Innovative Airflow Analysis, p. 60 • ARM Architecture: The Story From "Computer" to "MCU," p. 66

www.circuitcellar.com

# CIRCUIT CELLAR

THE MAGAZINE FOR COMPUTER APPLICATIONS

#231 October 2009

## SIGNAL PROCESSING

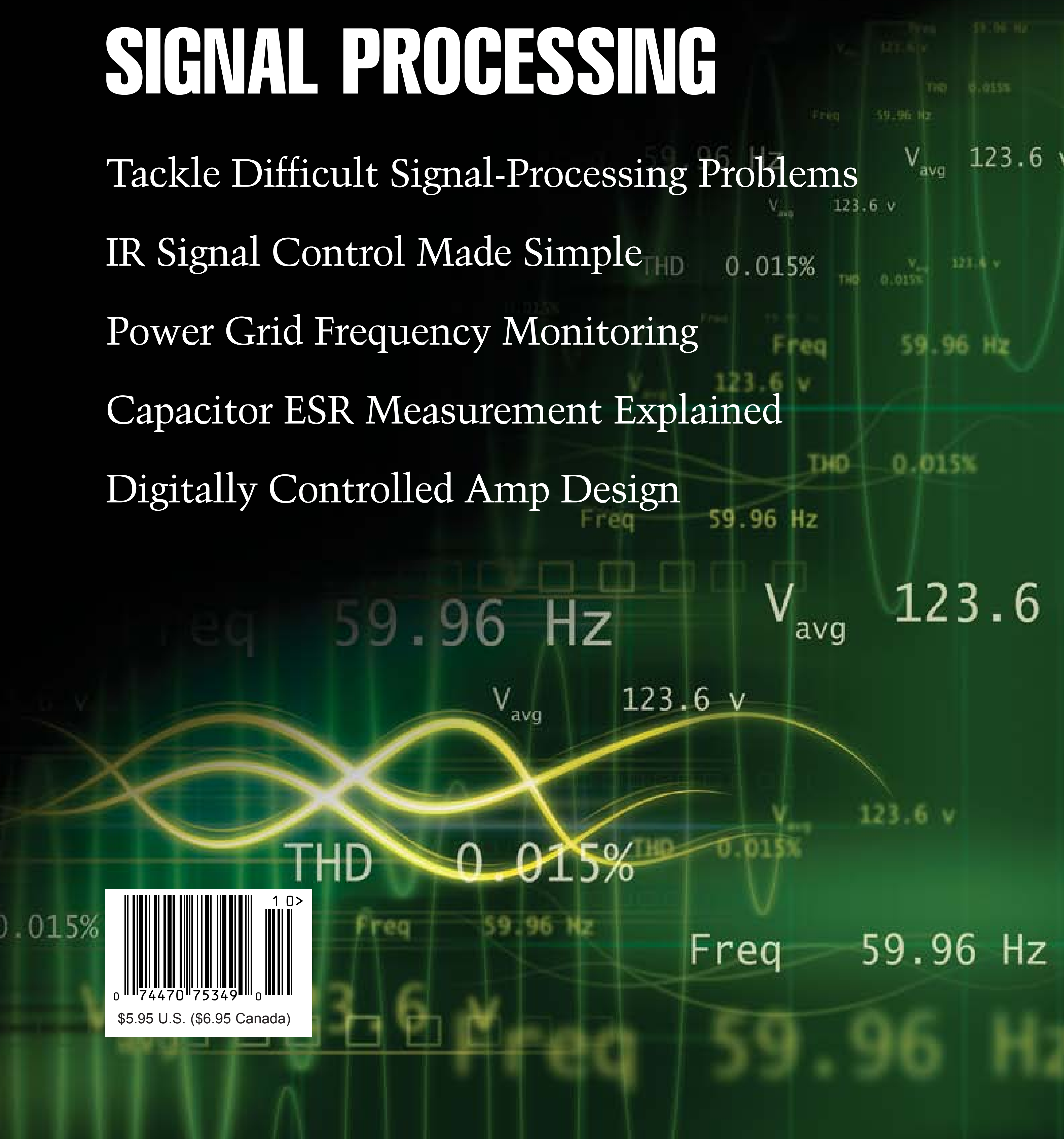
Tackle Difficult Signal-Processing Problems

IR Signal Control Made Simple

Power Grid Frequency Monitoring

Capacitor ESR Measurement Explained

Digitally Controlled Amp Design



$5.95 U.S. ($6.95 Canada)

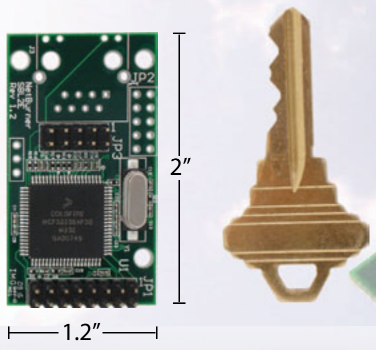
A Low Cost Solution for

# Industrial Serial to Ethernet

with Digital I/O and Analog to Digital Converters

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

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



**$19.95**
Qty. 1000

The all-new

**NetBurner SBL2e**

INDUSTRIAL SERIAL TO ETHERNET SOLUTION

*A NetBurner Complete Hardware & Software Solution*

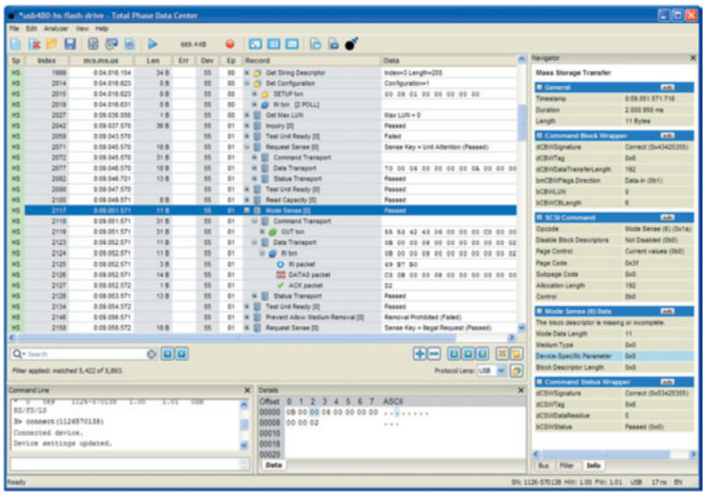
NetBurner
Networking in One Day!

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

freescale™
Alliance Member

Attention **USB** Developers!
Class is now in Session
I will get free HID, Mass Storage, Hub, and Audio class decoding.
I will get free HID, Mass Storage, Hub, and Audio class decoding.
BEAGLE™ USB 480
TOTAL PHASE
www.totalphase.com

## Debug USB Faster with Data Center Software



Real-time class-level decoding is now available in the Data Center Software for the Beagle USB 480 Protocol Analyzer.

Within a few clicks, class-level USB data is instantly decoded and displayed as it is being captured.

Debug USB with class. Get yours now.

Please visit our website for our complete line of USB, I2C, & SPI tools.

TOTAL PHASE
www.totalphase.com

# The Newest Semiconductors

*New Products from:*

Texas Instruments Authorized Distributor

*TMS320C672x Digital Signal Processor*
*www.mouser.com/tidsp/a*

Cirrus Logic

*CS48540 32-bit DSP*
*www.mouser.com/cirruslogic/a*

freescale semiconductor

*MC56F8006 / MC56F8002 Digital Signal Controller*
*www.mouser.com/freescale/a*

*The ONLY New Catalog Every 90 Days*

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

MOUSER ELECTRONICS®
a tti company

*The Newest Products For Your Newest Designs*

(800) 346-6873

**www.mouser.com**

*Over A Million Products Online*

## Development Solutions

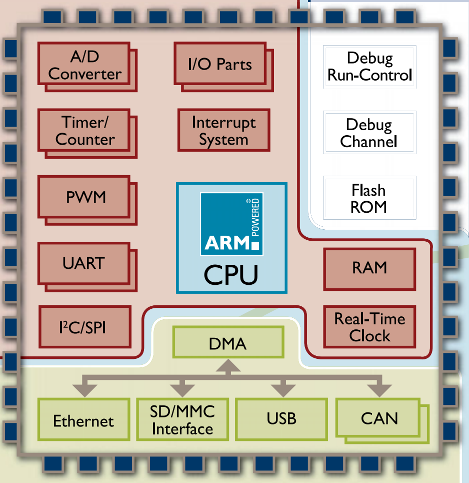
KEIL™ Tools by ARM

## for ARM®, 8051 & XE166 Microcontrollers

### C and C++ Development Tools
- Best-in-class Keil and ARM compilers for small, fast code
- Genuine Keil µVision® IDE/Debugger/Simulator
- Fast development and verification using Device Simulation
- Easy device configuration with Device Database support for more than 1700 devices

### Microcontroller Development Kits
- C and C++ Compilers
- Royalty-Free RTX RTOS
- µVision Device Database & IDE
- µVision Debugger & Analysis Tools
- Complete Device Simulation

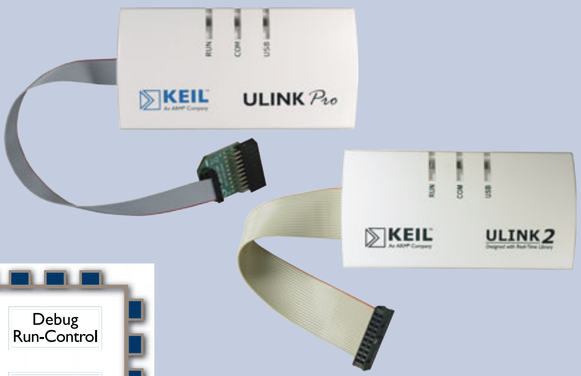
Examples and Templates



### RTOS and Middleware Components
- RTX RTOS Source Code
- TCPnet Networking Suite
- Flash File System
- USB Device Interface
- CAN Interface

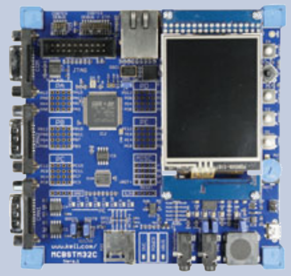
Examples and Templates

### RTOS and Middleware
- Easy implementation of complex applications
- Royalty-free RTX Real-Time Kernel
- TCP/IP Suite with Server Applications
- File System for ROM and Memory Cards
- Direct Support for USB and CAN interfaces



### ULINK® USB Adapters
- JTAG, SWD & SWV support
- Debugging and Flash download
- ARM device support
- Data Trace for Cortex-M3
- ARM Instruction Trace (ULINK*Pro*)
- 8 and 16-bit support (ULINK2)

### Evaluation Boards



*Keil provides a wide range of evaluation boards for 8, 16 and 32-bit devices*

**Keil PK51, PK166, and MDK-ARM**
support more than 1700 microcontrollers
**keil.com/dd**

**Download the µVision4 Beta Version**
**keil.com/uv4**

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

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

## TASK MANAGER

### Signal Significance

Readers and writers know we typically run about one or two "theme-related" articles per issue. We do this because we have a wide readership with diverse interests. We try to please as many people as possible. And we've been doing that well since 1988. What's always interesting about the Signal Processing issue is that almost every article we run (as well as many of those considered for publication) has something to do with signal theory, processing, or control in one way or another. You don't need a PhD with a focus on signal processing to be an embedded design engineer. But you definitely need a good grounding in signal theory to be a successful one, whether your end goal is a marketable embedded wireless product or simply a handy design for your workbench.

This Signal Processing issue is true to form. As you'll see, the topics of signal processing and control figure prominently in most of the subjects covered.

In "IR Signal Control," Naubert Aparicio presents a design for offsite control of an electrical system (p. 12). He uses the IR signal controller design to remotely operate his home's AC system from his cell phone. You can use the techniques Naubert covers to customize an IR signal control project of your own.

Frequency monitoring is the topic presented in Arnold Stadlin's article, "Frequency Sensing Made Simple" (p. 22). In this article, he describes a microcontroller-based power frequency monitor design that can indicate the relative stability of a power grid. The design measures power grid frequency via a standard electrical outlet. The acquired data is transferred to a PC, which displays the information and relays it to a web server for real-time analysis.

As you probably know, many instruments don't measure equivalent series resistance (ESR). Enter Ed Nisley. After studying the design of a classic pure-analog ESR meter, he built an analog front end for an MCU that measures ESR. In "Capacitor ESR Measurement," he covers the project's design and math (p. 28).

Starting on page 38, George Anderson presents a dsPIC30F2023-controlled, high-end vacuum tube stereo amplifier with distortion control, power output, and more. He describes unique circuitry and specialized processing software.

Turn to page 50 for information about multirate techniques and cascaded integrator-comb (CIC) filters. Robert Lacoste explains how you can use these moving average filters to tackle difficult signal-processing issues. When a FIR filter won't do the trick, this might be the perfect solution.

In "Airflow Analysis," Jeff Bachiochi describes an effective way to measure airflow in an air duct (p. 60). He explains how he took apart a small CPU fan, calculated RPMs, and implemented a circuit to output data to an LCD.

Remember when Acorn Computers introduced its RISC CPU? Tom Cantrell has been following the story for nearly two decades. In "Thumbs Up: The ARM Saga Continues," he brings you up to speed with a review of the ARM Cortex-M0 core (p. 66). The story continues.

cj@circuitcellar.com

C. Abate

# CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

**FOUNDER/EDITORIAL DIRECTOR**
Steve Ciarcia

**MANAGING EDITOR**
C. J. Abate

**WEST COAST EDITOR**
Tom Cantrell

**CONTRIBUTING EDITORS**
Jeff Bachiochi
Robert Lacoste
George Martin
Ed Nisley

**NEW PRODUCTS EDITOR**
John Gorsky

**PROJECT EDITORS**
Gary Bodley
Ken Davidson
David Tweed

**CHIEF FINANCIAL OFFICER**
Jeannette Ciarcia

**MEDIA CONSULTANT**
Dan Rodrigues

**CUSTOMER SERVICE**
Debbie Lavoie

**CONTROLLER**
Jeff Yanco

**ART DIRECTOR**
KC Prescott

**GRAPHIC DESIGNERS**
Grace Chen
Carey Penney

**STAFF ENGINEER**
John Gorsky

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

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

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

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

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

**PRINTED IN THE UNITED STATES**

**CONTACTS**

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

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

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

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

CIRCUIT CELLAR®, THE MAGAZINE FOR COMPUTER APPLICATIONS (ISSN 1528-0608) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Vernon, CT 06066. Periodical rates paid at Vernon, CT and additional offices. **One-year (12 issues) subscription rate USA and possessions $29.95, Canada/Mexico $34.95, all other countries $49.95. Two-year (24 issues) subscription rate USA and possessions $49.95, 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.

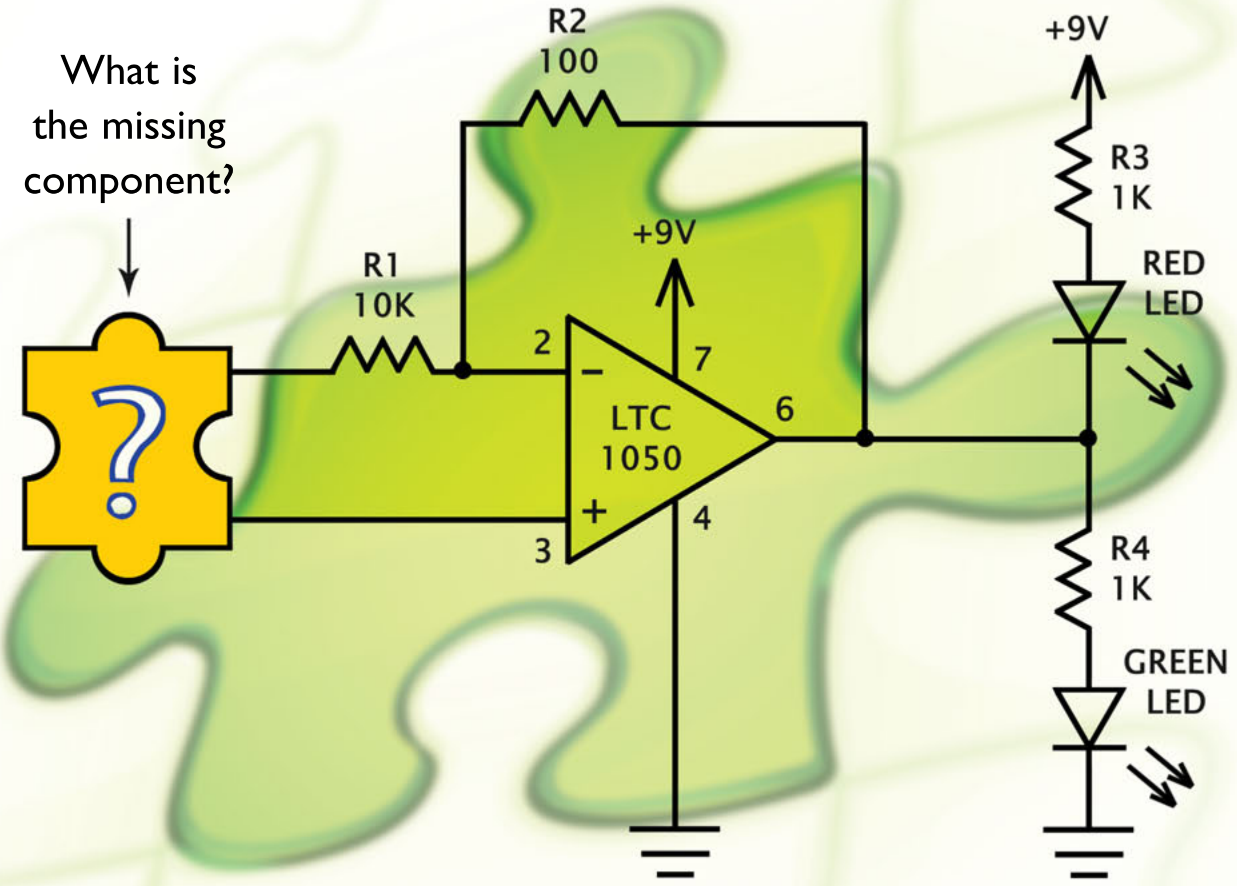
Circuit Cellar® makes no warranties and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors. Furthermore, because of possible variation in the quality and condition of materials and workmanship of reader-assembled projects, Circuit Cellar® disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from plans, descriptions, or information published by Circuit Cellar®.

The information provided by Circuit Cellar® is for educational purposes. Circuit Cellar® makes no claims or warrants that readers have a right to build things based upon these ideas under patent or other relevant intellectual property law in their jurisdiction, or that readers have a right to construct or operate any of the devices described herein under the relevant patent or other intellectual property law of the reader's jurisdiction. The reader assumes any risk of infringement liability for constructing or operating such devices.

Entire contents copyright © 2009 by Circuit Cellar, Incorporated. All rights reserved. Circuit Cellar is a registered trademark of Circuit Cellar, Inc. Reproduction of this publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.

# Are you up for a challenge?

What is the missing component?



Industry guru Forrest M. Mims III has created a stumper. Video game designer Bob Wheels needed an inexpensive, counter-clockwise rotation detector for a radio-controlled car that could withstand the busy hands of a teenaged game player and endure lots of punishment. Can you figure out what's missing? Go to www.Jameco.com/unravel to see if you are correct and while you are there, sign-up for our free full color catalog.

JAMECO® ELECTRONICS

1-800-831-4242

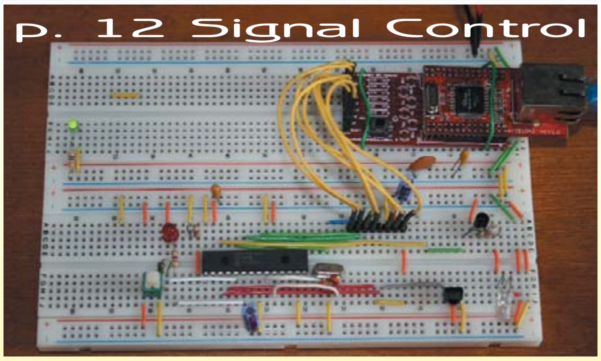
# INSIDE ISSUE 231

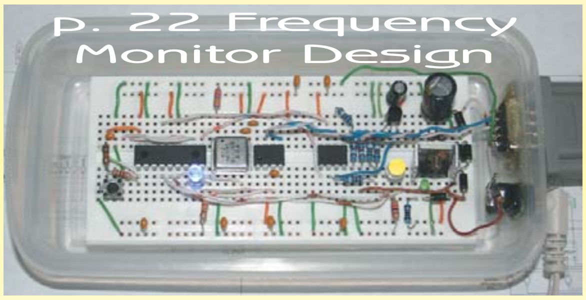
October 2009 • Signal Processing

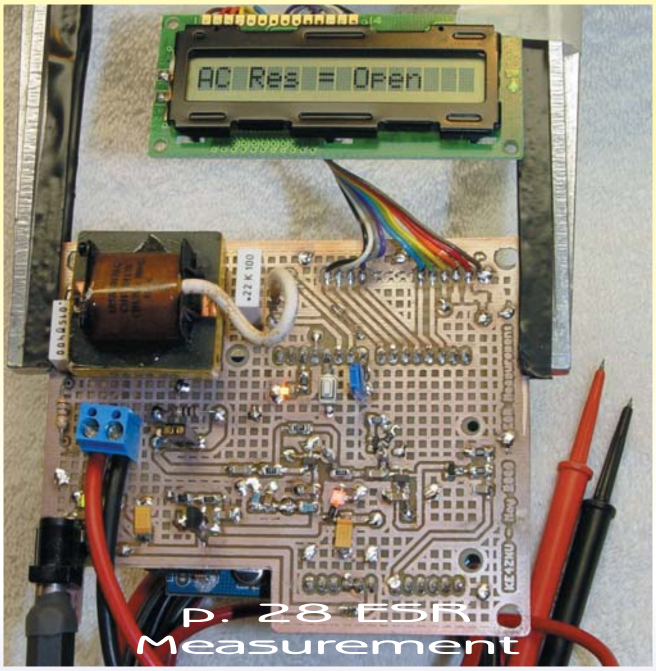
**BONUS CONTENT**

Buddy Memory Manager

Arduino-based Temperature Display

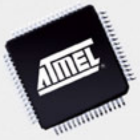






## Hammer Down Your Power Consumption with picoPower™!

## THE Performance Choice of Lowest-Power Microcontrollers



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

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

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

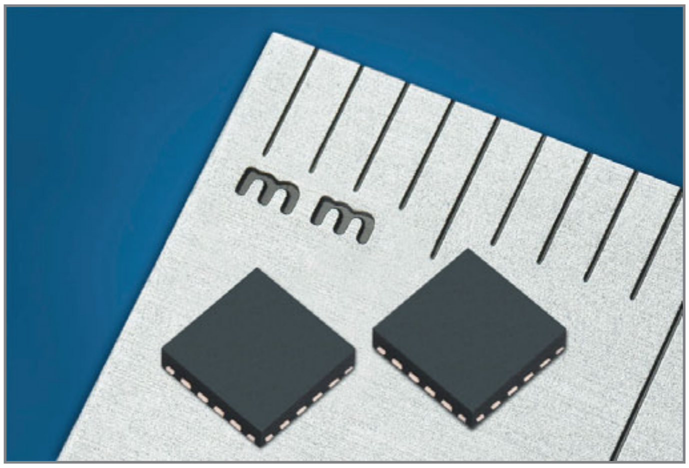
http://www.atmel.com/picopower/

ATMEL®
Everywhere You Are®

© 2008 Atmel Corporation. All rights reserved. Atmel®, logo and Everywhere You Are® are registered trademarks of Atmel Corporation or its subsidiaries.
Other terms and product names may be trademarks of others.

## FRONT END IC SIMPLIFIES BOARD DESIGN AND REDUCES COST

The **UPG2253T6S** Front End Integrated Circuit (FEIC) is a highly integrated device that combines a power amplifier, low-pass filter, and two single-pole, double-throw switches into a single chip. By integrating these components into one die, the FEIC reduces the component count, saves the total bill of materials cost, simplifies procurement, and decreases the PCB space required for many new embedded product designs. The FEIC eliminates the need for RF component matching, which greatly simplifies board design and further reduces component count.



The FEIC is superior to similar products on several fronts. Its smaller package size enables product designs with tiny footprints, while its better harmonic suppression eliminates the need for low-pass filters on most product designs. Few designs have the FEICs through/PA bypass feature, which enables the end product to switch to a high-power mode when greater range is needed, but automatically switches to a low-power mode when greater battery savings are optimal. Alternatively, the through/bypass path can be used as the receive path. Furthermore, the ability to integrate these functions on a single chip gives size reduction and performance advantages over companies using multi-chip modules.

The UPG2253T6S FEIC is shipping now. It costs **$1.10** in quantities of 100,000. Evaluation boards are available.

**California Eastern Laboratories**
**www.cel.com**

## EMBEDDED MODEM DEVICE SERVER

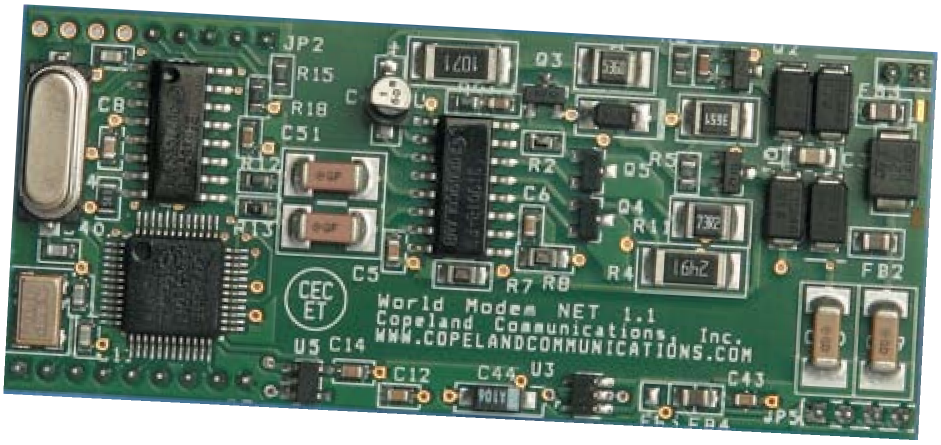
The **World Modem NET** is a compact 2400 to 56 K baud modem with embedded Internet link controller. The unit features a transformerless DAA meeting global telephone system requirements, no electromechanical components, and a flexible DSP data pump. The modem is user-configurable to meet virtually all global telecom requirements and conforms to the industry-standard World Modem mechanical and interface specifications. Because of its built-in features and flexibility, many applications and OEM products can be created using the device.

In addition, the modem mediates an Internet link via its embedded controller. This provides an extended AT command set for a simple, low-overhead interface to the Internet. The World Modem NET also provides interfaces to SPI, I$^2$C, and general-purpose I/O signals. The base version includes support for IP, TCP, UDP, DNS, HTTP client, and PING. SMTP, FTP, and other features are available as options. The embedded controller automatically configures the modem interface and is controlled by an extension to the AT command set.

The World Modem NET comes with a transferable FCC Part 68 license, CE, and other certifications. The World Modem NET requires only 65-mA maximum at either 3.3 or 5 V, and provides 5 kV of isolation.

The World Modem NET prices start under **$43** in 1,000-piece quantities. RS-232 and USB models are also available upon request. The World Modem NET is available through Saelig Company or Copeland Communications.

**Copeland Communications, Inc.**
**www.copelandcommunications.com**



NEW PRODUCT NEWS

Edited by John Gorsky

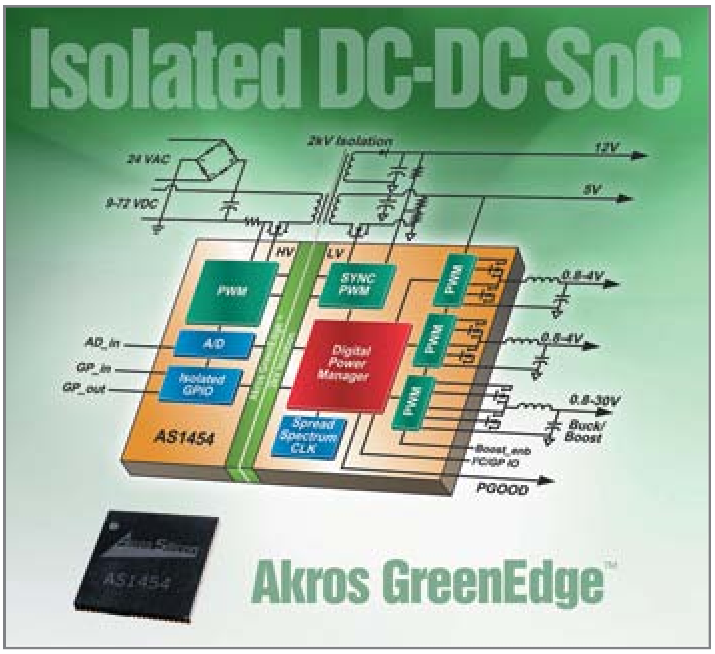
## TWO kV-ISOLATED, QUAD-OUTPUT POWER SoC

The **AS1454** is an isolated quad-output digital power SoC that sets a new integration benchmark for isolated DC-DCs in a wide range of industrial and other distributed power applications. The AS1454 eliminates the need for low-speed, bulky optocouplers and integrates the functionality of up to eight separate ICs used in current implementations. This leads to a substantial reduction in design footprint and bill-of-material costs. The AS1454's built-in cross-isolation DC-DC timing management and digital power control delivers 92% DC-DC efficiency with excellent light-load efficiency management for energy-efficient, green-power applications.

The AS1454 integrates a wide input range (9- to 72-V) isolated primary converter, a 2-kV isolation barrier, a high-current-capable buck or boost PWM controller, and two 2-A buck regulators into a single device. Its wide-input voltage range allows it to be used in 12 VDC/24 VDC/24 VAC distributed power applications, as well as 48-V distributed power applications and equipment requiring a 36- to 72-V input voltage range. It offers selectable spread-spectrum clocking on all PWM to reduce power-supply spectral noise by more than 15 dB to lower the EMI signature of the switch-mode power supplies and ease system design for EMC compliance.

Device pricing is available upon request.

**Akros Silicon, Inc.**
**www.AkrosSilicon.com**



PCBCORE

**China PCB Supplier**
**(Prototype thru Production)**

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

..... Day and Night

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

86(571)86795686 sales@pcbcore.com

**www.pcbcore.com**

**RF Specialists**

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

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

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

**Data Loggers**
Stand Alone and
Wireless Mesh Networking Logger

**GPS**
OEM Modules and USB ZigBee Sticks,
Mesh Networks

**ZigBee Pro**
OEM Modules and USB ZigBee Sticks,
Mesh Networks

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

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

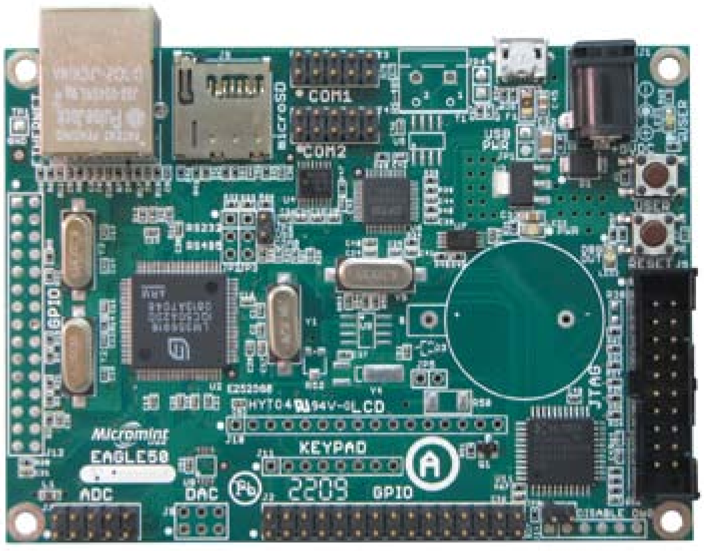
## COMPACT HIGH-PERFORMANCE SBC

The **Eagle 50** and **50E** are single-board computers (SBCs) designed for cost-sensitive control applications that require real-time performance, networking, and extensive support of popular peripherals. Available in the compact 100 mm × 72 mm picoITX form factor, they deliver 32-bit performance and features at a cost equivalent to legacy 8- and 16-bit controllers. Powered by a TI ARM Cortex-M3 core capable of 60 MIPS, these SBCs can fulfill demanding requirements in monitoring, instrumentation, data acquisition, process control, factory automation, and many other applications. Configurations include up to 256-KB flash memory, 64-KB SRAM, 40 digital I/Os, eight analog inputs, four analog outputs, RS-232/485, one microSD card, and a battery-backed RTC. The Eagle 50E also supports 100-Mbps Ethernet, allowing remote access via web or command line interfaces for off-site monitoring. An optional USB port can be used for programming or data transfers.

IAR and GNU programming tools allow development of fast and efficient applications in C. An extensive collection of code examples is included to get started quickly. Ports of popular open-source Basic and LUA development tools are also available to reduce application development time. Demos of FreeRTOS and NuttX are included for applications requiring an RTOS. The Eagle SBCs provide system designers and integrators with high functionality, a compact footprint, and a low cost.

In single quantities, the Eagle 50 starts at **$44.95** and the Eagle 50E at **$49.95**. Custom OEM versions can be designed for specific requirements.

**Micromint USA**
**www.micromint.com**



# COMPACT EMBEDDED SERVER

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

2.6 KERNEL

Windows CE

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

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

www.emacinc.com/servers/compact_sib.htm

Since 1985 OVER 24 YEARS OF SINGLE BOARD SOLUTIONS

EMAC, inc.
EQUIPMENT MONITOR AND CONTROL

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

# WIRELESS MADE SIMPLE®
BRING YOUR PRODUCT QUICKLY AND LEGALLY TO MARKET

**RF Modules** *Add INSTANT wireless analog / digital capability to your product.*

**OEM Products** *FCC PRE-CERTIFIED & ready to customize for your application.*

**Feature Products** *A closer look at Linx innovation*

**LOW-COST • LONG-RANGE TRANSCEIVER**
- Direct serial interface
- Low power consumption
- PLL-synthesized architecture
- RSSI and power-down functions
- Compact surface-mount package
- No external RF components (except antenna)

**REMOTE CONTROL TRANSCODER IC**
- Up to 8 inputs
- Bi-directional control
- Transmitter ID output
- Automatic confirmation
- Secure $2^{24}$ possible addresses
- Latched and/or momentary outputs

LINX® TECHNOLOGIES WIRELESS MADE SIMPLE
800-736-6677
159 Ort Lane · Merlin, OR 97532
www.linxtechnologies.com

# The Next Generation of In-Circuit Debugging

## The NEW MPLAB® ICD 3

The MPLAB ICD 3 In-Circuit Debugger is Microchip's most cost effective high-speed debugger for Microchip Flash PIC® Microcontrollers (MCU) and dsPIC® Digital Signal Controller devices. It debugs and programs PIC MCUs and dsPIC DSCs with the powerful, yet easy-to-use graphical user interface of MPLAB Integrated Development Environment (IDE).

- In-Circuit Debugging for PIC MCUs and dsPIC DSCs
- Full-speed, real-time emulation
- Source debugging, stopwatch, complex breakpoints and in-circuit programming
- MPLAB IDE compatible
- Firmware upgrade via MPLAB IDE
- Overvoltage and undervoltage protection
- High Speed USB 2.0 (480 Mbps)
- Target power, up to 100 MA
- Internal 1 MB memory buffer for increased download speed

## MPLAB® ICD 2 RECYCLE

Return your old MPLAB ICD 2 and receive 25% off the new MPLAB ICD 3, MPLAB REAL ICE or PICkit™ 3 Debug Express. For more information on this offer, please visit:

www.microchip.com/ICD2recycle

www.microchip.com/ICD3

MICROCHIP

The Microchip name and logo, the Microchip logo, MPLAB and PIC are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
PICkit is a trademark of Microchip Technology Incorporated in the U.S.A. and other countries. © 2009, Microchip Technology Incorporated. All Rights Reserved.

by Naubert Aparicio

# IR Signal Control

Do you want offsite control of your electronic equipment? With this handy IR signal controller design, you can remotely operate the system of your choice right from your cell phone. In this project, the inno vative design is used to control an AC system.

I live in Puerto Rico, which is a hot place, as you know . The temperature averages 82.4°F (30°C) throughout the year, but it can reach temperatures above 100°F. As a result, some of us are used to ar tificial climates—that is, air-conditioned environments—which cost money to maintain. Like many tropical islands, much of Puerto Rico's electricity is expensive because it comes from oil-based power plants. Thus, electricity conservation is the best way to keep my monthly utility bill down. To do so, I power-up my AC units only when someone is home. But when I get home at 6 PM after a long day of working and sitting in traffic jams, I don't like it when walking through my door feels like entering an oven.

I used to use timers to control my AC system, but I eventually stopped because I wasn't arriving at home the same time ever y day. It was then that I decided to design a way to control my AC system remotely. I wanted to be able to turn on my AC early enough to cool my house before arrival. Figuring that I'd typically use such a design from my car, I decided to use my cell phone as the remote control device.

After a few days of planning, I came up with my ACcontrol design (see Photo 1). It is a small stand-alone, Ethernet-enabled device that uses an infrared (IR) signal for controlling my AC system remotely. By connecting the device to my fixed Internet connection, I can continuously monitor a POP e-mail account for a specific e-mail subject. When it is found, the system sends an IR signal that enables AC control. I can use virtually any

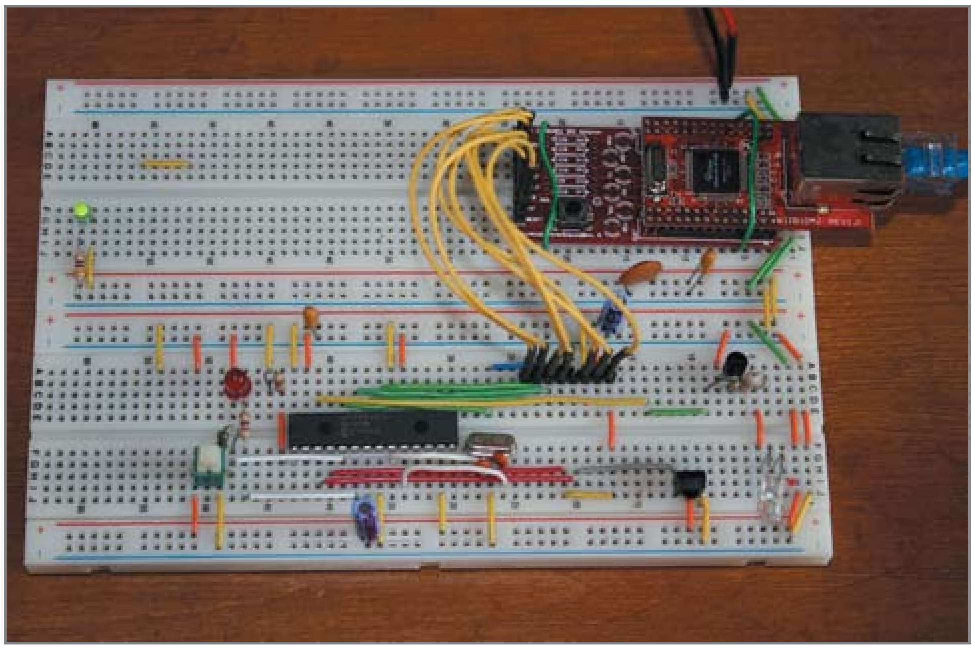


Photo 1—The ACcontrol features a WIZ810MJ module (top right) and a Microchip T echnology dsPIC30F4012 (center). A red LED mode indicator and a push button form a simple user interface. The IR sensor and the IR transmitter are on the lower right.

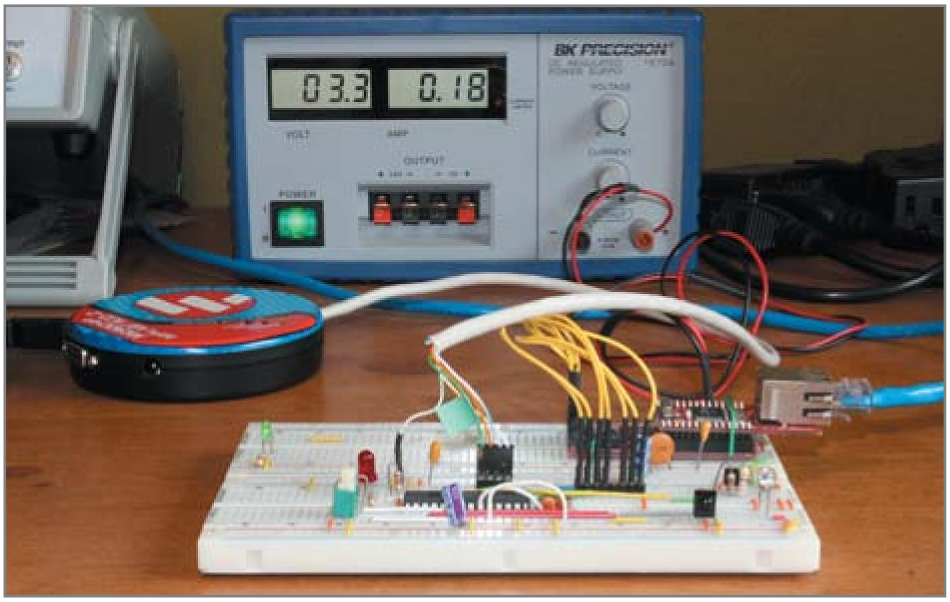


Photo 2—This is my development setup. I used a Microchip MPLAB ICE 2 to help with the debugging process. Being able to display the internal variables (e.g., the retrieved IP packets) through hardware breakpoints was important for accelerating the debugging process.

cell phone to send an e-mail (even with only SMS through an SMS-to-e-mail gateway). Upon receipt, the system sends a confirmation e-mail so I know the command was received.

In this article, I'll describe how I designed the control system around a WIZnet WIZ810MJ module and a Microchip Technology dsPIC30F4012 microcontroller. The W5100 chip embedded in the WIZ810MJ module is a high-level hardwired TCP/IP implementation that ultimately makes projects like this simple, fast, and affordable. I programmed the project in C language with short assembler routines using Microchip Technology's MPLAB IDE version 7.52 and the C30 compiler. I ported the W5100 driver to the Microchip architecture and adapted the WIZnet DNS routines to use the SPI. For simplicity, I extracted several utility routines from WIZnet application notes and adapted them to my code. I used a Microchip MPLAB ICE 2 programmer/debugger and an external power supply as my development setup (see Photo 2).

## USER INTERFACE

I purposely tried to make everything associated with this project as simple as possible. For instance, I didn't want to complicate the project with difficult or costly interfaces. Thus, once the system is configured, it requires no further user interaction.

I designed a simple set-up procedure and incorporated only one button and an LED indicator. The design has the ability to learn an infrared command, set the network configuration (IP, netmask, gateway, DNS IP), configure the POP account, set an access password, erase a forgotten password, and configure the specific e-mail subject to trigger the infrared signal. The network parameters are set from a local networked computer through a dedicated set-up program. Then, through a telnet interface, the rest of the configuration can be adjusted. You can develop a more elaborate graphical interface set-up program in the future with this same design.

## SYSTEM SET-UP

Setting up the ACcontrol system is simple (see Figure 1). First, the device is connected to the network and powered on. Pressing the button for 5 s activates the IP set-up mode. In this mode, the system waits for a network-based configuration packet that's sent by the set-up program in a nearby networked PC. I developed a Linux command line program for this project, but it could be ported easily to any operating system that supports TCP. The set-up program sends a special UDP broadcast packet that contains the IP number, netmask, gateway address, and DNS to be set in the device. When the ACcontrol is in IP set-up mode and sees a configuration packet that matches its MAC address, it sets its network parameters from the packet data. After this, you can telnet to the assigned IP and set the rest of the parameters. All the parameters are saved into the Microchip EEPROM to preserve its values. You can change the telnet password

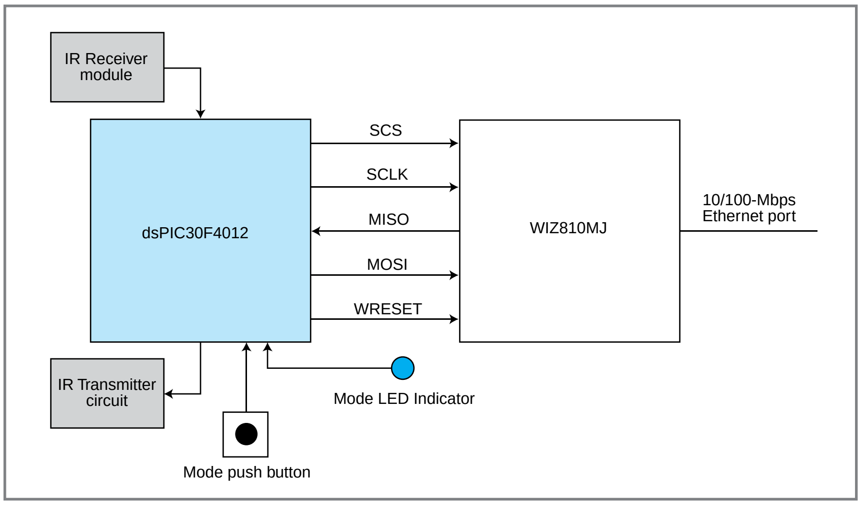


Figure 1—The WIZ810MJ module simplified this project. The only other major component is a dsPIC30F4012 that handles the higher-level TCP routines and IR logic. The WIZnet W5100 hardwired TCP/IP integrated circuit handles the lower-end TCP/IP stack. This resulted in cleaner code and reduced microcontroller hardware requirements.

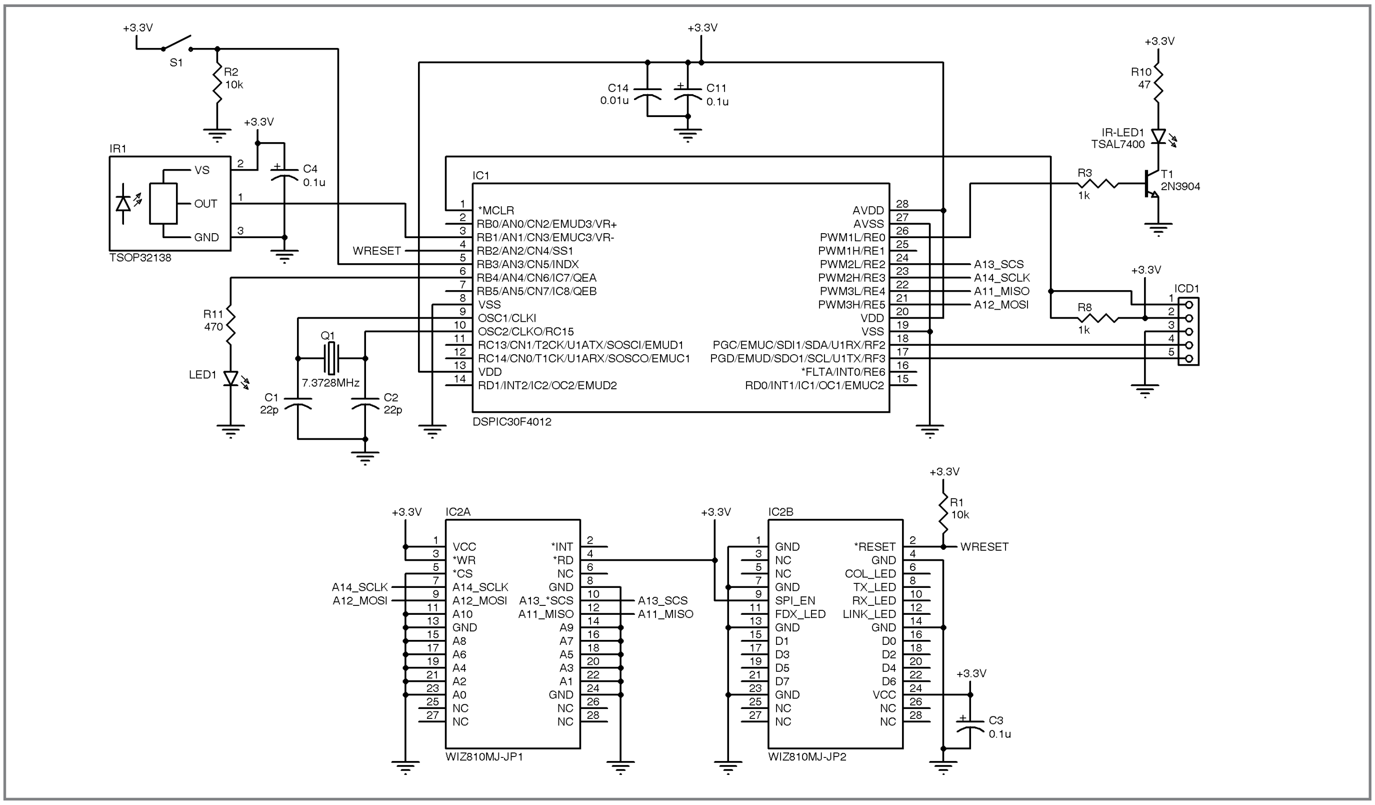


Figure 2—I used a hardwired TCP/IP hardware module and a SPI. It seems as though the software in the dsPIC30F4012 handles everything. But in reality, most of the TCP/IP complexity is hidden from the programmer in the W5100 IC located inside the WIZ810MJ module. A single 3.3-V regulated power source rated at 350 mA is enough to power this project.

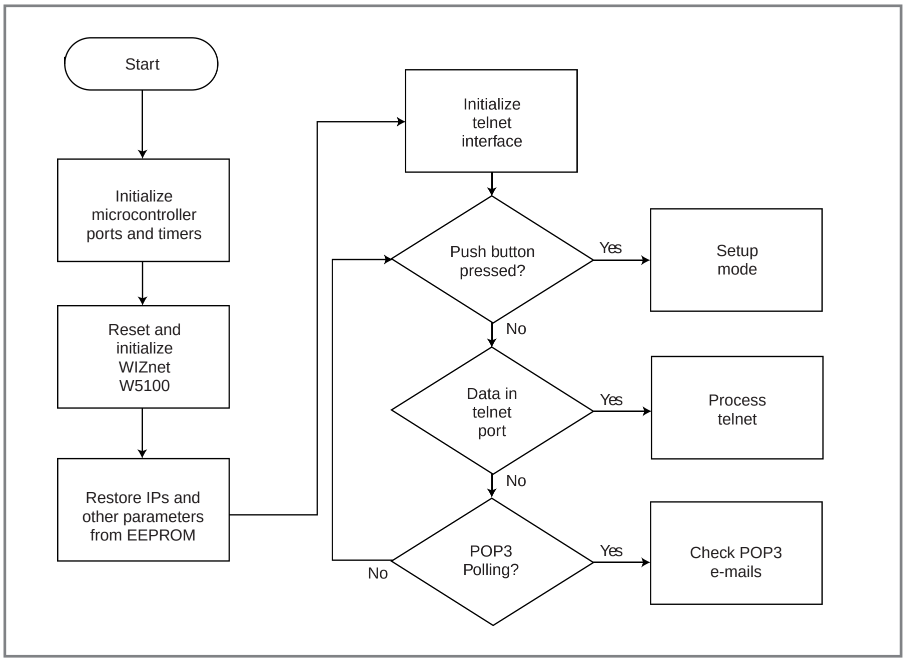
from the same telnet session. If you forget it, you can reset it by pressing the button for more than 5 s.

The IR command setup to power-up the AC is also simple. Pressing the Mode button once puts the ACcontrol in IR learning mode. The infrared command is then "learned" by pointing the original AC remote control to the IR-receiving sensor and pushing the Power button on the remote. The LED indicator remains on while learning mode is activated.

## HARDWARE DESIGN

The WIZnet module simplified the design. In addition to the WIZ810MJ module and the dsPIC30F4012 microcontroller, I used a Vishay Intertechnology TSOP321 38-kHz IR receiver module and a TSAL7400 high-output IR emitting diode (see Figure 2). The WIZnet module comprises everything needed for the Ethernet interface, including the RJ-45 socket, the support circuit, and a SPI. I chose the dsPIC30F4012 for its RAM, EEPROM, flash memory, and timer resources. I needed it to implement the IR learning and sending routines, as well as to implement the code needed for the DNS, POP3, and SMTP software components.

The dsPIC software handles the higher-level TCP protocols, all the communication with the W5100, the user interface, and the IR learn and



Start
Initialize microcontroller ports and timers
Reset and initialize WIZnet W5100
Restore IPs and other parameters from EEPROM
Initialize telnet interface
Push button pressed?
Yes
Setup mode
No
Data in telnet port
Yes
Process telnet
No
POP3 Polling?
Yes
Check POP3 e-mails
No

Figure 3—The segmented software makes it easy to develop and understand. The initialization process is straightforward. The rest of the code is an event loop, where an action is taken after a specific event is detected.

send functions. It controls the WIZ810MJ module through a four-pin SPI and a fifth pin for a RESET signal. I manually coded the SPI routines because the hardware SPI module shared the pins with the PIC debugging interface. The 38-kHz infrared module handles the IR code reception and sends a filtered signal to the dsPIC30F4012's CN3 input configured to generate an interrupt when the logic state changes. Finally, the dsPIC30F4012 is also connected through the RE0 pin to an IR LED through a 2N3904 amplifier. The RE0 is driven by the dsPIC PWM module set to generate the required IR frequency of 38 kHz. The only other two connections are for the push button and red LED as the user interface.

## SIMPLE SOFTWARE?

The hardware is simple, but is the software complex? No. If you're familiar with the TCP protocols (i.e., the POP3, DNS and SMTP protocols), the only complexity that remains is related to the W5100 interface and IR-handling routines. Let's review each software segment and see how they work, so that you can form your own conclusions about this project's complexity. I encourage you to download the source code from the *Circuit Cellar* FTP site so you can follow the subsequent explanations.

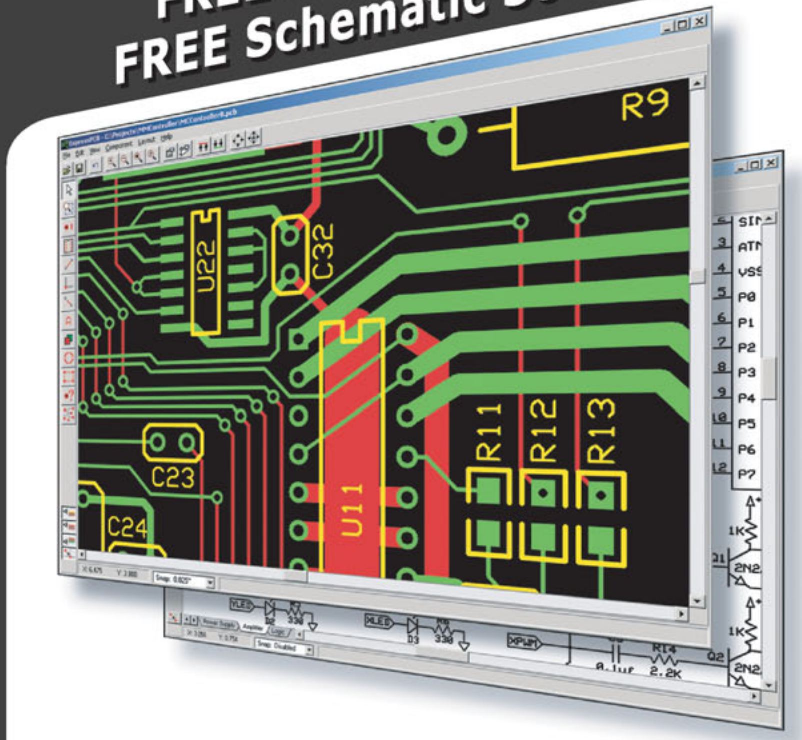
Figure 3 depicts the software flow. You can see the standard hardware initialization that covers the dsPIC resources and the W5100 chip. It is followed by the parameter initialization in the EEPROMs and the initialization of the telnet interface. The software then enters in a loop to look for the push button or data in the telnet port or data in the POP3 account. Each of these functions is handled in a separate code segment. This flow is shown as the `main()` routine (see Listing 1).

The first four routines initialize the system. The `init()` routine initializes the dsPIC30F4012 by configuring the I/O pins, programming the PWM module to the output IR frequency, sending and holding the RESET signal to the WIZnet module, and programming the systick timer used for the delay routines. The next initialization routine is for the IR system, `IR_init()`. This initializes the IR input port interrupt and the Timer3 module (TMR3) to count every 8.7-µs resolution used to measure the IR pulses. When I cover the IR routines later in this article, you'll see that these can easily cope with most of the IR protocols.

After setting up the IR hardware, the next thing to do is to prepare the W5100 hardware TCP/IP chip. This is done with the `reset_w5100` and `sysinit` calls. The former just resets the W5100. It is held in reset by the `init()` call, so in this second call it just frees it after the required hardware reset delay of more than 10 ms. A software reset command is then sent to the chip itself. Then comes the `sysinit` call, part of the W5100 driver, which configures the transmit and receive buffer sizes. The W5100 has four channels, each for

$51 For 3 PCBs

FREE Layout Software!
FREE Schematic Software!



01 DOWNLOAD our free CAD software
02 DESIGN your two or four layer PC board
03 SEND us your design with just a click
04 RECEIVE top quality boards in just days

expresspcb.com

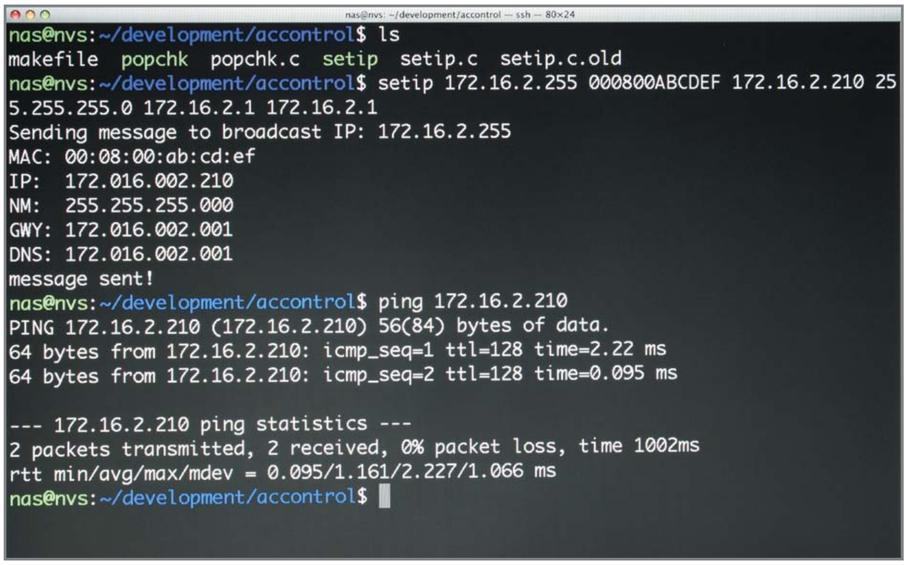
different TCP/IP connections. Also, the W5100 has 8 KB of internal memory. The amount of memory you assign to each channel depends on the simultaneously open sockets or connections in your software, which will determine the volume of traffic the system can handle. More memory assigned to a channel means that the software may be slower when attending the connection as the buffer is bigger and data will not be lost.

I used three channels, so I configured the W5100 in the only possible configuration: 1 KB for each transmit and receive buffer for four channels (8 KB in total). Thus, packets can't exceed the 1 KB size, which is not a problem for my configuration. In my software, channel 0 is assigned as an administration RAW port for IP address configuration, channel 1 is shared for the DNS and e-mail protocols, and channel 2 is for the telnet TCP interface. Channel 3 is reserved as a UDP socket for broadcasting the message to other receiving modules for future expansion.

Once the hardware is ready, the main code restores the IP parameters from the EEPROM, sets the fixed hardware MAC address, and initializes the IP values in the W5100 chip. It is important to note that the W5100 chips don't come with MAC addresses. You can purchase them separately at www.ieee.org, as noted on WIZnet's FAQ page. I created one for this demo, but please note that you need to obtain legal MAC addresses to be able to commercialize products with these chips. For the demonstration's sake, I just had to be sure that the selected MAC address didn't match any other MAC address in my lab.

**Listing 1**—After initialization, a simple loop is used to check for three dif ferent events: a push button press, data in the telnet port, or a POP3 e-mail trigger .

```
// ----- main program -----------------------------------------/
int main()
{
        // initialize microcontroller
        init();

        // initialize IR system
        IR_init();

        // reset w5100
        reset_w5100();

        // initialize w5100
        sysinit(0x00, 0x00);

        // initialize MAC/IP layer with default values
        restore_parameters();    // restore parameters from EEPROM
        setSHAR(my_mac);         // set MAC address
        init_ip();               // init IP values

        // initialize telnet port
        init_telnet();

        // process cycle
        systick = 0;
        for (;;) {
                // check if setup mode is requested
                if (PUSH_BUTTON == 1)
                    setup_mode();

                // check if telnet session
                if (getSn_SR(SOCKET_TELNET) == SOCK_ESTABLISHED)
                    process_telnet();

                if (email_hostname[0] && systick >= email_freq)
                    email_process();
        }

        return 0;
}
```



```
nas@nvs:~/development/accontrol$ ls
makefile  popchk  popchk.c  setip  setip.c  setip.c.old
nas@nvs:~/development/accontrol$ setip 172.16.2.255 000800ABCDEF 172.16.2.210 25
5.255.255.0 172.16.2.1 172.16.2.1
Sending message to broadcast IP: 172.16.2.255
MAC: 00:08:00:ab:cd:ef
IP:  172.016.002.210
NM:  255.255.255.000
GWY: 172.016.002.001
DNS: 172.016.002.001
message sent!
nas@nvs:~/development/accontrol$ ping 172.16.2.210
PING 172.16.2.210 (172.16.2.210) 56(84) bytes of data.
64 bytes from 172.16.2.210: icmp_seq=1 ttl=128 time=2.22 ms
64 bytes from 172.16.2.210: icmp_seq=2 ttl=128 time=0.095 ms

--- 172.16.2.210 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.095/1.161/2.227/1.066 ms
nas@nvs:~/development/accontrol$
```

**Photo 3**—The sample Linux set-up utility receives the following: the broadcast address of the PC and the MAC address of the device, the desired IP, the netmask, the gateway, and the DNS addresses. A UDP packet is broadcast addressed to port 9000, and it's captured by the ACcontrol's Raw mode, even if its IP doesn't belong to the current network.

The W5100's IP setup procedure is very simple, as you can see in the `init_ip` routine. I first set the IP address by calling the W5100 driver `setSIPR` routine. The netmask is set with the `setSUBR` call and the gateway address configuration requires a call to the `setGAR` routine. Each IP address and netmask is represented as a 4-byte array. The DNS address is saved in a local variable in the format required by the DNS search routines.

Finally, the initialization segment is completed by a call to the Telnet initialization routine, `init_telnet()`. It opens a TCP socket in the W5100's preassigned SOCKET_TELNET channel and sets it to listen mode. Please note that these calls are non-blocking. They just set the socket to listen. The actual packet "listening" is performed in the `process_telnet` segment code where the socket is tested for data. If you are familiar with the socket interface, you will realize that the WIZnet driver interface is similar to other operating systems like Unix. Some differences exist. In the current implementation, the programmer has to first check that there is data in a socket before calling the socket read routine (i.e., there is no select call and the read routines are non-blocking). This is needed because you don't want to stop the CPU waiting for a packet to arrive. You check for data, and if there is no data just continue to check other things. This technique is used a lot in embedded processing and avoids the need for a multitasking RTOS.

## PROCESSING LOOP & SETUP

Refer to the main routine (see Listing 1). After initialization, the code is a simple loop that checks for a button press, a new connection to the telnet socket, or a received trigger e-mail. Each event produces a call to a corresponding routine. A button press event puts the device in set-up mode via the `setup_mode` routine. A new telnet connection executes the `process_telnet` code, which implements the telnet interface. Finally, if a new e-mail is detected, the `email_process` routine activates the IR signal and sends you a reply e-mail.

Pressing the Set-Up mode push button activates the set-up routine. This makes the ACcontrol listen to a special UDP packet sent from a PC program that's designed to send the device IP configuration. In Set-Up mode, you execute a simple program to request the IP address, netmask, default gateway, and DNS IP and then send it to the network. But how can the ACcontrol, with no defined IP configuration, receive an IP packet?

Other devices on the market generally use DHCP or a fixed IP address for the initial contact. DHCP was undesirable for this case because it would be difficult for you or the set-up program to know the IP address required for the rest of the configuration. A fixed IP would force you to change the PC's IP to be able to communicate with the newly reset device. Thus, I decided to use another

**60 MHz - 1 GHz**
**Starting from $1,000**

Experience the New LeCroy Oscilloscopes*

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

LeCroy

*To learn more about LeCroy's full line of oscilloscopes, including the 30 GHz WaveMaster 8 Zi, the world's fastest real-time oscilloscope, visit www.lecroy.com or call 1-800-5-LeCroy approach and experiment with the W5100 chip's Raw Socket mode.

Set-Up mode opens a socket in Raw mode. If you are unfamiliar with Raw Socket mode, think of it as a "promiscuous" (as it is called in some circles) Ethernet interface that captures every packet, even if it isn't directed to the device. In this mode, I checked for a specific UDP packet sent from the PC with the required IP configuration. When the ACcontrol sees it, it takes the information from the UDP packet body and sets itself accordingly. The previous IP configuration doesn't matter because this method always works (see Photo 3). The `setup_mode` routine implements this as well as the IR learning and password reset functions.

For the IP configuration, there is a `do-while` loop in the code that checks for a packet arrival. It checks that it is an IP packet (0800 hex at the Ethernet-type field). Then it checks if it is a UDP packet directed to port 9000 and if the packet is a special packet sensed by a magic number in its first data position. If everything is fine, the configuration is extracted from the packet, applied to the system, and the Set-Up mode is exited.

Listing 2—The `IR_send` routine just replays the sampled transitions from the IR learning routine. The `ir_array` contains the elapsed time between changes of the learned signal. If a signal must be modulated at 38 kHz for the duration of the transition, the most significant bit of the word is set to one.

```
// send the learned IR command
void IR_send(void)
{
        uint8 i, logic;
        uint16 t;

        for (i = 0; i < ir_transitions; ++i) {
                t = ir_times[i];                        // get time
                logic = (t&0x8000)?1:0;                 // get logic
                IRpulse(logic, (t&0x7FFF));             // send pulse
of the corresponding logic
        }

        // wait lead time
        IRpulse(0, 2298);
}
```

## INFRARED COMMANDS

Most of the IR commands for remote control consist of a pulse-modulated 30- to 60-kHz signal. (Note that 38 kHz is the most common.) Although different protocols exist for embedding a command in the signal, what I wanted was just to learn a single command—the power On/Off signal—and then be able to replay it when needed. The system

THE ORIGINAL SINCE 1994
PCB-POOL®
Beta LAYOUT
Servicing your complete PCB prototype needs :
• Low Cost - High Quality
PCB Prototypes
• Easy online Ordering
• Full DRC included
• Lead-times from 24 hrs
NEW !
• Optional Chemical Tin finish
no extra cost
NEW !
Watch "ur" PCB®
Follow the production of your PCB in
REALTIME
FREE LASER STENCIL WITH ALL PROTOTYPE PCB ORDERS
email : sales@pcb-pool.com
Toll Free USA : 1 877 390 8541
www.pcb-pool.com
Beta LAYOUT

2B2C
Factory on Your Fingertip
Factory on Your Fingertip
From Idea to Reality
-> Design
-> Prototyping
-> Production
See Our Differences in
-> Quality
-> Service
-> Price
*Milling *Turning *Grinding *CNC
*Wire Cutting *Laser *Plasma
*Water Jet *Plastic Injection
*Sheet Metal *Gear *SLA
*FDM *SLS *LOM
CLICK TO MAKE
MachinePIER
Tel: 408-421-9840 Email: sales@machinepier.com
Website: http://www.machinepier.com

Stellaris®

**Connected. Versatile. Cost-effective.**

**Stellaris LM3S9B90 Ethernet Evaluation Kit**

featuring USB, Ethernet and CAN

**$99**

**Stellaris LM3S8962 Evaluation Kit**

featuring integrated Ethernet and CAN

**$89**

**Stellaris LM3S3748 Evaluation Kit**

featuring USB Host/Device

**$109**

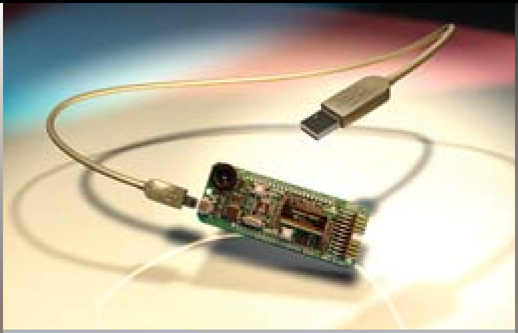
**Stellaris® Means:**

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

**Stellaris LM3S6965 Ethernet Evaluation Kit**

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

**$69**



**Stellaris LM3S811 Evaluation Kit**

featuring basic embedded control

**$49**

**Stellaris LM3S2965 CAN Evaluation Kit**

demonstrating a CAN network out-of-the-box

**$79**

ARM POWERED

Cortex Intelligent Processors by ARM

Texas Instruments

doesn't need to understand the data in the signal (i.e., decoding isn't needed), so it's simple to sample the signal and record the time lapses between signal changes. This will accommodate any protocol.

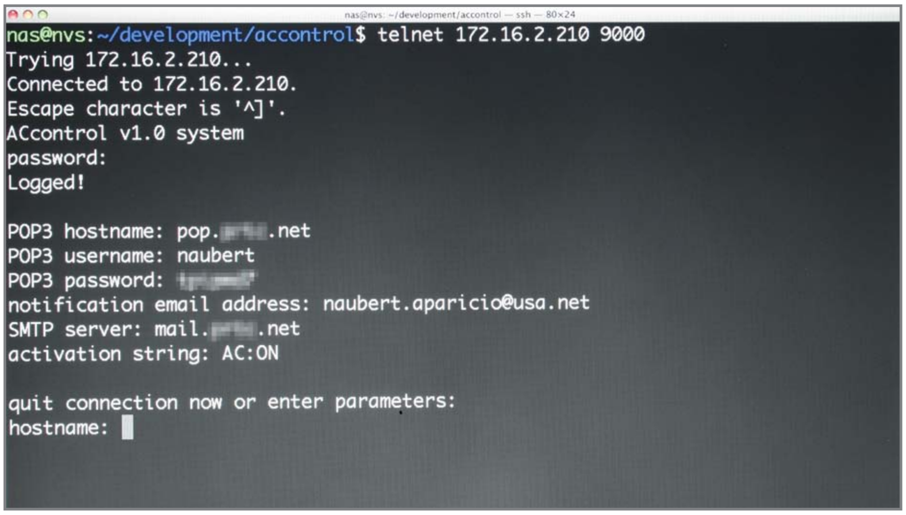
Replaying the "learned" IR signal is also easy. Just perform the modulation at the previously recorded time. As I already mentioned, I used a 38-kHz IR module that filters the signal and converts it to a logic 1, where a 0 is when modulation is present and a 1 otherwise. This signal is connected to the dsPIC30F4012's CN3 pin.

IR routines are included at the end of the ACcontrol.c source code file available on the *Circuit Cellar* FTP site. The `IR_capture` routine starts the interrupt at CN3 and resets the internal counters and variables that record the signal. The time between interrupts are saved into the `ir_times` variable. `IR_stop` stops the recording. Actual sampling and recording occur at the CN3 state change interrupt-handler routine called `_CNInterrupt`. In each logic state, the change of the CN3 input pin is recorded together with the time from the last interrupt. The signal recording stops when the array storage is filled or if no signal is received in 20 ms, as set in the `IR_init` routine. The fifteenth bit in each value in the `ir_times` array records the state at the capture: one if there was modulation, zero if there wasn't.

The `IR_send` routine is much simpler (see Listing 2). An iteration covers all the words in the `ir_times` array, checking the fifteenth bit or the modulation state and sending an IR signal for the recorded time. This is for all the learned transitions, thus replaying the signal exactly as it was sampled. The `IRpulse` function sends the required 38-kHz signal to the dsPIC30F4012's output pin using the previously configured PWM module.

## TELNET INTERFACE

The telnet interface is one of the most complex components of the ACcontrol project. Interactivity complicates things. Processing starts



```
nas@nvs: ~/development/accontrol — ssh — 80×24
nas@nvs:~/development/accontrol$ telnet 172.16.2.210 9000
Trying 172.16.2.210...
Connected to 172.16.2.210.
Escape character is '^]'.
ACcontrol v1.0 system
password:
Logged!

POP3 hostname: pop.[redacted].net
POP3 username: naubert
POP3 password: [redacted]
notification email address: naubert.aparicio@usa.net
SMTP server: mail.[redacted].net
activation string: AC:ON

quit connection now or enter parameters:
hostname: █
```

Photo 4—Once the device IP is set through the PC set-up program, you can connect via telnet to the 9000 port in the device to set the rest of the parameters: the POP3 account, the username, the password, the e-mail address that will receive the activation reply e-mail, the SMTP server where e-mails are to be sent, and the subject that will activate the infrared signal. You can also change the telnet password from the default "password" string.

when a connection is received at port 9000. The main program calls the `process_telnet` routine, which first checks that the connection is established and monitors it in case you or the connection fails during the telnet session (closing the socket in that case). When the connection and password prompt is sent, the telnet command IAC WILL ECHO is also sent, causing the telnet client to turn off its local echo so that the password isn't visible on the screen.

The telnet routine then waits until the `getSn_RX_RSR` function returns notification that data is available at the receiving buffer. Text processing filters the telnet protocol control characters and buffers a complete line before analyzing it. A finite state machine performs the interface actions in order. First, it gets the password and then asks for the POP parameters, enabling a subject string and a new password. Photo 4 shows the interaction. Using a finite state machine and moving from state to state simplified the coding because the software was not implemented in an RTOS. You can interrupt the telnet session at any time. Only the changed parameters are saved in the EEPROM.

## E-MAIL COMMUNICATIONS

The `email_process` routine is the main ACcontrol function. The main code calls it periodically. It just checks if there is an activation e-mail from the POP3 account. If there is, it activates the IR signal and sends a reply e-mail confirming the command. To check the POP3 account, it calls the `check_email` routine. This routine establishes a new TCP connection to the POP3 standard port at the server specified in the telnet configuration session. Once connected, it logs into the POP3 account sending the POP3 commands `USER` and `PASS`. If the log-on is successful, the routine requests the number of e-mails with the `STAT` command and then loops through the e-mails scanning the subjects to check for the activation substring. If a subject contains it, the `scan_email` routine sends the `DELE` command to delete that single mail and it signals the calling program that the IR command must be sent. This is received by the `email_process` routine, which then calls the `IR_send` command, turning on or off the air conditioning equipment. It also calls the `send_email` routine to send you a reply e-mail. This last routine uses SMTP commands to send a reply address to the registered address through the telnet configuration session. Like the `check_email` routine, it establishes a new TCP connection to the SMTP port at the registered mail server, authenticates it by sending the HELO command, and sends the e-mail using the `MAIL FROM`, RCPT TO, and DATA commands via the network connection. This implementation requires that the SMTP accepts an anonymous session.

## DESIGN SUCCESS

This project took just 20 days, as you can see in the log included with the source code posted on *Circuit Cellar* FTP site. The WIZnet module and its W5100 hardwired TCP/IP chip really simplified the programming process. They provided a high-level socket interface with raw socket processing to a lower-end microcontroller.

I now have a working design for controlling my air conditioner with my cell phone. Plus, I can say I have a great networked embedded hardware design project under my belt. Now it's your turn to design a similar project of your own. ◼

*Naubert Aparicio (naubert.aparicio@usa.net) is a computer science engineer with a degree from the Simon Bolivar University in Venezuela. As the technology director for Quantum Business Engineering in Puerto Rico, he manages design-critical Unix system architectures. In his free time, Naubert enjoys researching topics such as robotics, advanced digital design, and embedded systems.*

## PROJECT FILES

To download the code and a list of useful resources, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/231.

## RESOURCES

———, "TSOP321: IR Receive Modules for Remote Control Systems," 82229, 2007.

Microchip Technology, Inc., "dsPIC30F4011/4012 Data Sheet: High-Performance 16-bit Digital Signal Controllers," DS70135D, 2007.

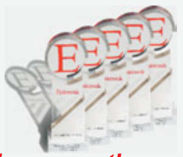
WIZnet, "WIZ810MJ Datasheet," Version 1.0. 2007, www.circuitcellar.com/Wiznet/WIZ810MJ%20Datasheet_V_1.0.pdf.

## SOURCES

**dsPIC30F4012 DSC**
Microchip Technology, Inc. | www.microchip.com

**Ethernet WIZ810MJ module**
WIZnet, Inc. | www.wiznet.co.kr/en/

**TSAL7400 Infrared-emitting diode and TSOP321 modules**
Vishay Intertechnology, Inc. | www.vishay.com



*It takes more than a top program to win the*
***Product of the Year Award****
*five times running*

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

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

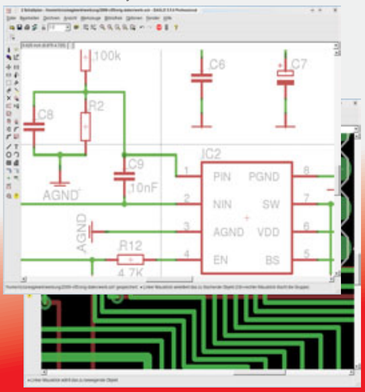
* Annual award of the leading German electronics magazine. The winners of eleven categories are determined by the reader's votes.

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

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

Follow-me Router

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



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

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

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

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

www.cadsoftusa.com
800-858-8355

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

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

# Frequency Sensing Made Simple

## Power Grid Frequency Monitor Design

This power frequency monitor indicates the relative stability of a power grid. Built around a microcontroller, a precision crystal oscillator, an op-amp, and various passive components, the design measures power grid frequency via a standard electrical outlet. It then sends the data to a PC, displays the information, and relays it to a web server for real-time monitoring.

by Arnold Stadlin

An indicator of a power grid's health is its ability and agility to respond to changes in the supply and demand of electricity. Measuring the stability of a power grid's frequency is an easy method for monitoring its health. Although there are other measurable grid characteristics, frequency is ubiquitous across the grid and less affected by local disturbances than other attributes like voltage and current.

"PHzMonitor" is a short name I affectionately use to refer to this project and its frequency sensor (see Photo 1). The PHzMonitor frequency sensor consists of a Microchip Technology dsPIC30F3012 microcontroller, a precision crystal oscillator, an op-amp, and various passive components (see Figure 1). The sensor measures the power grid frequency from a typical electrical outlet scaled down to less than 12 V by an AC/AC transformer (see Photo 2). A PC communicates with the PHzMonitor via its RS-232 serial port. Software on the computer displays the frequency measurements and optionally relays the data to a web server for live public viewing. You can save the data to disk via the PC software or download it from the web server for offline analysis.

## FREQUENCY MEASUREMENT

Let's review how the PHzMonitor measures power grid frequency. There are five steps in the process.

Step one: Use an AC-to-AC transformer to scale the standard electrical outlet voltage down to a safe level for your components. You need the AC voltage to be between 9 to 12 VAC. Step 2: Rectify the AC waveform to a half wave using diodes. Step three: Scale the AC half waveform to a safe level for op-amp input using voltage divider resistors. Step four: Create a square waveform from the half wave using a DC reference voltage and an op-amp in a rail-to-rail configuration. The op-amp outputs the amplified difference between the AC voltage and the DC reference voltage as 0 to 5 VDC. Step five: Use a crystal oscillator-driven microcontroller to measure the periodic cycle

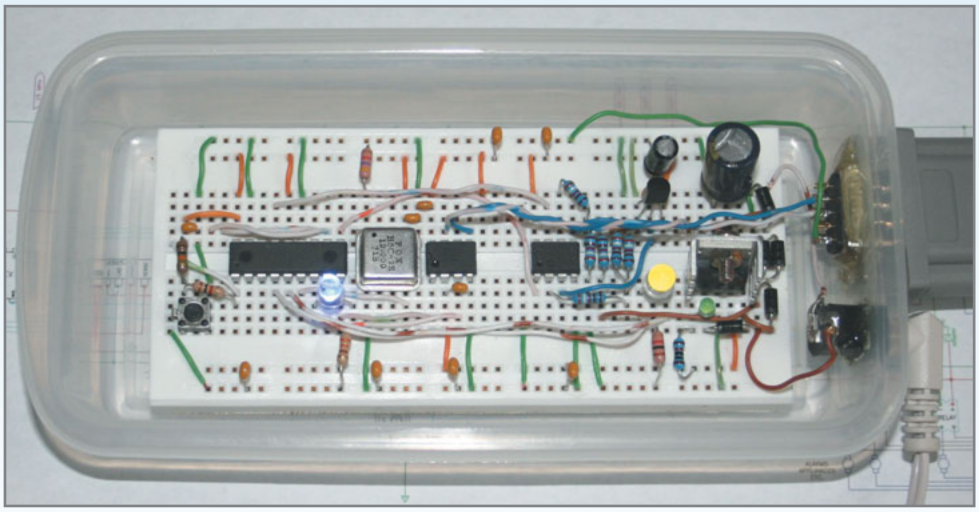


Photo 1—The PHzMonitor project is a useful, scalable system. Networked sensors along with the Internet enable you to monitor frequency conditions in a power grid. You can also compare the performance of different grids.

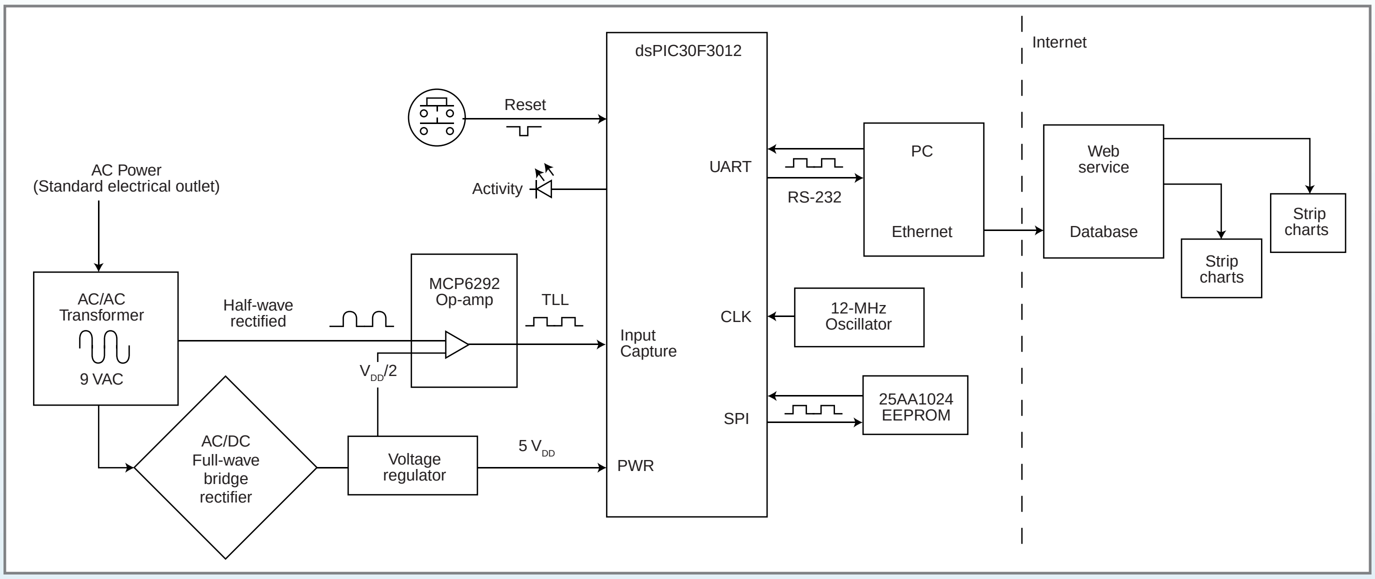


Figure 1—The PHzMonitor project includes a sensor input, microcontroller, computer, and web server components.

timespan by counting the number of oscillator cycles between the leading edges of the square wave.

## DESIGN & CONSTRUCTION

I'll be honest. I jumped into the design and constr uction of the PHzMonitor sensor *before* doing my homework! The sensor design concept is simple: convert alternating current to a square wave and then use a microcontroller to count timer ticks between leading edges of the digital signal. The concept for measuring the frequency of a square wave is similar to measuring the motion velocity of a quadrature encoder signal. The design concept is heavily biased toward this method because I was involved in a robotics project at the time.

What makes the PHzMonitor sensor unique is that it measures frequency at an arbitrary 2.5-VDC reference voltage level instead of measuring at the zero crossings. I used a reference voltage for two fundamental reasons. One, it's easier (for me, at least) to measure a voltage reliably than it is to measure no voltage reliably. Two, the cyclic nature of the power signal being measured enables me to set a voltage reference between logic level voltages and amplify the comparative results with acceptable accuracy.

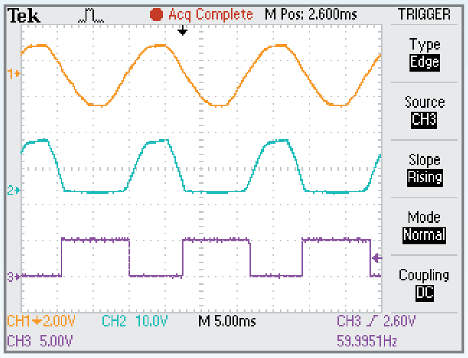


Photo 2—This oscilloscope screenshot displays (from top to bottom) the input AC sine wave, the rectified half wave, and the amplified 5-V square wave prepared for measurement.

The design uses a 9-VAC transformer for power and sensor input. The 5-VDC power supply for the integrated circuit components is tapped from the AC power input using a standard full-wave bridge rectifier, a linear voltage regulator, and an electrolytic capacitor. The half-wave signal samples are simply the resultant, complimentary, half-wave forms from the rectifier's inputs. The current design uses a dual-channel op-amp. If you really want to measure frequency at the zero crossings, the complementary half-wave forms can be used for calculating the zero-crossing point between two reference voltage comparisons. Both half-waves are positive, and both use the same reference voltage, so the zero-crossing time is one-half the time between the leading edges. Did I not mention that it is more difficult for me to measure no voltage?

I used a Microchip Technology MCP6292 op-amp in the sensor. I chose it for two reasons: it is designed to operate in 0- to 5-V rail-to-rail mode, and it has a high speed of 10 MHz. The 16-bit dsPIC30F3012 microcontroller has a built-in input capture peripheral, which is ideal for measuring the timespan of a periodic digital signal. This device has a generous 24 KB of program memory, 2 KB of RAM, and 1 KB of flash EEPROM. The dsPIC30F3012 is available in an 18-pin parallel dual inline packaging as well as surface-mount packaging.

A Fox H5C-2E 12-MHz crystal oscillator drives the microcontroller. The 12-MHz oscillator frequency was selected because it can be evenly divided by both 60 Hz and 50 Hz frequency values. A Maxim Integrated Products DS275 RS-232 transceiver is used for communications with a PC. A National Semiconductor LM340T5 (7805) linear voltage regulator is used to provide the integrated circuit component power. A Microchip Technology MCP1525 provides the 2.5-V reference. Other components include resistors, capacitors, silicon diodes, LEDs, and a push button for reset. The project is assembled on a 2″ × 4″ prototype board packaged in a plastic container.

## PROGRAMMING

The PHzMonitor's dsPIC30F3012 microcontroller is programmed with the Microchip C30 compiler. The programmed firmware configures an input capture timer to accumulate clock ticks in a 16-bit register. A high-priority ISR executes when an INPUT pin detects the leading edge of the square wave. This routine copies the input capture timer's register value to a RAM variable for program loop processing and clears the register. A normal priority ISR receives an RS-232 command byte and copies it to a RAM variable for program loop processing. The main program loop polls the RAM for command bytes recorded by the RS-232 routine. When a command is found, the main loop processes the command's procedure and repeats the loop.

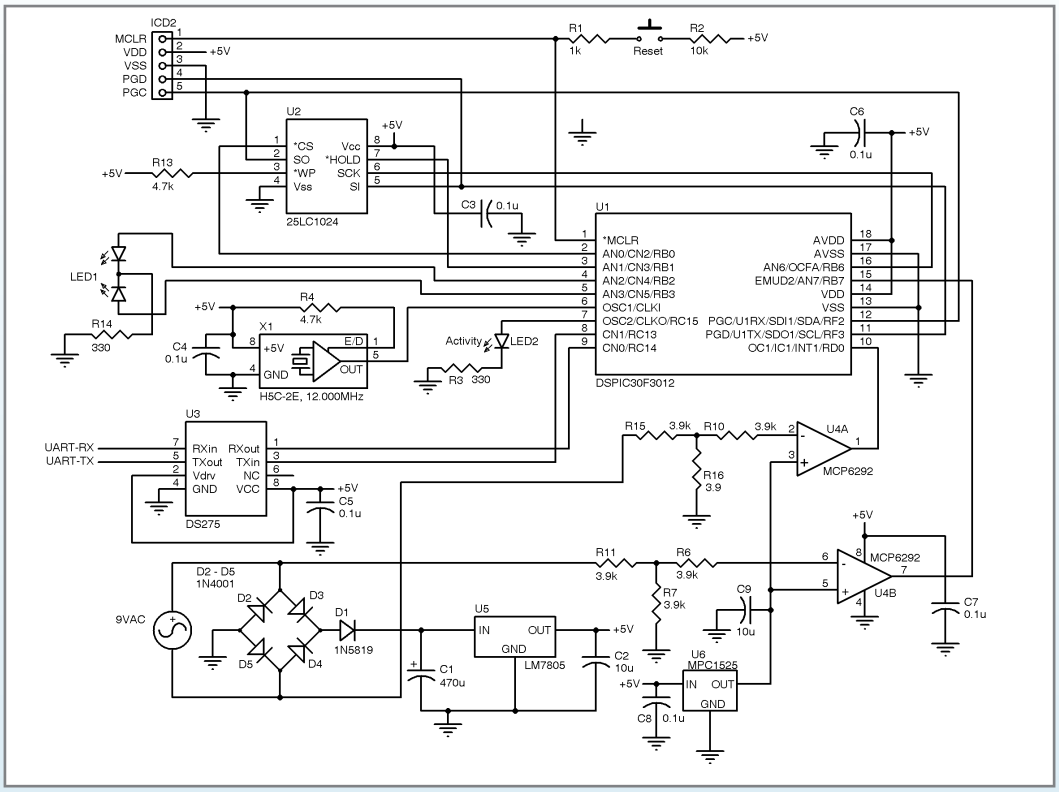


Figure 2—A dsPIC30F3012 sits at the heart of the PHzMonitor design. I used a Fox H5C-2E 12-MHz crystal oscillator to drive the microcontroller.

The PHzMonitor is programmed to respond to single byte command constants. There are only three commands. PING transmits the device's unique identification number and firmware version. TRANSMIT sends a data packet containing a start byte, 16-bit packet identifier, 16-bit last measurement, 16-bit average of the last 10 measurements, and a stop byte. DIAGNOSTIC transmits a continuous bitstream for RS-232 timing measurements.

The computer sends the PING command to verify that it is connected to the device and to get the device's unique identification number. When the computer is ready to receive data, it sends the TRANSMIT command. The device responds to the TRANSMIT command by sending a data packet. The computer software is programmed to transmit the commands in a separate thread from receiving and processing the data packets. This asynchronous flow permits the computer to perform various tasks like charting the data and responding to user input at the same time it is obtaining data from the device.

The computer's clock needs to be synchronized with a National Institute of Standards and Technology Internet time server (http://tf.nist.gov/service/its.htm). When the data packet is received, the computer software assigns it a timestamp before charting the data, saving it to disk, or relaying it to the webserver. This places the timestamp close to the data collection time. Using a command-driven protocol improves the communications because the requests for data can be transmitted by the computer when it is ready to respond to receiving the data. The computer can record the timestamp when it receives the data packet with minimal variance in latency between the actual time the sample is recorded and when the timestamp is applied to the data. The data is always the very last sample taken and the data packets are a fixed number of bytes transmitted at a fixed data rate.

## CYCLE PERIOD & FREQUENCY

The dsPIC30F3012 microcontroller oscillator configuration settings enable you to tune the instruction timing by setting a phase-locked loop (PLL) frequency multiplier. The resulting processor frequency, used by the programmable timers, can be prescaled to reduce the timer ticks frequency. To improve the precision of measurement, you need to select a processor frequency that enables you to measure frequency comfortably for both 60 and 50 Hz using a 16-bit unsigned integer value (0 to 65535).

A timer tick's length is four oscillator cycles. If you use a PLL multiplier of four to drive the microcontroller at 48 MHz, the following two equations provide the timer tick counts for 60 and 50 measurements using a timer prescale divider of eight.

$$\frac{\left(\frac{1\text{ s}}{60}\right) \times 12{,}000{,}000\text{ ticks/s}}{8} = 25{,}000\text{ ticks for a 60 period}$$

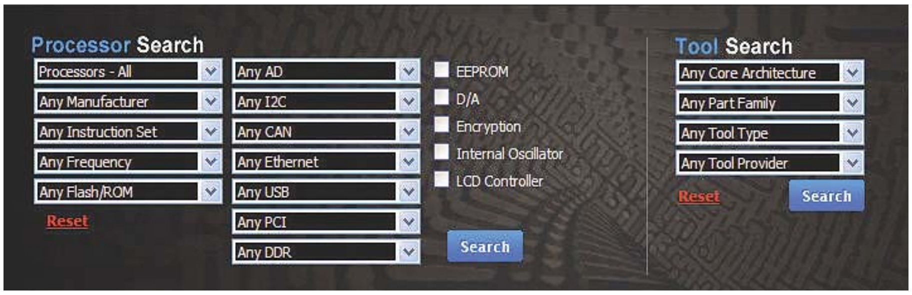
$$\frac{\left(\frac{1\text{ s}}{50}\right) \times 12{,}000{,}000\text{ ticks/s}}{8} = 30{,}000\text{ ticks for a 50 period}$$

The following formula converts the clock tick count to

# Pick a Chip. Any Chip.

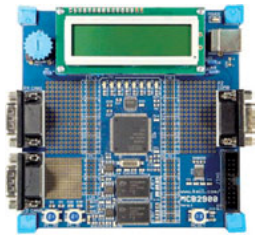
Find a Solution to your next Embedded Challenge.
Do the Research you should, but never had time for.

*Embedded Developer's intuitive research engine helps you speed your chip evaluation time. You don't have to know the manufacturer, chip family or part number--just select the features you want and let us do the rest.*



| Part Number | AT91SAM7X | MCF5208 | LPC2923 |
|---|---|---|---|
| Manufacturer | ATMEL | freescale semiconductor | NXP |
| Core Variant | ARM7TDMI | ColdFire V2 | ARM968E-S |
| Flash | 262144 | 0 | 262144 |
| RAM | 65536 | 16384 | 16384 |
| Max. Freq. | 55 | 166 | 125 |
| Dhrystone MIPS | 50 | 159 | 156 |
| Timer Bits | 16 | 32 | 32 |

*We help you research your best option. Nowhere else can you compare your best options side-by-side from different manufacturers. Click on the device you want, and a product page lets you select Distributor Buy / Quote options, send RFQs, download datasheets, and more. Plus--Hearst stock check gives you up-to-date inventory on every device.*



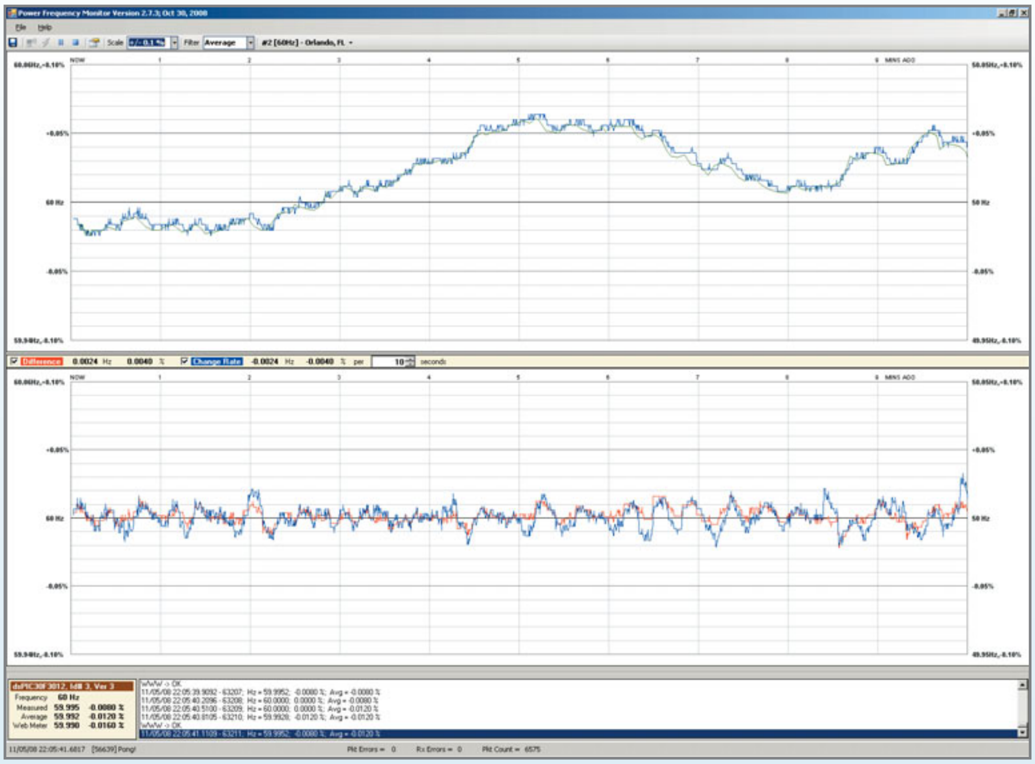
*Once you have the chip that meets your needs, review and compare the hardware and software development tools that support it from multiple manufacturers, and buy them on-line through our shopping cart.*

*Shave days off your schedule with Embedded Developer, the only site in the world where you're only clicks away from finding the chips and tools to get you up and running, quickly. Try EmbeddedDeveloper.com, or EmbeddedDeveloper.cn in Chinese.*

EMBEDDEDDEVELOPER.COM
FIND. COMPARE. BUY.

EMBEDDEDDEVELOPER.CN
查询. 选型. 采购

**The Sites for Engineers with a Job to Do.**



```
WWW -> OK
11/05/08 22:05:39.9092 - 63207;  Hz = 59.9952;  -0.0080 %;  Avg = -0.0080 %
11/05/08 22:05:40.2096 - 63208;  Hz = 60.0000;  0.0000 %;  Avg = -0.0080 %
11/05/08 22:05:40.5100 - 63209;  Hz = 60.0000;  0.0000 %;  Avg = -0.0120 %
11/05/08 22:05:40.8105 - 63210;  Hz = 59.9928;  -0.0120 %;  Avg = -0.0120 %
WWW -> OK
11/05/08 22:05:41.1109 - 63211;  Hz = 59.9952;  -0.0080 %;  Avg = -0.0120 %
11/05/08 22:05:41.6817   [56639] Pong!        Pkt Errors =  0     Rx Errors =  0     Pkt Count =  6575
```

Photo 3—Frequencies are displayed in the top chart. Frequency change and differences are plotted in the lower chart. The bottom of the screen displays a running activity log and various operational indicators.

frequency in hertz.

$$\frac{\text{counted ticks} \times 60}{25{,}000 \text{ ticks}} = \text{calculated}$$

You're also able to compare power grids with both 60- and 50-Hz nominal frequencies. You can normalize the data to a percentage deviation from the nominal frequency using the following calculation.

$$\frac{100\% \times [\text{measured frequency} - \text{nominal frequency}]}{\text{nominal frequency}} = \text{deviation } \%$$

Using this calculation, you can compare power grid frequency data. For example, you can compare New Zealand's 50 frequency (www.systemoperator.co.nz/power-system-overview) to the 60 frequency measured in the United States.

## ASSEMBLY & TESTING

Working with prototype boards is fun. But getting the prototypes to actually work can be a challenge. When building a prototype board assembly, it's good to keep the wiring short, neat, and color-coded. I like using category 5 solid conductor cable wire that gives me eight colors to work with. The convention I follow is to use the solid colors for power and voltage wires: green for ground, orange for 5 VDC, brown for rectified AC, and blue for the 2.5-V DC reference voltage. I use the white/color wires for wires carrying signals like RS-232 and LEDs. When trimming leads on resistors, capacitors, and other components, it is good to cut them to consistent lengths. Component leads of 0.5″ work well for this project.

The PHzMonitor's complete circuitry is shown in Figure 2. After the circuit is assembled and double-checked, the AC transformer is connected and the prototype is ready for programming and testing.

The firmware source code is posted on the *Circuit Cellar* FTP site. After the firmware is programmed, the PHzMonitor produces a 60-Hz pulse-width-modulated, 50% duty cycle, square wave output on pin 10. This 60-Hz signal can be connected to the op-amp input and used for calibrating the system if no other known frequency calibration signal source is available. Two LEDs should be active. The Power LED is on when the device has power. A large red/green activity indicator LED is toggled between on and off with each program loop pass. If the measured frequency is within ±1% of the programmed nominal frequency, the red/green activity indicator pulses green. If the frequency is outside of the ±1% tolerance, the activity indicator pulses red. The process loop performance can be measured from the activity indicator signal pulses using an oscilloscope.

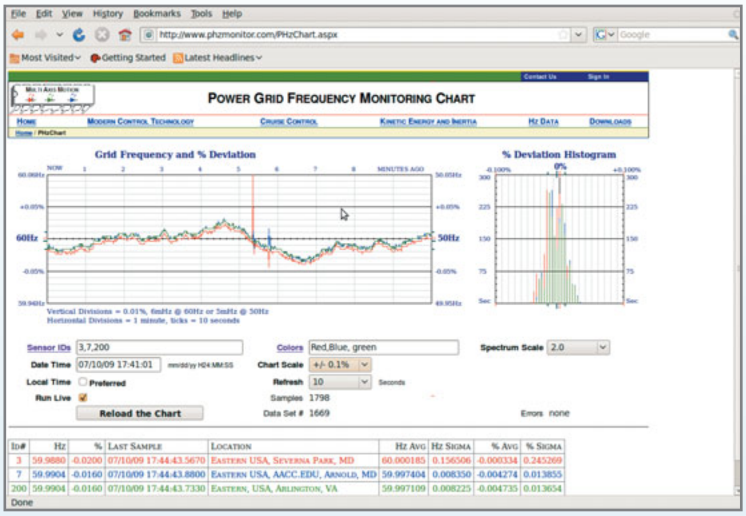
## COMPUTER SOFTWARE

After ensuring the device is functional, the next step is to connect it to the computer's RS-232 port and then load and run the computer software. The computer software for this project was developed using the Microsoft C# programming language, the readily available Visual Studio Express 2008, and .NET Framework development tools. The Microsoft Visual Studio Express editions are free download versions of the Visual Studio development tools. The source code for this project is available on the *Circuit Cellar* FTP site.

Photo 3 is a screenshot of the computer software with local sensor measurement frequency. A selected remote frequency is displayed in the top chart. The difference and rate of change between the two frequencies are plotted in the lower chart. The bottom of the screen displays a running activity log and various operational indicators.

## MONITOR VIA THE 'NET

There are currently multiple sources for grid frequency information. I created a new source! When the frequency sensor and computer are connected to the Internet, the data can be relayed to a web server where people from around the world can view the data live and download it for off-line analysis. The implementation details for setting up the web server for this project are beyond the scope of this article.


Power Grid Frequency Monitoring Chart
Grid Frequency and % Deviation
% Deviation Histogram
60Hz
50Hz

Vertical Divisions = 0.01%, 6mHz @ 60Hz or 3mHz @ 50Hz
Horizontal Divisions = 1 minute, ticks = 10 seconds

Sensor IDs: 3,7,200 — Colors: Red,Blue, green — Spectrum Scale: 2.0
Date Time: 07/10/09 17:41:01 mm/dd/yy H24:MM:SS — Chart Scale: +/- 0.1%
Local Time: Preferred — Refresh: 10 Seconds
Run Live: ✓ — Samples: 1798
Reload the Chart — Data Set #: 1669 — Errors: none

| Id# | Hz | % | Last Sample | Location | Hz Avg | Hz Sigma | % Avg | % Sigma |
|---|---|---|---|---|---|---|---|---|
| 3 | 59.9880 | -0.0200 | 07/10/09 17:44:43.5670 | Eastern USA, Severna Park, MD | 60.000185 | 0.156506 | -0.000334 | 0.245269 |
| 7 | 59.9904 | -0.0160 | 07/10/09 17:44:43.8800 | Eastern USA, AACC.EDU, Arnold, MD | 59.997404 | 0.008350 | -0.004274 | 0.013855 |
| 200 | 59.9904 | -0.0160 | 07/10/09 17:44:43.7330 | Eastern, USA, Arlington, VA | 59.997109 | 0.008225 | -0.004735 | 0.013654 |

Done

**Photo 4**—You can view the live power grid frequency-monitoring chart in a web browser. The chart shows two frequency disturbances between minutes 5 and 6.

However, the source code and database schema are available in the PHzMon_Web.zip file on the *Circuit Cellar* FTP site.

The PHzMonitor project server is a standard computer running Microsoft Windows Server 2008, Internet Information Services, and SQL Server 2005. The live chart display uses a combination of ASP.NET, AJAX, and Javascript. The live chart is shown in Photo 4. The chart graphics are generated using Javascript graphics routines developed by Walter Zorn (www.walterzorn.com). They are available for use with the GNU's Not Unix Lesser General Public License.

In the PHzMonitor project, the web server collects data using several different methods. The computer software for the design transmits the data to the web server using a web service. A web service is a server-side program module that presents a standard interface to the Internet. In my web service, the interface is simply a function call with sensor identification, password, frequency, and timestamp data as parameters. The function call returns an "OK" status if the upload succeeds or error messages if the upload fails .

There are other methods that the web ser ver uses to collect data. One is with a hyper text transport protocol uniform resource locator with sensor identification, a password, a timestamp, and frequency data as parameters. Another is with small custom gateway applications that scrape the frequency data published on other web sites. The web ser ver's time is automatically included with the data records upon receipt of the data. If the timestamps supplied with the data are significantly different from the server-applied timestamps, it means the local computer's clock is not synchronized to a time standard and thus the server timestamps are used for charts and calculations.

## INTELLIGENT DESIGN

The PHzMonitor project is a relatively simple, scalable, inexpensive system. Networked sensors used with the Internet enable you to monitor and study frequency conditions around the power grid—to better understand the grid's performance—and around the world to compare performance between grids.

As we develop more energy resources, they must be integrated efficiently and cost-effectively into our existing energy infrastructure—the power grids. Alternating current power generators must be synchronized to a nominal frequency to provide optimal service and a reliable electrical energy supply.

When you consume electricity, you place a load on the power grid. The technology exists to develop intelligent appliances that can measure the grid's frequency to monitor the health and stability of its performance. These appliances could then be configured to respond appropriately, in a semi-coordinated manner, by scheduling and reducing load during critical times to help maintain grid stability. With strategically located frequency sensors connected to the Internet, network-connected intelligent appliances could compare local conditions to wider area grid performance. This would enable the appliance to make "intelligent" responses that would result in a more stable power grid . ■

*Arnold Stadlin (ajstadlin@multiaxismotion.com) studies electrical engineering technologies at Anne Arundel Community College in Arnold, MD. His interests include manufacturing and industrial technologies, energy measurement and instrumentation, electro-mechanical engineering, robotics, information systems, and computer programming.*

## PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar .com/pub/Circuit_Cellar/2009/231.

## RESOURCES

Internet Time, National Institute of Standards and Technology, http://tf.nist.gov/service/its.htm.

Microchip Technology, "dsPIC30F2011/2012/3012/3013 Data Sheet: High-Performance, 16-bit Digital Signal Controllers," DS70139F, 2008.

———, "MCP1525/1541: 2.5V and 4.096V Voltage References," DS21653A, 2001.

———, "MCP6291/1R/2/3/4/5: 1.0 mA, 10 MHz Rail-to-Rail Op Amp," DS21812E, 2007.

Transpower New Zealand, "Total New Zealand Power System," www.systemoperator.co.nz/power-system-overview.

W. Zorn, Javascript Vector Graphics Library, 2004, www.walterzorn.com.

## SOURCES

**H5C-2E 12-MHz Crystal oscillator**
Fox Electronics | www.foxonline.com

**DS275 RS-232 Transceiver chip**
Maxim Integrated Products | www.maxim-ic.com

**dsPIC30F3012, MCP1525, and MCP6292**
Microchip Technology, Inc. | www.microchip.com

**LM340T5 Linear voltage regulator**
National Semiconductor Corp. | www.national.com

by Ed Nisley

# Capacitor ESR Measurement

Most commonly available instruments don't measure equivalent series resistance (ESR). Inspired by the design of a classic pure-analog ESR meter he studied on the Internet, Ed recently built an analog front end for an MCU that measures ESR. In this article he describes the design and the math associated with the project.

An ideal capacitor presents pure capacitive reactance to the circuit surrounding it. Real-world capacitors add both inductance and resistance, causing interesting effects that you've seen in previous columns and, perhaps, in your own circuits. Worse yet, electrolytic capacitors deteriorate as their electrolyte ages and their equivalent series resistance increases, causing gradual circuit failure.

It's easy enough to measure capacitance and inductance with the appropriate meter, but commonly available instruments don't measure equivalent series resistance (ESR). I decided it was time to add an ESR meter to my electronics workbench and, of course, building one is the best way to understand the principles involved.

While it's entirely possible to build an analog ESR meter, it's getting harder to find moving-coil meters and even more difficult to pry them apart to recalibrate their scales. I built an analog front end for an Arduino microcontroller that measures ESR and displays its value, with a bit of math in between to calibrate the reading.

## MEASURING AC RESISTANCE

The equivalent series resistance of a capacitor is an AC parameter that cannot be measured with a DC ohmmeter. If a capacitor shows a measurable DC resistance, it's either charging toward the ohmmeter's applied voltage or its dielectric has become *leaky*, perhaps in both the electrical and physical meanings.

Therefore, I used an AC technique to measure ESR: put the capacitor in series with a known resistor, apply a known square-wave AC signal, measure the voltage across the resistor, then calculate ESR from the known quantities.

The resistor's current is always in phase with respect to the applied voltage, because typical resistors have negligible stray capacitance and inductance. That is not true for capacitors, as the current through a capacitor has both quadrature and in-phase components with respect to the applied voltage, which arise from its reactance and ESR, respectively.

A capacitor's reactance depends on both its capacitance and the AC signal frequency.

$$X_C = \frac{1}{2\pi fC}$$

I picked a test frequency of 33 kHz, for reasons you'll see later, where a 10-µF capacitor has a reactance of about half an ohm. This circuit is thus most useful for relatively large electrolytic capacitors with correspondingly low reactances: precisely the ones most

## Let your geek shine.

Meet Landon Cox, proud father, SparkFun customer, and developer of a novel timing system for speed climbers that is both precise and easier on climbers' hands. Using laser switches, hand-built enclosures, and plenty of brutal tests by his speed climber daughter, Landon debuted his timing system at the USAC Nationals earlier this year.

Whether you're searching for a breakout board, or prototyping a climbing sensor to withstand high-impact abuse, the tools are out there. Create a project you'll love, and let your geek shine too.

sparkfun™ ELECTRONICS

**Sharing Ingenuity**
WWW.SPARKFUN.COM

©2009 SparkFun Electronics, Inc. All rights reserved. All other trademarks contained herein are the property of their respective owners.

Get the scoop on Landon's Speed Climbing System development at http://www.landoncox.com/ or http://sawdust.see-do.org/. Hey Chauncey, congratulations in taking second place in the women's open category at the ABS Nationals!

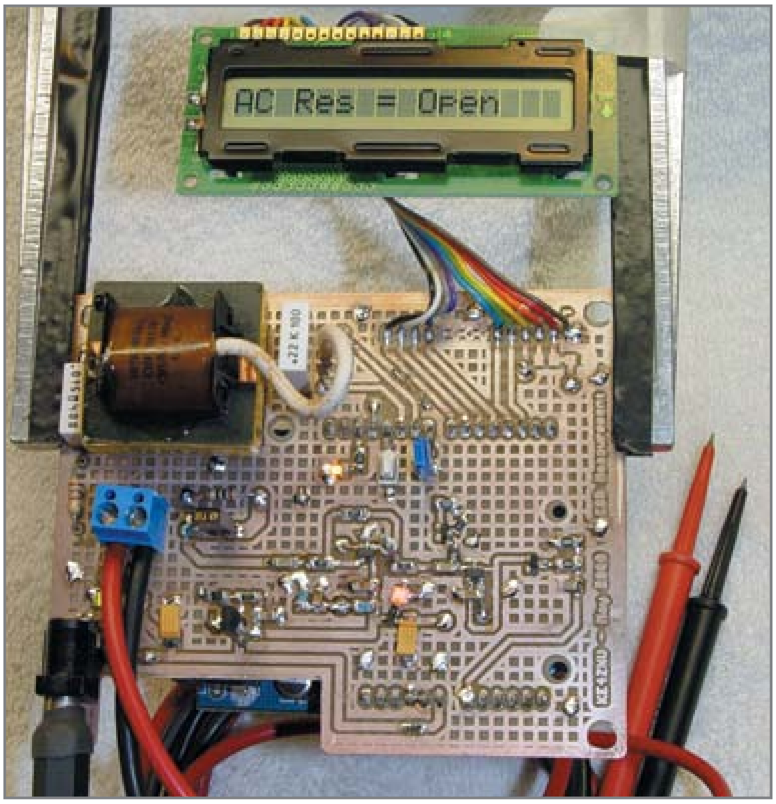


Photo 1—The Arduino Diecimila attached below this analog board drives the transformer's high-voltage winding with a pair of 5-V, push-pull signals to produce 400 mV$_{PP}$ on the low-voltage winding. The signal flow across the board generally follows the schematic, with the Diecimila reading the peak-detected result through an analog input.

likely to fail with high ESR.

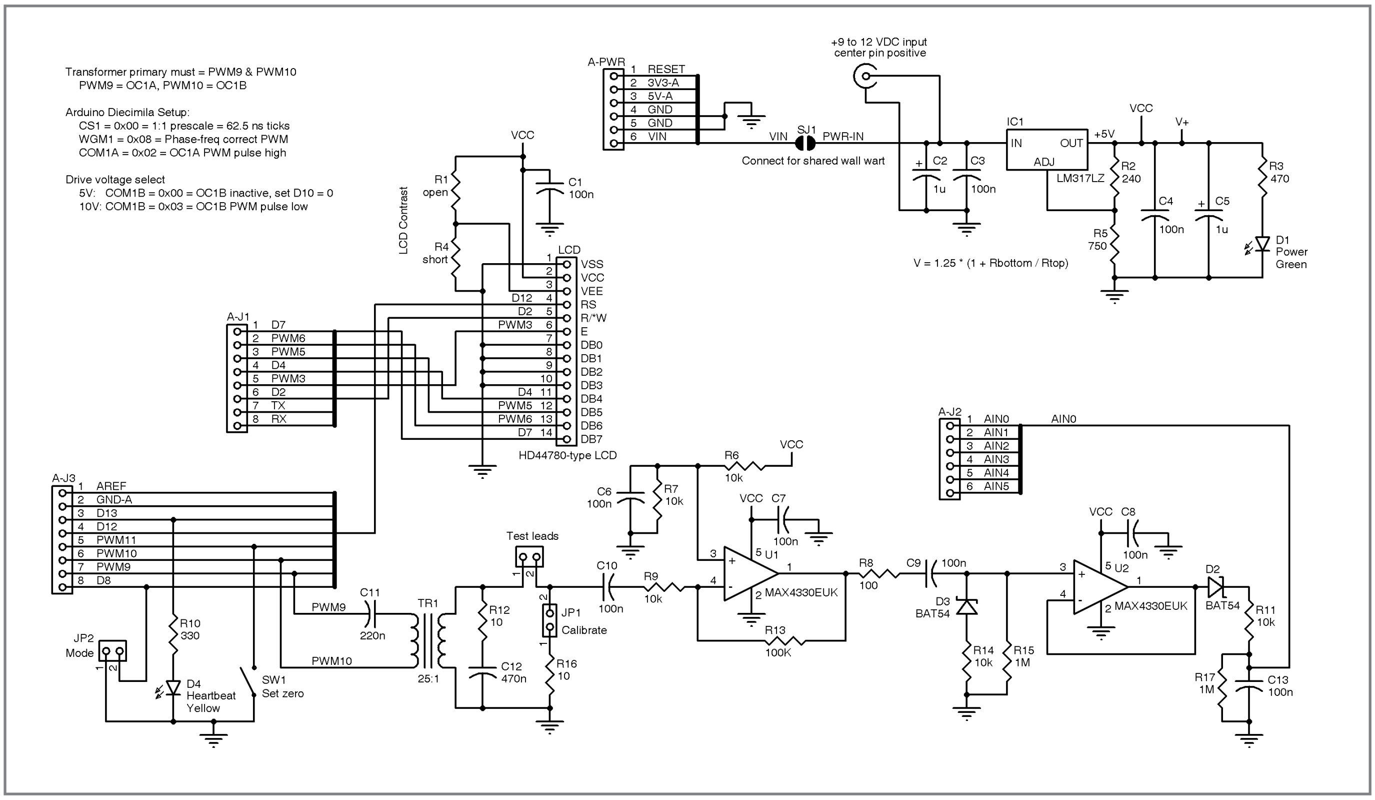
Assuming that the ESR and resistor are considerably larger than the capacitive reactance, then the components behave like a simple resistive voltage divider.

$$\frac{V_{RESISTOR}}{V_{AC}} = \frac{R}{ESR + R}$$

Solving for ESR shows that the ESR is inversely proportional to the voltage across the resistor, at least when that voltage is much smaller than the applied AC voltage.

$$ESR = R \times \left( \frac{V_{AC}}{V_{RESISTOR}} - 1 \right)$$

Two limiting cases pop out immediately: open circuit and dead short. With the test leads open, no voltage appears across the resistor and the voltage ratio is infinite. With the test leads shorted, the ratio is exactly unity. All capacitors should be somewhere in between those limits, at least according to that simple model, but that need not be exactly



Transformer primary must = PWM9 & PWM10
PWM9 = OC1A, PWM10 = OC1B

Arduino Diecimila Setup:
CS1 = 0x00 = 1:1 prescale = 62.5 ns ticks
WGM1 = 0x08 = Phase-freq correct PWM
COM1A = 0x02 = OC1A PWM pulse high

Drive voltage select
5V: COM1B = 0x00 = OC1B inactive, set D10 = 0
10V: COM1B = 0x03 = OC1B PWM pulse low

Figure 1—The transformer produces a 400-mV$_{PP}$ square-wave voltage that divides across the test capacitor's ESR and R16. Capacitors greater than about 10 µF have a negligible reactance at the 33-kHz test frequency. C10 blocks any residual DC voltage across the test capacitor. D3 clamps the amplified signal to 0 V so that the output voltage is very nearly the total peak-to-peak value.

FORMERLY ARM DEVELOPERS' CONFERENCE:

# ARM® techcon³

DESIGN TO THE POWER OF THREE

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

TO THE **SANTA CLARA CONVENTION CENTER**

## Get 3X the Solutions for your ARM Powered Designs

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

**Three Events in One!**

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

**Energy Efficiency**
Leveraging energy efficient SoC strategies to minimize power requirements

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

**Internet Everywhere**
Developing applications for a connected world

**REGISTER NOW for only $495***

**Use Promo Code: ARM09EarlyBird**

*A Discount of $200 off Std. Registration

WWW.ARMTECHCON3.COM

MEDIA SPONSORS:

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

BROUGHT TO YOU BY:

Convergence PROMOTIONS the RTC Group

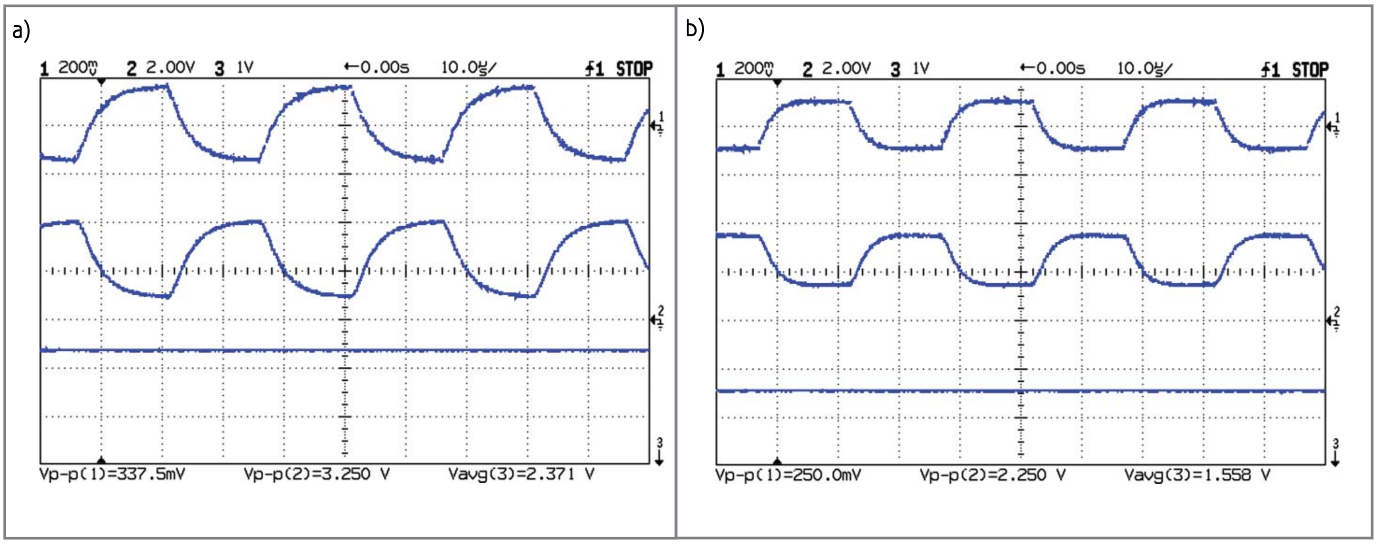


**Figure 2a**—The 10-$V_{PP}$ drive into the transformer's high-voltage winding puts 337 $mV_{PP}$ across the 10-Ω load resistor with the probe tips short-circuited, for a 2.371-V output to the Arduino. **b**—A poor-quality, 10-µF capacitor with an ESR of 7.4 Ω produces 1.558 V.

true in the real world.

Photo 1 shows that the ESR meter recognizes the open-circuit condition and displays a useful message. The short-circuit condition is a bit harder to distinguish from a very low ESR. The circuitry shown in Figure 1 implements that general outline, with an Arduino Diecimila microcontroller connected to the four pin-header strips.

## TEST DRIVE

Because the applied voltage divides across the capacitor and resistor in proportion to their resistances, the resistor should have the same order of magnitude as the expected ESR. A "good" capacitor's ESR will be less than a few ohms, at most, and is typically under 1 Ω. A failing capacitor will

Cellular and GPS capable

**Data Mover**

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

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

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

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

**JK *microsystems***

**AP CIRCUITS**
PCB Fabrication Since 1984

As low as...
**$9.95**
each!

Two Boards
Two Layers
Two Masks
One Legend

***Unmasked boards ship next day!***

**www.apcircuits.com**

VISA MasterCard PayPal IPC MEMBER ASSOCIATION CONNECTING ELECTRONICS INDUSTRIES BBB

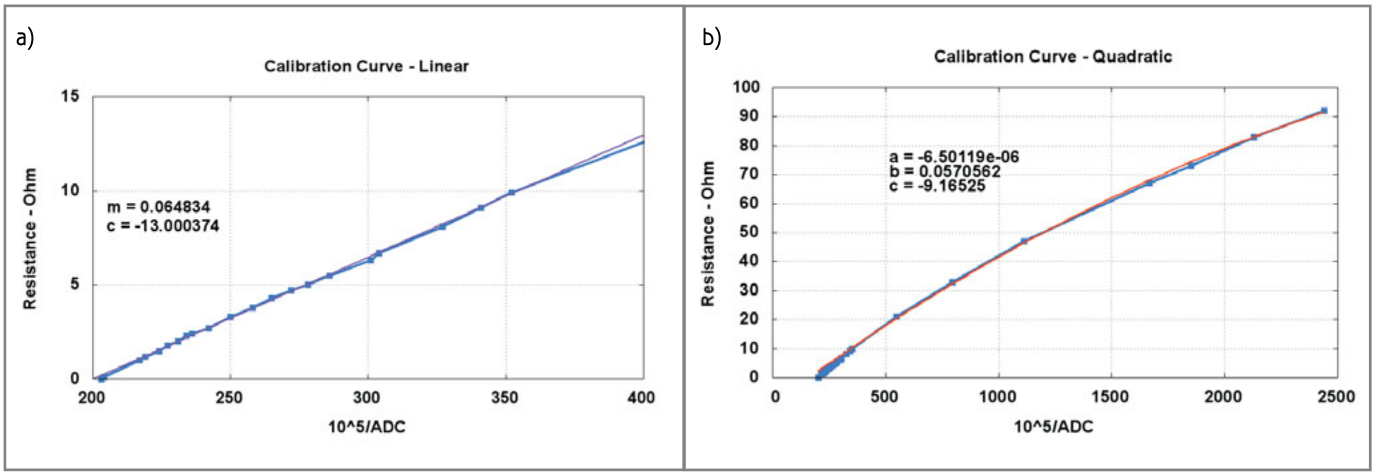


Figure 3a—A linear fit is almost embarrassingly accurate below 10 Ω, where the ESR meter is most useful. b—A quadratic curve fit to the range from 10 to 100 Ω gives sufficient accuracy to identify truly bad capacitors.

have an ESR ranging from a bit under 10 Ω to essentially an open circuit.

I picked a 10-Ω resistor, which sets the voltage ratio $V_{AC}/V_R$ of 11 for a 1-Ω ESR. On the other end of the range, the 100-Ω ESR of a dead capacitor will show a ratio of 1.1. ESR values below 1 Ω may be somewhat less accurate, but those values correspond to reasonably good capacitors.

Although it's easy enough to measure individual capacitors on the bench, you're generally confronted with a circuit board containing a few bad capacitors: unsoldering every one is both labor-intensive and hazardous to the surrounding circuitry. Measuring ESR by probing the capacitor in the circuit, without unsoldering anything, requires an AC voltage well below the usual semiconductor junction threshold voltage of about 500 mV.

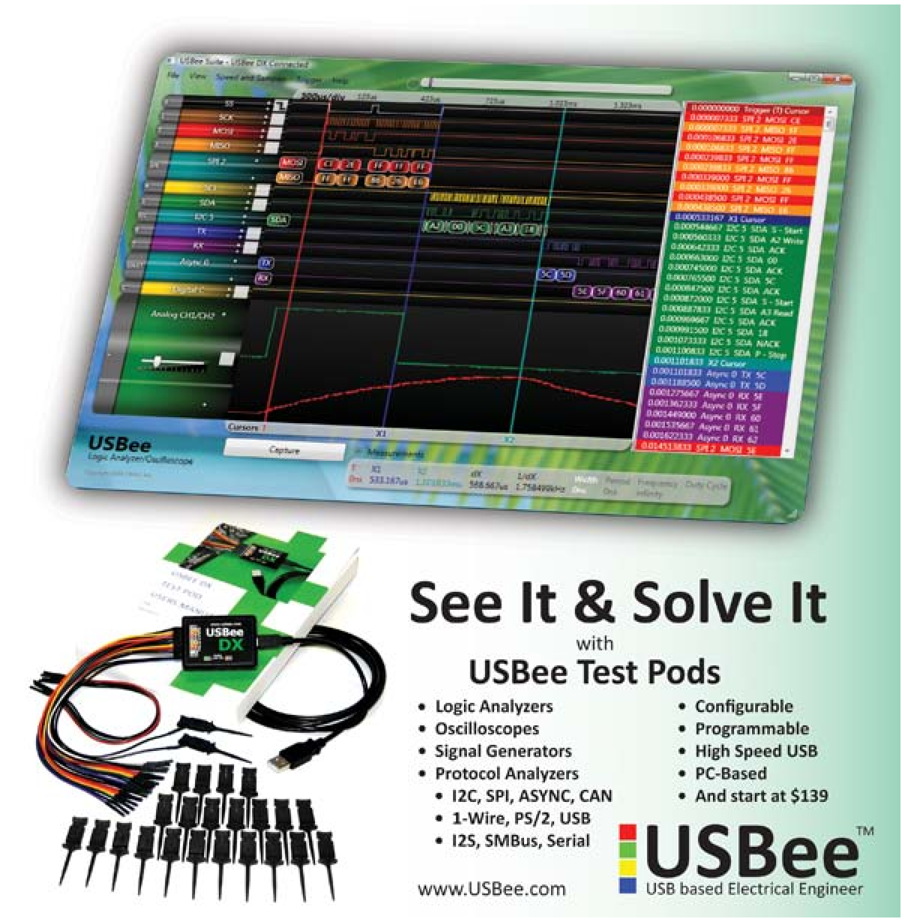
The ESR and reactance of the capacitor should be much lower than the other circuit resistances, so, with the junctions turned off, the capacitor is effectively isolated from the surrounding circuitry. Because the test voltage is so low, there's also no risk of damaging anything else.

The maximum test current occurs with the probe tips shorted together, which will produce:

$$50 \text{ mA} = \frac{500 \text{ mV}}{10\ \Omega}$$

That's a relatively high current, at least in terms of ordinary microcontroller outputs, and suggests that driving the capacitor and resistor directly from digital pins won't work, even using a resistive divider to cut down the voltage applied to the circuit.

Instead, I used a transformer to step the AC voltage down and the current up. In fact, this project came about when I realized that the transformer I used for the high-voltage dosimeter charger power supply ("A Blast for the Past: High-Voltage DC Dosimeter Charger," *Circuit Cellar* 229, 2009) had both the proper turns ratio for an ESR tester and enough core volume to support the output



current. All I had to do was drive the transformer backwards, with the microcontroller feeding the high-voltage winding, to get an AC voltage in the right range with a low source impedance.

I derived a Spice model for the transformer using the data I presented in that column, and then simulated a simplified version of the schematic in Figure 1. While I don't have room to discuss that here, you can fetch the model from the *Circuit Cellar* web site and see how it behaves for various ESR and capacitive loads.

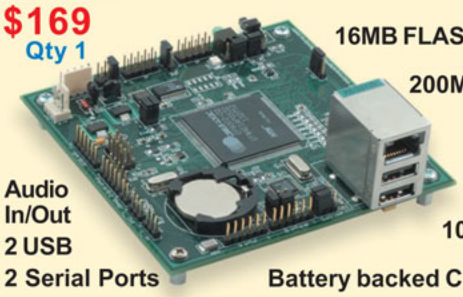
The Atmel ATmega168 on the Arduino board drives the high-voltage winding using a pair of PWM outputs running in push-pull mode. The transformer's 25:1 turns ratio converts that 10-$V_{PP}$ drive signal into 400 $mV_{PP}$ across the capacitor and 10-Ω resistor, so the surrounding in-circuit components see only ±200 mV, well below the threshold voltage of even those old-school germanium diodes.

**Listing 1**—The ATmega168 ADC hardware provides a 10-bit integer value. Finding the actual AC resistance uses a linear curve fit below 10 Ω and a quadratic fit up to 100 Ω. An open circuit produces ADC values near 0, which must be removed from the calculation. Low-value capacitors produce ADC readings indicating negative resistance, which the display code in Listing 2 will flag as invalid.

```
long int LinNumerator = 648341;
long int LinOffset = -1301;

float CoeffA = -6.50119e-6;
float CoeffB = 0.0570562;
float CoeffC = -9.16525;

float ScaleADC = 1e5;
float ScaleResistance = 10.0;

int RawSense;
int Resistance;
float FloatRes;
float x;

RawSense = analogRead(PIN_VSENSE);

if (RawSense <= 1) {
       Resistance = RESISTANCE_BAD;
}
else {
       Resistance =  LinNumerator / RawSense + LinOffset;
       if (Resistance > RESISTANCE_BREAK) {
       x = ScaleADC / RawSense;
       FloatRes = x * (CoeffA * x + CoeffB) + CoeffC;
       Resistance = ScaleResistance * FloatRes + 0.5;
       }
       if ((Resistance > RESISTANCE_BAD) || (Resistance == 0) ) {
       Resistance = RESISTANCE_BAD;
       }
}
```

**Easy Embedded Linux**

**$169**
Qty 1



16MB FLASH / 32MB RAM
200Mhz Arm9 CPU
16 Digital I/O
Watchdog
10/100 Ethernet
Battery backed Clock/Calendar
Audio In/Out
2 USB
2 Serial Ports

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

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

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

**JK microsystems**

**Prolific USA**

*World Class Solutions*

**www.ProlificUSA.com**

USB 3.0
USB 2.0
GPS
SoC

Listing 2—Displaying the resistance requires working around the DMC16117 LCD's peculiarities, as well as the Arduino library's limited floating-point support, while sorting out the meaning of the resistance value.

```
lcd.home();
lcd.print("AC Res =");
lcd.setCursor(0,1);

if (Resistance >= RESISTANCE_BAD) {
        lcd.print(" Open");
}
else if (Resistance < 0) {
        lcd.print("Low cap?");
}
else {
        lcd.write(' ');
        lcd.print(Resistance / 10,DEC);
        if (Resistance <= RESISTANCE_BREAK) {
        lcd.write('.');
        lcd.print(Resistance % 10,DEC);
        }
        lcd.write(' ');
        lcd.write(CHAR_OMEGA);
}
lcd.print("        ");
```

The maximum test current with shorted probes is therefore just:

$$20 \text{ mA} = \frac{200 \text{ mV}}{10\ \Omega}$$

I had previously verified that the transformer core could support about 60 mA in its low-voltage winding, so the core remains far from saturation in this application.

Op-amp U1 amplifies the voltage across R16 by a factor of 10, increasing the 400 $\text{mV}_{\text{PP}}$ signal to about 4 $\text{V}_{\text{PP}}$, and centers it at 2.5 V. C10 isolates the input from any residual DC charge on the test capacitor: even with the external circuit turned off, dielectric absorption in large electrolytic capacitors can hold several volts for several days.

Diode D3 clamps the bottom of the amplified AC waveform to 0 V, putting the highest voltage at 4 V and providing a factor-of-two gain over a simple half-wave rectifier. Because the op-amps have very high input resistances, R15 draws a tiny DC current to pull the waveform voltage toward ground.

U2 buffers that signal and D2 stores the peak voltage on C2 for the Arduino's ADC input, with R17 ensuring that the capacitor doesn't store any voltage when the input drops to zero.

Figure 2a shows the voltages with the probes short-circuited. The transformer supplies 337 $\text{mV}_{\text{PP}}$, rather than the 400 $\text{mV}_{\text{PP}}$ predicted by the turns ratio, due to real-world factors of winding resistance and leakage inductance. The waveform obviously isn't as square as you'd expect.

This transformer was originally intended for a high-voltage power supply, so it's overbuilt for this application. I measured about 28 µH of leakage inductance while determining the values for snubber R12–C12.

The time constant of the secondary circuit depends on the inductance and the total resistance. The secondary winding adds 2 Ω of DC resistance to the 10-Ω resistor. The primary winding's 350 Ω gets scaled down by the square of the turns ratio to 0.6 Ω, for a total of about 13 Ω, making the time constant rather long.

$$\tau = \frac{L}{R} = \frac{28\ \mu\text{H}}{13\ \Omega} = 2.1\ \mu\text{s}$$

That's in rough agreement with what you see in Figure 2a. The leading and trailing edges of the "square wave" don't reach their final values during the 15-µs half-periods, so the time constant must be on the order of 4 µs: about 20 µs and five time constants to reach 99% of the final value. That's why I picked 33 kHz for the AC signal frequency; a smaller

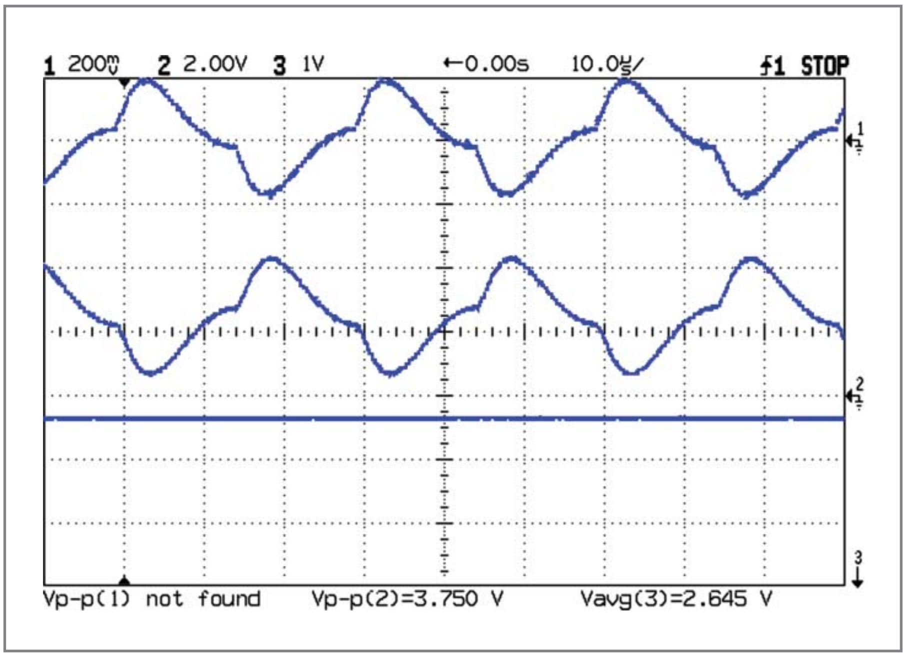


Figure 4—Small-value capacitors form a resonant circuit with the transformer's leakage inductance. The snubber damps the oscillation, but the result no longer resembles the square-wave drive and violates the assumptions built into the ESR meter. Fortunately, these capacitors generally don't have ESR problems.

transformer with lower leakage inductance would permit a higher frequency.

Figure 2b shows the voltages for an old 10-µF capacitor with a (bad!) 7-Ω ESR. The increased resistance reduces the secondary circuit time constant so that the signal much more closely resembles a square wave.

The difference between the peak-to-peak voltage at the output of U1 (middle trace) and the DC voltage across C13 (lower trace) in Figures 2a and 2b isn't a constant. The combined effect of diode nonlinearity in the clamp and peak-hold circuits is just one more thing to consider when you're building something like this.

You can explore those effects, as well as the time constant, using the Spice model.

## CURVE FITTING

Much of analog circuit design depends on the assumptions of linear component behavior. Unfortunately, as you've seen, real components may be only approximately linear, so making accurate measurements can introduce bewildering complexity. A dab of digital circuitry and programmed logic can both improve accuracy and reduce complexity, at the cost of introducing all the usual software problems.

"A dab of digital circuitry and programmed logic can both improve accuracy and reduce complexity, at the cost of introducing all the usual software problems."

Rather than forcing the measured values through an idealized equation relating voltage to ESR, I decided to calibrate the circuit by simply recording the ADC readings for various known resistances, then fitting a low-order curve to the results. The two plots in Figure 3 show how that worked out.

The ADC produces a 10-bit value covering the range from 0 to 5 V (with a 5-V power supply). The reciprocal of that value should be roughly proportional to the ESR. To preserve an integer result, I divided the ADC value into $10^5$ to get a number between 97 and 50,000. I eliminated ADC values 0 and 1, which correspond to the open-circuit condition, to prevent a divide-by-zero problem.

Gnuplot plotted the measured points and fit curves to the data, producing coefficients that I hard-coded directly into the Arduino program's source. In a more production-ready program, you'd want to update the coefficients without reprogramming the chip, but that's in the nature of fine tuning.

Figure 3a shows the relation between the actual resistance and the scaled reciprocal for resistances below 10 Ω: an almost embarrassingly straight line. The highest ADC value was around 450 for the probes-shorted condition, giving a scaled reciprocal around 200.

Figure 3b shows that the relation for resistances above 10 Ω isn't nearly as linear, so I had Gnuplot fit a quadratic curve to those points. The result is within about an ohm of the actual resistance, which is close enough for my purposes and probably better than the accuracy of my other resistance meters.

Homework: have Gnuplot fit the proper reciprocal equation to the raw ADC values. Compare and contrast the results with my equations.

Because the quadratic coefficients didn't lend themselves to easy integer scaling, I simply defined some `float` variables, which hauled the Arduino floating-point package into the program. Surprisingly, that added very little to the overall size: the entire program still occupies only a quarter of the available Flash ROM space.

Listing 1 shows the code that converts ADC inputs to resistances. The Arduino output routines don't handle floating-point variables, so I scaled the `Resistance` variable to 10 times the actual resistance: 1 Ω corresponds to `Resistance=10`. The linear coefficients incorporate that value, but it's applied separately to the quadratic result.

The code in Listing 2 displays the numeric value on the LCD, a process somewhat complicated by the bizarre nature of the (obsolete) DMC16117 hardware. This could probably be cleaned up a bit with some `sprintf` trickery, but the overall result wouldn't be much simpler.

## MEASURED RESULTS

The resistance calculated from the ADC values will be positive for resistances between zero (probes shorted) and infinity (probes open). The straight-line fit to the low-resistance points is negative for ADC readings below the shorted-probes value, which shouldn't happen under normal circumstances.

Right?

Wrong: the real world intrudes once again, in the form of resonance. The transformer leakage inductance of about 28 µH forms a tank circuit with the external capacitance, the snubber capacitor, and the winding's parasitic capacitance. Knowing the driving frequency and the tank inductance, the capacitance is easy to find.

$$C = \frac{1}{(2\pi f)^2 L}$$

$$850\ \text{nF} = \frac{1}{\left(2\pi\ 33 \times 10^3\right)^2 \cdot 28 \times 10^{-6}}$$

The snubber prevents any actual oscillations and lowers the Q, but external capacitances below 1 µF put nice peaks on the nominally square waveform.

Figure 4 shows the waveforms produced by a perfectly good 220-nF polyester capacitor. The peak-to-peak and ADC voltages are higher than a dead short, so the linear-fit equation would indicate a negative resistance. Obviously, the model has diverged from reality: no capacitor has a negative ESR!

Fortunately, low-value capacitors with solid-film dielectrics generally don't have ESR problems. For those high-value electrolytic capacitors troubled with bad ESR, this circuitry and software give reasonably accurate results.

I found a few ancient aluminum-can electrolytic capacitors in my junk box, which I now know are suited only for the scrap pile. Three of the four sections in one can were still within 20% of their nominal capacitance, but had resistances ranging from 3.2 to 28 Ω.

Most of my newer surface-mount caps have ESRs well under 1 Ω, although a group of 10-µF, 50-V units measure around 7 Ω. All of the high-value ceramic caps are indistinguishable from shorted probes: precisely what you want in a capacitor!

The only remaining challenge is stuffing the circuit board and LCD into a case, but that's just a machine-shop project.

## CONTACT RELEASE

If you must measure capacitors with extremely low ESRs, substituting a 1-Ω resistor for R16 will improve the resolution. However, the reduced secondary resistance will require a much lower test voltage or a higher test current, all with far more attention to circuit noise levels.

The classic pure-analog ESR meter in the Resource list inspired my version. I wanted to discuss the transformer and math a bit more, so adding a microcontroller seemed like a nice fit. You may prefer the elegance of a moving-coil meter needle. ■

*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 schematics, a PCB layout, and Spice simulation, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/231.

## RESOURCE

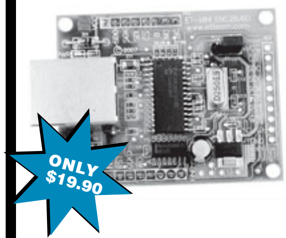
Pure analog ESR meter, www.ludens.cl/Electron/esr/esr.html.

## SOURCES

**Arduino Diecimila**
Arduino | www.arduino.cc/en/Main/ArduinoBoardDiecimila

**ATmega168 Microcontroller**
Atmel Corp. | www.atmel.com

**Save Up To 60% On Electronic Components**

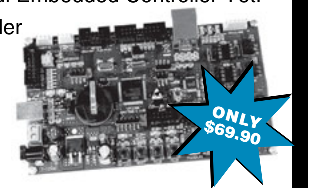


**Exciting New Mini-Boards**

Wide Range Including,
- New Ethernet Board
- SD Card
- Memory Cards
- Real Time Clocks
and Much More

**Powerful New ARM2368 Controller**

Now available our most powerful Embedded Controller Yet!
- Includes LPC2368 Microcontroller
  - 512kb Flash Memory
- SD Card Socket
- Ethernet LAN Connection
- 2 CAN Network Controllers



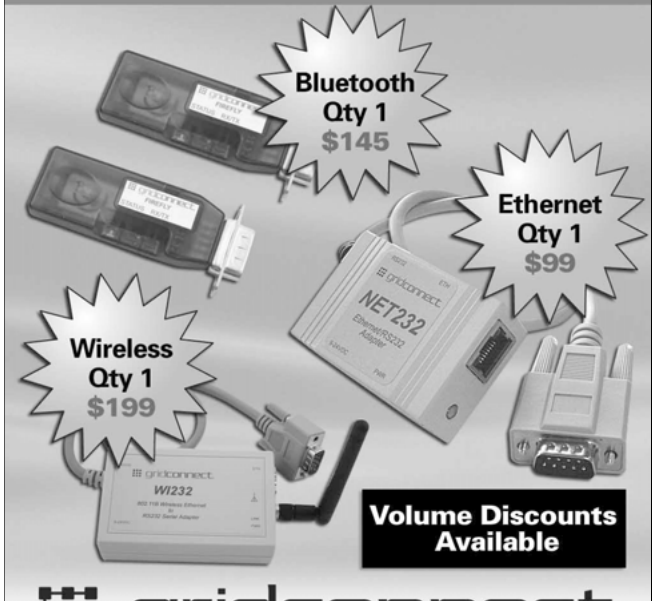
**Save Heaps on Components**

Just arrived our new range of Solar Controllers, Gas Sensors and Wireless Data Link Modules, all at very competitive prices.

We are your one-stop shop for Microcontroller Boards, PCB Manufacture and Electronic Components.

**www.futurlec.com**

**Got Serial, Need Network?**



**Volume Discounts Available**

gridconnect™
**www.gridconnect.com**
**+1 800 975-4743**

by George Anderson

# Digitally Controlled Amplifier

## Connect Vintage Analog to Modern Digital Tech

The dsPIC30F2023-controlled MiniTron is a high-end vacuum tube stereo amplifier with distortion control, power output, and more. The fully functional amplifier successfully blends its unique circuitry and specialized processing software to precisely match the radically different worlds of high-voltage analog and low-voltage microcontrollers.

When I first became interested in electronics, vacuum tube technology was the only thing available to the average experimenter. (Junk radios and TV sets offered a plentiful supply of free parts.) As a result, I learned to build guitar amps, stereo equipment, and ham radio equipment with vacuum tubes. Later, I learned to use the bipolar transistor, MOSFET, and integrated circuit when they became available. I quickly embraced the digital revolution and even built a music synthesizer in the early 1970s using RTL logic ICs. Then, when the Microchip Technology PIC16C54 appeared, I created all sorts of cool gizmos and a few commercial products using that and also newer Microchip devices.

Since those first crude amplifiers constructed from old TVs in the 1960s, I have been interested in audio technology. I've built most of my stereo equipment—everything from low-powered headphone amps to kilowatt-level monster amps, preamps, guitar amps, and effects. These used bipolar transistors, MOSFETs, ICs, and, yes, vacuum tubes. My audio-related tools and parts—especially the vacuum tube equipment—are segregated from my other electronics. I actually have a separate audio-related workbench that shares only the scope and computer.

Audio electronics technology has evolved a lot in the past 40 years. We now have class G amplifiers, class H amplifiers, and even pure digital (class D) amplifiers with DSPs that process audio in the digital domain and convert it to analog only in the switch-mode output stage. Modern audio equipment offers power output, efficiency, distortion, and size advantages that weren't even dreamed of only a few years ago. Despite these advantages, there remains a small but growing minority of audiophiles who believe that vacuum tubes offer better sound quality despite poorer measured performance. A vacuum tube purist will tell you that the audio path should not contain any silicon and there should not be any negative feedback used to improve linearity. Often, the preferred design is a simple single-ended class A circuit. I am not going to choose sides. My vacuum tube amplifier designs often use silicon in the signal path and moderate amounts of feedback.

Vacuum tube audio amplifiers are notoriously inefficient. A class A single-ended amplifier may operate at a plate efficiency of 5% or less. Great advances have occurred in the field of efficiency improvement in the solid-state world. Class G audio amplifiers use multiple power supply rails and switch between them based on the instantaneous power demand. Class H audio amplifiers use a variable power supply that tracks the audio signal giving the output devices enough voltage headroom to do their jobs, and no more. The voltage across the output devices stays constant regardless of the power output. This improves efficiency and offers the added benefit of removing certain distortion products. This technique has been applied to other electronic devices, particularly radio transmitters like cellular phones and cellular base stations to reduce energy consumption.

Specialized Microchip Technology dsPIC ICs make modulated supply rails fairly simple, so I recently chose to design a dsPIC-enhanced vacuum tube amplifier. Refer

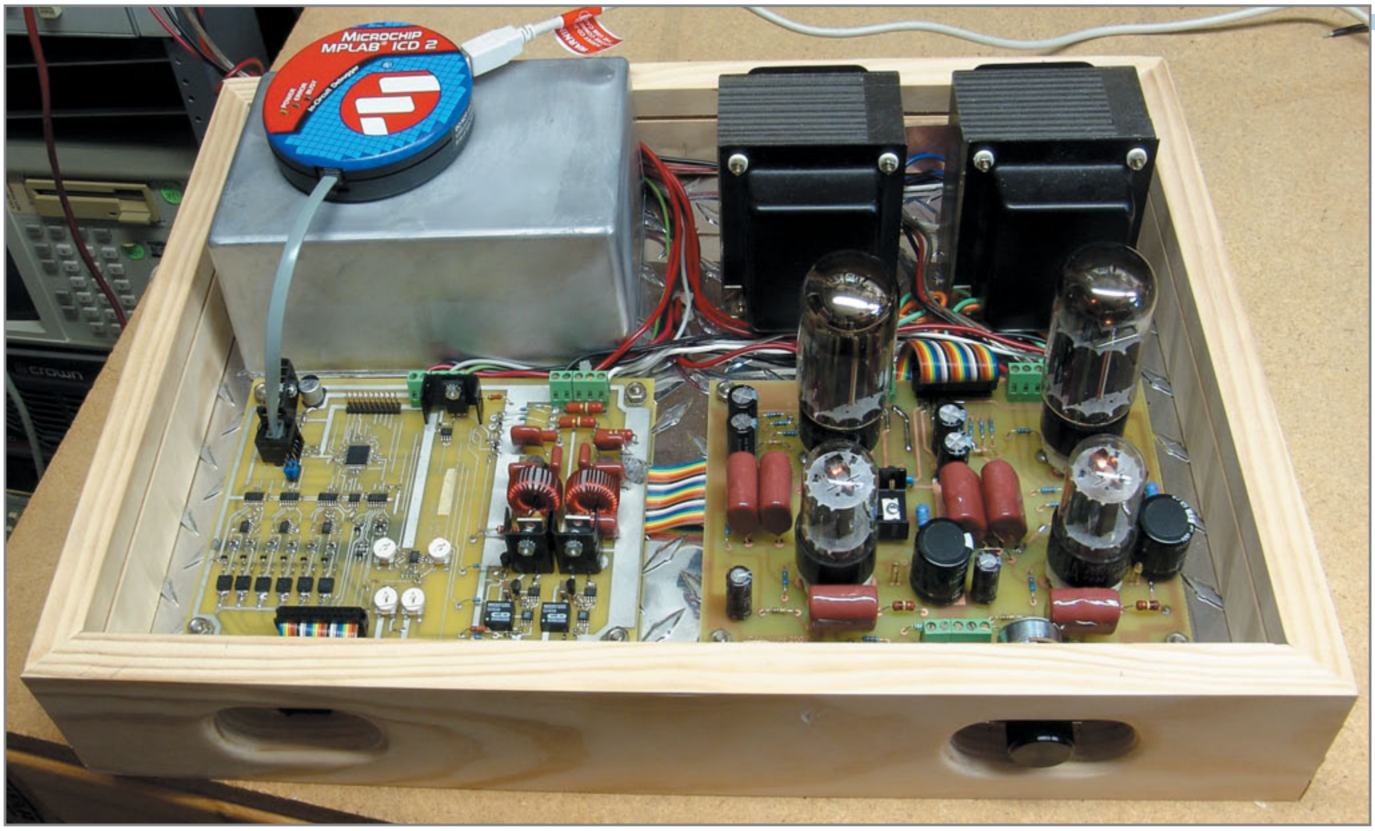


Photo 1—This is the finished MiniTron amplifier. I removed the Lexan cover for debugging. I took this photo during a debugging session. I optimized the amplifier's performance with a laptop computer.

to Photo 1 to see my first attempt at a digitally controlled amplifier design. I call it "MiniTron." The hardware is intended to be a development platform to learn how to connect the vintage analog world to the modern digital world. I included circuitry for software features that have not yet been implemented. As you can see in Figure 1, three main subsystems comprise the design: an amplifier, a controller, and a power supply. Each resides on its own PC board.

In this article, I'll describe my design in detail. You can apply the techniques I'll present to countless other applications. But before I get into the specifics, let's focus for a moment on vacuum tubes.

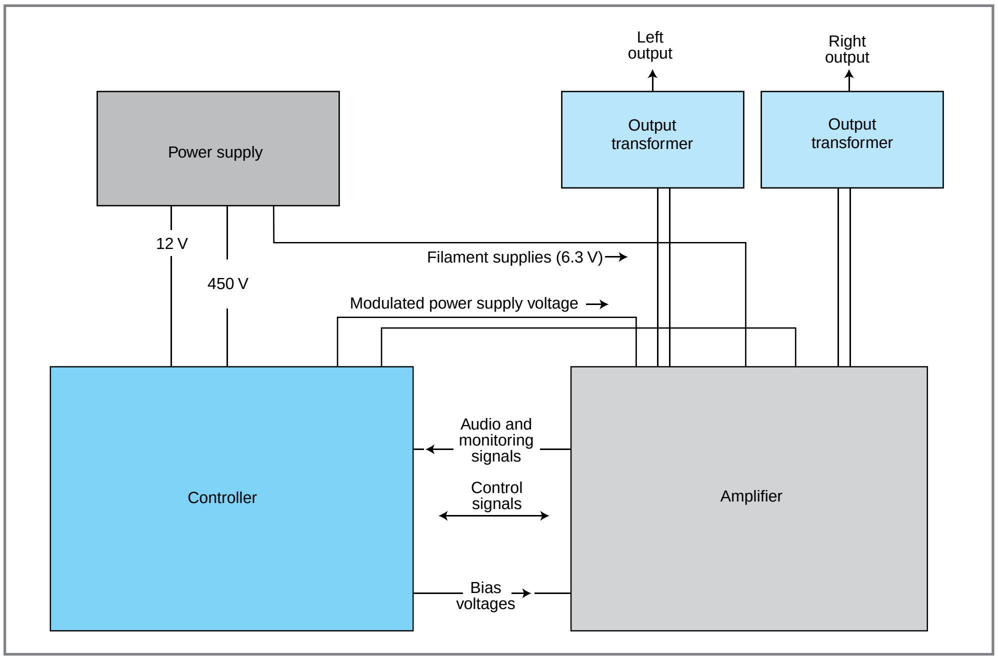


Figure 1—Look at how the controller, vacuum tube amplifier, power supply, and output transformers are interconnected. The individual blocks on this diagram are in the same relative position as their respective subsystems in Photo 1.

## VACUUM TUBE THEORY

Vacuum tubes work on the principle of thermionic emission. Electrons are emitted from a hot cathode and travel through a vacuum toward a positively charged electrode, the plate. Charged wire structures between the two electrodes have the ability to regulate the current flow by altering the electric field in a manner somewhat similar to the gate in a MOSFET. Most vacuum tubes function like depletion-mode MOSFETs.

Celebrating Our 10th Year Anniversary

**PC Scope & Generator IN ONE!**

PCSGU250
Oscilloscope + Function Generator
$238.55

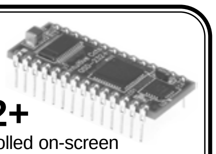
PCGU1000
USB PC
Function Generator
$242.85

PCSU1000
USB PC
Oscilloscope
$298.99

DesignNotes.com
*What Your Electronic Hobby Stores Used To Be*

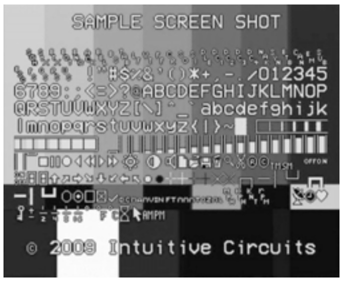
1-800-957-6867 www.DesignNotes.com

New!



**OSD-232+**

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

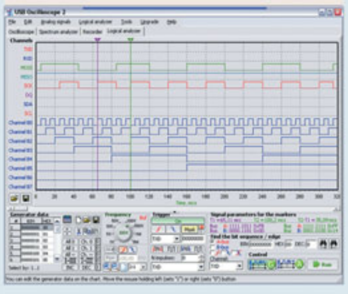

SAMPLE SCREEN SHOT
!"#$%&'()*+,-./012345
6789:;<=>?@ABCDEFGHIJKLMNOP
QRSTUVWXYZ[\]^_`abcdefghijk
lmnopqrstuvwxyz{|}~
© 2008 Intuitive Circuits

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

USB Oscilloscope for $169.50

Logic and Spectrum Analyzers, Generator.

www.HobbyLab.us



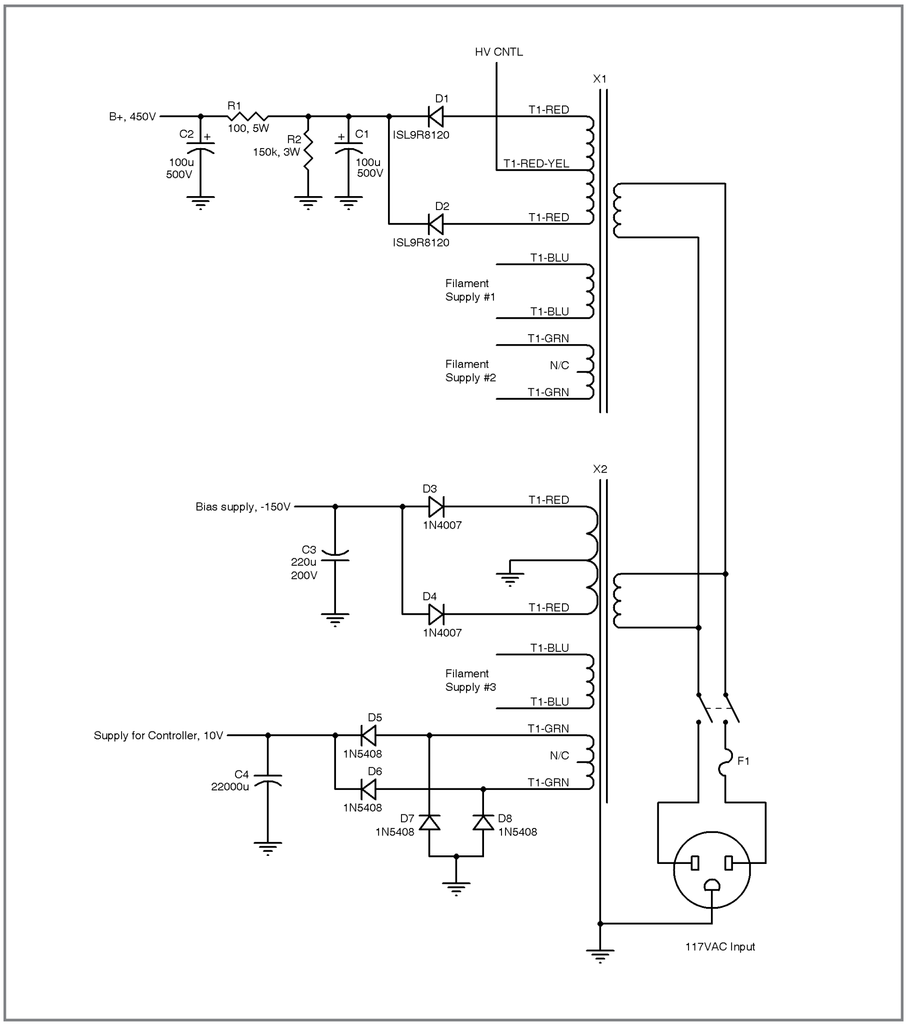


Figure 2—The MiniTron uses a conventional linear power supply built with toroidal transformers that is housed in a die-cast aluminum box. An SMPS is planned.

The current flow through the device is large and uncontrolled unless a negative "bias" voltage is applied to the control grid. The application of plate voltage without "negative bias" voltage results in a large, often destructive, current flow.

Keep this basic information about vacuum tubes in mind as you read the rest of this article. Now, back to my project.

## POWER SUPPLY

The amplifier's power supply is a simple unregulated design (see Figure 2). It uses conventional rectification and filtering to provide voltage sources for the design. The filament supplies are 6.3-V AC connected directly to the tubes. The 10-V supply feeds a 5-V regulator and a MOSFET driver on the controller board. The –150-V supply is connected to the bias generator on the controller board. The high voltage is routed directly to the amplifier board to supply the first two stages and to the agile buck converter on the controller board.

## AMPLIFIER

The amplifier is a class H design using modulated supply rails. The audio path is purely vacuum tube (no feedback paths exist), and the circuitry is a single-ended class A design (current flows for the entire audio cycle).

Refer to Figure 3. Each vacuum tube section is represented by a circle with three elements inside it. The cathode is the element across the bottom. It functions like the source in a MOSFET. The cathode has a heating element. It is often shown separately from the device

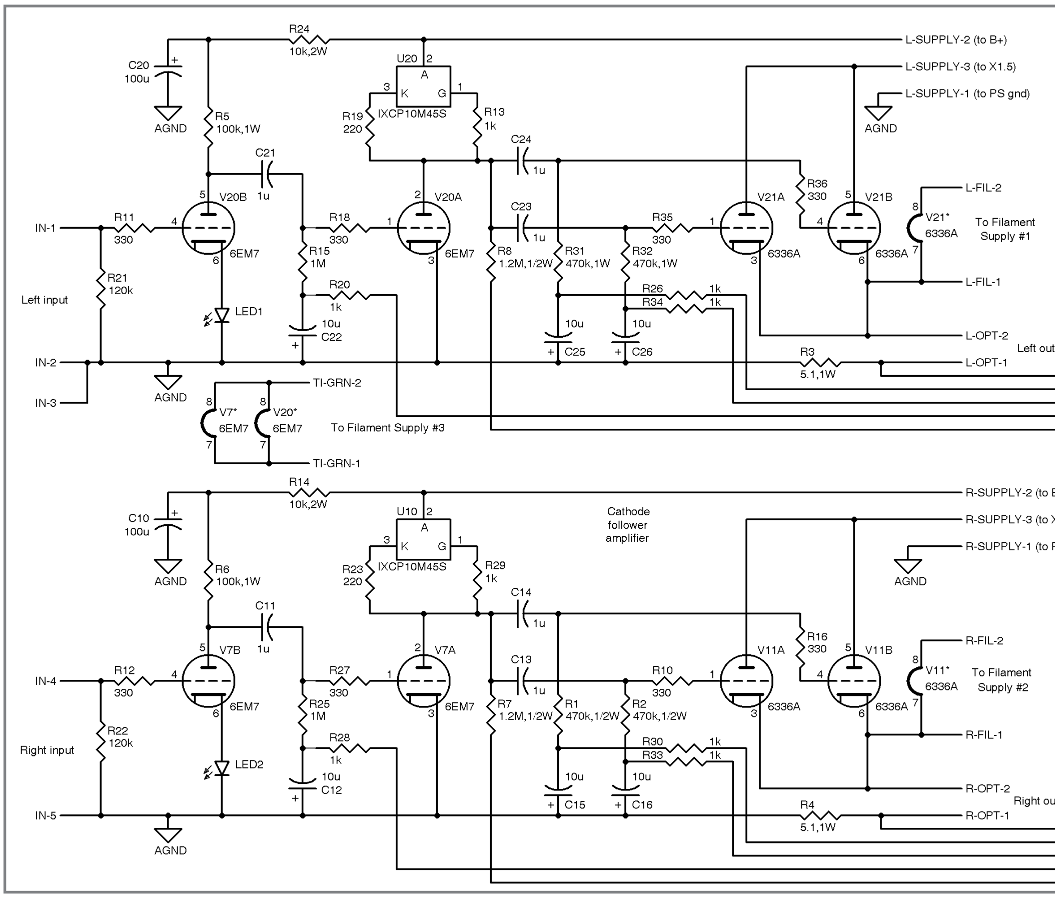


**Figure 3**—The amplifier is a class H design with modulated supply rails. The audio path is purely vacuum tube, no feedback paths exist, and the circuitry is a single-ended class A design (current flows for the entire audio cycle). There are three gain stages.

WIZnet
Enjoy your Creativity at iMCU Design Contest on WIZwiki.net
* Nov. 2009 - Jun. 2010
* Sponsored by Circuit Cellar
* W7100 is an Internet MCU integrating HW TCP/IP core with 8051 processor
WIZnet iMCU W7100 P4FW0-162 0903

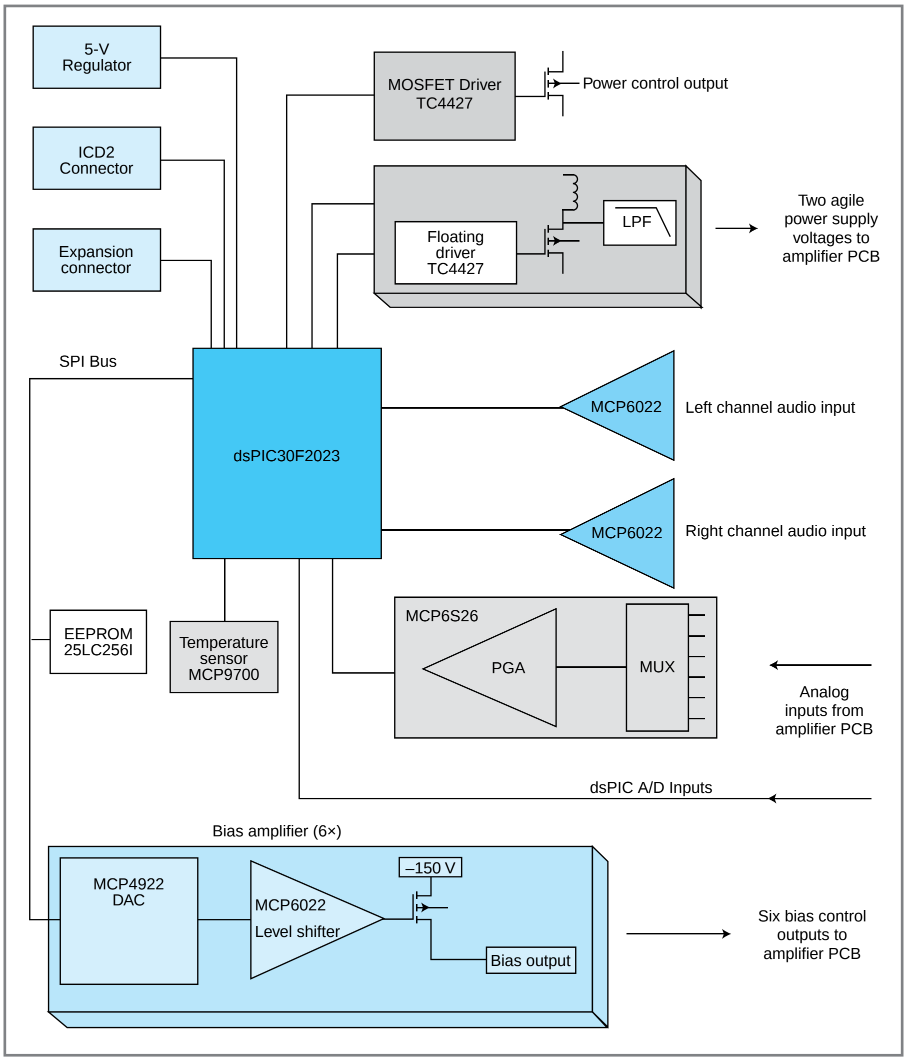


Figure 4—These are the individual controller elements.

itself in a manner like the power and ground pins on digital logic ICs. It is connected across a power supply for "heater power" or "filament power." The grid is the middle element represented as a series of dashes (usually three or four), functioning like the gate in a MOSFET. The plate is the upper element corresponding to the drain in a MOSFET. There are only electron tubes (N channel devices). There are no positron tubes (P channel devices). Many tubes have two or more sections in a common glass envelope. They are shown as individual tubes on the schematic in a manner similar to a quad NAND gate. Sometimes the sections are identical (the output tubes in this design); sometimes they are not (the input and driver sections). It isn't always possible to determine this from the schematic alone.

There are three gain stages. The first two are conventional, using the two sections of a dual vacuum tube triode. Both stages are common cathode (like common emitter) voltage amplifiers. The first stage uses an LED in series with the tube's cathode. It functions as a low-voltage (1.8-V) Zener diode, fixing the bias of the stage. The second stage uses a constant current source (CCS) IC as the plate load. Both of these "modern tricks" serve to lower the distortion of the amplifier. These two stages provide all of the voltage gain. The third stage is rather unique. It is a cathode follower that functions in a manner similar to a MOSFET source follower. It provides no voltage gain (a small loss), but it provides a large current gain. Cathode follower output

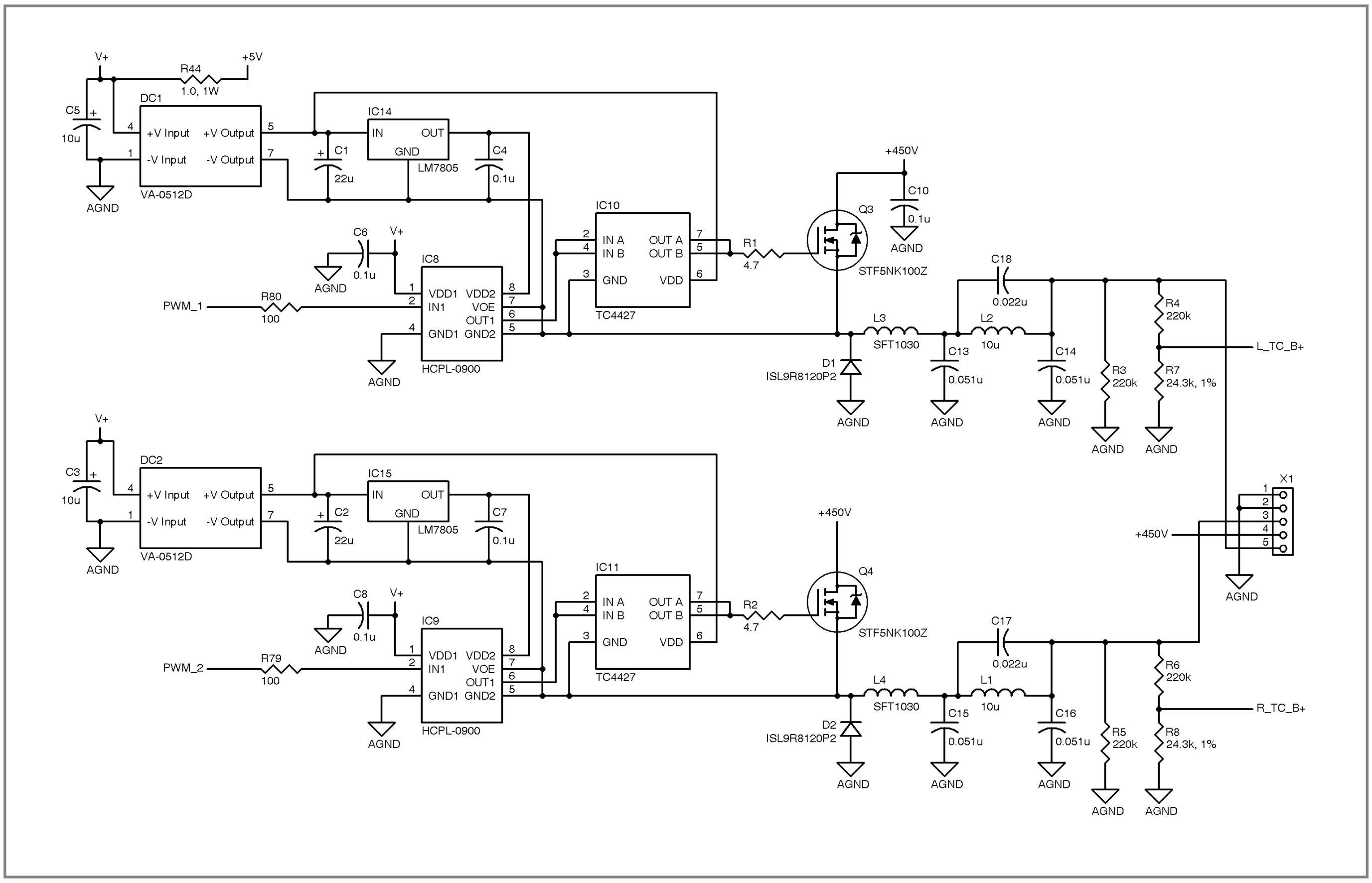


Figure 5—The floating buck converters allow a low-voltage output from the dsPIC to control a floating high-voltage supply. There is one for each stereo channel. It can't get much simpler than this.

stages are rare in audio amplifiers because of the extreme drive voltage requirements. The tube is a dual triode with identical sections. Both sections are wired in parallel, except for the provision to adjust the bias independently for each section. These are both adjusted by the controller. A cathode follower typically has a high PSRR so that the power supply can vary without affecting the output, as long as there is sufficient headroom for the signal being processed. It is desirable to operate the third stage with a constant voltage across it, having the tube's current varied by the signal. Since the cathode of the tube is the output and the plate is the voltage supply pin, the plate voltage must be varied in step with the signal voltage. This is the function of the agile switch-mode power supply (SMPS) converter.

The amplifier has two identical channels. Each channel has three bias voltage inputs to control the tube parameters. There are current-sense resistors in series with the output transformers and a resistive tap from the driver tube. These are used to measure the tube's plate voltage and sample the audio for the agile SMPS. There is also a temperature sensor mounted in the area of highest heat found on the first prototype PCB. These signals are routed via a ribbon cable to the controller PC board.

## THE CONTROLLER

Figure 4 is a simple depiction of the controller. Figure 5 through Figure 8 detail the aspects of the controller's circuitry: floating buck converters, the analog I/O, the bias generator, and the connections to the microprocessor.

There are three dsPIC ICs from the Microchip Technology "30F" family that are designed for use in intelligent switch-mode power supplies. I used the dsPIC30F2023, as you can see in Figure 8. The unique SMPS resources in these ICs do most of the hard stuff usually required for an SMPS design. This leaves ample processing power available for other activities and opens the door for all sorts of new digital implementations of power control circuitry that was previously an analog-only domain. In fact, I have the amplifier running with no code at all in the main loop of the software. The interrupt service routine can run the ADC and the SMPS by itself. The simple software is based on an example on Microchip's web site, although they probably never intended for such an application. This project design has the hardware capabilities for full closed-loop converter operation; however, closed-loop operation has not been tested. Closed-loop operation should not have any advantages in this application.

In addition to the agile SMPS, the hardware is designed to allow development of advanced control features of the vacuum tube amplifier. This platform is currently being used to develop algorithms and software for use in future digitally controlled vacuum tube amplifier products.

## AGILE SMPS CONVERTER

The controller's main function is the agile SMPS. It is the project's "enabling technology" and reason for its existence. The agile SMPS is used to vary the supply voltage on the output tube in track with the audio signal, thereby keeping the voltage across the output tube constant. The plate voltage relative to the cathode stays at a low value (100 V in this design). The absolute plate voltage is constantly moving with the audio signal, swinging from 420 to –250 V (actual measured values). The plate voltage in a conventional design would need to be fixed at 420 V to afford the same headroom and power output capabilities. From this it can be seen that the power dissipated in the output tube is reduced by a factor of about four. The power dissipation is usually the limiting factor in a vacuum tube amplifier, so these techniques can be used to increase the power output for a given amplifier size and cost.

Audio is tapped from the second stage in the tube amplifier and routed to an op-amp for easy gain and offset adjustments in the early development cycle. These can be done in software, so the op-amp will likely be eliminated in later designs. The op-amp gain and level shifters are shown in Figure 4 and located in the center of the analog I/O schematic (see Figure 6). The DC offset potentiometer controls the DC output voltage from the agile converter, which sets the tube voltage. The op-amps feed an A/D pair on the dsPIC30F2023 chip. The ADCs are triggered by the PWM module, and an interrupt service routine simply takes the A/D values and transfers them to the PWM modules duty cycle registers. Math such as scaling or offset can be added here.

Two PWM outputs control two unique floating MOSFET driver circuits that are each capable of driving a high-side MOSFET switch operating at any voltage from 2,500 to –2,500 V. You can see this in Figure 4 and the floating buck converter schematic (see Figure 5).

I determined early in the design cycle that the converter's output voltage could be "pulled" below ground potential by the large reactive load presented by the output transformer and loudspeaker in the vacuum tube amplifier circuit. The source of the switch FET operates in the range of 500 to –400 V. A typical high-side driver IC, such as an International

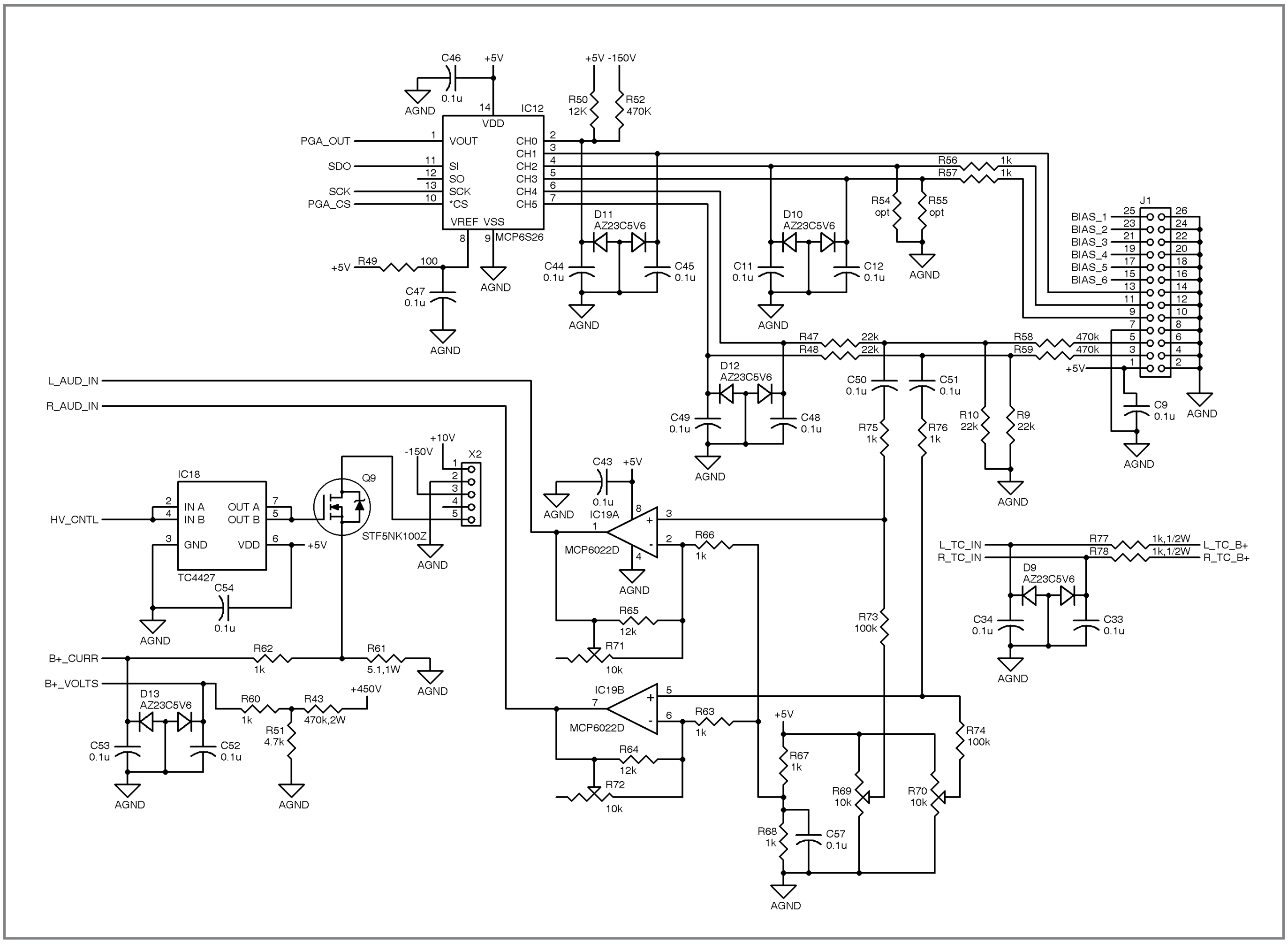


Figure 6—This is the analog I/O. IC12 is used to multiplex one A/D input for several voltage readings. IC19 provides buffered audio. IC18 is a switch for enabling the high voltage.

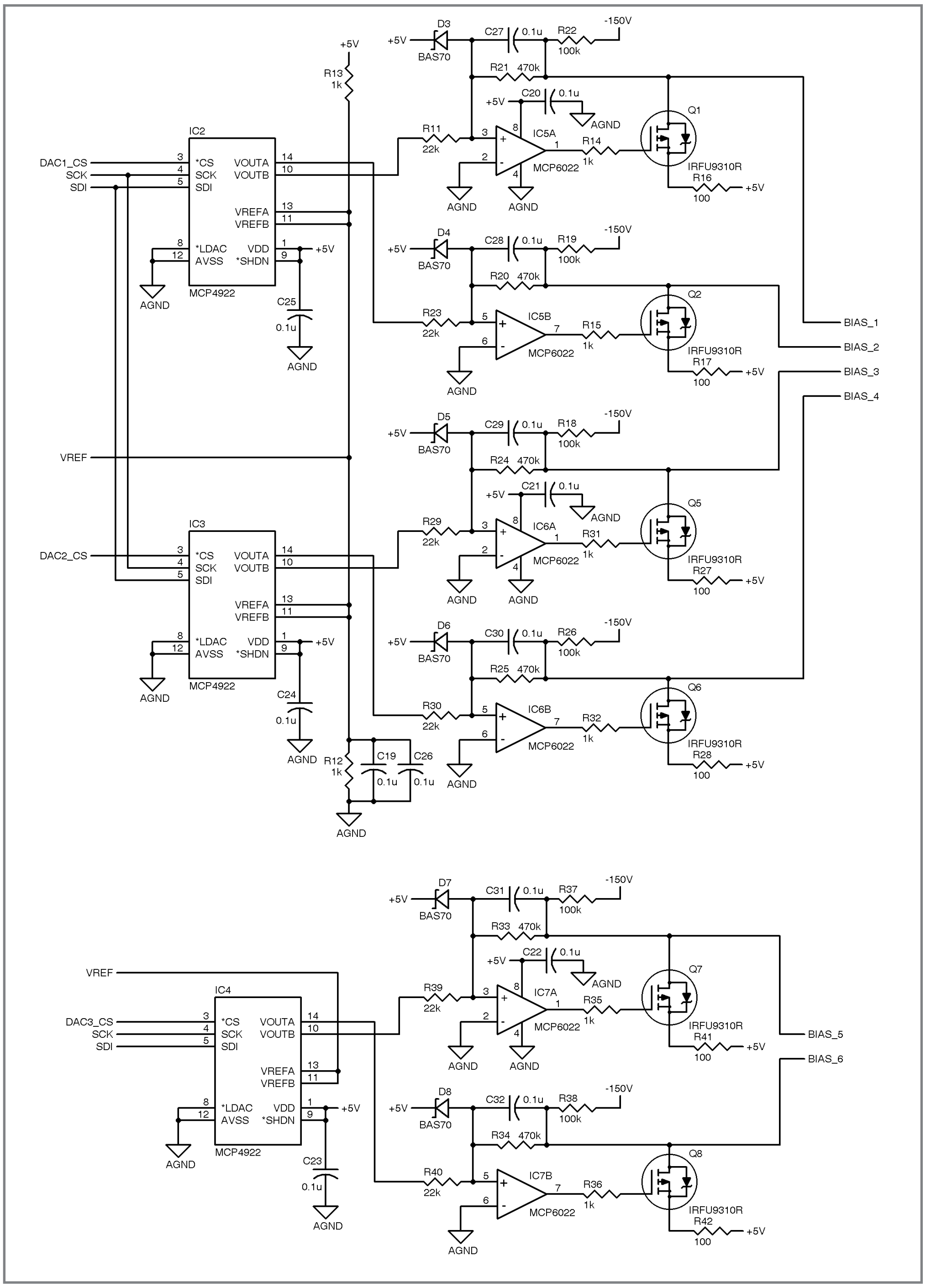


Figure 7—There are six identical bias generators. These are negative voltage supplies under DAC control. The current drain from each supply is only a few microamps.

the magnetic isolator and the DC-DC converter enables the floating supply to experience 1,000-V swings at a 1-MHz rate without issue. This enables a common low-side MOSFET driver to drive an N-channel MOSFET in a high-side switch application. The agile SMPS output is ground referenced in this application, but it does not need to be. The driver can operate in a totally floating arrangement if the circuit design dictates it.

The rest of the SMPS converter is textbook buck converter stuff, but the requirements are a bit unique. The output voltage must operate over a wide range. Operation into the usual resistive test load differs from the operation in the amplifier due to the aforementioned reactive load. The inductor value (L3) was determined empirically. The output capacitor (C13) was also determined through iterative testing, as was the output filter network (L2, C18, C14). The filter must allow the converter to be modulated at an audio rate. This modulation should be flat up to 20 kHz with minimal phase shift. The filter should attenuate the switch frequency by at least 35 dB to avoid intermodulation effects in the amplifier. There is a resistive voltage divider at each output. This is routed to an on chip A/D input pair to enable closed-loop operation if desired.

Rectifier IR2213, will fail catastrophically when its output goes negative. I devised the floating driver circuit for this situation.

An isolated DC-to-DC converter module is used to generate a floating 12-V supply. It has a 3,000-V isolation rating. This floating 12-V supply powers a Microchip Technology TC4427A low-side MOSFET driver IC. The ground-referenced PWM signal from the dsPIC30F2023 is applied to the input side of an Avago magnetic isolator IC, which has separate input and output circuits and carries a 2,500-V isolation rating. The input side is powered from the main 5-V supply. The output side is powered by a 7805-type regulator off of the floating 12-V supply. The 12-V supply floats at the source potential of the switch FET. The low internal capacitance of

## AMP SUPERVISION & CONTROL

There is ample processing power available for amplifier monitoring and control. There is hardware designed into this amplifier for the development of supervisory and control features. A MOSFET and driver circuit is included to control the

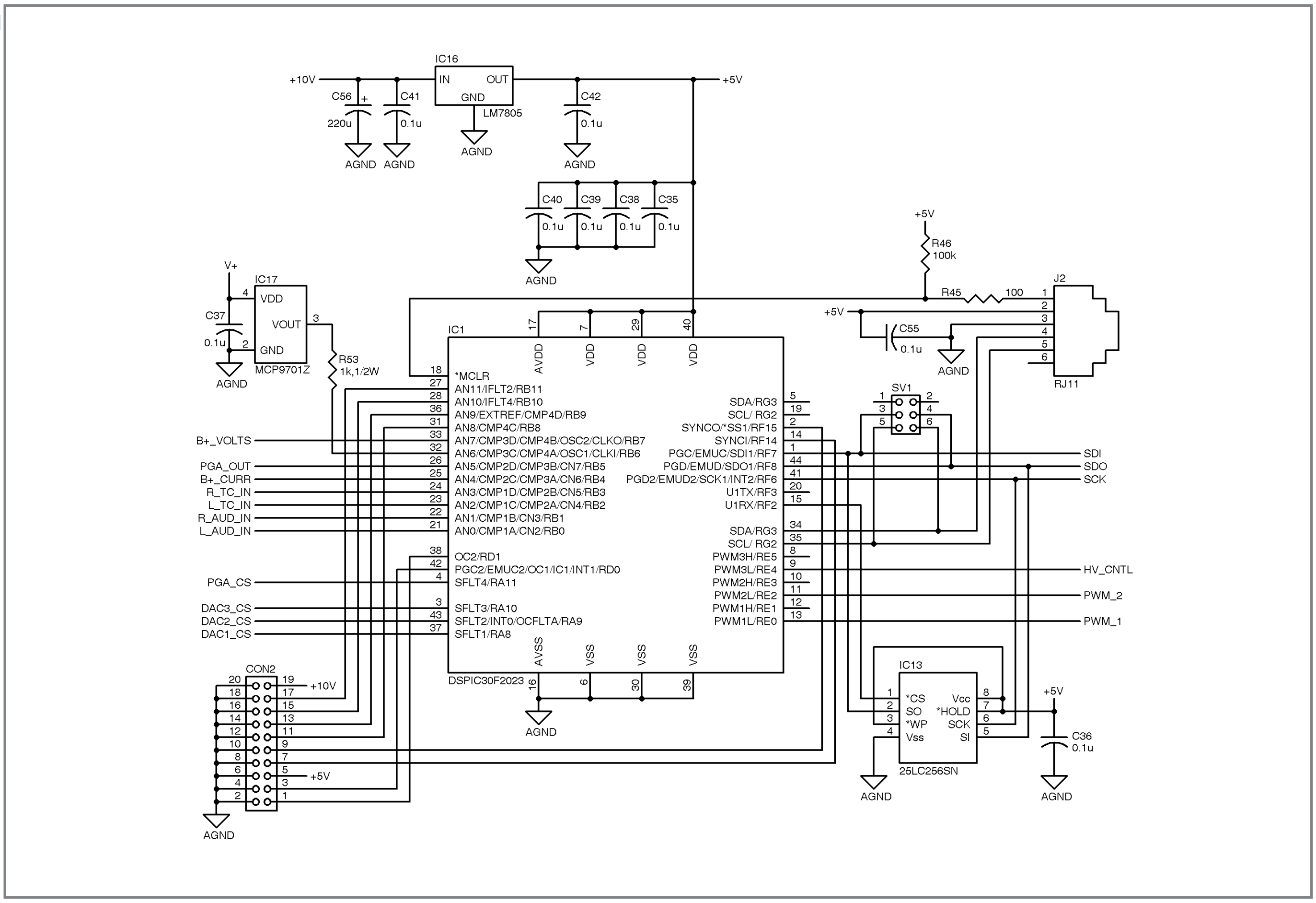


Figure 8—The circuitry includes a dsPIC30F2023, EEPROM, voltage regulator, and temperature sensor. A jack for the ICD2 is included for program development and testing.

high-voltage supply. It can be used to enable or disable the HV, or ramp it up slowly using PWM if desired. This feature has not been tested or connected yet. This is shown in Figure 4 and in the lower-left portion of the analog I/O schematic (see Figure 6).

Tubes require a negative bias voltage to control the current flow. The negative bias voltages are derived from the –150 $V_{SOURCE}$ in the bias generator circuit shown in Figure 4 and in the bias generator schematic (see Figure 7). Each bias voltage is controlled from a D/A output, which is level-shifted from 0 to 5 V to 0 to –150 V using an op-amp and a MOSFET. In the current software, the bias voltages are hard coded to a predetermined value.

The unused A/D inputs monitor several operating voltages in the amplifier. Any input that may be subjected to high voltages is protected with Zener diodes and capacitors. Sufficient series resistance is included to limit rise times. The microprocessor does not have enough A/D inputs to monitor all of the desired signals. A Microchip Technology MCP6S26 PGA/multiplexer is used to switch between the infrequently scanned inputs under SPI control. The total power supply current signal, the SMPS temperature, and the tracking converter output voltage signals are routed to the four on-chip comparators. Any of these can be used to generate an interrupt that may be used to shut down the system if potentially damaging conditions exist. These monitoring functions are shown in Figure 4 and in the analog I/O schematic (see Figure 6).

An EEPROM is included for storing start-up parameters, data logging, and keeping track of operating points over time (see Figure 8). An expansion connector is provided for connecting additional hardware. It is expected that this will be used for a keypad/display module.

## SOURCE CODE

I outlined a serious feature set for the amplifier design. It requires a major software development effort supported by several experiments to develop the algorithms needed for a full-featured amplifier. This effort is underway.

The code I'll present here is for a fully functioning amplifier. I have been listening to it for a few weeks and I have made basic performance measurements. All of the bias voltages were determined experimentally and hard coded as initial conditions for the appropriate variables. The gain and DC offset values for the SMPS are set via the trim potentiometers. The amplifier works very well and meets my original goals: it demonstrates enhanced performance over a conventional design, it provides a platform

ICbank
Gateway to Electronics

# 3 PORT INTERFACE RS-485 to Ethernet Converter

Only $170

RS-485 to Ethernet Converter

## Powerful feature

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

## Specification

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

ICbank
Gateway to Electronics

www.icbank.com

**Pololu**
**Robotics & Electronics**

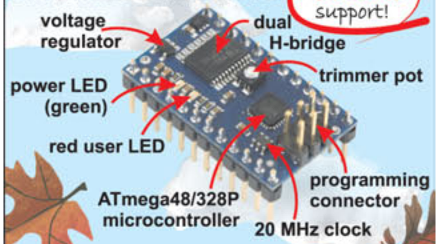
**Robot Kits**
Line followers
Robot arms
Hexapods
Chassis

3pi Robot
$99.95
High-performance, C-programmable, ATmega328P-based robot (with Arduino support)!

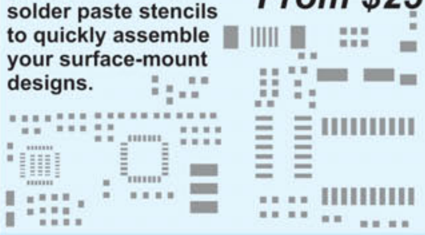
**Mechanical Components**
Motors, servos
Wheels, ball casters

**Motion Control**
Motor controllers
Servo controllers
Now with USB!

**Robot Controllers**
with Arduino support!



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



**Custom Laser Cutting**
From $25

Cut your own custom chassis, front panels, and more!

**1-877-7-POLOLU**
**www.pololu.com**
6000 S. Eastern Ave. 12D, Las Vegas, NV 89119

for feature development, and it enables hardware improvement.

Note that the main loop for the code is completely empty at this time. Once set up, the SMPS and ADC modules that are critical for the amplifier's function are interrupt-driven. There is ample room for feature development as long as the performance of the ISR is not impacted.

## PC BOARDS

I designed two unique PC boards for this project. I will design a third for the SMPS to replace the current linear power supply. The linear supply is built on perf board, so it doesn't use a true PC board.

The density of the amplifier PC board is very low, there are few through holes, and the trace size is large. The PC board is well within the capabilities of home fabrication, so I made one myself.

The controller PC board has two main sections (which I laid out independently). The high-voltage section (the right side of board) operates with voltages approaching 500 V. The HV section is kept completely isolated from the logic section. It has a "moat" of ground around it, and all connections into this section are made with leaded resistors, except ground. A mixture of leaded and surface-mount components is used in the HV section. In many cases, surface-mount components do not have the required voltage or current ratings. The logic section of the PC board is pretty conventional. Surface-mount components are used in most cases. This board is beyond the capabilities of home fabrication, so I sent it to a quick-turn PC board house.

## CHASSIS/CABINET

This project is intended to be a development platform, but I also intend to bring it to listening sessions and possibly an audio show or two. I needed a functional chassis to support the circuitry and protect onlookers from the deadly voltage, but still allow easy access to the electronics for development purposes. I also must be able to replace or upgrade the components or PC boards as required. Plus, I can "show off" all of the unique circuitry, not hide it under a chassis.

I mounted all of the components on top of a piece of aluminum diamond plate that slides into a groove in a wood box. The rear panel of the box is removable to allow chassis plate removal. I cut another groove for a piece of clear Lexan 1.5" above the chassis plate. This allows the PC boards to remain covered yet visible. I can remove the Lexan for experiments.

## PERFORMANCE TESTING

As I write this article, the amplifier project has been operational for about a month. I haven't had the time to tweak the design or develop any new software. I made some quick measurements, but detailed testing must wait for now.

Maximum power output at 5% distortion is 38.3 W. The power at 1% distortion is 24.6 W. Distortion at 10 W is 0.33%. In contrast, similar measurements on a conventional amplifier (one of my designs) of similar size and weight generated different results. Maximum power output at 5% distortion is 9 W. The power at 1% distortion is 4.7 W. Distortion at 1 W is 0.78%. Clearly, these techniques allow for more undistorted power output and far lower distortion at the power levels found in most listening.

Does this project meet my original design goals? Yes, and it does so in a big way.

I wanted a development platform for writing code. I got it. I was expecting to double the power output of a conventional amplifier of equal size. I found much more. This technology can be applied conservatively to generate triple the power output. I have not made any efficiency measurements yet, but it looks like similar sized gains have occurred. The distortion levels have dropped to levels not usually seen in vacuum tube amplifiers, especially in the 0 to 2 W range, where most listening takes place. Many vacuum tube proponents claim that it is this distortion that gives a vacuum tube amplifier its special "tube sound." I've listened to this amplifier for a total of about 10 hours, and I can say that the "tube sound" is still in there!

At first, you might think this design is not cost-effective because the cost adder for the agile SMPS/controller is about $150. But when you step back and realize that it can be used to double or triple an amplifier's power output and figure out what it would take to double the power output by traditional means, you realize that adding the controller is an overall cost savings.

This project is a high-end stereo amplifier. Most people have some sort of music reproduction or home theater equipment. Few people have (or even want) high-end equipment due to its high cost. This project is one step in direction of reducing the cost of high-end equipment. You can apply this technology to solid-state amplifiers and vacuum tube amplifiers for the musical instrument market.

## WHAT'S NEXT?

My amplifier is back on the workbench. I'm using this project to develop the software for many of the features I outlined earlier in this article. There will be the inevitable hardware upgrades as well because Microchip has released a second generation of dsPIC processors made specifically for SMPSs. There are also some new ICs from Analog Devices that reduce the floating MOSFET driver to a single chip.

I plan to continue taking this design down the road toward possible production. I will likely do this in stages, with the SMPS to replace the analog power supply coming first. I may also develop a vacuum tube guitar amplifier using the same technology. These techniques can go a long way to improve the reliability of vacuum tube guitar amplifiers while enabling all sorts of new and unique distortion profiles. ■

*George Anderson (tech@tubelab.com) worked at Motorola for 36 years as an RF engineer focused on cell phones and two-way radios. He holds a BS in computer engineering from Nova Southeastern University (graduated at age 40) and an MSEE from Florida Atlantic University (graduated at age 46). George's technical interests range from ham radio to various microprocessor-related technologies.*

## PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/231.

## SOURCES

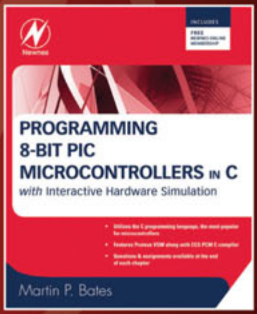
**dsPIC30F2023 DSC, MCP6S26 PGA/multiplexer, and TC4427A MOSFET driver IC**
Microchip Technology, Inc. | www.microchip.com

N e w n e s P r e s s

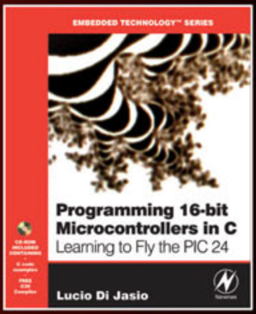
Got a PIC Question? We have the answers!

Embedded Technology™ Series
Programming 32-bit Microcontrollers in C
Exploring the PIC32
Lucio Di Jasio

By Lucio Di Jasio
ISBN: 9780750687096
$59.95


Programming 8-bit PIC Microcontrollers in C
with Interactive Hardware Simulation
Martin P. Bates

By Martin Bates
ISBN: 9780750689601
$39.95


Embedded Technology™ Series
Programming 16-bit Microcontrollers in C
Learning to Fly the PIC 24
Lucio Di Jasio

By Lucio Di Jasio
ISBN: 9780750682923
$49.95

Advanced PIC Microcontroller Projects in C
From USB to RTOS with the PIC18F Series
Dogan Ibrahim

By Dogan Ibrahim
ISBN: 9780750686112
$39.95

Look for the latest titles from Newnes Press to help you maintain your competitive edge at Amazon.com or your favorite online retailer.

Newnes

www.newnespress.com

Register for our e-news at newnespress.com
Receive our best discounts · Hear about books before they publish
Access to free sample chapters, video tutorials and more!

by Robert Lacoste

# Multirate Techniques and CIC Filters

Cascaded integrator-comb (CIC) filters are specific forms of moving average filters that enable you to tackle difficult signal-processing problems. With these filters and some helpful multirate signal-processing techniques, you have another option when a FIR filter doesn't fit the bill.

Welcome back to The Darker Side. Faithful readers will remember my column on finite-impulse response (FIR) filters a couple of years ago titled "No Fear with FIR" (*Circuit Cellar* 207, 2007). In a nutshell, FIR filters enable you to implement a digital filter with virtually any response curve in a DSP, FPGA, or even microcontroller—but they do so at the expense of a significant number of arithmetic operations. For example, to implement a 32-tap FIR filter, you need to compute 32 multiplications and 31 additions per sample, which translates into a lot of MIPS if your application requires a high sampling rate.

A reader contacted me not so long after my *Circuit Cellar* 207 article appeared in print. He wanted to use a high-end DSP to filter a signal sampled at 16 Msps, and he was having some trouble implementing the required FIR filter. Basically, his input signal was a nearly 5-MHz sine wave, and he needed to analyze the signal's small perturbations around 5 MHz—say, from 4.95 to 5.05 MHz. I'm sure you could find plenty of situations like this in applications ranging from digital radio receivers to ultrasonic systems. The signal was 5 MHz, so the choice of a 16-Msps sampling frequency made sense. As you know, in order to avoid any nasty aliasing, the minimal sampling frequency should be at least twice the highest frequency present in the signal. So, in this instance, 2 × 5 MHz was 10 Msps. Using 16 Msps allowed the reader to implement a reasonable antialiasing low-pass filter with a low attenuation at 5 MHz and a high attenuation at 10 MHz (i.e., 20/2). Then he needed to implement a digital filter to isolate the frequencies of interest from background noise, meaning a band-pass filter centered at 5 MHz with a 100 kHz or so band-pass. Using a FIR filter would require at least a 256-tap filter to get this resolution (see my previous article), and this translates into 256 × 2 operations executed 16 million times per second, giving an 8,192 MIPS requirement. I wasn't surprised that the reader was having some difficulties with DSP.

What does this mean? Simply that the direct use of a FIR filter isn't a good idea for a case like this! In this article, I'll show you how to deal with such problems. You can use multirate signal-processing techniques and the companion filters of choice: cascaded integrator-comb (CIC) filters.

## MULTIRATE?

Real-life signals are full of information. However, only a small part of this information is usually useful, and life is far easier if you succeed in reducing this flow down to the information you actually need in order to process it efficiently. This is exactly the job for a good assistant who reads the daily newspapers for you and drops a review on your

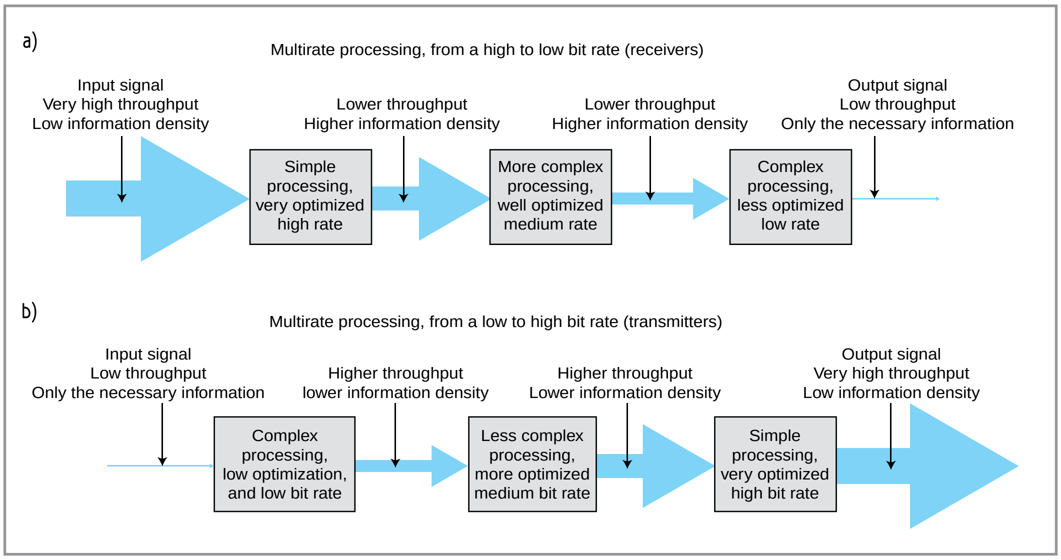

a)
Multirate processing, from a high to low bit rate (receivers)
Input signal
Very high throughput
Low information density
Lower throughput
Higher information density
Lower throughput
Higher information density
Output signal
Low throughput
Only the necessary information
Simple processing, very optimized high rate
More complex processing, well optimized medium rate
Complex processing, less optimized low rate
b)
Multirate processing, from a low to high bit rate (transmitters)
Input signal
Low throughput
Only the necessary information
Higher throughput
lower information density
Higher throughput
Lower information density
Output signal
Very high throughput
Low information density
Complex processing, low optimization, and low bit rate
Less complex processing, more optimized medium bit rate
Simple processing, very optimized high bit rate

**Figure 1a**—Multirate processing is a technique that gradually moves a high sampling rate signal down to more manageable speeds, while increasing the density of useful information. **b**—The same approach can be used in the other direction, usually to build transmitters.

desk. By the way, you should have a book on information theory on your shelf (e.g., content from Claude E. Shannon or others).

Let's look at the previous example. The 16-Msps sampling rate is dictated by the signal's 5-MHz frequency, but the actual interesting information is only included in a 100-kHz-wide channel. Thus, the application was throwing away 98% of the information. For instance, if the target is to simply send the decoded signal to a DAC, a 200-ksps sampling rate on the output side will be enough to transmit a 100-kHz-wide signal. So it is fundamentally useless to calculate a filtered value 16 million times per second because only 2% of the calculated information will be used on the output. How do you optimize such a situation? Through multirate, of course. (Just to be complete, you could use another approach, under-sampling, to achieve roughly the same results. But I won't cover that approach in this article.)

The initial signal has a high bit rate but a low proportion of useful information. Multirate processing is an algorithmic technique with which you reduce (step by step) a signal's sampling rate while keeping required information. This means it increases the density of information in a step-by-step fashion (see Figure 1a). The trick is to move (as far as possible) the algorithmic complexity down to lower sampling rates, thus reducing the overall processing requirement. Basically, the idea is to start with a high bit rate signal, apply simple algorithms, reduce the bit rate, apply slightly more complex algorithms, and so on, as long as required. Conversely, you can use the same techniques to increase (step by step) the data rate of a signal with a reduction of information density (see Figure 1b).

The aforementioned example from my reader is definitely a good candidate for multirate processing. So, let's use it as our main example here.

## DIGITAL MIXING

Let's start with a 5-MHz signal sampled at 16 Msps. You need to reduce it in the processor before executing the computing-extensive analysis algorithms. Usually, the first step is to bring the portion of interest lower in frequency but still with the 16-Msps sampling rate. In an analog radio receiver, such a frequency translation is done using a heterodyne mixer, a technique Reginald Fessenden experimented with as early as 1900. In the analog heterodyne stage, the input signal is multiplied by a sine local oscillator. A digital mixer is nothing more than the computerized implementation of the same principle (see Figure 2). Let's see how such a mixer works. Do you remember your trigonometric formulas?

$$\cos(a)\cdot\cos(b) = \frac{1}{2}\cos(a-b) + \frac{1}{2}\cos(a+b)$$

So, if you multiply each sample of a 5-MHz signal with, say, a 4.95-MHz synthesized local oscillator, you will get a difference signal—which is 5 MHz –

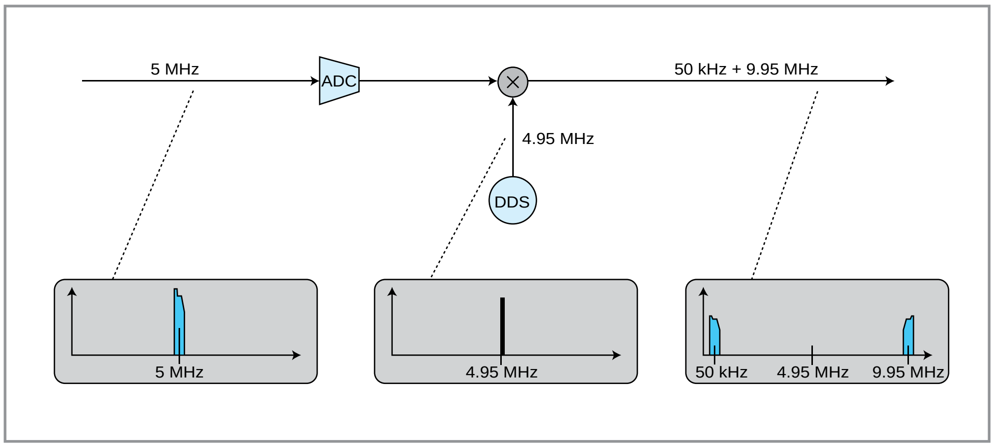


**Figure 2**—A digital mixer is nothing more than the digital equivalent of the old heterodyne system. Multiplying a signal with a locally generated carrier allows it to shift its frequency, but this adds an image frequency that must usually be filtered out.

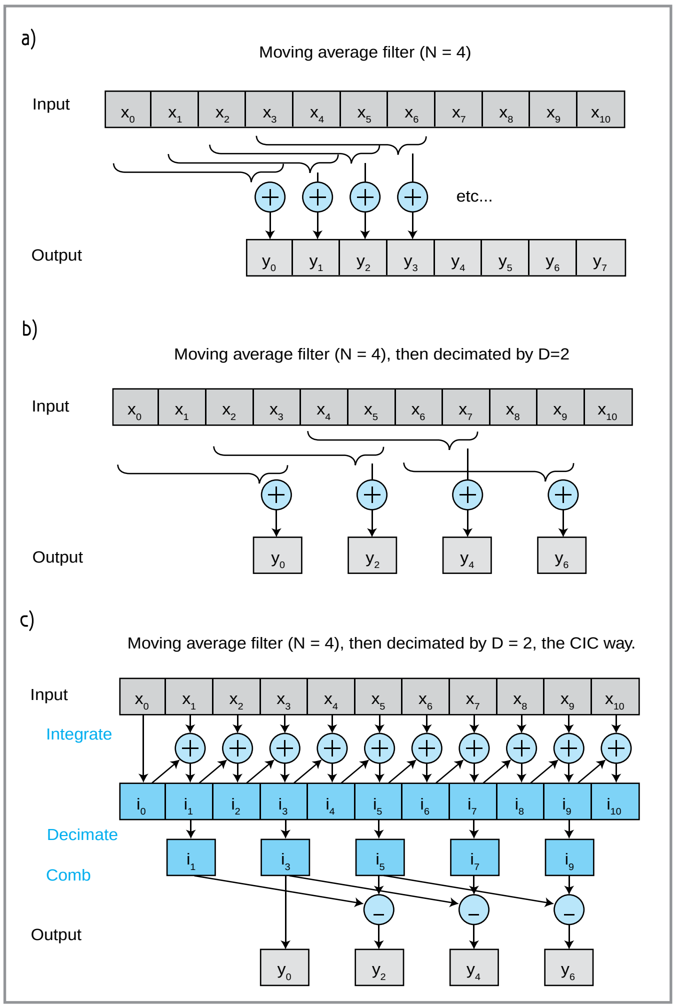

a)
Moving average filter (N = 4)
Input
$x_0$ $x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$ $x_7$ $x_8$ $x_9$ $x_{10}$
etc...
Output
$y_0$ $y_1$ $y_2$ $y_3$ $y_4$ $y_5$ $y_6$ $y_7$
b)
Moving average filter (N = 4), then decimated by D=2
Input
Output
$y_0$ $y_2$ $y_4$ $y_6$
c)
Moving average filter (N = 4), then decimated by D = 2, the CIC way.
Input
Integrate
$i_0$ $i_1$ $i_2$ $i_3$ $i_4$ $i_5$ $i_6$ $i_7$ $i_8$ $i_9$ $i_{10}$
Decimate
Comb
Output

Figure 3a—A moving average filter is not very effective in terms of calculation, even coupled with a decimation, and even if only the required averages are calculated (b). For example, here the sum $X_2$ + $X_3$ is calculated twice. The CIC algorithm (c) is simply a more optimized algorithm to calculate exactly the same values.

4.95 MHz =50 kHz (named the intermediate frequency, or IF)—and a sum signal at 5 MHz + 9.95 MHz = 9.95 MHz (the image frequency):

$$\cos(2\pi \times 5{,}000{,}000t) \cdot \cos(2\pi \times 4{,}950{,}000t) = \frac{1}{2}\cos(2\pi \times 50{,}000t) + \frac{1}{2}\cos(2\pi \times 9{,}950{,}000t)$$

Note that as this relationship is linear, the spectrum of the signal is maintained. For example, if the original signal included frequency components at 5 and 5.001 MHz, the down-converted signal will have components at 50 and 51 kHz, respectively.

In practice, such a 4.95-MHz "oscillator" can be implemented as a software-based DDS oscillator, which will just require one addition and one sine table look-up per sample. (For more information on this topic, refer to my 2008 article, "Direct Digital Synthesis 101," in *Circuit Cellar* 217.) You will then have to calculate only one multiplication at each signal sample to perform the mixing, so the process will use only a small number of CPU cycles.

I must highlight a difficulty. This simple digital mixing scheme works only if you're sure that your input signal has frequency components close to 5 MHz (more exactly, from 4.95 to 5.05 MHz). For instance, imagine you have a strong input signal of 4.90 MHz. The translated output would have a component at –50 kHz (i.e., 4.90 – 4.95). But, as cos(–x) = cos(x), it will be added to the translated 5-MHz input signal and result in a mess. In such a case, you'd need to filter the input signal prior to the ADC (to increase the intermediate frequency), or you'd have to use a more sophisticated technique like IQ mixing. I plan to present this in a future article, so stay tuned. For the moment, I will focus on a simple mixing.

## DECIMATION

You started with a 5-MHz, ±50-kHz signal and sampled it at 16 Msps with a fast ADC. You then multiplied it with a signal generated by a digital 4.95-MHz local oscillator. This translated the 5-MHz input signal down to a signal at about 50 kHz (more exactly, from 0 to 100 kHz). But you still have two issues to address. One, you have an image signal at 9.95 MHz, which inevitably will be added to your signal of interest. Two, the sampling rate is still 16 Msps.

Reducing the sampling rate is known as decimation, which is a simple process. Do you want to reduce the sampling rate from 16 Msps to 200 ksps? Since 16,000/200 is 80, keep just one sample each 80 samples and throw away the other 79! This is completely legitimate, but only if you're sure the signal can be represented reliably with a 200-ksps sampling rate, which means it shouldn't have a frequency component above 100 kHz. And this is why you have a problem with the 9.95-MHz image signal. If you just decimate the mixed signal, the output will be useless because both images will be merged together. You need first to filter out the image by implementing a low-pass digital filter.

Are you back to the original problem associated with implementing a digital filter at 16 Msps? No. The two signals have much different frequencies, so you can use a crude digital filter that will be feasible on the computational side.

## MOVING AVERAGES & CIC FILTERS

You've probably observed that averaging enables you to remove "noise" in a dataset. In other words, an average—or more specifically, a "moving average" (also called a "running average")—is a simple way to implement a low-pass filter. Just take N consecutive samples of the signal, calculate their average (or their sum, which is the same number multiplied by N), and use the result as the filter's output. Figure 3a is an example with N = 4.

A moving average doesn't require anything other than additions, so you can use such a filter for this problem. With a proper choice for N, a moving average filter will enable you to filter out the 9.95-MHz image and keep only the 50-kHz signal. Then a decimation step will be legitimate. Of course, it would be useless to calculate all the

Listing 1—This Scilab code calculates and plots the frequency response of a CIC1 filter with N = D = 8. It is simply done by calculating the FFT of an eight-point rectangular pulse.

```
// Simulation length
LENGTH=1024;

// Width of the window (in samples)
D=8;

// Generate the filter impulse response (rectangular, length D)
imp=zeros(1:LENGTH);
imp(1:D)=1/D;

// Calculate its frequency response through FFT
freqresponse=abs(fft(imp));

// Plot it, in linear and log scale !
subplot(1,2,1);
plot2d((1:LENGTH/2)/LENGTH,freqresponse(1:$/2));
xtitle('Moving average filter - D=8 (linear scale)');
subplot(1,2,2);
db=20*log10(freqresponse+1e-200);
plot2d((1:LENGTH/2)/LENGTH,db(1:$/2),rect=[0,-25,0.5,0]);
xtitle('Moving average filter - D=8 (dB)');
```

moving averages before decimation. For example, if you want to decimate by a factor of two, you just need to calculate one average every two samples (see Figure 3b). However, note that the algorithm in Figure 3b isn't very efficient because you calculate the same operation several times. For example, the first output value is $x_0 + x_1 + x_2 + x_3$ and the second output value is $x_2 + x_3 + x_4 + x_5$. You've calculated two times the addition $x_2 + x_3$. The situation will be worse if the average is calculated on a larger number of points, which is usually the case. So, if you needed to use an average of N = 256 points with a decimation factor of D = 8, you would calculate 32 times each addition. There should be a more efficient way, right? Yes, at least if you restrict the size N of the average to be a multiple of the decimation factor D. That isn't a big concession.

This brings you to the so-called CIC filter (see Figure 3c). The most basic form of a CIC filter—a one-stage CIC (i.e., CIC1)—gives the same result as a moving average filter, but it is far more efficient on the computational side. The idea is simple. Rather than calculate the sum of the samples $x_{101}$ to $x_{110}$, why not calculate the sum of the samples $x_0$ to $x_{110}$ and subtract the sum of the samples $x_0$ to $x_{100}$? The result would be exactly the same. This may seem silly because you wouldn't calculate it that way by hand, but it's a good idea for a computer. As you can see in Figure 3c, the optimized algorithm is a three-step process.

First, calculate the sum of all the previous samples. This is easy. Just add each new sample to an accumulator. This operation is mathematically an integral. Then you can perform the decimation: do nothing except at each D^th^ sample, for which an output value must be calculated. And then you can calculate this output value as the difference between the current accumulator and one of its previous values—more exactly, the previous one if N = D and the one before if N = 2D. In signal processing, the addition or subtraction of two values shifted in time is called a comb. Now you know why the overall filter is called a "cascaded integrator-comb filter."

## CIC FILTER

You can efficiently implement a CIC only if the size N of the averaging window is a multiple of the decimation factor D. If N = 4D, then the filter's comb section will need to implement a four-word first-in/first-out buffer to store the accumulator's last four values. That's why CIC filters are usually implemented with N = D—the same averaging factor as the decimation

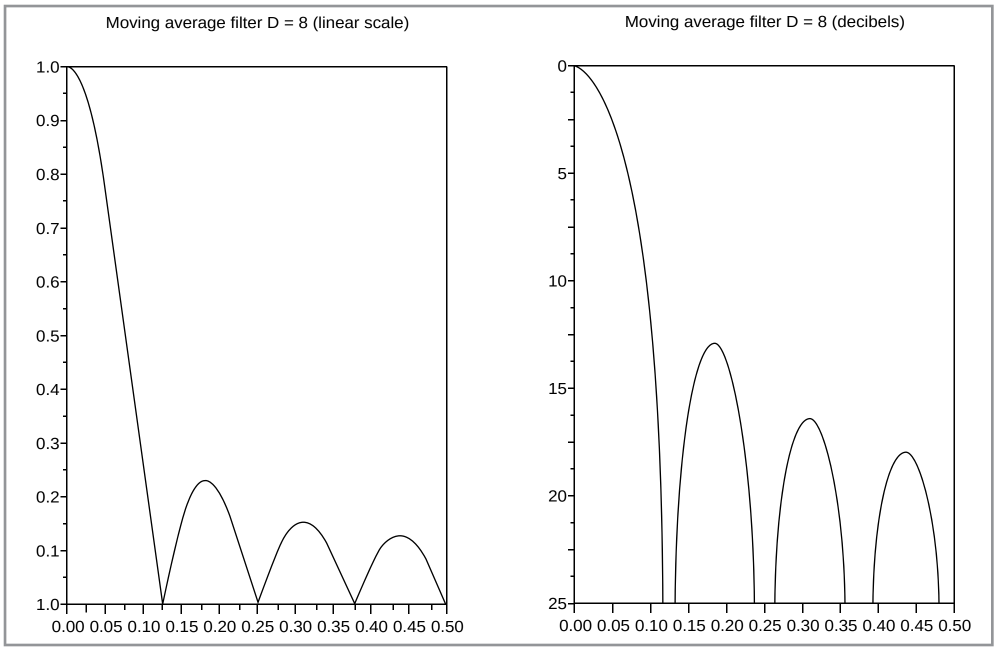


Figure 4—Here is the frequency response of a CIC1 filter with an averaging window of eight samples and a decimation factor of eight, both with linear scale (left) and logarithmic scale (right). Mathematically speaking, this is a sin(x)/x shape, with a main low-pass lobe but high residual side lobes, up to –13 dB for the first one.

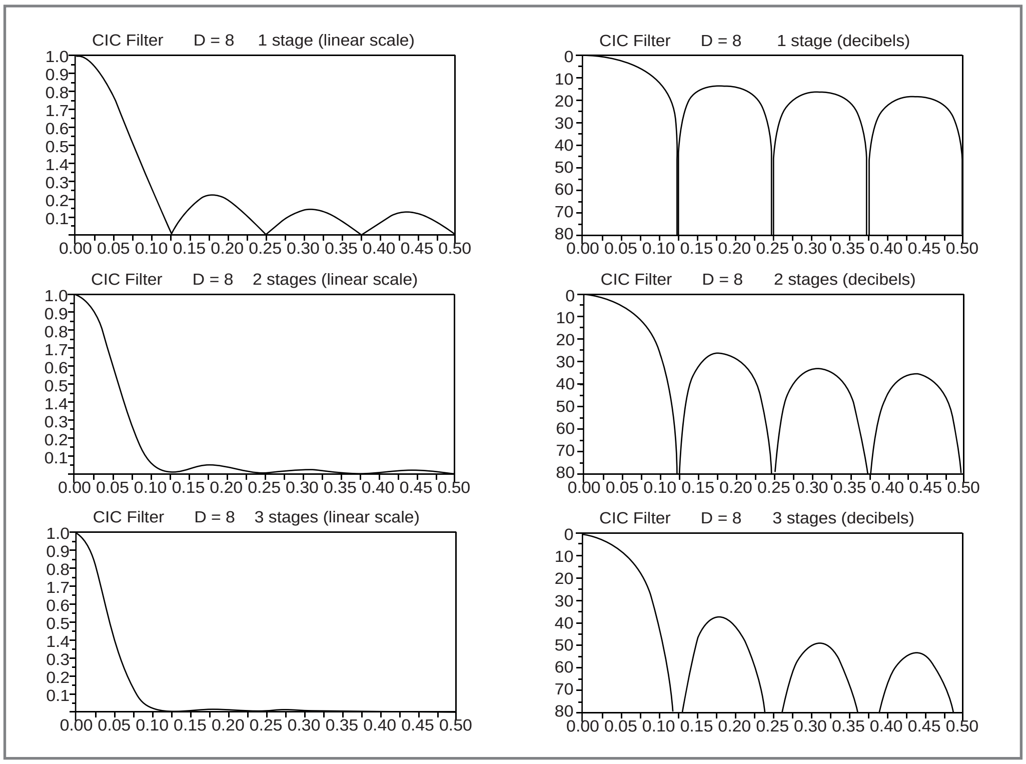


Figure 5—Combining several identical CIC stages reduces the side lobes, at the cost of a slightly higher processing expense. This is a comparison of CIC1 (top), CIC2 (middle), and CIC3 (bottom).

factor, which enables you to avoid any FIFO buffer—or N = 2D or 3D, but not more. For simplicity, I'll focus on the simpler case of N = D.

What is the frequency-domain characteristic of such a one-stage CIC filter, meaning its attenuation at a given frequency? The same as the characteristic of a moving average filter because they are just two implementations of the same filter. But what is it? You have three ways to find it: intuitively, via a simulation, or mathematically.

Let's start with the math. Do you remember my article on FIR filters? A moving average filter is in fact nothing more than a special FIR filter with all coefficients set to one. Yep. So you will remember that its frequency response is the Fourier transform of a rectangular window, which is a common function in signal processing named sinc(x) and equal to sin(x)/x. Do you want to see its shape? I wrote a short Scilab script to plot the frequency response of a D = N = 8 CIC filter. The code is straightforward. I'm sure you will understand it, even if you don't have experience with Scilab or Mathlab (see Listing 1). Download Scilab on your PC, run this script, and you will get the plot shown in Figure 4.

Let's move on to the topic of intuitive analysis. There is a first lobe with a low-pass shape as planned. The gain is lower and lower when the frequency increases up to a point where it is null. For D = 8, this point is at a frequency F = 1/8 = 0,125 times the sampling rate. Think twice. This is normal. If the input signal has such a frequency, one of its periods will be exactly as long as the averaging window, so the average will be null. However, as the frequency continues to increase, the filter response rises again, with a second lobe

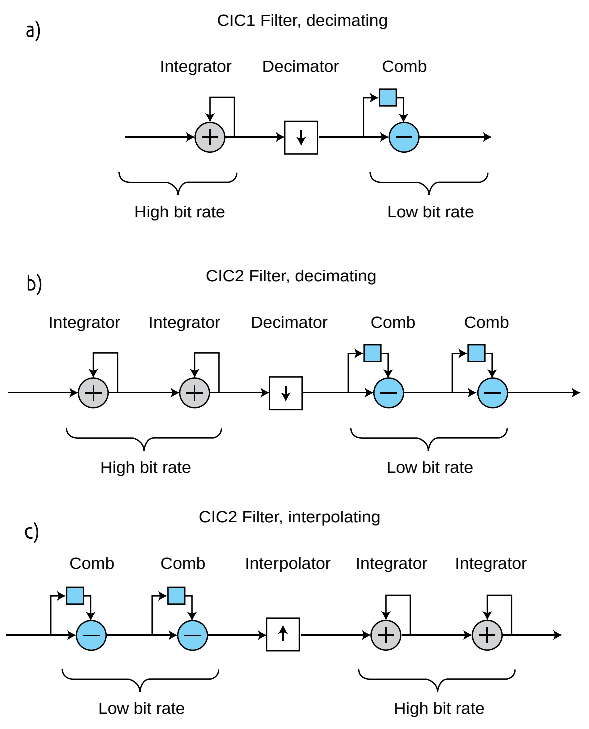


Figure 6a—A CIC1 filter is nothing more than an integrator, a decimator, and a comb, which is a simple subtraction. b—A CIC2 filter can be built efficiently by grouping the two integrators and the two combs together. c—A CIC filter can be used to increase the bit rate just by exchanging the integrator and comb sections and replacing the decimator by an interpolator.

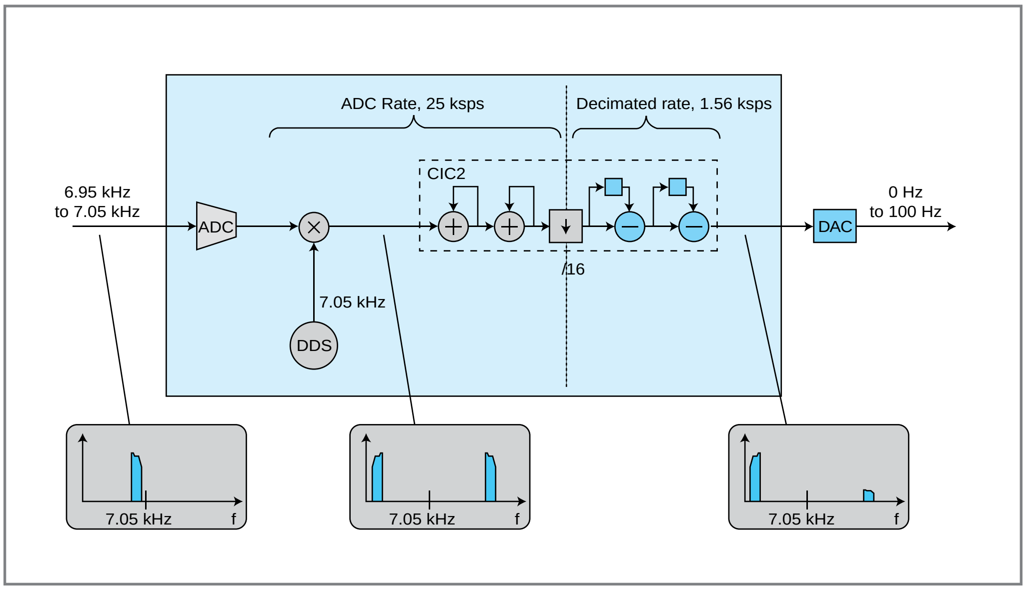


Figure 7—This is the architecture for the example I implemented on a PIC24FJ64GA002 microcontroller. From left to right, you see a digital mixer and a CIC2 low-pass filter. This design digitizes a 7-kHz signal and zooms out its frequencies from 6,950 to 7,050 Hz.

And, of course, nothing prevents you from building CIC3, CIC4, or more complex filters. Just be sure the accumulators are wide enough, because the numbers can get big.

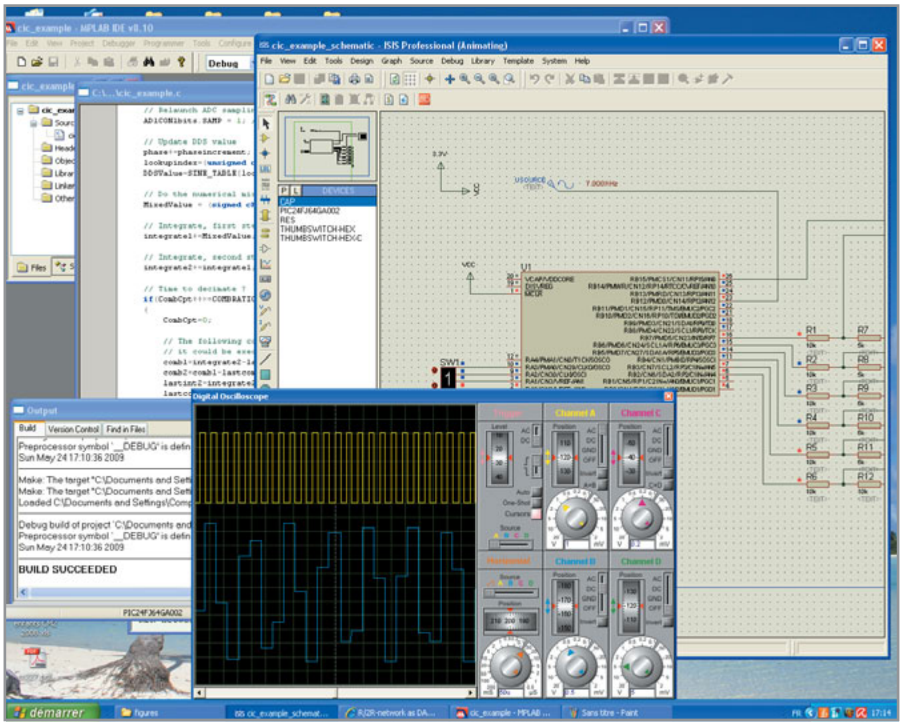
You can also use CIC filters in the other direction to increase a sampling rate. You just have to inverse the operations: comb first, interpolate, and then integrate (see Figure 6c). An interpolator in this context is nothing more than a latch, copying several times the same value on the output. You can check it. With such a CIC filter implemented, you will get a linear interpolation between each input sample.

that has a minimum attenuation of 13 dB. This patter n repeats, with nulls at frequencies that are multiples of the first one. This is the characteristic of a moving average filter. Maximal attenuation occurs each time the input signal's period is a sub-multiple of the averaging duration.

Such a filter may be enough for your application, but what can you do if the attenuation is not as steep as you want? One solution is to increase the size of the averaging, either by increasing the decimation factor D or by implementing a FIFO with N = 2D or 3D. But this is not always possible because it would alter the bandwidth. Another solution is to cascade several identical filters: their attenuation will be added, thus decreasing the importance of the side lobes (see Figure 5).

In practice, there is an efficient way to cascade several CIC filter stages (see Figure 6). For example, let's say you want to cascade two CIC filters with the same parameters N and D just to double the attenuation in the stop band. You then have the following steps: integrate, decimate and comb, and then integrate, decimate, and comb. These operations are linear, so you'll get the same result even if you shuffle them around. An efficient order is the following: integrate, integrate, decimate, comb, comb. This is the usual implementation of two-stage CIC2 filters (see Figure 6b).

The pass-band of a CIC filter is all but flat. The attenuation is 0 dB only at DC. It then increases steadily up to 3 dB regularly, not abruptly, as you would expect with a good low-pass filter. This is the price to pay for efficiency . However, nothing prevents you from cor recting this behavior with computations. A usual implementation is to add (after decimation) a FIR filter with a response cur ve precisely calculated to compensate for the behavior of the CIC filter in



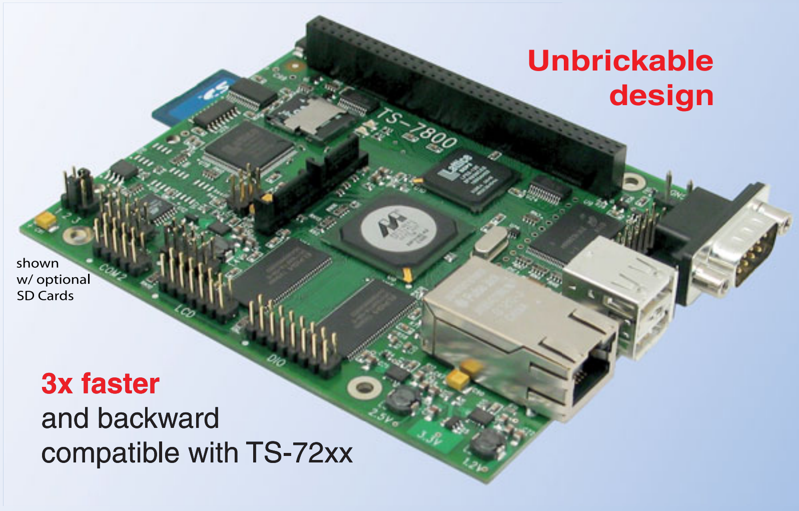
```
// Relaunch ADC samplin
AD1CON1bits.SAMP = 1;

// Update DDS value
phase+=phaseincrement;
lookupindex=(unsigned c
DDSValue=SINE_TABLE[loc

// Do the numerical mi
MixedValue = (signed c

// Integrate, first st
integrate1+=MixedValue;

// Integrate, second st
integrate2+=integrate1;

// Time to decimate ?
if(CombCpt++==COMBRATIO
{
    CombCpt=0;

    // The following co
    // it could be exe
    comb1=integrate2-la
    comb2=comb1-lastcom
    lastint2=integrate2
    lastco
```

```
Preprocessor symbol '__DEBUG' is defin
Sun May 24 17:10:36 2009

Make: The target "C:\Documents and Sett
Make: The target "C:\Documents and Sett
Loaded C:\Documents and Settings\Comp

Debug build of project 'C:\Documents and
Preprocessor symbol '__DEBUG' is defin
Sun May 24 17:10:36 2009

BUILD SUCCEEDED
```

Photo 1—The Labcenter Electronics VSM simulates both the hardware and the embedded firmware. On my desk you can see Microchip's MPLAB IDE on the left and Labcenter's VSM on the right, with the virtual scope in action.

# Embedded Single Board Computers

## High-End Performance with Embedded Ruggedness


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

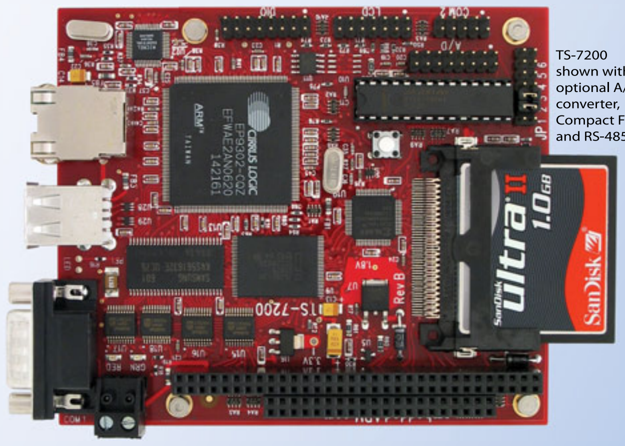
**TS-7800**
500 MHz ARM9

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

**$229** qty 100

**$269** qty 1

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


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

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

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

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

as low as **$99** qty 100

**$129** qty 1

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

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

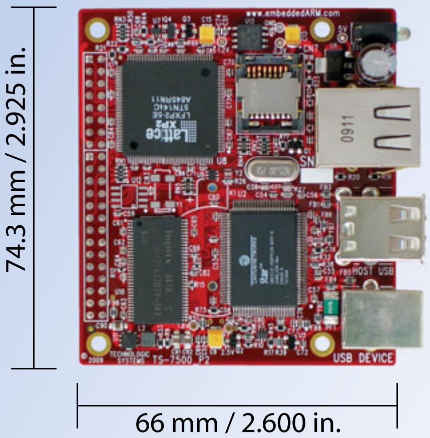
## New Products

**TS-7500**
250 MHz ARM9

NEW!

Our Smallest Computer at Our Best Price Point

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



$84 qty 100

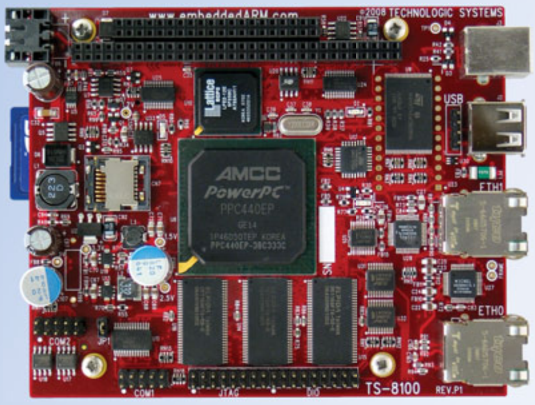
$99 qty 10

**TS-8100**
Ultra Reliable w/ 128MB ECC RAM

NEW!

400 MHz PowerPC with Floating Point Unit

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


shown w/ optional SD Card

$229 qty 100

$269 qty 1

Technologic Systems

We use our stuff.

Visit our TS-7800 powered website at

www.embeddedARM.com

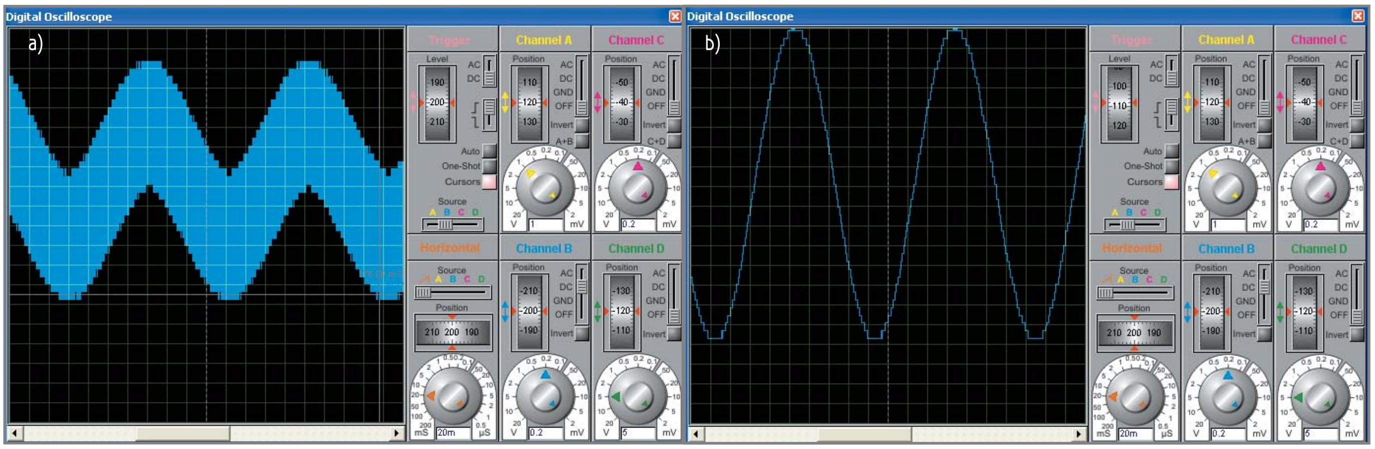


Photo 2a—The digital mixer's output shows (as expected) a 50-Hz signal along with some high-frequency noise, which is nothing other than the 14-kHz image frequency. b—When the CIC filter is in action, the virtual scope shows a clean 50-Hz sine wave.

its pass-band, a filter usually named invsinc (inverse sin(x)/x). The good news is that such a correcting FIR filter will be on the low bit-rate side and thus far easier to implement.

## FIRMWARE IMPLEMENTATION

Like all *Circuit Cellar* readers, I know that you prefer actual implementations rather than theoretical presentations, right? In high-speed systems like software-defined radios, CIC and multirate techniques are usually implemented either in high-end FPGAs or DSPs. There are also some generic digital receiver chips like Analog Devices's AD6640 that implement what I've covered in this article: a digital oscillator and mixer, a high-speed 65-Msps CIC filter, and a correcting FIR filter. The only difference is that it includes a complex mixer and two filter chains in order to be used for digital modulation systems. I'll provide more information in a future article.

CIC and multirate techniques can simplify low-cost microcontroller-based designs too. Let's look at an example. Suppose you need to design a Doppler sound velocity measurement system. Let's assume that you have a loudspeaker that sends a 7-kHz audio signal in the air and that close to it is a microphone to grab the sound bounced back by small moving objects. You want to know the relative speed of the moving objects. You can determine this by analyzing the received sound's Doppler shift: any object coming closer to the microphone will generate a frequency slightly higher than 7 kHz. Reciprocally, any object moving away will generate a sound frequency below 7 kHz. The frequency shift is simply 7 kHz times the ratio between the object's speed and the sound's speed in air, which is about 340 m/s. For example, if the object's speed is 1 m/s, the shift will be 20.5 Hz (i.e., 7 kHz × 1/340). So, if you analyze the microphone signal's frequency spectrum from 7 kHz – 50 Hz = 6,950 Hz to 7 kHz + 50 kHz = 7,050 Hz, you will detect and measure the speed of any object in a 2-m/s window backward or forward.

Analyzing the signal directly results in poor resolution, so you need to "zoom" on the frequency window prior to detailed analysis

Touch Panel PC

10.2"

CUPC-P80 $699 / Qty.1

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

7"

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

From $299 / Qty.1

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

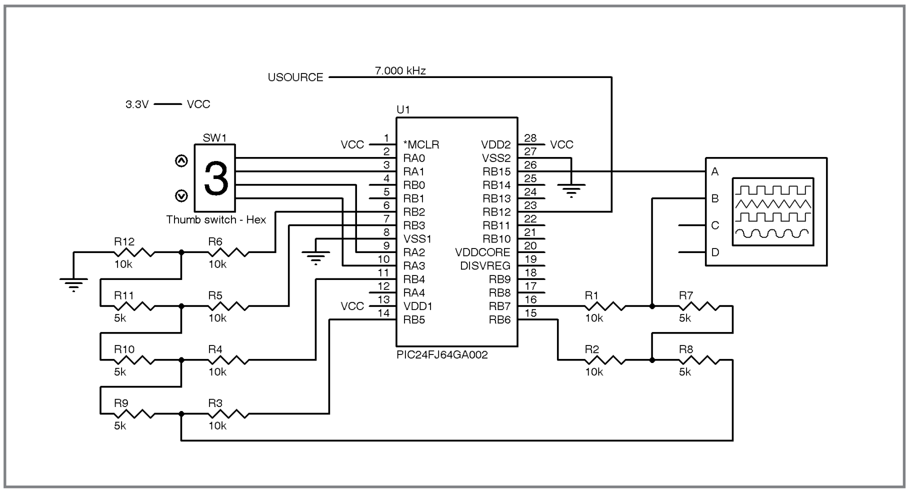


Figure 8—The input signal is sent directly to a PIC24FJ64GA002 ADC input. The firmware reads a BCD switch connector to RA0..3 in order to select which data to send out on RB0…7: ADC value, DDS value, mixed value, or filtered output. Then a crude 6-bit R2R DAC converts it in analog form. Lastly, a virtual scope displays it. Pin RB15 is just used to signal when the interrupt routine is active.

with an FFT (or something similar). This seems like a good application for multirate processing, right? Of course, this is exactly the same problem as the one I covered in the first part of this article. I moved the operating frequency from 5 MHz down to 7 kHz in order to use a low-speed processor and to illustrate the concept on a simple design.

Figure 7 shows the corresponding design. The input signal (6,950 to 7,050 Hz) is first digitized at 25 ksps with an ADC and mixed with a 7.050-kHz digital synthesized oscillator. This translates the signal of interest in low frequencies (0 to 100 Hz), but unfortunately with an image frequency close to 14 kHz. Then a CIC2 filters out the image frequency and reduces the sampling rate by a factor of 16, down to 1.562 kHz, which is enough for a 100-Hz bandwidth.

With such a frequency range, a low-cost, 16-bit processor like a Microchip Technology's PIC24FJ64GA002 is enough. Figure 8 shows you the corresponding schematic, which is more than simple. I coded the corresponding firmware in C language. It is less than two pages long, so don't hesitate to take a look at it on the *Circuit Cellar* FTP site. I haven't actually built this project, but thanks to Labcenter Electronics's VSM mixed-signal simulator, I tested it on my PC. Have a look at Photo 1. The beauty of a tool like VSM is that it enables you to simulate the design's analog portion like any Spice-based software (here the 7-kHz oscillator and the output DAC) and the firmware executed inside the PIC24FJ64GA002. The result is close to what you see in Photo 2. Thus, I am more than confident that this design would work without a hitch, even if I didn't build it with a soldering iron.

## HANDY TECHNIQUES

Here we are. In summary, CIC filters are actually simple moving average filters with a twist. The filters enable you to break down difficult signal-processing problems and reduce the requirements on computing architecture, especially when combined with software-based DDS and frequency mixers. So, any time you think that the useful information is far smaller than the actual sampling rate, multirate techniques can help. And now these techniques are no longer on the darker side for you. You have one more tool in your pocket!

*Robert Lacoste lives near Paris, France. He has 20 years of experience working on embedded systems, analog designs, and wireless telecommunications. He has won prizes in more than 15 international design contests. In 2003, Robert started a consulting company, ALCIOM, to share his passion for innovative mixed-signal designs. You can reach him at rlacoste@alciom.com. Don't forget to write "Darker Side" in the subject line to bypass his spam filters.*

## PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/231.

## RESOURCES

Altera Corp., "Understanding CIC Compensation Filters," Ver. 1.0, Application Note 455, 2007, www.altera.com/literature/an/an455.pdf.

R. Lyons, "Understanding Cascaded Integrator-Comb Filters," Embedded.com, 2005.

M. P. Donadio, "CIC Filter Introduction," http://users.snip.net/~donadio/cic.pdf.

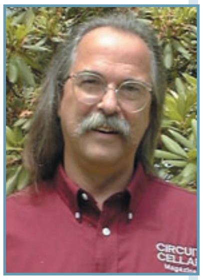
## SOURCES

**AD6620 Signal processor**
Analog Devices, Inc. | www.analog.com

**Proteus VSM mixed-signal simulator**
Labcenter Electronics | www.labcenter-electronics.com

**PIC24FJ64GA002 Microcontroller**
Microchip Technology, Inc. | www.microchip.com

by Jeff Bachiochi



# Airflow Analysis

Do you need a way to measure airflow in an air duct? Try deconstructing a small CPU fan, calculating the RPMs, and putting together a circuit to output data to an LCD. Airflow analysis made simple.

In an effort to improve my home's ability to sustain a comfortable temperature as efficiently as possible, I upped the insulation, added double-pane windows, and sealed all the potential cracks. Whether I want it cooler or warmer than the outside temperature depends on the climate (and the time of year). By reducing the transfer of heat through my home's walls, I reduce the energy necessary to warm it up or cool it down. In doing so, I am also saving money.

Along with my new cozy environment, I eliminated, or at least reduced, the exchange of air with the outside. This prevents evils like high pollen counts from infiltrating my home, but it also keeps in unwanted particulate matter like radon gas, mold spores, or even the common cold virus. The U.S. Environmental Protection Agency (EPA) has put together a comprehensive set of indoor air quality (IAQ) specifications recommended for new constructions. The "EPA Indoor airPLUS" label identifies qualified homes that have been verified to meet these specifications. There are three basic strategies for improving air quality: eliminate or control the sources of pollution, increase ventilation, and install air-cleaning devices. Usually, the most effective way to improve indoor air quality is to eliminate individual sources of pollution or reduce their emissions. Some sources, like those that contain asbestos, can be sealed or enclosed. In an enclosed area, second-hand smoke continues to affect all, so I don't allow smoking indoors. I try to stunt the growth of some sources of biologicals, such as mold and mildew, by keeping the relative humidity level to 50% or less in my home. Paints, varnishes, and wax all contain organic solvents, as do many cleaning, disinfecting, cosmetic, degreasing, and hobby products, so I store them outside whenever possible.

I eliminated air exchanges to increase energy savings, so improving ventilation when weather permits (the air temperature differential is minimal) is as simple as opening a few windows around the house. Exhaust fans in the bathroom or attic can aid this process if there is no natural breeze outside to move the air through the house. Remember, exhaust fans will do no good if there is no way for air to get into your home. While ventilation can exhaust those pollutants pent up inside your home, you may be trading one pollutant for another when you bring in "fresh" air, especially on days with high pollen counts.

Lastly, if I can't eliminate the source or exhaust it, I can attempt to clean it from the air. Air cleaners may be built into a home's HVAC system or operate as a portable standalone unit. The effectiveness of an air cleaner depends on how well it collects pollutants from indoor air and how much air it draws through the cleaning or filtering element. A very efficient collector with a low air-circulation rate will not be effective, nor will a cleaner with a high air-circulation rate but a less efficient collector. The long-term performance of any air cleaner depends on whether or not you maintain it in accordance with the manufacturer's specifications.

## SIZE MATTERS

First off, let's get an idea of what particle sizes we're dealing with when we're talking about pollutants (see Table 1). To be an effective air cleaner, the filter must be able to trap pollutants. It does this by preventing them from passing through its physical structure. A membrane filter acts as a sieve to catch particles larger than the openings between the fibrous strands that make up the filter screen or mat. Particles smaller than the sieve's openings may get trapped by other means, such as interception, inertial impact, or diffusion. A high-efficiency particulate absorbing (HEPA) filter can theoretically remove 99.97% of dust, pollen, mold, bacteria, and any airborne particles with a size of 0.3 µm at 3 cubic feet per minute (CFM). The American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE) uses a minimum efficiency report value (MERV) rating of 1 to 16 to designate filter effectiveness. Table 2 shows how this rating relates to the particle size and the filter's ability to remove it.

When you inhale, particulate matter less than 100 µm can enter your body through your nose and mouth. Your nose does a pretty good job of filtering those particles greater than 10 µm. Smaller particles move into the lungs where they can interfere with the exchange of oxygen. Any particulate matter less than 5 µm is considered hazardous.

The ASHRAE recommends a ventilation rate of 0.35 air changes per hour (ACH) for new homes, and some new homes are built to even tighter specifications. Particular care is taken in such homes to prevent the build-up of high levels of indoor air pollutants. What is ACH and how does this relate to clean air?

## ACHOO

Say you live in a 10′ × 10′ room with 8′ ceilings. This room has 100 ft$^2$ of floor space and 800 ft$^3$ of volume. With no openings (doors, windows, or vents), the 800 cubic feet of air in the room could move around the room; but over the period of 1 hour, no air would be exchanged with any on the outside, and thus the room would have 0 ACH. On a breezy day, if you opened a 1-ft$^2$ window at one end of the room, you might get a bit of air trying to enter. If air is forced into the room through the window, pressure would build within the room because the walls don't flex. The air has no way to escape except back out the window. Circulation ceases when the air pressure trying to exit equals the force of the fresh air trying to enter. While the air exchange is not quite zero, you might as well close the window.

Things can change dramatically when you open a second window at the opposite end of the room. Pressure building within the room from air entering at one side now has an escape route. It is clear to see that the velocity of the breeze entering the window will determine how quickly the air moves through the room. If you measure this velocity, you can determine how much air is moving through the window. Let's say you determined that the stiff breeze was moving at a rate of 1′ per second. The window opening is 1 ft$^2$, so the breeze is moving 1 ft$^3$ of air into the room each second. One cubic foot per second is 60 cubic feet per minute, or 60 CFM. In 14 minutes, that breeze can completely exchange all the air in the room (i.e., 60 CFM × 14 minutes = 840 ft$^3$). In 1 hour, the air could be changed four times (60 minutes per hour/14 minutes per air change = 4 ACH). This is approximately 11 times more than the ASHRAE recommendations for new home construction.

You won't be depending on Mother Nature to provide your home with adequate ventilation year round. If you have both heating and cooling in your home, there's a good chance you have a Heating, Ventilation, and Air Conditioning (HVAC) system. An HVAC system provides climate control throughout living and working spaces. Ducts or channels carry conditioned air from the heating and cooling unit to each

| Particle | Particle size (microns) |
|---|---|
| One inch | 25,400 |
| Dot (.) | 615 |
| Eye of a needle | 1230 |
| Glass wool | 1000 |
| Spanish moss pollen | 150 – 750 |
| Beach Sand | 100 – 10,000 |
| Mist | 70 – 350 |
| Pollens | 10 – 1,000 |
| Cayenne pepper | 15 – 1,000 |
| Textile fibers | 10 – 1,000 |
| Fiberglass insulation | 1 – 1,000 |
| Human hair | 40 – 300 |
| Dust mites | 100 – 300 |
| Saw dust | 30 – 600 |
| Mold spores | 10 – 30 |
| Starches | 3 – 100 |
| Red blood cells | 5 – 10 |
| Mold | 3 – 12 |
| Spider web | 2 – 3 |
| Spores | 3 – 40 |
| Combustion–related carbon compounds from motor vehicles, wood burning, open burning, industrial processes | Up to 2.5 |
| Fly ash | 1 – 1,000 |
| Talcum dust | 0.5 – 50 |
| Asbestos | 0.7 – 90 |
| Auto and car emission | 1 – 150 |
| Copier toner | 0.5 – 15 |
| Liquid droplets | 0.5 – 5 |
| Insecticide dust | 0.5 – 10 |
| Anthrax | 1 – 5 |
| Yeast cells | 1 – 50 |
| Carbon black dust | 0.2 – 10 |
| Atmospheric dust | 0.001 – 40 |
| Smoldering or flaming cooking oil | 0.03 – 0.9 |
| Corn starch | 0.1 – 0.8 |
| Sea salt | 0.035 – 0.5 |
| Bacteria | 0.3 – 60 |
| Bromine | 0.1 – 0.7 |
| Lead | 0.1 – 0.7 |
| Radioactive fallout | 0.1 – 10 |
| Tobacco smoke | 0.01 – 4 |
| Viruses | 0.005 – 0.3 |
| Typical atmospheric dust | 0.001 – 30 |
| Sugars | 0.0008 – 0.005 |
| Pesticides and herbicides | 0.001 |
| Carbon dioxide | 0.00065 |
| Oxygen | 0.0005 |

Table 1—This list gives you a good sense of the size of many common items, including those we consider pollutants.

| Minimum efficiency reporting values (MERV) ASHRAE Standard 52.2 | | | | | | |
|---|---|---|---|---|---|---|
| Group no. | MERV Rating | E1 Average Particle Size Efficiency (PSE) 0.3 – 1.0 Microns | E2 Average particle size efficiency (PSE) 1.0 – 3.0 Microns | E3 Average particle size efficiency (PSE) 3.0 – 10.0 Microns | Average Arrestance (ASHRAE 52.1) | Minimum final resistance (in. W.G.) |
| 1 | MERV 1 | – | – | Less than 20% | Less than 65% | 0.3 m |
| | MERV 2 | – | – | Less than 20% | 65 – 69.9% | 0.3 m |
| | MERV 3 | – | – | Less than 20% | 70 – 74.9% | 0.3 m |
| | MERV 4 | – | – | Less than 20% | 75% or greater | 0.3 m |
| 2 | MERV 5 | – | – | 20% – 34.9% | – | 0.6 m |
| | MERV 6 | – | – | 35% – 49.9% | – | 0.6 m |
| | MERV 7 | – | – | 50% – 69.9% | – | 0.6 m |
| | MERV 8 | – | – | 70% – 84.9% | – | 0.6 m |
| 3 | MERV 9 | – | Less than 50% | 85% or greater | – | 1.0 m |
| | MERV 10 | – | 50% – 64.9% | 85% or greater | – | 1.0 m |
| | MERV 11 | – | 65% – 79.9% | 85% or greater | – | 1.0 m |
| | MERV 12 | – | 80% – 89.9% | 90% or greater | – | 1.0 m |
| 4 | MERV 13 | Less than 75% | 90% or greater | 90% or greater | – | 1.4 m |
| | MERV 14 | 75% – 84.9% | 90% or greater | 90% or greater | – | 1.4 m |
| | MERV 15 | 85% – 94.9% | 90% or greater | 90% or greater | – | 1.4 m |
| | MERV 16 | 95% or greater | 95% or greater | 95% or greater | – | 1.4 m |

Table 2—Filters are rated according to their ability to trap particles down to 0.3 µm. MERV values of 1 to 4 can handle particles greater than 10 µm, while higher values must be used to trap smaller particles with greater efficiencies.

room and back again. From the previous example, it is clear that the size of a window opening (or duct in this case) will affect the CFM potential for any room. Remember: you used air velocity and the size of the opening to determine CFM, and then CFM and room volume to determine ACH.

## PUSH VS. SHOVE

An HVAC system uses a motor and fan to circulate air through the ducts to each room and back through return ducts again. The size of the motor determines how much air can be moved. Figure 1 shows the relationship between the amount of air that can be moved and the pressure at which it can move the air. You can think of the pressure as the push required to overcome any obstacle preventing air movement (e.g., duct size or an air filter). Like Mother Nature, the motor/fan will only be able to provide airflow if the pressure remains below the unit's designed limit. Note that HVAC pressures are measured in inches of water.

Providing air quality in the HVAC system requires the use of filters to remove contaminates from the air. Since HEPA filters are expensive, it is good practice to add a (inexpensive) prefilter to prevent large particles from clogging up a HEPA filter. As you can see in Figure 2, HEPA filters can introduce rather large pressure drops to an HVAC system due to the small openings in the filter material. A HEPA filter attempts to reduce the required pressure by adding more square feet of surface area (deep corrugation pockets) to the oncoming airflow.

While initial system design may provide adequate CFM to ensure a minimum number of ACH, contaminants

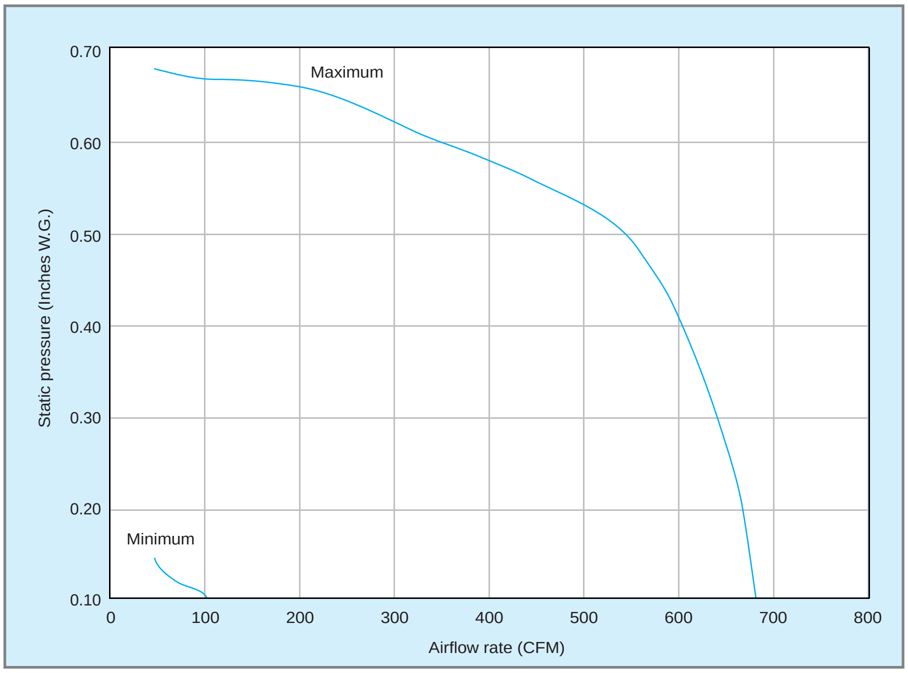


Figure 1—A system loses its ability to move air as its path is impeded. Ducts, filters, and louvers cause a rise in pressure measured in inches of water. This pressure slows down the motor/fan's effectiveness reducing the CFM.

begin to restrict airflow as filters pick up particulate matter. While this can actually increase the efficiency of the filters, it raises the system pressure and the CFM of air that the motors can produce goes down. Over time, the system works harder and may fail in providing the required ACH. If this is unmonitored, you have a problem that continues to get worse. To some, this can give rise to health issues.

## DAYLIGHT SAVINGS TIME

Daylight savings time continues to be controversial. But required time changes remind us to take maintenance measures like changing the batteries in a smoke detector. We can all use this as a spring board to getting to all of those other chores that require a periodic look see, like cleaning the gutters or changing the filters. If you are like me, you clean the gutters after you notice them overflowing during a storm. I'd prefer to know when a filter needs replacing, rather than arbitrarily changing it.

Recently, I was asked to develop a way of measuring airflow within an air duct. From a measured velocity, the volume (CFM) of air can be calculated for the duct size. If these measurements were taken in a number of key areas, you could paint a picture of the HVAC system's performance. There are three basic techniques for how this might be accomplished: hotwire, pressure, and mechanically.

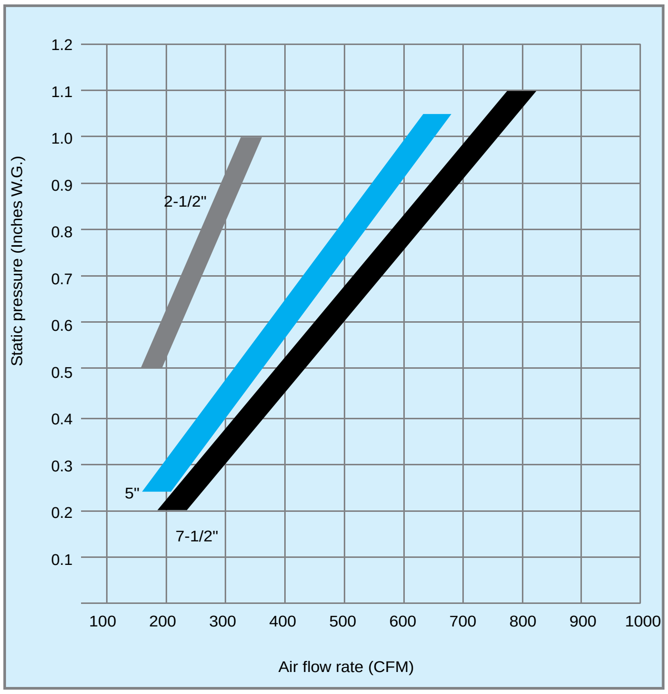


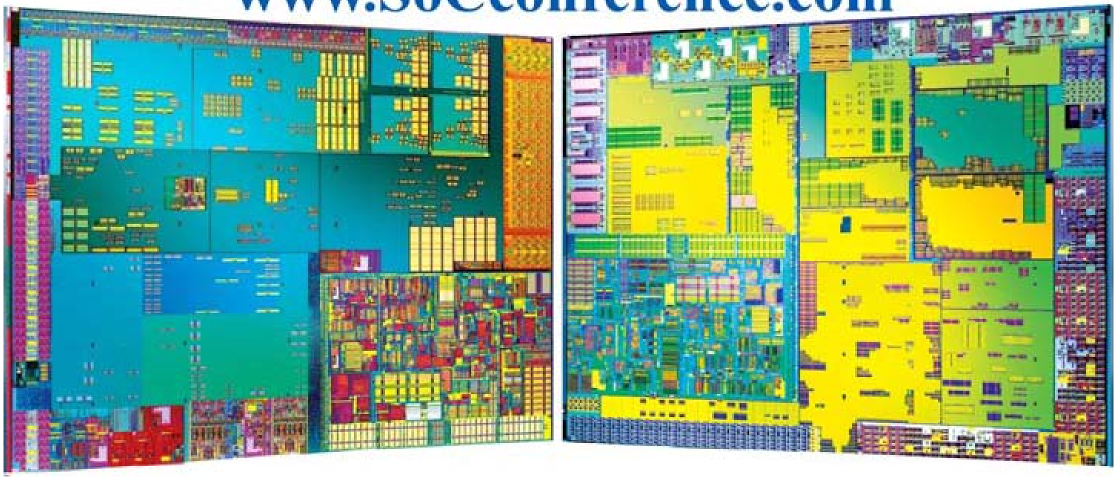
**Figure 2**—HEPA filters cause large pressure drops because the air must work to find its way through the filter material. One way to reduce a HEPA filter's pressure is to choose one with a deeper corrugation, which increases its surface area.

**7th International System-on-Chip (SoC)**

**Conference, Exhibit & Workshops**

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

**www.SoCconference.com**



SoC 7th

INTERNATIONAL SYSTEM-ON-CHIP CONFERENCE

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

**www.SavantCompany.com**

**Die images courtesy of Intel Corporation.**

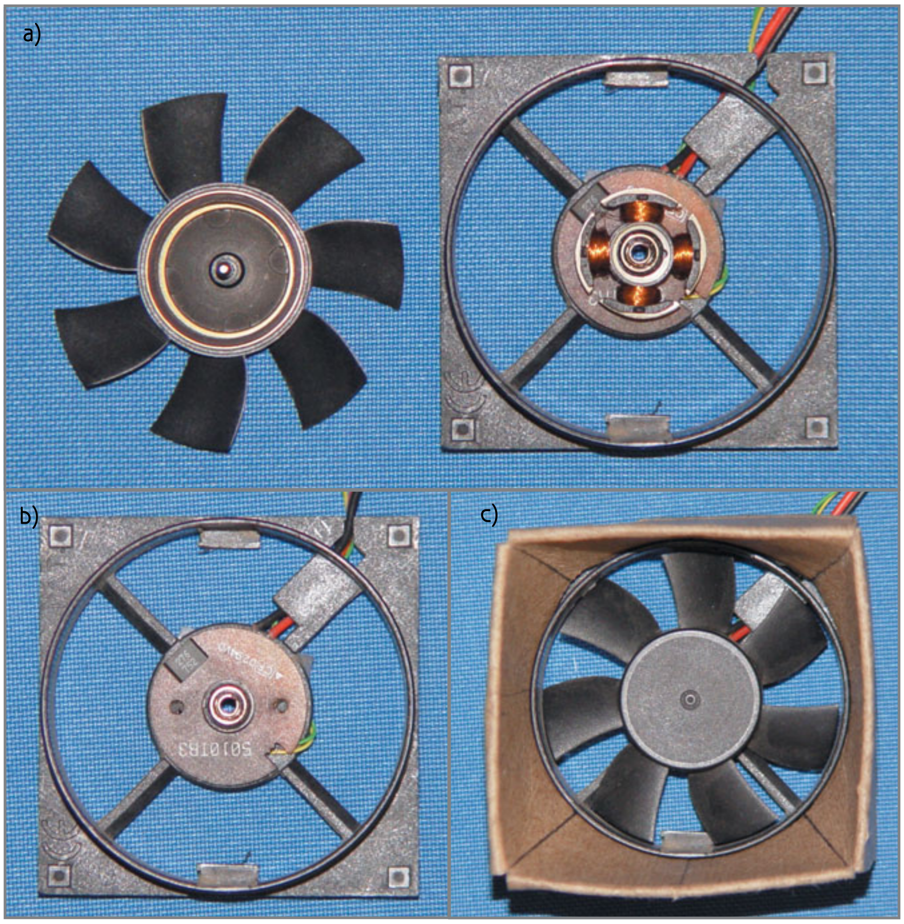


Photo 1a—Once you remove the fan blade, you can take out the stator yoke and coils. Note the ring magnet on the inside of the fan impeller. b—With the stator yoke and coils removed, you can see the Hall effect device, which was used to alternately power the coils. Now it becomes the pulse output used as a tachometer. c—To reduce turbulence, I constructed a shroud over the fan's frame. This allows a smooth transition between a square opening and round hole.

The hotwire method monitors the current flowing through a heated element. The faster air flows across the exposed element, the more heat is drawn off the element. The corresponding rise in current—which is necessary to keep the element at a specific temperature—is directly proportional to velocity. The pressure method uses the difference in pressure between the directional (total) and nondirectional (static) components of the flow to compute dynamic pressure. The dynamic pressure is measured in inches of water that can be converted directly to velocity. The mechanical method uses a flap or vane that is physically pushed by the air. The amount of movement is directly related to the velocity of the air pushing on it.

If a fan is used to produce air flow, why not use a fan to measure it? Many of today's DC fans use a plastic blade molded around a magnetic rotor ring, which rotates freely on sleeve or ball bearings. The internal stator is made up of coils wound around a stationary multi-pole yoke. When the fan is unpowered, the magnetized rotor is attracted to a laminated metal yoke. This magnetic bond takes a bit of force to overcome and as such won't allow the rotor to turn freely in low airflow. The yoke must be removed. So the search begins for a fan that allows the easy disassembly and removal of the yoke and coil assembly.

After purchasing a number of fans and dissecting them, I was having no luck. Either I couldn't get the rotor off or the yoke laminations were molded into the frame. Finally, I was able to get inside a COOLTRON fan and extract the innards. Although it seemed as though the blades rotated freely, it still had too much friction in the sleeve bearing and wasn't as sensitive as I needed. Then I found a small CPU fan I had pulled off a junk board. It had ball bearings and came apart easily (see Photo 1a). With the guts out, it spun with the smallest puff of air. Now I had to find a Hall effect switch to use as an RPM counter.

## TACHOMETER OUTPUT

I didn't have to look far. In fact, the drive electronics for the yoke coils in this CPU fan are based on a Hall effect device. That meant a free tachometer output from the fan's leftover circuitry (see Photo 1b). How much better can it get? I connected 5 V to the 12-V fan leads and probed the Hall outputs. It has two clean 5-V toggles (the second 180° off from the first) that flip twice per revolution. It looks like the yokes were originally driven directly by the Hall effect device's outputs (see Photo 1c).

Next, I had to figure out how the fan's RPMs relate to velocity. The datasheet lists that this fan runs at 4,200 RPMs. I needed a quick circuit that could output some text on an LCD. Having recently finished a few projects with LCDs, I prototyped the

```
    4=RPS    V=01.472
CFM=0088.2    (1sqft)
```

Photo 2—The LCD displays RPS and velocity. RPS is the Hall effect count/2 (two counts/revolution). The velocity is calculated from a calibration value and the RPS.

idea pretty quickly. The plan was to count revolutions as an interrupt. With a timer set to overflow every second, I found I could grab the count or revolutions per second (RPS), clear the count, and print the results while continuing to count revolutions for the next conversion. Counting revolutions is the most critical item so it is handled by an interrupt and can take over when necessary. Working with a 1-s time frame makes everything easy. Since I would be working in seconds, I needed to convert the 4,200 RPMs into RPS—that's 70 RPS (i.e., 4,200 RPM/60 s).

## RPS VS. VELOCITY

The LCD reported the RPS count accumulated over the previous second as "xxxxx=RPS" (see Photo 2). Now I needed to find the relationship between the air velocity that flows through and pushes on the fan blades and the RPS produced. To do so, I grabbed my car keys and drove to a vacant parking lot where I could take some readings.

First, I charted RPS based on a steady-state MPH. Then, I repeated the testing charting MPH based on RPS. I knew that if the data for both tests was comparable, I was in the ballpark. I found that at the rated RPS (70) my speed measured 18 MPH, and 1 MPH was 5,280′/hour, or approximately 1.5′/s. Therefore, I could calculate the air velocity equal to 1.5′/s for each MPH. (At 18 MPH, that's 27′/s!) This meant that each revolution was equal to 27′/s per 70 RPS, or 0.386′. Thus, the measured air speed was RPS × 0.386′.

I calculated the air velocity by multiplying the RPS count by a factor I obtained from the calibration testing. To keep away from floating-point, I scaled the fraction by 1,000. The second line of the display presents the result as "Velocity=xx.xxx."

$$1 \text{ RPS} = 0.386 \text{ ft/s}$$
$$V \times 1{,}000 = \text{RPS} \times 386 \text{ ft/s}$$

## CFM

A CFM calculation is based on the velocity and the cross sectional area of the duct. Let's say that the air you're measuring takes 1 s to travel 1′ (1′/s). The air duct measures 1′ × 1′, for an area of 1 ft$^2$. Air moving 1′/s through this air duct has a volume or rate of 1 ft$^3$/s, or 60 CFM. Based on a 1 ft$^2$ duct, CFM is equal to: velocity × 60. I use this duct size to display CFMs as "CFM=xxxx.x (1sqft)." I assume (somewhat incorrectly) the air speed being measured by the fan is the average air speed through any duct. The air through a duct actually flows slower near the duct's walls than in the center of the duct. Turbulence is introduced by any attempt to change the airflow's direction. This might be from a turn in the duct or an object (like my fan) placed within it. For the best results, try to measure at a point at least a few duct diameters from any turn. If you will be measuring at the duct's outlet, you can easily take multiple readings around the opening and average them.

In the previous case, 60 CFM was based on a velocity of 1′/s and cross-sectional area of 1 ft$^2$. To relate this to your duct size, simply measure its area in inches, either L × W or $\pi R^2$. For a 5″ × 10″ duct, that's 50 in$^2$. (It's about the same for a 8″ round duct.) Then find the ratio of your duct by dividing its area by 144 to see how it relates to a 1 ft$^2$ duct (50/144 = 0.34). Now calculate your CFM by multiplying the measured CFM by your ratio. If the fan is measuring a CFM as 60 and your duct's ratio is 0.34, your duct's CFM rating is 20.4 CFM (i.e., 60 CFM × 0.34).

To relate this number to your room's size, go back to the total volume of the room. Remember the 10′ × 10′ room with 8′ ceilings? With a CFM of 20.4, it will take 800 ft$^3$/20.4 CFM, or approximately 39 minutes, to complete an air change. This is roughly 0.66 changes per hour. Of course, this is based on a heating or cooling cycle of 100%. To conserve the most money, I'm talking about having a low "on-time" percentage. If you are achieving a 10% on-time, then 0.66 changes per hour becomes 0.066 changes per hour. This is below the minimum ACH level recommended by the EPA. Many HVAC systems can run air circulation without adding or removing heat. Although this will increase the ACH, it also increases energy consumption.

## PORTABLE ACH

The poor souls who happen to be overly sensitive to pollutants—even if only during certain times of the year—may find relief by installing portable air cleaners. These give localized comfort to a room by supplementing an HVAC system with additional (HEPA-filtered) air changes without running the HVAC continuously. If you purchase one of these systems, you may want to periodically verify that the unit is still giving you the manufacturer's rated CFM by doing a velocity measurement on the unit's output vent! When the CFM is down, change that filter. ■

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

## PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/231.

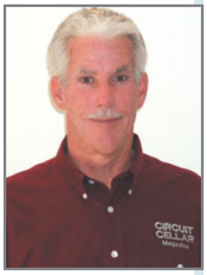
## RESOURCES

Pure Air Systems, "103 - Filtration Fundamentals," www.pureairsystems.com/Pure-Air-University-103-Filtrat1.html

The Engineering Toolbox, "Particle Sizes," www.engineeringtoolbox.com.

U.S. EPA/Office of Air and Radiation, "The Inside Story: A Guide to Indoor Air Quality," 2009, www.epa.gov/iaq/pubs/insidest.html.

by Tom Cantrell



# Thumbs Up

## The ARM Saga Continues

It's been more than two decades since Acorn Computers introduced its RISC CPU. The saga continues with the latest chapter in the ARM story, the ARM Cortex-M0 core.

Britannia may no longer rule the waves, but UK-based ARM marches on as a superpower in the MCU biz. With heavyweight allies (e.g., Texas Instruments, NXP Semiconductors, Atmel, and many more) projecting power across the globe, the sun never sets on the ARM empire.

I've enjoyed seeing the saga of the little Acorn Computers RISC machine as it has matured into a mighty oak towering over the 32-bit MCU market. Has it really been more than 20 years? Time flies when you're having fun!

And the party isn't over yet. This month, let's take a look at the latest chapter in the ARM saga, some big news about a small chip.

## LESS IS MOORE

"When we decided to do a microprocessor on our own, I made two great decisions. I gave them two things National, Intel, and Motorola had never given their design teams: the first was no money, the second was no people. The only way they could do it was to keep it really simple." That's the story of the birth of the ARM1 processor in the mid-1980s according Dr. Hermann Hauser, one of ARM's original founders.[1] Back then, computer chips were simple because they had to be. But the years since have seen an explosion of complexity as Moore's law has inexorably driven the cost of silicon towards zero.

Right now I'm looking at the ARM programmer "Quick Reference Card," and it's a wonder to behold. At six 8.5″ × 11″ pages, it's more like a full "deck" than a "card." You want to build a Cray-in-drag? No problem, thanks to the "Unsigned Sum of Absolute Differences and Accumulate" instruction. Want to do double-duty as a DSP? The "Multiply Unsigned Double Accumulate Long" instruction is just the ticket. There are instructions for bit fields, semaphores, cache preload, coprocessors, memory barriers, compiler hints, and more.

Don't get me wrong. I'm as much a fan of the big super-duper chips with all the fancy architectural trinkets as anyone. But it just seems that along the way the Silicon wizards forgot that simplicity can be a virtue in its own right, even though it is no longer the only option.

After many architectural revisions and embellishments over the years, today's high-end ARM chips bear little resemblance to the lean and mean originals. A top-of-the-line Cortex-A8 accommodates three instruction sets: native ARM with 32-bit opcodes, the 16-bit opcode Thumb subset, and now the variable length Thumb-2 codes. It's plenty advanced, but there's nothing "reduced" about this RISC with its expanded repertoire of complex features for signal processing, vector math, graphics, security, Java, multithreading, multiprocessing, and on and on.

At the other end of the spectrum, ARM7-based parts, and now Cortex-M3 flash MCUs, are quite capable and popular. But even these

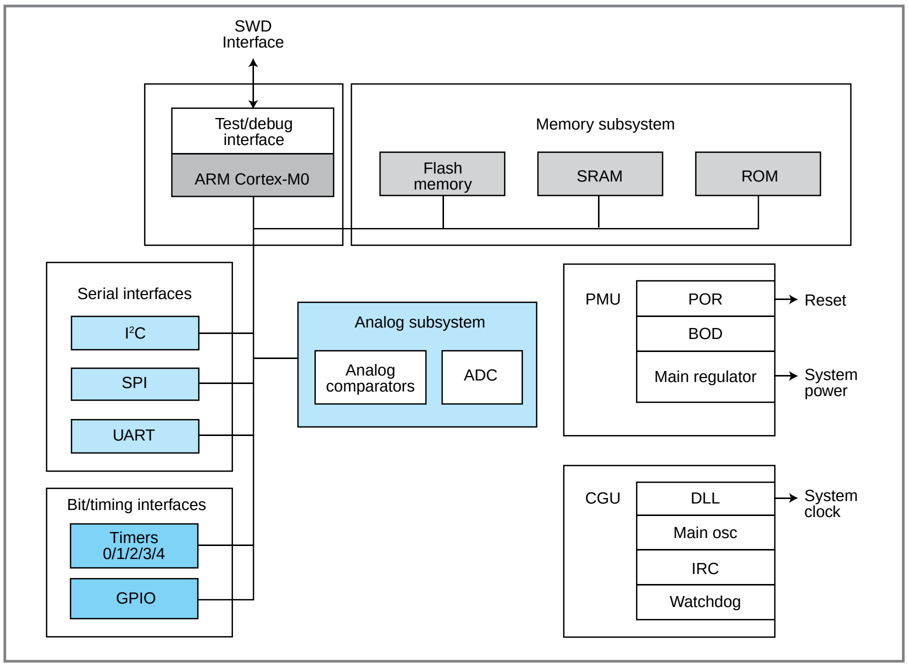


**Figure 1**—The ARM Cortex-M0 core heads back to the future with a design that recalls the simplicity of the original "Acorn RISC machine."

are arguably fatter than they need to be when it comes to the bluest of blue-collar apps. More power to them (literally).

Enter the ARM Cortex-M0 core, championed by NXP as the basis for their new LPC1100 flash MCU lineup. It may be the newest ARM chip on the block, but it's simpler than anything else in NXP's—or anyone else's, for that matter—catalog of ARM chips.

Actually, 'M0 isn't "new" in the strictest sense. It's the same basic instruction set architecture as the Cortex-M1 soft core I wrote about in 2008 ("Icy Hot," *Circuit Cellar* 217). The difference is more about the significant performance, power, and price advantages that come with LPC1100 optimized hard-core implementation.

## SOCKET ROCKET

Put a 150-HP engine in a car and you've got a yawn-inspiring econobox. Put a 150-HP engine in a motorcycle and you've got an aptly named "crotch rocket" that will burn rubber and get 50 MPG while doing it.

That's the analogy that comes to mind as I ponder NXP's new LPC1100 lineup. With a small package (33-pin QFN, 48-pin LQFP), a moderately sized memory (up to 128-KB flash memory, 16-KB RAM), and a middle-of-the-road mix of peripherals (PIO, SIO, PWM, ADC, etc.), it could be mistaken for a typical 8-/16-bit MCU (see Figure 1). But when you twist the LPC1100's throttle, you'll discover there are plenty of MIPS on tap courtesy of a 32-bit Cortex-M0 core coupled to an optimized flash interface.

One way to describe the 'M0 core is to trace its lineage through the historical meanderings of the ARM architecture. But instead of a blow-by-blow account, I'll just sum up 'M0 as "16-bit Thumb instructions plus a few system-oriented Thumb-2 instructions" (see Figure 2). 'M0 is so simple you really don't need a history lesson to figure it out.

Lest there be any confusion about the "16-bit" aspect of 'M0, make no mistake: it's a real 32-bit processor. The "16-bit" attribute simply refers to the fact most 'M0 instructions fit in just 2 bytes. But the 'M0 programmer's model, ALU, and internal registers, busses, and memory are all a full 32 bits wide. As they say, there's no substitute for cubic inches.

The "16-bit" story is usually cast as one of "code density" (i.e., shorter opcodes = shorter programs = less flash = less cash). There's a lot to that. After all, even though memory is ever cheaper, it still doesn't cost zero dollars and it doesn't consume zero power.

But there's also a "performance" story that goes like this. In typical applications, general-purpose processors are usually instruction-fetch

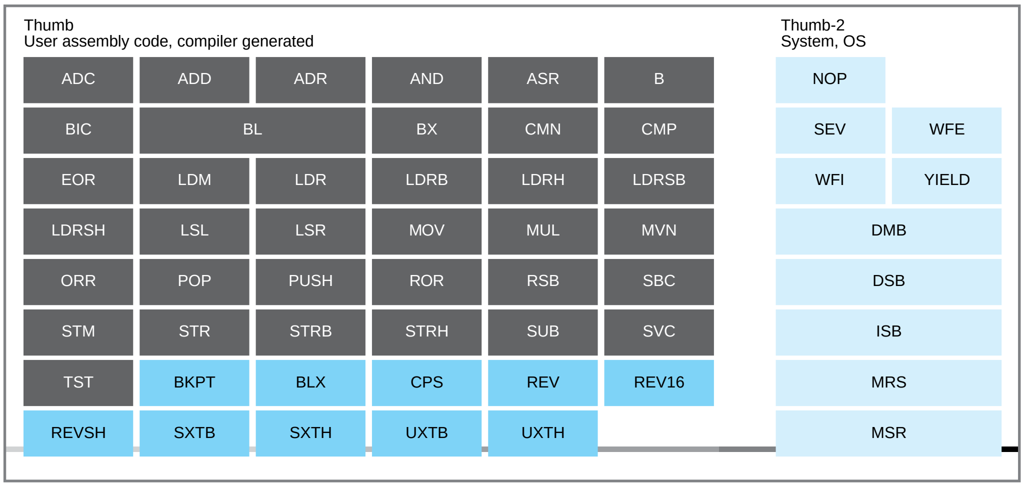


**Figure 2**—With just 56 instructions, the 'M0 instruction set is truly reduced compared to the verbosity of today's "complex" RISCs. Most 'M0 opcodes are just 16 bits long, which both reduces the amount of flash required and helps bypass the flash bottleneck.

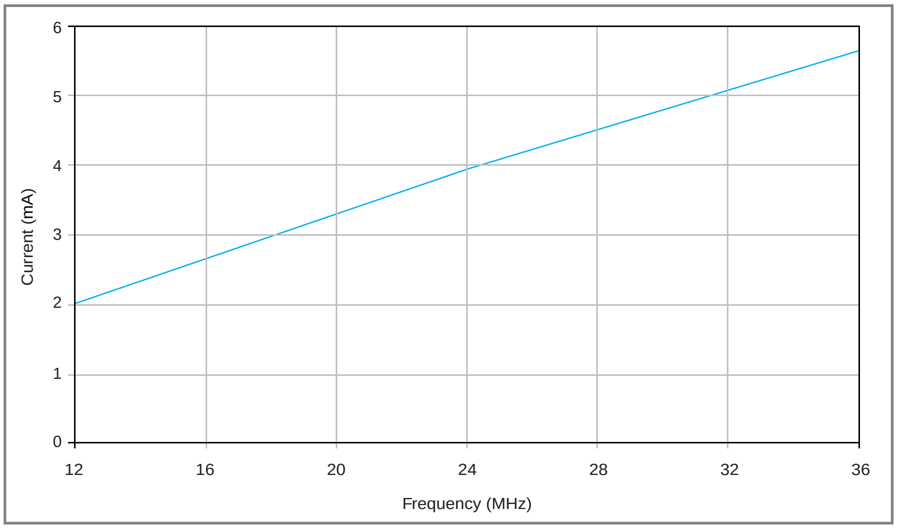


**Figure 3**—At 150 µA/MHz, the LPC1100 delivers on the promise of 32-bit performance with 8/16-bit power consumption. Whether by reducing execution time or running at a relatively slower clock rate, the LPC1100's 32-bit processing capability can reduce the total energy spent completing a given task.

bound. It's all the worse for a single-chip MCU where slow flash memory holds the processor back. Shorter 16-bit opcodes sent two at a time over the internal 32-bit bus help the LPC1100 bypass the dreaded "flash bottleneck," which allows the 'M0 core to reach its full potential, said to be 0.9 DMIPS/MHz. (For what that's worth, refer to the "Dhrystone RIP?" sidebar.) That's better performance than an ARM7 running Thumb instructions (0.74 DMIPS/MHz). Better yet, the 'M0 does it with less silicon, since an ARM7 has to carry around the baggage of the full ARM instruction set even if it's mainly running Thumb code.

Put it all together and you've got an LPC1100 that's fast, nimble, and gets great mileage too (see Figure 3). Yeah, it's not what you'd choose for hauling a bloat-load of software, but there are plenty of bigger chips for that. If you need a simple and inexpensive way to get from stoplight to stoplight really quickly, hop on!

## TIMING CHAIN

Features that well serve "computers" may not be useful when a 32-bit architecture is drafted for duty in embedded applications. Indeed, they can even make matters worse.

The classic example is cache. It works great in your PC, but is just a headache in real-time interrupt-driven apps. The statistical nature of cache behavior introduces timing uncertainty—"jitter," if you will—that can be a real headache. If you're not careful, a heavily interrupt-driven app can turn into a cache-thrashing thrill ride with more fills and spills than useful processing. Sure, there are usually ways to get around cache "issues"—for example, by pre-initializing and locking the cache. Yes, you can coerce a cache into acting like a RAM, just one that happens to be really wasteful (silicon, power) and slow.

There are other less obvious sources of "jitter" to watch out for. Things like MMUs and branch prediction bring along their own caching behavior. Bus interfaces can be too smart for their own good, with features like burst mode, merging, and reordering, making bus timing a your-guess-is-as-good-as-mine proposition. There may even be cases where instruction timing is variable, such as an "early-out" math instruction where the execution time varies depending on the value of the arguments. Don't even get me started on out-of-order execution and speculation.

By contrast, the LPC1100 features a blessedly simple and predictable timing model. ALU operations (including 32-bit multiply) take just one cycle. Loads and stores require two cycles because the 'M0 (like ARM7) uses a shared bus for instructions and data. Branches take three cycles. Period.

The 'M0 takes advantage of a built-in "Nested Vectored Interrupt Controller" (NVIC), the same one earlier integrated in the 'M3. High-priority interrupt response time is 16 clocks, which seems a bit leisurely until you remember the NVIC automatically handles register stacking and unstacking so that interrupt service routines can be written completely in C (i.e., without an assembly language dispatcher or wrapper).

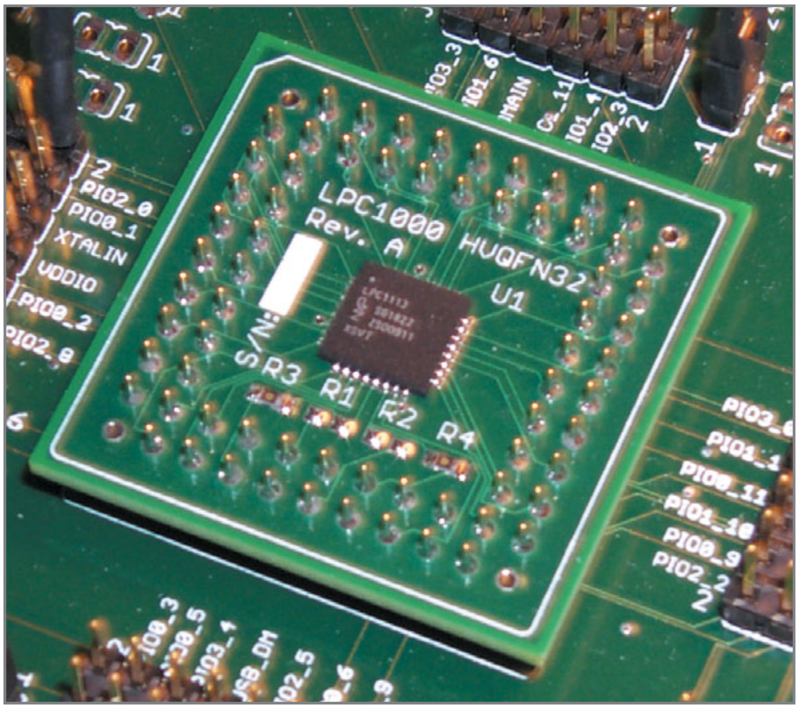
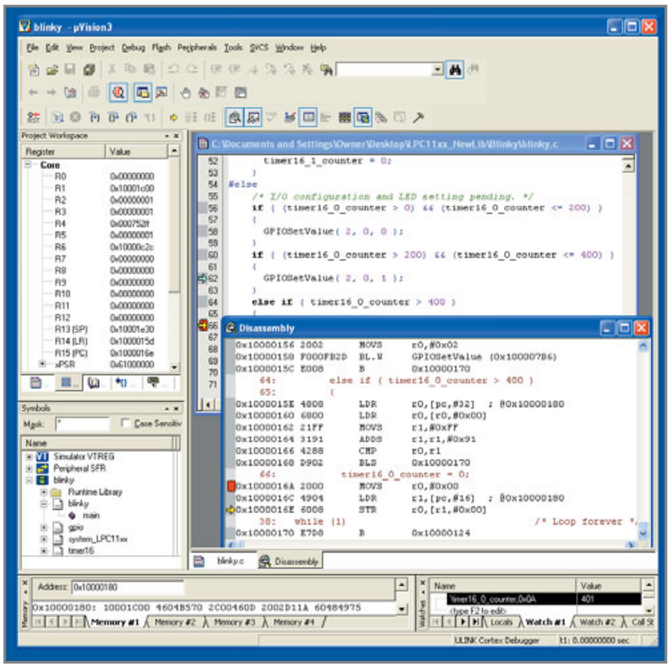


**Photo 1**—With their LPC1100 MCU, NXP is first out of the starting gate with the new ARM Cortex-M0 core. Think 32-bit performance and headroom with 8/16-bit price and power consumption.



| Register | Value |
|---|---|
| Core | |
| R0 | 0x00000000 |
| R1 | 0x10001c00 |
| R2 | 0x00000001 |
| R3 | 0x00000001 |
| R4 | 0x000752ff |
| R5 | 0x00000001 |
| R6 | 0x10000c2c |
| R7 | 0x00000000 |
| R8 | 0x00000000 |
| R9 | 0x00000000 |
| R10 | 0x00000000 |
| R11 | 0x00000000 |
| R12 | 0x00000000 |
| R13 (SP) | 0x10001e30 |
| R14 (LR) | 0x1000015d |
| R15 (PC) | 0x1000016e |
| xPSR | 0x61000000 |

C:\Documents and Settings\Owner\Desktop\LPC11xx_NewLib\Blinky\blinky.c

```
      timer16_1_counter = 0;
    }
#else
    /* I/O configuration and LED setting pending. */
    if ( (timer16_0_counter > 0) && (timer16_0_counter <= 200) )
    {
      GPIOSetValue( 2, 0, 0 );
    }
    if ( (timer16_0_counter > 200) && (timer16_0_counter <= 400) )
    {
      GPIOSetValue( 2, 0, 1 );
    }
    else if ( timer16_0_counter > 400 )
    {
```

Disassembly

```
0x10000156 2002      MOVS     r0,#0x02
0x10000158 F000FB2D  BL.W     GPIOSetValue (0x100007B6)
0x1000015C E008      B        0x10000170
   64:          else if ( timer16_0_counter > 400 )
   65:          {
0x1000015E 4808      LDR      r0,[pc,#32]  ; @0x10000180
0x10000160 6800      LDR      r0,[r0,#0x00]
0x10000162 21FF      MOVS     r1,#0xFF
0x10000164 3191      ADDS     r1,r1,#0x91
0x10000166 4288      CMP      r0,r1
0x10000168 D902      BLS      0x10000170
   66:            timer16_0_counter = 0;
0x1000016A 2000      MOVS     r0,#0x00
0x1000016C 4904      LDR      r1,[pc,#16]  ; @0x10000180
0x1000016E 6008      STR      r0,[r1,#0x00]
   38:   while (1)                                /* Loop forever */
0x10000170 E7D8      B        0x10000124
```

```
Address: 0x10000180
0x10000180: 10001C00 4604B570 2C00460D 2002D11A 60484975
```

| Name | Value |
|---|---|
| timer16_0_counter,0x0A | 401 |
| <type F2 to edit> | |

Photo 2—It is no surprise that Keil, having been acquired by ARM, already has Cortex-M0 support in their MDK-ARM package. Other third-party suppliers should be on-board soon, because adding 'M0 support to their existing ARM tools will be easy given the common "Thumb" roots. Speaking of "Thumb," notice the preponderance of 16-bit opcodes in the program shown here.

Another 'M3 feature carried forward (or is it backward?) is a power-management unit with three low-power modes supported with a plethora of dynamically programmable clocking options, including on-chip 1% trimmed 12-MHz RC oscillator and a PLL (see Figure 4). A Wakeup Interrupt Controller (WIC) enables quick wake-up from deep sleep using a variety of sources (i.e., not just RESET). The LPC1100 will run off anything between 1.8 and 3.6 V and there's a four-stage brownout detector to keep a close eye on the power supply.

## CHIPS OF FUTURE PASSED

Usually, I don't write-up chips until they—and the documentation, tools, etc. that go with them—are on the shelves. But as I write this column, very little information is publicly available. NXP will only say the plan is to ramp production by "the beginning of 2010." So why am I making an exception by writing up a chip with little more than press releases and couple of presentations to go on?

Even though the official introduction is months away, I've got a chip (see Photo 1), board, and tools up and running right now. It turns out NXP already has preproduction silicon and a board for testing and doing their homework (e.g., tool integration), and they sent me a kit, albeit a work in process.

Let's see. Here's a board and a JTAG dongle (Keil ULINK2) and some files an apps engineer from NXP e-mailed to me. There's the JTAG header on the board, and I guess the PC plugs into this USB connector. Easy enough, but what about tools?

I surfed over to Keil.com and crossed my fingers as I did a search for "cortex m0." I let out a sigh of relief when I discovered the latest version of their Microcontroller Development Kit (MDK) had been upgraded with 'M0 support, and you can download a 32-KB, code-limited evaluation version for free.

The new release of the MDK comes with an optional "MicroLib" C library optimized to produce tight code for embedded applications. Here's another example (à la the "Dhrystone RIP?" sidebar) of how the library routines, not just the compiler, can make a big difference (see Figure 5). MicroLib streamlines code by tossing features (e.g., C++ support), detuning others (e.g., options for less-than-full STDIO support), and trading off some speed for smaller code.

Also new is something called "CMSIS," which stands for Cortex Microcontroller Software Interface Standard. CMSIS recognizes the fact that "CPU compatibility" is only half the battle when it comes to migrating applications between MCUs that may have the same CPU, but completely different ways of handling low-level I/O (e.g., peripheral registers) and system functions (e.g., "systick" timer). CMSIS provides standard definitions and a level of hardware abstraction that boosts compatibility and software reuse to minimize reinventing the wheel.

With the MDK installed, I fired up everything and looked for smoke. Given the newness of it all, I frankly had my doubts. But to my surprise, the lash-up immediately showed signs of life. I was at least expecting some of the typical installation issues like "drivers" for the JTAG pod. But it turns out the ULINK2 uses a USB human interface

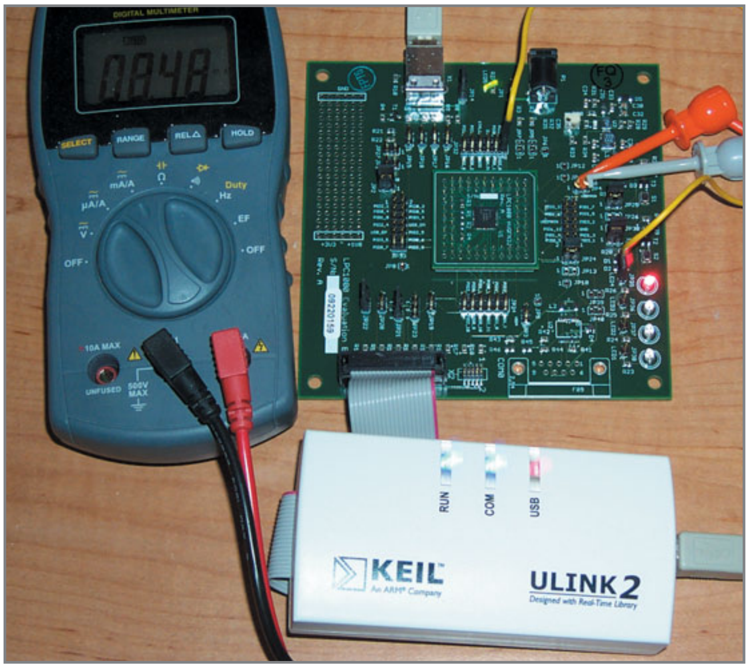


Photo 3—In the old days, chip performance was all that mattered, but now "green" designs and longer battery life are taking center stage. With NXP's Cortex-M0-based LPC1100 MCU, you can have it all: more MIPS and less milliamps.

device (HID) profile that's already built into Windows (a trend I've noticed with other embedded USB gadgets).

Better yet, right off the bat, I was able to compile, download, and debug a simple "blinky" (blinking LED) application, kind of the embedded equivalent of "Hello World" (see Photo 2). The program seemed to be running as I could set breakpoints, single-step, and so on. But there was just one problem: the LED on the board wasn't blinking. Turns out there needs to be a jumper between the I/O pin and LED on the board because the LEDs aren't hardwired to anything. That's actually a useful approach for experimenting because you can connect the LEDs to any signal of interest as kind of a poor man's logic probe.

## DHRYSTONE RIP?

I don't subscribe to the old saw, "Benchmarks lie." I think benchmarks tell the truth, although no doubt as the benchmarker happens to see it. Where folks stand on the validity of a benchmark depends on where they sit in the result list.

There are plenty of fancy benchmark suites that do a good job measuring big-ticket system and application performance. But when it comes to a simple metric for comparing blue-collar MCUs, the venerable Dhrystone benchmark (created by author Reinhold P. Weicker in 1984) has filled the gap.

Like all benchmarks, Dhrystone has "issues"—such as compiler and library "advances" (some might say, "cheating")—that, for example, optimize away semantically superfluous code. For instance, let's say you hack your own quickie "synthetic" benchmark like the following, fully expecting to run 1,000 iterations of a loop, each with an add, multiply, and store (i.e., many thousands of instructions in total):

```
FOR I = 1 TO 1000
  A(I) = I*(I+1)
NEXT I
```

So now I tell you I've got a processor that can run your benchmark in zero cycles! Snake oil? Not at all. Just a compiler smart enough to realize it can do all the work ahead of time, turning the benchmark "program" into a big data statement:

```
A: DATA 2, 6, 12…
```

The computation is done at compile time, and all the "stores" are performed when the "program" is loaded into RAM. The result (i.e., 1,000 calculated values stored in array A) is exactly the same as running the loop, but there is no "runtime."

It's questionable whether overly clever optimizations are applicable to real-world apps. Note: It's easy to make your quickie benchmark actually run the loop. Just add an input statement at the front end à la:

```
INPUT J
FOR I = J TO 1000
  A(I) = I*(I+1)
NEXT I
```

The compiler doesn't know what number you're going to type in when prompted for J at runtime, so it can't calculate the array values at compile time.

Beyond the compiler itself, embedded software pros will tell you, "It's the library, stupid." The compiler relies on the library routines to handle much of the dirty work and heavy lifting. The good news is that the library can be tweaked and fine-tuned to best fit a particular application. The bad news is that it can also be tweaked to optimize a benchmark score.

Measuring the cleverness of a compiler and optimized library may be useful, but does it accurately reflect the hardware's capabilities in day-to-day applications? Other Dhrystone issues include a legacy spanning multiple versions of the benchmark over the years and a lack of standardization in the way results are presented.

EEMBC (www.eembc.org) is an outfit well regarded for their higher-level benchmark suites targeting real-world applications, such as telecom, automotive, and infotainment. Now they're introducing CoreMark, which, as the name implies, is intended as a simple measure of a processor core's basic performance capability.

The CoreMark benchmark consists of a mix of routines designed to succinctly mimic the behavior of typical programs, including list-processing (pointers), state machine (conditional branches), matrix ops (math), and CRC—the latter is also used to self-validate the runtime results. CoreMark is easy to port to any MCU with the addition of a few lines of hardware-specific timing code, and it will fit in 16 KB or so, making it suitable even for 8-bit MCUs. The benchmark is specifically hardened against unrealistic compiler optimizations and doesn't make any library calls in the timing loop.

Although new and improved in a technical sense, CoreMark does recognize the aspects of Dhrystone that have stood it well. The CoreMark code is freely available and everybody is welcome and encouraged to give it a whirl. While EEMBC will certify CoreMark results for their paying members, they encourage nonmembers to post results too, so virtually every chip is fair game. Yes, there's nothing preventing some misguided advocate from going overboard (i.e., lying through their teeth), but I suspect peer pressure and the ease of replicating (or not) results will quickly root out and banish the worst offenders. Check it out at www.coremark.org.

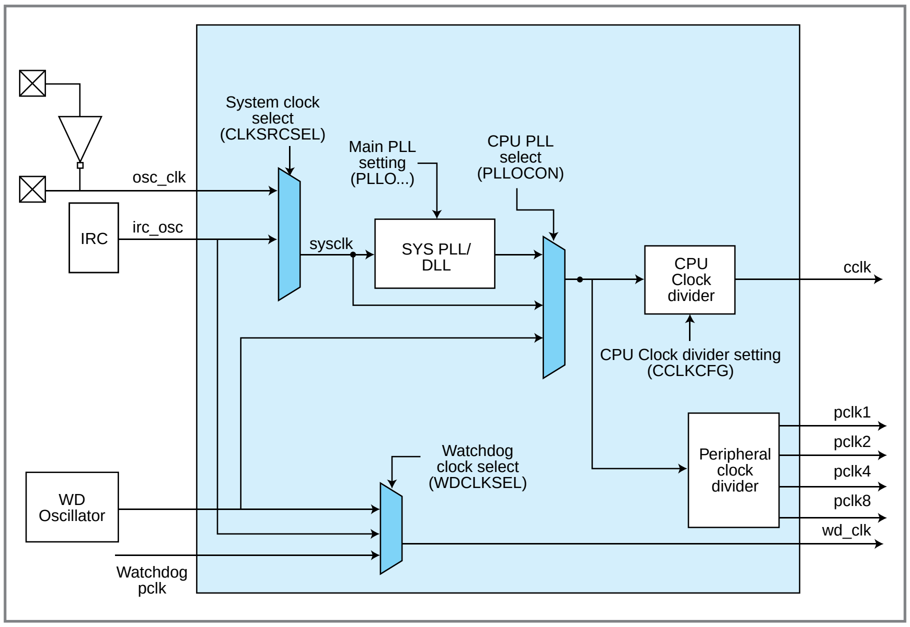


Figure 4—Clock generators play a pivotal role in power management. The LPC1100 reflects that trend with a combination of programmable and selectable clock sources that allow dynamic fine-tuning of the trade-off between performance and power consumption.

Anyway, I surely felt impressed (and lucky) to get this far this easily with the bleeding-edge setup. Nevertheless, it soon became apparent there were still some rough edges. For example, for many MCUs, the MDK has the ability to simulate I/O devices, but as you might expect nothing yet for the NXP LPC1100. The basic breakpoint and watch-point features worked, as did the ability to modify memory locations without stopping the CPU. But more advanced debug features like profiling and the MDK software "logic analyzer" weren't happening. At first, the MDK didn't know how to program the LPC1100 flash, so I was limited to running programs in the MCU's RAM. But then I received a hot-off-the-press flash driver, which worked without a hitch once installed in the MDK.

I fully expect wrinkles of a preproduction nature will be ironed out by the time LPC1100 chips hit the street. For now, I will say that the things the MDK could do were done very well. The software is fast, responsive, and stable without feeling temperamental like other high-end toolchains I've tried. There's clearly a lot of advanced functionality on tap, but as one who just dabbles in software, I appreciated not being forced to deal with it needlessly. Even the user interface, window-laden it may be, seemed easier to navigate and less fussy than others I've tried.

With time running out before deadline, I at least wanted to hang a meter on the chip to reality-check the low-power claims. I discovered what seems to be an interesting phenomenon—namely, that power consumption was significantly lower (about 25% less), running the test program in RAM instead of flash. Only a final datasheet will tell,

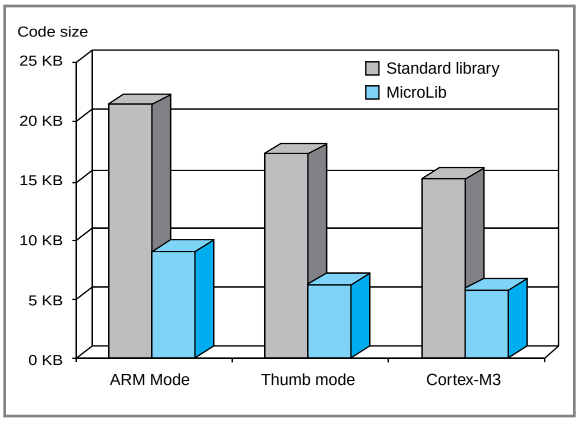


Figure 5—The latest version of the Keil MDK toolchain comes with an optional "MicroLib" C library optimized to reduce code size.

MACH64
PROGRAMMABLE LOGIC STARTER KIT
NURVE NETWORKS LLC

Based on the Lattice ispMach 4064.

Includes 250 page lab manual.

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

Learn to turn *software* into *hardware!*

www.XGAMESTATION.com

PIC-SERVO
MOTION CONTROL

MOTION CONTROLLERS FOR BRUSH, BRUSHLESS AND STEPPER MOTORS.

• controller chips
• controller boards

www.picservo.com
JEFFREY KERR, LLC

**Add USB to Your Designs**

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

**USB Complete**
**The Developer's Guide**
**Fourth Edition**
Jan Axelson

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

but putting high duty-cycle code in RAM would appear to be an effective power reduction technique.

Running flat out at 48 MHz from RAM current consumption was just 8.5 mA (see Photo 3). That's decently close to the 150 µA/MHz claim (i.e., 8.5 mA/48 MHz is 177 µA/MHz) when you consider power consumed by non-core extras (memory, I/O, clock generator, etc.). Anyway, absent a production chip, a final datasheet, and proper test software, there's no need to quibble over the as-yet-non-existent fine print. For now, it's safe to say the LPC1100 squeezes high-revving, 32-bit performance from just a few milliamps.

## MORE BITS, LESS FILLING

With the Cortex-M0, the evolution of the ARM architecture from "computer" to "MCU" is now complete. Admittedly, the journey was rather roundabout; but in this case, the destination is the reward. With chips like the LPC1100 designed by folks who "get it" when it comes to embedded apps, the much-hyped premise that 32-bit chips can truly compete for 8/16-bit MCU sockets has come true.

That's not to say 32-bit MCUs will "replace" smaller chips across the board anytime soon. The little chips have legions of followers (e.g., NXP still sells a heck of a lot of '51s) and they can still get the job done. But with 'M0 and the LPC1100, we see the first ARM MCU that can truly compete with 8/16-bit parts on every level without compromise, while at the same time offer an upgrade path to Cortex-M3 and beyond.

So hang on to your hats because we're headed back to the future. Or is it forward to the past? Either way, it'll be a fun and exciting ride. ◘

*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] D. Manners, "How ARM1 Got Built By Steve Furber," *Electronicsweekly.com*, 2007.

## SOURCES

**ARM Microcontroller development kit (MDK-ARM)**
Keil | www.keil.com/arm/mdk.asp

**LPC1100 Cortex-M0 MCU**
NXP Semiconductors | www.standardics.nxp.com/products/lpc1000/lpc11xx/

These convenient PDF archives contain complete issues, just as they appeared when they were printed. Get back on track and follow along the uninterrupted Circuit Cellar timeline.

For package deals on all of our archives, visit: www.circuitcellar.com/archives/

# MP3P DIY KIT, Do it yourself

*(Include Firmware Full source Code, Schematic)*

## • myPIC

*Only*

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

### Powerful feature

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

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

## • myWave (MP3 DIY KIT SD card Interface)

*Only*

**$150** qty 100

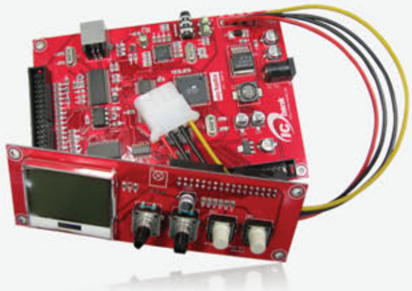
**$200** qty 1

### Powerful feature

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

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

## • myAudio (MP3 DIY KIT IDE)



*Only*

**$180** qty 100

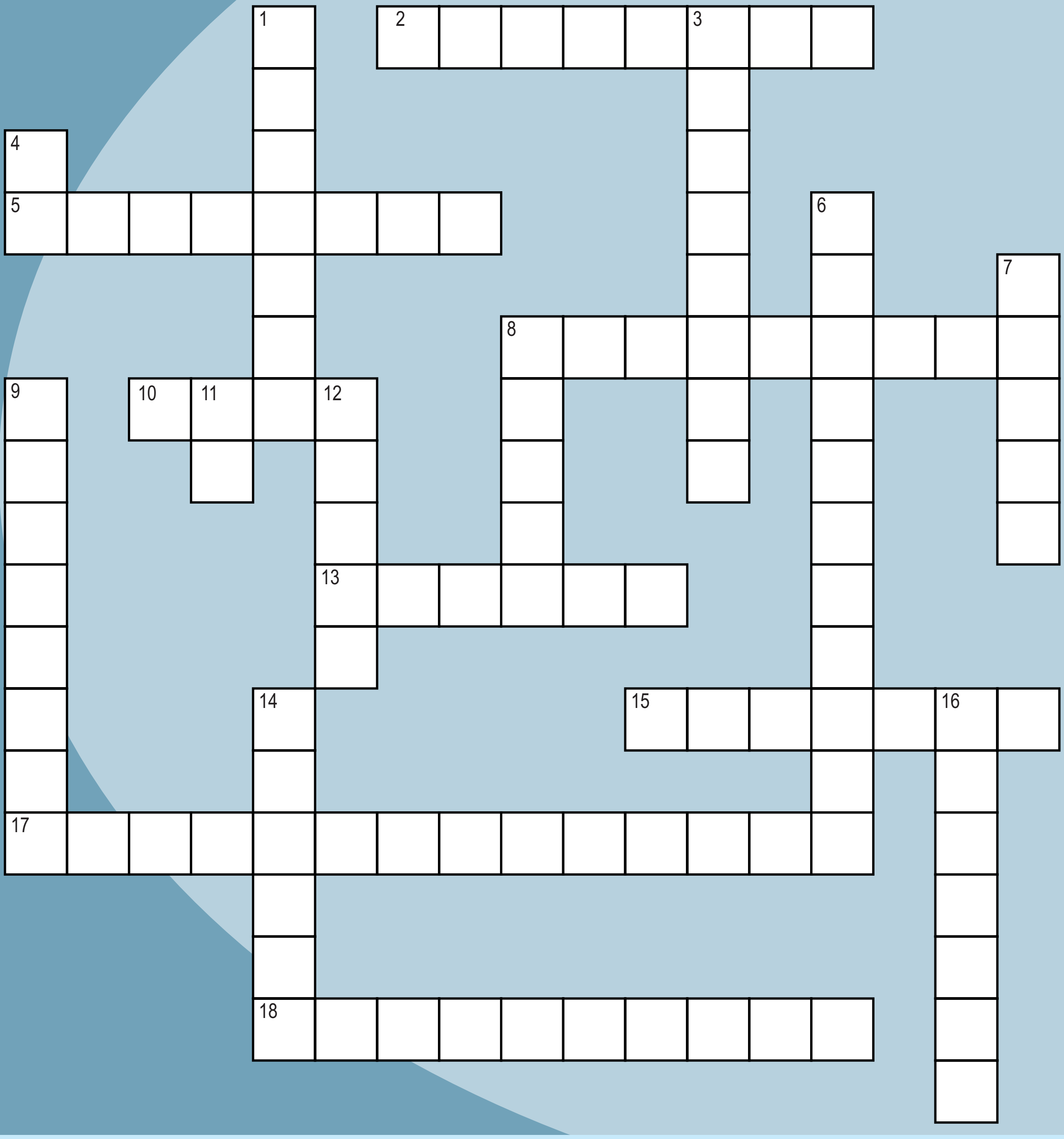
**$220** qty 1

### Powerful feature

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

www.icbank.com

# CROSSWORD



**Down**

1. Rx
3. The "+" in 5 + 6 = 11
4. Recommended Standard
6. V.5 [two words]
7. Low-power
8. Floating-point operations per second
9. The "5" and "6" in 5 + 6 = 11
11. Return to zero
12. Transmission equation
14. Operation code
16. What type of potentiometer is a D-Pot?

**Across**

2. Cat
5. sup, as in sup1,2,3 = 3
8. !
10. Reference voltage
13. Symbol ⊂
15. Antenna invented in 1926 in Japan [two names]
17. $SiO_2$ [two words]
18. X

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

# IDEA BOX

## THE DIRECTORY OF PRODUCTS AND SERVICES

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

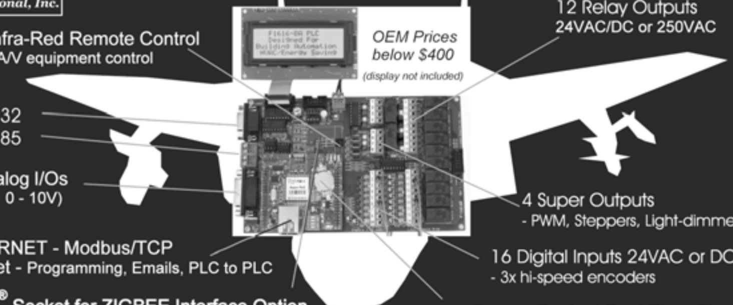
The Vendor Directory at www.circuitcellar.com/vendor/
is your guide to a variety of engineering pr oducts and services.

**The PLC with everything you ever wished for !**
Lead the Green Revolution with our Smart Buildings control model.

Triangle Research International, Inc.

*From the Programmable Logic Controller specialists who brought you the Super M-series, now .....*

***The Ultimate F-series***



Infra-Red Remote Control
- A/V equipment control

*OEM Prices below $400*
*(display not included)*

12 Relay Outputs
24VAC/DC or 250VAC

1x RS232
2x RS485

12 Analog I/Os
(12-bit, 0 - 10V)

4 Super Outputs
- PWM, Steppers, Light-dimmer

ETHERNET - Modbus/TCP
Internet - Programming, Emails, PLC to PLC

16 Digital Inputs 24VAC or DC
- 3x hi-speed encoders

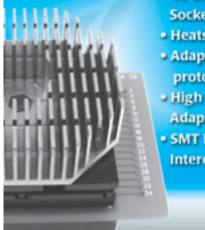
XBEE® Socket for ZIGBEE Interface Option
*XBEE is a reg. trademark of Digi International*

Battery-Backed Real Time Clock

www.tri-plc.com/cci.htm | **At prices below most other basic PLCs !** | Call 1 877 874-7527

**High Performance Sockets & Adapters**

Quick-Turn Customs



- Package Converters, SOIC/DIP, BGA/QFP, BGA/BGA and More
- 40 GHz Bandwidth BGA/QFN Sockets
- Heatsinks available up to 100W
- Adapters for probing/test/prototype
- High Volume Production Adapters-BGA, QFP and More
- SMT Package Emulation/Interconnect

RoHS

Ironwood ELECTRONICS 1-800-404-0204
www.ironwoodelectronics.com

Order online at:
www.melabs.com

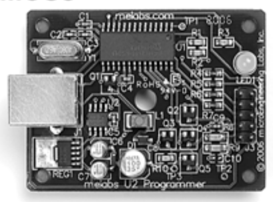
**microEngineering Labs, Inc.**
*Development Tools for PIC® Microcontrollers*

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

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

RoHS Compliant

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



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

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

**LAB-X Experimenter Boards**

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

**Pricing from $79.95 to $349.95**

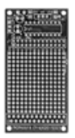
**BASIC Compilers for PICmicro®**

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

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

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

**PICPROTO™ Prototyping Boards**



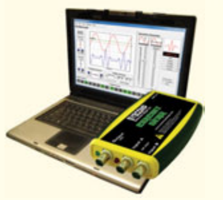
*Double-Sided with Plate-Thru Holes*
*Circuitry for Power Supply and Clock*
*Large Prototype Area*
*Boards Available for Most PIC® MCUs*
*Documentation and Schematic*

**Pricing from $8.95 to $19.95**

**Serial LCDs**
2-line x 16 $39.95
4-line x 20 $49.95
*Quantity Discounts Available!*

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

**7 in 1 Scope !**



**CircuitGear CGR-101™** is a unique new, low-cost PC-based instrument which provides the features of seven devices in one USB-powered compact box: 2-ch 10-bit 20MSa/sec 2MHz oscilloscope, 2-ch spectrum-analyzer, 3MHz 8-bit arbitrary-waveform/ standard-function generator with 8 digital I/O lines. It also functions as a Network Analyzer, a Noise Generator and a PWM Output source. What's more – its open-source software runs with Windows, Linux and Mac OS's! **Only $180**

Saelig.com unique electronics 1-888-7SAELIG info@saelig.com www.saelig.com

**DESIGN IN *HALF* THE TIME**

CCS C Compiler for all Microchip PIC® MCU

✓ Over 307 Built-in Functions for optimized code

✓ 133 Example Programs for quick start-up

**Compilers starting at $150**

www.ccsinfo.com/Design

sales@ccsinfo.com · 262.522.6500 x35

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

Looks like Flash Drive — USB — USB-DAQ — A/D - I/O - SPI - I2C BASIC programming USB Status & Control

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

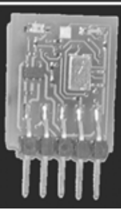
*www.hexwax.com - Mouser - Farnell - Digikey*

***Amazing*** PIC programmer

Most devices supported ICSP, SQTP, & copy limits

***$32*** at Digikey & Mouser

*www.flexipanel.com*



# ATTENTION

## PRINT MAGAZINE READERS - BONUS CONTENT NOW AVAILABLE

The following Circuit Cellar bonus content is now available for you to read online or in a downloadable PDF. Just visit Circuit Cellar's home page and click on the link to All Bonus Content.

**Issue #227: Time-Triggered Systems (Part 3)**
Tackle Data Acquisition
*Michael Smith & Lizie Dunling-Smith*

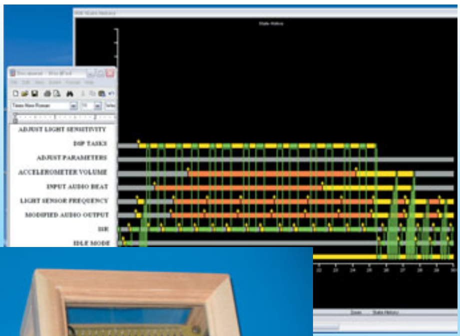
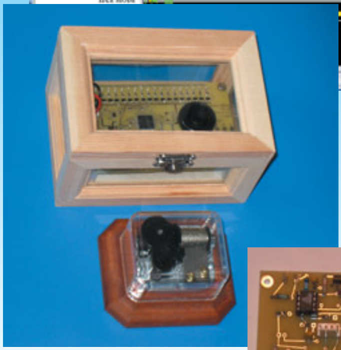
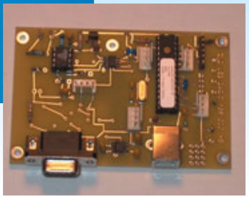
**Autonomous Vehicle Design** (Video example supplement only)
Embedded Systems, Sensor Technology, and Motor Control
*Chris Britney, Paul Green, Andy Heath, Stephen Lau, Kylee Lathrop*

**Issue #228: NimbleSig III**
A New and Improved DDS RF Generator
*Thomas Alldread*

**Sound Synthesis Made Simple** (Full article plus video example)
A Multi-MIPS Music Box
*Peter McCollum*

**Issue #229: USB I/O Expansion**
*Brian Millier*

**Issue #230: Verification and Simulation of FPGA Designs**
*Sharad Sinha*

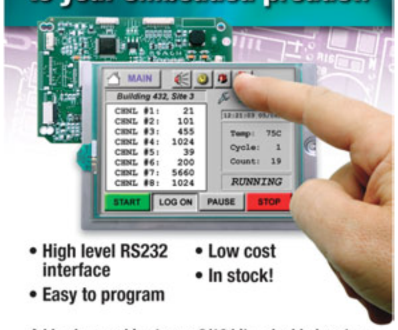




Are you interested in writing for Circuit Cellar? Consider a submission to Circuit Cellar's bonus section in the Digital Plus venue. As you see from this statement of availability, the bonus section of Digital Plus is available to all Circuit Cellar readers. Authors are choosing to be published in our bonus section for a variety of reasons. These reasons include but are not limited to:

- Articles of various lengths can be published in the digital venue
- Follow-up articles are published in the bonus section without concern for the impact on the current issue's theme
- Articles may include audio or video enhancements
- Speed to publication. Space restrictions in the print magazine can delay publication. There are fewer restrictions on the digital side.

Whether you want to submit an article for print publication or for publication in the bonus section of Digital Plus, please write to editor@circuitcellar.com to present your ideas.

Add a color touch interface to your embedded product!



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

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

REACH TECHNOLOGY INC.

www.reachtech.com • 510-770-1417

842 Boggs Avenue • Fremont, CA 94539

Solve complex signal acquisition problems...

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

Google Checkout
Fast checkout through Google

www.stx104.com

Apex Embedded Systems

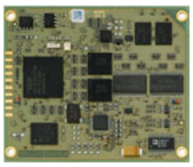
help@stx104.com • 608-256-0767 x24

PHYTEC

Inside great products. Behind great ideas.

phyCORE® System on Modules:

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



phyCORE-LPC3250

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

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

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

www.can232.com

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

Only $108 €89

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

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

Only $154 €129

www.canusb.com

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

*Versatile Programming LCD module
*USB, SPI, RS232/TTL Interfaces
*Bright 350 Nit LED Display (320 x 240)
*Integrated Touch Screen
*LUA Scripting Language capable-
For stand alone embedded apps
*Memory 3.8 MB + SD to 2G
*ezLCD's are also available in 2.7", 5.6", 6.4", 8.0", 10.4"

3.5" ezLCD
p/n:ezLCD+103

Call for Custom Display Configurations

www.earthlcd.com

I²C™/SMBus

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

MCC
Micro Computer Control

I²C is a trademark of Philips Corporation

www.mcc-us.com

T-Box™

Low cost Temperature Data Acquisition and Control

NEW!

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

$89
OEM board

TERN INC.

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

AVOCET SYSTEMS, INC.
www.avocetsystems.com

Microcontroller Development Tools

68000 68010 683xx
68HC11 68HC12
68HC16 Z80/EZ80
6502 65816
PIC16

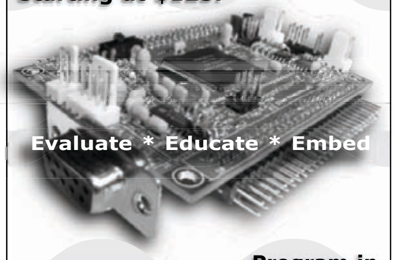
Macro Assemblers
C Compilers
Software Simulators

(207) 596-7766
sales@avocetsystems.com

**Adapt9S12XDP512**
**Modular Prototyping System**

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

**Starting at $125!**



**Program in Assembler, BASIC, C, and Forth**

**www.TechnologicalArts.com**

**ALL ELECTRONICS CORPORATION**

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

**www.allelectronics.com**
***Free 96 page catalog***
***1-800-826-5432***

**PRINTED CIRCUIT BOARDS**
QUALITY PRODUCT
FAST DELIVERY • COMPETITIVE PRICING

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

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

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

PROTOTYPE THROUGH PRODUCTION

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

**techsol**
TECHNICAL SOLUTIONS INC

**TPC-43A**

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

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

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

**From $339**

**www.medallionsystem.com**

**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

**$129 Development System**

**Development Kit Includes:**
- 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**

**Weather Instruments for PCs**

AAG electrónica

**www.aagelectronica.com**

***It writes your Full Speed USB Code!***

***NEW!* HIDmaker FS for Full Speed FLASH PIC18F4550**

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

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

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

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

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

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

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

HIDmaker FS Combo: Only $599.95

**Trace SYSTEMS, Inc.**

DOWNLOAD the HIDmaker FS Test Drive today!
www.TraceSystemsInc.com
301-262-0300

**I2C SPI 1 Wire**

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

**www.i2cchip.com**

# INDEX OF ADVERTISERS

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

| Page | | Page | | Page | | Page | |
|---|---|---|---|---|---|---|---|
| 78 | AAG Electronica, LLC | 15 | ExpressPCB | 77 | Lawicel AB | 34 | ProlificUSA |
| 32 | AP Circuits | 41 | ezPCB | 17 | LeCroy | 78 | Pulsar, Inc. |
| 31 | ARM techcon$^3$ Conference | 76 | FlexiPanel Ltd. | 9 | Lemos International Co. Inc. | C3 | Rabbit, A Digi International Brand |
| 78 | All Electronics Corp. | 37 | Futurlec | 10 | Linx Technologies | 77 | Reach Technology, Inc. |
| 77 | Apex Embedded Systems | 37 | Grid Connect, Inc. | 77 | MCC (Micro Computer Control) | 63 | SoC Conference |
| 7 | Atmel | 40 | HobbyLab, LLC | 18 | MachinePIER | 76 | Saelig Co. |
| 77 | Avocet Systems, Inc. | 78 | I2CChip | 11 | Microchip Technology, Inc. | 29 | Spark Fun Electronics |
| 33 | CWAV | 47, 73 | ICbank Inc. | 75 | microEngineering Labs, Inc. | 78 | Technical Solutions, Inc. |
| 21 | CadSoft Computer, Inc. | 40 | Intuitive Circuits LLC | 2 | Mouser Electronics | 56, 57 | Technologic Systems |
| 58 | Comfile Technology, Inc. | 75 | Ironwood Electronics | C2 | NetBurner | 78 | Technological Arts |
| 76 | Custom Computer Services, Inc. | 32, 34 | JKmicrosystems, Inc. | 71 | Nurve Networks LLC | 77 | Tern, Inc. |
| 40 | DesignNotes | 78 | JKmicrosystems, Inc. | 9 | PCBCore | 19 | Texas Instruments |
| 10 | EMAC, Inc. | 5 | Jameco | 18 | PCB-Pool | 1 | Total Phase, Inc. |
| 77 | Earth Computer Technologies | 71 | Jeffrey Kerr, LLC | C4 | Parallax, Inc. | 78 | Trace Systems, Inc. |
| 49 | Elsevier | 3 | Keil Software | 77 | Phytec America LLC | 75 | Triangle Research Int'l, Inc. |
| 25 | Embedded Developer | 71 | Lakeview Research | 48 | Pololu Corp. | 42 | WIZnet |

## PREVIEW of November Issue 232

Theme: **Analog Techniques**

**Precision By Encoder:** A Precision Linear Encoder Display

**Energize a Circuit:** An Innovative Switched-Mode Power Supply Design

**How to Repurpose a Development Platform**

**Inexpensive Vehicle Locator:** Create an Embedded Linux-Based Controller

LESSONS FROM THE TRENCHES **Passing Parameters:** The Basics of Passing Parameters and Design Partitioning

FROM THE BENCH **Managed Devices and SNMP:** A Simple Network Management Protocol

SILICON UPDATE **Power Pitcher:** Wireless Power on a Microelectronic Scale

## ATTENTION ADVERTISERS

**December Issue 233 Deadlines**

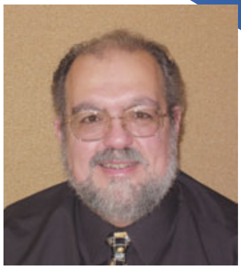
Space Close: Oct. 13
Material Close: Oct. 20

**Theme**
Programmable Logic

**Bonus Distribution**
SoC Conference

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

# PRIORITY INTERRUPT



by Steve Ciarcia, Founder and Editorial Director

## A Broadening Experience

OK, I'll admit that my technical interests seem schizophrenic at times, but I contend that all engineers have their quirks. I just get to play more of mine out in public so I seem quirkier. I have a cell phone, but I don't want voicemail or too many "smarts" (yet). I want a computer-controlled house, but I don't want an invasive intelligent media system anymore. (Been there, done that.) I want Internet connectivity everywhere, but I don't want social networking. (Call me old-fashioned.) I want the whole world to be wireless, but I still string copper wires for every new home-control signal. (Call me insecure.) ;-)

Regardless of the fact that I prefer bulletproof hardwired connections on my fixed-location home control system (HCS), virtually all mobile-computing applications these days are wireless. Like the ubiquitous dial-up modem of the past, all laptops come with built-in Wi-Fi that makes reading e-mail and local news at the breakfast table the new cultural norm. Extending mobility to include constant social network connection, sports score updates, and casual browsing away from home territory just involves linking to cheap Wi-Fi hotspots or a private mobile broadband network.

I got mobile broadband two years ago as a defensive measure (smart phones typically use a mobile voice/data plan, whereas I'm talking about a mobile Internet connection). We have a timeshare condo in Newport, Rhode Island, and like most tourist traps, it can be rip-off city, especially when it comes to Wi-Fi. Don't get me wrong, the condo, the city, and virtually every place in the town has Wi-Fi—at $10 or more per day! The only free Wi-Fi in town was at Panera Bread, where the Wi-Fi was free but occupying the table imposed a different obligation. In protest, I drove over to the Sprint store in Newport and signed up for mobile broadband. Here's what I've learned during the last two years.

Not every mobile connection is broadband. If it isn't 3G (third generation), then it really isn't "broadband." The three major U.S. networks—Verizon, AT&T, and Sprint—are all 3G and their two-year subscription cost is the same. Beware of the advertising hype when anyone starts talking about coverage and download speeds. Depending on the position of the moon, what you had for lunch, the number of iron train bridges in the area, and other stupid stuff like that, any one of the mobile providers can appear to have the highest speed during a particular test. For 3G, they all seem to be approximately equal under optimum conditions. Of course, out there in the real world, results may vary; but in the end, download speed is simply a function of how close you are to a cell tower.

For the most part, about 1 Mbps is what you can expect in a good signal area. I remember signing on and getting 3.2 Mbps one time, but on average I see 1 Mbps in highly populated areas. The farther you are from the cell tower, the worse it gets—and I've experienced connection speeds of less than 100 kbps at times. If you need wide-area mobility along with high data rates all the time, you may be out of luck with current technology and coverage maps.

Mobile Internet for your embedded application or laptop sounds like a solution made in heaven, but rest assured there is always a "gotcha" that reinjects fire and brimstone into the equation. Unlike smart phones, mobile Internet plans currently limit the amount of data downloaded to 5 GB/month before they start adding charges. (I checked my plan and it seems to still say "unlimited download," for whatever that's worth.) This may seem like a lot of data until you start using your laptop like it's still tied to your landline ISP. For example, while e-mails are typically 3 KB, the average webpage is 150 KB, a downloaded song is 4 to 8 MB, and a non-HD one-hour movie is about 800 MB. Going over the 5 GB limit can be hazardous to your wallet.

Depending on the provider, the overcharge can range from barely reasonable to absolutely absurd. Sprint charges $0.05 per megabyte over the 5 GB limit. One gigabyte of extra data will cost you $51.20 over the $60/month regular cost. Based on their current pricing, the same additional gigabyte on Verizon would cost $256 and on AT&T it would be an unbelievable $503.31. Watching Hulu during the morning train commute can get very expensive very fast!

Finally, to add insult to injury, all providers seem to have fine print saying they can reduce broadband speed if they think you are using too much bandwidth (especially for us unlimited download guys) or for certain kinds of data transactions (i.e., VoIP). Heaven forbid you aren't paying them another $30 to $50 a month for regular cell phone service and try sneaking your phone calls through their towers using Skype. My guess is that video streaming is also on their poison list someplace too.

So, this isn't a rant or a testimonial for any particular ISP. It's merely a statement that mobile broadband is something I have, and it has been useful to some extent. I suppose there will be a day when I can no longer watch Hulu while the wife is shopping, or when I can't leave the HCS webcams streaming data for hours, but I'll remember that it was fun while it lasted. The irony is that you sign up for all these services based on claims of unlimited benefits, and then after you do, the services get rationed. I guess that it was too much to hope that devious tactics like these were limited to government.

steve.ciarcia@circuitcellar.com

Steve

# Sweet!

## Introducing the MiniCore™ Series of Networking Modules

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

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

**Wi-Fi and Ethernet Versions**

**MiniCore Module Development Kits**

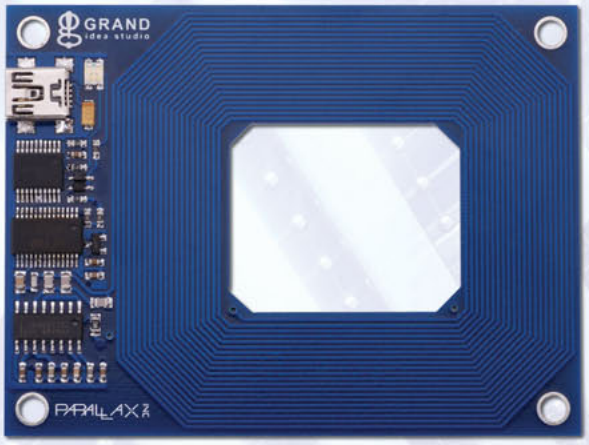
**From $99** Limited time offer.

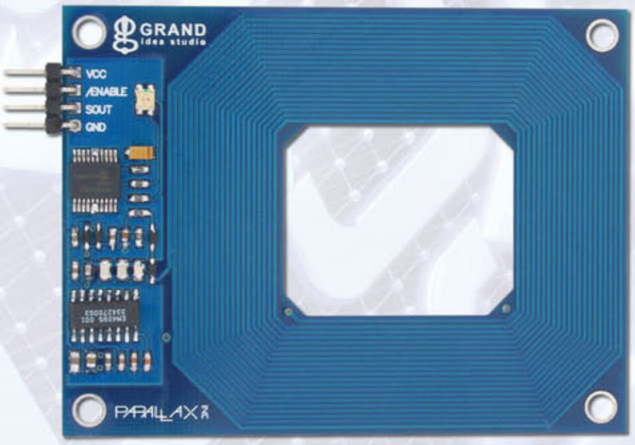
Buy now at: **1.888.411.7228 • rabbitwirelesskits.com**

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

RABBIT™

09009





# RFID READER MODULES

Designed in cooperation with Grand Idea Studio, our (RIFD) Reader Modules are convenient low-cost solutions to read EM 4100 low-frequency (125 kHz) passive RFID transponder tags. The RFID Reader Modules can be used in a wide variety of hobbyist and commercial applications, including access control, automatic identification, robotics, navigation, inventory tracking, payment systems, and car immobilization. Board dimensions are 3.25 x 2.45 in (8.25 x 6.22 cm).

**RFID Reader Module - USB** (#28340; $39.99) - This module connects directly to a USB port both for power and to output transponder tag data to your custom PC application. A sample Visual Basic (VB.net) application is provided on the 28340 product page. *Features include:*

- Built–in USB interface for easy connection to your PC
- 2400 baud serial output stream to PC is accessible from any programming language that can open a COM port
- Transponder tag ID data given as a 12-byte ASCII string
- Bi-color LED for visual indication of activity
- Power required: 5 V (USB connection); 10 mA idle, 90 mA active

Note: a USB A to Mini-B cable is required, sold separately (#805-00006; $3.99). Several transponder tag styles are available from Parallax, www.parallax.com.

**RFID Reader Module - Serial** (#28140; $39.99) - This 4-pin module is designed for easy prototyping and integration into stand-alone microcontroller-based applications. Parallax provides sample code in PBASIC and SX/B on the 28140 product page. *Features include:*

- 4-pin 0.1 inch SIP header for power, ground, enable, and data-out connections
- 2400 baud TTL serial output (81N) to any Parallax microcontroller
- Transponder tag ID data given as a 12-byte ASCII string
- Bi-color LED for visual indication of activity
- Power required: 5 VDC; 10 mA idle, 90 mA active

We also offer the RFID Reader Modules as part of sampler kits which include an assortment of four RFID transponder tags.

**RFID Reader (USB) and Tag Sampler Kit** (#32395; $42.99; right)
**RFID Reader (Serial) and Tag Sampler Kit** (#32390; $42.99)

Order **RFID Reader Modules** at parallax.com or call our Sales Department toll-free at 888-512-1024 (Monday-Friday, 7 a.m. - 5 p.m., PDT).

Prices are subject to change without notice. Parallax, and the Parallax logo are trademarks of Parallax Inc.

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

CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

BONUS ARTICLE

by Sitti Amarittapark

# Buddy Memory Manager

When memory matters but you don't have the proper support for dynamic memory management, you may find the Buddy Memory principle to be the perfect solution. This article covers a useful memory management library based on the Buddy Memory principle. With a thorough understanding of the principle, you'll be able to put the library to good use.

Most of the firmware development toolsets for microcontrollers provide little or no support for dynamic memory management. The memory buffers must be statically defined when the firmware is written, which makes it difficult to write firmware that can change the memory usage based on the workload.

In this article, I'll describe a C language implementation of a stand-alone dynamic memory-management library based on the Buddy Memory principle. The library can manage memory space of any size as long as the memory space can be linearly addressable in C language. After I describe the concept of the Buddy Memory and the library implementation, I'll present an example of how to adapt the library to a specific processor architecture.

## DYNAMIC MEMORY MANAGEMENT

A dynamic memory-management library is the keeper of a contiguous piece of memory space (*heap*) for the rest of the system. The application code can request (*allocate*) a piece of memory from the library for temporary usage. The library guarantees not to give out the same piece to anyone until the piece is returned back to the library (*freed*). The number and size of memory pieces that are allocated from the heap can change over time and may appear in any sequence. The library's task is to efficiently manage the heap so that it can maximize the usage of the heap.

## DIVIDE BY HALF

Imagine taking a block of 16 bytes from a 256-byte block located at address 0. One way to do it is to divide the 256-byte block by half, but the new two blocks of 128 bytes still would be too big. So, you repeat the step by taking one of the two identical blocks and keep dividing it by half again and again until you get a pair of 16-byte blocks, from which one of the blocks can be taken. Figure 1 shows the steps involved.

## POOL OF BLOCKS

After a 16-byte block request is taken away, the heap is left with blocks between 16 to 128 bytes. These blocks form a pool of free blocks from which subsequent memory allocation will come. From the example in Figure 1, suppose two more 16-byte blocks are needed. The first can be taken from an existing 16-byte block in the pool. The other can come from a 32-byte block when the block is divided by half. Refer to Figure 2 for the steps to take

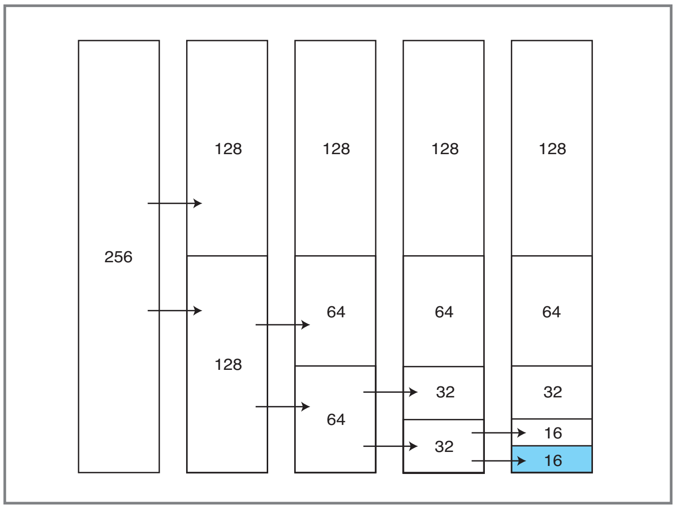


Figure 1—Taking a 16-byte block from a 256-byte space

two more blocks from the pool.

The previous two examples demonstrate a memory allocation method: Look for a block of the exact size you need. If one doesn't exist, continue looking for a bigger one. When one is found, divide the block by half until reaching the needed size. The unneeded blocks are kept in the pool of free blocks.

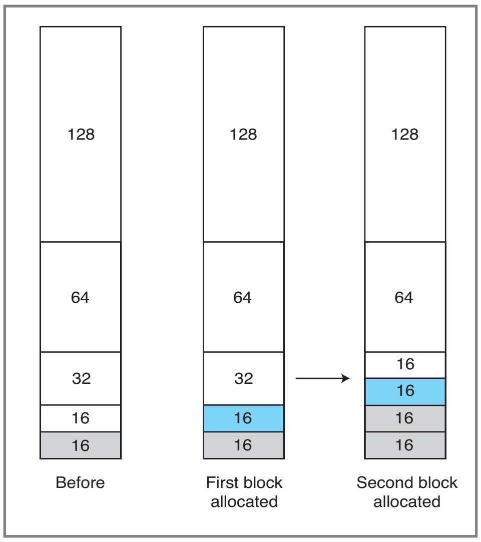


Figure 2—Taking additional two blocks from the 256-byte space

## MERGE BLOCKS

When a block is freed, the library puts it back into the memory pool so that it will be reused in a subsequent allocation. There is, however, an additional step to do. Imagine that 16 blocks of 16 bytes are allocated from a heap of 256 bytes. After all 16 blocks are freed, if nothing else is done, the heap will become a pool of 16-byte blocks. This makes it impossible to allocate a block of 64 bytes, even if there is enough space in the heap.

The additional step is to reverse the process done during allocation. In the example shown in Figure 1, only one 16-byte block is allocated from the space of 256 bytes. Freeing the block is done by merging the 16-byte block to the adjacent block of identical size in the pool to produce a 32-byte block. This 32-byte block, in turn, is merged with an adjacent 32-byte block to produce a 64-byte block. This process continues until you get back the original 256-byte block. Basically, two adjacent and equal-size free blocks that were previously apart during allocation are merged into a larger block (see Figure 3).

## BUDDY MEMORY SYSTEM

Not all pairs of adjacent and equal-size blocks of memory should be merged. The pair must also come from the same block, which was divided during a previous allocation. From the example shown in Figure 2, suppose that one more block of 16 bytes is allocated (see Figure 4). A total of four 16-byte blocks (at addresses 0, 16, 32, and 48) are allocated from a 256-byte space. The block at address 16 is freed first, but there is no free block around it, so it is left in the pool of free blocks. Next, the block at address 32 is also freed. This block is adjacent to the previous block, which is of equal size and is also free. However, the two can't be merged because they were not divided from the same block (see Figure 2). As a result, both blocks will have to be kept independently in the pool until the one at address 0 or 48 is freed. The one at address 0 can merge with the one at address 16. Likewise, the one at address 32 can merge with the one at address 48. The blocks at addresses 0 and 16 are a *buddy* pair and so is the one at addresses 32 and 48.

The buddy memory system is based on a symmetrical operation of dividing large blocks down by half during allocation and merging pairs of small blocks to form bigger blocks during free. This means memory blocks of all sizes, except the smallest one, must be divisible by two. This makes the integral power of two to be the obvious choice for block sizes.

To this point, I've explained the general principles of the buddy memory system. Now I'll focus on specific techniques for implementing the buddy memory concept in an embedded system.

## DATA STRUCTURE

The library's internal data structure consists of an array of pointers and a heap. Each pointer in the array is dedicated to a particular memory block size in the heap.

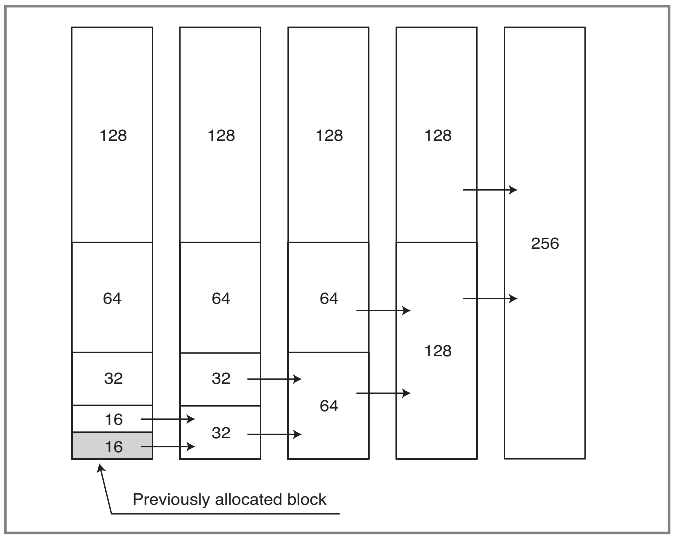


Figure 3—Merging blocks together

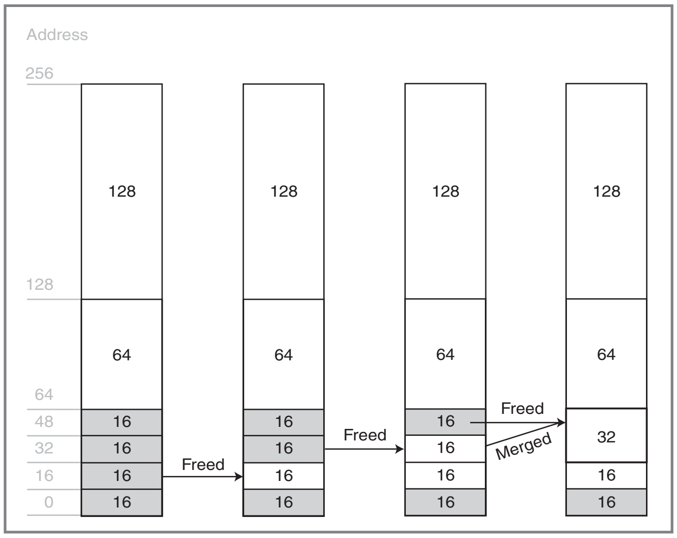


Figure 4—Merging buddy blocks together

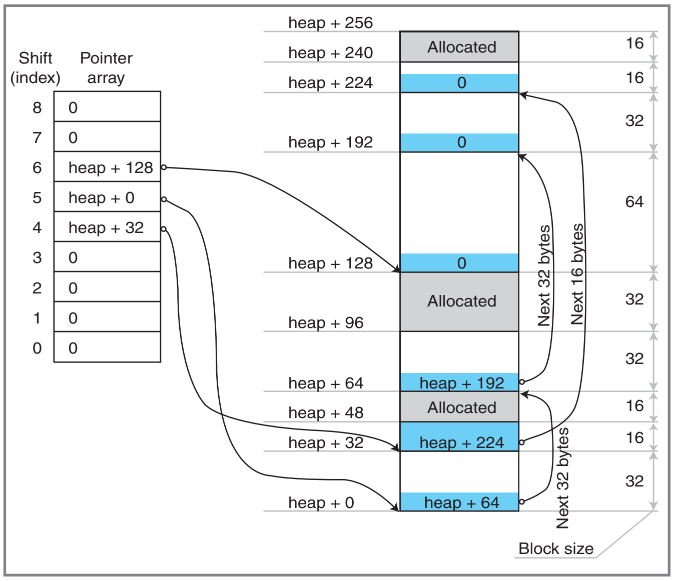


Figure 5—A linked list of free blocks in a 256-byte heap

A pointer in this array points to a free block in the heap, which, in turn, contains another pointer pointing to another free block. The pointer in the free block resides in the first few bytes (header) of the block. This forms a one-way linked list of free blocks for a particular block size. The linked list terminates with a null pointer (value = 0) at the last free block. The blocks in the linked list are sorted by address in ascending sort order. This data structure provides the library a direct access to all free blocks of all sizes from the pointer array.

To cover all the block sizes, the number of pointers is equal to N + 1, where $2^N$ is the size of the heap. For example, a system with a 256-byte heap needs pointers for block sizes 1, 2, 4, . . . 256, or $(2^0) – (2^8)$. The total number of pointers in the array is 8 + 1 = 9 (see Figure 5).

## BLOCK HEADER

A block contains a multiple-byte header at the beginning of the block. The header's content and size depend on the block's status. As I've already stated, a free block's header contains an address pointer linking to the next free block. In this case, the header is as large as the size of a pointer. If a block is allocated, the header contains a number that indicates the size of the block in a power of two. A 32-byte block's header contains number 5, because $2^5$ equals 32, for instance. I refer to this kind of number as *shift* because programmatically shifting 1 to the left by this number produces the block size. For example, 1 << 5 produces 32.

The header size, when it contains a pointer, depends on the system's address space. A pointer for a 16-bit address space occupies 2 bytes, while one for a 32-bit address space requires 4-byte space. To contain a shift, one byte would be enough. After all, 2 << 0xFF is a large number. However, the size of the header should be an even number so that the address next to the header will be an even address.

Because a block has to be able to contain a header, the header size of a free block limits how small a block can be. Between two statuses of a block, the header for a free block, containing a pointer, will be equal or bigger than one with a shift. Therefore, the size of a pointer limits how small a block the system may have.

| Block size | Block address | Buddy block address (exclusive-OR result) |
|---|---|---|
| 0x10 | 0x20 | 0x30 |
| 0x10 | 0x30 | 0x20 |

Table 1—Buddy block address calculation

## ALIGNMENT & CALCULATION

In order to free a block, the library needs to know the address of the block's buddy (see Table 1). The buddy's address is calculated by exclusive-OR the block size to the address of the block.

Since exclusive-OR is symmetrical, you can calculate one buddy address from another address regardless of which one of the buddy addresses you start from. For example, an exclusive-OR block size of 16 (0x10 hexadecimal) to address 32 (0x20 hexadecimal) produces address 48 (0x30), while exclusive-OR 0x10 to 0x30 also produces 0x20.

## ACTUAL BLOCK SIZE

During allocation, the requested block size, plus the size of a block header, is rounded up to the nearest binary number. This is the size that will be allocated from the pool. After a block is identified, the library writes the block size into the header at the starting address of the block; it then offsets the block address by the size variable before returning the result address. For example, if the library receives a call to allocate 22 bytes. The library adds the size of header (say, 2 bytes) to 22. So, the block has to be 24 bytes or larger. After rounded up, 24 become 32. When the library allocates a 32-byte block at 0x80, the library puts the header information into address 0x80 and returns the address 0x82 back to the caller. When the library frees the block at address 0x82, it reads the actual size from the block header at the address 0x80. Based on this number the library puts this block back into the free list. The actual size of header depends on the application (see Figure 6).

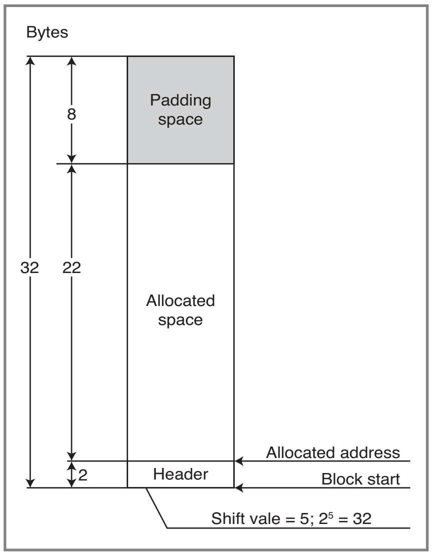


Figure 6—An allocated block

By embedding the size into the

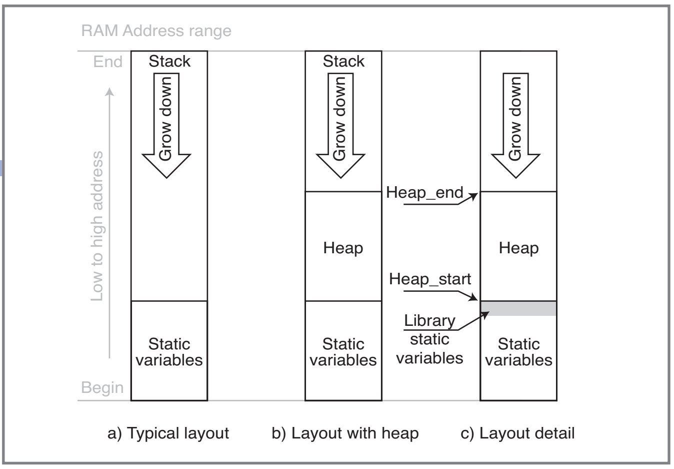


Figure 7—Memory layout

block itself, the allocated block is always bigger than what the application asks for (see Figure 6). If the requested size is an integral power of two, the actual space for the block will be twice as big as the requested size.

## MEMORY LAYOUT

The buddy memory system shares the system RAM with other purposes. In a typical embedded system without virtual address space, the system RAM is divided between static variables (static) and stack (dynamic). Refer to Figure 7a. The static variables include the static constants and static variables that the compiler decides to place into RAM, including the global variables and non-constant static variables defined in functions.

In most computer architectures, the stack space grows from a high to a low address, so the stack space is arranged to start from the highest RAM address at system reset and grows dynamically down to the lower RAM address. The compiler places static variables at low addresses and grows toward high addresses. These two kinds of memory usage spaces are placed at the opposite end of the RAM space to minimize the possibility that the stack grows over the static variables (see Figure 7a).

The obvious location for the heap is between the statically defined memory and the stack space (see Figure 7b). The buddy memory is implemented as a library with some static variables of its own. When the library is linked with the rest of the application, the library's variables must be the last set of variables to be linked so that its static variables locate at the highest address of all static variables in the system (see Figure 7c). The remaining RAM space between the library's static variables to the end of the RAM address space is devoted to the dynamic use of the system, which includes the heap and stack. The heap space occupies the lower address compared to stack space, but there is no implicit boundary between them. You must specify the size of the heap (and stack space). During run time, the heap space must not be violated. An overgrown stack will override the heap's information and corrupt the heap.

## HEAP INITIALIZATION

Before the memory manager can start allocating blocks, the heap and its internal variables must be initialized. The heap space begins just above the last static variable defined in the system and ends at the location defined by the user. As a result, adding or removing static variables may change the heap's starting address and size. The buddy memory, however, requires that the block size must be an integral power of two and located at the address that is divisible with its size. This makes it necessary to initialize the heap to fit into the available space.

Let's consider a system with addressable RAM in the 0x0000–0x0FFF range. Of the total 4-KB space, the static variables take up the first 1.25 KB (0x500) and the stack takes the high address of 0.25 KB (0x100). This leaves 2.5 KB (0x0A00) in the middle at the 0x0500–0x0EFF address range that can be used for the heap (see Figure 8a). The heap size is not a $2^N$ number. The biggest $2^N$ sized block aligned to the address in the available space is a 1-KB block between 0x0800–0x0BFF. However, simply using only this block for heap means that the spaces at 0x0500–0x07FF and 0x0C00–0x0EFF are wasted.

The solution is to format the available space into blocks, each with size $2^N$ and aligned to the address. Starting from the heap beginning address, the library computes the biggest block in the heap that fits the size-address restriction and puts it into the memory pool. This process repeats on the remaining space until the

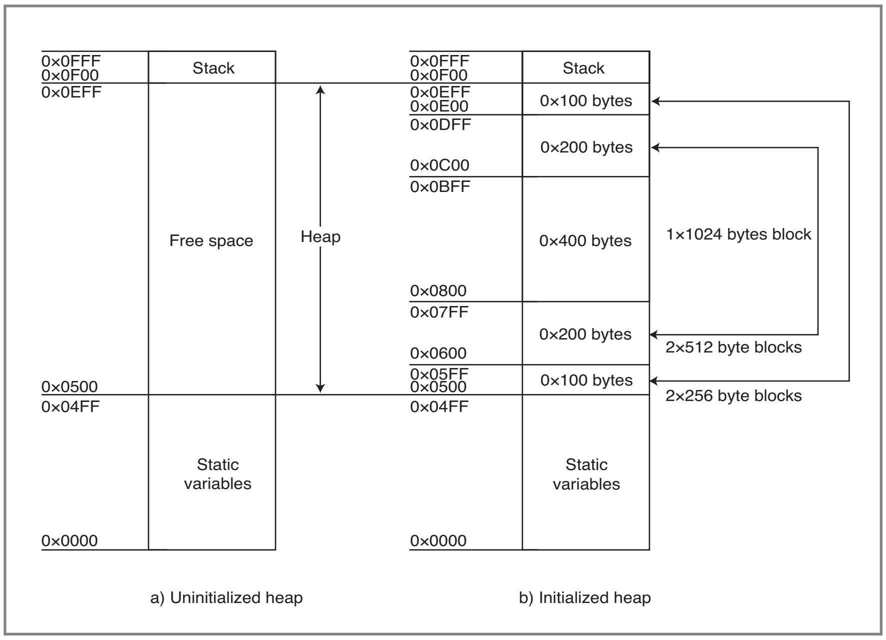


Figure 8—Initializing a heap

Listing 1—The heap's data types

```
// Definition of free list linked list
typedef struct _MB_free_list_s {
  struct  _MB_free_list_s*    next;
} _MB_free_list_t;

// Definition of the library's global variable
typedef struct {
        ptr_t          heap_start;// Actual heap start address
        ptr_t          heap_end;  // Actual heap end address
                                  // Pointer array
                                  // (free lists entry points)
        _MB_free_list_t free_list[MAX_BLOCK_SHIFT + 1];
        mem_size_t     startpoint;// First byte of heap space
} _MB_global_t;
```

Listing 2—The `_MB_init_global` function

```
static void
_MB_init_global(mem_size_t  heap_start, mem_size_t  heap_end)
{
  mem_size_t  heap_space;
  mem_size_t  block_size    = 0;
  shift_t     shift_highest = MAX_BLOCK_SHIFT;
  shift_t     shift;

  // Erase the free list
  memset((ptr_t)&gGlobal, 0, sizeof(gGlobal));

  // Partition the total heap space into free blocks
  // - Started with the total heap size and . . .
  // - Continue as long as the heap space is big enough
  // - At the end of each loop, reduce the heap space by
  //   the size of the previous free block taken away.
  for (heap_space = heap_end - heap_start;
     heap_space >= MIN_BLOCK_SIZE;
     heap_space -= block_size) {

     // Reduce the comparison to the biggest shift size
     // possible for the remaining heap space
     for (;
          SHIFT_TO_SIZE(shift_highest) > heap_space;
          --shift_highest) {
          // empty
     }

     // Find the correct shift for this address or return
     for (shift = shift_highest;
        (MASK_OF(shift) & heap_start) != 0;
        --shift) {
        if (shift < MIN_BLOCK_SHIFT) {
          // We get here because the remaining heap
          // size is big enough for a block, but
          // the address isn't fit into smallest
          // shift
          return;
        }
     }

     // Put the block into the free list
     _MB_add_to_free_list((ptr_t)heap_start, shift);
     // Move the remaining heap_space up
     block_size = SHIFT_TO_SIZE(shift);
     heap_start += block_size;
  }
 }
```

entire heap space is divided and added into the memory pool. Applying the method to the heap at 0x0500–0x0EFF yields the result as shown in Figure 8b.

## SOURCE CODE

The interface to the buddy memory library consists of three public functions to initialize the heap, allocate memory, and free memory. The heap must be initialized once before the library can be used. Let's look at the important parts of the three functions.

The `MB_INIT` function qualifies the heap address range and initializes the heap. The function takes two inputs: `heap_start` and `heap_end`. If the two values are not identical, both the `heap_start` and `heap_end` are absolute addresses. If the two numbers are identical, `heap_start` will be replaced by the `gGlobal.startpoint` address, which is defined when the library is linked (see Listing 1). This variable is the last static variable. `heap_end` must be the address just 1 byte above the last byte of the heap. These two input values are passed down to `_MB_init_qualify_request`, which validates and converts the two numbers into the absolute starting and ending addresses. Padding space is added to align the start address to the smallest block size. The address range result from this routine is passed down to the last step in `_MB_init_global`.

`_MB_init_global` partitions the heap's address range into binary size blocks. The function identifies the largest binary block that aligns to the starting address of the heap. This biggest block is added into the list (for the size) of free block. The process repeats until the remaining space is partitioned (see Listing 2).

## ALLOCATION

Allocating a piece of memory from heap is done in three steps. First, translate the requested block size to the shift number. You begin by adding the space for the header to the requested size. Then

_MB_size_to_shift rounds the number up to the nearest shift number. The size can't be too big (return NULL at line 14).

Next, a freed block is retrieved from a free list. Look for a free list starting from the exact size and going up until a free list is found (lines 22–27). If you find one, trim the block down to the requested size by breaking the block into halves and putting the higher address back into a free list (lines 45–49). Lastly, at line 53–59, the routine puts the shift number into the header, adjusts the block address, and returns the block to the caller (see Listing 3).

The MB_Free function frees a memory block (see Listing 4). The block's address is validated (lines 7–13) against the known valid heap's address range. If the block address is valid, the actual block's address is recovered (line 16) before the block is added back into the pool.

Listing 3—The MB_Alloc function

```
1   void *
2   MB_Alloc(mem_size_t req_size) {
3     ptr_t        mem = NULL;
4     shift_t      req_shift;
5     shift_t      shift;
6
7     // 1. Translate requested size to shift
8     //    1.1 Include the space for the size variable
9     req_size += sizeof(shift_t);
10    //    1.2. Size to shift (power of 2 value)
11    req_shift = _MB_size_to_shift(req_size);
12    if (req_shift == 0) {
13       // too big
14       return NULL;
15    }
16
17    // 2 Get a free block of the size
18    //    2.1 Find a non-empty free list containing
19    //    the smallest free block that can fit this
20    //    req_size. This free block could be bigger
21    //    than the req_size.
22    for (shift = req_shift;
23       (shift <= MAX_BLOCK_SHIFT)
24       && (gGlobal.free_list[shift].next == NULL);
25       ++shift) {
26         // empty
27    }
28    if (shift > MAX_BLOCK_SHIFT) {
29       // Run out of memory for this req_size
30       return NULL;
31    }
32
33    //    2.2. Remove the block from the free list
34    {
35       _MB_free_list_t*    addr =
36         gGlobal.free_list[shift].next;
37       gGlobal.free_list[shift].next = addr->next;
38
39       mem = (ptr_t)addr;
40    }
41
42    //    2.3 If the block is bigger than the req_size,
43    //    reduce it down to the req_size. The extra
44    //    space goes back into the pool.
45    for (--shift; shift >= req_shift; --shift) {
46       // Put the excess capacity back into the free list
47       ptr_t   buddy = (ptr_t)BUDDY_OF(mem, shift);
48       _MB_add_to_free_list(buddy, shift);
49    }
50
51    // 3 Add the size information into the block and
52    //   adjust the return pointer.
53    {
54       // Prefix the shift number into the memory block
55       shift_t*    addr = (shift_t*)mem;
56       *addr = req_shift;
57       addr += 1;
58       mem = (ptr_t)addr;
59    }
60
61    return mem;
62  }
```

## ADD TO THE POOL

The _MB_add_to_free_list function adds a block back into the memory pool. The inner for loop (lines 15–25) locates the buddy of the block. Each time in the loop has four possible outcomes. If the buddy is found, the loop terminates and moves on to the next statement (line 30). If the block address is bigger than the current one in the list, keep on looking (line 24). The list is sorted in ascending order. If the block address is smaller than the current one or reaches the end of the list (lines 16–23), the function inserts the block into the list and the function ends.

In lines 30–31, the buddy is removed from the list and combined with the block so that the block size is doubled. This new block is passed to another iteration of the for loop at lines 6–32. If the shift size is larger than the maximum limit, the function returns an error (see Listing 5).

## EXAMPLE

In order to use the library in an actual application, customization procedures are required to change definitions for the library, change the object file's link order, and add a library initialization to the application source code. Now I'll describe some hardware, walk you through the customization process, and present the resulting memory layout and statistics.

Listing 4—The MB_Free function

```
int
MB_Free(void *mem)
{
  shift_t*     addr;

  // Sanity check
  if (mem < gGlobal.heap_start) {
    return MA_E_ADDR;
  }
  if ((gGlobal.heap_end != 0)
      && (gGlobal.heap_end <= mem)) {
    return MA_E_ADDR;
  }

  // Remove prefix to get to the real block address
  addr = (shift_t*)mem - 1;

  _MB_add_to_free_list((ptr_t)addr, *addr);
  return 0;
}
```

Consider an NXP Semiconductors LPC2138 ARM-7 with flash memory at 0-0x7FFFF and RAM at 0x40000000–0x40007FFF. The development toolset is Keil µVision3. The toolset assigns the stack space to a low address range; therefore, the heap space can span from the last static variable to the end of the RAM space. The RAM space layout looks like what you see in Table 2.

## PROCESSOR-SPECIFIC DEFINITIONS

The library source code is organized into four files: MB.c, MB.h, MB_prv.h, and MB_processor.h. MB.c contains the code. MB.h is the header file that must be included in the target application source files. MB_prv.h is a private header file for the library itself. MB_processor.h is the translator file and the only one that needs to be customized for each of the processor architectures. The other three files are used with no change. MB_processor.h defines variables and data types that must be changed for each processor. Let's review.

MIN_BLOCK_SHIFT is the size of the smallest block in the system expressed in a power of two. This size must be larger than or equal to the size of a pointer variable in the system. For a system with 16-bit address space, a pointer will occupy 2 bytes so this number will be defined as 1.

MAX_BLOCK_SHIFT is the size of the biggest block in

Listing 5—The _MB_add_to_free_list function

```
static int
_MB_add_to_free_list(ptr_t mem, shift_t shift)
{
   _MB_free_list_t*     addr = (_MB_free_list_t*)mem;

   for (; shift <= MAX_BLOCK_SHIFT; ++shift) {
        _MB_free_list_t*     prv = &gGlobal.free_list[shift];
        _MB_free_list_t*     cur = prv->next;
        _MB_free_list_t*     buddy = (_MB_free_list_t*)BUDDY_OF(addr, shift);
        // Look for the buddy in a list. There are four possible outcomes:
        // 1) A buddy is found
        // 2) The block (addr) is larger than cur
        // 3) The block (addr) is between prv and cur
        // 4) Reach the end of list (cur == NULL)
        for (; cur != buddy; prv = cur, cur = cur->next) {
           if (addr < cur || cur == NULL) {
              // (3) and (4)
              // - Put the block into the list
              // - Done
              prv->next = addr;
              addr->next = cur;
              return 0;
           }
           // (2) continue to the next loop
        }
        // (1) Found a buddy:
        //     - Remove the buddy from the current list
        //     - Combine the two buddies to create a bigger block
        //     - Repeat the seach again in the next list of larger block size
        prv->next = cur->next;
        addr = MIN(buddy, addr);
   }
   // Error, the block is larger than the maximum size
   return MA_E_TOO_BIG;
}
```

| Address | Usage |
|---|---|
| 0x40007FFF | RAM End |
|  | Heap (starting from gGlobal.startpoint) |
|  | Statically defined variables (including gGlobal but not gGlobal.startpoint) |
|  | Stack spaces for various CPU modes |
| 0x40000000 | RAM Start |

Table 2—RAM space layout for the LPC2138

the system expressed in a power of two. This number depends on the available RAM address space.

Consider MAX_FREE_SHIFT. The free function doesn't combine a freed block with its buddy if a block shift is equal to or larger than this number. This number is used for performance-optimization purposes.

FAST_SHIFT indicates that the processor supports native instructions to shift a number by an arbitrary number of bits. This function block is usually called a barrel shifter. If this definition is defined, the shift value is converted to the actual block size by a shift operation. If not, a look-up table is used for the conversion instead.

shift_t is a data type for the header that's embedded in the block after the block is allocated (see Figure 6). Although the library uses only 1 byte in the header, a larger header size may be required. If the header is too small, the block's start address won't be aligned with the size of the variable that will be stored in the block. A problem with an unaligned pointer will not be detected while compiling the code. The example project chooses a 4-byte header to make sure that a pointer can be stored in an allocated block.

The following definitions are guarded by the #ifdef _LPC213x_ statement and added to MB_processor.h. The processor name is then defined in the project/make file. This makes it possible to put the definitions for other processors into the same file so the library can be used anywhere (see Listing 6).

To enable the compiler to use the definitions guarded by _LPC213x_, the symbol _LPC213x_ must be defined. In some toolsets, this processor name is automatically defined, but this toolset doesn't have one. So we need to add the _LPC213x_ definition to the toolset, which resides in the Define section in the Preprocessor Symbols pane under the C tab of the target option window (see Photo 1a).

In addition to the aforementioned modification in MB_processor.h, if you want the linker to automatically put the heap into the right location, you need to customize the linker as well. For most of the compiler/linker toolsets, this means that the (static) data section of the library must be the last one to be linked.

To change the link order for Keil µVision3, you have to add a linker directive ?DT0?MB (LAST) into the toolset. You do so in the User Segments field located under the LA Locate tab in the target option window (see Photo 1b).

## LIBRARY INITIALIZATION

After the aforementioned changes to the project file, you can add the library source files to the project, add a statement to initialize the library, and build. The initialization call looks like this:

```
MB_Init(0x40008000, 0x40008000);
```

The start and end addresses are identical and is just one byte over the last RAM address. This means that the heap space is between gGlobal.startpoint and 0x40007FFF (end of RAM).

The linker links the library's variables after all the other modules' variables. Therefore, the linker places gGlobal.startpoint—the last variable of the library—above all the other variables. Notice that the heap space spans up to the end of RAM because the stack is located at a low address.

## BUILD RESULT

The debugger shows that the linker places the heap starting address gGlobal.startpoint at 0x40000800. The initialization routine partitions the heap as shown in Table 3.

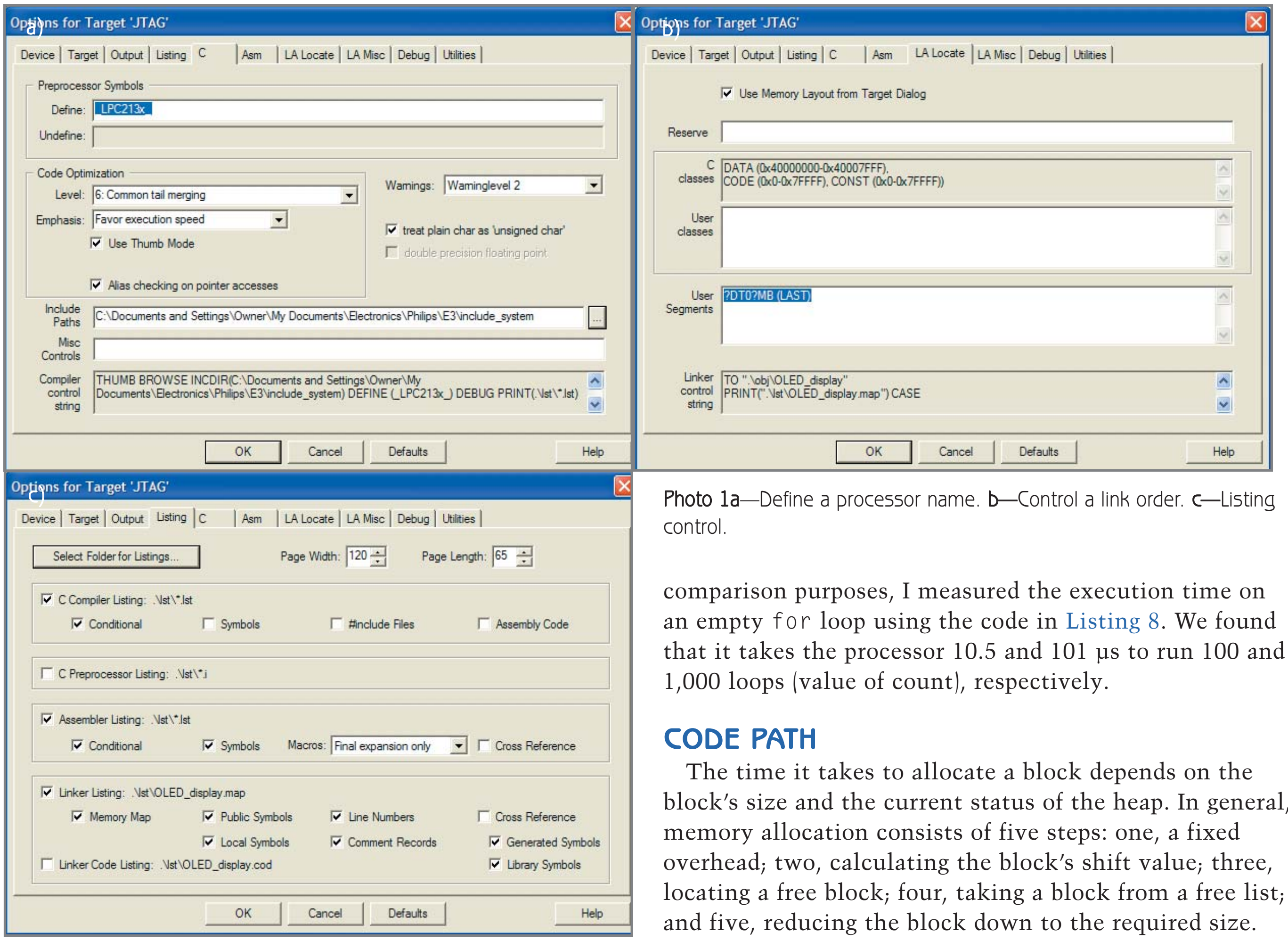
In addition to the debugger, most of the toolsets can generate a link map that shows how the code modules are linked together. For a Keil µVision3 project, you can add checkmarks to generate the link map and associated details at an input pane labeled Linker Listing under the Listing tab in the target option window (see Photo 1c). When we build the project again, the toolset will generate a map file containing link result. Listing 7 is a snippet of code from the sample project's map file.

## CODE

Notice that the first module in the code section (flash) is STARTUPCODE located at address 0. The data section (RAM) begins with STACK at the address 0x40000000 and ends with variables of the MB module. The linker reports that the library's data section takes up 0x50 (80) bytes, which is the space for gGlobal. However, the heap starts

Listing 6—Customized parameters

```
#ifdef _LPC213x_
#define MIN_BLOCK_SHIFT 2          // Min block size is 4 bytes
#define MAX_BLOCK_SHIFT 16         // Max block size is 64 KB
#define MAX_FREE_SHIFT  16         // Max shift when freed
#define FAST_SHIFT                 // This processor has a barrel shifter
typedef mem_size_t      shift_t;   // Define the size of block header
#endif
```


a)
Options for Target 'JTAG'
Device | Target | Output | Listing | C | Asm | LA Locate | LA Misc | Debug | Utilities
OK
Cancel
Defaults
Help

Preprocessor Symbols
Define: _LPC213x_
Undefine:
Code Optimization
Level: 6: Common tail merging
Emphasis: Favor execution speed
[✓] Use Thumb Mode
[✓] Alias checking on pointer accesses
Warnings: Warninglevel 2
[✓] treat plain char as 'unsigned char'
[ ] double precision floating point
Include Paths: C:\Documents and Settings\Owner\My Documents\Electronics\Philips\E3\include_system
Misc Controls:
Compiler control string:

```
THUMB BROWSE INCDIR(C:\Documents and Settings\Owner\My
Documents\Electronics\Philips\E3\include_system) DEFINE (_LPC213x_) DEBUG PRINT(.\lst\*.lst)
```

b)
Options for Target 'JTAG'
Device | Target | Output | Listing | C | Asm | LA Locate | LA Misc | Debug | Utilities
OK
Cancel
Defaults
Help

[✓] Use Memory Layout from Target Dialog
Reserve:
C classes:

```
DATA (0x40000000-0x40007FFF),
CODE (0x0-0x7FFFF), CONST (0x0-0x7FFFF))
```

User classes:
User Segments: ?DT0?MB (LAST)
Linker control string:

```
TO ".\obj\OLED_display"
PRINT(".\lst\OLED_display.map") CASE
```

c)
Options for Target 'JTAG'
Device | Target | Output | Listing | C | Asm | LA Locate | LA Misc | Debug | Utilities
Select Folder for Listings...
OK
Cancel
Defaults
Help

Page Width: 120
Page Length: 65
[✓] C Compiler Listing: .\lst*.lst
[✓] Conditional [ ] Symbols [ ] #include Files [ ] Assembly Code
[ ] C Preprocessor Listing: .\lst*.i
[✓] Assembler Listing: .\lst*.lst
[✓] Conditional [✓] Symbols Macros: Final expansion only [ ] Cross Reference
[✓] Linker Listing: .\lst\OLED_display.map
[✓] Memory Map [✓] Public Symbols [✓] Line Numbers [ ] Cross Reference
[✓] Local Symbols [✓] Comment Records [✓] Generated Symbols
[ ] Linker Code Listing: .\lst\OLED_display.cod [✓] Library Symbols

Photo 1a—Define a processor name. b—Control a link order. c—Listing control.

at the address of `gGlobal.heap_start` (a 4-byte variable). Therefore, the data header for the library is 76 bytes.

The library code consumes 668 bytes of the ARM Thumb instruction set while the data takes 76 bytes. Refer to Table 4 for more details.

## MEASUREMENT METHOD

While the actual execution times to allocate and free memory blocks are not constant, you can follow certain guidelines to estimate and tune up the execution times. The execution time is measured by running a test code that allocates and frees memory blocks in a loop and uses an oscilloscope to capture the execution time. Each time in the loop, the code repeats the allocation and free in the pattern that we want to measure. To measure the execution time with an oscilloscope, you toggle output pins in the loop to generate a signal pattern that represents execution time at different points in the loop. The test code looks like what you see in Table 5.

The test project runs on an LPC2138 running the ARM Thumb instruction set at 60 MHz. The code is compiled using the "6: Common tail merging" option and optimized for execution speed. All the interrupts are disabled to ensure the accuracy of the measurements. For comparison purposes, I measured the execution time on an empty `for` loop using the code in Listing 8. We found that it takes the processor 10.5 and 101 µs to run 100 and 1,000 loops (value of count), respectively.

## CODE PATH

The time it takes to allocate a block depends on the block's size and the current status of the heap. In general, memory allocation consists of five steps: one, a fixed overhead; two, calculating the block's shift value; three, locating a free block; four, taking a block from a free list; and five, reducing the block down to the required size. Steps one and four always take the same amount of time. At step two, the execution time directly depends on the block's size. The execution times for steps three and five depend on the difference between the required block size and the size of the first available free block. If the required block size is immediately available, the search for a free block (going up the y-axis) and reducing block size (going down the y-axis) will be eliminated. If only a free block at the maximum shift is available for use, the execution time will be the highest (see Figure 9).

The time it takes to free a block also depends on the heap's condition. It begins with a fixed overhead and it then searches for the buddy block in the appropriate free list. If a buddy block is found, the two blocks are merged together and the resulting block is moved to the next

| Address | Usage | | | | |
|---|---|---|---|---|---|
| 0x40007FFF | RAM End | | | | |
| | From | To | Size | Shift | Function |
| | 0x40004000 | 0x40007FFF | 16 KB | 14 | Heap's blocks |
| | 0x40002000 | 0x40003FFF | 8 KB | 13 | |
| | 0x40001000 | 0x40001FFF | 4 KB | 12 | |
| | 0x40000800 | 0x40000FFF | 2 KB | 11 | |
| | 0x40000000 | 0x400007FF | | | Stack and statically defined variables |
| 0x40000000 | RAM start | | | | |

Table 3—RAM space layout for the LPC2138 after initialization

Listing 7—Map file output

```
MEMORY MAP OF MODULE:  .\obj\OLED_display (STARTUP)

START      STOP        LENGTH    ALIGN  RELOC   MEMORY CLASS   SEGMENT NAME
====================================================================================
00000000H  0000010FH  00000110H  4      AT..    CODE           STARTUPCODE
00000110H  0000013BH  0000002CH  4      UNIT    CONST          ?CON?Application
0000013CH  0000017DH  00000042H  4      UNIT    CONST          ?CON?App_Main

Data Section

40000000H  4000048FH  00000490H  4      UNIT    DATA           STACK
40000490H  4000060FH  00000180H  4      UNIT    DATA           ?DTO?App_Main
40000610H  40000635H  00000026H  4      UNIT    DATA           ?DTO?Command
40000636H  40000637H  00000002H  ---    ---     **GAP**
40000638H  400006D5H  0000009EH  4      UNIT    DATA           ?DTO?Serial
400006D6H  400006D7H  00000002H  ---    ---     **GAP**
400006D8H  4000071EH  00000047H  4      UNIT    DATA           ?DTO?OLED_display
4000071FH  4000071FH  00000001H  ---    ---     **GAP**
40000720H  40000723H  00000004H  4      UNIT    DATA           ?DTO?Shell
40000724H  4000077DH  0000005AH  4      UNIT    DATA           ?DTO?SSP
4000077EH  4000077FH  00000002H  ---    ---     **GAP**
40000780H  400007A5H  00000026H  4      UNIT    DATA           ?DTO?tSysEvent
400007A6H  400007A7H  00000002H  ---    ---     **GAP**
400007A8H  400007B3H  0000000CH  4      UNIT    DATA           ?DTO?TimerUnit

400007B4H  40000803H  00000050H  4      UNIT    DATA           ?DT0?MB
```

larger free list. Steps 2 and step 3 are repeated until the required buddy can't be found (see Figure 9).

Notice that the execution time for memory allocation depends on the time it spends on the y-axis. When freeing a block, the code spends time on both the x- and y-axes.

## SYMMETRICAL

To study the library's performance, let's begin with a study of execution times in a controlled condition. You can compare the execution time to allocate and free small blocks from large and small free blocks in the heap. Maximize the size difference between the small and large blocks as much as you can. In my test, choose 8 bytes for the size of small blocks. The large block's size, however, depends on the system.

By observing heap space layout after the library is initialized using the default method (see Table 3), I found that the biggest block size in the heap is 16 KB. So, the large block in the study is 16 KB. The last step is to change an `MB_Init` argument so that the heap starts at the address of the 16-KB block. After the heap is initialized, the heap will contain just a single 16-KB block.

| Function name | From address | To address | Size (decimal) |
|---|---|---|---|
| MB_Init | 0x00003504 | 0x00003547 | 0x00000044 (68) |
| MB_Alloc | 0x00003548 | 0x000035CB | 0x00000084 (132) |
| MB_Free | 0x000035CC | 0x00003613 | 0x00000048 (72) |
| _MB_size_to_shift | 0x00003614 | 0x00003633 | 0x00000020 (32) |
| _MB_add_to_free_list | 0x00003634 | 0x000036CF | 0x0000009C (156) |
| _MB_init_qualify_request | 0x000036D0 | 0x0000372B | 0x0000005C (92) |
| _MB_init_global | 0x0000372C | 0x0000379F | 0x00000074 (116) |

Table 4—Code space

Refer to Table 5. You allocate two of 8-byte blocks consecutively (A then B), and then free both of them in reverse order (B then A) in a measuring loop. By doing so, there is no change to the heap between allocation and free of block B. It is also true for block A. Any change introduced by allocation for block B will be reversed when block B is freed. By keeping the heap status symmetrical, you can compare the execution times between allocating and freeing the same block (see Table 6).

It takes much longer to allocate block A than it does for block B. The difference comes from the time spent in steps three and five (see Figure 9) when block A is allocated. When allocating block B, the heap already has a free block of the exact size. Thus, steps three and five are skipped. On the opposite side, freeing the block B is very short. The free function just puts block B back into an empty free list. Searching and merging blocks isn't required. Freeing block A, on the other hand, triggers search and merges all the way from $2^3$ to $2^{14}$.

For the same block size, allocation takes longer than freeing when a small block is allocated from and freed back to a large block (28.60 versus 13.20 µs). The allocation takes longer because the code has to travel up and then down the array of free lists, while the free code only travels up the free lists. Although it takes longer to allocate than freeing a block in the aforementioned comparison, the allocation time doesn't change so much with the heap's usage pattern. The aforementioned allocation time for block A and B are the maximum and minimum execution times for allocation in this system. Likewise, the execution time for freeing block B is the shortest free

| Initialize the heap so that the heap has only blocks of $2^Y$ | |
|---|---|
| Loop step | |
| | Set I/O pins J and K to 1 |
| | Run a fix delay loop |
| | Set I/O pins J and K to 0 |
| 1 | Allocate a $2^X$ block A, where X < Y |
| | Set IO pin J to 1 |
| 2 | Allocate a $2^X$ block B |
| | Set IO pin J to 0 |
| 3 | Free a memory block B |
| | Set IO pin J to 1 |
| 4 | Free a memory block A |

Table 5—Symmetrical case-measuring loop

code path (before optimization) of the system. Unfortunately, you cannot draw any conclusion yet about the maximum execution time to free a block.

## ASYMMETRICAL

I covered the free operations that spend fairly short amounts of time searching for a buddy (in a relatively short y-axis). Now let's cover the effect of the x-axis on the execution time. When an application allocates a large number of blocks of a given size and then frees some of them in an irregular pattern so that the blocks can't merge, the free list can grow to be really large (having a long x-axis). The time to search for a buddy block increases proportionally with the length of the x-axis. To study the impact of this asymmetrical usage pattern, allocate 1,000 blocks of the same size and then free every other one of them (e.g., the 1st, 3rd, 5th, ... 999th, in that order). By freeing every other block, none of the freed blocks will have a buddy. Therefore, the number of blocks in the free list keeps on growing. In addition, the address of every new freed block is always higher than all of the blocks in the list at that time, so the new freed block is always added to the end of the list. The net result is a very high execution time. As you can see in Table 7, the longest time to free a block in this experiment

Listing 8—Sample delay loop

```
{
   int      i;
   for (i = count; i > 0; --i) {
   }
}
```

reaches 210 µs, as opposed to the shortest time of 3.48 µs (see Table 6).

As part of the same experiment, the other 500 blocks are freed after the first set in the order from 2nd, 4th, ... 1000th. The result is also shown in the last column in Table 7.

The above experiment demonstrates an extreme condition that could happen. Although the chance for this condition to occur in an actual application is very slim, you should should be aware of the possibility and its impact. Now I'll cover a way to limit the variation in execution time.

## OPTIMIZATION

I've covered the two main factors that can change the execution time. The first is the distance on the y-axis between the size of block being allocated and the nearest free block in the heap. The second is the distance on the x-axis between the first block in the free list and the insertion point or buddy block in a free list. The key to improve performance is to control the distances in both the y- and x-axes.

The x distance can be reduced by simply limiting the size of the block that will be merged. For instance, if you limit the block size that will be merged in the experiment to $2^3$, or an 8-byte block, the free code path of block A and B will be equal or almost equal because there is no merge.

Another change is also in the free code path. You can add another condition so that at a certain block size there won't be an attempt to search for a buddy at all. A free block will be added in the front of the free list. This effectively reduces the code part on the x-axis. From a certain block size up, the execution time for the code path in the x-axis becomes a constant. The net effect of applying the two aforementioned changes together is quite interesting. Table 8 shows the result of setting the limit (for both kinds) to a block size of 8 bytes or larger.

The change is done in function `_MB_add_to_free_list` (see Listing 9, a modification of Listing 5). The function takes an additional argument `shift_limit`, which limits the value of `shift` in line 6. The code works the same way as long as `shift` is less than `shift_limit`. When `shift` is equal to or greater than `shift_limit`, the code at line 33 takes over. The value of `shift_limit` is `MAX_BLOCK_SHIFT` for almost all of the

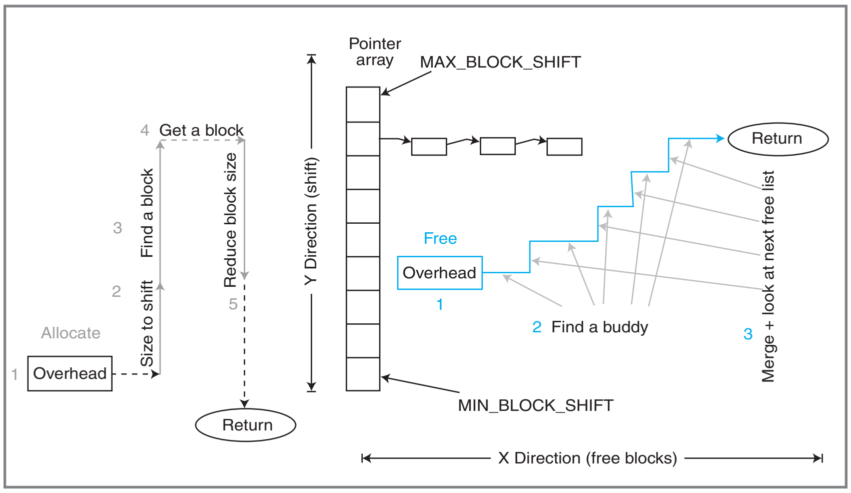


Figure 9—Symmetrical allocation and free

| Execution time (µs) to allocate and free an 8-byte block from heap with one 16 KB | | | |
|---|---|---|---|
| Allocate Block A | Allocate Block B | Free Block B | Free Block A |
| 28.6 | 2.92 | 3.48 | 13.2 |

Table 6—Symmetrical case-measuring result

| Execution time (µs) to allocate and free an 8-byte block in a heap with 2- to 16-KB blocks | | | |
|---|---|---|---|
| Average of allocating 1000 blocks | Average of freeing 500 odd numbered blocks | Free the last (999th block) | Average of freeing 500 even numbered blocks |
| 4.76 | 119.2 | 210 | 4.76 |

Table 7—Asymmetrical case-measuring result

| Execution time (µs) to allocate and free an 8-byte block in a heap with 2- to 16-KB blocks | | |
|---|---|---|
| Average of allocating of 1000 blocks | Average of freeing 500 odd numbered blocks | Average of freeing 500 even numbered blocks |
| 2.44 | 2.48 | 2.48 |

Table 8—A repeat experiment for the asymmetrical case with a maximum merge of an 8-byte block

cases. The exception is in the free code path, when the value of shift_limit is MAX_FREE_SHIFT (see Listing 9).

This method comes with a cost. When a block that is equal to or larger than the limit is freed, the library makes no attempt to merge. Over time, the library may not be able to provide a free block that is larger than the limit because all the blocks in the heap are fragmented. Basically, with the optimization, the library doesn't exactly use the buddy memory principle.

## APPLICATION

After setting up the library, you can follow a few simple guidelines to ensure that the library works reasonably well with your application. One, run the application and call library initialization, and use the debugger to find the heap layout. Observe the location of the biggest block. Two, change the library call to create a heap with just one large block. Three, decide on the smallest block size that will be used in the system and set MIN_BLOCK_SHIFT accordingly. Remember to include the size of the block header when calculating the block size. Four, decide on the biggest block size that will be used in the system and set MAX_FREE_SHIFT accordingly. Five, run a symmetrical test to identify the range of execution times for allocation and free code paths. You will get the maximum and minimum execution times for allocation as well as the minimum execution time for free. Six, set MAX_FREE_SHIFT to MAX_BLOCK_SHIFT, run the symmetrical test again, and observe the maximum time to free. The previous two steps should provide you reasonable performance boundaries. Seven, MAX_FREE_SHIFT in step four should be used in the actual system.

## CHANGE THE APPLICATION

In addition to optimizing the library, you can also optimize the application code to reduce the execution time even more by pre-allocating blocks or by avoiding freeing blocks. The first method is applicable when the application code can predict future actions and allocate memory blocks for the future actions before the blocks are actually needed. Basically, the memory allocations are done during a wait instead of a time-sensitive section of the application.

Another method involves not freeing any memory block at all and handling the situation the same way as you do when the library runs out of memory. When the memory runs out, an allocation call will fail and the function returns NULL. The application must prepare for this situation by checking the return value and acting appropriately. A normal response would be to free some or all of the blocks. However, freeing all the blocks at once may be faster and easier. To use this method, the code sections that use dynamic memory must be separated from the rest of the application. When memory runs out, you can simply reset the code section and reinitialize the library while the rest of the system can simply keep on working.

## THINK FIRST, THEN PROCEED

The buddy memory library may not be applicable to your application. As a tool, the library can efficiently solve the problem of not having enough memory. Its

Listing 9—The modified _MB_add_to_free_list function

```
1  static int
2  _MB_add_to_free_list(ptr_t mem, shift_t shift, shift_t shift_limit)
3  {
4    _MB_free_list_t*     addr = (_MB_free_list_t*)mem;
5
6    for (; shift < shift_limit; ++shift) {

. . . unchanged . . .

32   }
33   if (shift <= MAX_BLOCK_SHIFT) {
34     _MB_free_list_t*     prv = &gGlobal.free_list[shift];
35     _MB_free_list_t*     cur = prv->next;
36     prv->next  = addr;
37     addr->next = cur;
38     return 0;
39   }
40   // Error, the block is larger than the maximum size
41   return MA_E_TOO_BIG;
42 }
```

principle of using a $2^N$ block size keeps the code simple and helps reduce the chance of memory fragmentation. It is one of the best methods used in large systems like PCs and servers. However, it will increase the cost of your project, raise the execution time, amplify the code's complexity, and, potentially, introduce another class of software defects. So, use the library only when it is appropriate to do so, and be sure to use statically defined variables whenever possible. In general, the dynamic memory is most useful when it is used to share memory between the tasks that run at different times. The tasks don't last very long and don't run very often. The requirements for memory blocks that are small, fixed in size, and used often are best served by statically defined variables.

Another disadvantage of using the library is that the total effective amount of RAM available through the library is smaller than what it seems. A portion of the RAM is spent on the block headers. Another portion is spent in the padding space within blocks to keep block sizes in $2^N$ numbers. For this reason, avoid allocating a block in $2^N$ size. When doing so, the library will need to add more space for the block header and the effective block size will double.

The software defects related to the memory library are difficult to debug. Thus, note the following.

The library is not interrupt-safe. The library is not designed for a multithreaded environment. Memory allocation and free change the library's internal data structure. Changing the library's internal data structure from multiple threads simultaneously will create inconsistency in the internal data and lead to a heap corruption. Do not call the library from an interrupt service routine.

Access (write) within the block. A block allocated from the heap must be strictly used within its boundary. A write access outside the block is prohibited. Writing beyond the block's boundary could corrupt the internal data embedded in the heap and result in a heap corruption.

Free only once. A memory block can be freed only once after it is allocated. After it is freed, it must not be used or freed again. Freeing a block more than once could corrupt the heap.

The memory library may not be useful to everyone or every application. If you need it, be sure to take the time to understand how the library works and use it wisely.

*Sitti Amarittapark (sittiama@yahoo.com) earned a degree in electrical engineering at the Prince of Songkla University in Thailand. Since 1995, he has been developing software for desktops, handheld devices, and server systems. Currently, Sitti is working on disk-management software at Data Domain in Santa Clara, California. His personal interests include interprocess communication, electronics projects, and photography.*

*Author's note: I'd like to acknowledge Mitch Haile for reviewing the document and Chunqi Han for the heap initialization technique.*

## PROJECT FILES

To download the code and a list of useful resources, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/231.

## RESOURCE

A. Tenenbaum, Y. Langsam, and M. Augenstein, "Storage Management," *Data Structures Using C*, Prentice-Hall International, Englewood Cliffs, N.J., 1990.

CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

PROJECT CORNER

by Mahesh Venkitachalam

# Arduino-Based Temperature Display

After recently discovering the Arduino microcontroller platform, I built a microcontroller-based temperature display (see Photo 1). Arduino provides a high-level programming language and an easy-to-use development environment, the combination of which allows a newcomer to get a project up and running easily.

My goals for this project were to understand how to process analog input and display it using a microcontroller. For the analog part, I chose a temperature sensor. For the display, I used a pair of common cathode seven-segment LED displays.

## DESIGN

For the temperature input, I used a National Semiconductor LM35 IC, which is a calibrated sensor that outputs a voltage of 10 mV per 1°C. This analog voltage is fed into one of the microcontroller's analog pins. The analog values read in the 0–1023 range are then scaled to get the temperature value in Celsius. The values are averaged over 10 readings to avoid fluctuating values in the display and rounded off to the nearest two-digit number.

A seven-segment LED display has seven LEDs (leaving the decimal point aside) to address, and a pair of these requires 14 lines to address it. Although theoretically this can be done by 14 separate digital outputs from a microcontroller, this would be cumbersome to code and you would run out of digital outputs! A better way is to use a shift register, a device that takes a serial input of bits and converts that into a parallel output of individual bits. I used a pair of 74HC595 8-bit shift registers for this purpose, and they are chained together to address the 2 displays. To display a particular digit on the segment, a bit pattern is sent to the display by the shift register that lights only the relevant LEDs. The Arduino platform provides a `shiftOut()` method, which helps to conveniently "shift out" the 2 × 8 bits required to display the 2 digit of temperature.

This simple temperature display project is a great introduction to the world of microcontrollers, analog sensors, shift registers, and LED displays. You can replace the LM35 IC with an LM34 for a Fahrenheit-based temperature output. ■

*Author's note: The code and additional project-related details are available at: http://electro-nut.blogspot.com/2009/07/arduino-based-temperature-display.html.*

*Mahesh Venkitachalam is a software engineer living in India with his wife and son. As a child, he enjoyed tinkering with electronic circuits. Today, he has a strong interest in microcontrollers. You can view his electronics projects at http://electro-nut.blogspot.com/.*

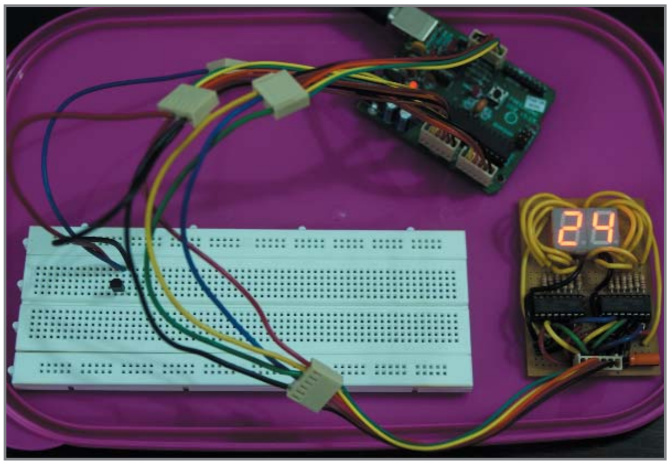


Photo 1—This is the Arduino-based temperature display.

## RESOURCES

E. J. Mastascusa, "Temperature Sensor - The LM35," Bucknell University, www.facstaff.bucknell.edu/mastascu/elessonshtml/Sensors/TempLM35.html.

C. Maw and T. Igoe, "Serial to Parallel Shifting-Out with a 74HC595," November 2006, www.arduino.cc/en/Tutorial/ShiftOut.

## SOURCES

**Arduino board** | http://arduino.cc

**LM35 Precision Centigrade temperature sensor**
National Semiconductor Corp. | www.national.com