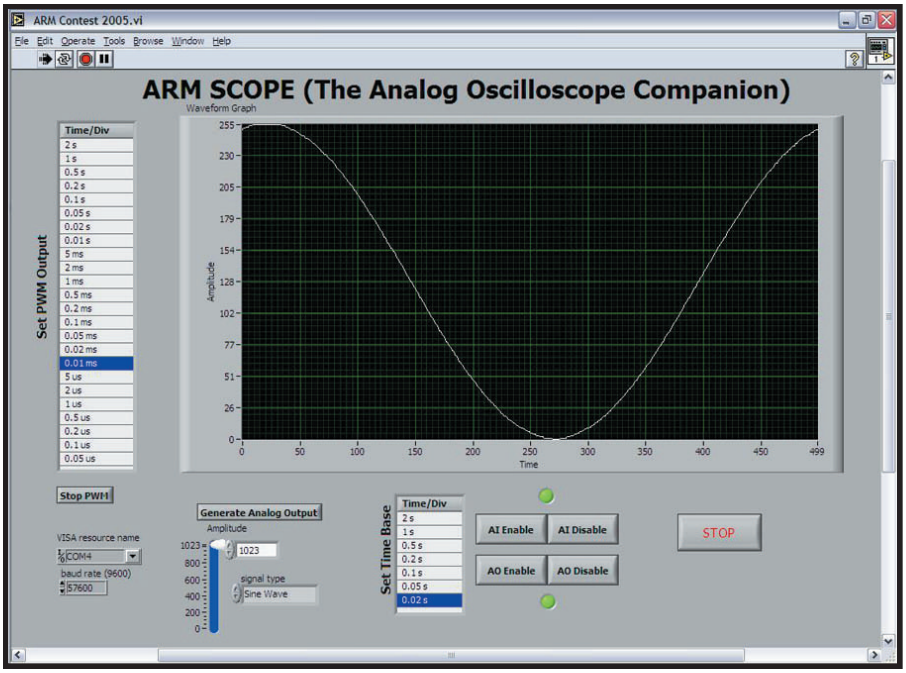


Photo 2—*Take a look at the ARM Scope's GUI. The virtual oscilloscope displays the incoming analog data and enables point-and-click control over the operation of the processor.*

ADC operation within the Timer0 interrupt. Command E clears a flag that disables the ADC operation within the Timer0 interrupt.

Command F fills the analog output data buffer with 500 values. These values are sent as ASCII text representation that gets converted into 10-bit binary representation for storage in the data array. Command G sets a flag that enables the DAC operation within the Timer0 interrupt.

Command H clears a flag that disables the DAC operation within the Timer0 interrupt. The LED indicators on the evaluation board are also set and cleared to provide operational status. More commands could easily be appended to the list as needed. A sample of the command handler is shown in Listing 1.

The `main()` function starts by calling all set-up and initialization functions. These include the UART, PWM, DAC, ADC, and Timer0. It also lights an LED that let's me know when the processor has initialized and is ready for operation. Following this, the main loop does nothing but poll for data arriving via the UART and calls the command processing function if the command is ready to be processed.

A nice thing about the Keil µVISION 3 IDE is the start-up configuration utility. Rather than sort through the datasheet and set up multiple registers, the configuration utility enables you just check or uncheck boxes and type in values as needed. I used this utility to set up the PLL to provide the 60-MHz clock rate.

Most of the work is done within the Timer0 interrupt. Some designers would argue that the interrupt routine should just set a flag and get back to the main routine as quickly as possible. There is not much going on in the main loop, so I felt that running the code in the interrupt would be fine. Maybe next time I'll do it the other way just for the sake of functional upgrades over time. The ADC operation comes first. If the flag that allows this operation is set, the voltage on the pin is read and then sent to the PC via the UART. The data is appended with a `\n` termination character so that the receiving application knows that the data has been sent.

The DAC operation comes next. If the flag that allows this operation is set, the analog data array index is incremented to the next piece of data, which is sent through the DAC and then out the pin as a voltage. Listing 2 shows what the interrupt routine looks like.

USER INTERFACE

I developed the user interface with

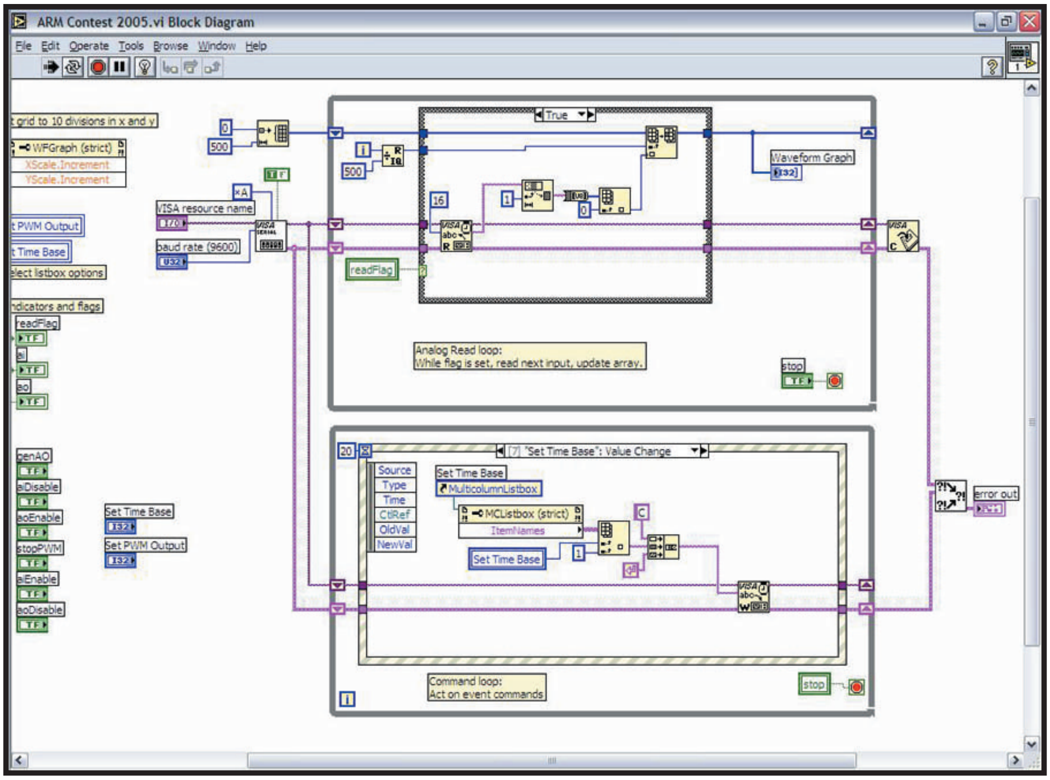


Photo 3—*This is where all the graphical code is developed. Note that there are two main sections of code that run in separate threads.*

National Instruments's LabVIEW (see Photo 2). LabVIEW is a graphical programming language meant for engineers to easily acquire and display data. Unlike a text-based language, code is created using graphical virtual instruments (VIs), which are the equivalent of functions. You can create a function and then save it as an icon. Then multiple functions can be used by a larger function, again represented as an icon. Hieratical code is easy to build, and it's easy to reuse functions.

LabVIEW is a data flow language, which means the flow of data has full control over the order of events. Each of the inputs and outputs of a VI are represented as terminals on the icon. To pass data from VI to VI, you must draw wires from the output terminal of one to the input terminal of another. Built-in functions are found on a graphical palette rather than within `.h` files. Functions are logically grouped and are as easy as drag and drop. Code can be executed immediately. If there is something that will prevent it from running, the environment is very clear as to where the problem lies.

The view used to create the program code is called the block diagram. The view used to create the user interface is called the front panel. To create the user interface, all of the controls, indicators, and graphs are also logically grouped in their own palette so you can drag and drop them. When you drop a control on the user interface, it gets a representative icon on the block diagram as a source of data. When you drop an indicator on the user interface, it gets a representative icon on the block diagram as a sink for data.

In the block diagram, data is represented by wires similar to those you'd see in a schematic capture package (see Photo 3). Arrays of data appear as thicker wires. As dimensions are added to the array, the wires appear even thicker. Also, like in the C language where structures are used to create custom data types that contain multiple types of data, LabVIEW uses something called a cluster. For loops and while loops are just drawn around the code that you want to iterate on. Local data is sent around the loop via shift registers. LabVIEW handles alternate code execution by way of case structures. For every anticipated case (and a default), you draw code within the structure. Debugging is also very easy. LabVIEW lets you watch data flow through the wires in slow motion. You can also set break points and create probes (watch windows).

The other thing that LabVIEW is famous for is its connectivity to hardware and other applications. National Instruments sells hardware such as data acquisition devices with prebuilt VIs to access the functionality with little or no programming. Many equipment vendors such as Agilent provide LabVIEW drivers for their products. LabVIEW can also handle ActiveX and .NET code, Internet protocols, and much more with ease.

The hardware that interested me was my RS-232 serial port. I needed only four functions to handle the serial port in Windows XP: Open, Read, Write, and Close. It just doesn't get any easier than that. LabVIEW also produces a multithreaded application by simply drawing separate sections of code on the same block diagram. It can also handle some event-driven programming that responds to button clicks on the front panel.

The user interface was created to look like an oscilloscope screen. The drag-and-drop graph is set up to display 500 points at a time. Clicking in a display list sets the timebase for the oscilloscope. The analog inputs and outputs are enabled or disabled by clicking on appropriately labeled buttons. The amplitude of the analog output is set by a slider control, while the shape of the wave is selected from a dropdown box. The PWM output is set by clicking in a list that displays all of the available times bases on my analog oscilloscope.

FUNCTIONAL CODE

I developed two separate pieces of functional code for the AVR Scope. Both attach to the same serial port for talking to the ARM processor.

PIN	Function
Pin 7	PWM Output (duty cycle = one oscilloscope division)
Pin 21	PWM Output (duty cycle = eight oscilloscope divisions)
Pin 28	Analog input (0 to 3.3 V)
Pin 25	Analog output (0 to 3.3 V)
COM0	Used to program the device via built-in bootloader
COM1	Set to communicate with computer at 57,600 bps

Table 2—*Instead of providing a schematic for the evaluation board, I decided to show you the pins used for inputs and outputs. Remember that the signals need to be scaled appropriately outside the scope of this project.*

The first piece of code continually collects the analog data and displays it on the graph. The second piece of code is linked to everything you can click on the front panel. When you click a button or list, the event handler sends an appropriate command along with its data or parameters to the ARM processor. Each of these functions runs in its own thread. The event handler thread remains asleep most of the time until a button is pressed. The analog receive and display loop is paced by the incoming data. It will sit and wait for data to be received at the serial port.

SYSTEM DEVELOPMENT

The ARM Scope system met all three of my goals. It can output a PWM signal for verifying my old analog oscilloscope's timebase through its entire range. It can capture and let me visualize analog signals at a slower rate than my oscilloscope can. And it can output any analog waveform that can be generated and sent to the data buffer. I set up my graph display so that it had 10 divisions just like a real oscilloscope. As it is, the timebase for the analog input and output functions can be set to a range from 2 s per division to 0.02 s per division.

I'm considering a few upgrades for the code. Of course, both the processor code and the user interface code need to be updated as a matched set if functionality is added. I know that the ADC is capable of reading much faster. I can see filling a buffer up quickly and sending the data for display at a much more relaxed rate. For that matter, collecting extra data could be useful for triggering a signal or zooming in on the data. I would have to decide if this would be done by the user interface or in the processor code. LabVIEW also has some powerful built-in math and DSP functions, so I could analyze the data in various ways.

Note that I did not include any analog circuits. This project was meant to get analog data into and out of a processor. The way I scale the signal from a sensor to match the 0 to 3.3 V range would be another project altogether. There was no one-size-fits-all solution for this, so I left it to be application specific. If you add some control circuitry, there will be plenty of pins left over and the processor will have plenty of horsepower left.

By the way, I didn't even make a dent in the amount of flash memory. I stayed well under the limits of the compiler as well, so there is plenty of room to grow the code as needed. The pins I used for inputs and outputs are shown in Table 2. ■

Greg Cloutier (greg.cloutier@cox.net) works as a test and measurement engineer for JDS Uniphase Corp., which develops and manufactures fiber optic communications components. He first became interested in microcontrollers at work and now works with them in his spare time.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2006/197.

RESOURCE

NXP Semiconductors (founded by Philips), "UART/SPI/I^{2}C Code Examples," rev. 01, AN10369, 2005.

SOURCES

MCB2130 Evaluation board
Keil
www.keil.com

LPC2138 Microcontroller
NXP Semiconductors (founded by Philips)
www.nxp.com

by Samir Lohani

Digital Instrument Panel

A Simple Panel for Diesel Engines

With Samir's digital panel for diesel engines, you can view critical engine-related data on a PC, HMI console, or Bluetooth-enabled portable device. You can use the panel as a stand-alone device or as a diesel engine's preprocessor in a PLC-based system.

I recently designed a system to serve as a "black box" that interfaces with a diesel engine's pressure, temperature, and RPM sensors. The design generates serial packet data for a digital instrument panel that can be viewed on a PC, on a human machine interface (HMI) console, or via a Bluetooth link on a mobile device.

I designed the system for medium-capacity diesel engines on railroad vehicles and gensets (environments subjected to heavy switching where EMI/EMS noise considerations are important). It's also been used to simplify the design of PLC-controlled test beds for engines. The system can interface with existing industrial-grade sensors fitted on an engine, which may be nonlinear. It also includes intuitive alarm annunciations that are based on recorded messages. You can use this system on its own or as a diesel engine preprocessor in a PLC-based system.

SMALL BLACK BOX

I have often wondered why the instrument panels for industrial diesel engines, gensets, and medium-power machinery are so monotonous and dull. Contrast such panels with the colors, chrome rings, and rich accents on modern automobile dashboards! Aesthetics aside, maintenance engineers must be able to keep a record of an engine's parameters and monitor its performance.

I took a close look at the conventional engine panels while I was setting up a PLC-controlled test bed for diesel engines in a maintenance unit. Interfacing the existing analog instrument panel would have enabled the PLC to detect abnormal conditions in advance without operator intervention. However, fitting PLC-compatible sensors and instrumentation would have been a nuisance. I wanted a black box for the existing instrumentation.

I decided to build a separate unit to connect to the PLC via a serial link in order to handle all of the interfacing requirements for diesel engine sensors. A parallel Bluetooth link was included for wireless connectivity.

I used a Renesas Technology M16C/62P SKP microcontroller board in view of the superior EMI/EMS specifications demanded by these environments. The generous amount of flash memory and its in-built DAC prompted me to include an audio interface for intuitive alarm annunciations. This offers interesting possibilities for conveying alarm messages through a common VHF channel on walkie-talkies.

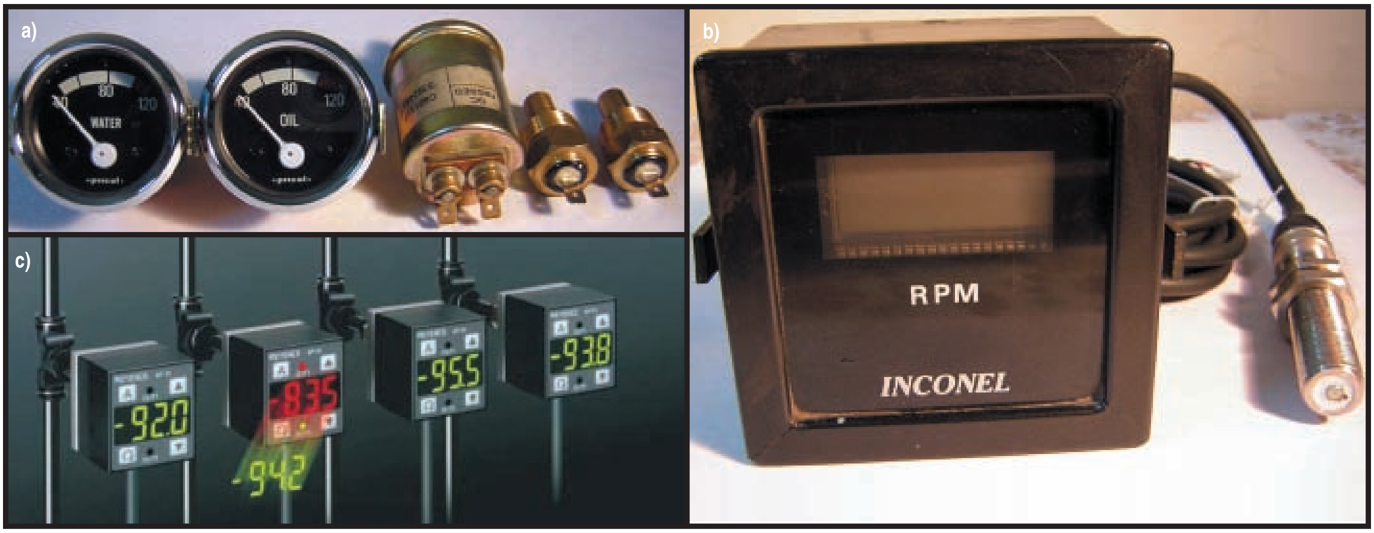


Photo 1—*I'm interested in several gauges and sensors: analog gauges with pressure and temperature senders* ***(a)****, an RPM gauge with a magnetic pick-up sensor* ***(b)****, and multipurpose digital pressure gauges with 5-V analog output* ***(c)****.*

Now the operator is no longer constrained to stay close (or within limited Bluetooth range) to the instrumentation.

The data sent by the unit consists of ASCII packets, which can be processed by the PLC or viewed on a PC or a mobile device. Incorporating the sensor interfacing and processing functions in a single unit also simplifies the ladder programming and improves the scan response of the PLC. The end result is a black box that's small enough to fit inside existing instrument panels for a well deserved and painless instrumentation upgrade.

GAUGES & SENSORS

Industrial diesel engines typically use electrical resistive sensors called senders to sense fluid temperatures and pressures. These are used along with corresponding analog electrical gauges so information can be displayed on a panel. To sense engine speed (RPM), magnetic pick-up sensors are mounted close to the crankshaft gear teeth. Revolutions of the crankshaft gear cause the change of magnetic flux across gear teeth. This in turn causes an AC voltage to be generated by the sensor whose frequency is directly proportional to engine RPM. Analog or digital RPM gauges provide the readout.

Conventional sensors are robust and designed to work in an engine's operating environment. The magnetic pick-up sensors also work reliably in the oil-splashing environment of the crankcase. However, senders are nonlinear and designed to be used only with matching panel meters. The output voltage of magnetic pick-up sensors also varies with the rate of change of magnetic flux (i.e., RPM) and their air gap with the gear teeth.

Multipurpose digital pressure gauges (e.g., the Keyence AP-30 series) mark a new development in pneumatic pressure sensors (see Photo 1). These gauges provide a direct digital readout of positive or negative air pressure, which can be very useful for determining, say, clogging in a filter element or the turbocharger boost pressure. For remote sensing, they also provide a 1- to 5-V analog output. I found them to be extremely useful and reliable. Their analog output also makes interfacing them a breeze!

CIRCUITRY

I designed the circuit for the system as an extension to the M16C/62P SKP microcontroller board. As you can see in Figure 1, it is divided into several functional blocks. The circuitry is shown in Figure 2 (p. 24).

The UART0 and UART2 interfaces provide two independent serial ports for connections to a PLC/PC and a serial Bluetooth dongle, respectively. I provided a 5-V supply for the Bluetooth dongle port. The Bluetooth dongle port is wired as DTE, whereas the PC/PLC port is wired as DCE.

To measure rotational speed, signal-shaping stages are provided for magnetic pick-up RPM sensors. These work over a wide range of sensor input voltages from 0.5 V to beyond 50 V. These stages have been designed with high HF noise rejection to ensure that clean digital signals are generated for feeding to TimerA input lines.

The analog gauges are read via the microcontroller's A/D port lines through voltage divider networks. HF bypass capacitors at signal inputs shunt noise transients. Schottky diode clamps provide protection against overvoltage/reversed connections. Channel AN2 is used for measuring the supply voltage (12/24 V) required for ratiometric measurements. Channel AN3 is configured as a 0- to 5-V input to directly connect to digital pressure gauges with 5-V analog output. The four remaining channels (AN4–AN7) are for measuring voltages across analog gauges. (The measuring span is kept at 16 V to cover all common P/T gauge-sensor pairs in 24-V systems.)

The D/A section is used to implement the speech interface. Output DA1 is connected to a low-pass second-order active Butterworth filter for antialiasing. The components are selected for passing speech while adequately rejecting the sampling frequency of 11.025 kHz and its harmonics. The resulting output is fed to an audio amplifier designed around an LM386 IC. The audio output can be connected to a walkie-talkie VHF unit's microphone input (through an attenuator) for communicating alarm messages. Most VHF units that support hands-free headsets will then automatically switch to Transmit mode through their internal voice-activated circuitry.

The power supply has been designed to operate from 9 to 40 VDC so that the circuit can be powered from both 12- and 24-V systems (automotive /industrial supply). Note that 6.5 V is used for the audio subsection, whereas a 5-V digital logic supply is for the M16C/62P SKP microcontroller board and logic circuits. Adequate heat sinking must be provided for the LM317 regulator. You can use an SMPS converter in place of the LM317 linear regulator for better conversion efficiency.

If your circuit is going to be used in an automotive or electrically noisy environment, you must observe EMI/EMS design issues. The electronics must withstand load-dump conditions, surges that result from the switching of inductive loads, and other disturbances on the electrical lines. Although it greatly helps to use the M16C/62P microcontroller with superior EMI/EMS specifications, suppressing noise and its coupling to other stages is best carried out close to the input itself during board design.

Design the PCB with a generous ground plane, good power-supply decoupling, and low-impedance noise bypass capacitors where the signals enter the

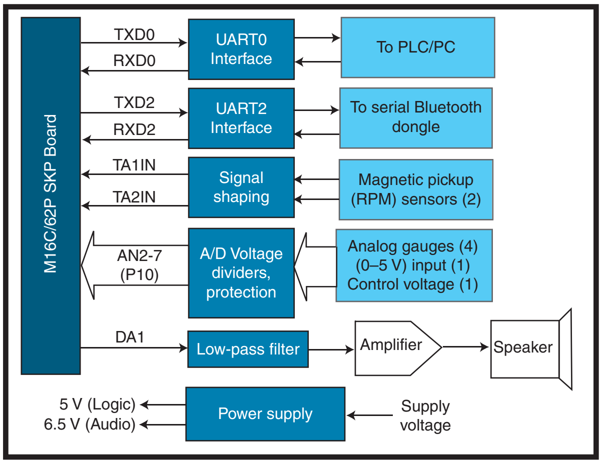


Figure 1—*I neatly broke up the system into distinct blocks.*

PCB. I used shielded cables for connecting to the sensors and a balanced transient suppression filter at the power supply input. I also used Linear Technology's LTSpice/SwCAD III simulator to optimize the critical stages for high noise immunity (see Photo 2). I have successfully used the circuit interfaced to a PLC-based monitoring system in a railroad environment, where switching currents of several hundred amperes are common!

CONSTRUCTION

Building the prototype was fairly

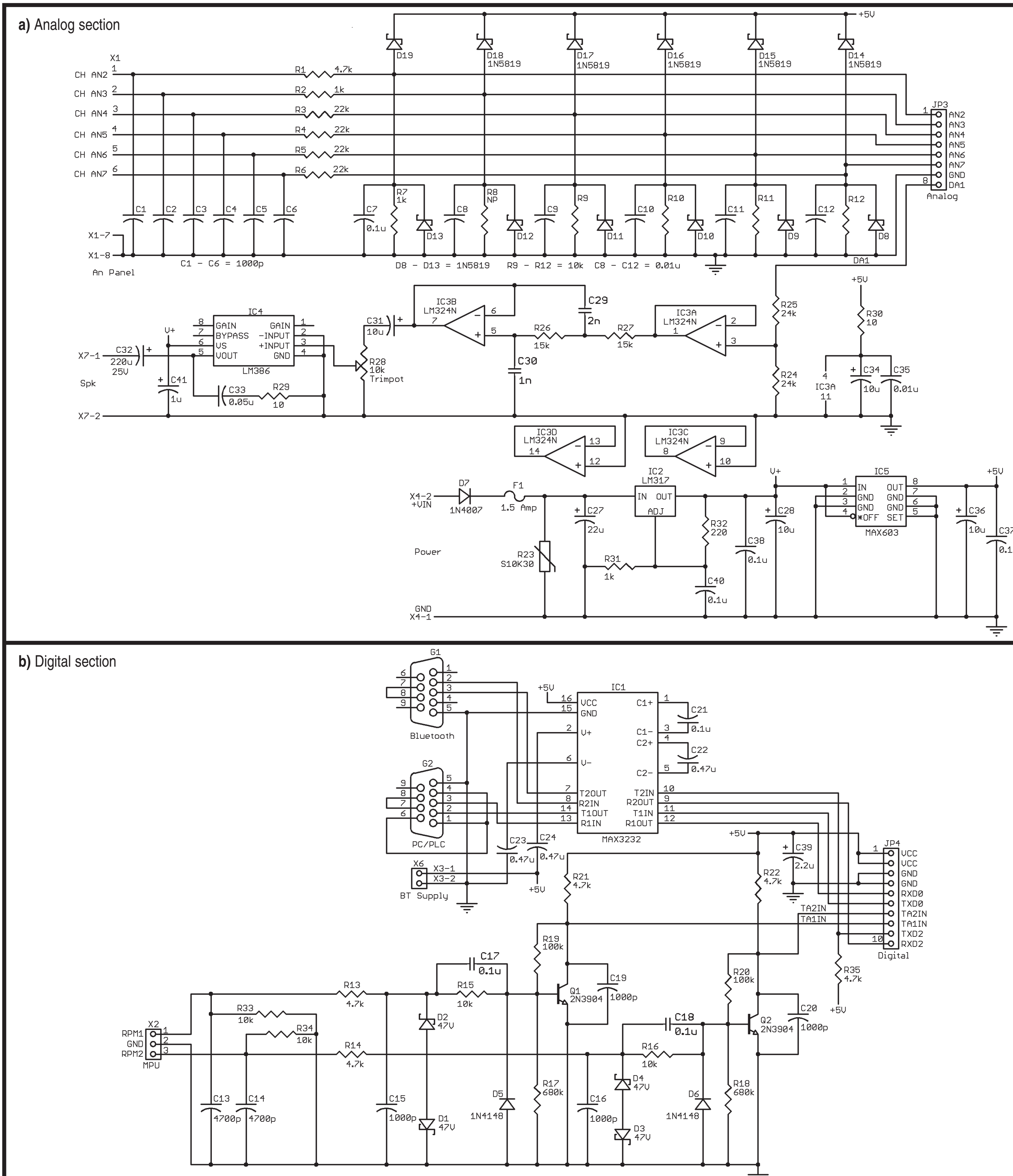


Figure 2a—*The analog subsections comprise voltage divider networks, a low-pass active filter/amplifier, and a power supply.* **b—***The digital subsections comprise two independent serial ports and signal-shaping stages for magnetic pickup (RPM) sensors.*

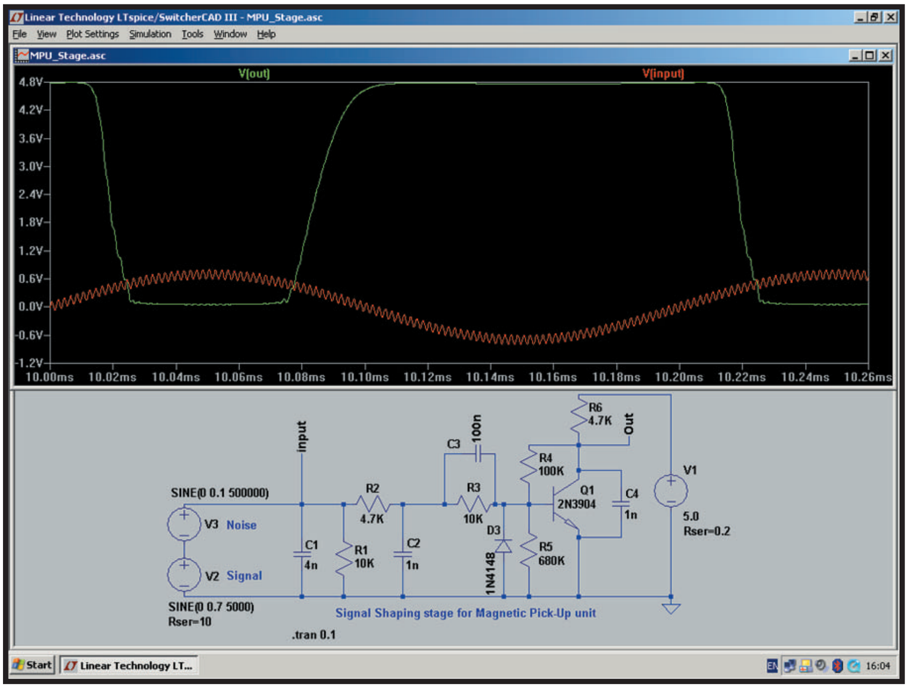


Photo 2—*I tested for noise rejection. The output signal of the magnetic pick-up signal-shaping stage remains clean despite an input signal of only 0.7-V_{PEAK} with superimposed noise.*

simple (see Photo 3, p. 26). I made the connection to the SKP board through ribbon cables soldered to a 2 × 25 pin header. I used a Kingene BT210S serial Bluetooth dongle for wireless connectivity. This dongle provides the SPP Bluetooth profile. Its Bluetooth radio is rated for Class 2, with a communication range up to 25 m. The power requirements are 5 V at 150 mA. I chose the dongle for its simplicity. The defaults for its communication parameters are factory set, and it doesn't require any configuration commands. Pressing the Blue button on the dongle enables nearby Bluetooth devices to perform device and service discovery and pair up with it. A single multicolor LED indicates its status. Once paired, it appears to the microcontroller as a simple serial device.

MCU SOFTWARE

The software for the M16C/62P microcontroller was written in C language so it could be compiled with the Renesas HEW IDE. The program has two main sections: the sensor acquisition mode loop and the calibration mode loop. In the former, a number of tasks are performed sequentially. First, the engine's speed is measured. Two channels for simultaneous measurement are provided (RPM1 and RPM2).

The $TA1_{IN}$ and $TA2_{IN}$ input lines that receive the processed speed-sensor signals are configured in Event Counter mode. A 100-µs clock-tick accumulator (`ticks_100us`) controls the gating period. The timer interrupt of TA0 updates this accumulator every 100 µs. Following this, the analog measurements are carried out. The ADC operates in Single Sweep mode. Other than the two channels used internally in the M16C/62P SKP board, the six remaining analog channels of P10 are used to measure external analog signals.

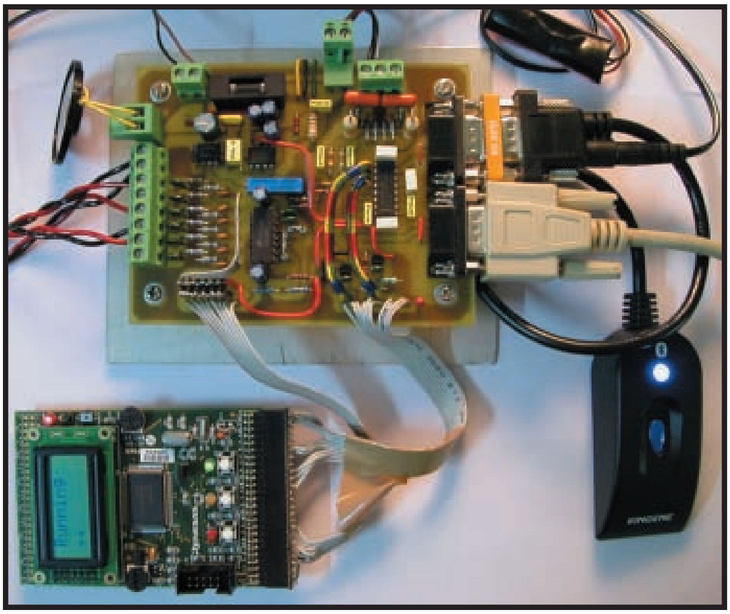


Photo 3—*The Bluetooth link is used to connect to a laptop via the Kingene BT210S serial Bluetooth dongle. The second serial link hooks up to a PLC. The sensors are connected through the screw terminal blocks. Also note the small speaker for playing WAV files at 11.025 kHz.*

In automotive and railroad environments, the nominal battery voltage may vary considerably depending on the battery's state of charge and the switching of heavy loads. The electrical gauges therefore measure a reference current in addition to the current through the sensor. The industrial pressure/temperature gauges have three pins that connect to Supply, Ground, and Sensor, respectively. The readings shown by the gauge are a function of the sensor voltage and the supply voltage (i.e., a ratiometric reading). Thus, the program processes the ratio of voltage across the sensor to the supply voltage to compensate for variations in the supply voltage. This is then converted to the gauge reading using a transfer function. The program can cater to any sensor whose transfer function can be represented by a mathematical equation. The transfer function for an existing sensor can be determined using Calibration mode and a spreadsheet program.

The RPM values and the calculated gauge readings are then converted to

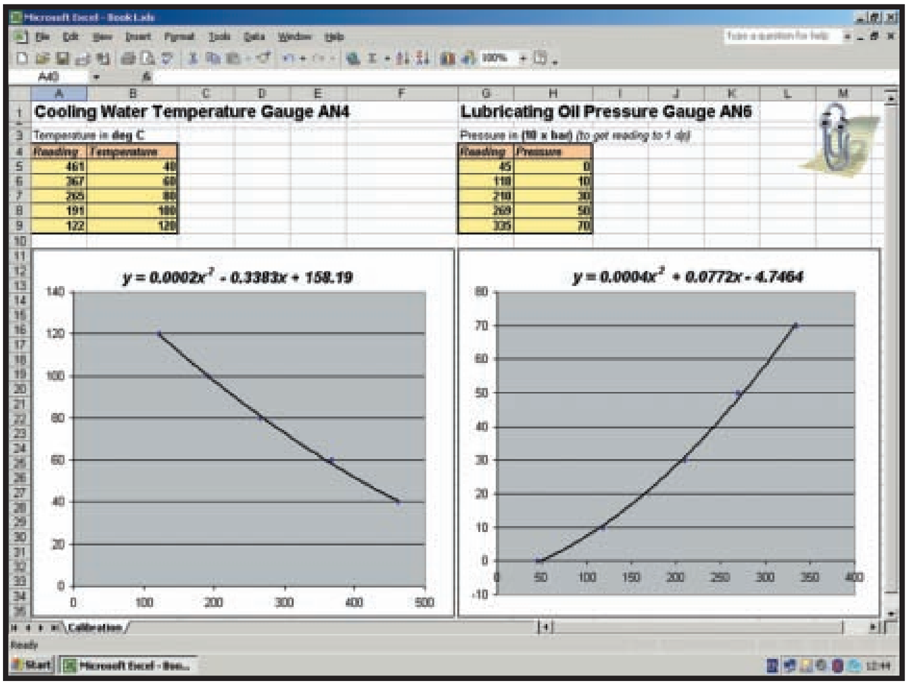


Photo 4—*I used Microsoft Excel to determine trend-line equations for calibration.*

an ASCII data packet. The data packet is a fixed-length string containing ASCII data for the two RPM channels (0000–9999) and the six analog channels (000–999) in the following order: <RPM1><RPM2><AN4><AN5><AN6><AN7><AN3><AN2>. The data packet is sent through UART0 and UART2 delimited by STX and ETX characters. This makes it fairly easy to pass on the data to any PLC capable of receiving block text data in its data memories. A BREAK signal is also sent on UART0 before the first data packet to initialize the communication buffer and data rate setting of the PLC. This protocol was used to interface with a Keyence KV-16 PLC. Small changes may be required for a PLC having a proprietary protocol. The specifics are usually detailed in the PLC's manual.

Finally, the resources in the M16C/62P make it fairly easy to implement alarm annunciations based on recorded speech messages. This is implemented by the `wavplay()` function in `main()`. For the prototype, I kept the programming simple for minimal speech feedback. If any alarm condition has been recorded in queue, the yellow LED on the M16C/62P SKP board lights up. If you press the corresponding S2 switch, the system plays back the necessary sequence of WAV files to build the error annunciation message. Pressing the S3 switch clears the error queue.

The program enters Calibration mode when you press and hold S1. In the Calibration mode loop, the value of the analog channels read is displayed in decimal form. Pressing S2 cycles through the channels AN2 through AN7. The absolute values are displayed for voltage measurement channels AN2 and AN3. Ratiometric values are displayed for channels AN4 through AN7, which are used to connect to gauges. You can exit Calibration mode by pressing the RESET switch.

SPEECH SUPPORT

The simplest speech files to deal with are in WAV format, and they consist of uncompressed audio data after a header block. To use WAV files in the HEW environment, I concluded (after an extensive search on the Internet) that it would be best to include the WAV file audio data as a data array of constants in a separate C source file. In a 2002 Freescale Semiconductor application note entitled "Audio Reproduction on HCS12 Microcontrollers," Grant More describes using this approach for HCS12 microcontrollers. The note is accompanied by essential software that includes a DOS executable file for conversion.

To ensure the clear and correct pronunciation of words, I downloaded the required WAV files from Merriam-Webster's web site (www.m-w.com). I then opened the files in Windows Sound Recorder and resampled in the format PCM 11025 Hz, 8-bit, Mono so that the Freescale utility could process them. Freescale's utility neatly extracted the audio data producing C data array files. I edited the C source files created by renaming the array and removing the `#pragma` directive from the top of the file. In this way, I included 12 WAV files (in C source format) in the source code.

I was very impressed with the quality of the WAV files played back by the M16C/62P even though all of the files had an 11.025-kHz sampling frequency. This helps in giving a very professional user interface to the project. The M16C/62P's large flash memory enables many more files to be stored for playback.

Listing 1—*These are transfer functions of the channels pasted in the code.*

```
switch (scanloop)
{
  case 0:
  // Process Gauge across AN4
  // Cooling Water Temperature gauge
  ratio_accu = ad4_value * 1023.0;
  ratio_val = (unsigned int)(ratio_accu / ad2_value);
  gauge_read = (unsigned int)(0.0002*ratio_val*ratio_val -
    0.3383*ratio_val + 158.19);
  // Suppress Error readings if no sensor is connected
  if (gauge_read > 150)             // obviously absurd!
    gauge_read = 0;
  if (gauge_read >= LIM_WATERTEMP)
    err_watertemp = TRUE;
  break;

  case 2:
  // Process Gauge across AN6
  // Lubricating Oil Pressure gauge
  ratio_accu = ad6_value * 1023.0;
  ratio_val = (unsigned int)(ratio_accu / ad2_value);
  eq_value = (0.0004*ratio_val*ratio_val + 0.0772*ratio_val - 4.7464);
  // This equation can yield negative value with no sensor
  // Check and then assign value
  if (eq_value > 0)
    gauge_read = (unsigned int)(eq_value);
  else
    gauge_read = 0;
  if (gauge_read >= LIM_HIGHOILPR)
    err_highoilpr = TRUE;
  // Lubricant pressure is checked once engine has at-least idle RPM
  if (tim_rpm >= IDLE_RPM) {
    if (gauge_read <= LIM_LOWOILPR)
      err_lowoilpr  = TRUE;
  }
  break;
  // code for other case conditions has been removed in this snippet
}
```

CALIBRATING THE UNIT

In comparison to existing analog gauges (whose needles can get stuck and are also affected by vibration), the readings for a properly calibrated system can be relied upon for repeatability and accuracy. A calibration procedure must be followed once to calibrate the analog inputs. The analog channels must be connected to the required gauge inputs. The system voltage (12 or 24 V) must also be connected. There is no need to disturb the existing analog panel circuitry. Just "tap" the required signals from the panel!

As an example, assume that the temperature gauge channel is to be calibrated. The temperature sensor must be kept in an oil bath and heated while remaining electrically connected to the system. At different bath temperatures (measure with an accurate mercury thermometer), the readings shown on the M16C/62P SKP's LCD are noted. (The system must be in Calibration mode.) The readings and corresponding temperature are used to plot a graph in a spreadsheet program such as Microsoft Excel (see Photo 4. Select the Display Trend-Line option and an appropriate curve fit such as second-degree polynomial. Excel will then display the relation between temperature and reading as a mathematical equation. This equation is the transfer function for the channel that should be pasted in the source code (see Listing 1).

PC USER INTERFACE

The PC user interface was written in Visual Basic. The normal Visual Basic runtime files must be present on

Photo 5—*Take a look at the PC user interface. Several of the gauges shown in Photo 1 are recreated here as virtual gauges in an intuitive instrument panel.*

the PC or can be downloaded from Microsoft's web site. The program uses a free third-party lightweight LED control (akLED). The control files must be in the current directory. Alternately, run the akLED.exe control installation program. The executable and source code files are posted on the *Circuit Cellar* FTP site.

The program displays the readings in analog type meters, which are easy to read. It works with serial cable or a Bluetooth SPP emulation port. The program also has data-logging capabilities and writes the data log to the `DIPlog.csv` file in the current directory. You can open this file directly in Microsoft Excel for analysis.

Even without hardware, the Visual Basic terminal program can be run in Local Simulation mode. Feel free to experiment with the scrollbars and data logging capabilities to get used to the user interface. The display is optimized for 1,024 × 768 mode.

The meter faces may be replaced with images of meters present in the existing analog panel to give a realistic feel. I chose the meters accordingly and edited the digital camera photographs of the meters in Adobe Photoshop to remove the needle. Then I smoothened out the images using the Gaussian blur transformation and converted the images to GIF files with a transparent background. I have also included these GIF files so that you can create similar files for other meter faces. The resulting interface is shown in Photo 5.

GOING MOBILE

Bluetooth connectivity greatly extends the range of devices that can

Photo 6—*The user interface on my Sony Ericsson W800i phone is Bluetooth-enabled.*

provide the user interface for the digital instrument panel. I was particularly excited about using my new Sony Ericsson W800i mobile phone for this purpose. In fact, any mobile phone that has Bluetooth and J2ME support (MIDP 2.0) can be programmed with a Java MIDlet to provide the functionality of the user interface while retaining its normal functions. MIDP 2.0 also includes a rich API set for high-resolution graphics, multimedia, connectivity via SMS, MMS, or the 'Net, and access to the phone's memory stick and SD card. You're limited only by your imagination when producing an intelligent, sophisticated display.

Bluetooth consumes a fraction of energy as compared to Wi-Fi (IEEE 802.11 b/g) while having a respectable data rate, high immunity to interference, and a typical working range of approximately 10 m. This makes it ideal for battery-powered mobile applications. On the software side, the standard SPP profile provides serial port emulation. This simplifies the software considerably. As far as the circuit hardware is concerned, I kept everything simple by using a Bluetooth dongle, which appears to the circuit as a serial port device while its internal hardware and firmware encapsulates the Bluetooth stack and radio implementation.

To demonstrate the concept, I wrote a small Java MIDlet. A minimal MIDlet is required for performing Bluetooth device and service discovery, reading and parsing the ASCII packet data received from its SPP peer, and implementing a Canvas to display the panel data. Java programmers can then add other bells and whistles such as data logging, SMS/MMS support (and, of course, graphical panel meters in a similar way to what I described for the Visual Basic application).

Sony Ericsson's 2005 article entitled "Connecting to a GPS receiver using Bluetooth (JSR 82)" includes bundled Java source code for such an application. This was what I was looking for. It includes code for discovering and connecting to a Bluetooth SPP device, and then parsing and displaying the data. However, it is customized for a GPS unit that generates NMEA data packets. In my case, the fixed length ASCII delimited data packets are much easier to parse. The J2ME source consists of five well-organized Java modules. It is only necessary to rewrite two of these modules: the one responsible for data parsing (GPSReader.java) and the one that provides a canvas to display the acquired data (GPSCanvas.java). It took me just a few hours to write my customized modules. I compiled the source to Java binaries using Sun's free

Java Wireless Toolkit and then downloaded it to my mobile phone. The program worked flawlessly!

APPLICATION DEVELOPMENT

The Java source code for the modules that I wrote, as well as instructions for compiling the code, are available on the *Circuit Cellar* FTP site. These should help you develop a working J2ME application on your mobile phone for personal use. Note that Sony Ericsson's software license prohibits me from reproducing part of the source code that's available on its web site (so as to make a complete project) or to include the Java binaries.

The Bluetooth mobile phone interface implemented in this project provides a new generation of sleek and intelligent displays to any microcontroller with a UART (see Photo 6). Working with a 24-bit color LCD and the connectivity resources in my mobile phone, I wondered why I had been spending money on graphical LCDs and keypads for my projects when I could have achieved better results by running a small Java MIDlet on my mobile phone. Welcome to the world of embedded programming on mobile phones! ■

Samir Lohani (slohani90182@yahoo.com) is a mechanical engineer working for Indian Railways. Control systems and shop floor automation are among his areas of interest. In his spare time, Samir works on electronics and embedded programming projects.

PROJECT FILES

To download code and additional files, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2006/197.

RESOURCES

G. M. More, "Audio Reproduction on HCS12 Microcontrollers," AN2250/D, Freescale Semiconductors, 2002.

Sony Ericsson, "Connecting to a GPS receiver using Bluetooth (JSR 82)," 2005, http://developer.sonyericsson.com. (Refer to the "Java ME—Tips, Tricks, and Code" section.)

SOURCES

akLED Control
akTools
http://www.rentmaster.co.nz/aktools/akled.htm

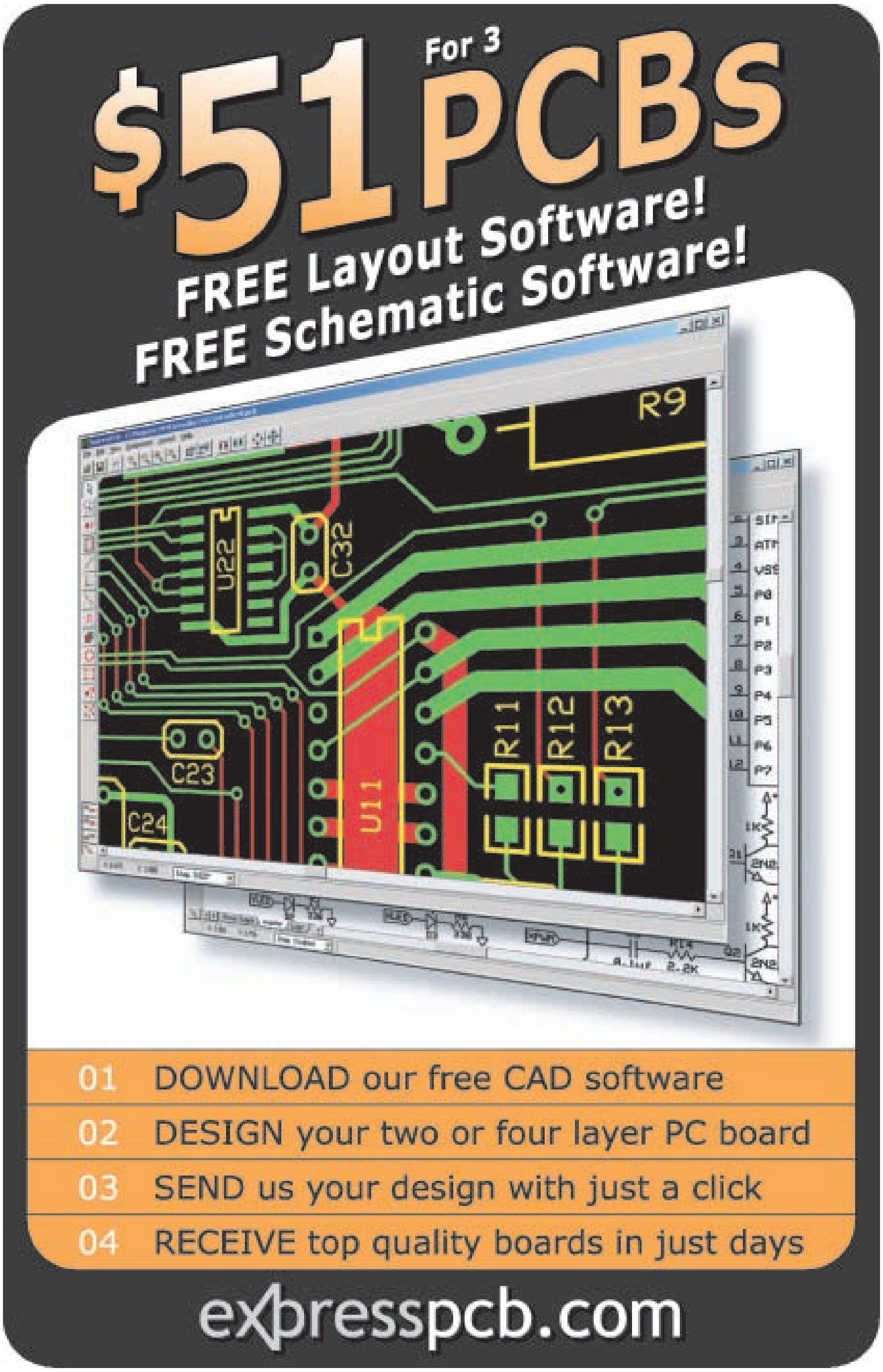
AP-30 Self-contained pressure sensors
Keyence Corp.
www.keyence.com

BT210S Serial Bluetooth dongle
Kingene Technology Corp.
www.csr.com/comps/kingene.htm

LTSpice/SwCAD III Simulator
Linear Technology, Inc.
www.linear.com

M16C/62P Microcontroller
Renesas Technology Corp.
www.renesas.com

Java Wireless Toolkit
Sun Microsystems, Inc.
http://java.sun.com/javame/downloads/



FEATURE ARTICLE by Kyle Cameron, Jeremy Warren, Shane Reierson, Chris Rittaler, & Nick Brady

Multitiered Security System

Use this multitiered PIC18F4580-based security system to bump up the security in your home or office. You can switch the system back to normal mode with only a few clicks of your mouse.

Advancements in modern technology demand expandability. Take your PC for example. Anyone with any kind of computer knowledge would be more inclined to buy a PC with empty slots over a similar system that was jammed full.

Now, when that graphics card comes out using PCI Express, you can pop it right in there. When Windows Vista comes out, you can run it no problem! Need more speed? Throw in some more RAM. OK, we got a little excited there. What we're trying to say is you will have this PC a lot longer, you'll spend less money, and you won't have to buy an entirely new system every time you want to add an option.

We designed a modular (hardware and firmware) security system that offers comparable freedom (see Photo 1). In this article, we'll describe how we did it.

SYSTEM OVERVIEW

Our security system incorporates a "multitiered" design that includes modules with radio frequency identification (RFID), fingerprint authentication, and voice recognized password authentication. Wow, that was a mouthful. To put it simply: you swipe your RFID card (or keychain, finger ring, or even an implant in your hand!), and you're identified using a 4-byte (hex) code. After the system identifies you, it can either authenticate your identity using a fingerprint or a voice-spoken password or just simply grant you access. Every time you are granted access, you will be logged in a database located on the central server.

The central server is the brains of the operation. In its database, it contains all of the 4-byte codes used to identify users and optionally the users' fingerprint information (called a template) and voice passwords. Communicating through an Ethernet controller, the server provides all of the modules connected to it with the information needed to determine whether you should be granted access. At the same time, it provides a responsive GUI to interact with the server administrator.

Once you're granted access at a given access point, you will be allowed to enter (a door is unlocked, an alarm is disabled, or some kind of other lock is disabled). This is where some of the customizable functionality comes in. We set aside digital I/O for the purpose of driving a solenoid or electronic lock that can be chosen to fit specific applications. In a demonstration at Camosun College, for example, we used a pulse-modulated solenoid that pulled a bar out of a locking mechanism when a user was granted access. The bar was pushed back in once the user moved through the door.

In the following sections, we'll explain how our system is interfaced to the PC and how it's set up for expandability. We'll also provide you with information about some very popular new technologies and show you how easy they are to add to your design.

MODULAR DESIGN

Each individual module consists of two to four PCBs (called submodules)

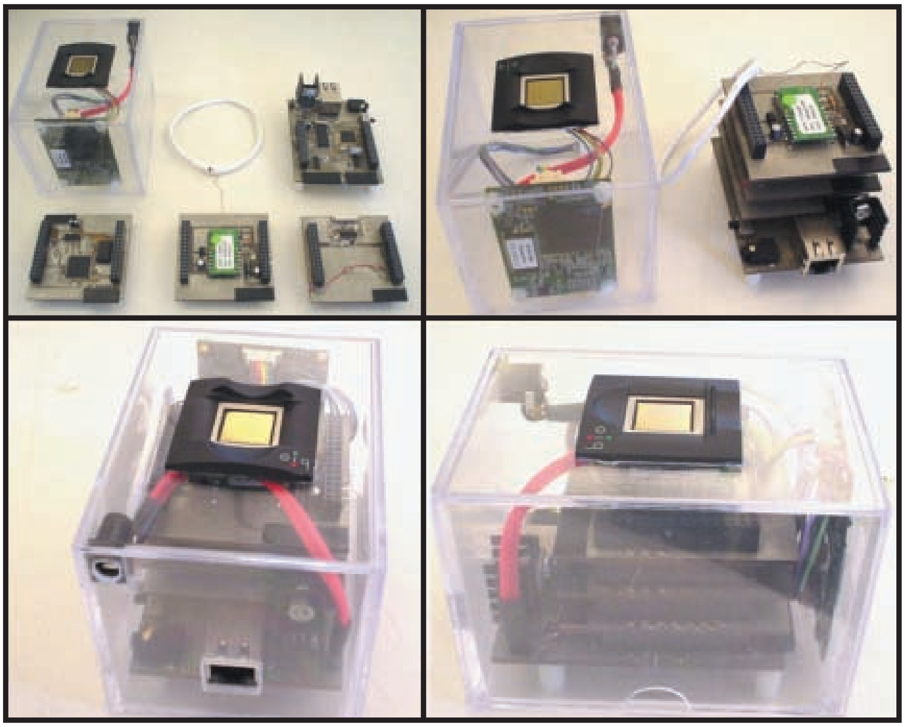


Photo 1—*Take a look at our complete display module. Looking at the top left image, the voice recognition board is on the bottom left. The RFID receiver board with the antenna attached is in the middle. The fingerprint board is on the bottom right. The Bioscrypt fingerprint module is shown inside our display case. The main board is on the top right.*

that all connect together to form a complete system. These PCB boards include a main board, an RFID board, a fingerprint board, and a voice recognition board. Every embedded module contains at least a main board and an RFID board and optionally a fingerprint and voice recognition board (see Photo 1).

Now, why not just have all of this on one board? The most important reasons are expandability and easy maintenance. Think of the PC analogy we used earlier. Wouldn't you like a security system that will enable you to add new, more secure technologies to in the future? Furthermore, would you rather replace a main board or the entire module if something were to go wrong?

This is a very important concept in any design. Just like all of our teachers here at Camosun College have taught us time and time again: keep your designs modular! Doing so makes it easy to add new options in the future, reuse designs, troubleshoot problems quickly, and replace failed parts at a minimal cost! Shall we go on?

MAIN BOARD

Our main board includes a Microchip Technology PIC18F4580 microcontroller, a Microship ENC28J60 Ethernet controller, and connectors for extra I/O and a programming cable (see Figure 1, p. 35). The PIC18F4580 controls communication to all of the peripherals (the Ethernet controller, the fingerprint sensor, the RFID Module, and the voice recognition board) and grants or denies access based on the security level.

The ENC28J60 Ethernet controller is a handy little chip with a SPI, onboard CRC checking, and various receive filters to kick out all of those annoying broadcasts such as ARPs and ICMPs generated by a PC, as well as any packets that aren't destined for the MAC of a particular module. The SPI greatly reduces the I/O needed for such a chip and simplifies its implementation. All we needed to do was send it commands using the Microchip C18 compiler's SPI functions (through four wires and power and ground) and we were up and running. We were sending packets after only a few days of playing around!

Photo 2—*We used an eight-pin header to make a programming cable that included an RJ-11 connector for in-circuit programming and a DE9 connector for access to the USART on the PIC18F4580.*

For the sometimes feared task of programming our microcontroller after it was on the PCB, we composed a simple cable that implemented an RJ11 connector and a DE9 connector (see Photo 2). Doing this enabled us to program a bootloader onto the PIC18F4580 (using an in-circuit debugger) and then simply use the DE9 connector to bootload our code.

FINGERPRINT BOARD

As you may already know, fingerprint sensing and authentication is not exactly an easy task. It takes very complicated algorithms and lots of processing power. We had only three months to design and build this system, so we decided to go with something that could perform all of the sensing and authenticating on its own. However, if you take the time to try and find something like this that can be implemented into an embedded system, good luck!

After several days of researching, emailing, and making numerous long-distance phone calls, we came across Bioscrypt, an Ontario-based provider of finger scan technology. Write this down: Bioscrypt offers complete embedded systems that incorporate Texas Instruments DSPs and can talk to your microcontroller through a simple RS-232 interface!

We used a Bioscrypt MV1210 fingerprint module in our system. All we needed for our fingerprint submodule was an RS-232 level translation chip and a connector into which we could plug the sensor module. How easy was that?

RFID BOARD

Radio frequency identification is becoming more and more popular. You probably saw the guy in the news that had a tag implanted in his hand! The best part is that it's so cool and so easy to implement into your design!

IB Technology's MicroRWD transceiver works with tags in a number of different forms. This receiver module has an extremely simple serial interface, and development kits are available for around $220! Using this development kit, we had RFID working in our project in about 4 h! Great choice!

Our RFID board consisted of power supply filtering and a spot to insert the RFID receiver. That's it!

VOICE RECOGNITION BOARD

To add to our project's cool factor, we included voice-recognizable password technology instead of an old-fashioned keypad. To do this, we needed some form of speech recognition to convert the spoken password into a number that we could use in our code.

We were already dealing with Microchip parts, so we decided to check out what was offered in this field and found exactly what we were looking for. We decided to go with Microchip's speech recognition firmware for use on the dsPIC series of microcontrollers. More specifically, we used the dsPIC30F6014A digital signal controller for our system because a development kit featuring this chip was available at Camosun College where we study.

Like the fingerprint module, we made the choice to dedicate the dsPIC30F6014A to voice recognition and interface it with the PIC18F4580 microcontroller, thus saving our microcontroller's precious RAM and processing power for future developments. Once the code was developed and working, we simply designed a PCB to host the dsPIC and audio codec, thereby creating the voice recognition board.

STATE MACHINE

We've warmed you up on the hardware, but now you're probably wondering how we control all of this. Remember when we said that you should make your designs modular? Modularity is essential when it comes to coding, especially when you're working with a team of designers. How else are you going to divide up the work? How will you (or someone

else) add code later on down the road? Think about this before you start typing!

Let's take a look at the code for each implementation file. Doing so should help you understand how we broke everything up.

Our firmware consisted of a main state machine used to control all of the different events in the embedded module. We used external files that were included in our main file for each peripheral. Our main state machine simply had to call functions from other files to tell the Ethernet controller to send packets or to request verifications from the submodules. As you can see in Figure 2 (p. 36), the state machine just sits around and waits for either a command from the server or an RFID tag to enter the field. Once something happens, it determines how to deal with that event and then goes into a state that takes action. This way any addition to the system will simply require an implementation file (with the necessary functions) to make it work and a new state to take action.

Compared to using `if` and `else` statements, state machines are very good examples of modular code when it comes to a control system. This is because they are easier to understand, and they facilitate code additions almost seamlessly (if coded properly).

We divided up the work. One person built the state machine. Other team members worked with a peripheral until it functioned properly and then submitted their code to the state machine builder for implementation.

FINGERPRINT FIRMWARE

Most of the complex code for the fingerprint module was already done for us in the MV1210, so implementing this thing was rather simple. The implementation file consisted of a series of functions to serially talk to the fingerprint board according to the serial protocol discussed in the documents provided with the SDK.

We were able to send the fingerprint board commands to enroll and transfer

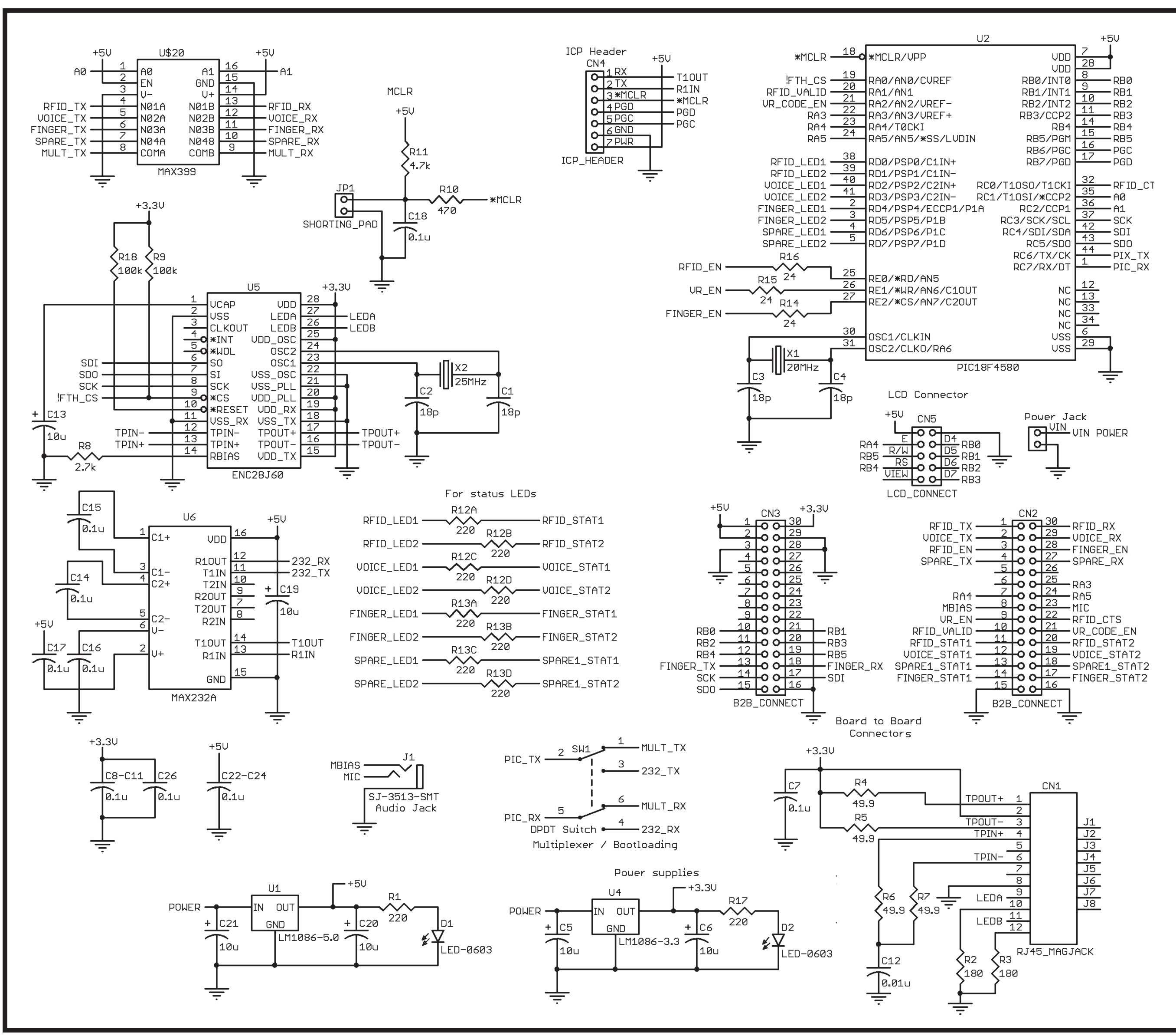


Figure 1—*The main board's power supplies are at the top. The Ethernet connectivity is shown on the left side. The main processor and I/O connections are on the right.*

and to verify users. This enables us to scan a user's finger and save the template in a buffer on the PIC18F4580 (enroll and transfer). This template can then be sent to the server in a single UDP packet and saved in the database.

Once we had a template to work with, we used the verify command to ask the MV1210 to scan the finger and then compare the scanned image with the template we provided. This hardly even touched the MV1210's capabilities, but it did exactly what we needed it to do. The main functions used by our state machine included these two functions and a function to ask the fingerprint board if it was ready to talk (or if it was busy).

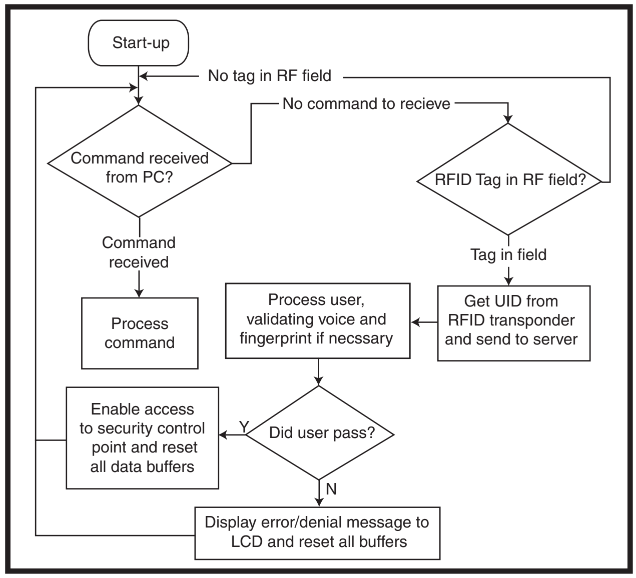


Figure 2—*An RFID tag in the field starts the authentication process. At the same time, the server can send commands to lock down and reset the embedded module.*

RFID FIRMWARE

Probably the simplest of all implementation files, the RFID modules we used required wieldy 1-byte commands. Our implementation file included one function used by the state machine to get the ID from the tag.

The RFID receiver module indicates that there is an RFID tag in the field by pulling the CTS line low. Our main state machine polls this pin to see if there is a tag in the field and waits for a packet from the server at the same time. If there is a tag present in the field, the `GetUID()` function is called. `GetUID()` sends a command to the RFID receiver requesting it to send back the contents of its memory location containing the unique ID from the RFID tag in the field. The state machine then deals with this ID by sending it to the server and requesting the information associated with the user who owns that 4-byte number.

VOICE RECOGNITION FIRMWARE

The dsPIC30F6014A had to be programmed with our customized firmware, so we needed to write the program and get it working before we could make an implementation file. The code we developed for use with the dsPIC30F6014A was formed using a word library built with Microchip's word library builder (and some code to interface the dsPIC30F6014A to the PIC18F4580).

Communication from the PIC18F4580 to the dsPIC30F6014A was achieved using a simple protocol that we developed. The PIC18F4580 first acquires the 4-byte password from the server, sends it to the dsPIC30F6014A, and then pulls an ENABLE line low. As soon as the dsPIC30F6014A sees the ENABLE line go low, it attempts to acquire the 4-byte voice password from you. If successful, the dsPIC30F6014A compares these bytes with the ones received from the PIC18F4580 and then returns a pass or a failure. If unsuccessful, the dsPIC30F6014A will time out and send an error byte back to the PIC18F4580.

All we needed in terms of an implementation file was a function to request a voice verification from the dsPIC30F6014A with the password to compare with as a parameter. This function then returned pass or failure based on the result of the verification.

SOFTWARE

By now, you've heard a lot about the server and what it does. But how exactly does it handle so many things at once without crashing? To say the least, it's a UDP-sending-and-receiving, database-creating security machine! Using Microsoft Visual C# (on the .NET framework) in all its glory, we were able to create this machine which represents the backbone of our security system. To do this, we needed a method of providing responsive interaction with the user and at the same time reliable communication with every module connected to it.

INTRO TO THREADING

A daunting task for any embedded developer is creating a GUI that can interact with the embedded portion of a design. When developing a GUI to support your embedded application, it is important that your communication does not crash your GUI (or you're operating system). A handy way to do this is to use multiple threads.

Multitasking systems use time slicing to appear to run more than one process at a time. Today's processors are fast enough that any human (with the exception of the bionic man) can't tell. This gives you a powerful tool for building a responsive GUI that can maintain communication with your embedded devices.

Most operating systems offer development frameworks supported by SDKs, thousands of classes, and a strong online support network. Microsoft's newest framework (.NET 2.0) is an example.

The first step in developing a multithreaded GUI is to define a front end and a back end. The simplest way to do this is to make all user interaction the front end and everything else the back end. If you think of these two ends as your threads, you could say that your back end thread listens for information from your embedded application, while your front end thread interacts with the user.

You're probably thinking: "OK, I see the genius behind these words, but what about when the two threads need to communicate?" Not only is this a common issue in threading, but it's also our second step: thread synchronization.

Thread synchronization can be done in many ways, so it's important to identify which type of cross thread communication is needed. Some examples would have one thread wait for another, pass data from one thread to another, or stop a thread based on data from another. All of these thread synchronization applications are valid; however, what's often used in embedded control GUIs is the need to block (pause) one thread until another thread completes or a certain amount of time passes.

Take an RFID scan. You may have a button in your GUI that tells your RFID scanner to return the RFID of a card in its field. When you press the button, your GUI will send the command and wait for a response. Your options are to block your GUI for a certain period of time or until the back end thread returns the data. This is an example of event synchronization, where one thread waits for the other to raise an event.

In the front-end thread, you must declare a thread signal:

```
EventWaitHandle frontEndSignal;
```

The following is for the back end thread:

```
EventWaitHandle backEndSignal;
```

A signal can be thought of as a state machine with three basic states: Set, Reset, and Wait. Set enables the owning thread to run. Reset blocks the owning thread. Wait blocks until another thread signals it to continue or a period of time has passed.

Now back to our example! Our user has requested an RFID, and we are willing to wait 5 s for someone to scan a RFID card.

```
frontEndSignal.WaitOne(5000, false);
```

The `5000` is in milliseconds, and the `false` is a default. Prior to telling the thread to wait, we would have sent the command. Now, in the back end thread, we send our command and either receive the RFID or not. If we receive the data successfully, we run:

```
backEndSignal.Set();
```

Our main thread can display the RFID. Or, after 5 s, a timer will expire and display an error message.

This threading concept is the basis for how our server handles talking to the various modules connected to it. If you expand the back-end thread idea into many threads that handle communication with each module in need of information from the server, you can begin to see how such a simple process can be expanded to solve complex problems. Each back-end thread either waits for a response from the module in question or times out. This will not affect the main thread. It will result in a very responsive GUI.

The server GUI is written in C# with Visual Studio .NET. We choose both C# and Visual Studio .NET because we were under a tight three-month deadline. It is often true that you can write a tighter GUI (or any code for that matter) using C or even C++ if you have the time. But when you're on a tight schedule, it's hard to ignore a visual editor, the support of thousands of classes, and online developer support.

The .NET framework provides easy-to-use functionality for handling connectivity and data transfer over the 'Net. Standard support for UDP, TCP, and FTP is all there! But standardized support is a double-edged blade. And although it's easy to use and quick to implement (we had the server sending and receiving UDP packets in about 30 min.), you may not find it when you need a lower level of control.

FUTURE DEVELOPMENTS

The design we've used for our board layouts and state machine enables us to add new features to our system easily in both hardware and software. Future additions may include things like facial recognition, a digital camera for monitoring access points, and triggers for indicating special events.

We still have error conditions to be exercised and real-world testing to take place before we have a finished product. We hope this article will help you start designing your own security system. There are some very cool technologies out there that are getting easier and easier to implement! ■

Kyle Cameron (kcameron796@shaw.ca) completed his electronics engineering technologist diploma at Camosun College. He is currently studying for a degree in electrical engineering. He finds embedded systems and programming fascinating and enjoyable.

Jeremy Warren (jeremywarren@shawlink.ca) is an electronics engineering technologist who received his certification from Camosun College. He is a volunteer member of the Canadian Coast Guard Auxiliary and spends his spare time playing with home control and automation.

Shane Reierson (reiersonshane@shaw.ca) is a graduate of Camosun College's computer engineering technology program. He is currently working toward a degree in computer engineering. His professional interests include automation, security, and system control.

Chris Rittaler (chris_rittaler@hotmail.com) completed the electronics engineering technology program at Camosun College. He is currently enrolled in the electronics engineering bridge program at Camosun College. His technical interests include microcontroller circuits and vacuum tube circuits (guitar amplifiers).

Nick Brady, an electronics engineering technologist, graduated from Camosun College in 2006. He lives with his wife in Port McNeill, British Columbia. His technical interests include embedded systems, trouble-shooting hardware, and things with flashing lights. You may contact him at sir_brady@hotmail.com.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2006/197.

SOURCES

MV1210 Fingerprint module
Bioscrypt, Inc.
www.bioscrypt.com

MicroRWD Transceiver
IB Technology
www.ibtechnology.co.uk

dsPIC30F6014A Digital signal controller, ENC28J60 Ethernet controller, and PIC18F4580 microcontroller
Microchip Technology, Inc.
www.microchip.com

by Ed Nisley

Filter Figures

Ed revisits a filter design he worked on back in 2003, but this time around, he has better instruments. Read on for information about filter shapes, flatness, and trade-offs with skirt steepness. He also describes the effect a wider filter has on the output in the application.

The power-line frequency monitor project I presented in my December 2003 column ("Multiplying, Dividing, and Filtering," *Circuit Cellar* 161) derived a stable 11.25-MHz microcontroller clock from a GPS-locked 10-MHz sine wave. The clock frequency is nine-eighths of the input, with divisors of 12 and 60 to suit both the microcontroller and the 60-Hz power line. The circuitry in Photo 1 includes two filters that isolate the third harmonic of two square waves, thus multiplying the input frequency by a factor of nine.

Tom Napier, a long-time *Circuit Cellar* reader and feature writer, pointed out that tight band-pass filters introduce an interesting problem: mechanical changes can cause phase modulation. This wasn't an issue in my project, but if you're designing precise hardware that must work despite thermal, aging, and shock stresses, it's worth considering.

In this column, I'll take a look at how the actual hardware behaved and then examine how broad filters can solve one problem while introducing another.

SOLDER & SOFTWARE

The circuitry in Photo 1 performs the 10- to 11.25-MHz frequency conversion. The 10-MHz input passes through a digital divider to become a 5-MHz square wave with a precise 50% duty cycle. A band-pass filter (in the red ellipse) extracts the 15-MHz third harmonic. Subsequent circuits divide the frequency to 7.5 MHz, triple it to 22.5 MHz, and then divide it again to create an 11.25-MHz square wave for the microcontroller, which lives on a separate board.

That 15-MHz filter has the frequency response shown by the blue curve in Figure 1: a 0.6-MHz bandwidth centered at 15.3 MHz (p. 40). Although the design frequency was 15 MHz, I built and tweaked those filters using a tedious manual process, and it's not surprising the final result is off by a bit. Since then, I've acquired an HP 8591E spectrum analyzer and a TAPR/Ten-Tec Vector Network Analyzer (VNA), both of which agree on the filter's characteristics. The HP box reports that the signal source I used is off by about 0.1 MHz, which would explain a big chunk of the error.

The green trace in Figure 1 shows that the filter's phase response passes through –180° at the middle of the peak. Although the phase changes smoothly from about 0° through –380°, the VNA wraps its output values to remain within ±180°. The vertical discontinuity is thus an artifact, not a physical event.

I don't have room to list the VNA's output file, but the numbers show that the phase changes by 1° per 3.8 kHz around the –180° point. Put differently, if the filter's center frequency changes by 3.8 kHz, the signal's phase changes by 1°. A 3.8 kHz change represents a mere 0.025% frequency shift at the filter's center frequency, so the phase can vary dramatically based on very small physical changes.

Although I enjoy building hardware as much as anyone, it's easier to explore circuit options with mathematical models. The schematic in Figure 2 (p. 40) includes measured values for the filter's components and produces the purple curve in Figure 1, which matches up nicely with the actual hardware response, apart from a constant magnitude offset. The phase response, which isn't shown, matches just as well.

The double-tuned filter in Figure 2 is essentially two tank circuits (L1-C3 and L2-C5) linked by C4, a small "gimmick" capacitor made from a short length of RG-174 coaxial cable. Each tank produces a sharp resonant peak with an abrupt phase change and, in the filter I built, two identical tanks combine to produce a single peak. Starting from that circuit, some empirical fiddling retuned the tanks to slightly different frequencies, producing the double-peaked response shown in red in Figure 1.

Tom pointed out the relatively gentle slope in the middle of the phase response reduces the effect of mechanical changes, at least for a single-frequency signal. The inevitable trade-off appears in the reduced attenuation on either side of the passband: about 25 dB less. Investigating that effect led to some different filter models.

FILTER FLATNESS

The double-tuned filter design equations produce a flat-topped Butterworth response across the desired bandwidth, so my tweakage was obviously suboptimal. I decided to switch to a standard Butterworth filter topology so that I could easily change the filter's order.

Figure 3 (p. 41) shows a third-order Butterworth band-pass filter with a 10-MHz bandwidth centered at 15 MHz. For a band-pass filter, the "order" is half the number of poles, which corresponds to the number of capacitors and inductors. A first-order filter with two poles consists of just L1-C1. A second-order filter adds L2-C2 to get four poles.

A band-pass filter's order determines how rapidly the attenuation increases outside the passband. For reasons well beyond the scope of this column, the attenuation increases by 6 dB per octave times the filter's order, at least for frequencies more than an octave beyond the band-pass edges. Figure 4 (p. 41) shows the close-in response so you can see the details, but it's obvious the third-order filter (red) is dropping much more steeply than the second-order filter (purple), even though both have a 4-MHz bandwidth.

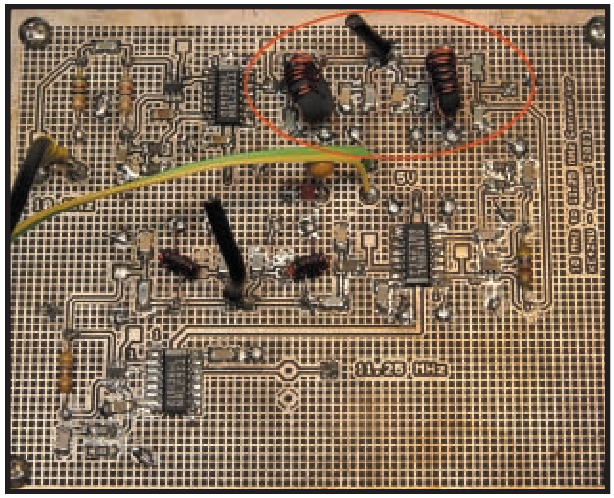


Photo 1—*The red oval highlights the 15-MHz double-tuned filter that extracts the third harmonic of a stable 5-MHz square wave.*

The green trace represents a third-order 2-MHz bandwidth filter, which is much broader than the original double-tuned filter shown in blue. Notice that the phase changes almost linearly across the passband, reaching –360° at the center, without the kink found in my tweaked double-humped model.

A band-pass filter's Q is the ratio of its center frequency to its bandwidth, a standard parameter in filter design equations. It's unrelated to the filter's order, which determines the attenuation behavior outside the passband.

The original double-tuned filter has a reasonably large Q of 25:

$$Q = 25 = \frac{15 \times 10^6 \text{ Hz}}{0.6 \times 10^6 \text{ Hz}}$$

Increasing the bandwidth to 2 MHz lowers the Q to 7.5, and going to 4 MHz reduces it to 3.75.

High-Q filters require close attention to detail during design and construction. The frequency error in my original filter shows that better test equipment makes sense, too. Fortunately, isolating a third harmonic depends more on attenuation than passband tolerances.

HARMONIC EFFECTS

Representing a square wave in the frequency domain combines an infinite number of sine waves, as shown by the familiar Fourier transform of a unit-height square wave of period T:

$$f(t) = \frac{4}{\pi} \sum_{n=1,3,5,\ldots}^{\infty} \frac{1}{n} \sin\left(\frac{2\pi n t}{T}\right)$$

Ignoring the 4/p scaling factor, the fundamental frequency component

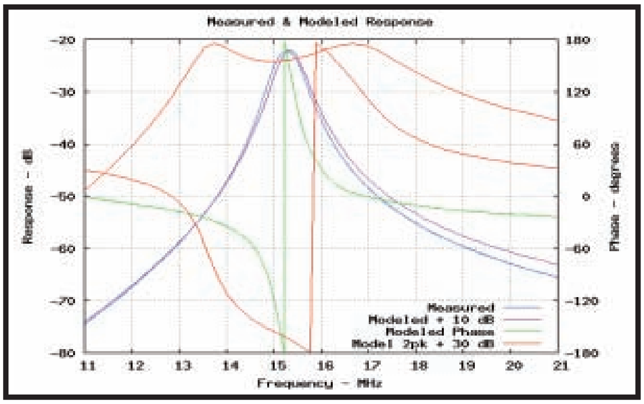


Figure 1—*The measured (blue) and modeled (purple) magnitude behaviors match up almost exactly. The measured phase (green) passes through –180 at the filter's peak response. The phase of a modified double-tuned filter (red) can be flatter in the middle, but the magnitude response shows two distinct humps.*

has an amplitude of 1, the third harmonic 1/3, the fifth 1/5, and so on. Expressed in decibels relative to the fundamental, that's 0, –9.5, –14, and so on.

The horizontal scale in Figure 5 (p. 42) ranges from the 5-MHz fundamental on the left to the 35-MHz seventh harmonic on the right. The TAPR/Ten-Tec VNA's dynamic range is about 70 dB and the measurements at 5 and 25 MHz were obviously invalid, but eyeball extrapolation of the double-tuned filter's response (blue) shows that it attenuates the fundamental by about 70 dB and the fifth harmonic by 50 dB.

Passing a square wave through that filter would produce a signal with a –60-dB 5-MHz component and a –54-dB 25-MHz component, relative to the 15-MHz signal at 0 dB. For comparison, a golden-ears audio amplifier might have 0.1% distortion: –60 dB. Obviously, the filter produces a rather pure sine wave.

The purple curve in Figure 5 comes from the 10-MHz bandwidth filter shown in Figure 3. A square wave would emerge from that filter with a 5-MHz component at –35 dB and a 25-MHz component at –30 dB. In fact, even the 35-MHz seventh harmonic would be at –47 dB. Considering that –35 dB represents about 2% distortion, the result might look odd.

Figure 6 (p. 43) shows the time-domain output of those two filters, both fed with a 5-MHz square wave. The dark blue trace, from the double-tuned filter, has no visible artifacts. The light blue trace from the 10-MHz wide filter, however, has obvious problems: the peaks repeat in a regular three-cycle pattern.

The frequency-multiplying circuitry converts the filter output into a

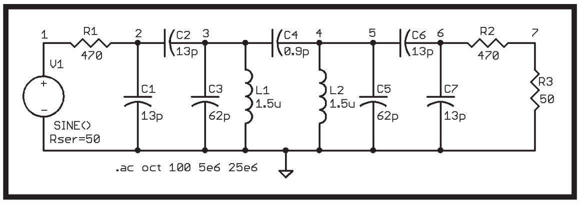


Figure 2—*This Spice model approximates the double-tuned filter as I built it, with measured capacitor and inductor values, but excluding stray capacitance, wiring inductance, and other imponderables. The filter has 500-W input and output impedances, rather than the usual 50 W, to better match the capabilities of the surrounding circuitry.*

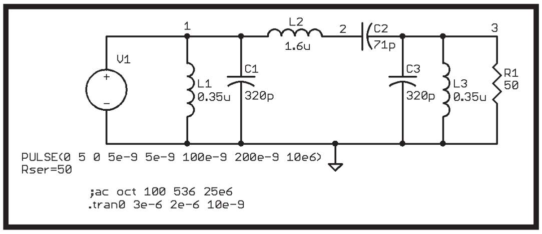


Figure 3—*A third-order Butterworth filter resembles the double-tuned filter, but with a 10-MHz bandwidth and standard 50-Ω impedances.*

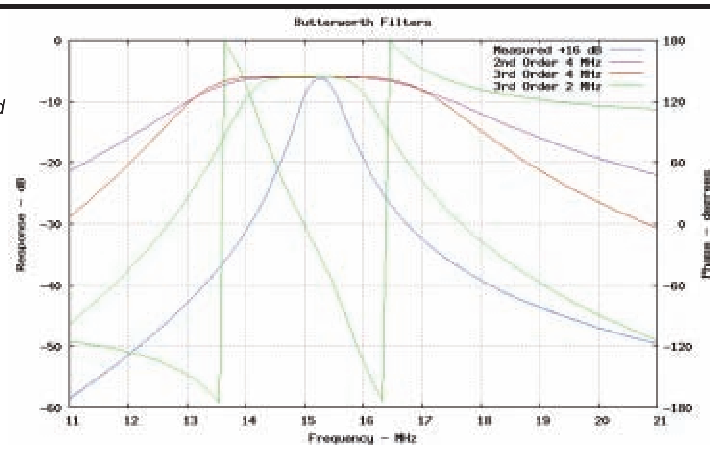
square wave using a high-speed comparator triggered at what should be the zero crossings. Amplitude differences move the average value away from the midpoint, so the comparator triggers at a different phase on each cycle. A three-cycle distortion has a particularly nasty effect, as the next circuit divides the frequency by two: successive cycles will have different periods.

In contrast, a smooth phase change caused by mechanical variations will have little overall effect, because the gadget's purpose is to compare the power line frequency with the 10-MHz reference. Changing the phase does require changing the frequency, but when the mechanical parts settle down, the overall frequency will be the same.

In a different application, however, small phase changes can be catastrophic. Video filters come to mind, where the color displayed on the screen depends on the phase relation between the signal and the color subcarrier. In that situation, you must design filters with very stable phase characteristics, even at the expense of amplitude variations.

To show how that works, the two filters in Figure 6 have an obvious phase difference resulting from the same 5-MHz square wave. The double-tuned filter has the as-built 15.3-MHz center frequency you've seen before. I designed the 10-MHz bandwidth filter with –3 dB points at 10 and 20 MHz, so the center frequency is the harmonic mean of those two values:

Figure 4—*Butterworth filters have smooth passbands and relatively broad skirts. The filter phase varies smoothly across the passband.*



$$F_0 = 14.4 = \sqrt{10 \times 20} \text{ MHz}$$

Homework: determine the actual phase difference for both signals at 15.0 MHz and then decide if what you see in Figure 6 makes sense.

POWER PROBLEMS AHEAD

I had unsoldered the filter's input and output connection to attach the VNA through calibrated coaxial cables. When I was all done, I restored the connections, reassembled the box, plugged it in…and it didn't work. The display's LED backlight lit up, but no digits appeared.

Because the whole point of the filter was to generate the microcontroller clock, I figured I'd probably shorted a PC trace and, indeed, I had. That's easy to fix, but before rereassembling the box, I plugged it in and…and it still didn't work.

A bit of probing showed that the filter was working, the clock was running, and everything seemed fine. Then I probed the microcontroller's RESET pin and found it was high with a periodic dip every 8.33 ms. Bingo!

When I originally built the gadget, I used a "6 VAC" wall wart that actually produced about 8 VAC under the minimal load this gadget applied. That was close to the lower limit for the LM317 regulator's input voltage, but I figured that reducing the power dissipation in the regulator was a Good Thing and just ran with it.

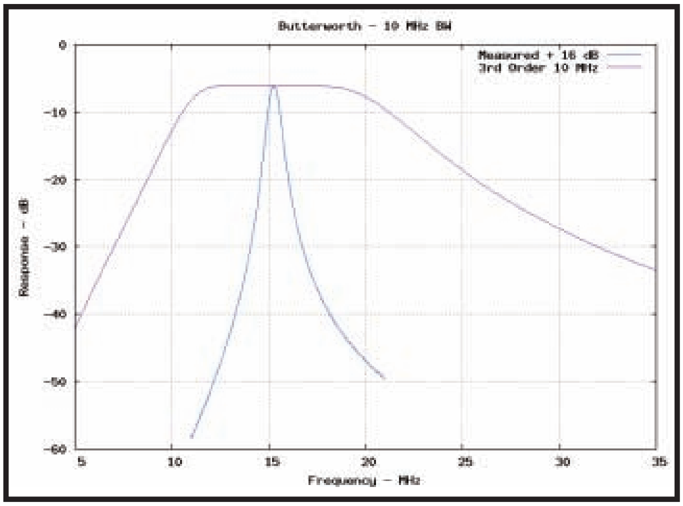


Figure 5—*The 10-MHz bandwidth filter in Figure 3 has relatively poor attenuation at 5 and 25 MHz, exactly where the 5-MHz square wave input has high energy.*

I'm writing this in the middle of a 100°F mid-summer heat wave, with extremely high power demand, and the AC line voltage is now 117 on one leg and 107 on the other, rather than the usual 120+ V. Guess which leg feeds the outlets at my electronics workbench?

I swapped in a 12-VAC wart, put a heatsink on the LM317, and the line-frequency monitor works fine again. I'd prefer a lower voltage wart, but there aren't any in my collection.

Obviously, I neither designed nor tested my "product" for low-line conditions. That's mostly a function of knowing my customer base: me. If

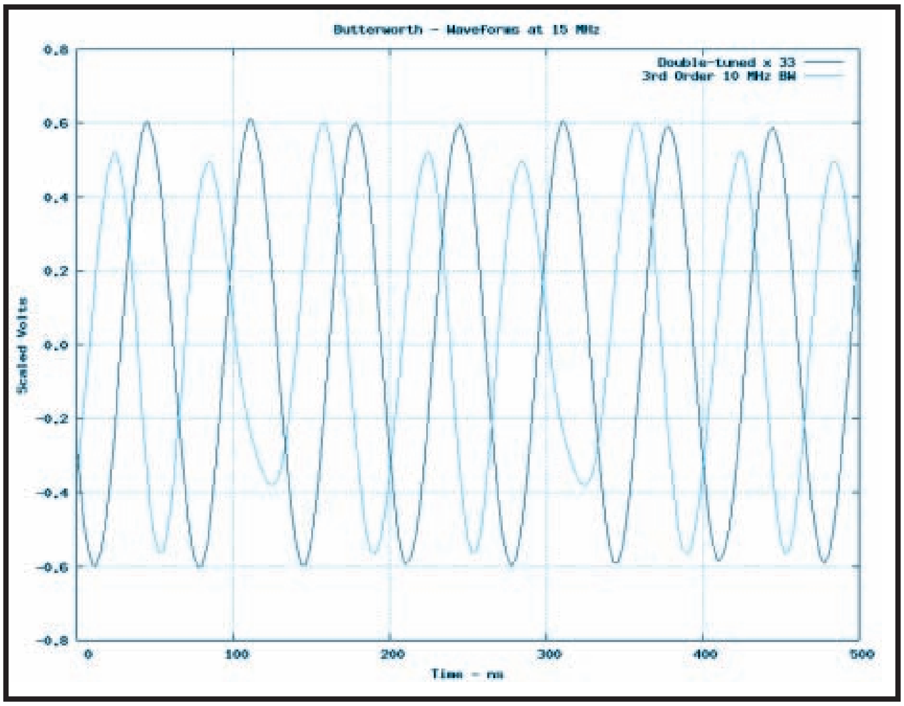


Figure 6—*The 15-MHz third harmonic appears distorted by the fundamental and fifth harmonic leaking through the skirts of the 10-MHz BW filter (light blue). The as-built DT filter (dark blue) produces much better results, but with a significant phase difference at 15 MHz.*

(when!) a gizmo fails, I simply tear it apart, figure out what happened, and perhaps tell you the story.

However, the more important message is that the nominal "120 V ±5%" design point for North American equipment may need some revision in the years ahead. The national power distribution system has run out of headroom, ensuring that we'll have more overloads, brownouts, and rolling blackouts. That's not even considering the lack of generating capacity, but I won't get into that subject at all.

Perhaps I should build a power-line voltage monitor, too?

CONTACT RELEASE

I created the graphs in this column by exporting the VNA measurements and SwCAD simulation data to text files, then using Gnuplot. This is much easier than fiddling with screenshots, even if the results don't have quite the same hands-on-the-knobs appearance. ■

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

PROJECT FILES

To download the SwCAD Spice models, graphs, and Gnuplot scripts, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2006/197.

RESOURCES

Fourier Series: Square Wave, Wolfram MathWorld, http://mathworld.wolfram.com/FourierSeriesSquareWave.html.

Gnuplot, http://gnuplot.info/.

W. Hayward, R. Campbell, and B. Larkin, *Experimental Methods in RF Design*, ARRL, Newington, CT, 2003.

LC Filter Design, http://www-users.cs.york.ac.uk/~fisher/lcfilter/.

Power line information,http://en.wikipedia.org/wiki/Mains_electricity.

SOURCES

CAD III Spice simulator
Linear Technology Corp.
www.linear.com/company/software.jsp

TAPR Vector network analyzer
Ten-Tec, Inc.
http://radio.tentec.com/Amateur/vna

by Aubrey Kagan

Visual Basic 2005 and the Serial Port

The capabilities of Visual Basic 2005's serial port model may seem limited, but if you know what you're doing, you can definitely use it for numerous projects. Aubrey recently got creative with Visual Basic 2005. Now you can too.

Pity the serial port on the PC. It no longer gets the respect it deserves. As a result of its simplicity and reliability, it still forms a major part of embedded computer communications. Microsoft's belief in the superiority of the USB bus has led to the hardware being excised from our computers. Yet there must be a feeling of remorse somewhere in the Visual Basic language development teams because the latest release still has a serial port facility. Apparently, the feelings were not too intense because precious little information about it appears in the Help files and other publications. There are also some problems, although you will see that the serial port is still usable.

In earlier iterations of Visual Basic, you had access to MSCOMM32 (provided you had the Enterprise or Professional editions of the language). It was also possible to enable this in other versions including the macro language of Microsoft's applications: Visual Basic for Applications (VBA). In many cases, however, there were problems with getting the control to appear in Visual Basic.[1] Visual Basic was always event driven, but with the introduction of VB.net, Microsoft decided to enhance the language by including a far more object-oriented programming (OOP) approach.

I write code in Visual Basic about once every 18 months. Typically, all I want is a simple interface that I don't have to relearn each time I use the language. The problem with this approach is that the time will come when the gap between what I know and what I need to know is too great to bridge. I may be at that point already. The books for beginners ignore the fact that they are simply recycled ideas for those of us familiar with earlier versions of Visual Basic. Either from boredom or disinterest in the applications, I haven't learned much from them. The more advanced books and the Visual Basic 2005 (VB2005) Help files are far more difficult. I understand most of the words and some of the sentences. From what I understand at the end of the paragraph, however, it seems to be written in a foreign language.

So why not resort to some other implementation of Basic? Well, I am sure the newer versions of VBA used in Excel and Word will evolve into a .net approach. In addition, Microsoft is giving away not only VB2005, but also the entire Visual Studio (with some restrictions, obviously). These versions are referred to as the Express edition.[2]

Although it appears to be possible to use MSCOMM32, VB2005 includes a serial port class in the Express edition.[3, 4] As a result, I thought I could use it to learn about the OOP approach to programming. Talk about jumping in at the deep end. I know the old saw that a bad workman blames his tools. I don't doubt that an element of the difficulty I was experiencing had to do with my lack of understanding, but I feel as though the Help files and other documentation are wanting. I have never been anywhere near Missouri, but it seems to me that I have an affinity for Missourian philosophy when it comes to learning. Unfortunately, we are shown very little. Few examples, reference to examples that are not there, and obfuscation with difficult concepts are all part of the problem. In addition, some of the examples work only under certain circumstances (and worst of all, sometimes they're wrong). The available support comes from the user forums.

This article is for all of you fellow Missourians at heart. If you aren't from North America, please note that Missouri is known as the "Show-Me" state.

DEFINITIONS

You intuitively know what a physical object is. This concept is extended into software by analyzing the components that make up an object. You can create boundaries around the concept, attach a name, describe things that can happen to it (events), define different components that affect or show its behavior (properties), and ascribe certain actions that it can do (methods). Put the events, methods, and properties together, and they are called the object members. An object can be generalized into a class. You can use the class as a prototype to generate several identical objects.

Sometimes the object is not quite what you want and you need additional members. Under such circumstances, you can create a class that uses (inherits) the members of the founding class and adds the additional members. The serial port is a class. When you use it to create an instance, it becomes an object with all of the members of that class. Later, I'll describe two serial port objects created from the same class.

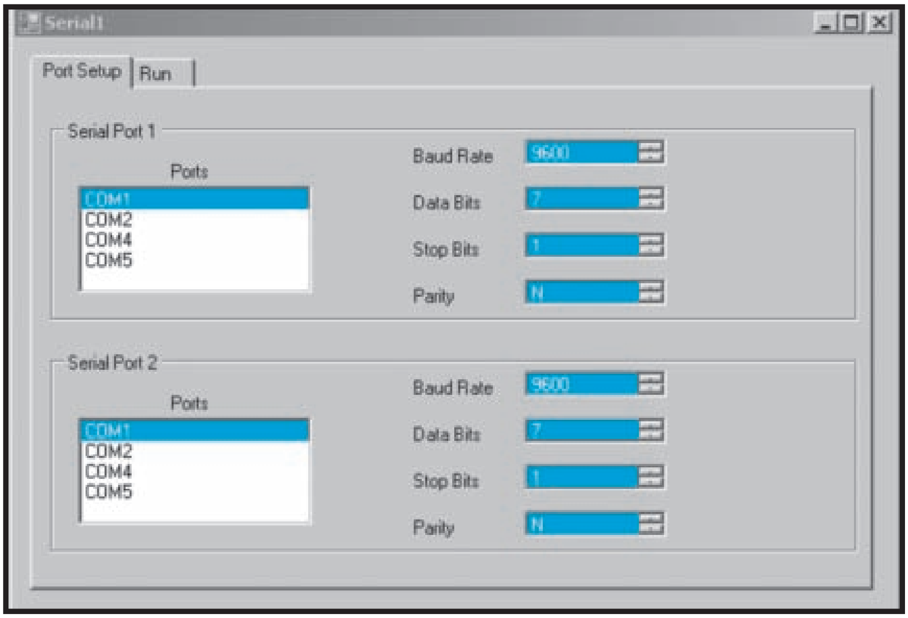


Photo 1—*All the options are built into the software. You can't enter any data. You can select only from the options by clicking in each list box.*

EVENTS

There are four possible events listed in the Help files: Data Received, Error Received, Pin Changed, and Disposed. The first three are self-explanatory, and the last is a mystery of no consequence. Events are asynchronous. One of the problems associated with the VB2005 implementation is that you can't associate an error detected with a particular data byte received. Reception and Error Detection are two separate events, and you can't guarantee the order in which they will occur and their relative timing. It is possible to use the serial port without using the events, but there are certain things that you can't do as a result of the class model's shortcomings. For instance, you can't read the error status in any way. You can detect if an error has occurred only if the event is triggered.

You deal with events through pieces of code called Event Handlers, which I will cover soon enough. Just a word of caution though: if you have an event handler in your code and that event occurs, then the code will be executed. There is no way to disable this as you would disable an interrupt in hardware. If the code is not there, nothing will happen and everything will continue as planned. This is not normally a problem because a project normally takes a particular approach and stays with it.

When an event occurs, data are passed to it by way of arguments, which can be processed even further depending on the event. For instance, when an error occurs, the error that caused the event (e.g., parity error) is passed as an argument.

METHODS & PROPERTIES

There are many possible methods that exist for the serial port that can be investigated in the VB2005 Help files. I will describe only a few of them. Unfortunately, some of the methods don't always work (or work exactly as advertised). I will mention them at a more appropriate time.

A plethora of properties are available in the serial port class. Again, you can find them in the VB2005's Help files. The different Read methods load data into a buffer that is sized according to the property `ReadBufferSize`. Unfortunately, this is a buffer of only received data. No information like parity is stored with the data, and it's impossible to correlate an error with the particular byte that caused the error. You could try to reduce the buffer size to one or two to isolate the byte. I have seen reports that this has been done, but in contradiction the Help file description of the property states that any value below 4,096 is rounded up to 4,096. I have been unable to judge which is correct. The `ReceivedBytesThreshold` property determines how many bytes are received before the `Receive` event occurs.

SERIAL PORT CONTROL

Perhaps the first hint that there are shortcomings with the serial port control (in VB2005) happens right after

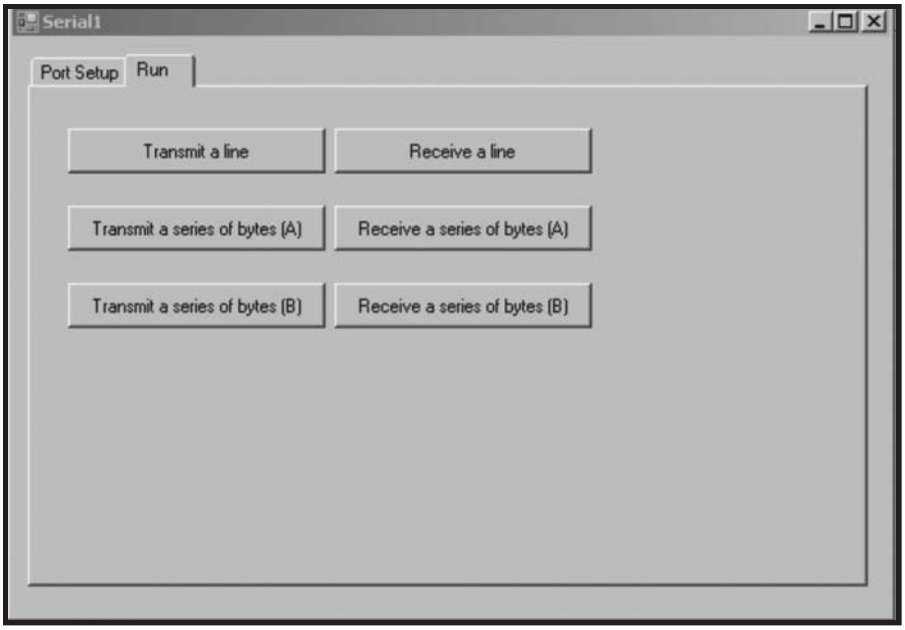


Photo 2—*The user interface triggers transmission and reception for the different approaches in the Serial1 project.*

you drag the Serial Port Control from the toolbox and drop it on the Form. When you investigate the events associated with the control, you will discover that there is only one event imported. There are, however, several. A closer look shows that the properties (unimportant things like the data rate!) don't seem to affect the port's operation at all.

VB2005 has early and late binding options. In most modern software, when writing code that accesses an object, you start with the object and then place a period and add a member possibly followed by additional periods and detail. For instance, you could have `IO.Ports.SerialError.RXParity` as a particular property that becomes accessible through an event within the serial port object.

Early binding means that Visual Basic has foreknowledge of the object and prompts you with all of the available choices each time you type a period. This is a very handy feature. It is not necessary to place the Serial Port Control because you won't be using it directly. But placing it does allow for early binding, so I recommend that you do it. Late binding means that the compiler doesn't consider the validity of the elements used in the object until the project is compiled. An error is generated only at that time (only if there is one, of course).

If you are writing a program and you know your exact hardware configuration, you can configure the ports directly. However, you may want to generalize so that you list all of the serial ports that are available on the PC. It would also be prudent to allow a change in data rate, parity, and other settings. I created a tabbed user interface as the Serial1 example, which is posted on the *Circuit Cellar* FTP site. The Setup tab has controls for two serial ports. I worked with two so that I could transmit from one to the other just to make sure the software was working. Photo 1 shows the Port Setup screen in operation.

In addition to placing and sizing the controls, some code is required to make all of this happen. Article size restrictions prevent repeating all of the listings here, so please refer to the `Serial1_Load` procedure in the project. This code runs every time the application starts. The entries for each list box are loaded in the form load event using the `Add` method. The default selection is achieved by changing the `SelectedIndex` property. Note that a space followed by an underscore (as seen in the actual `sub` declaration) indicates a line continuation.

In order to use a serial port, you need to `Open` it. There are two ways to do this. The first is the `OpenSerialPort` method (written as a single line in the code):

```
Using SerialPort1 As IO.Ports.Serial
   Port = My.Computer.Ports.OpenSerial
   Port("com4", 19200,_IO.Ports.Parity.
   Even, 8, IO.Ports.StopBits.One)
```

Unfortunately, in using this approach, some methods like `DiscardInBuffer` won't work at run time and will return an error message. I recommend using the second method (`SerialPort.Open`), which is seen in the project as the `OpenPorts` procedure. It implements the serial port parameters that appear under the user interface's Setup tab.

You can open the port anywhere, but because you are only going to transmit from within the Run tab, I placed the `OpenPorts` procedure in the tab select event `tabRun_Enter` so

Listing 1—*In the Help files concerning the topic of* `serialport.write`*, there is something about having to convert from ASCIIencoding to UTF8Encoding because any character above ASCII 127 is converted to a question mark. Ignore this comment. If you format the data that is being transmitted as a byte, all bits will be transmitted as expected.*

```
Private Sub btnSendBytes_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnSendBytes.Click
    Dim bytSequence() As Byte = {&H83, &H45, &H33, &H88}
    SendBytes(bytSequence)
End Sub

Public Sub SendBytes(ByVal data() As Byte)
    SerialPort1.Write(data, 0, data.Length)
End Sub
```

Listing 2—*All I am doing is acquiring the bytes and displaying them on the screen. Again, you need to click on the Transmit a Series of Bytes (A) button. After that, click on the Receive a Series of Bytes (A) button (see Photo 2).*

```
Public Sub ReceiveBytes()
    ' receives it on  serial port2
    Dim bytReceived(20) As Byte
    'dim includes the 0 case
    'must be dimensioned or you will get a run time
    'error with the "bytReceived(iI) = SerialPort2.ReadByte"
    'below- this will be generalized
    Dim iI As Integer = 0

    While SerialPort2.BytesToRead > 0 Or iI = 0
        'note that if we are waiting for a message
        'the >0 will show as false immediately,
        'so the iI=0 condition is added
        bytReceived(iI) = SerialPort2.ReadByte
        iI = iI + 1
        End While
        MsgBox("Number of bytes received- " & Str(iI) & vbCrLf & _
        BitConverter.ToString(bytReceived), _
        MsgBoxStyle.OkOnly, "Message Received")
End Sub

Private Sub btnReceiveByte_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnReceiveByte.Click
        ReceiveBytes()
End Sub
```

that it runs every time you click the Run tab. You can't open a port if it's already open, and it should always be closed when it becomes available for the rest of the system. The safest place to put this is in the `FormClosing` event. (Once again, this is in the project listing.)

LINE VS. BYTE TRANSMISSION

The serial port class enables two forms of data communication: a line or a byte. With the line approach, everything is treated as 7-bit ASCII text, so the most significant bit of the data is always zero. The communication also has a line termination character so that the end of the line can be detected. Although the end-of-line uses the C standard of 00_{HEX}, it can be modified by changing the `NewLine` property. I'm sure you will be involved with 8 bits of data, so I won't spend a lot of time on the line communication.

There are a number of buttons with descriptive names under the Run tab (see Photo 2). Clicking on one of the buttons will trigger the associated event, which in turn will call one of the routines.

LINE TRANSMISSION

Data transmission is extremely easy in Visual Basic. It requires no more than the following instruction:

```
SendLine("Sending a line of data")
```

You can also use a string variable as the parameter passed to the `SendLine` procedure.

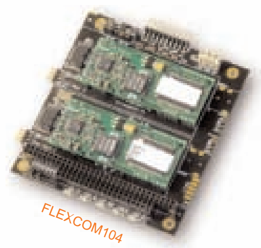
The Visual Basic designers decided that everything should be event driven. You will see later how data reception, errors, control line change, and several other occurrences are treated as events. I find it ironic that the transmission sequence is not. The processor will not emerge from the `SendLine` until the last character is being shifted out.

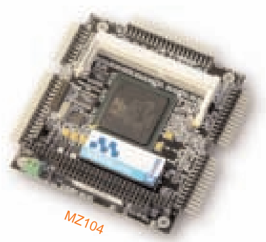
BYTE TRANSMISSION

Transmitting bytes is not that different than line transmission (see Listing 1). You can achieve 8-bit transmission, and there is no termination character. A user-selected communications protocol determines how the receiving end will detect an end of message. This protocol could be a fixed message length or it could embed the number of bytes in the message. It could use a reserved character or a sequence of characters for message synchronization. It could also add error detection

Listing 3—*Here you see byte reception with a timeout. The message is returned. As in previous examples, you need to click on the Transmit a Series of Bytes (B) button and then click on the Receive a Series of Bytes (B) button. Clicking on the Receive button alone will force the timeout error.*

```
Public Function bytReceiveBytes() As Array
    Dim iRx(254) As Byte
    ‘dimension as the largest possible number of bytes
    Dim iI As Integer = 0
    Dim iNumberOfBytes = 255
    ‘initiate timer- message must be received within 1 second
    Timeup2.boolExpired = False
    Timer2.Interval = 1000     ‘1 sec delay
    Timer2.Start()
    iRx(0) = 0     ‘prepare flag for pass

    ‘wait for all the bytes to be received, or a timeout occurs
    While (iI < iNumberOfBytes) And (Timeup2.boolExpired = False)
     ‘so don’t overflow buffer,

     Application.DoEvents()  ‘this allows Windows to “multitask”
     ‘the timer interrupt will not happpen without this line
     If SerialPort2.BytesToRead > 0 Then
        ‘there is a byte or more in the buffer
        iRx(iI + 1) = SerialPort2.ReadByte
        ‘offsetting by 1 to allow for iRx(0) to be used for status
        ‘now checking on the numebr of bytes
        If iI = 1 Then
           ‘this is the 2nd byte
           iNumberOfBytes = iRx(iI + 1)
           ‘if fixed length message then can ignore this if clause
        End If
        iI = iI + 1  ‘bump to read next byte
     End If
    End While
    If (Timeup2.boolExpired = True) Then
        iRx(0) = 1 ‘set
        iI = 0
    End If
    ReDim Preserve iRx(iI)  ‘should be iI+1, because increment within
    ‘while loop, but iRx(0) has been inserted
    bytReceiveBytes = iRx
    ‘set up return values
End Function

Private Sub btnSendBytesB_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnSendBytesB.Click
    Dim bytSequence() As Byte = {&H83, &H5, &H33, &H88, &H99}
    ‘5 bytes embedded as 2nd byte
    SendBytes(bytSequence)
End Sub

Private Sub btnReceiveBytesB_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnReceiveBytesB.Click
    Dim bytBytes() As Byte
    bytBytes = bytReceiveBytes()
    If bytBytes(0) <> 0 Then
        ‘error detected- only timeout at the moment
        MsgBox(“Timeout detected”)
    Else
        MsgBox(“Return Code + Bytes received- “ & _
    BitConverter.ToString(bytBytes), _
    MsgBoxStyle.OkOnly, “Message Received”)
    End If
End Sub
```

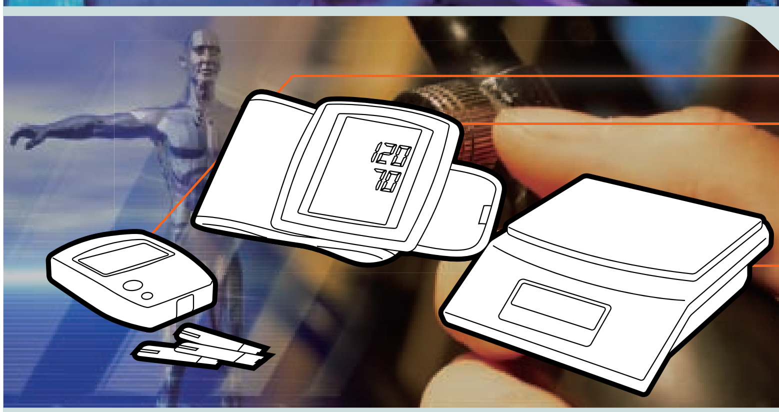





HOLTEK
www.holtek.com

- Wire/Wireless Keyboard
  HT82K96E/95E/94E/68E
- 10-key with Calculator
  HT82K94E

- USB Phone
  HT82A832R
- Touch Panel
  Full Speed USB MCU
- USB Joystick/Game Pad
  HT82J97E



- Blood Glucose Meter
- Blood Pressure Meter
  Data can be transmitted to PC
  HT46RB50/70
- Digital Electronic Scale
  Data can be transmitted to PC
  HT46RB50/70

- Card Reader
  Full Speed USB MCU
- Writing Tablet
  HT82K96E
- Infrared Transmitter/Receiver
  Full Speed USB MCU

USB Intelligent MCU Devices - Total Functional Integration

HOLTEK SEMICONDUCTOR INC.

Holmate Semiconductor, Inc.
46729 Fremont Boulevard,Fremont, CA 94538
Tel: (510) 252-9880 Fax: (510) 252-9885
www.holmate.com

ZigBee™/ IEEE 802.15.4 Transciever

The CC2420 from Chipcon was the industry's first single-chip 2.4 GHz IEEE 802.15.4 compliant and ZigBee-ready RF transceiver. Chipcon's goal is to become the leading provider of this technology.

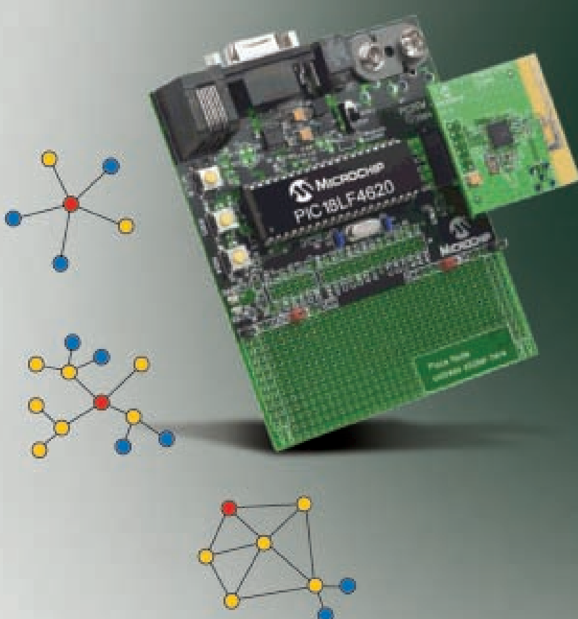
Chipcon Products from Texas Instruments mouser.com/chipcon/a

2.4 GHz ZigBee™ Embedded OEM Modules

These IP-Link modules simplify wireless integration and reduce development cycles by six months or more. Each module includes an IEEE 802.15.4-compliant radio, a Silicon Laboratories 8051 microcontroller, programmable I/O, flexible antenna and range solutions, as well as ZigBee-ready IP-Net networking software.

Helicomm mouser.com/helicomm/a

Turn to Mouser for All Your ZigBee™ Product Needs!



Mouser delivers the newest products and the latest technologies, including ZigBee -- the hottest wireless technology protocol in the marketplace today.

Based on the IEEE 802.15.4, ZigBee is an open standard for reliable, cost-effective, low-power wireless applications. And it's a standard that works globally and can be configured for star, peer-to-peer, and mesh networks.

Experience Mouser's time-to-market advantage! Our vast selection of the NEWEST products, NEWEST technologies, **new catalog every 90 days,** no minimums, and same-day shipping on most orders, gets you to market faster. We make it easy to do business with Mouser!

mouser.com **(800) 346-6873**

MOUSER ELECTRONICS
a tti company

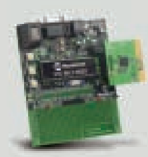
The Newest Products For Your Newest Designs

2.4 GHz ZigBee™ Development Kit

Everything you need to immediately begin developing star, cluster-tree, and mesh networks. Each of the six development boards included in the kit contains a high-performance C8051F121 MCU, an IEEE 802.15.4 compliant RF transceiver, the MAC firmware, and the Helicomm ZigBee Protocol stack.

SILICON LABS mouser.com/silabs/a

PICDEM™ Z Demonstration Kit



This kit is an easy-to-use ZigBee™ Technology wireless communication protocol development and demonstration platform. The kit includes the ZigBee protocol stack and two PICDEM Z boards, each with an RF daughter card. The demonstration board is also equipped with a 6-pin modular connector to interface directly with Microchip Technologies' MPLAB® ICD 2 in-circuit debugger (DV164005).

MICROCHIP mouser.com/microchip/a

ZigBee™ Evaluation Kits

Short range, low power, 2.4 GHz ISM band transceiver. Contains complete 802.15.4 PHY/MAC, ZigBee protocol stack supporting star, and mesh networking. The MC13193 can be a stand-alone transceiver or part of the Freescale ZigBee-ready platform when combined with an appropriate microcontroller.

freescale semiconductor mouser.com/freescale/a

The NEWEST Semiconductors | Passives | Interconnects | Power | Electromechanical | Test, Tools & Supplies from Mouser Electronics

Listing 4—*Properties can be simple variables or they can actually do some work. In this example, the incoming series of bytes is written to the `ReceivedMessage` property. When this is done, the process sets a flag that signifies that there is in fact a message and it's stored in the object. It also analyzes the data. In this case, it looks at the received message and checks to see that the first byte is A5hex (a make-believe header in this instance). If it is, then a `HeaderSeen` property is set. As you can see, two objects are instantiated.*

```
Public Class ProcessHandshake
    Public ErrorDetected As Boolean
    Public HeaderSeen As Boolean
    Public MessageArrived As Boolean
    Private Received_Message As Array
    Public Property ReceivedMessage() As Array
     Get
      ReceivedMessage = Received_Message
     End Get
     Set(ByVal value As Array)
       Received_Message = value
       'store the message
       MessageArrived = True
       'indicate there is something in the buffer
       If Received_Message(0) = &HA5 Then
        'must be 0xff in header for correct message (for example)
        'this flag is used indicate that it recognizes the information
        HeaderSeen = True
       Else
        HeaderSeen = False
         End If
       End Set
      End Property
End Class

Public Class Serial2

       Dim IF1 As New ProcessHandshake
       Dim IF2 As New ProcessHandshake
.
.
.
```

techniques or comply with a published standard.

LINE RECEPTION

In order to demonstrate reception, I connected `SerialPort1` to `SerialPort2`. The `OpenPorts` procedure and `FormClosing` event are modified to include the extra port. In basic form, line reception is trivial, as you will see in the project in the `ReceiveLine` procedure. It is invoked when you click on the Receive a Line button and trigger the `btnReceiveLine_Click` event.

How do you detect an error in transmission or missed transmission? I will deal with detailed error detection later. (Remember that the line communication is just covered in passing.) Because the receiver "knows" when the end-of-line occurs, it's possible to detect a timeout failure, as you will see in the `Try..Catch` construction, so that an error is detected when a timeout occurs. It is easy enough to test. Just click on the Receive a Line button without clicking on the Transmit a Line button (see Photo 2). You can also force an error by clicking on the Transmit a Series of Bytes button.

Listing 5—*This code is added to the `OpenPorts` procedure, although it can just as easily be omitted. I don't know if this is a flaw in Visual Basic or if it's merely intended to enable you to have several handlers of different names and switch between them by giving them different handler names. Including them, however, helps documentation in detailing what is expected to happen.*

```
AddHandler SerialPort2.ErrorReceived, _
AddressOf SerialPort2_ErrorReceived
'must appear after port is opened
'the name SerialPort2_ErrorReceived is arbitrary,
'but must match the handler name

AddHandler SerialPort2.DataReceived, _
AddressOf SerialPort2_DataReceived
'the name SerialPort2_DataReceived is arbitrary,
'but must match the handler name
```

BYTE RECEPTION

According to the Help files, the `Read` method should return with all of the bytes in the message. I could never get it to return more than a single character. At any rate, I prefer reading byte by byte because it gives me a greater feeling of control. Listing 2 shows the first stab at receiving bytes on `SerialPort2`.

Real-world applications can fall foul of the simple approach presented here. It could result in an endless loop. What if the bytes are not received as a spurt but are separated by a varying period of time? How do you know that the message is complete? Did an error occur? Let me expand slowly.

I could implement a timeout using the `ReadTimeout` property of the serial port object (as seen in line reception description), but this implementation appears to have some limitations. It does not appear possible to stop the timer, and it is unclear whether writing to the property restarts the timer. In many applications, I would like to restart the timer between each byte received. You need to introduce an external timer. To do this, you need to add a timer control to the project form in the same way that you added the serial ports.

When the timer reaches its termination count, it generates an event. You then need to know how to communicate the results of this to some other process. The simplest way to intercommunicate between them is to use external variables, and you can certainly do that. If you're like me, you come from a world where many variables, if not all, are global. Despite that, I get the distinct feeling that global variables are frowned upon in today's software community. I decided to create an object that had properties needed to indicate the status of the timer to any code.

Before you create an object, you need to create a class (a prototype of the object). After you have the class, you can instantiate it to create one or many objects. The benefit is that you deal with them in the same manner. All that changes is the name. As an example, look at how you treat the serial port. The only difference is whether you are dealing with `SerialPort1` or `SerialPort2`.

The class definition lives outside the `Form` class in the program code. I am not going to get into why the `Form` is a class and there isn't an instance of it. Philosophy is not my strong suit. If you look at the project, you will see how the `Timeup` class is created (at the end of the project) and how it is instantiated as an object using the `dim` instruction before any procedures or functions that may use it.

As I mentioned before, in any protocol, the length of the message is fixed, implied, or somehow embedded in the message. For the sake of this example, assume that the second byte received contains the number of bytes. If you have a fixed byte length protocol, then the check that reads this byte need not be done. In addition, you can change the procedure to a function so that the read message can be returned for processing in the calling procedure. The function will return an array. I can reserve the first byte (array member (0)) of the array as a status byte. If it is zero, then the message is valid. A nonzero indicates a fault, and it can have different numeric values associated with different faults like checksum error and timeout. I use only a timeout code with a value of one. The incoming message starts at array member (1).

Listing 3 shows how all of this pans out. If the application has a longitudinal check on the bytes like a CRC or checksum, and provided the program considers for the validity of this check, then this approach will be sufficient without resorting to the built-in error detection.

EVENT-DRIVEN RECEPTION

It would have been nice to have had

Listing 6—*This is an example of the `DataReceived` and `ErrorReceived` events. Be sure your code is more robust than the simple case presented here, especially when deciding when a full message has been received.*

```
Public Sub SerialPort2_ErrorReceived( _
ByVal sender As Object, _
ByVal e As System.IO.Ports.SerialErrorReceivedEventArgs) _
  Handles SerialPort2.ErrorReceived
  'NOTE: <space>_ is a line continuation
  Dim iI As Integer
  iI = SerialPort2.BytesToRead
  'read the number of bytes received to help with understanding
  'the failure of discard in buffer to work. indicate error
  IF2.ErrorDetected = True
  'it is conceivable that the reception event will not fire,
  'so set that flag as well
  IF2.MessageArrived = True
If e.EventType = IO.Ports.SerialError.RXParity Then
  'you could look for all the errors e.Event is a passed parameter
  MsgBox("Parity Error", MsgBoxStyle.OkOnly, "Transmission error")
  SerialPort2.DiscardInBuffer()
  'beware of the style you use to open the serial port
  'My.Computer.Ports.OpenSerialPort method will generate a fault
  'condition here stating the the port is not open.
 End If
End Sub
Public Sub SerialPort2_DataReceived( _
ByVal sender As Object, _
ByVal e As System.IO.Ports.SerialDataReceivedEventArgs) _
   Handles SerialPort2.DataReceived
  Dim bytReceived(5 - 1) As Byte
   'dim includes the 0 case
  Dim iJ As Integer
  Timer2.Interval = 1000 '500mS
  Timer2.Start()
  'inititae timeout
  Timeup2.boolExpired = False
  iJ = 0
  While SerialPort2.BytesToRead > 0 And Timeup2.boolExpired = False
    Application.DoEvents()
    bytReceived(iJ) = SerialPort2.ReadByte
    iJ = iJ + 1
  End While
  If Timeup2.boolExpired = False Then
    IF2.MessageArrived = True
    IF2.ReceivedMessage = bytReceived
    SerialPort2.DiscardInBuffer()
  Else
    SerialPort2.DiscardInBuffer()
    'because of the asynchronous nature the events this clear may
    'only have cleared the bytes to date, not all of them
  End If
  'now and error could have occurred. Because of the
  'asynchronous nature the events it may not be detected for a
  'while start a delay and after that check if an error was detected
  Timer2.Interval = 500 '500mS
  Timer2.Start()
  'inititae timeout
  Timeup2.boolExpired = False
  While Timeup2.boolExpired = False
    Application.DoEvents()
    'ensure the timer interrupts happen
  End While
  If IF2.ErrorDetected = True Then
    IF2.MessageArrived = False 'ensure no message is passed
  End If
  If IF2.MessageArrived = True Then
    'if there is a message- respond
    Send2Bytes(IF2.ReceivedMessage)
   End If
End Sub
```

an external device that communicated with the PC to help demonstrate the performance of the serial port within VB2005. I needed a simple device because if I had had to explain its operation, it would have been a distraction. In the end, I decided to extend the concept that I have used to date. Using both serial ports, `SerialPort1` would be the master that initiated the communications, and `SerialPort2` would be a virtual remote device. `SerialPort2`'s function is to merely take the received message received and return the bytes over the serial communication lines. If any error is detected, `SerialPort2` sends no return message. The "protocol" is simply a five-byte message. The first byte is $A5_{HEX}$. `SerialPort1` uses the techniques I have used until now. `SerialPort2` is event driven.

I had to create a new project called "Serial2." Why? When there is an event handler within the code, there is no way to disable it. If the event occurs, it is "handled" even if it pertains to something else like a different example. As with all event handlers in Visual Basic, if the handler doesn't exist, the event is ignored. The `setup` for the project is identical to "Serial1."

You can use the `DataReceived`, `ErrorReceived`, and `PinChanged` events independently. There is no precondition for the use of any. The events are asynchronous and so there is an issue of how to pass data from one to the other. As you can see in Listing 4, I again resorted to an object. Listing 5 shows the code that was added to the `OpenPorts` procedure.

The code for both handlers is shown in Listing 6. After the port is initiated, no code ever needs to treat the port directly, but it can access the `IF2` object to see the status. One of the problems with the Windows/Visual Basic approach to events is that they are asynchronous. You're never sure of the sequence or the relative times of each event. As a result, it is impossible to tie an error to a particular byte so that error correction is not feasible. It's possible that a message is read, processed, and completed by the `ReceivedMessage` event before the `ReceivedError` event is processed. It isn't possible to read the error properties directly. Microsoft, if you ever get to read this, please make these properties accessible. My solution was to add a delay after the message was received to allow the error (if it has been detected) to be processed. Hardly elegant!

BREAK DETECTION

It's possible to change the value of the RS-232 control outputs and read the RS-232 control inputs directly via the associated properties. The same is true of break detection and generation. It's possible to have the change of any of the control inputs or the break detection to generate a `PinChanged` event. The ring indicator (RI) can also generate an event, although it isn't a property of the serial port object.

You can find an example of how to approach this in the `SerialPort2_PinChanged` event in the project. As I already mentioned, you don't need to use an `AddHandler` declaration, but you could insert it for good form.

SHORTCOMINGS

The VB2005 `SerialPort` class has a few shortcomings. For instance, the control placed from the toolbox doesn't link to the object, and it has only one event. In addition, event-driven and asynchronous events lead to uncertainty as to whether a message has been successfully received and whether the process is completely over. Clearing the In buffer doesn't always clear all of the bytes because some may arrive after the command is executed. There is no buffer of faults, and there is no linkage to bytes so that an error can't be associated with a particular byte. Furthermore, it isn't possible to read faults directly.

There are other shortcomings to consider. Different methods of the `SerialPort` object may not work depending on how the control is initiated (opened). The `Receive` method doesn't work for bytes. Also note that the incoming buffer size (less than 4,096) is ignored (according to the documentation), so it isn't possible to narrow down the buffer so that an error is associated with a given byte. Finally, there is no end-of-transmission event so that the PC resources are dedicated to writing the series of bytes or characters to the serial port while doing nothing else.

MODBUS INTERFACE

Although the abilities of the serial port model in VB2005 are severely limited, you can still use it for many different applications. This article stands on its own, but I have used VB2005 and this implementation of the serial port to create a Modbus interface. In an upcoming article, I will describe Modbus and a VB2005 interface to Microsoft Excel. ◼

Aubrey Kagan is a professional engineer with a B.S.E.E. from the Technion—Israel Institute of Technology and an M.B.A. from the University of the Witwatersrand. He works at Emphatec, a Toronto-based design house for industrial control interfaces and switch-mode power supplies. In addition to having written several articles for Circuit Cellar *and having published ideas in other periodicals, Aubrey wrote* Excel by Example: A Microsoft Excel Cookbook for Electronics Engineers *(Newnes, 2004).*

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2006/197.

REFERENCES

[1] R. Ashby, "COM Port Woes," *EE Times*, Feb. 2002, http://archive.chipcenter.com/eexpert/rashby/rashby057.html.

[2] Microsoft Corp., "Visual Basic 2005: Overview," http://msdn.microsoft.com/vstudio/express/vb/.

[3] Microsoft Corp., "How to Access Serial and Parallel Ports by Using Visual Basic .NET," 823179, 2006, http://support.microsoft.com/default.aspx?scid=kb;en-us;823179&Product=vb6#EWACAAA.

[4] C. Cavalli, "Serial Communication With VB.Net," www.codeworks.it/net/VBNetRs232.htm.

RESOURCE

Innovatic, "Serial COM Port Communication," 2006, www.innovatic.dk/knowledg/SerialCOM/SerialCOM.htm.

by Alan Bate

Third Overtone Crystal Clock Oscillator

Alan describes a requirement for a low-jitter, 54-MHz third overtone design. It's a good solution for high-volume manufacturing.

Conventional quartz crystals are limited in their fundamental frequency range to around 20 to 35 MHz. This is a physical process limitation that requires thinner and more fragile quartz wafers with increasing frequency that results in lower manufacturing yields due to wafer breakage.

To obtain higher frequency operation, conventional crystals are often operated in a harmonic or overtone mode. The idea is to achieve frequency multiplication while maintaining the high-accuracy, high-stability, low-temperature drift associated with quartz resonance. The current trend to use phase-locked loop (PLL) frequency multiplication has phase jitter limitations. This can be a problem with increasing use in modern digital chips with internal core clocks of several hundred megahertz that are derived from a basic low-frequency system clock.

In the set-top box design field in which I work, achieving high clock stability at low cost is an issue. In this article, I focus on a requirement for a low-jitter, 54-MHz third overtone design for high-volume manufacturing. Hence, preset "tweaking" is not an option.

QUARTZ CRYSTALS

Quartz crystals are unique components with exceptionally high Q factors, typically ranging from 10,000 to higher than 100,000. This is three to four orders of magnitude higher than what is obtainable from most real-world LC resonant circuits. Even low-loss, air-cored coils barely reach Q factors of more than 200. The only Q challenger to the crystal would be a microwave cavity resonator machined out of solid aluminum.

The crystal industry began in the 1930s to serve ham radio operators, but it grew during World War II to meet the huge military demand for radar and telecommunications technology. This has enabled today's industry to produce crystals that have been ground, polished, and trimmed to high orders of dimensional and frequency accuracy. By the judicious slicing of the crystal with respect to its atomic lattice, an entire range of temperature coefficients is obtainable. Crystals are typically packaged in resistance-welded, hermetically sealed leaded or surface-mount metal cans that are either vacuum-sealed or filled with dry nitrogen. For less than $1 for high-volume orders, who can beat that? Quartz crystals have excellent frequency stability because we are essentially talking about the dimensional stability of a piece of rock. Fortunately, the crystals also have excellent frequency stability.

A typical commercially available crystal will age at a rate of ±5 ppm per year. (Aging occurs mostly in the first year. The rate improves considerably when soldered cans are replaced by resistance-welded cans.) Such a crystal has an operating range from 0° to 70°C and a frequency tolerance between 10 and 100 ppm. The maximum equivalent series resistance (ESR) ranges from 35 to around 800 Ω depending on the frequency. For the frequency range discussed in this article, a maximum ESR of 40 Ω is typical. Its load capacitance is 12 pF. The drive level is between 100 µW and 2 mW (maximum) depending on the package and type.[1]

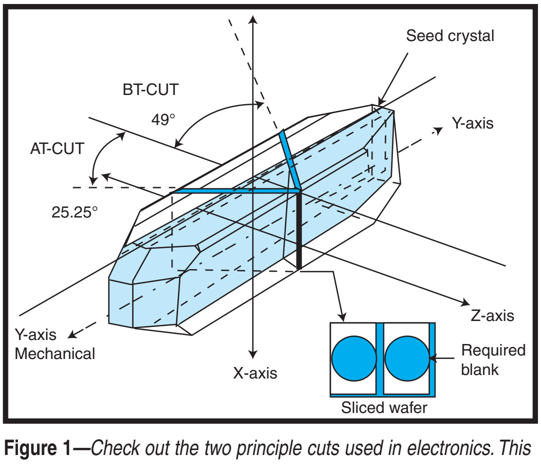


Figure 1—*Check out the two principle cuts used in electronics. This cultured quartz Y-bar is grown from seed crystal. Circular blanks are machined from the cut wafers using only the grown crystal regions.*

CRYSTAL OPERATION

Quartz or silicon dioxide (basically a crystallized version of sand) is a material that has polarized or permanently charged molecules. All such material exhibits the piezoelectric effect. If the material changes its physical dimensions from an applied pressure, then a corresponding electric field is developed at opposite faces of the material because of the charged molecules aligning so that their charges add. The process also works in reverse so that an identical applied electric field will produce a corresponding mechanical distortion of the crystal.[1]

To ensure consistency in quality and cost, modern crystal manufacture uses pure cultured quartz grown from a seed crystal. This is similar to the semiconductor industry's manufacturing of pure silicon. After slicing the quartz bar into wafers, the quartz is accurately ground and shaped into a disc or a plate. It's then polished to create an efficient miniature high-Q mechanical resonator.

There are three main modes of vibration depending on the type of slice or cut taken relative to the quartz bar longitudinal or Z-axis, which is referred to as the optical axis: Longitudinal mode, Low-Frequency Face-Shear mode, and High-Frequency Shear mode.

The precise cutting angle is controlled at high levels of accuracy with x-ray diffraction techniques. The most commonly used cuts have names like AT-Cut, which produces an S-shaped frequency temperature characteristic, and BT-Cut, which has a parabolic characteristic. Figure 1 illustrates the two principle cuts from the cultured bar. The quartz bar is a sandwich in which the seed crystal is the filling and the bread slices are pure Z-grown synthetic quartz, which are known as lumbered M-bars.

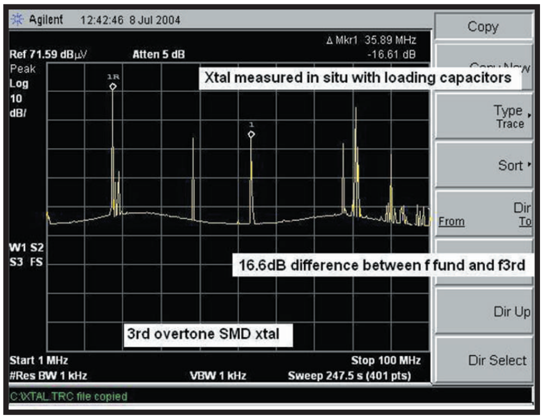


Photo 1—*Check out this network analyzer plot of the transfer function between the oscillator terminals with the crystal in-situ and the tank circuit removed. The spurious resonances around the fundamental and higher overtones are easy to see.*

SPURIOUS MODES

In practice, spurious resonances are also produced. Spurious modes are additional unwanted parasitic resonances to the fundamental. These additional modes are normally close to the desired overtone. Photo 1 shows a network analyzer frequency sweep of an in-circuit 18-MHz fundamental crystal and its associated overtones. The careful design of the quartz's diameter, thickness, and mounting (along with the tight process control of the polished surface finish) all ensure that the spurious modes are sufficiently suppressed. This helps prevent an oscillator from starting up and locking to the wrong frequency.

The older more bulky HC49 style of through-hole crystals with their larger quartz mass have lower phase noise. Also they have fewer problems with spurious operation when using higher power levels and lower loss than the more recently introduced low-profile surface-mount welded can varieties.[1]

The expensive inverted-mesa process has helped manufacturers overcome the fundamental frequency limit problem associated with conventional crystal manufacturing. This process generates thin quartz through the chemical milling or etching of a "well" on opposite faces of the quartz blank, which is surrounded by a thick ring of unetched quartz for strength. At present, inverted-mesa crystals have a fundamental frequency range of up to approximately 500 MHz.[1]

The mesa process is not without its problems, though. The smaller mass of quartz means there are more residual contaminants.

EQUIVALENT CIRCUIT

If you examine an impedance plot for a typical crystal, you'll see that the

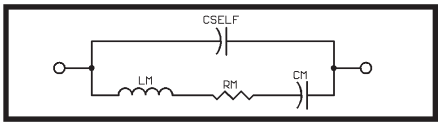


Figure 2—*This is a simple model of a crystal's impedance seen at the fundamental mechanical resonance. L_M, C_M, and R_M are apparent values of LC and R that represent the mechanical piezoelectric performance. The ESR is not shown, but it's made up from R_M and the dielectric loss in the quartz self capacitance.*

crystal has close series and parallel resonances at the fundamental and all the overtones (see Figure 2). There are two ways in which the circuit can resonate. One is from the resonance of the motional inductance (L_M) representing the quartz mass combined with the capacitance (C_M) representing the quartz compliance or elasticity, which provides a series resonance. At resonance, the inductive and capacitive reactances cancel, thereby reducing the series impedance to the equivalent series resistance.

The motional resistance (R_M) is mainly due to frictional losses. In series resonance, you have a low impedance, making the crystal resonance nearly independent of any external circuit. The other mode, parallel resonance (or anti-resonance), provides a high dynamic resistance at resonance. This occurs when the combination of motional inductance (L_M) has an inductive reactance that equals the total effective capacitance seen by the crystal (i.e., the self capacitance combined with the oscillator circuit and board parasitic capacitance). Therefore, crystals designed to operate in the parallel resonance mode have a specified load capacitance. As a result, the oscillator design must provide this load capacitance for correct frequency operation. In practice, the discrete capacitor values are reduced from their nominal values to allow for any stray circuit capacitance.

The crystal's quality factor (Q), equivalent series resistance (ESR), and motional resistance (R_M) are derived from 50-Ω network analyzer measurements. The equivalent series resistance represents the total loss, including motional friction loss, and the quartz dielectric loss in the quartz self-capacitance and the RF loss in the crystal connections.

The inductance can be derived from the measured Q factor, the specified ESR, and the C_{SERIES} from the series resonant frequency (F_R). For example, assuming a typical Q factor of 100,000 and a specified maximum ESR of 40 Ω, then:

$$Q = \frac{\omega L}{ESR}$$

You can rearrange the formula in terms of the motional inductance:

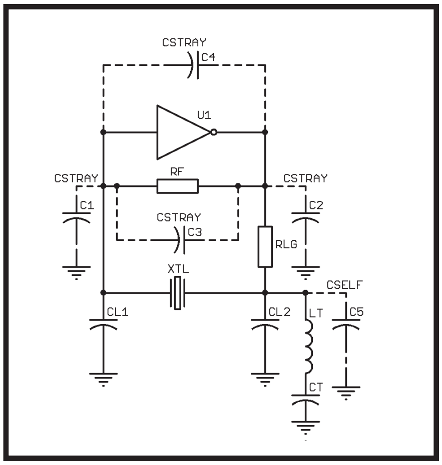


Figure 3—*This is a conventional asymmetric third overtone circuit.*

$$L = \frac{Q \times ESR}{\omega} = 100{,}000 \times \frac{40}{2\pi F_R}$$

In this project, the fundamental frequency is 18 MHz. Therefore, the effective motional inductance is around 35 mH. The crystal resonant frequency is:

$$F_R = \frac{1}{2\pi\sqrt{L_M C_M}}$$

where L_T and C_T represent the motional inductance and capacitance. The motional capacitance (C_M) is:

$$C_M = \frac{10^{15}}{\left(2\pi 18^{6}\right)^{2} \times 35.37 \times 10^{-3}}$$

The model values can be applied to a SPICE circuit simulation. The effective motional capacitance for series resonance is 2.2 femtofarads. (Yes, that's 10^{-15}.) These are typical crystal model values.

OSCILLATOR DESIGN

Digital IC clock oscillator circuits invariably use a Pierce oscillator, which is a derivative of the Colpitts oscillator. The Pierce topology lends itself well to CMOS IC design. The gain element need only be a single high-speed logic inverter with low propagation delay. The intrinsic IC pin and PCB capacitances to ground can all be conveniently absorbed into the required crystal loading capacitance. The crystal merely replaces the inductor in the original Colpitts circuit. Figure 3 shows a Pierce circuit modified for overtone operation with the addition of an LC tank circuit.

Because of the high gain of the logic inverter, you can safely assume there will be enough loop gain for oscillation. Therefore, you need to consider only the phase criteria. For oscillation, the required total loop phase shift of 360° is met by the inverting amplifier's nominal phase shift of 180°. The parallel resonating crystal and loading capacitors form a pi network referenced to ground, adding another nominal 180° of phase lag. The lag resistor (R_{LG}) and the crystal loading capacitor (C_{L2}) add a small extra phase lag to make up for any shortcomings in the overall loop phase shift due to circuit losses.

In practice, the crystal operates at or near the peak of the positive reactance at the onset of parallel resonance or anti-resonance, not the theoretical true resonant point of zero phase shift, where the crystal impedance becomes a high dynamic resistance. But this is academic when working with crystal Q factors of 10,000 and higher.

Any propagation delay through the inverter is absorbed into the overall phase shift by reducing the phase lag of R_{LG} and C_{L2}. The CMOS gate output impedance adds to the required series resistance R_{LG}. The R_{LG}-C_{L2} pair also forms a low-pass filter that provides the high-frequency filtering of noise and harmonics. It also helps to limit the crystal drive power, typically specified to be 1 mW for the larger HC49 types. This is important for avoiding premature aging, unwanted overtone, and even fractures. The loading capacitors C_{L1} and C_{L2} are in series across the crystal along with all of the stray board capacitances to ground and the IC amplifier port capacitances.

THIRD-OVERTONE OPERATION

Although any crystal can be operat-

ed in an overtone, crystals manufactured for overtone applications must have low loss and be cut to enhance the mechanical resonance at the desired overtone. Therefore, overtone crystals are special. Typically, the maximum ESR of the crystal is specified as 40 Ω. This limit has proven to be sufficiently low loss without affecting production yield. The overtones can be simply modeled as additional series resonant circuits to the fundamental series resonance.

To achieve overtone operation, the circuit must suppress the crystal fundamental mode and ensure an overall loop phase shift of 360° with a gain greater than 1.0 at only the required overtone. There are three solutions, but only one is viable for mass production.

The first two methods both involve band-pass tuning the loop to provide gain at only the overtone. This can be done by making the inverter a tuned amplifier or adding the band-pass filter (LC tank circuit) at one of the inverter ports to filter the crystal network. It requires the filter to have a high dynamic impedance that is resistive at the overtone frequency (i.e., the tank circuit would need to be high Q and tuned to resonate precisely at the overtone).

The filter circuit therefore requires high-Q low-loss components with tight tolerances to define the circuit by design. If low-loss components with loose tolerances are chosen, then the circuit would require a manual calibration step in the manufacturing process to set the tank resonance.

You would need a high-quality inductor with a self-resonant frequency well above my 54-MHz requirement. This will limit any surface-mount chip inductor value below 1 µH and therefore limit the possible Q. Air cored coils would be far better, but they are more bulky and not generally convenient for high-volume manufacturing.

The first two solutions are unacceptable for commercial mass production. The third method is for phase control using an LC tank circuit. The requirements can be met without the aforementioned tolerance issues by simply providing a network that adds a positive reactance +jX (or inductive) at the fundamental frequency of 18 MHz. This prevents oscillation by reducing the overall phase lag below the 360° phase criteria. The network should also provide a nearly –90° (capacitive) phase lag at the overtone frequency.

This is met by a simple LC parallel resonant circuit circuit formed by C_{L2} and L_T. The resonance is situated midway between the fundamental and the overtone frequency (i.e., at two-thirds of the third overtone frequency). The LC circuit is not critical in its resonant frequency, but it needs sufficiently high Q to ensure the circuit tolerances are not dominated by inductor parasitic losses and can be defined by design.

With the LC resonance at two-thirds of the third harmonic frequency, and assuming an inductor Q of only 20, the phase lead at the fundamental is approximately:

$$\cong \tan^{-1} Q\left(\frac{f_{RESONANCE}}{f} - \frac{f}{f_{RESONANCE}}\right)$$

$$\cong \tan^{-1}\left[20 \times \left(\frac{36}{18} - \frac{18}{36}\right)\right]$$

$$= 88° \text{ phase lead, inductive}$$

This is a healthy phase lead at the fundamental preventing any oscillation.

At the overtone, the phase has gone capacitive, approaching a –90° lag. In other words, the tank circuit's presence virtually disappears and the circuit reverts back to the crystal's half loading capacitance (C_{L2}), thereby enabling normal oscillator operation (see Figure 3).

INDUCTOR CONSIDERATIONS

For a quick ballpark calculation of inductance, the required inductance is:

$$L_T = \frac{1}{\left(2\pi \frac{2}{3} f_{OVERTONE}\right)^2 (C_{L2} + C_{SELF})}$$

Allowing 5 pF for inductor self capacitance (C_{SELF}) and strays, the nominal 32-pF loading capacitance (C_{L2}) is reduced to 27 pF:

$$\text{Required inductance} = \frac{1}{\left(2\pi \times \frac{2}{3} \times 54 \text{ MHz}\right)^2}$$

$$\times (27 \text{ pF} + 5 \text{ pF})$$

$$= 611 \text{ nH}$$

Standard inductor values follow the E24 series, so choose 560 nH as the nearest preferred value. The nominal tank resonant frequency is not critical and will only change by the square root of the deviation in inductance from the nominal design value or about 4.5%.

Ideally, the inductor self-resonant frequency should be appreciably above the operating frequency. Looking at the data for a typical surface-mount chip inductor, the Taiyo Yuden LEM2520 series of 0603-style chip inductors easily fulfill the requirement with such a modest inductance. The LEM2520 has an inductance of 560 nH (10% tolerance), a minimum self-resonant frequency of 300 MHz, a minimum quality factor of 30 (measured at 25.2 MHz), and a DC resistance of 0.25 Ω.

Note that Q = ωL/ESR. Assuming *Q* is constant (it isn't), the ESR at the inductor test frequency of 25.2 MHz is:

$$\frac{2\pi \times 25 \times 2 \times 10^6 \times 560 \times 10^{-9}}{30}$$

The ESR is 3 Ω. Strictly speaking, this applies at only a 25.2-MHz test frequency.

The chip inductor self capacitance (C_{SELF}) is derived from inductor self resonance:

$$C_{SELF} = \frac{1}{\pi^2 L}$$

$$= \frac{1}{\left(2\pi \times 300 \times 10^6\right)^2 \times 560 \times 10^{-9}}$$

$$C_{SELF} = 0.5 \text{ pF}$$

But soldered onto a PCB, this will be more like 1 to 2 pF because of board parasitic capacitance.

C_T is simply a DC block. It must be large enough to bring the unwanted series resonance with L_T well below and out of the way of the crystal's fundamental resonance. When C_T is 10 nF, the tank series resonance will be an acceptable value:

$$\text{Tank series resonance} = \frac{1}{2\pi\sqrt{L \times C}}$$

$$= \frac{1}{2\pi\sqrt{560 \text{ nH} \times 10 \text{ nF}}}$$

$$= 2.13 \text{ MHz}$$

At the crystal's fundamental frequency, the tank circuit will be above its series resonance and have a net inductive reactance. This adds a healthy 80°-plus phase-lead to the loop, thereby preventing the overall 360° phase shift requirement.

R_F provides DC feedback to bias the CMOS inverting logic buffer into a class A operation to act as a quasi-linear amplifier. It remains a sufficiently high-value resistance so as to not unduly load the crystal circuit. Typical values are from 1 to 22 MΩ. This resistance is often provided internally with IC oscillator circuits.

At the overtone frequency, the LC tank will have gone through resonance and become nearly all-capacitive. Consider C_{L2} and L_T. The loading capacitance C_{L2} will be partly neutralized by the inductance. The inductor can be viewed as a negative capacitor, reducing the capacitance of C_{L2}. Let's call the net reactance formed by L_T, C_{L2} $X_{EFFECTIVE}$, the two reactances in parallel. Now for just two reactances in parallel:

$$X_{EFFECTIVE} = \frac{X_{PRODUCT}}{X_{SUM}} = \frac{X_{CL2} \times X_{LT}}{X_{LT} + X_{CL2}}$$

Thus:

$$C_{EFFECTIVE} = \frac{1}{\omega X_{EFFECTIVE}}$$

Therefore, substituting for $X_{EFFECTIVE}$:

$$C_{EFFECTIVE} = \frac{\left(\frac{1}{j\omega C_{L2}} + j\omega L_T\right)}{j\omega\left(\frac{1}{j\omega C_{L2}} x j\omega L_T\right)}$$

This simplifies to:

$$C_{L2\,EFFECTIVE} = \left(C_{L2} - \frac{1}{\omega^2 L_T}\right)$$

The inductance acts as a "negative capacitance," reducing the C_{L2} value by:

$$\frac{1}{\omega^2 L_T}$$

Ignoring the inductor for the moment, the loading capacitance seen by the crystal is simply C_{L1} and C_{L2} in series. The effective capacitive load is then:

$$\frac{1}{C_{LOAD}} = \frac{1}{C_{L1}} + \frac{1}{C_{L2}}$$

$$C_{LOAD} = \frac{C_{L1} \times C_{L2}}{C_{L1} + C_{L2}}$$

C_{L2} is modified to $C_{EFFECTIVE}$. The actual C_{L2} must be the following to accommodate the partial neutralizing by the inductor:

$$C_{EFFECTIVE} + \frac{1}{\omega^2 \times L_T}$$

The usual design procedure is to equally split the loading capacitors C_{L1} and C_{L2}:

$$C_{L1} = 2 \times \text{specified crystal load}$$

$$C_{L2} = 2 \times \text{crystal load} + \frac{1}{\omega^2 \times L_T}$$

My 54-MHz crystal requires 16 pF of loading. As a result, C_{L1} is 32 pF. But at the third overtone, the actual C_{L2} must be the following due to the presence of the inductor:

$$C_{L2} = 32\text{ pF} + \frac{1}{\omega^2 \times L_T} = 32\text{ pF} + \left[\frac{1}{\left(54 \times 10^6 \times 2\pi\right)^2 \times 560 \times 10^{-9}}\right]$$

$$= 32\text{ pF} + 15.51\text{ pF} = 47.5\text{ pF}$$

Now you adjust for the inductor self-capacitance (C_{SELF}) of 5 pF and any strays at the circuit junction. Hence, the actual discreet value of C_{L2} is:

$$C_{L2} = 47.5\text{ pF} - 5\text{ pF} - C_{STRAYS}$$

I would choose 39 pF for an initial E24 design value.

This drops the tank resonance by roughly √32/47, but this does not matter in practice because the phase lead at 18 MHz has such a large margin. A SPICE simulation with an inductor of 560 nH and 7 Ω of ESR showed a phase lead of +70° with the 15.5 pF added at C_{L2}. The tank resonance had fallen to around 32 MHz.

Consider the 560-nH inductor. This is a lower, more suitable value for a chip inductor than the differential design I'll cover later. I chose the nearest E24 value of 560 nH.

Finally, because of the specific PCB layout, the value of R_{LG} is adjusted during development for a reliable start-up and crystal drive power limitation. Monitoring the crystal current with a wideband clip-on current probe and oscilloscope checks the crystal power dissipation. Use the smallest possible wire loop for the probe.

I use a simple equation for determining crystal power:

$$\text{Crystal power} \approx (\text{RMS crystal current}) \times (\text{RMS drive voltage at the crystal})$$

Because of the high Q filtering, the crystal current is a quasi-sine wave. This enables you to derive the root mean square voltage and current values by reading the peak-to-peak values displayed on the oscilloscope.

FREQUENCY ACCURACY

The circuit must be adjusted to the specified loading capacitance for the crystal to operate at the correct frequency. The total effective loading capacitance will be affected by all of the discrete capacitors plus all of the stray capacitance from the crystal terminals to ground and between the crystal pins. Due to variations in board layouts and board dielectrics, the circuit will need to be trimmed by adjusting C_{L1} and C_{L2} in order to absorb the PCB strays into the effective loading capacitance.

With typical values for C_{L1} and C_{L2} of only 10 to 32 pF, it's important to offer no additional loading from any test equipment. Even the Tektronix and LeCroy active scope probes currently on the market have an input capacitance of around 1 pF, which is a noticeable percentage loading of the C_{L1} and C_{L2} capacitors.

To correctly calibrate the circuit, selected crystal samples should be obtained from the supplier, which are spot on nominal frequency and nominal loading capacitance. The capacitors also need COG or NPO dielectric with a tolerance of 1% or better, or they are selected for nominal capacitance using a precision LCR meter accurate to 0.1%

The inductor tolerance of 5% is not so critical. The tank resonance is proportional to the square root of the inductance. This reactance is virtually removed at the overtone frequency.

NO BUFFERED CLOCK

If there is no buffered clock port available from the IC, then the only acceptable way to test the circuit is to use a sniffer cable (coax with the first 5 cm of screening stripped back) with an accurate spectrum analyzer fitted with a TCXO-based clock. The typical initial frequency accuracy for these instruments is 0.5 PPM. The sniffer probe should be held 5 to 10 cm from the circuit so that there is minimal coupling. It may be useful to disable all of the digital circuitry and memory in the unit for this measurement.

The procedure involves setting the spectrum analyzer's center frequency to the nominal clock frequency (54 MHz in this example). First, set the resolution bandwidth to minimum and the amplitude sensitivity to maximum. Following this, turn on the video averaging and initially set the analyzer span to 10 kHz. Next, fix the sniffer cable (around 5 cm from the clock output pin). When you locate the signal, reduce the SPAN setting (the horizontal frequency range) to the minimum setting (typically 1 kHz). You should then adjust the spectrum analyzer's amplitude sensitivity for the signal peak to be between 80% and 90% of full scale. Finally, set the cursor to Peak Search and read the frequency.

BUFFERED CLOCK

If a buffered clock output is available, there should be no problem with loading. You need to use only the frequency/timer counter with a minimum resolution of eight digits. It should be fitted with a TCXO frequency standard of 3-PPM accuracy (e.g., the 12-digit, 225-MHz Agilent 53132A universal timer/counter fitted with a TCXO oven option).

Alternatively, you can use a low-cost 200-MHz frequency/timer counter like the Thurlby Thandar Instruments TF810. Use the external precision reference option. The reference should be precision 10 MHz at TTL levels and traceable to the National Physical Lab (NPL), which maintains the U.K.'s national standards.

You can also obtain a suitable precision reference source from a GPS time and reference receiver such as the now obsolete HP58503A or the currently available Symmetricon 58503B. In addition to GPS, you also have the option (in the U.K.) of using an AM off-air reference PLL receiver that's tuned to the Rugby or Droitwich long-wave AM carriers that are traceable to the NPL.

GOING DIFFERENTIAL

Crystal clock oscillators can be a source of EMC problems if you aren't careful with the layout. One well-known issue in the set-top box design field in which I work is RF radiation from the current loop formed by the crystal and loading capacitors. Laying the PCB signal traces to have minimal loop area and arranging the loading capacitors to be star-point connected to the ground plane can minimize the problem.

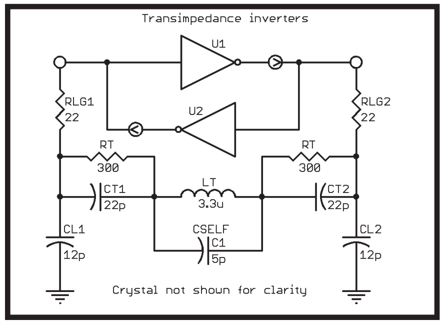


Figure 4—*The differential signal developed across the crystal gives a balanced signal around the supply rail and ground. The result is a low net signal current entering the supply and ground terminations.*

The common asymmetric circuit accentuates this problem by not only creating additional current loops with the LC tank circuit, but also placing an unbalanced signal current in the ground plane. This can add to the overall common-mode noise generated within the PCB.

Good commercial design today requires PCB layouts to have inherently low common-mode emissions in order to meet the international EMC thresholds. In ideal situations, this doesn't add the costs of extensive shielding and common-mode filter components.

I recently looked inside a competitor's product and found a poor PCB layout. The company had added expensive common-mode ferrite chokes to all of the internal cabling from the main digital board in addition to the usual common filtering at all of the digital I/O ports.

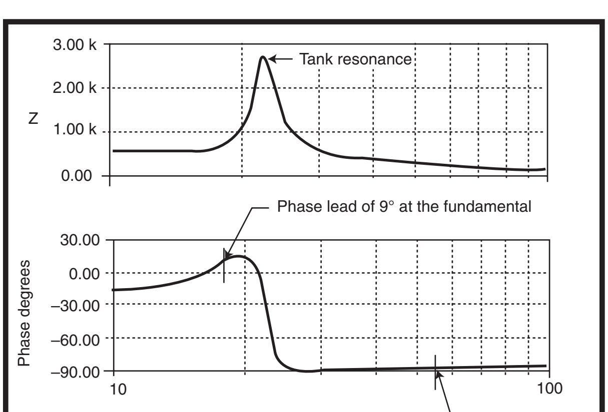


Figure 5—*Take a close look at this SPICE simulated impedance Bode plot of the tank network seen by the crystal.*

DIFFERENTIAL OSCILLATOR

The modern IC approach to the overtone oscillator is to use a balanced circuit that generates a differential signal across the crystal using a push-pull arrangement of CMOS inverting gain elements (see Figure 4).[2] The circuit uses transconductance instead of voltage amplifiers, where the external network impedance forms the load to each amplifier. This is necessary with the back-to-back configuration. Otherwise, with voltage amplifiers, each oscillator port would look into the low output impedance of one inverting amplifier and short the external network. Transconductance amplifiers are volt-in to current-out devices. As a result, they have high output impedance and avoid the problem.

By using star-point connections to ground and the internal IC supply rail, the net signal current over an entire cycle into the ground plane and the supply rail is zero. This reduces the need for efficient high-frequency local supply decoupling. In addition, the high output impedance of the transimpedance inverters provides good rail noise rejection just like the current mirrors of a typical linear IC. Figure 4 illustrates the concept without the amplifier bias circuit detail that makes it all practical.

DIFFERENTIAL LC TANK

The tank circuit performs the same function as in the asymmetric design with a slight refinement. The tank circuit is split symmetrically to preserve the balanced network impedance to ground at each crystal terminal. Remember: the common-mode rejection capability of any differential circuit is effective only by making the shunt impedances from each differential terminal to ground symmetrical. Splitting the series-resonant capacitor also helps (doubling the value of each tank capacitance). It makes it a more practical value that's less comparable to the circuit

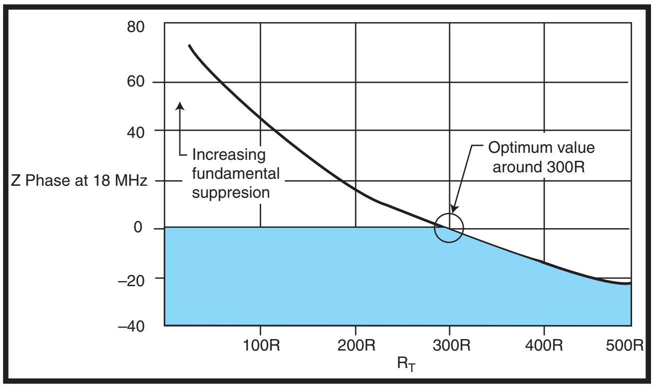


Figure 6—*I plotted the damping resistor value versus the LC tank phase. You can see the critical region for R_T.*

strays. The LC tank is wired across the crystal in keeping with the differential operation; otherwise, you need two matched tank circuits at each crystal terminal to ground to maintain a balanced network around ground.

One disadvantage is the inductance, which like the crystal sees the loading capacitors in series halving the effective parallel tuning capacitance and therefore doubling the required size of the inductance. Placing the inductor across the crystal adds another complication. If the inductor Q is too high, the tank resonance will become the dominant loop resonance and the circuit will oscillate at the LC tank frequency. With the Pierce circuit, the crystal merely replaces the inductor of the original Colpitts circuit. To prevent this and to provide a defined phase control at the suppressed fundamental, a damping resistor is placed across the series resonance capacitance.

This can be simplified to a parallel resonant LC network with lossy inductances (L) and capacitances (C). Although it's slightly simplified, this version still requires a lot of tedious math in order to derive the overall phase/frequency characteristic for which standard solutions can be found in older reference books. Therefore, it was decided to model the actual distributed network in SPICE for the sake of accuracy and development time. The values of the damping resistors were iteratively adjusted to give a small but adequate positive phase to suppress the fundamental at 18 MHz. This was also the maximum practical damping that could be applied to suppress the tank inductor Q.

Figure 5 is a Bode plot of the total network impedance seen by the crystal. The SPICE circuit simulation of the optimized damping gave 9° at 18 MHz and –85° at 54 MHz. The shortfall in phase lag was due to circuit loss. It was made up by the amplifier propagation delay and the RC lags R_{LG1}, R_{LG2}, C_{L1}, and C_{L2}. Figure 6 shows the damping resistor value R_T versus phase lead at the fundamental of 18 MHz. This indicates the optimum R_T value to be around 300 Ω.

My circuit required an inductance between 2.7 to 3.3 µH. This was just about practical for a surface-mount chip inductor because such a large chip inductance meant the self-resonant frequency was just high enough.

Datasheets typically specify self-resonance at 41 MHz. The modest Q factor of 45 was less of a requirement because the tank circuit had been damped anyway. However, this meant the circuit loss couldn't have been as well defined by the series damping resistors R_{T1} and R_{T2} as I had wished. Figure 4 is the differential overtone circuit.

GOOD PCB LAYOUT PRACTICE

With a clock frequency starting at 54 MHz, you can reckon on dealing with major harmonic levels to at least the tenth, or 540 MHz. In a mixed-signal system like a set-top box, this can cause intermodulation products in the RF front-end mixers that in turn produce unwanted in-band spurs. This is a real headache for analog video channels. You must lay out the PCB carefully in order to maintain the good EMC properties of the differential clock.

In order to fully exploit the noise-canceling properties of the differential design, the circuit must be symmetrical. The impedance to ground seen from each crystal terminal is identical. The loading and the stray capacitances to ground should be connected to a common star ground point for minimum noise injection into the ground plane. Finally, the loop formed by the crystal and loading capacitors should be laid out to achieve minimal loop area. This minimizes RF radiation to nearby circuits.

Figure 7 shows a tried-and-tested layout. The oscillator signal traces are brought from the IC at a minimum trace width of 0.005″ separated by the same amount in order to reduce board capacitance and (more importantly) the minimum current loop area. The loading capacitors are star-point grounded into the ground plane. Symmetry in the circuit layout is maintained in order to keep the inherent common-mode isolation of the differential circuit with balanced impedances to ground at each crystal terminal. The traces to the through-hole connections of the HC49 leaded crystal fold back on themselves in "bifilar-wire" fashion to cancel the trace inductance and avoid spur connections to the crystal terminals. Small spur lengths of high-frequency signal traces make good antennas!

Assuming you are using a multilayered board, the capacitive loading on the clock circuit is minimized by cutting a window through the ground planes. The edges of the ground planes, and any adjacent copper ground flooding on the signal and power layers, are stitched with vias to reduce RF ground impedance.

GUARDING THE CLOCK

When it comes to clock signal

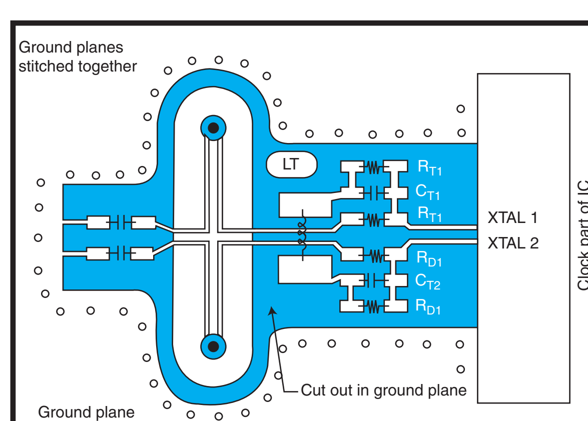


Figure 7—*A good PCB layout is important for signal integrity and low common-mode board emissions.*

integrity and low common-mode board emissions, a good PCB layout includes guard traces to each high-speed clock signal and high-frequency signals in general. Before the days of multilayer boards and ground (more accurately, reference) planes, there were double-sided boards without a ground plane. If you wanted to send a fast signal around a board, you had to provide a well-defined return signal path or guard trace. The guard trace was placed along the outgoing signal path or on the opposite side of the PCB shadowing the outward signal trace. This provided a low-inductance signal path with a smooth characteristic impedance. The low inductance comes from the field cancellation of the adjacent go and return signal currents. The guard trace is therefore the return path of choice for the high-frequency signal.

Guard tracing generates good signal integrity because of the well-defined transmission path with no impedance discontinuities and no reflections. Low crosstalk and low radiation are the result of the minimal circuit loop area.

The same advantages apply when you use guard tracing with a multilayer board. The guard trace provides the clock signal with a private return path, taking the high-frequency signal current out of the common and often the overworked system ground plane with its mere 35 µm of copper foil!

How do you ground the guard traces? Trial and error has shown that the best arrangement is to pin the guards at the near and far ends to the immediate local grounds. On a large system chip, this can mean locating the nearest associated ground pin to the IC's internal circuit.

Alan Bate (alan.bate@homecall.co.uk) holds a degree in electrical and electronics engineering from the University of Leeds. He is a senior hardware design engineer for Pace Microtechnology.

PROJECT FILES

For a list of useful resources, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2006/197.

REFERENCES

[1] MtronPTI, "Understanding Quartz Crystals," www.mtronpti.com/pdf/contentmgmt/understanding_quartz_crystals.pdf.

[2] Broadcom Corp., "BCM7038: Dual High-Definition Digital Video, System-on-Chip Solution for Cable, Satellite, and DTV," www.broadcom.com/products/Cable/Digital-TV-Solutions/BCM7038.

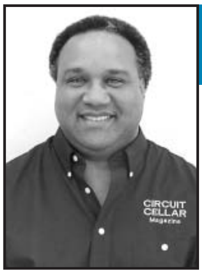
SOURCES

53132A Universal timer/counter
Agilent Technologies, Inc.
www.agilent.com

BCM7038 VLSI IC
Broadcom Corp.
www.broadcom.com

LEM2520 Inductor
Taiyo Yuden (U.S.A), Inc.
www.t-yuden.com

TF810 Frequency counter
Thurlby Thandar Instruments
www.tti-test.com



by Fred Eady

Ethernet on a Chip

This month, Fred describes his recent work with the ASIX Electronics AX11005 development kit. Using a Keil C compiler and some Digital Core Design hardware, he created a powerful Ethernet development suite that now enables him to produce some pretty exciting Ethernet-based equipment.

I recall standing in a room full of Token Ring equipment some years ago and one of my coworkers looked at all of the new stuff we had just installed and said, "Ethernet is dead." We both should have turned around and took a closer look at the new Ethernet stuff the other guys were installing on the other side of the room. Why? That "dead" Ethernet equipment is still in use to this day, while the Token Ring stuff is nowhere to be found.

Ethernet technology has come a long way since then. It has become faster and can travel over many types of media. In essence, the core of Ethernet technology hasn't really changed that much over the years. However, the way Ethernet is used changes daily. For instance, Ethernet is now being incorporated into embedded devices just as regularly as RS-232. In fact, Ethernet is more and more becoming a familiar resident on microcontroller silicon. We're all watching the Ethernet-on-microcontroller revolution happen with all-in-one parts emerging from Microchip Technology, Freescale Semiconductor, and ASIX Electronics, to name only a few.

Thanks to the folks at ASIX Electronics. During this go-round, I got to take a look at its new AX110*xx* development kit, which is based on the new AX11005 single-chip microcontroller with TCP/IP and 10/100-Mbps fast Ethernet MAC/PHY.

ETHERNET SoC

The AX11005 is designed to do a bit more than just service Ethernet frames. The core of the AX11005 is a fast (up to 100 MHz) 100% binary-compatible clone of the venerable 8051. The 19-bit flat program addressing mode uses the 80C390 instruction set to enable bankless 0- to 512-KB access of on-chip program flash memory. The AX11005 can also execute instructions using 16-bit large program addressing mode, which calls on the 80C51 instruction set. A 16-KB SRAM area is used for program flash memory mirroring. In addition, 32 KB of on-chip user SRAM is allocated in the external data memory area.

It almost seems as though the AX11005's 10/100 Ethernet capability is an afterthought because the microcontroller supports a 1-Wire interface, three RS-232 ports, I^{2}C, PWM, counter/timers, three SPI masters, a SPI slave, and 16 bits of general-purpose I/O. The idea behind the multitude of communications interfaces is to put the AX11005 into an application space that can convert RS-232 serial data to Ethernet frames or communicate with a ZigBee radio via the SPI port and bidirectionally transfer data between the ZigBee PAN and Ethernet link. With its many ways of communicating, the AX11005 is powerful enough to act as a network processor serving other link-attached or physically attached microcontrollers.

A single 3.3-VDC power source powers the AX11005. Keeping with the system-on-a-chip (SoC) concept, an on-chip 1.8-VDC regulator feeds the CPU core. The

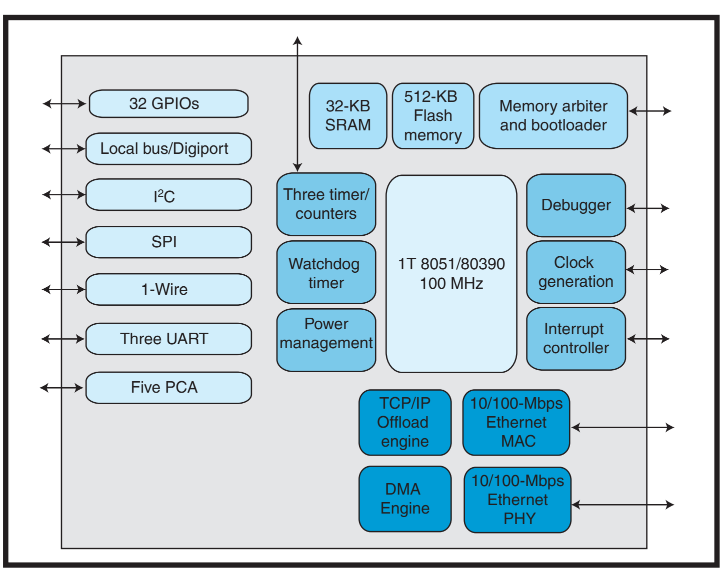


Figure 1—*If you ignore the TCP/IP offload engine and the 10/100-Mbps MAC, the AX11005 looks like a high-end microcontroller. Here you see 32 bits of general-purpose I/O, which is correct for the AX11005's big brother, the AX11015. The AX11005 supports only 16 bits of I/O.*

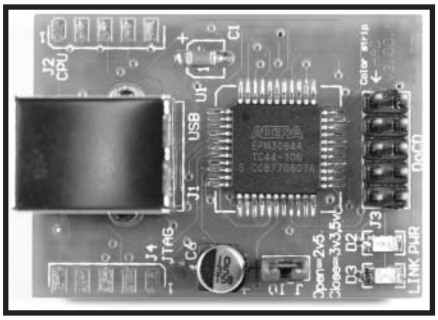


Photo 1—*The HAD2 is powered by the USB port and manages communications between the AX11005 on-chip ICE circuitry and the debugging application running on a companion PC.*

AX11005 requires only a single 25-MHz crystal for all of its internal clock generation processes. There's even a built-in, on-chip, power-on-reset circuit.

The AX11005 will most likely find itself embedded in an install-it-and-forget-about-it environment, so simple methods of upgrading the SoC's firmware must be available. The AX11005 can accept updates by way of the Ethernet port or the UART. You can also use the UART to program the device.

I'm particularly interested in the TCP/IP offload engine (see Figure 1). But before I delve into that, let's look at what it takes to support the AX11005.

I'VE BEEN HAD!

From what I've seen thus far, the AX11005 is a pretty heavy little microcontroller. It would be close to impossible to attempt to apply an application to the AX11005 hardware without a good debugging system. The AX11005 doesn't disappoint. It comes standard with an on-chip in-circuit emulator feature that's designed to interface with a piece of external ICE hardware. The task assigned to the ICE is to manage communications between the AX11005's on-chip ICE circuitry and a software debugger application running on a PC. The external ICE hardware that mates with the AX11005's on-chip ICE circuitry is called the Hardware Assisted Debugger (HAD). The latest version of HAD is called HAD2. As you can see in Photo 1, I've been HAD.

Powered by the USB port, the HAD2 uses USB Full Speed technology to interface the AX11005's internal debugging system to the debugging application running behind the USB interface on a PC. My HAD2 hardware is from Digital Core Design, which also supplied the Windows-based debugging application software. The HAD2 and Windows debugging software come bundled together. The HAD2 debugging software package is fully compatible with all existing 8051/80390 C compilers and assemblers. In addition to debug duty, the debugging software can be used as a software simulator too.

The HAD2 package was originally designed for SoC designers that needed the capability of debugging applications before they were to be committed to silicon. It's pretty obvious now that the AX11005 contains a Digital Core Design (DCD) 8051/80390 IP Core, because the AX11005's DCD IP Core is specifically designed to work with the HAD2 and the DCD Debug IP Core. This mix of IP cores and the HAD2 is called DCD on Chip Debug System (DoCD).

The Debug IP Core is a real-time hardware debugger that enables a full nonintrusive view of the on-chip registers, memories, and peripherals associated with the DCD IP Core, which is the 8051/80390 IP core in this case. The DoCD debugging system doesn't require any resources from the target. Thus, all of the target device's features and peripherals can be accessed during the debug phase. As you've already ascertained, the DoCD system is complete and needs nothing else in the way of features to make it better. But it does get better. You can also use the DoCD system to program the target device's program flash memory.

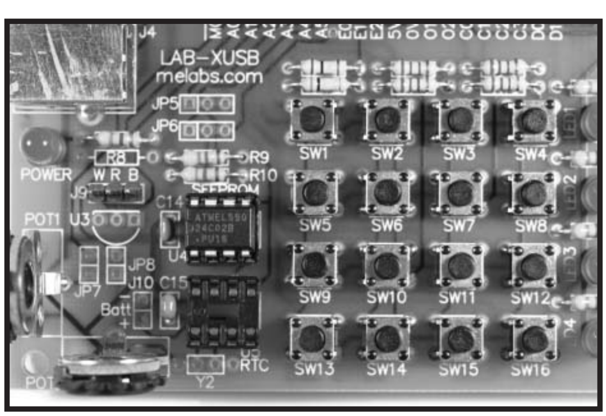


Photo 3—*The MicroEngineering Labs LAB-XUSB shown here was really designed to get you started with embedded USB. I was elated to see an I²C EEPROM socket on this board. It that meant I didn't have to fire up the soldering iron and build up the EEPROM circuitry from scratch.*

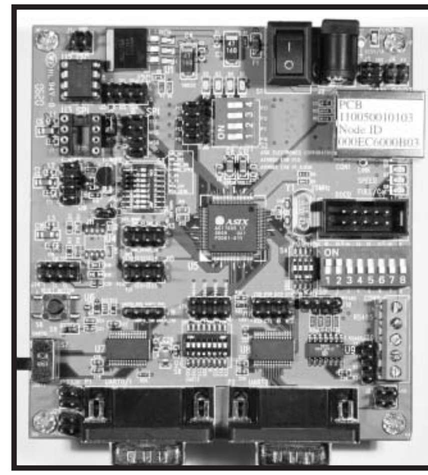


Photo 2—*This is one busy board. The HAD2 connects using the shrouded and keyed pins below the Ethernet magnetics. Move down below the eight-position dip-switch and note the inclusion of screw terminals to facilitate RS-485. The AX11005 boot EEPROM can be seen in the upper-left corner.*

AX110*xx* DEVELOPMENT KIT

I have an AX11005 part, so you can safely assume that all of the pre-silicon work on it has been done. That means you can use the HAD2 in conjunction with the AX11005's debug IP Core. To help facilitate faster prototype cycles, ASIX offers the AX11005 development kit.

My AX11005 development board is shown in Photo 2. All of the microcontroller's bells and whistles are accessible by pin or connector. The schematic diagram for the AX11005 development board is six pages deep. So, instead of trying to nudge the schematic into this column, I posted it on the *Circuit Cellar* FTP site for easy access.

Although the AX11005 is an awesome piece of silicon, the kit's real strength lies in the abundance of AX110*xx* software modules and utility programs. There is a source code module supplied for every AX11005 peripheral. AX11005 utility programs that come with the kit include a flash memory programming utility, HEX2BIN utilities for both

DOS and Windows, and a TFTP/DHCP server application (just to name a few).

My AX11005 development board came preloaded with a TCP/IP stack and sample code for what the ASIX folks call a "lightweight" TCP/IP stack. It came on the kit's CD-ROM. A comprehensive software user's guide goes into detail about connecting your application to the software modules' internal API function calls. For instance, to use the AX11005's TCP/IP offload engine to set the IP address, all that needs to be coded is a call to the `STOE_SetIPAddr()` function. `STOE` refers to an API call that works against the TCP/IP offload engine. The source code also uses `stoe` (lowercase) to delineate driver calls from API calls. Examples of the API and driver calls I've referenced are shown in the top part of Listing 1.

The only hardware change you will have to make to give the HAD2 access to the AX11005 is a bit change within the Atmel AT24C02B EEPROM. Bit 7 of the byte located at address 0x01 within the AT24C02B determines if the CPU debugger pins are muxed to the Ethernet status LEDs or to the HAD2. The default state of bit 7 is a one, which muxes in the Ethernet status LEDs. It's easy to lash up a little PIC I^{2}C interface and change the bit. I used a MicroEngineering Labs LAB-XUSB, which already has the socket laid in for an I^{2}C EEPROM (see Photo 3).

With a little bit of help from my Custom Computer Services PIC C compiler, I had the EEPROM bit banged to zero in a couple of minutes. The compiler is great for stuff like this because it's fully tilted toward the PIC architecture. It includes built-in PIC-specific functions that take most of the effort out of coding things like I^{2}C, SPI, and RS-232. My little bit-twiddling C code is shown in the lower portion of Listing 1.

KEIL'S BEEN HAD2!

The AX11005 software guide tells you up front that Keil's C development tools were used to craft the AX11005 software drivers, so it would be a wise decision to continue to use Keil's C platform for your development work. One advantage of sticking

Listing 1—*Here's an example of a standard read register driver call versus a set IP adder API call. The AX11005 driver and API code is not commented too heavily, but it is relatively easy to follow and understand. The core of the I^{2}C code was pulled directly from the driver source code that comes with the Custom Computer Services PIC C package. All I really had to do was match up my code's I^{2}C interface pins with the LAB-XUSB and write the EEPROM read/write code in the* `main` *function.*

```
// Examples of driver and API routines in the stoe.c code module
void stoe_ReadReg(U8_T regaddr, U8_T XDATA* pbuf, U8_T length)
{
  U8_T  isr;
  isr = EA;
  EA = 0;
  TCIR = regaddr;
  while (length—)
  pbuf[length] = TDR;
  EA = isr;
} /* End of stoe_ReadReg */

void STOE_SetIPAddr(U32_T ip)
{
  if (ip != PNetStation->StationIP)
  {
  PNetStation->StationIP = ip;
  stoe_WriteReg(STOE_IP_ADDR_REG, (U8_T XDATA*)&PNetStation->StationIP, 4);
}
} /* End of STOE_SetIPAddr*/
*********************************************************
//Simple CCS PIC code to modify the contents of the AX11005 Boot EEPROM
#include <18f4550.h>
#device *=16

#fuses HS,NOWDT,NOPROTECT,NOBROWNOUT,NOLVP,NOWRT
#use delay(clock=20000000)

#ifndef EEPROM_SDA

#define EEPROM_SDA  PIN_A4
#define EEPROM_SCL  PIN_A5

#endif

#use i2c(master, sda=EEPROM_SDA, scl=EEPROM_SCL)
#define EEPROM_ADDRESS BYTE
#define EEPROM_SIZE    256

void init_ext_eeprom() {
  output_float(EEPROM_SCL);
  output_float(EEPROM_SDA);
}

BOOLEAN ext_eeprom_ready() {
  int1 ack;
  i2c_start();              // If the write command is acknowledged,
  ack = i2c_write(0xa0);    // then the device is ready.
  i2c_stop();
  return !ack;
}

void write_ext_eeprom(BYTE address, BYTE data) {
  while(!ext_eeprom_ready());
  i2c_start();
  i2c_write(0xa0);
  i2c_write(address);
  i2c_write(data);
  i2c_stop();
}

BYTE read_ext_eeprom(BYTE address) {
  BYTE data;

  while(!ext_eeprom_ready());
  i2c_start();
  i2c_write(0xa0);
  i2c_write(address);
  i2c_start();
```

(Continued)

Listing 1—*Continued.*

```
  i2c_write(0xa1);
  data=i2c_read(0);
  i2c_stop();
  return(data);
}

BYTE data_in,data_out,x;

void main(void)
{
  init_ext_eeprom();
  data_in = read_ext_eeprom(0x01);
  data_out = data_in & 0x7F;
  write_ext_eeprom(0x01, data_out);
  data_in = read_ext_eeprom(0x01);
  ++x;
}
```

with the C tools is that the Keil C tool suite natively supports all of the AX11005/8051/80390 stuff that you need to drive AX11005 code. In addition, when you initially install the DoCD drivers, the necessary HAD2 drivers and support files are automatically inserted into the Keil C tool suite framework.

Adding the DoCD drivers to the Keil µVision IDE pulls everything together into a single seamless AX11005 development environment. The µVision IDE incorporates the DoCD drivers into its debugging system and enables the AX11005 application programmer to edit, compile, program, run, and debug without having to leave the µVision window.

Showing you a screenshot of a debug screen full of little windows and data would be confusing at best. So, Photo 4 is a screenshot of the Load window that supports the AX11005 inside the µVision environment. This should give you an idea of how smooth the DoCD/µVision really is.

ETHERNET MODULE

Now you know which hardware and software tools you'll need to be successful with the AX11005. Although the AX11005 is a feature-rich microcontroller, my real interest (and hopefully yours) is the AX11005 Ethernet module. With that, let's take a look at how it works.

The AX11005 Ethernet module consists of a 10/100-Mbps Ethernet MAC and PHY interface. The thing that really makes the module interesting and different is the TCP/IP offload engine (TOE). Thus, the module is based on a trio of submodules: the TOE, the MAC, and the PHY. Module driver source code for the TOE, MAC, and PHY is included with the AX11005 development package. As I showed you earlier, API calls are mingled in with the TOE and MAC driver source code modules.

The TOE submodule handles all of the TCP/IP offload engine functionality. There are two modes in which the TOE code can execute and both of them support TCP/IP, UDP, ICMP, and IGMP hardware checksum functions. Transparent mode doesn't support the AX11005's hardware acceleration functionality, but Nontransparent mode uses the services of the AX11005's hardware acceleration package. Hardware acceleration in this sense is the automatic handling of the Layer 2 ARP Cache, ARP requests, ARP responses, and the automatic processing of IP packet headers in Layer 3 (see Table 1).

If you have studied and written TCP/IP stacks, you know that lots of code time is absorbed in the handling of packet headers as they pass from layer to layer within the stack. The term "automatic" here with relation to Nontransparent mode means that the AX11005 hardware takes care of the headers instead of your software.

When a socket is established, the TOE submodule grinds out a memory segment that includes a buffer descriptor page (BDP), a receive packet buffer ring (RPBR), and a transmit packet buffer ring (TPBR). If you don't need to modify the buffer sizes, the TOE submodule will allocate 8 KB of RPBR and 4 KB of TPBR memory space. That's pretty much in keeping with other buffer allocations I've made with other Ethernet engines. There's enough room for a couple of incoming packets and plenty of space to queue up transmit packets as you so desire. The BDP is fixed at one page of 256 bytes. Transparency mode and buffer memory sizes are statically defined using standard `#define` constructs in the stoe_cfg.h file, which is part of the TOE submodule code set.

A TOE header is employed regardless of the Transparency mode. Every other Ethernet engine I've ever worked with has provided the length of the packet and a pointer to the next packet in some type of header arrangement. The AX11005 is no different. The first 2 bytes of the 6-byte TOE contain the pointer to the next packet.

Offset address	Bit field	Field name	Size	Description
00h	bit[15:0]	NPR	2 bytes	Next pointer
02h	bit[11:0]	Length	12 bits	Packet length
	bit[14:12]	BPBB	3 bits	RX buffer used page number before REPP boundary (only valid when PCB = 1)
	bit[15]	PCB	1 bit	Packet crossing boundary flag
04h	bit[7:0]	Data offset	1 byte	Data offset address of RX data payload
	bit[15:8]	Protocol	1 byte	Protocol type. It should be equal to the protocol field of the IP header for IP packets. It should be set to 0xFF for non-IP packets.

NPR	Length	BPBB	PCB	Data offset	Protocol	IP Header (for nontransparent)	Data payload
2 bytes	12 bits	3 bits	1 bit	1 byte	1 byte	or Ethernet header (for transparent)	

Table 1—*This is how the AX11005 sets up a packet for processing. All of the header information and the data payload can be easily accessed by with the AX11005 driver and API calls that are provided with the AX11005 development kit.*

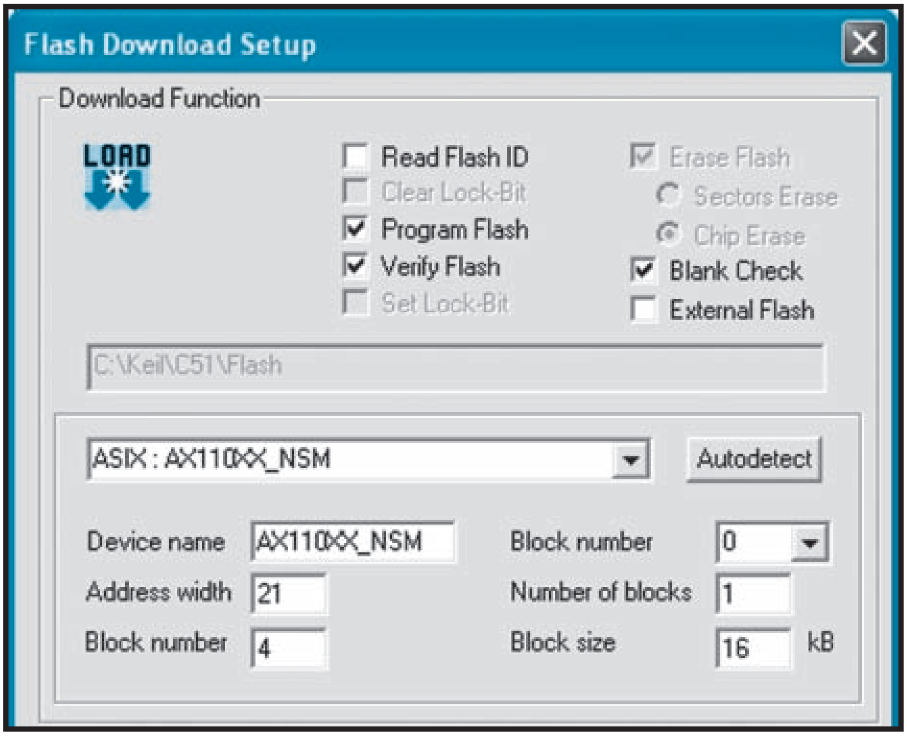


Photo 4—*This is a shot of the Flash Download Setup window that is found within the µVision set-up dialog area. The check boxes make this window self-explanatory.*

The 12-bit packet length follows the next packet pointer. Other information offered up the TOE deals with packet boundaries and the beginning offset address of the packet's data payload. The final byte of the TOE header identifies the packet's protocol. The TOE packet protocol byte is identical to the protocol type found in the IP header. If the packet is not an IP packet, the TOE packet protocol byte is set to 0xFF. The application must make sure the correct protocol type is contained within the TOE header to avoid checksum errors.

Transparent mode will leave the Ethernet header intact so that layers of the stack can process it. The Ethernet header will immediately follow the TOE header if Transparent mode is in play. Nontransparent mode will strip the Ethernet header as the AX11005 hardware performs the header operations. The IP header will be placed immediately behind the TOE header when Nontransparent mode has been selected. The data payload will follow the Ethernet header of IP header depending on the Transparency mode that has been selected by the programmer.

It's the application's responsibility to populate the TOE header before passing it to the TOE submodule. The TOE submodule will add the TOE header at the beginning of the transmit packet, store the packet to be transmitted in the TPBR, and kick off the mechanism that will transfer the packet to be transmitted from the TPBR to the MAC layer SRAM.

Incoming packets will have the TOE header placed at the front of the packet by the TCP/IP offload engine. The packet is then stored in the RPBR. After the incoming TOE-headered packet in the RPBR, the application can use the TOE submodule's basic driver and API calls to parse the TOE header and access the packet data.

The MAC submodule is used to initialize the AX11005 MAC and PHY interfaces to enable normal Ethernet packet transmission and reception. The application can use API calls to handle PHY link change events, set up receive filter modes, and enable wake-up functions such as Magic Packet and external pin wake-up.

The PHY submodule has no API call access. The MAC submodule calls any functions performed by the PHY submodule. The PHY submodule primarily checks and retrieves the PHY Media mode for the MAC submodule.

At first glance, the AX11005 driver code seemed a bit confusing. However, there's nothing going on here that doesn't go on in similar implementations. Basically, the application code uses TOE and MAC submodule API calls to crank up the hardware driver firmware in the TOE, MAC, and PHY submodules.

SERIOUS STUFF

The AX11005 development kit is very comprehensive. When you drop the Keil C compiler and the DoCD hardware into the AX11005 development kit mix, you create a potent Ethernet development suite that will enable you to produce some pretty heady Ethernet-based equipment.

Personally, I've got some ZigBee ideas that I would like to apply to the AX11005. If I can pull them off, you'll see them in a future article. From what I've seen thus far, the AX11005 development kit isn't complicated. It's embedded. ■

Fred Eady (fred@edtp.com) has more than 20 years of experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems design and communications.

PROJECT FILES

To download the schematics for the AX11005 development board, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2006/197.

SOURCES

AX11005 Development kit
ASIX Electronics Corp.
www.asix.com.tw

PICC C Compiler
CCS, Inc.
www.ccsinfo.com

DoCD HAD2 Debugger
Digital Core Design
www.dcd.pl

PK51 C Compiler
Keil
www.keil.com

LAB-XUSB Board
MicroEngineering Labs
www.melabs.com

You need these cutting edge products to quickly solve your toughest design and measurement problems!
New Products
Dualcore....dual ARM IC
Emulin....LINbus/J2602 emulator
WIZnet....TCP/IP IC
Hunt RTG....instant FPGA/DSP
Byteparadigm....USB 2.0 I/O
Systembase....Quad UART chip
Owasys....embedded wireless
HumidiProbe / PT104
HumidiProbe - Temperature and Humidity Logger for PC. PT104 - High resolution and accuracy. Platinum sensors. USB powered - no power supply required. Free s/ware.
from $241/759
Touch-sensing ICs
Quantum ICs- World-beating capacitive sensor ICs for switching and control with patented features like Adjacent Key Suppression, Spreadspectrum, adjustable sensitivity. QRG ICs in use worldwide by the world's largest consumer/appliance manufacturers
<$1 (10K)
Video Modem
VPORT - MPEG4 with Web browser/Ethernet interface and built-in motion detection. 352 x 240 color video streams at 30fps with 24 kbp audio coding.
96047627 $450
USBwiz
USBwiz - Easily add USB peripherals to your system. Includes FATfile system for USB thumb & external USB hard drives. No USB knowledge needed, plug-n-play!
IC $29.95 Demoboard $59.95
2/4ch 12-bit USB scope
PP298/PP300- High-perf. USB2.0-pwrd scope-in-a-probe! Up to 100MS/s, 25MHz 24kS buffer. +/-100mV to +/-20V. 20ns/div - 50s/div C/VBDel
VEE drivers.
FATfile Storage
Touchpanel Controller
CAN-USB
Tiny USB
Stepper Motor
Easy-Step™ 3000 - advanced stepper motor drive and control system for nipolar stepper motors up to 35V/3A nd 3 Amps - ideal for robots/industrial ontrol – turns you into an instant otor control expert! from $119
Cleverscope CS328
PC Oscilloscope, Logic Analyzer, and Signal Generator all-in-one!
Austronics AEEMA '2005 Best New Product' award!
Cleverscope CS328 - NEW 100MHz USB-connected mixed-signal 2-ch PC scope with 8 digital inputs and sig.gen. Huge 4MSample buffer for deep data drilling and features exceeding most expensive stand-alone scopes (sophisticated analog/digital signature/ pulsewidth triggering, glitch capture, whole buffer filestore, "copy/paste" results, etc.). Find hard-to-isolate waveforms! Blows other scope adapters away! Display math-modified and digital signals together. Join multiple units for more channels!
"CS328 is absolutely wonderful, I'm thrilled to bits with it. The software is light-years in front of that (other PC scope). The Math function makes me almost believe I'm using a LxCxxx." Roger – EFE Ltd
CS328 - usually $1045 - only $999 !
Cleverscope software resources include:
• Separate, freely moveable and resizeable windows to display the signal, a zoomed signal view, and the frequency spectrum of the signal, and control panel.
• Spectrum analysis with conditioning windows - log or linear.
• Signal averaging (exponential, block and peak hold) and LPF.
• Math calc including + - / * sqrt, integral, differential, and filtering.
• Signal msmt. including Pk-Pk, RMS, DC, pulse width, T and F.
• Copy and Paste graphic or data to other applications.
• Save and Open from disk.
• User defined units, signal names and scaling (offset and gain).
• Text annotation of each graph.
• Web server for remote viewing of LAN connected unit.
• 2 x 10-bit, self-calibrating A/D channels - 100 MSa/s simultaneously!
• 8 digital inputs - 100 MSa/s simultaneously!
• Huge 4M Samples storage per channel – drill deep!
• Simple "Copy and Paste" into other applications such as Word, Excel
• Highly flexible mixed-signal triggering, scope and spectrum display.
• View small ac signals on large ac or dc offset
• Built-in math capability - multiple display: A-B, invert, diff/integrals, etc.
• Includes all probes cables power supply, s/w, manuals.
• Optional Signal/sweep generator CS700 - +$199.00
Sine, Square: 0.2Hz–10MHz; Triangle: 0.2Hz – 1MHz - Frequency Shift, Phase Shift diff., phase/gain - great for filter, amplifier testing!
microVGA - graphics a allows micros to displ graphics on any VGA Connect to any hos controller, embedded de PC with a USB or se Clever & simple to use.
SM PCB Adapters
Adapt SM miniboards two footprints on either Now use your ultra fine pitch components with more 0.1" inline spaced holes. -one pinouts. ~$10.00
EG-SR-7150MJ - ready-t serial-to-Ethernet Gateway. Si set-up for 10/100Mbps Ethe - 230kbps serial. Ether Internet-enable any serial p
EG-SR-7150MJ
-Serial Adapters
rt for laptops without COM instantly updates 32 products to USB! 's using COMport up el.
only $12.99
only $30 (1) or $25 (100)
Easy USB
microUSB - Tiny USB ready-made USB Interface to 1 Mbps 0.7" x 0.5" based on SiLabs CP2101. Gnd/RX/TX Suspend, Reset, & 3.3V 100mA source –40°C +85°C.
microUSB only $25
TrackStick™ - gumpack-sized logger tracks position data for shipments, vehicles, personnel, etc. Records date, time, location, speed, heading and altitude at pre-set intervals - shows route/speed on PC! $249
ANT16 Matchbox-sized USB-powered logic analyzer. Sample 16 channels at up to 500 million samples-per-sec. View traces on PC. Print or save for later review! (8-ch. ver. too!) $299
EMI shielding tents made from Ni/Ag fabric. -86dB 20MHz to 18GHz. Portable - tabletop to room/lab size for on-site, on-demand pre-compliance testing. External PVC or aluminum frame. from $3500
USB Bus Analyzers
Tracker 110/Explorer 200 high-perf. economical USB 1.1/2.0 protocol analyzers. $999/$2999
USBEX300 for debugging USB wireless Wimedia <$29,500
New - Bluetooth Sniffer <$10K
NEW Low-Cost!
LCD Scope
PDS5022 - 25MHz 2-ch + trig. Standalone bench scope with USB connection and 7.8" color LCD. Built-in meter - great for your service kit.
PDS5022 $399
I2C for PCs
PCI93LV: industry-standard I2C card for PCs. WINI2C/PCI software gives windows-interface to develop and debug I2C bus systems. UCA93LV is new USB version. NOW! - transparently monitor at 400kHz! only $499
FrontDesign
FrontDesign - software for designing professional-looking front panels - with a host of drawing functions. Print, mill or engrave, or export final at 600 dpi.
FrontDesign only $79.95
Ether-IO
Ether-IO - UDP/IP-controlled 24 X digital I/O board. 3 x 8-bit TTL ports each independently programmable. Connects to any TCP/IP Ethernet network.
Ether-IO 24 from $99
Signal Wizard
Signal Wizard - easy-use real-time DSP-based filter board for audio bandwidth signals. Design filters in seconds without any DSP knowledge!
Signal Wizard II only $399
USB to I2C
USB I2C/IO - provides a simple "drop-in" solution for connecting your PC to 90kHz I2C + 20 I/O lines. Free software. Use multiple boards for more I2C/IO.
USB I2C/IO only $79
FREE COFFEE
Call 1-888-7SAELIG
Mention Offer# SBW
to get a free STARBUCKS Card with your order!
While supplies last - not available with any other offers
Saelig Co. searches the world to bring you unique, easy-to-use control and instrumentation products from overseas. Customers include: Intel, Philips, NEC, Kodak, Nokia, US Military, Microsoft, Dell, Xerox, Universities, T.I., Harris, Sony, J&J, Thomson, Sandisk, General Dynamics, H-P/Compaq, etc.
Saelig.com unique electronics
888-7SAELIG info@saelig.com
Also: Wireless Temp Loggers, Stepper Motor Controller, PC for harsh environments, Automotive Diagnostic Kit, PC for Healthcare, Wireless controller, Oscilloscope, JTAG testing, Indicator modules, Video Motion Detectors, Industrial IO modules, Software for soundcard - front panels - logic simulation and hardware control - CAD
If you don't see what you need maybe we can find it for you? - Ask for sales!

PC/104 Single Board Computers

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 256 M (with ARM Tool Chain), USB WiFi

200 MHz ARM9
Power as low as 1/4 Watt

- 5 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, NetBSD

$99 qty 100

$129 qty 1

NEW!
- SD card option
- VGA video

Off-the-Shelf Solutions ready to design into your project using DOS development tools

TS-5600 Shown with optional flash modules, A/D, RS-485 and Merlin cellular modem

- options include:

RS-485 Half and Full Duplex, A/D Converter up to 8 Channels at 12 bits, DAC up to 2 Channels at 12 bits, Extended Temperature

133 MHz 586

- Power as low as 800mA
- Fanless, no heat sink
- SDRAM - up to 64MB
- COM Ports - up to 4 ports
- Ethernet Ports
- DIO Channels - up to 40
- PCMCIA II adaptor
- Compact Flash adaptor
- USB Ports (Except on TS-5300)

$229 qty 100

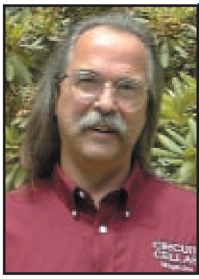
$259 qty 1

see our website for 33 MHz 386 configurations

- 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

by Jeff Bachiochi



Portable Lightning Detector

Looking to protect your home and electrical systems from lightning? Jeff describes lightning protection technology and explains how to build a PIC16F687 microcontroller-based portable lightning detector.

Our local Boy Scout troop has weekly meetings that rarely deviate from a standard 7:00 to 8:30 P.M. schedule of indoor activities. During the last meeting in June (just before taking a break for the summer), we held an outdoor track meet at one of the local high schools. It gave the boys a physical outlet that up to that point had been restricted to monthly camping trips. Spring in the Northeast is prime thunderstorm season, but we had never run into any foul weather during our "Track Nights." This year, however, the skies were darkened by scattered cloud cover, which moved at a brisk pace. We could see for quite a distance thanks to the surrounding cornfields. The high heat and cool grass invited a shoeless game of Frisbee soccer.

We camp rain or shine every month in whatever weather nature provides. So, the light sprinkling of rain on that particular evening didn't faze anyone. The game's intensity increased as the score bounced back and forth and continuously shifted each team's destiny. It was like a World Cup playoff. No one gave the weather a second thought. We saw nothing. We heard nothing. But it seems pretty clear now that a lightning strike must have hit the ground somewhere close by. How close is unknown. Those clad in bare feet were instantly hit with a burning sensation in their toes. We were experiencing surface current that was traveling along the ground like spokes extending from a wagon wheel's hub (the point of impact). Half of the field emptied in panic as those affected began running in every direction, mostly toward the shelter of vehicles in the nearby parking lot. Meanwhile, the sneakered players were dumbstruck at the sight of so many instantly possessed by an unseen force.

Photo 1—*Staying in touch with nature enhances outdoor safety. My portable lightning detector offers its take on storm activity in the area. The LED's relative intensity indicates when a storm is approaching or going away.*

In "Ground Zero: A Real World Look at Lightning," Steve Ciarcia and I presented you with some basic information about lightning (*Circuit Cellar* 90, 1998). We also included the steps needed to change the status of Steve's house from "lightning rod" to "protected by the lightning rod." Trying to direct a lightning strike is like playing with fire. One weak link and a hit will light your fire.

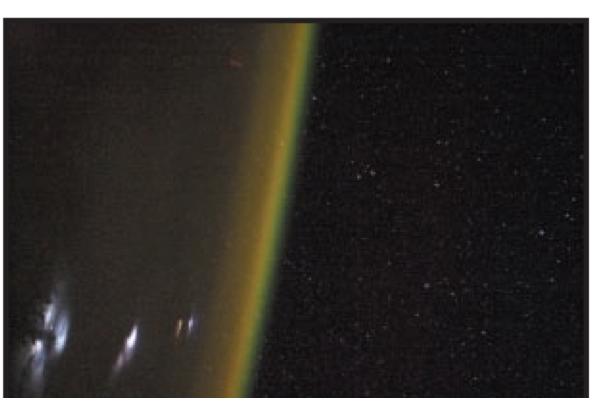


Photo 2—*Check out this photo of lightning taken from space in 2003. Here you can see the thin veil of Earth's atmosphere running vertically. Stars are evident on the right side against the darkness of space, and an electrical storm rages over a landmass on the left.*

For those times when you may find yourself exposed to hazardous lightning strikes, being aware of storm tracks makes healthy sense. After all, the best way to survive lightning is to avoid it. With this month's project, you can take charge of lightning strikes with your own portable lightning detector (see Photo 1). This combines an RF analog front-end detector with a microcontroller. LEDs indicate the status of a storm's activity.

BREAKING THE SOUND BARRIER

The potential buildup of unlike charges in thunderclouds seems to occur from the rising and falling of ice particles. This was evident a few weeks ago as a thunderstorm passed over Crystal Lake here in Connecticut. The soft splashes of rain falling amid the cracking of thunder gave way to a couple of minutes of clinking. This rapid change in timbre on my office window led me to the porch where I watched 0.5″ hailstones fall from the sky while the thermometer was approaching 90°!

Cloud-to-cloud lightning is far more prevalent than the cloud-to-ground events caught by many photographers. Theories abound on just how a discharge path between a cloud and Earth is determined. A strike takes about the same amount of time as a single execution cycle in many microcontrollers! Because of the intense energy (millions of volts and thousands of amps) of a discharge, the surrounding air is superheated. The expanding heated air causes a shockwave along the entire discharge path. The shock wave propagates as a thunderclap with varying duration, depending on your orientation to the discharge. Count the seconds between seeing the flash (the speed of light) and hearing the sound of thunder (the speed of sound) to determine the strike's distance. The distance to the strike is about one mile for each 5 s of time delay.

If you follow *NASA Select* by satellite, cable, or via the Internet, you're familiar with the nighttime views of the Earth from 200 miles up (from either the Shuttle or the ISS). The most prominent feature from above is the clouded sky featuring spectacular lightning at a rate that can exceed one strike per second (see Photo 2). Imagine the quantity of available energy. Harnessing this energy has the potential of eliminating oil as the world's most important resource and changing forever the face of world politics.

DETECTION TECHNOLOGY

Today's technology for detecting lightning activity can be divided into three varieties. The first is via optical monitoring. Photographers often use slave flashes to augment their camera mounted flash. These devices monitor illumination levels and can trigger their secondary flash based on "seeing" the camera's flash. The optical monitoring of lightning is similar, although this type of detection is most difficult on sunny days. A stormy sky offers enough contrast for a strike to be sensed easily. When used in conjunction with LF (audible) waves, the optical sensor can be used to initiate a simple count between flash and crash to produce a distance-to-strike value.

A second type uses RF detection. If you've listened to AM radio, you know that lightning's discharge produces static noise across the frequency spectrum. Mother Nature would certainly fail the FCC's allowed emission testing. Luckily, she is covered by the grandfather exemption. Although this spectrum noise is easily detectable, other noise sources can produce false triggers. Monitoring for coinciding triggers at multiple frequencies can reduce false triggering.

The last method measures electric fields. A cloud and the Earth act as plates in a capacitor. As the potential increases between the plates, this can be measured as a change in the Earth's electric field. Under sunny skies, the ionosphere is responsible for producing less than 100 V/m. During storm activity, this value can rise to thousands of volts per meter. These values rise with the approach of potential strike, so the electric field can be used to warn of an impending strike.

SEARCH FIRST

In searching the Internet for available information about lightning detection, I ran across an interesting project that was a step in the right direction. Charles Wenzel has some great circuits on his web site (http://www.techlib.com/electronics/lightning.html). His analog circuit is a perfect match for this project. As you can see in Figure 1, the front-end antenna/tank circuit gathers any activity in the 300 kHz range. Oscillations are amplified through T1 and applied to the base of a flip-flop (T2/T3). Sufficient signal will turn on T2 (and T3) and discharge timing capacitor C11. Because C11 provides the biasing for T2/T3, these transistors are turned off as the capacitor is discharged. A diode between C11 and the pull-up resistor (R7) allows C11 to discharge quickly but through R7 more slowly. The minimum low time of the output pulse is approximately 20 ms.

Each strike produces an output pulse from the analog front end. By monitoring these pulses, you can determine a few things about the storm's movement. From a fixed position (wherever you are), every storm will pass you by. Assuming that a storm moves in a straight path, it can be moving either toward or away from you. Whether or not you can actually detect it depends on how close you are to its path and the size of its umbrella. I use the word umbrella to help visualize that a storm has dimension to it. A lightning strike may originate anywhere beneath this umbrella and not just at the storm's centroid moving along its path.

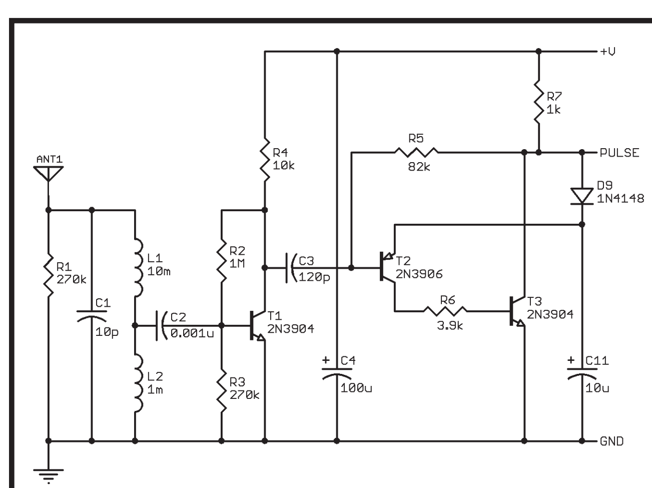


Figure 1—*A lightning strike produces EMI across the frequency spectrum. The analog front end of this project picks up activity in the 300-kHz region and produces a 20-ms pulse indicating its presence.*

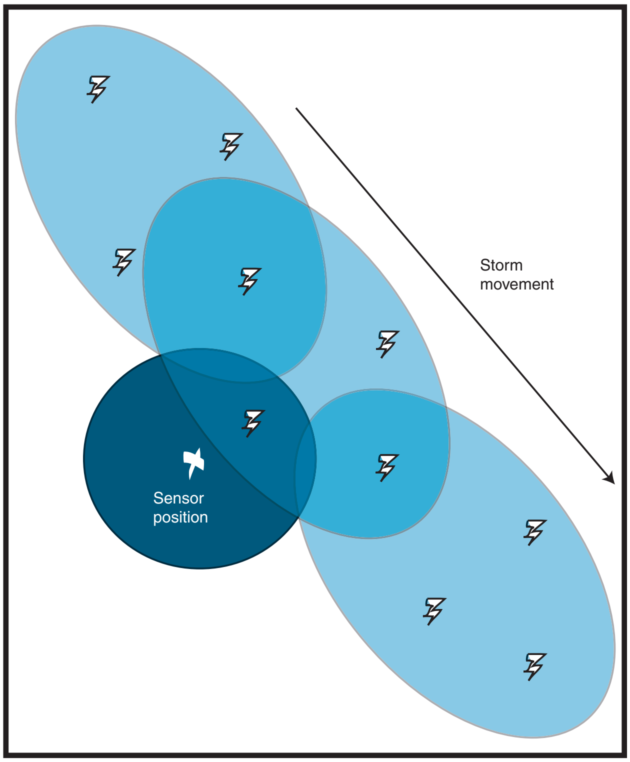


Figure 2—*A storm's umbrella coinciding with the sensing umbrella of the PLD makes the sensing of a lightning strike possible. The more overlap (closer the storm), the higher the potential strike rate. PLD activity is a function of both storm speed and proximity.*

An approaching or receding storm can be beyond the detection distance. The detection distance can also be thought of in terms of an umbrella. As a storm umbrella overlaps the detection umbrella, it becomes possible for a strike to be detected. However, if any strike activity were to take place on the far side of the storm's umbrella, these strikes may be outside the detection range (see Figure 2). While a storm's activity may remain constant, its strike rate from our vantage point changes.

DIGITAL PORTION

The detection method used here does not measure energy levels. The status warning level is based on the hit rate of a storm, which should increase as the storm's umbrella overlaps more of the detection umbrella. As the storm's centroid passes its closest point to the detector, the hit rate should begin to decrease as the overlapping areas decrease.

The status warning level of a storm is displayed in a bar graph. One of the microcontroller's ports drives eight LEDs. Figure 3 shows the digital portion of the device. The first LED is used to indicate that the system is operating. LEDs 2 through 8 show storm activity. An approaching storm activates LED 2 (green). As the activity intensifies, LED 3 (yellow) is triggered. Red LEDs 4 through 6 indicate maximum activity. LED 6 signifies that the storm is beginning to leave the area. LED 7 (yellow) confirms that the worst is over, while LED 8 (green) indicates the final fringe activity.

As I stated earlier, the storm status is based on the time between lightning strikes. Determining a storm's status begins with a clean slate. The slate is cleaned (variables initialized) whenever the circuit is powered on or a time of 256 s has passed without a strike producing pulse from the analog input. `PeakTime` (a variable holding the shortest interstrike time in seconds) is initialized to maximum. The FIRST flag is also set to indicate when you haven't received any strikes yet.

HIT ME, BABY

The program written for this project is interrupt based. Although most of the time is spent idle in the main execution loop, all of the action takes place within interrupt routines. Each action has its own routine. A change-of-state (COS) interrupt handles push button presses. The strike pulse produced by the analog front-end edge-triggers an external interrupt. Seconds are accumulated by overflow interrupts of Timer1. The piezo element receives a 4-kHz tone burst via the Timer0 interrupt routine.

The falling edge-triggered external interrupt monitors pulses from the analog circuitry. When triggered, the external interrupt routine sets the BEEP flag so that a short 4-kHz burst will allow the piezo device to audibly signal each strike. If the FIRST flag is set, this indicates that no strikes have been detected up to this point. So, the external interrupt routine needs to clear only the FIRST flag and the count in Tic, which is the variable holding the count in seconds since the last strike. (Tic continuously increments based on the Timer1 overflows at a 1-s rate.) With the FIRST flag

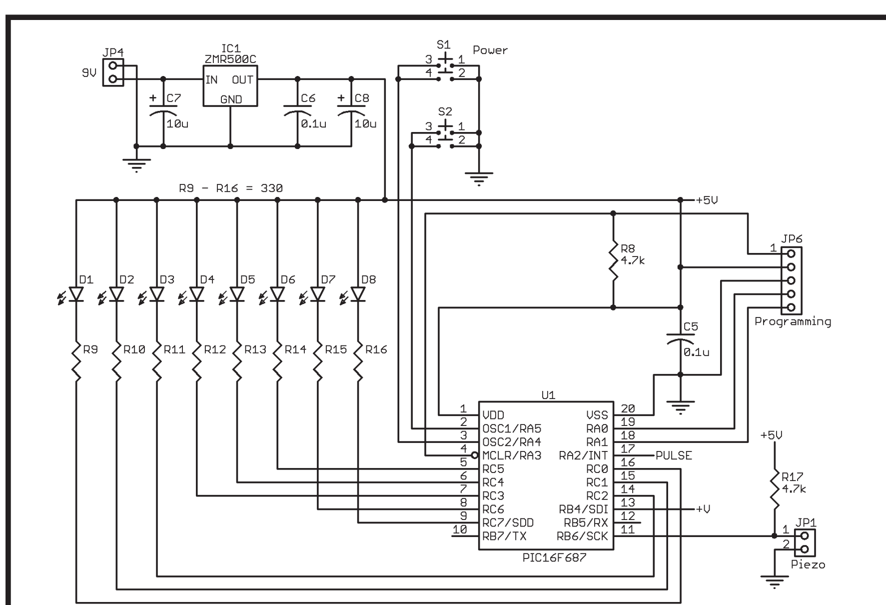


Figure 3—*The digital portion of this project monitors the time between pulses (strike rate) and produces an LED status display based on preconfigured constants.*

Present time	Warning level constants	Present level	DIR Flag (0 = Approaching, 1 = receding)	LED 2–8
> 128 s	—	0	—	none
127–64 s	128 (level 1)	1	0	2 (green)
63–32 s	64 (level 2)	2	0	3 (yellow)
31–8 s	32 (level 3)	3	0	4 (red)
< 8 s	8 (level 4)	4	0	5 (red)
31–8 s	32 (level 3)	3	1	6 (red)
63–32 s	64 (level 2)	2	1	7 (yellow)
127–64 s	128 (level 1)	1	1	8 (green)
> 128 s	—	0	—	none

Table 1—*The PresentTime is compared to each of the four level constants to determine the appropriate warning level (PresentLevel). The DIR flag is based on the PresentTime and the PeakTime. The four constants (Level 1:4) make it easy to experiment with the warning level parameters.*

cleared, any subsequent strike triggering the external interrupt saves the Tic count into the `PresentTime` variable. Tic is then cleared again, beginning the count to the next strike. Based on `PresentTime`, a level of warning is determined.

There are four (warning) levels set by constants. These predefined times are compared to PresentTime to determine PresentLevel and PeakLevel (see Table 1). PeakLevel is a peak detector, and it increases as PresentLevel increases. However, if PresentLevel should fall below PeakLevel, PeakLevel won't decrease and the DIR flag is set to indicate that the storm is moving away. Finally, based on PresentLevel and DIR, the proper LED is enabled by clearing the appropriate bit in LEDFlags prior to leaving the external interrupt routine.

ONCE UPON A SECOND

Besides having the responsibility of incrementing Tic, the Timer1 overflow interrupt routine has other functions. If a storm abruptly ends or the PLD is not triggered by radiated noise from some other source, the rollover of Tic is monitored. After 256 s without a strike, the system reverts back to the initialized state. This takes care of those situations in which activity has ceased, and it requires more than 4 min. without any detection.

Before leaving the Timer1 interrupt routine, the LEDs are updated. Under normal operation (no storms), only the power LED 1 would be cleared in LEDFlags. In stormy conditions, an additional LED (2 through 8) warning level may be cleared. The TOG flag is alternately set and cleared by this interrupt routine at the routine's 1-s rate. When TOG is set, the power LED 1 is turned on. When TOG is cleared the LEDs 2 through 8 are turned on. Only one LED draws current at a time.

Additionally, this routine monitors the BEEP flag. If a beep has been requested, the Timer 0 interrupt is enabled and the BEEP flag is cleared by setting the BEEP flag elsewhere. Otherwise, the Timer0 interrupt is disabled. This action allows the Timer0 overflow interrupt routine to toggle the output bit (tied to the piezo device) at each Timer0 interrupt. Because the Timer0 interrupt will be disabled the next time through the Timer1 interrupt routine, the 4-kHz tone will last for just 1 s.

ZZZZZZ

By allowing no more than one LED to be on at any time, you keep the operating current to approximately 10 mA (depending on the LED's series resistors). The analog section requires very little current. I used an I/O bit on a Microchip Technology PIC16F687 microcontroller to power this portion of the circuit. This allows the analog section to be powered down before entering Sleep mode. When the circuit has battery power applied, it will begin initializing and executing code. Push button switch SW1 toggles the power LED on and off. The main routine's only function is to monitor the status of this bit and go to sleep if SW1 turns off the lower LED 1 (power). The PIC16F687 microcontroller can exit

Sleep mode on MCLR reset or any enabled interrupt source. Individual peripherals can be disabled prior to going to sleep to prevent a wake-up from an unwanted source. I'm allowing only the COS interrupt from SW1 to wakeup the processor.

In Sleep mode, the current in the microcontroller is down in the microamp range. However, the battery current in Sleep mode was approximately 4 mA. After much head scratching, I realized the culprit. The 78L05 voltage regulator has a 4 mA idle current specification. This 4 mA looks like a 2.2-kΩ load to my 9-V battery. That's about 50 h (or two days) independent of whether or not the circuit is sleeping! By switching to a Zetex Semiconductors ZMX500C voltage regulator (still in a TO-92 package), the quiescent current is reduced to less than 50 µA. At 50 µA, that same 9-V battery would last about a year. That seems sufficient without having to worry about using either a lever switch to cut power completely or design in a regulator with an on/off input like the Toko TK11150CS voltage regulator (with an off current of less than 1 uA).

PACKAGING

Serpac's H series enclosures have either a twin AA or 9-V battery compartment option. The enclosure looks like a TV remote (see Photo 1). I also got the enclosure with a belt clip, which is really handy for sticking the system on my belt, in my pack, or in any other handy location.

I used a 9-V battery with this circuit. However, using two AA batteries would probably work well because the microcontroller will operate down to 2 V at 4 MHz. (I'm running at 2 MHz.) Of course, there are other parts of the circuit that may run out of steam (i.e., piezo volume and LEDs).

HINDSIGHT

I despised trying to use a whip antenna, so I attempted a few versions of the RF front end using a loopstick antenna. I had a serious episode of brain fade. I forgot about the directional characteristics of this antenna. There is a cute commercial by Corona that suggests this (sort of). You see an AM radio on a table playing tropical tunes as waves roll in on the deserted beach in the background. A bottle of Corona next to the radio leaves the field of view (refreshing its owner off camera). When it's placed back on the table the radio reception is rather awful. Adjustments to the bottle's position relative to the radio clear up the reception. Well, because I couldn't fit multiple loopsticks with the right orientations into the Serpac's enclosure, I buckled under the pressure and made provisions for an extendable whip antenna.

Let me tell you one last story on the subject of lightning. One of the Boy Scout troop leaders was working around his house when he heard a mighty crash. The volume was sufficient to pique his curiosity, so he went outside to see if the local intersection was featuring a confrontation between metal beasts. With no traffic in sight, he scanned the horizon to see if he could find any abnormalities. Everything seemed normal, apart from the missing chimney. Ah, that's why there were bits of brickwork strewn around the yard! The chimney had been blasted apart by lightning. This made no sense. The yard was littered with huge trees towering over the roof of the house. You'd think, "A strike must have been attracted to one of those giants."

After inspecting the remains of the chimney in the attic and determining that the roof was intact, there didn't seem to be any interior damage to the structure. So, he shrugged it off and began searching for masons in the phonebook. About 30 min later, he smelled a hint of smoke. Recognizing the importance of swift action, he called 911. Members of the town's volunteer fire department began arriving within minutes. Because of his quick call, firefighters were able to assess the situation before flames could take hold. From the attic, they zeroed in on the top of a wall that had smoldering insulation. Downstairs, they opened up the wall and removed the offending insulation with little interior damage. No one could make any sense of the strike's selection process. Why did the bolt avoid all of the taller targets and hit a chimney that didn't have any antennas or lightning rods?

We certainly have a lot to learn about lightning and our world in general. Who knows what insight further investigation will bring? Nature is both helpful and dangerous. It might be obvious that swinging a golf club on hilly terrain that's devoid of structures is potentially dangerous. And, after seeing or hearing a storm, any sane person would seek shelter. But if you've ever been anywhere near an actual lightning strike, you know that this is something you do not wish to mess with. Any additional warning can mean the difference between life and death.

Jeff Bachiochi (pronounced BAH-key-AH-key) has been writing for Circuit Cellar *since 1988. His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circuitcellar.com.*

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2006/197.

RESOURCES

The National Severe Storms Laboratory, "Questions and Answers About Lightning," 2004, www.nssl.noaa.gov/edu/ltg.

C. Wenzel, "Lightning Detectors," *Wenzel Associates, Inc.*, www.techlib.com/electronics/lightning.html.

SOURCES

PIC16F687 Microcontroller
Microchip Technology, Inc.
www.microchip.com

H-65, 9-V Enclosure
Serpac Electronic Enclosures
www.serpac.com

TK11150CS Voltage regulator
Toko America, Inc.
www.toko.com

ZMX500C Voltage regulator
Zetex Semiconductors
www.zetex.com

Back a winner!

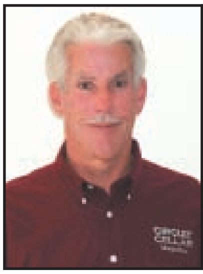
Atmel's ARM®-based 32-bit microcontrollers are winners. They have already picked up four prestigious awards from readers of industry leading magazines*. Why? Because they give you exactly what you want. On-chip Flash memory. USB & Ethernet connectivity. DMA to eliminate internal bottlenecks. Supervisory functions. All this at the lowest possible power consumption and unit price, plus code compatibility across the entire product family. So, make your application a winner by backing a winner: Atmel's AT91SAM Smart ARM-based microcontrollers.

* EEProductCenter Ultimate Product (Processor & Memory) for Q4 2004 and again for Q4 2005, Embedded Control Europe Gold Award (Micros & DSP) for H2 2005, 2006 EETimes ACE Award Ultimate Product of the Year.

Learn more about our AT91 products by visiting our web site, at www.atmel.com/ad/at91 and register to qualify for a free AT91SAM7 development kit. You will also receive an AT91 DVD with extensive product documentation, training material, application notes and code samples.

Check out Atmel's AT91 solutions at www.atmel.com/ad/at91

Everywhere You Are®



SILICON UPDATE **by Tom Cantrell**

Ferro-51

Twenty-five years after the debut of the original, a new '51 MCU is making headlines. Tom describes Ramtron's VRS51L3074 FRAM-enhanced '51 MCU and prepares you for what's to come.

By the time you read this, I'll have given my "Is ARM the '51 of Tomorrow?" presentation at the ARM Developers' Conference in Santa Clara, California. Can ARM match the popularity and longevity of the venerable '51? Is it MCU déjà vu all over again?

Leaving those questions for another day, this month's topic draws from one of the punch lines in my presentation. In one sense, ARM can't be the '51 of tomorrow because, ta-da, the '51 is the '51 of tomorrow.

That's right. While there's no doubt that low-cost flash memory 32-bit MCUs are taking off, the idea that they will replace 8-bit MCUs overnight is ludicrous. Rather, my take is that while the 8-bit MCU growth rate may be topping out, the 8-bit story is far from over.

One thing both ARM and the '51 do uniquely share is a multisourced architectural bandwagon. That's important because it increases the variety of specialized parts designers have to choose from. Case in point is this month's topic, yet another "new" '51 making its debut fully a quarter century after the original.

MEMORY MAPPING

As I was researching last month's article on alternative memory technologies, I came across an interesting press release on Ramtron International's web site. As I mentioned in that column, when it comes to MRAM, FRAM, and the like, I think embedded applications are a real market, one that's overlooked in all the Holy Grail hype of replacing stand-alone DRAM and flash memory. Thus, it's no surprise that "Ramtron Introduces the First FRAM-Enhanced 8051 MCU" was right up my alley and definitely got my attention.

Ramtron's been in business selling FRAM memory chips for a long time, but I didn't know they made '51s. Ah ha, it turns out they're new to the MCU game, buying into the market with their recent (August 2005) acquisition of Goal Semiconductor, a niche supplier of mixed-signal '51s.

Imagine an MCU that gets by with just FRAM instead of the motley collection of flash memory, RAM, EEPROM, ROM, etc. that clutters today's chips. Better yet, FRAM would eliminate all the granularity issues (e.g., code versus data storage) associated with today's fragmented multimemory setups. Faster, lower power, nonvolatile, cheaper, and more flexible: that's what I'm talking about!

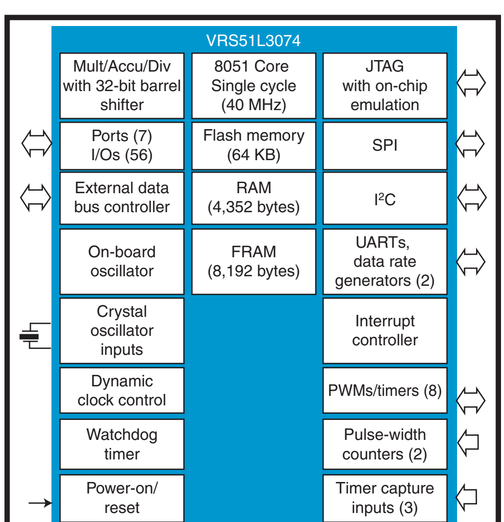


Figure 1—*Ramtron makes their move into the MCU business with an assist from Goal Semiconductor, which they acquired last year. The headline feature that sets their VRS51L3074 apart from the crowd is 8 KB of FRAM, but this '51 is also notable for its performance features.*

Sounds good, but that's not what the new Ramtron VRS51L3074 microcontroller is. The press release is careful to say that the '3074 is just the first step along the way. Witness the fact it still has flash memory (64 KB) and SRAM (4 KB + 256 bytes) along with the 8 KB of FRAM (see Figure 1). Indeed, under the hood, I'm pretty sure you're looking at separate MCU and FRAM chips rather than true unification of FRAM and CMOS processing on a single die.

Based on the specs of Ramtron's very SRAM-like 1-Mb chip that I covered last month, I expected to find a similar FRAM on the '3074. Thus, I was a bit surprised to find that the integrated FRAM on the '3074 isn't nearly as capable as the stand-alone part.

First the good news. Once all is said and done, access to the on-chip FRAM is via the '51's MOVX instruction (i.e., the same instruction used to access the on-chip SRAM as well as external memory and I/O devices). There's no need for the multi-instruction software bit-banging or programming routines typically required by flash memory or EEPROM. A bit of software is required to set up and enable the FRAM; but after that, a MOVX is all it takes.

Last month's 1-Mb FRAM chip did a decent job of delivering on the promise of next-generation memories by offering the speed and read/write symmetry near that of an SRAM. However, the FRAM on the '3074 isn't especially fast or sym-

	FRAMCLK[1:0] = 00 (Sysclk/2)		FRAMCLK[1:0] = 01 (Sysclk/4)	
Mode	**Read**	**Write**	**Read**	**Write**
Normal	1.9 µs	2.6 µs	3.6 µs	4.9 µs
Burst*	1.1 µs	0.4 µs	2.3 µs	0.7µs
Read burst*	0.7 µs	0.4 µs	1.6 µs	0.7 µs
*Based on 100 consecutive read and write operations in Burst mode.				

Table 1—*FRAM has a lot of potential, but the implementation on the '3074 could be better (i.e., faster with symmetric read/write times). Burst modes help, but impose system timing constraints that may trip up the unwary.*

metric, with minimum access times of 1.9 and 2.6 µs for reads and writes, respectively. Note that the MOVX is interlocked so that the processor stalls until the FRAM read or write completes.

Like their stand-alone memory chip, the '3074 FRAM does support Burst modes that can increase throughput somewhat (see Table 1). But Burst mode has some gotchas that can trip up unwary designers. Once you put the FRAM in Burst mode, you must read only consecutive addresses or write consecutive addresses. Mixing reads and writes or skipping addresses isn't allowed.

Furthermore, there is a timing constraint limiting the idle time allowed between each transfer in a burst. If you dally too long, the FRAM automatically exits Burst mode. Although the idle time allowed is somewhat programmable (four choices), it does require that you guarantee your software can meet the required deadline. For example, this may require disabling interrupts during a burst transfer or at least extra software to correctly restart an interrupted burst.

The scary part is that if you make any mistakes (e.g., mixing reads and writes, nonsequential address, missing the burst timeout deadline) the FRAM can lose data, and there's no interrupt or error flag to let you know that happened. If you must use Burst mode, it's there, but be careful.

If there's a bright side, the relatively slow access time in conjunction with ever-improving endurance specs virtually eliminates concerns about wear out. The '3074 FRAM retention time spec is a whopping 45 years! Although I couldn't find any retention time or endurance specs for the flash memory in the preliminary datasheet, I presume they're similar to those of other flash memory MCUs (e.g., ten(s) years, thousand(s) cycles).

MIPS MASTER

The original '51 was rather pokey, maybe hitting 1 MIPS with a tailwind. By contrast, like practically all '51s these days, the '3074 offers good performance by a combination of architectural turbocharging (fewer clocks per instruction) and more revs (i.e., faster clock up to 40 MHz). I figure a typical application program mix would average three to four clocks per instruction for an honest 10+ MIPS.

The '3074 features a 40-MHz oscillator on-chip that, with 2% precision, should be accurate enough to eliminate the need for crystal in many applications. In addition, an external 4- to 40-MHz crystal can be connected, and you can dynamically switch between the two. In turn, the chosen oscillator feeds a 16-level power-of-two prescaler (i.e., divide ratio = 1, 2, … 32,768), which allows software to control the throttle depending on the task at hand. There's even a feature that can automatically hit the gas (i.e., set clock divide ratio to one) when an interrupt occurs.

For number crunching, the '51's built-in MUL and DIV instructions are standouts at just two clocks (i.e., 50 ns at 40 MHz). But the math capability of the '3074 goes way beyond just speeding up those native instructions. As you can see in Figure 2, the chip integrates a 16/32-bit arithmetic coprocessor and barrel shifter that can significantly accelerate math-intensive applications (e.g., DSP loops). There is a bit of software overhead (preferably ASM for speed) to drive the coprocessor, but ultimately, it returns answers as fast as you can feed it operands. It's no DSP, but the math coprocessor does deliver signal processing capability far beyond that of most 8-bit MCUs.

Another embellishment is a second DPTR (data pointer) register to bypass the well-known bottleneck imposed by the originals single pointer. For example, block memory operations can use one of the data pointers for the source address and one for the destination rather than having to coerce a single DPTR into serving double duty.

There are also additional instructions that streamline access to the all-important special function registers (SFRs) that control the various I/O ports. In the interest of avoiding any possible software compatibility issues that might arise with the new opcode (which is an NOP for regular '51s), this feature can be dynamically enabled and disabled by software.

Speaking of I/O, '3074 upgrades fall into two categories. First, be reassured the historic '51 I/O features are all there and can work the same as before. Indeed, the pinout for the 44-pin version of the MCU ('3174) is the same as that of many traditional '51s. However, going beyond compatibility, there are also options that enhance the traditional I/O functions with new capabilities.

For example, the original '51 required using a general-purpose timer as a bit rate generator for its built-in UART. By contrast, not only does the '3074 provide dedicated bit rate generators, but the

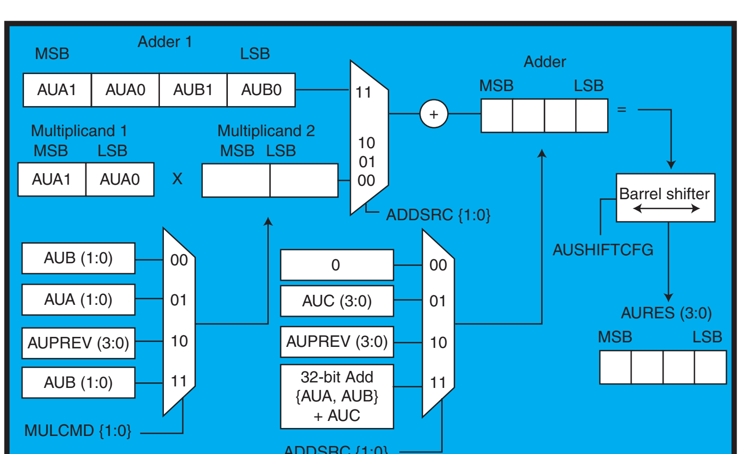


Figure 2—*With a combination of 32-bit multiplier, adder, and barrel shifter, the '3074A built-in math coprocessor turns this '51 into a mini-me DSP.*

Photo 1—*The '3074 evaluation kit is relatively standard fare, but with a notable FRAM bias. In addition to the FRAM on the MCU itself, the board includes three stand-alone serial FRAM chips.*

general-purpose timers also feature new modes as well. For example, timers that are limited to 8- or 16-bit operation in a traditional '51 can be chained together to deliver 24, 32, and even 48 bits of resolution. Another upgrade supplements the general-purpose (i.e., parallel) I/O lines with a pin monitor function that can detect a change (edge or level sense) on any and all pins of interest.

The second class of I/O enhancements includes new features not found on the original '51. These include SPI (enhanced to support transfers up to 32 bits in length) and I²C (both Master and Slave modes), as well as even more timing and counting capability in the form of an eight-channel PWM and a two-channel pulse-width counter. The PWM features 16-bit resolution and can be used for generic timing if PWM capability isn't required. The pulse-width counter feature works with the general-purpose timers (i.e., T0 and T1) to boost event timing and gating flexibility by allowing the count start and stop conditions to be programmably defined in various ways. For instance, you could configure the timer to start counting on a rising edge of one pin and stop counting on the falling edge of a different pin.

FREE "C"

Given the maturity and popularity of the '51, one thing you can be sure of is that there's no shortage of development tools. Even as a new player, Ramtron has fairly decent support, including packages from major suppliers like Keil and American Raisonance. Better yet, over the years, a variety of low-cost and even free alternatives have emerged that belie the traditional truism that you get what you pay for. With Ramtron's $99 '3074 evaluation board in hand, let's see how far freeware can take you (see Photo 1).

The board itself is rather minimalist and straightforward with little more than the '3074 chip, RS-232 transceivers for its twin serial ports, some LEDs, and a connection for a JTAG debugger. Somewhat ironic considering the '3074's built-in FRAM, the only notable add-ons are a trio of the company's stand-alone FRAMs, including a processor companion chip and SPI and I²C variants.[1] It's a veritable FRAM traffic jam! That seemingly odd situation is explained by the fact the same board is used for evaluating the various FRAM chips and non-FRAM variants of the MCU.

The JTAG debugger dongle itself gets marked down by virtue of the fact it uses a printer port interface with the PC. That was once a common hack, but it has fallen out of favor with the rise of USB and the fall of PCs equipped with parallel ports. No problem you say, I'll just use a USB or serial-port-to-parallel-port adapter. Sorry, Charlie. Those are designed to talk to printers, and the manual cautions that they won't work. Fortunately, my desktop PC has a parallel port, and luckily I didn't even have to twiddle any BIOS (EPP, ECP, etc.) or driver settings (port addresses or interrupts) to get the thing to work.

Along with evaluation versions of the pro tools (Keil, etc.), the kit comes with the bits and pieces of software that allow you to cobble together a freeware solution. Compared to the elegant (and pricey) one-screen-does-all IDEs, it may seem a bit of a hack, but I actually found the old-school solution quite serviceable.

The low-budget lash-up comprises three components: a programmer/debugger from Ramtron, a GNU-based "Small-Device C Compiler" (SDCC) suite, and an editor of your choice. (I installed the freeware "Crimson" editor that came on the CD.)

Ramtron's JTAG software packages flash memory programming and software debugging in a single program, although each function is relatively stand-alone. The programming portion handles basic flash memory operations and is really simple. The streamlined interface offers one-click programming (i.e., erase, write, and verify), while the ability to flash individual blocks and multiple files supports customization (e.g., serial number and calibration data).

Once you burn a program into flash memory, the debugger provides the critical link between your source code and the relatively sophisticated debug hardware built into the '3074 chip. There are six hardware breakpoints that work in the usual fashion (i.e., stopping the processor when a particular address is accessed). In addition, there's a "value"

```
ORG 2000h
Start:
  MOV  A, #10  ; Load ACC with 10
Loop:
  DEC  A
  JNZ  Loop
  LCALL Function ; Call a function
  JMP  Start
ORG 3000h
Function:
  RET          ; Dummy function
```

Branch	Loop repeat	Loop skip
1	0x2003	0x2003
2	0x2003	0x2005
3	0x2003	0x2008
4	0x2003	0x2003
5	0x2003	0x2005
6	0x2003	0x2008
7	0x2003	0x2003
8	0x2003	0x2005
9	0x2003	0x2008
10	0x2003	0x2003
11	0x2005	0x2005
12	0x2008	0x2008
13	0x2003	0x2003
14	0x2003	0x2005
15	0x2003	0x2008
16	0x2003	0x2003
17	0x2003	0x2005
18	0x2003	0x2008
19	0x2003	0x2003
20	0x2003	0x2005
21	0x2003	0x2008
22	0x2003	0x2003
23	0x2005	0x2005

Table 2—*The '3074's built-in debug hardware includes breakpoints and a low-budget real-time trace capability. As shown here, the latter has the option of tracing into or around tight loops.*

breakpoint option that stops the processor based on the value of a data item. Although a "value" breakpoint consumes two breakpoint slots, it can be a real timesaver compared to stopping on every access.

There's even a measure of real-time trace capability built into the chip. This is something I've seen on bigger and newer chips, but I can't recall seeing it on a '51. It tracks branches (i.e., JMPS, but unfortunately not CALLS or RETS) nonintrusively (i.e., unlike software "non-real-time" trace schemes) and has the option of skipping repeated entries (i.e., tight loops). Yes, it's kind of clunky just showing the raw addresses (see Table 2). Maybe a future revision could link those raw addresses back to labels in the source code or even provide a "Step Backwards" button. Nevertheless, like the "value" breakpoints, this mini-me real-time trace is a feature that could come in really handy when you're scratching your head over some intermittent "How in the heck did I end up here?" bug.

As a C compiler that supports "small devices" like the '51, '68, and PIC, I can say SDCC has its heart in the right place. Although not fully ANSI-compatible, it does have specific provisions for things like I/O, interrupts, and in-line assembler that reflect a sensitivity to the needs of memory-constrained real-time applications. For example, while the compiler fully supports 32-bit floating-point math, it also offers streamlined versions of library routines (e.g., `printf`) that minimize the associated bloat.

One thing that you don't get with SDCC (and don't pay for either) is a fancy IDE. Frankly, I think a lot of the popular commercial setups have feature-creeped beyond the needs of simple 8-bit applications. On the other hand, the old-school command-line setup (i.e., DOS box) used by SDCC, although capable enough in the hands of an expert, seems increasingly dated.

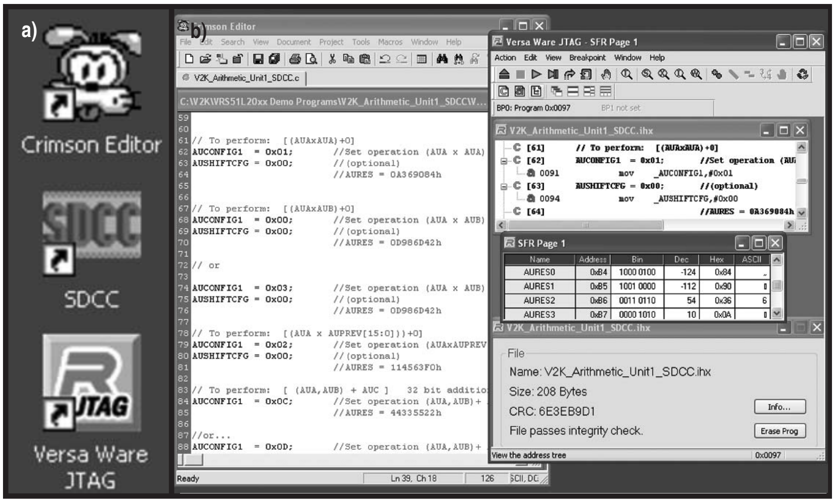


Photo 2—*Combining an editor with SDCC and the Ramtron JTAG debugger software* ***(a)*** *would seem like a rather primitive solution, but in fact it's surprisingly effective and easy to use* ***(b)****. And with a $0 price tag, you're definitely getting more than you pay for.*

A decent compromise is to exploit the fact that most programming-oriented editors include the capability to define editor commands and hotkeys that invoke external programs. Following the directions included with the Ramtron kit, I was able to hook SDCC into the Crimson editor for seamless point-and-click (i.e., no command line) operation (see Photo 2).

NEWS FLASH

After all is said and done, I'd say the "FRAM Enhanced" headlines for the '3074 kind of miss the point. First, as I described earlier, the actual FRAM functionality seems a bit disappointing, especially in light of the more SRAM-like (fast, symmetric read/write) specs for their 1-Mb memory I covered last month. Indeed, I wonder if a better stacked-die option wouldn't just combine that chip with a de-flashed (and maybe even de-RAMed?) '51 MCU core.

Further irony given all the "flash is dead" hoopla is found in the fact that the flash memory on the '3074 actually works quite well, notably delivering the goods at 40 MHz with no wait states. Yes, it's got the usual endurance, retention, and write speed/power limitations, but at least it's easy to use by virtue of a built-in flash program interface (FPI) supplementing the external (i.e., JTAG) programming option. Invoking the FPI shadows in high-level flash memory functions (overlaying the top 1 KB of code space) that take care of all the dirty work for in-application programming.

Thanks to the FPI, using the flash memory is as easy and arguably less prone to glitches and head-scratching than using the FRAM.

This isn't a knock on FRAM in general. Ramtron's standalone parallel and serial memory chips have really grown up and are already proving themselves in real applications. Rather, it's just the specific implementation in the '3074 that seems a bit puzzling. It is all a wakeup call that FRAM and other new age memory technologies are headed in the right direction but have a ways to go.

The popularity and longevity of the '51 uniquely derives from a "more the merrier" supply side—the Ramtron parts being the latest, although surely not last, addition. Putting aside the "FRAM Enhanced" headline for the moment, maybe the story worth noting is that the '3074 and Ramtron's other '51s seem pretty darned competitive. Sure, there are still some rough edges, but I'd say their combination of FRAM know-how and mixed-signal '51 expertise gives Ramtron a decent chance for success. ■

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

REFERENCE

[1] B. Millier, "Battery-Free Nonvolatile RAM," *Circuit Cellar* 193, 2006.

SOURCES

Crimson Editor
www.crimsoneditor.com

VRS51L3074 FRAM-Enhanced '51 MCU
Ramtron International Corp.
www.ramtron.com

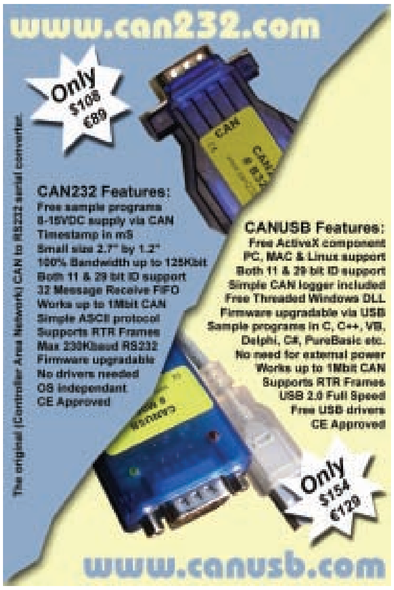
Small-Device C Compiler
http://sdcc.sourceforge.net/

IDEA BOX

THE DIRECTORY OF PRODUCTS AND SERVICES

AD FORMAT: Advertisers must furnish digital submission sheet and digital files that meet the specifications on the digital submission sheet. **ALL TEXT AND OTHER ELEMENTS MUST FIT WITHIN A 2″ × 3″ FORMAT**. Call for current rate and deadline information. Send your disk and digital submission sheet to: IDEA BOX, Circuit Cellar, 4 Park Street, Vernon, CT 06066 or e-mail adcopy@circuitcellar.com. For more information call Shannon Barraclough at (860) 872-3064.

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


www.can232.com
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
The original (Controller Area Network) CAN to RS232 serial converter.
Only $154 €129
www.canusb.com

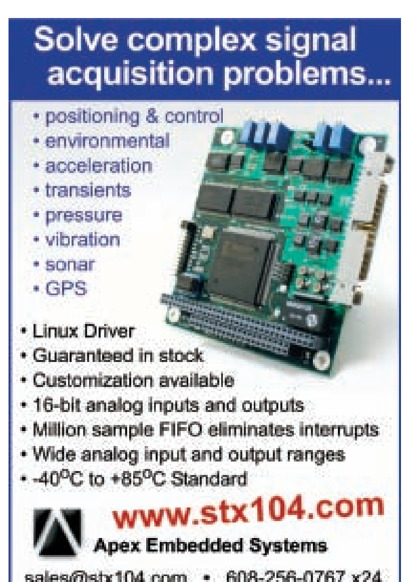

ANSI C software development tools for:
IMAGEcraft
NEW!: Global Optimizer
CypressM PSoC
AVR & tinyAVR
HC08,11,12,16
TI MSP430
ARM
Version 7 Tools
Full-featured 45-day
demos on our website
hardware kits:
Elektronikladen - HC08 & HCS12 / 12 ref. design & starter kits
CANDIP - AVR/CAN on a chip!
Ethernut - Complete AVR TCP/IP solution
high powered development, garage powered prices!
www.imagecraft.com
VISA / MC / AMEX
Check / Money Order
info@imagecraft.com
706 Colorado Ave. Palo Alto, CA 94303
(650) 493-9326 • FAX (650) 493-9329
...say 'NO' to JURASSIC PRICES!

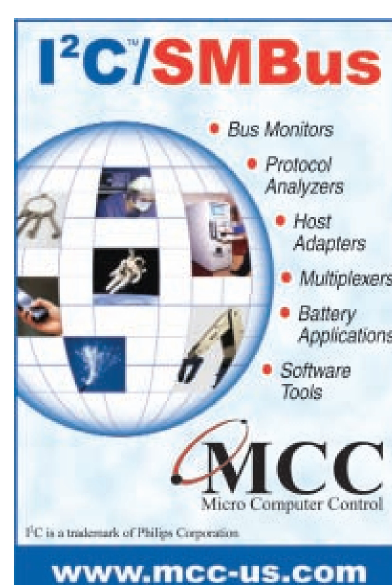

Get The Whole Picture
Camera Interface Application Kit for remote monitoring, security and event-capture via the internet.
Kit includes:
• RCM3365
• Serial Camera
• Servos
• Dynamic C®
• Dev. Tools
Buy Now $499
RABBIT Semiconductor
www.rabbitappkits.com


When contacting our advertisers, tell them where you found their ad...
Advertiser
...tell them, "Circuit Cellar sent me."


JTAG & In-Circuit
EMULATORS
IDEs, Compilers, Programmers
80196
AVR
SENSORY
XEMICS
8051
ARM
PIC
bq
MAXQ
MSP430
Phyton
718.259.3191
www.phyton.com


NEED AN INTERFACE TO YOUR KEYBOARD PORT?
INTERFACE:
- KEYPADS
- SWITCHES
- RS-232
TO YOUR PC'S KEYBOARD OR USB PORT
888-690-9080
HAGSTROMELECTRONICS.COM
11 FIDDLERS GREEN, LANSING NY 14882

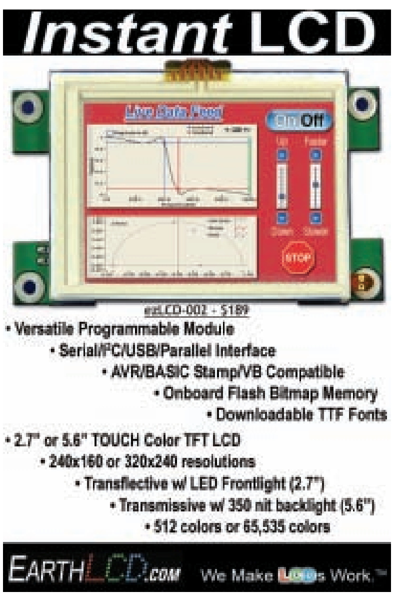

Solve complex signal acquisition problems...
• positioning & control
• environmental
• acceleration
• transients
• pressure
• vibration
• sonar
• GPS
• Linux Driver
• Guaranteed in stock
• Customization available
• 16-bit analog inputs and outputs
• Million sample FIFO eliminates interrupts
• Wide analog input and output ranges
• -40°C to +85°C Standard
www.stx104.com
Apex Embedded Systems
sales@stx104.com • 608-256-0767 x24

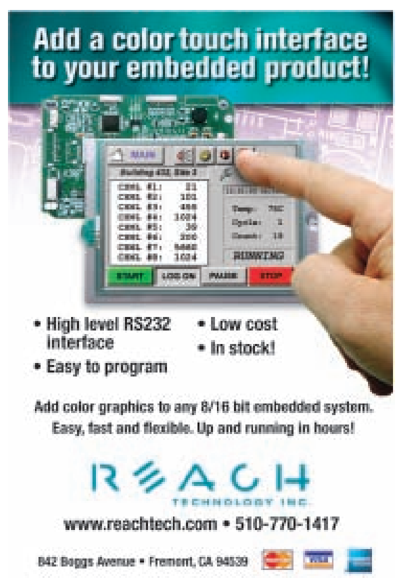

I²C/SMBus
• Bus Monitors
• Protocol Analyzers
• Host Adapters
• Multiplexers
• Battery Applications
• Software Tools
MCC
Micro Computer Control
I²C is a trademark of Philips Corporation
www.mcc-us.com


MYLYDIA, INC.
Layout Gerber,
Prototype Making
QUICK TURN
PCB & Turnkey
@
The Best Prices
Tel: 800-695-9342
Sales@mylydia.com
WWW.MYLYDIA.COM


CHINA PCB SUPPLIER
DIRECT SALE FROM CHINA
PROTOTYPE TO PRODUCTION
instant online quote
shopping cart ordering system
China competitive prices
free Electrically test
WWW.PCBCART.COM

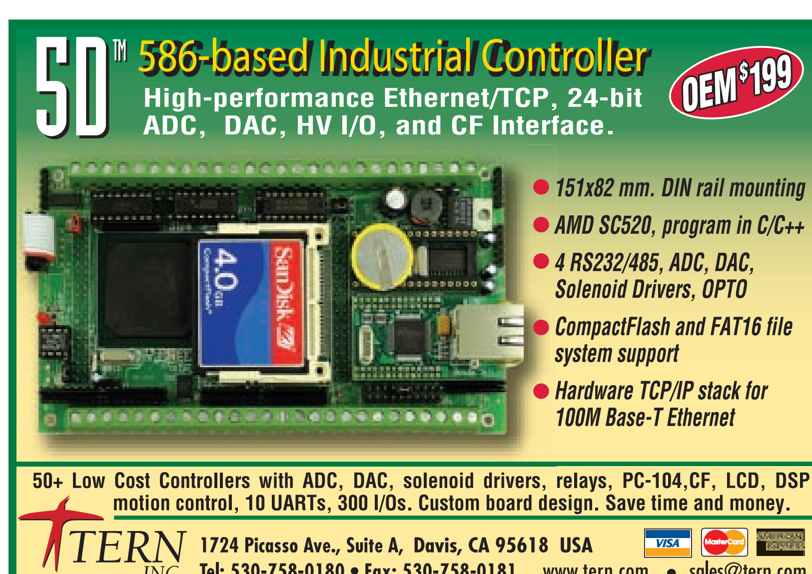

SmartHomeUSA
SAVE 10%
ENTER COUPON CODE CELLAR DURING CHECKOUT
www.SmartHomeUSA.com
DIY Home Automation, Security & Entertainment Products


Instant LCD
ezLCD-002 - $189
• Versatile Programmable Module
• Serial/I²C/USB/Parallel Interface
• AVR/BASIC Stamp/VB Compatible
• Onboard Flash Bitmap Memory
• Downloadable TTF Fonts
• 2.7" or 5.6" TOUCH Color TFT LCD
• 240x160 or 320x240 resolutions
• Transflective w/ LED Frontlight (2.7")
• Transmissive w/ 350 nit backlight (5.6")
• 512 colors or 65,535 colors
EARTHLCD.COM We Make LCDs Work.


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


PCBFABEXPRESS
High Quality PCBs @ Low Impact Prices
Get
5 PCBs for
$13 each in
5 days
2 layer, 20 sq. in., FR-4, 0.062" thick
FREE Tooling, FREE Soldermask,
FREE Silkscreen
Visit website to see all offers
on 2 to 8 layer PCBs
ORDER ONLINE
408-522-1500
www.PCBFabExpress.com
Also see our fantastic
PCB Assembly prices online


MEASUREMENT SCIENCE CONFERENCE 2007
APPLY METROLOGY . . .
RULE the WORLD
January 22-26, 2007
Long Beach Convention Center
Long Beach, California
GREAT New Venue
EXCELLENT training opportunities
MORE than 100 Exhibit Booths (Wed-Fri)
NETWORK with experts from
government, industry, academia
Register today!
(866) 672-6327
www.msc-conf.com


5D™ 586-based Industrial Controller
High-performance Ethernet/TCP, 24-bit ADC, DAC, HV I/O, and CF Interface.
OEM $199
● 151x82 mm. DIN rail mounting
● AMD SC520, program in C/C++
● 4 RS232/485, ADC, DAC, Solenoid Drivers, OPTO
● CompactFlash and FAT16 file system support
● Hardware TCP/IP stack for 100M Base-T Ethernet
50+ Low Cost Controllers with ADC, DAC, solenoid drivers, relays, PC-104,CF, LCD, DSP motion control, 10 UARTs, 300 I/Os. Custom board design. Save time and money.
TERN INC.
1724 Picasso Ave., Suite A, Davis, CA 95618 USA
Tel: 530-758-0180 • Fax: 530-758-0181 www.tern.com • sales@tern.com


Serial • Ethernet
Web Server
Simple, DTE, DCE
No SW Changes
$99.95
Qty 1
ET232
+1 630-245-1445
Naperville, Illinois USA
www.gridconnect.com
gridconnect


NAND FLASH
FILE SYSTEM
• Wear Leveling, Garbage Collection
• Power-Fail Safe
• ECC and Bad Block Handling
• File System with Std C Lib API
• FAT File Structure
• Page Caching to Maximize Performance and Minimize RAM
• 20KB Code Size
• RAM Required < 1% Flash Size
www.smxrtos.com/ffscc.htm
Micro Digital Inc
RTOS Innovators
800.366.2491 sales@smxinfo.com


1-Wire
Weather
Instruments
for PCs
AAG
electrónica
www.aagelectronica.com


High Quality Enclosures
The Most Competitive Price
* Sheet Metal Cabinets
* Aluminium Extruded Enclosures
* 19inch Standard Cabinets
* Plastic Injection
* Precision Aluminium Die Casting
* Highly Custom Made
* Fast Prototyping
* From 1pc to 10, 000, 000 pcs
WWW. EzPCB. COM

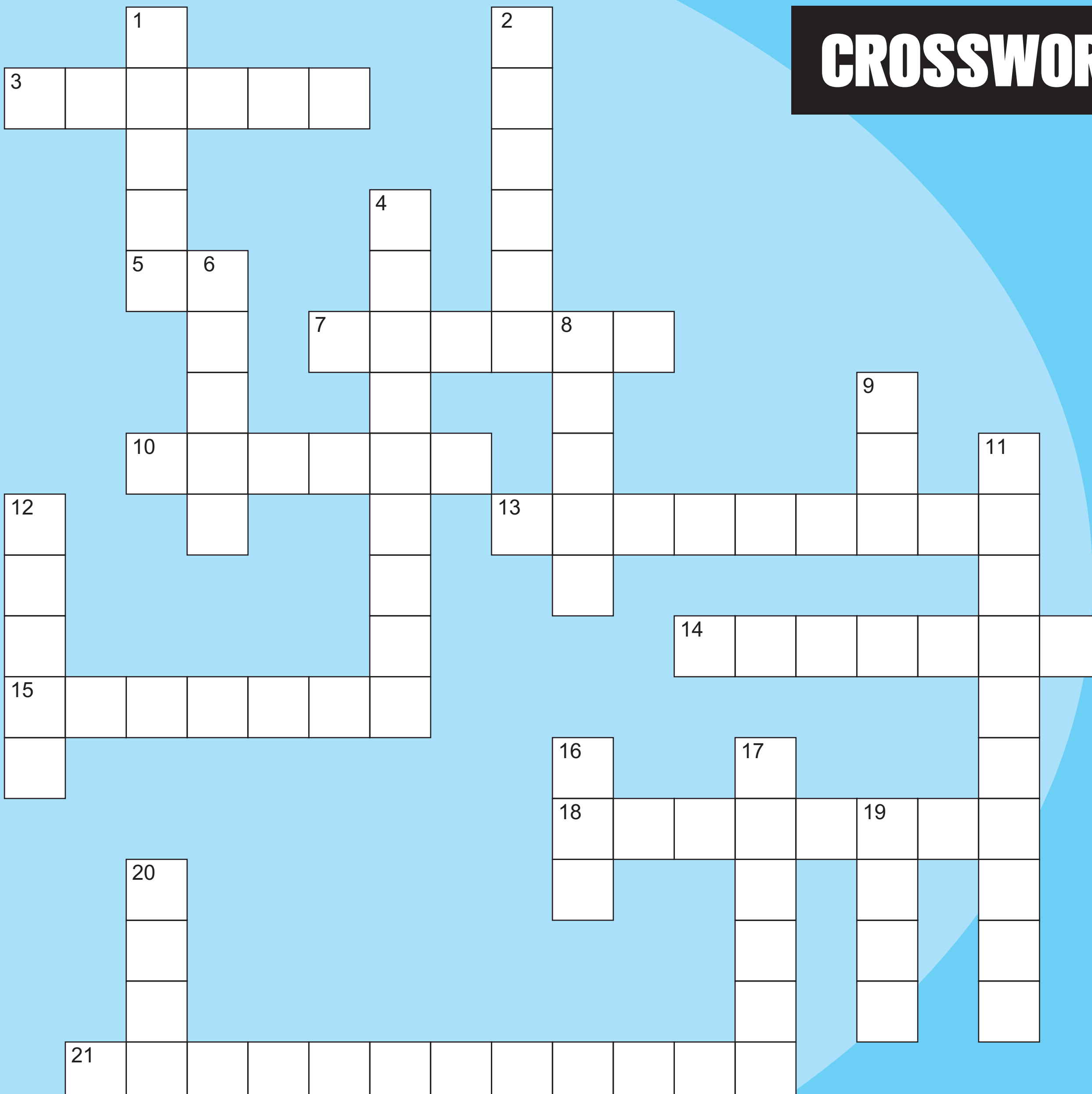

Embedded Ethernet
ONLY $98
• 10Base-T Ethernet
• 186 Processor @ 40 MHz
• DOS w/ Flash File System
• 16 Digital I/O • 5V DC
• Console / Debug Port
• Hardware Clock/Calendar
• Socket for DiskOnChip
• 512K DRAM & 512K Flash
• (2) Serial Ports
• (2) 16-bit Timers
• Watchdog Timer
• 3.75" x 2.50"
NEW! picoFlash
Development Kit Includes:
$129
Development System
• picoFlash Controller
• Borland C/C++ 4.52
• TCP/IP, PPP & Web Server
• Serial Driver Library
• AC Adapter and Cables
Call 530-297-6073 Email sales@jkmicro.com
On the web at www.jkmicro.com
JK microsystems


NOW YOU CAN HAVE IT ALL!
• Easy to Use
• Low Cost
• Low Power
• Tiny Size
• High Speed
• DOS Environment
with the
DOS Stamp™
From $99 Q.100
Easy Software Development: Use your C/C++ or Basic compiler to produce DOS EXE. Download EXE to flash disk via serial port. EXE runs on power up.
Standard Features: BIOS & DOS-ROM, 128K flash disk, 512K SRAM, 40 MHz CPU, 16 digital I/O (opto rack interface), 2 RS-232, 2 timer/counters, simple bus interface, real-time clock
Options: 8-ch 12-bit ADC, flash disks up to 288 MB, 1 RS-485, TCP/IP
Tiny Size, Low Power: 2"x2.6", 5V @ 200 mA at full speed, 40 mA at lowest speed.
Visit http://www.bagotronix.com for info, prices, and FAQs
BAGOTRONIX
2900-1 Crescent Drive
Tallahassee, FL 32301
850-942-7905 phone & fax


ANYONE
Can Now Easily
Hand Solder Surface-Mount Components!
Even A 10
Year Old!
www.schmartboard.com


PRINTED CIRCUIT BOARDS
QUALITY PRODUCT
FAST DELIVERY
COMPETITIVE PRICING
PROTOTYPE through PRODUCTION
• 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
SPECIAL OFFER:
10 pcs (3days) 1 or 2 layers $249
10 pcs (5days) 4 layers $695
(up to 30sq. in.ea.) includes tooling, artwork, L.P.I. mask & legend
PULSAR, INC
9901 W. Pacific Ave. Franklin Park, IL 60131 • Phone 847.233.0012
Fax 847.233.0013 • www.pulsar-inc.com • sales@pulsar-inc.com

CROSSWORD



Accross

3. GND
5. Modem light: "Ready"
7. Former U.S. president (b. 1924) with a background in nuclear physics and the study of reactor technology. Hint: 39
10. A partition, or screen, placed between a microphone and a source of sound.
13. A punctuation mark, number, letter, or symbol.
14. Eight-sided polygon.
15. A character that you are not allowed to use when naming a file.
18. A low-density white metallic element.
21. The *H* in H-menu.

Down

1. A place online where you can meet other engineers and discuss important topics and ideas.
2. 1,000,000,000,000,000,000 bits
4. Postal delivery. Hint: mollusk
6. The IBM computer introduced in 1956 that featured disk storage.
8. Twenty-four is the least common multiple of 12 and which number?
9. Extension for a batch file.
11. Field of study that deals with how humans interact and work with machines.
12. The dog *Circuit Cellar* founder/editorial director Steve Ciarcia has mentioned in his editorials.
16. Text message: "By the way"
17. A PC that features an Intel processor and a Mac operating system.
19. Text message: "In my humble opinion"
20. A standard for connecting peripherals and PCs. Hint: "scuzzy"

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

INDEX OF ADVERTISERS

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

Preview of January Issue 198
Theme: Embedded Applications

Multi-Input Temperature Logger

The Power of Flash: Use Flash Memory for More than Storing Code

Arby: An Arbitrary Waveform Generator with a Twist

Voltage Solutions: Harness the Power of Voltage Converters

Self-Powered Solar Data Logger

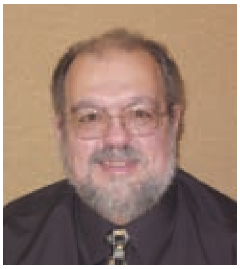
QuickComs: An M16C/62P-Based RS-232 Analyzer

FROM THE TRENCHES Hello World ... Want Cookie

APPLIED PCs Dive Into the ZigBee Pool: An Easy Way to Start Moving Messages

FROM THE BENCH Green Energy

SILICON UPDATE Hot Chips 18



PRIORITY INTERRUPT

by Steve Ciarcia, Founder and Editorial Director

Going for the Brass Ring

I don't know whether it's genetic predisposition, but engineers are very predictable. Think about other people for a minute. Someone with new-found artistic talents paints a masterpiece and what does he do? Open a studio or an art gallery? Nope. He hangs it on the wall. Another person aspiring to be a gourmet cook creates the piece de resistance at a dinner party and what is their next thought? Start a restaurant? Nope. They just want everyone to dig in and eat it. An accomplished administrator refinishes and converts an old bureau into a magnum opus by night. Is his next thought to open a restoration shop? Nope. It's how to finish the meeting and find the next old piece of furniture.

The business crowd likes to refer to us as a bunch of geeks, but I think it's quite the opposite. An engineer designs a widget and publishes it in *Circuit Cellar* or is among the winners in our design contests and what is his next thought? Go to work tomorrow and do the same old job? Nope. He's trying to figure out how to start a company to manufacture it.

More than any other subject, the question of what should engineers do in the long term fills my in-basket. Perhaps it is the realization that all of our technical training is aimed at designing widgets that someone else makes money manufacturing, or maybe it's the insecurity of knowing that we become dispensable when we can no longer create great widgets for an employer, that makes us feel more like a traded commodity than a tenured professional. Longevity in this business isn't assured, and this isn't a union job.

The typical career path that most engineers take only adds to the dilemma. While there are some companies that still have a promotion and salary structure that rewards engineers who stay "technical," the majority of companies reserve high-paying jobs strictly for business-side management. After a few years of designing widgets, the only way to raise your salary is to become a project manager or otherwise "oversee" others doing the real design work. This only adds to fears of obsolescence. Are you a real engineer who can get another job, or are you now a project manager (whatever that is)?

I got an e-mail this week from a Luminary Micro contest contestant. Like many of our readers, he is entering a contest project specifically because of engineering career anxiety. After 11 years of working for a big company in Silicon Valley, he now finds himself worrying about being outsourced or merged. His logic is that the contest serves a dual purpose. Knowing Stellaris ARM refreshes his engineering credentials with leading-edge design knowledge for his resume and being able to reference a winning or Distinctive Excellence design posted on *Circuit Cellar* and Luminary Micro adds instant credibility. His title may be a project manager, but this proves he is still an engineer.

The second and loftier goal is using the contest as an incentive to invent the perfect widget. Deep down, most engineers know that the only hedge to the corporate rat race is being top dog instead of one of the minions. Of course, not everyone is cut out for self-employment, and getting there has great risks. The goal for many engineers is to bootstrap the process by designing something on the side that hopefully receives royalties or attracts manufacturing investment money from others. This happens more often to our contest project entrants than you might think.

I'm not sure why anyone would want to ask my opinion, but perhaps it's because I seem to have followed the career path they seek. A long time ago, I was one of those engineers in corporate America wondering about the future. It took five different jobs, but ultimately I could see the handwriting on the wall all too clearly. As you know, I designed a bunch of widgets on the side and my publicity was *BYTE*. Ultimately, the interest in the designs was enough that they were manufactured. Like the fantasy, I got to play top dog at the company for a number of years and start a magazine too.

I guess I've followed the course that most engineers seek and had a good time. Before you quit your day job to go for the brass ring, however, I should caution you about taking it in steps. If you start a company and work at it for a reasonable length of time, you join the club. Club benefits include a good income, no office politics, freedom to do what you want, and as many company perks as you and the IRS can negotiate. The downside "benefit" is that you can't go back. Once you have a taste of real business freedom, you become non-corporate (no longer a committee person). At that point, you either have to make it really big and retire early or stay in the club because you can never be happy working in big-company America again (and all the crap that goes with big-company jobs).

So, it's safe to continue renewing your subscription for quite a while. I may be an engineer who still knows a thing or two technically and could get a job. But, I've definitely determined that present club membership precludes it.

steve.ciarcia@circuitcellar.com


DigiView
100 MHz, 18 Channel, Portable Logic Analyzer
Auto Hardware Compression Captures from 128K to 30 Billion Samples @10ns resolution
only $499.00
DigiView
TechTools Logic Analyzer
www.tech-tools.com
• Automatic Real-time Hardware Compression can save up to 5 minutes of data @ 10ns.
• Display I2C, Synchronous (SPI), Asynchronous (RS-232), State, Boolean, Bus and Analog Data.
• Edge and Pattern Triggers.
• Pattern Searches with Match, Duration & Animation.
• Specialized Sequential Searches for Serial and State Mode Signals.
• Drag & Snap Markers.
• Dual Waveform View with signal edge snapping.
• Multi-Signal Data Tables with Link Groups.
• Specialized Exports and List Views for Serial Signals.
• Print or Save Images with custom comments.
• USB Powered for Portability.
• USB 1.1 & 2.0 compatible.
Now with Serial Signal Analysis
Download and try the Software!
Cables & Micro-Clips Included
Professional Capture & Analysis Software Included
QuickWriter™
PICmicro® MCU Programmer
Multi-Function In-Circuit & Gang Operation
only $199.00
• Supports 10, 12, 14, 16 & 18F Series PICmicro MCUs.
• Edit Code, EE Data, IDs and Configuration Fuses.
• Auto or Manual Serialization in Code or EE Data.
• Firmware auto synchronizes to Software version.
• Control Files protect Option Settings.
• Accepts Command-Line Parameters.
• Tests and Monitors Voltages.
• Control Signal for Self-Powered Circuits.
• Single and 4 Gang Adapters available.
• Includes Cables, Software & U.S. Power Supply.
EconoROM III™
ER3
EPROM & FLASH Emulation at its best!
from $179.00
FlexROM III™
FlexROM III
STATUS
OUT
IN
QuickLoader™
TechTools
www.tech-tools.com
(972) 272-9392 • sales@tech-tools.com
Copyright © 2006 TechTools • DigiView, FlexROM, EconoROM and QuickWriter are trademarks of TechTools • PICmicro is a registered trademark of Microchip Technology Inc.

GPS Module

Does your latest project need to be able to track its position globally? When used in an outdoor environment, the Parallax GPS Receiver with its integrated patch antenna can provide standard raw NMEA0183 strings or specific data from up to 12 satellites via a standard serial interface.

The module provides current time, date, latitude, longitude, altitude, speed and travel direction/heading among other data, and can be used in a host of applications, including navigation, tracking systems, mapping, fleet management and robotics.

The Parallax GPS Receiver Module includes a programmable Parallax SX microprocessor and open-source control firmware for advanced users who wish to modify it.

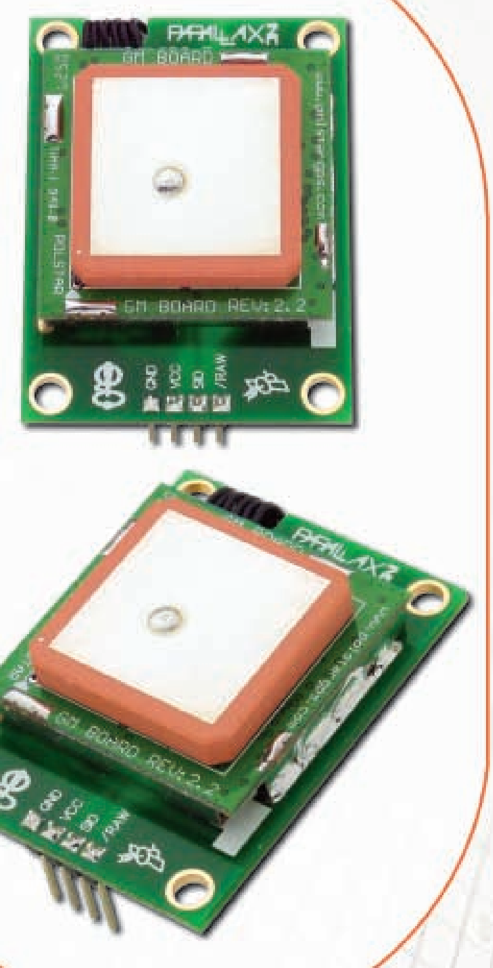
Features:

- Single-wire 4800 bps TTL Serial interface to BASIC Stamp®, SX and Propeller microcontrollers.
- Provides either raw NMEA0183 strings or specific data, making it easier to use on the BASIC Stamp microcontroller.
- Simple/raw mode selectable via I/O control.
- Single +5VDC supply @ 115 mA (typical) Standard 4-pin header interface (0.100” spacing).

To order online visit www.parallax.com or call the Parallax Sales Department toll-free at 888-512-1024 (Mon-Fri, 7am-5pm, PT).

PARALLAX

www.**parallax**.com



introductory sale price

$79.95

#28146