Battery-in-a-Chip Technology p. 62 • Up & Running With C Language p. 70

www.circuitcellar.com

CIRCUIT CELLAR

THE MAGAZINE FOR COMPUTER APPLICATIONS

#228 July 2009

INTERNET & CONNECTIVITY

Internet-Based Weather Data Acquisition

A Compact Webcam Design From Start to Finish

Ethernet-Controlled HERMS

Text Library for Real-Time Translation

I²C Master Bus Controller

0 74470 75349 0 07>
$5.95 U.S. ($6.95 Canada)

SECURE SERIAL-TO-ETHERNET SOLUTION

Low-cost

32-bit Performance

SSH/SSL Secured

The complete, secure hardware and software solution

- Simple Ethernet connectivity for serial devices
- Works out of the box - no programming is required
- Enable data encryption to prevent unauthorized monitoring
- Customize to suit any application with development kit

Features:

- 10/100 Mbps Ethernet
- SSH/SSL/TCP/UDP modes
- DHCP/ Static IP support
- Web-based configuration
- Two TTL serial ports

SB70LC

$49.00 Qty. 100

Need a custom solution?

Customize with the NetBurner SB70LC Development Kit for only $99.

Customize any aspect of operation including web pages, data filtering, or custom network applications.

Kit includes: platform hardware, ANSI C/C++ compiler, TCP/IP stack, web server, e-mail protocols, RTOS, flash file system, Eclipse IDE, debugger, cables and power supply

Kit enables communication with peripherals that use: SD/MMC Flash Card (including SDHC), SPI, I^2C, or the general purpose digital I/O interface

The NetBurner Security Suite Option includes: SSH v1, v2 and SSL support

NetBurner
Networking in One Day!

Board Part Number | SB70LC-100CR
Development Kit Part Number | NNDK-SB70LC-KIT
Information and Sales | sales@netburner.com
Web | www.netburner.com
Telephone | 1-800-695-6828

freescale™
Alliance Member

5 Competitive Advantages

Overseas Manufacturing

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

CAM USA

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

Quick-Turn Production

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

Shipping Logistics

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

Significant Price Saving

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

For details please visit our website or call us

Capabilities

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

ITAR, ISO 9001 : 2008

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

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

24 YEARS IN BUSINESS...AND STILL GOING STRONG

Do your MCUs suffer from bloated code?

HI-TECH C® PRO
Denser code, better performance

HI-TECH C
COMPILERS
by Microchip Technology

HI-TECH C PRO compilers feature Omniscient Code Generation™ (OCG)

OCG is a whole-program compilation technology that can determine the usage of every register, stack, pointer, object and variable in your program, and generate fine-tuned code accordingly, effectively doubling your memory capacity, letting you fit more code in your MCUs.

Denser code, better performance: www.htsoft.com/ocg

HI-TECH C PRO compilers are available for Microchip PIC10/12/16/18/32 MCUs, from www.microchipdirect.com and Microchip distributors.

MICROCHIP

www.microchip.com/HI-TECH

Let your geek shine.

Meet Vanessa Carpenter and Diesel Møbius, SparkFun customers and developers of the Critical Corset. Using a Polar heart rate monitor, an Arduino, and a cleverly hidden air pump system, Vanessa and Diesel designed a corset that explores the rules of attraction. As the user's heart rate increases, the corset gently tightens, creating a more confident posture.

Whether you need a heart rate monitor or just a handful of LEDs, the tools are out there. Create a project you'll love, and let your geek shine too.

sparkfun™ ELECTRONICS

Sharing Ingenuity

WWW.SPARKFUN.COM

For more info on Vanessa and Diesel's project visit www.illutron.dk.

TASK MANAGER

'Net Tech and You

With each passing year, we receive more and more article submissions about projects in which Internet and Ethernet technologies figure prominently. Why is that?

Perhaps all you designers out there are constantly hard at work trying to develop the next big thing in 'Net-connected technology. There's a lot of money to be made designing innovative 'Net-related hardware and software. Or maybe you're just so connected that the majority of your designs are linked to the 'Net out of sheer necessity?

The answer is that it's probably a bit of both.

We begin this issue with three examples of such projects. Perhaps one of theses articles will provide you with just the right info to start your next exciting 'Net-connected design.

On page 16, Steven Nickels presents his Internet Weather Display design. Many people have weather stations, but this one is unique in that it doesn't require external sensors. The design gathers weather data and alerts from the Internet and displays it on a color monitor.

Turn to page 26 to learn how Minas Kalarakis combined his interest in embedded design, cameras, and the Internet to build his own webcam. The compact design pans the camera horizontally and vertically, and it can change its IP and gateway address to match a network. An Ethernet module transmits the packets over the Internet.

We've run articles about HERMS projects in the past, and Kirt Weakman took notes. In "iMash," he describes how he updated his own Ethernet-controlled heat exchange recirculating mash system (p. 38). This project required Kirt to implement skills ranging from metal-bending and welding to working with embedded Linux and VB.NET code.

In previous *Circuit Cellar* articles, Enoch Hwang covered the topics of ALU chip design and VGA monitor control. In this issue he tackles a new topic: implementing an I^2C master bus controller using an FPGA. Turn to page 46 to learn how to build the controller using VHDL or Verilog code.

If you're thinking about adding text-to-speech capability to a project, check out "Embedded Speak" on page 56. Jeff Bachiochi explains how to build a design's vocabulary by developing a text library for real-time translation.

In "LiOn King," Tom Cantrell presents an exciting new development in "Green" technology: a battery-in-a-chip (p. 62). This is truly an advance in embedded energy harvesting.

George Martin wraps up the issue with some useful information about C language (p. 70). This time around he explains how to get a C program up and running on an embedded system.

All of this exciting new content should keep you busy well into August! Let me know how it goes.

cj@circuitcellar.com

C. Abate

Note: Some background graphics on this issue's cover were made available courtesy of the U.S. National Oceanic and Atmospheric Administration (NOAA, www.noaa.gov).

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

MANAGING EDITOR
C. J. Abate

WEST COAST EDITOR
Tom Cantrell

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

NEW PRODUCTS EDITOR
John Gorsky

PROJECT EDITORS
Gary Bodley
Ken Davidson
David Tweed

CHIEF FINANCIAL OFFICER
Jeannette Ciarcia

MEDIA CONSULTANT
Dan Rodrigues

CUSTOMER SERVICE
Debbie Lavoie

CONTROLLER
Jeff Yanco

ART DIRECTOR
KC Prescott

GRAPHIC DESIGNERS
Grace Chen
Carey Penney

STAFF ENGINEER
John Gorsky

ADVERTISING

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

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

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

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

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

PRINTED IN THE UNITED STATES

CONTACTS

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

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

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

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

CIRCUIT CELLAR®, THE MAGAZINE FOR COMPUTER APPLICATIONS (ISSN 1528-0608) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Vernon, CT 06066. Periodical rates paid at Vernon, CT and additional offices. **One-year (12 issues) subscription rate USA and possessions $23.95, Canada/Mexico $34.95, all other countries $49.95. Two-year (24 issues) subscription rate USA and possessions $43.95, Canada/Mexico $59.95, all other countries $85.** All subscription orders payable in U.S. funds only via Visa, MasterCard, international postal money order, or check drawn on U.S. bank. **Direct subscription orders and subscription-related questions to Circuit Cellar Subscriptions, P.O. Box 5650, Hanover, NH 03755-5650 or call 800.269.6301.**

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

The Newest Interconnects

Micro USB
Industrial Ethernet
Hard Metric
Fiber Optic
MicroTCA
HDMI
SFP
Mini SAS
SATA
QSFP
SFP+
Displayport

molex
HDMI Cable Assemblies
www.mouser.com/molex/a

Tyco Electronics
Authorized Distributor
SFP+ Cable Assembly
www.mouser.com/TycoAmp_sfpcable

FCI
Mini SAS / SATA4x External Cable Assembly
www.mouser.com/FCI_Cable_Assembly

www.mouser.com
Over A Million Products Online

New Products from:

PHOENIX CONTACT
INSPIRING INNOVATIONS
Variosub Push-Pull Ethernet and Power Connectors
www.mouser.com/phoenix_pushpullconnectors/

HRS
HIROSE ELECTRIC CO., LTD.
SMT Coaxial Connector: U.FL Series
www.mouser.com/hiroseconnector/a

Amphenol®
DisplayPort™ Connectors
www.mouser.com/amphenolcommercial/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

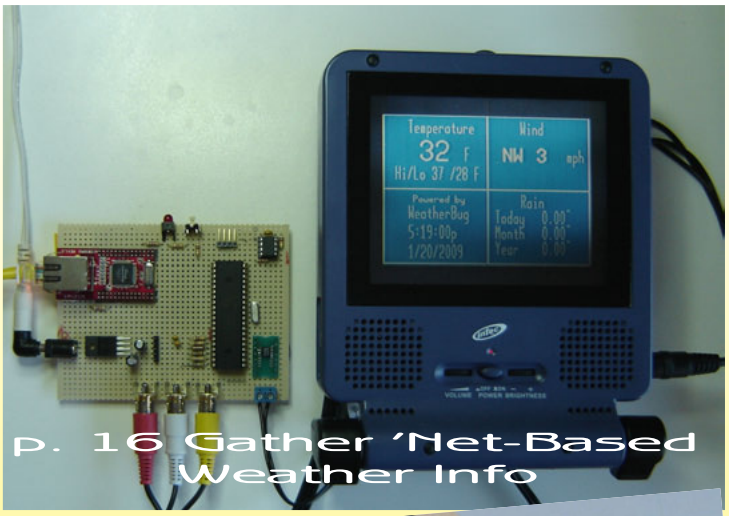
INSIDE ISSUE 228

July 2009 • Internet &Connectivity

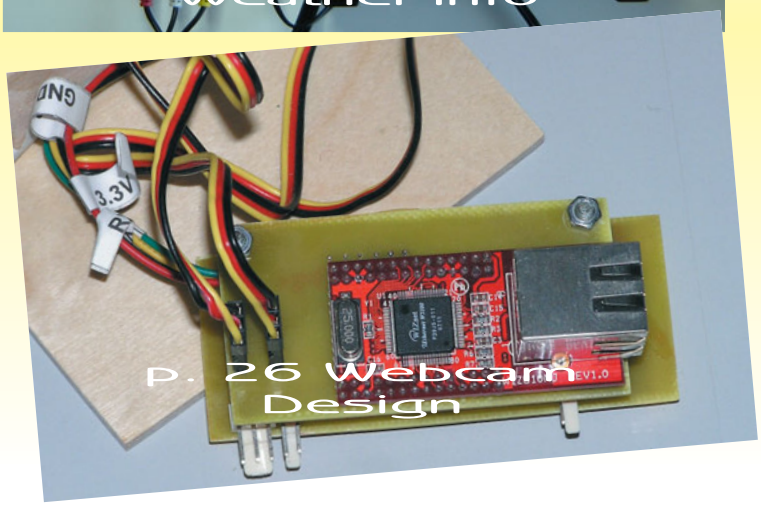
BONUS CONTENT

NimbleSig III— A New and Improved DDS RF Generator

Sound Synthesis Made Simple— A Multi-MIPS Music Box



p. 16 Gather 'Net-Based Weather Info



p. 26 Webcam Design


p. 38 Homebrewed HERMS

Hammer Down Your Power Consumption with picoPower™!

THE Performance Choice of Lowest-Power Microcontrollers

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

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

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

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

Everywhere You Are®

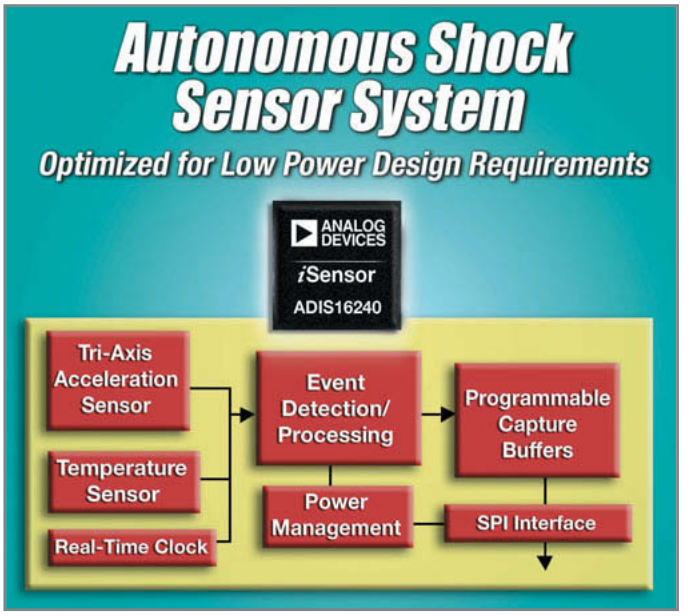
PROGRAMMABLE MEMS SHOCK AND VIBRATION SENSORS

The **ADIS16240** is a fully integrated digital shock detection and capture system that combines iMEMS technology with an embedded signal-processing solution. It enables designers to cost-effectively convert critical motion data into valuable system information in low-power applications, such as condition-monitoring, impact detection, security sensing, and tamper detection. The device's low-power, triple-axis shock and impact sensor and recorder includes an integrated on-chip programmable event-capture buffer, which monitors for user-programmed shock thresholds and holds the pre- and post-event data, time stamp, temperature, and voltage.

Programming features and data access are controlled from a standard SPI port, making the ADIS16240 compatible with many existing systems. Once implemented in the system, the device operates autonomously, capturing multiple events for later analysis or providing real-time indication of excessive shock environments.

The ADIS16240 consumes less than 1 mA in continuous sampling mode, making it ideal for use in portable or battery-powered security devices, remote safety monitors, or products that monitor the condition of goods or shipments. With a usable bandwidth of up to 1.6 kHz, the ADIS16240 is significantly more capable of capturing critical shock events than previously available solutions.

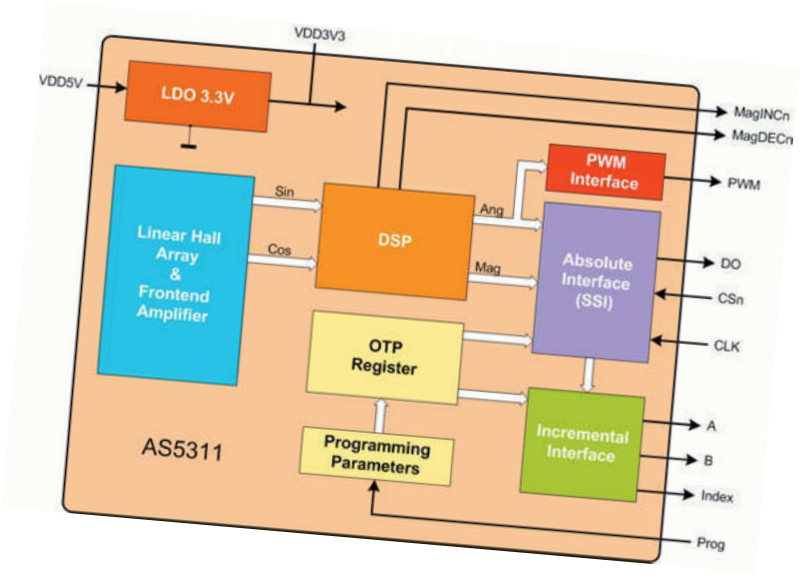
One-thousand-piece pricing is **$26.58**.



Analog Devices, Inc.
www.analog.com

HALL ENCODER IN THE SUBMICRON RESOLUTION RANGE

The **AS5311** is the first Hall effect sensor-based linear magnetic encoder to offer sub-micron resolution. This integrated linear Hall encoder can be used as an alternative to optical encoders. For linear motion sensing, a multi-pole magnetic strip is used. For rotary motion sensing, the magnetic strip is replaced by a multi-pole magnetic ring. An incremental output with a resolution of 10 bits per pole pair and a traveling speed up to 650 mm per second is available. Using, for example, a multi-pole magnetic ring with a diameter of 41.7 mm, a resolution of 16 bits (65.536 steps per revolution) can be achieved. In conjunction with a 2-mm pole-pair magnetic strip, the AS5311 provides a 1.95-µm resolution signal at its incremental output and a 488-nm resolution signal at its serial output.



The encoder can be operated with either 3.3- or 5-V supply voltage and is available in a 20-lead TSSOP package. It is specified for an ambient temperature range from –40° to 125°C.

The AS5311 costs **$5.23** in 1,000-piece quantities.

austriamicrosystems AG
www.austriamicrosystems.com

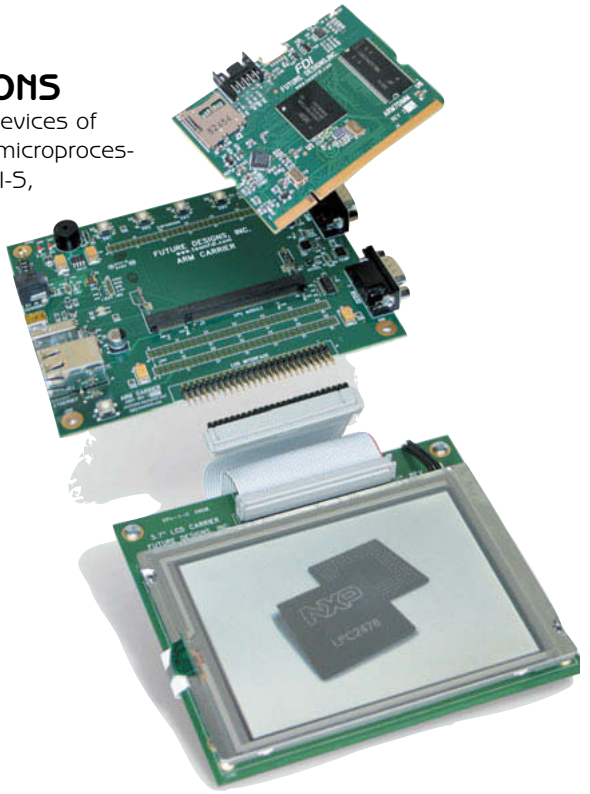
NEW PRODUCT NEWS

Edited by John Gorsky

TOUCH SCREEN PLATFORM FOR HMI/GUI APPLICATIONS

A series of modular development platforms that supports touch screen LCD devices of various sizes from multiple vendors is now available. The design supports ARM microprocessor and microcontroller devices in three popular families: ARM926EJ, ARM7TDMI-S, and Cortex-M3. The initial product, the **ARM-57TS-LPC2478**, includes a Toshiba 5.7" TFT LCD with integrated touch screen and is based on the NXP LPC2478 ARM7 microcontroller with integrated LCD driver. The "brain" of the system is a SoC module called the ARM7DIMM-LPC2478. This modular 200-pin SODIMM is 2.66" × 1.89" and contains the LPC2478 microcontroller, along with 8 MB of external SDRAM and support circuits. The kit includes the uEZ Rapid Development Environment running the FreeRTOS Operating System, and the IDE is the Rowley CrossWorks compiler.

The ARM-57TS-LPC2478 development kit costs **$425**. The ARM7DIMMLPC2478 costs **$74.50**. An ARM9-based touch screen kit running Linux and a SoC module for the NXP LPC3250 costs **$480**. Kits for the NXP LPC1778 Cortex-M3 will be released later in the year.



Future Designs, Inc.
www.teamfdi.com

CLEAR OBJECT SENSORS

The **WORLD-BEAM QS30 Clear Object Sensor** provides reliable clear object detection to solve challenging applications. The Clear Object Sensor utilizes an advanced optical design to allow the sensor to see translucent materials, ensuring the dependable detection of clear targets including PET bottles and glass containers. Additionally, the optical design prevents the sensor from being tricked by specular reflections, allowing the sensor to detect optically engineered surfaces such as mirrors, LCD glass with polarizing films, and semiconductor wafers.

The Clear Object Sensor is easily configurable using either the push button on the sensor or via a remote input line. The user can select from one of three switching thresholds—light, medium or dark, depending on the target's transparency level—to optimize detection and maximize sensing stability. The sensor also features an automatic compensation algorithm, which adapts the switching threshold to the sensor's environment in real time. Small changes due to dust or contamination on the sensor and reflector or small changes caused by ambient temperature shifts are filtered out by the microcontroller. These features ensure long and trouble-free sensor operation.

The QS30 Clear Object Sensor offers a range of 100 mm to 2 m, with a fast 0.5-ms response rate, and it includes an easy-to-read bar graph display for easy configuration and status monitoring during operation. It also provides three selectable thresholds based on the type of target being detected and a visible red emitter beam for simple alignment. Bright LED indicators show power and output status, and the unit operates off of 10- to 30-VDC supplies.

The sensor costs **$179**.

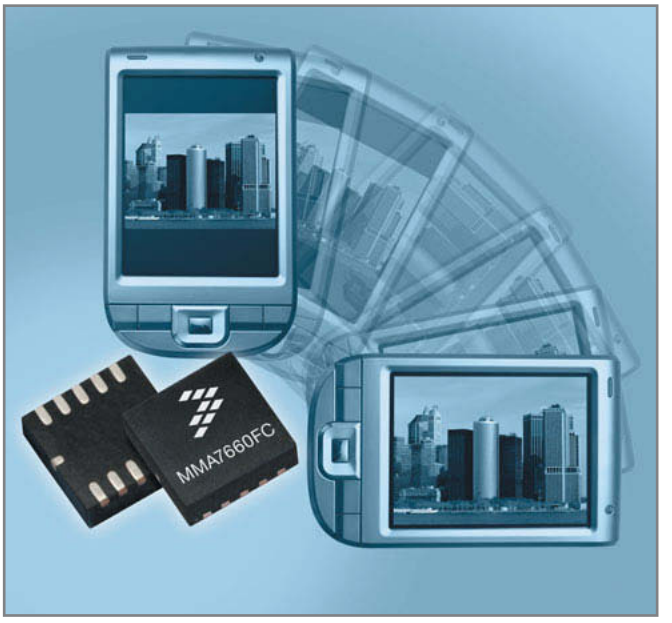
Banner Engineering Corp.
www.bannerengineering.com

SMART MOTION SENSOR ENABLES PRECISE HAND MOVEMENT CONTROL

The **MMA7660FC** is a highly advanced, low-power sensor based on proven micro-electromechanical system (MEMS) technology specifically engineered for handheld portable electronic devices. The three-axis accelerometer enhances user interfaces for mobile phones, small appliances, and gaming by allowing the user to tap, shake, or orient the device for specified commands. The device also includes smart power management features to help extend battery life.

The MMA7660FC accelerometer is designed to achieve up to five times longer battery life than current solutions on the market when continuously operating to determine motion. Configurable power-saving modes and a power-select capability help designers achieve optimal current consumption by choosing one of eight sample rates. System-level power is reduced through a configurable auto wake-up/sleep feature achieved without intervention or polling by the host processor. The MMA7660FC accelerometer provides conversion to digital values at a user-configurable output data rate, offering proportional savings in supply current and power.

The MMA7660FC sensor is available now for a suggested resale price starting at **$1.39** in 10,000-piece quantities. To help shorten development cycles, the RD3803MMA7660FC kit includes the evaluation board, plus the daughterboard and PC application. Corresponding device collateral is also available. The kit is available at a suggested resale price of



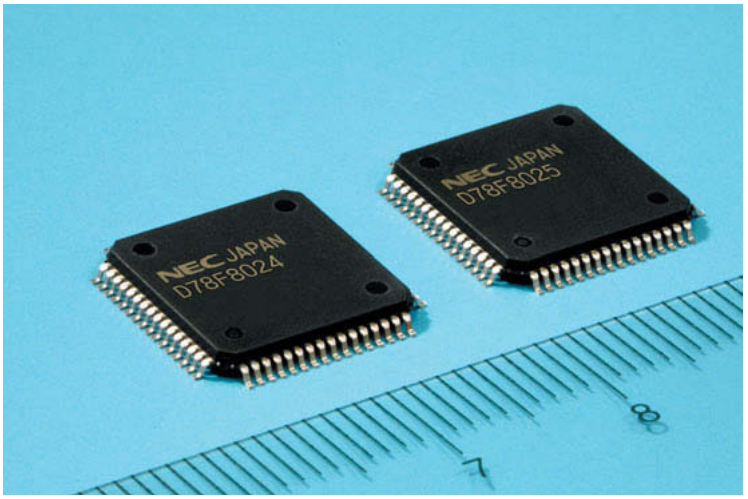
$119. For customers in the prototype stage, the daughterboard is also available separately as KIT3803MMA7660FC at a suggested resale price of **$35**.

Freescale Semiconductor
www.freescale.com

MICROS WITH INTEGRATED CONSTANT HIGH-CURRENT DRIVERS

The **µPD78F8025** is a new 8-bit, all-flash, general-purpose microcontroller with integrated constant high-current drivers and 32 KB of flash memory. The new device is suitable in applications such as home electric cookers, battery chargers, and LED lighting. Based on the µPD78F8024 MCU, the µPD78F8025 has expanded memory to meet increasing software requirements.

The microcontroller features integration of an all-flash MCU and constant current driver in a single chip for smaller board space and fewer off-chip components. It also includes an ADC and I^2C and UART interfaces, offering precise sensing and flexible communication interface. Additionally, it features built-in current drivers with protection circuits for efficient and highly reliable drive systems. Both the new µPD78F8025 and µPD78F8024 devices offer higher performance in a compact single-chip design.



In addition to constant-current circuits, the chips integrate protection circuits such as over-current, thermal shutdown, and voltage lockout to improve power control, efficiency, and reliability of the overall system. Both the µPD78F8024 and µPD78F8025 chips can be used in voltage-booster topology or buck topology, giving flexibility to designers to construct optimal systems.

Prices start at **$5** per unit.

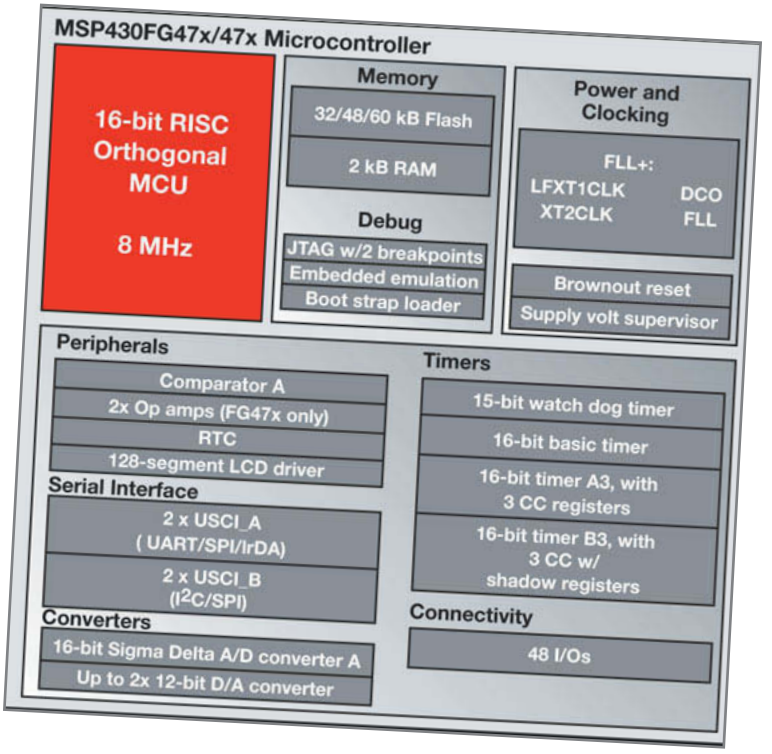
NEC Electronics Corp.
www.am.necel.com

SIGNAL-CHAIN-ON-CHIP MCUs

To quickly and efficiently develop reliable, convenient, and low-cost medical devices, engineers require microcontrollers that provide low power consumption, high performance, and targeted peripheral integration. Addressing this need is the new **MSP430FG47x** ultra-low-power MCU series. The FG47x MCUs offer on-chip integration of the complete signal chain, reducing design complexity and resulting in significant space and cost savings. These devices will help developers improve the quality and accessibility of healthcare through products such as blood glucose meters, digital thermometers, pulse oximeters, and blood pressure/heart rate monitors.

MCUs in this series offer a complete signal chain integrated on chip, consisting of two configurable op-amps, a 12-bit DAC, a comparator, and a 16-bit ADC. This reduces board space, bill of materials, and time to market. It also offers a 128-segment LCD driver with contrast control for convenient diagnostic display. Multiple memory options are available with up to 60-KB flash memory and 2-KB RAM for easy programmability.

FG47x MCUs are currently available and start at **$5.50** (1,000-piece unit pricing). They are fully compatible with the **$149** MSP-FET430U100 and the **$99** MSP-FET430UIF flash emulation tool kits.

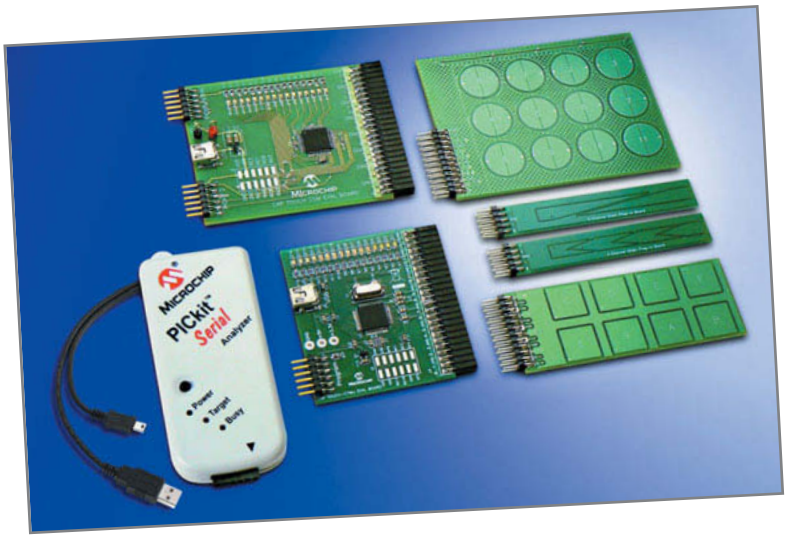


Texas Instruments
www.ti.com

NPN

mTouch CAPACITIVE TOUCH EVALUATION KIT

The **mTouch Capacitive Touch Evaluation Kit** enables designers to quickly and easily develop capacitive touch user-interface applications using Microchip Technology's 8- and 16-bit PIC microcontrollers. The flexible, comprehensive kit includes: two main boards (one populated with a PIC16F72*x* 8-bit MCU and the other with a PIC24F256GB110 16-bit MCU); four daughter-boards for developing capacitive-touch keys, sliders and a matrix; a PICkit Serial Analyzer; an easy-to-use GUI; and several code and schematic examples. The modular kit makes it easy for designers to try different keypad configurations and experiment with touch-pad sizes and shapes using the motherboards.



Touch-sensing technology is increasingly being adopted to improve the look and durability of user interfaces in appliances, consumer-electronic devices, medical electronics, automobiles, and many other markets and applications. The new kit provides a one-chip, highly-integrated solution based upon either the PIC16F72*x* 8-bit or the PIC24FGB 16-bit general-purpose MCUs, providing a flexible evaluation platform that lowers costs and shortens time to market.

The mTouch Capacitive Touch Evaluation Kit (part # DM183026) costs **$84.95**.

Microchip Technology, Inc.
www.microchip.com

THREE NEW GAS SENSORS

Three new sensors are available for gas presence detection designs. The **MQ-7**, **MQ-2**, and **MQ-5** are each designed to sense a different gas. The MQ-7 (part # 605-0007) is a carbon monoxide (CO) sensor designed for use in gas detection equipment. It can be used to detect the presence of carbon monoxide in home, automotive, or industrial settings. The MQ-7 is highly sensitivity to CO gas. It is stable, offers long life, and requires a simple drive circuit.

The MQ-2 (part # 605-00008) is similar and designed for sensing methane (CH_4). It also requires a simple drive circuit and offers fast response and high sensitivity. The third sensor is the MQ-5 (part # 605-00009) LPG gas sensor, which can be used to detect the presence of propane in home, automotive, or industrial settings.

The sensors are priced at **$4.99** each. Quantity discounts are available.

Parallax, Inc.
www.parallax.com

RUGGED PICmicro MICROCONTROLLER

The **MIAC** is a flexible controller for the industrial and experimenter markets. It is a rugged PICmicro microcontroller designed to allow those with little or no programming experience develop highly functional control systems. The free software supplied with MIAC allows users to design a program using standard flowchart icons, simulate the program on-screen, and then download the program to the MIAC using a standard USB cable.

The MIAC unit itself is packed with features including eight analog or digital inputs, four 10-A relays, four motor outputs, a keypad, an LCD, and a CAN bus interface, which enables networks of MIACs to be developed. The unit is powered by an advanced 18 series PICmicro microcontroller and is also compatible with all third-party PICmicro compilers. The MIAC and Flowcode 3 graphical programming software costs around **$180**.

Matrix Multimedia
www.matrixmultimedia.co.uk

Saelig

UNIQUE PRODUCTS & SUPPORT

www.saelig.com

Testgear

Color LCD Scope

Best Seller

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

PDS5022S / PDS6062T

2-ch 1GSa/s Scope

NEW! RIGOL

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

DS1000E

Scope + Analyzer

NEW! RIGOL

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

DS1022CD

July Offer while supplies last!

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

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

You must ask for offer# CC7

AT-70T

FREE COFFEE

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

While supplies last - not available with any other offers

Mention offer# SBW

Handheld Scope

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

HDS1022MN / HDS2062M

Pen Scope

10/25MHz USB powered scope-in-a-probe! Up to 100MS/s. **$193 / $308**

PS2104/PS2105

6 in 1 Scope

200kHz 2-ch 10-bit scope, 2-ch spectrum analyzer, 16-ch 8MHz logic analyzer, 5-ch sig gen, 8-ch pattern gen. **$199**

PoScope with Probes

Amazing 7 in 1 Scope! $180

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

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

Alan Lowne
Saelig CEO

CGR-101

Low-Cost Scope

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

PS2203/4/5

1/2GHz RF Generators

High accuracy/stability, wide range, low phase noise/leakage, serial control.

TGR1040 / TGR2050

Mixed-Signal Scope

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

CS328

USB Bus Analyzers

Best Value

Packet-Master™ - USB 1.1/2.0 analyzers and generators. **$699 +**

USB12 / 480+ / 500AG

16-Ch Logic Analyzer

Intuitive full-featured 16-ch 4MB 200MHz sampling memory. **$299**

LAP-16128U

SPI Bus Analyzer

Protocol exerciser/analyzer for standard SPI and non-standard 4-wire and 3-wire serial protocol interfaces up to 50 Mbps.

SPI Xpress

EMC Spectrum Analyzer

RF & EMF Spectrum Analyzer 1Hz to 7GHz for WiFi, mikes, etc.

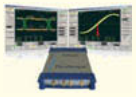
EMC RF & EMC Spectrum

EMC Spectrum Analyzer

Handheld Palm PC-based 2.7GHz Spectrum Analyzer.

PSA2701T

Worlds's Fastest



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

PS9200

Waveform Generator

USB2.0 speed 16-bit digital pattern or arbitrary waveform generator.

Wave Xpress

I2C Xpress

Versatile USB 2.0 I2C protocol exerciser and analyzer.

I2C Xpress

Automotive Testing

Kits turn your PC into vehicle-electrics diagnostic tool.

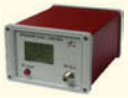
PS3423 / KLAR-MOD

CAN Gateway

Janz - Full-featured standalone fanless industrial Linux PC.

emPC-x133

RF Generator



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

APSIN6000

RF Testing/EMI Tents

Portable RF test enclosures & shielding tents with external frame.

RF Testing / EMI Tents

Wireless Data Loggers

Log and display temp, hum, volt, event-time or pulse-counting data

RTR-50

Multiparameter Loggers

Mini-logger with built-in temp/hum/pressure/3-axis accel sensors.

MSR145S

USB Logger

Standalone USB temp / hum / volt / current loop data logger. **$49+**

EL-USB-1/2/3/4

Electronic DC Load

Const. current, resistance, conductance, voltage & power modes

LD300

60/100/120MHz AWG

NEW! RIGOL

60/100/120MHz USB 14-bit ARB with USB RS-232, LAN/GPIB.

DG3061A/3101A/3121A

TorqSense

Configurable, patented USB-output non-contact SAW digital rotary torque transducers with integral electronics.

RWT320

Misc

USB to I2C

"Drop-in" solution connects PC to I²C/SMBUS + 32 I/O lines. **$89**

USB2C/IO

FTDI USB ICs

Lowest Prices

Popular UART and FIFO chips. Upgrade Legacy designs to USB.

FT232RL

CAN-USB

Intelligent CAN connection from PC's USB port. **$299**

CAN-USB

Serial-Ethernet Cable

Network serial product easily without a PC using this 28" cable. **$89**

eCOV-110-P

Lorlin Switches

Fantastic array of stock and custom switching devices.

Lorlin

PSoC Starter

Get going quickly with PSoC visual design environment.

PSoC Starter

Keyboard Simulator

USB board adds 55 I/O and 5 x 10-bit A/D inputs, 1 x 10-bit analog O/P.

PoKeys55T

Instant Ethernet

No OS needed. TCP/IP offload, ICs improve system performance.

WIZ110SR / W5100

Ethernet-IO

UDP/IP-controlled 24 digital I/O board 3 x 8-bit TTL ports.

Ether-IO 24

FPGA Systems

Ready-to-go out-of-the-box FPGA/DSP designs for beginners and experts!

RTG005

Wireless Solutions

Analog input, bluetooth wireless modules 433/868/915MHz.

EmbedRF / Adeunis

Temp/RH Sensors

Novel ambient sensors & modules accurately measure temp/RH.

UPSICAP / DLP-TH1

.NET Board

Small (2.2" x 2.2") lowest cost .NET Micro Framework dev system.

USBizi

Easy OLED Display

Compact, economical smart OLED with graphics drive from USB or RS232.

uOLED-96-G1

RF Modules

Simultaneously transmit composite video and stereo audio signals.

AWM6XX TX/RX

RS232 to 422/485

9p-9p or 25p-25p self-pwrd, isolated RS232-RS422/485

KK Systems

1/2/4/8/16 x RS232

Add 1-16 COMports via your PC's USB Port easily.

USB-COM

USB-Serial

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

CE-USB

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

Above are some of our best-selling, unique, time-saving products - see our website for 100s more: WiFi/910MHz antennas, wireless boards, LCD display kits, Ethernet/IO, USB/RS232/485, USB-OTG, instant Ethernet-serial, CAN/LINbus, USB cables/extenders, line testers, logic analyzers, color sensors, motion controllers, eng. software, wireless boards, SMD adapters, I2C adapters, GPS loggers, automotive testing, security dongles, video motion detectors, crystals/oscillators, custom switches, barcode scanners, DSP filters, PLCs, Remote MP3 players, etc. FREE Starbucks card with your $50 order!

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

WirelessHART DEVELOPER KIT

The **WirelessHART SmartStart Kit** is designed to speed up the development and testing of new WirelessHART products. The SmartStart Kit supplies everything needed to design, develop, test, and roll out new WirelessHART-compliant sensor networking solutions. The kit includes a network manager and 15 mote modules, 10 of which are equipped with a general-purpose microprocessor for fast prototyping, RF certified for FCC, IC, and CE.

Also included is a preconfigured SmartMesh IA-510 intelligent network with a GUI and dynamic bandwidth to demonstrate multiple processes along with software libraries, code samples, sample WirelessHART command definitions, and utilities to accelerate development. A complete documentation package to help speed up development, testing, and WirelessHART device certification is also included.

OEMs using the SmartStart Kit can dramatically shorten development time—often going from concept to WirelessHART product in half the time required by most WirelessHART development cycles.

The WirelessHART SmartStart Kit is priced starting at **$5,000**.

Dust Networks, Inc.
www.dustnetworks.com

FANLESS INTEL CORE 2 DUO INDUSTRIAL PLATFORM

The high-performance **GS-L10** system is a fanless Mini-ITX computer capable of supporting Intel's Core 2 Duo mobile processors. The GS-L10 exhibits impressive thermal management by utilizing a complex array of heat pipes to conduct the heat away from the CPU and chipset to the extruded aluminum heatsinks that comprise the top and sides of the case. This system can cool CPUs with a thermal design power (TDP) of up to 35 W, including Intel's T9*xxx* series of Core 2 Duo mobile processors.

The GS-L10 is now available as a complete system with an MSI IM-GM45 (Montevina-based) mainboard. Featuring Intel's GMA 4500MHD Graphics, up to 4-GB RAM, HDMI, DVI, two 10/100/1000 LAN ports, and a comprehensive array of on-board I/O, this system is ready for a wide range of industrial applications that require high-end processing power.

System prices start around **$900**, with project pricing available upon request.

Logic Supply
www.logicsupply.com

GIGABIT ETHERNET I/O CUBE

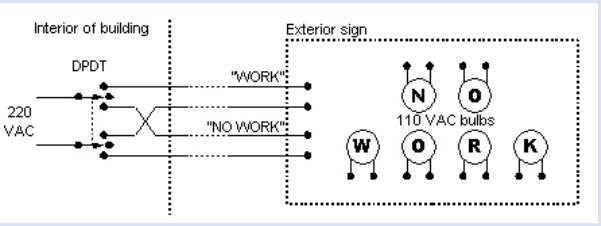
The new **PowerDNA PPCx-1G Gigabit Ethernet I/O Cube** is available in two basic models: a three-I/O layer (five-slot PPC5-1G) or a six-I/O layer (eight-slot PPC8-1G). The new GigE Cube offers higher speed and greatly enhanced diagnostics capability relative to the standard Cubes, but it remains compatible with all 30-plus UEI I/O board types. The six-layer model (PPC8-1G Cube) can provide up to 150 analog inputs, 192 analog outputs, 288 digital I/O, 48 counter or quadrature channels, 72 ARINC-429 channels, and 24 serial or CAN-bus ports.

Software for the GigE Cube is provided in the UEIDAQ Framework. The Framework provides a comprehensive, easy-to-use API that supports all popular programming and operating systems such as Windows, Vista, Linux, and most real-time operating systems (e.g., QNX, RTX, and RT Linux). In addition, the product is fully supported by LabVIEW, MATLAB/Simulink, DASYLab, or any application that supports ActiveX, OPC, or Modbus TCP control.

The DNA-PPC5-1G costs **$1,395**. The DNA-PPC8-1G costs **$1,595**.

United Electronic Industries
www.ueidaq.com

Problem 1—Jack operates an employment business of sorts and has erected a new sign in front of his establishment. It is to be operated from within his office by a DPDT switch with center off. In the up position, it will illuminate the word "WORK" on the sign with four 110-VAC bulbs, one behind each letter. The down position will light these four plus two more behind the letters "NO" or "NO WORK." The only source of power is 220 V AC, both wires being hot with respect to ground. There are only four conductors between the switch and the sign with no other means of ground or earth return. For safety purposes, all four wires must be floating or disconnected when the switch is in the center or off position. Assuming the bulbs will be run at their rated voltage (110 VAC), how are the six bulbs wired to the four conductors? No other components, active or passive, are used.



Problem 2—So, with this solution, what happens if one of the bulbs burns out?

Joe Mueck contributed Problem 1 and its answer.

What's your EQ?—The answers are posted at www.circuitcellar.com/eq/

You may contact the quizmasters at eq@circuitcellar.com

by Steven Nickels

Internet Weather Display

The Internet Weather Display is a weather station that operates without external sensors. The design gathers weather data and alerts from the Internet and displays it on a color TFT monitor. An LED flashes when alerts are transmitted.

It seems like a weather station is one piece of equipment that every electronics enthusiast has to have. But for people who live in apartments, condominiums, or townhomes, mounting exterior sensors is typically difficult or prohibited. Even some single-family home neighborhoods have strict homeowner association rules against "unsightly" objects outside the home. My Internet Weather Display is a weather station you can operate without exterior sensors. The project gets its data from professional weather stations located in your neighborhood, most often at schools or other government buildings. Like a backyard weather station, it shows current conditions. An added bonus is that you get forecasts from professional meteorologists and alerts issued by the U.S. National Weather Service (NWS).

Photo 1 shows the project in action. The system retrieves weather data from the Internet and then displays it on the 5″ color TFT monitor. The simple user interface includes a push button to select current conditions, the forecast, and active alerts. The TFT monitor includes

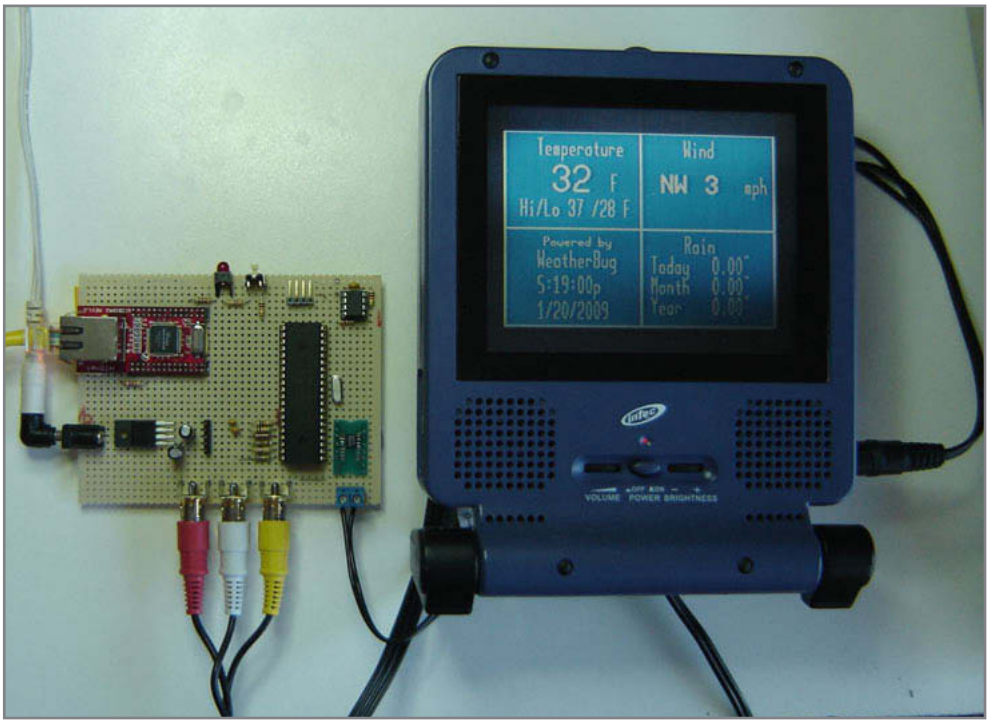


Photo 1—The Internet Weather Display gathers weather data from the Internet and presents it on a monitor. It displays current conditions, the forecast, and alerts issued by the U.S. National Weather Service.

built-in speakers for sounding an alert when new weather alerts are received. An LED also flashes while alerts are active to notify people who are hard of hearing.

WEATHERBUG API

During the past few years, several weather data providers have popped up on the Internet. WeatherBug is one such provider. It offers an application that resides in the task tray of your Windows PC and constantly shows the temperature of a local weather station. It can even alert you when an NWS message has been issued. While this application keeps you informed of the weather while you are actively using your PC, many people do not keep their PC fully powered on all day and night, so they could potentially miss an important weather alert. It's also inconvenient to wait the 30 seconds to 2 minutes for a PC to power-up just so you can check the forecast. A "real" weather station must remain on and be able to show wind direction immediately.

If you go to the WeatherBug website and get past the pages for the general consumer, you'll find the WeatherBug Labs page. There, you learn how to install WeatherBug on your Linux PC, cell phone, or personal webpage. A simple device like my Internet Weather Display has limited resources, so it needs a way to get just the raw data, and this is provided through a service called the "WeatherBug API." I recommend you review its terms of use. As users of the Windows PC application have seen, WeatherBug is supported by revenue from advertisements or through a yearly subscription. It does not charge for access to the API, and there are few restrictions if you use it for noncommercial purposes. But if you plan on selling a device that uses the WeatherBug API, your device must be able to open links to the WeatherBug website and you may need to compensate WeatherBug with a portion of your revenue.

The first step in getting access to the API is to register on the WeatherBug Labs website. Upon successful registration, you are assigned a unique access code that must be included with the request messages that are sent to the server. The server watches how often your device requests data and may refuse to respond if your device is polling for data too often. Watch out for bugs in your code that may cause a loop to send out a request message even though your intent was a 1-minute polling period.

There are two message formats available: XML and "pipe delimited." The former is a bit more difficult to use because you must format headers and keep track of special strings of characters used to identify the data. The pipe-delimited format is simpler because it uses readable characters where the data is separated by the "|" (ASCII 0x7c) character. The Internet Weather Display uses the pipe-delimited format. Refer to the WeatherBug API website for the complete documentation.

To request weather information, a message like this is sent:

http://a1111111111.isapi.wxbug.net/WxAlertISAPI/WxAlertIsapi.cgi?GetAlert60&Magic=160&ZipCode=80234&Units=0&RegNum=0&Version=7&t=1005&lv=0

Note the use of the HTTP high-level protocol. Only the "GET" command is necessary. As I already mentioned,

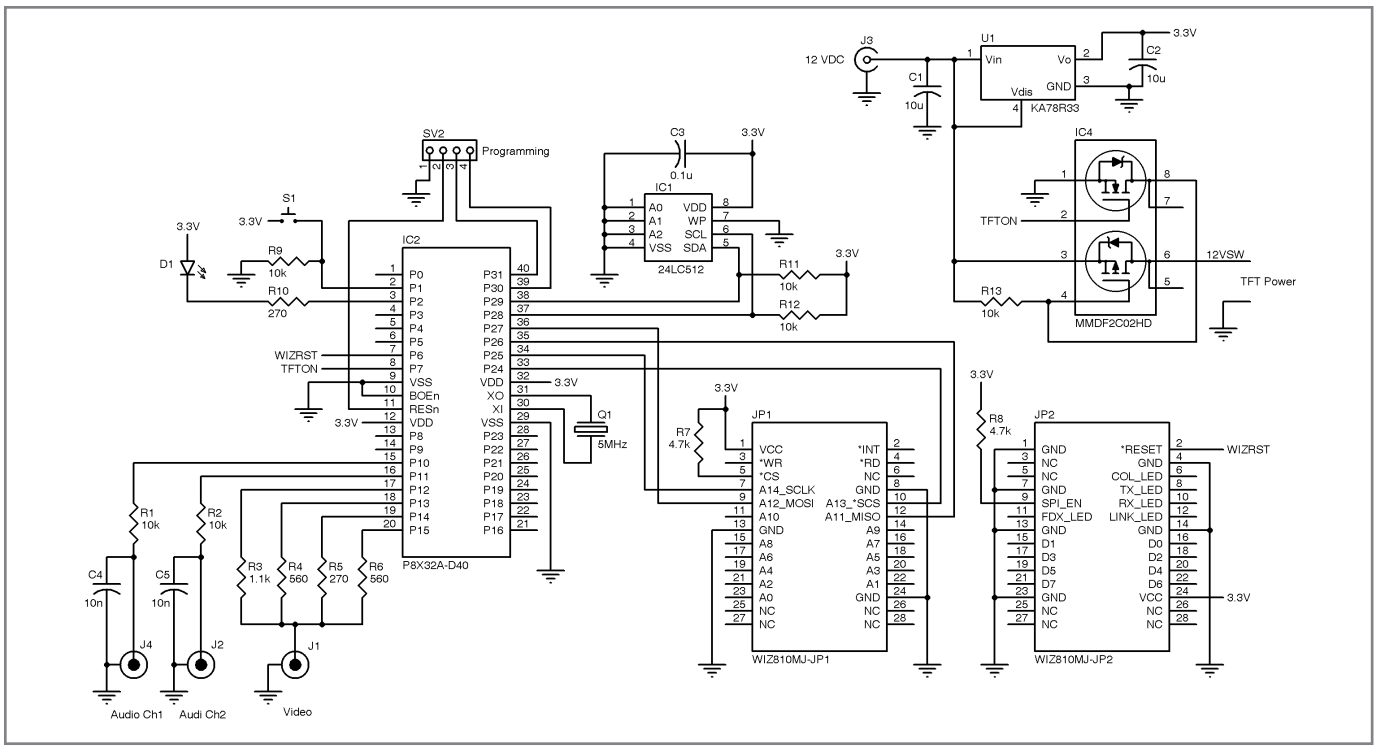


Figure 1—The Internet Weather Display uses a Parallax Propeller microcontroller to drive the video and audio interface. A WIZnet W5100 module handles all the Ethernet messaging up to the TCP/IP level.

the request message includes the unique code assigned to you when you registered at the WeatherBug API website. The number following "Magic=" identifies the data you're requesting. Use "10991" to request the current/live conditions, "10992" to request the two-day forecast, and "160" to request the active alerts. The three-day forecast and a list of weather station IDs are also available. The number after "ZipCode=" identifies the area for which you want weather information. The WeatherBug server will pick a station near the specified zip code. You could also specify "StationID=####," where "####" is a unique station number from the aforementioned list of station IDs.

The response is a stream of readable characters that includes HTTP protocol information followed by the weather data values. At the front of the weather data information is the "magic" ID. Verify this value to run the proper parsing routine. The data values that follow are separated by the "|" character. The following is an example of a response that contains alert information:

160|5|3|1|2|1197848582|2|3|1197848642|3|21|1197848702|

To extract the data, use a simple string-parsing routine to get the characters delimited by the "|" character. To reduce the number of bytes that the server must send, some fields use a number value to index into a locally defined string table. For example, the response for the alerts message includes an "alert type" value. The text for the alert is stored in a table of strings in the Internet Weather Display's memory. The table is searched for the matching "alert type" number, and the text to display is extracted from the table.

VERSION 1.0 TO 2.0

I entered the first version of my Internet Weather Display project in the 2007 WIZnet iEthernet Design Contest. In that version, I used an NXP Semiconductors ARM processor with a monochrome LCD. The "i" consumer devices have set a new standard in user interface, so a character or monochrome LCD just doesn't cut it any more. For this updated version, I went with a bit more color. Color TFTs are available, but few larger than 2.8" have a built-in controller, and I definitely wanted something bigger than 4". I also needed a microcontroller that could handle a large color display. The Parallax Propeller seemed to be a perfect fit for this application.

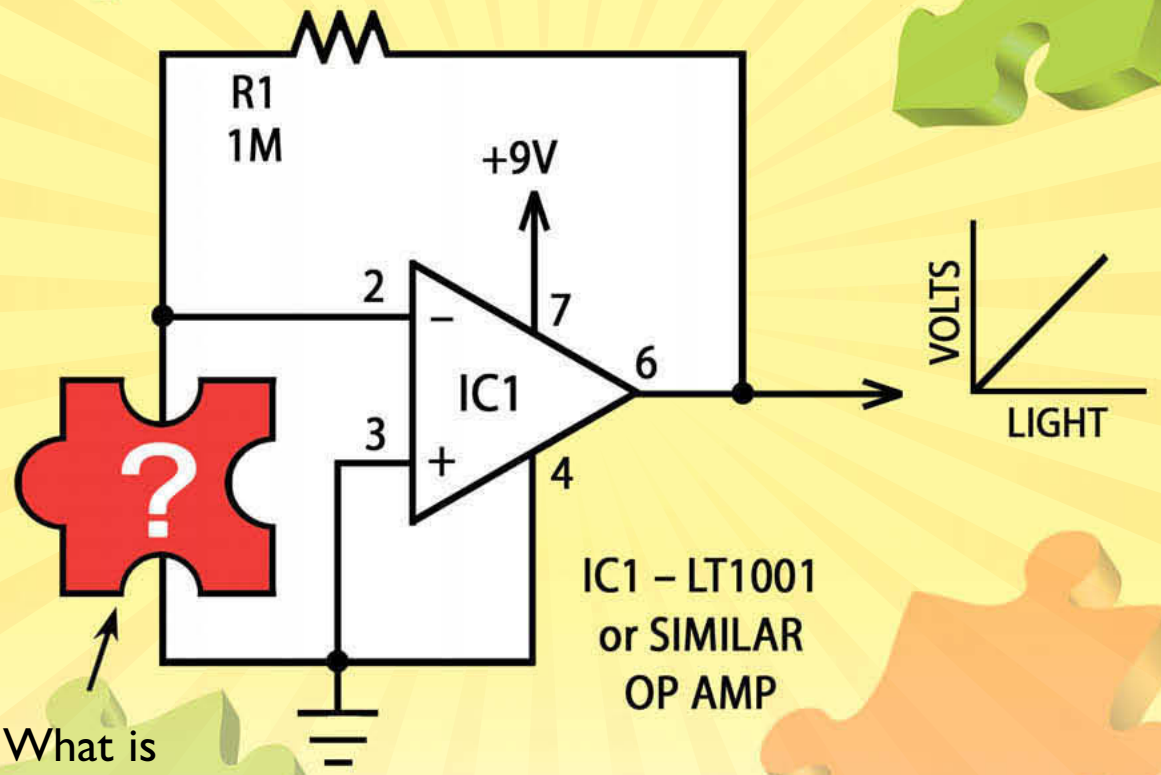
A while ago, I saw the ads for the Propeller microcontroller and thought, "Finally, something new in the area of microcontrollers." But because most of my projects used a microcontroller costing less than $10, the Propeller's $25 price tag at the time restricted my desire to learn more about it. Fortunately, Parallax has since lowered the price to a more comfortable $12, so I took another look and found it could easily generate the signals for either composite video or a VGA monitor. I decided to go with composite video because medium-size TFT monitors with composite video inputs are readily available for around $55, thanks to the in-car entertainment market. I could even connect the project up to my HDTV and have my own "weather channel."

The Propeller P8X32A-40 microcontroller is laid out in much the same way as the demonstration board offered by Parallax (see Figure 1). The video signal is generated by three output pins that set up a resistor digital-to-analog converter. Two audio channels are used to create an attention-getting, warble-tone alert sound when new alerts are received.

Listing 1—The `main` loop handles the user interface and polls the WeatherBug server for updated weather data.

```
Set I/O
Initialize display driver
Initialize WIZnet W5100
Initialize variables

Main Loop (1ms)

  If button is pressed (debounce),
    If sleeping,
     Turn on the display
  If new alerts,
     Show alerts
  Else
     Show current/live
     Else
         Show next screen
     Reset alert sound
     Set flag to update screen
     Reset sleep timer

  If sleep timer expired
     Stop the display driver
     If no new alerts,
         Power off the display
         (We need power to the speakers for alert sound)

  If screen update needed & not sleeping
     Show current/live, forecast, or alert(s)
     Reset screen update flag

  If new alerts, sound alert

  If 5 min expired,
     Poll for weather data
     Set flag to update screen
     Check for new alerts
```

Are you up for a challenge?



What is the missing component?

IC1 – LT1001 or SIMILAR OP AMP

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

JAMECO® ELECTRONICS

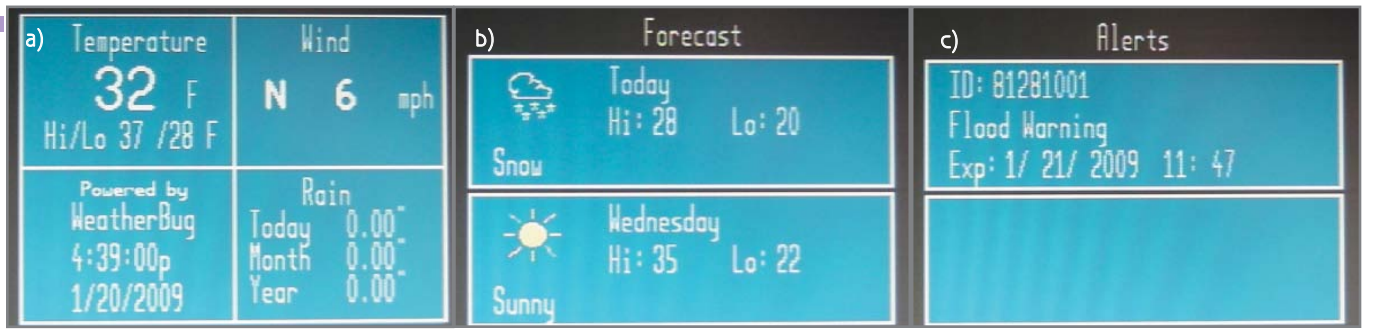


Photo 2a—The Current Conditions screen shows rainfall totals, temperature, and average wind direction and speed. **b**—The Forecast screen shows the high and low temperatures, as well as the expected conditions for the next two time periods. **c**—The Alerts screen displays active weather alerts and updates.

An LED flashes while there are active alerts, and a push button is used to select a new screen image. The program code is stored on the EEPROM. When the Propeller powers up, its resident bootloader copies the first 32-KB from that device into its internal RAM, starts an operating system driver called the "SPIN interpreter," and then starts executing the program code from RAM.

The WIZnet W5100 Ethernet controller is the perfect companion chip for the Propeller microcontroller. Most other 32-bit microcontrollers have a version that includes integrated Ethernet. The Propeller does not; in fact, it doesn't come with many peripherals at all. An external MAC+PHY could be used, but a TCP/IP stack would eat into the limited program memory. A better solution is to use the W5100 to handle all the message passing on the Ethernet up to the TCP/IP level. The W5100 chip has both a SPI and a parallel interface. I chose the SPI to keep the design simple, and because I didn't need the high throughput rate available in the parallel interface.

The project was constructed using soldered wire connections on a perfboard. RCA connectors were used for video and audio lines just in case I wanted to try out different TV monitors. I could've used direct wire connections if the board had been mounted inside the same plastic case as the monitor.

The only required external connections were the Ethernet cable and power from a 12-VDC wall transformer. If you use a small monitor like the one in Photo 1 and you can connect it to the same 12 VDC, be careful that you have the right size wall transformer. The Internet Weather Display board components require less than 100 mA, but my monitor required 500 mA, so a good choice was a 1-A transformer. If you're using a monitor or TV with its own power source, a 250-mA transformer is sufficient. To add a bit of "green" to the project, the software turns the TFT monitor off after a sleep period using a MOSFET switch pair.

SOFTWARE

This project was the first time I used the Propeller, so I had a little bit of a learning curve to overcome. I had three options for code development: Parallax's custom SPIN language, C language using an ImageCraft compiler, and assembly language. I chose the SPIN language because it appeared simple to learn and it didn't cost me anything. Parallax provided the Propeller Tool IDE that enabled me to create the SPIN source code files and download them to the chip through a small USB programming adapter. The Propeller did not offer an emulator, nor did the chip have any on-chip debug, so I had to be a little creative while debugging. Luckily, the included library code that drove the composite video signal worked right away and I was able to use the TFT screen to watch values.

Listing 1 presents pseudocode for the `main` loop. If you press the push button, the `show` variable is incremented to select a different display screen. Pressing the push button also stops the alert sound if it's on. Every 5 minutes, the WeatherBug server is polled for the weather data. The alert information is checked to see if there are any new alerts. If there are, the alert sound is turned on. To be "green," the display is turned off 20 s after the last button press. You may want to remove this feature if you use a standard television set instead of a small TFT monitor or if you want the display to be on all the time. Also, if you manually turn off the TV to save power, then connect the audio outputs to separate amplified speakers so that you're alerted to new weather statements.

Photo 2 shows the three screens. The Current Conditions screen includes temperature, average wind direction and speed, and rainfall totals (see Photo 2a). The Forecast screen shows the high and low temperatures and expected conditions for the next two time periods (see Photo 2b). The Alerts screen shows active weather statements (see Photo 2c). The display can show only two alerts at a time. You can view additional alerts, if they exist, by pressing the push buttons. The screens are drawn using the Graphics library included with the Propeller Tool. For more information about how the Propeller microcontroller draws images, refer to Chris Cantrell's article "Tile Graphics" (*Circuit Cellar* 209, 2007).

Listing 2 shows pseudocode for the steps needed to get data from a

CIRCUIT CELLAR

Reader Survey
&
Sample Product Evaluation Pool

This year we've decided to combine our annual Circuit Cellar reader survey with a special sample pool evaluation program. This gives readers the chance to provide Circuit Cellar with vital feedback while also registering for sample product consideration.

In addition to survey participants having the chance to win Circuit Cellar CD archives and subscriptions, a larger group of qualifying survey participants will also be able to receive a wide variety of product samples from our sponsors.

See page 55 of this issue for a few of the participating sample pool sponsors. (Additional sponsors may be added after this issue prints.)

The world of publishing has changed dramatically within the last 1-2 years. Now more than ever it's crucial that Circuit Cellar has your input so that we can maintain a course that's most beneficial to our core readership.

Please take a moment to complete this reader survey today!
To participate, visit www.circuitcellar.com/RS

> The W5100 does most of the 'heavy lifting.' It handles the entire Ethernet interface up to the TCP/IP level. It has a simple command interface to load data for sending outgoing data and for reading received data.

WeatherBug server. Since there are multiple servers and the IP addresses of those servers may change, the first step is to perform a domain name system (DNS) transaction. After the response is received, the WeatherBug server's IP address is known, and the request messages for live/current conditions, forecast, and alerts can be sent. After each request is sent, we get the responses and save them to specific buffers.

The DNS transaction consists of sending a query message to a DNS server. The server then responds with a message that includes the IP address that must be used. In my network setup, my DSL modem acts as a gateway to a DNS server out on the Internet. The query message is sent to my DSL modem, and then the modem forwards the message to a DNS server on the Internet. The modem also forwards the response back to my device. The project's software extracts the WeatherBug server IP address from the response. To keep things simple, the DNS transaction is performed each time the weather data is updated. If you want a higher update rate, it would be better to parse the DNS response for the "time-to-live" parameter and perform only the DNS transaction after

Listing 2—This pseudocode shows the sequence needed to get weather data from the WeatherBug server. First, DNS must be used to obtain the IP address of one WeatherBug server. A request is made to the server to send back the current/live conditions, a two-period forecast, and a list of active alerts.

```
-DNS
Set destination IP to 192.168.0.1 (my DSL modem)
Set destination port to 53
Open UDP socket
Send DNS query message
Wait for response
Get the response
Parse for the WeatherBug server IP

-Connection Setup
Set destination IP to WeatherBug server IP
Set destination port 80
Open TCP socket

-Get Live/Current Weather Data
Connect
Send request for live data, HTTP "GET"
Wait for response
Get response
Parse for weather information, save to specific buffer
Disconnect

-Repeat above for Forecast

-Repeat above for Alerts

-Connection teardown
Close socket
```

Accelerate your Design Time with EmbeddedDeveloper.com

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

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

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

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

HEARST *business media*
ELECTRONICS GROUP

EMBEDDEDDEVELOPER.COM
FIND. COMPARE. BUY.

ConVergence PROMOTIONS

that time expires.

If you look at the code that's posted on the *Circuit Cellar* FTP site, you'll notice that little code is required to support the W5100. The W5100 does most of the "heavy lifting." It handles the entire Ethernet interface up to the TCP/IP level. It has a simple command interface to load data for sending outgoing data and for reading received data. I ported the W5100 driver code from the first version of the project without too much trouble. The Propeller's SPIN language is similar to C, but it has some interesting nuances, such as the strict use of indentation rather than braces to define statement blocks.

FEATURE CREEP

There are many ways to customize this project to your liking. First, there are other weather data providers. *The Weather Channel* provides data, but its terms of use are a bit strict. The NWS provides data with very few rules, but the interface is more complicated. The advantage of the NWS's data is that it is detailed, so you can see exactly how much snow is predicted and which direction a storm is moving.

Weather is just one type of information the project can display. Other data providers support news and stock updates. Be prepared to learn a new protocol like XML, SOAP, or RSS because most services don't provide simple data interfaces like WeatherBug's pipe delimited format.

My design uses a 64-KB EEPROM, where only 32 KB are used to support the Propeller's 32-KB RAM copy of the program code. You can use the remaining 32 KB to log data or store configuration values. You can also put additional devices on the I²C bus for more storage. In addition, you can use the EEPROM to store image data so that more RAM is available for program code. Images are currently used on the forecast screen and the raw data is combined with the program code.

One nice feature would be to show radar images in a loop. Although it isn't part of the WeatherBug API, you could "get" the image from the WeatherBug web site, store the last four images locally, and set up the display to loop the images. You may want to go with a different microcontroller and TFT to support this feature. The radar images are usually in JPEG format, so you'd need the code to convert the image file to a bitmap. There are a few open-source solutions available, but the memory requirements are significantly greater. The Internet Weather Display's graphics capabilities are somewhat limited, so it might take a lot of work to get a clean radar image. Consider using a higher-resolution TFT with a digital interface.

The Internet Weather Display uses simple tones to alert you when new alarms are detected. The Propeller can generate complex sounds with its StereoSpatializer and VocalTrack libraries. You can use voice announcements or musical tunes if you want sounds that are more pleasing to hear. There are few things worse than waking up to a loud monotone beep during the middle of the night. You can modify the software to play different sounds based on the type of alert. Loud, attention-getting sounds can be used for warnings. A quiet, single "ding" sound could be used for advisories.

Do you want every feature? First off, step away from the "dark side," because you're starting to think like a person in a marketing department. The current version of the Propeller microcontroller has memory limitations, and my code uses just about every byte. The graphics library in Propeller Tool normally uses the double-buffering of a 12-KB image buffer. This project doesn't use animation, so I was able to use only a single buffer and I got back a good amount of memory for program space. There are other Propeller hardware platforms that support larger memory configurations by swapping program code between the RAM and external EEPROM as needed. Also, Parallax is currently working on the next version of the Propeller chip, and it will undoubtedly

include more RAM.

If you want to try and make this into a sellable product, make sure you check the data provider's terms of use. They will likely require some sort of compensation. You will also want to make the code much more robust by adding features such as the ability to select the WeatherBug data source, and the ability for the device to get its IP address using DHCP. All Ethernet devices must have a unique hardware/MAC address. The IEEE administers these addresses, and you can purchase a range. Note that if you use the project's code, both the MAC address and the WeatherBug unique identifier code have been set to illegal values. You must obtain your own unique values. If building a version for your own use, then perhaps use the MAC address from on old PC Ethernet card that's sitting in your junk box. You'll get a unique WeatherBug code when you register at the WeatherBug Labs website.

GO SENSOR-FREE

The Internet Weather Display project enables everyone to have a weather station no matter where they live. Even if you don't face the same restrictions associated with mounting exterior sensors as some other users, this design may be better than a backyard weather station because you receive accurate, professional forecasts and NWS alerts. You don't have to worry about ideal sensor placement or the cost of maintaining and replacing the sensors. You can even modify the design to connect with other data providers to display important data such as news headlines and sound stock market alerts.

Steven Nickels (ssea000@gmail.com) has a B.S. degree in electronic engineering technology from Minnesota State University, Mankato. He is a senior software engineer at Medtronic Navigation in Louisville, CO. Steven has not yet received any complaints from neighbors about the odd-looking equipment around his house.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/228.

RESOURCES

Parallax, Inc., "Propeller Manual," V1.01, www.parallax.com/dl/docs/prod/prop/WebPM-v1.01.pdf

WeatherBug, "WeatherBug API," Pipe Delimited Format, http://weather.weatherbug.com/desktop-weather/api-documents.html.

W5100 and WIZ810MJ Datasheets and information, WIZnet, www.wiznet.co.kr/en.

SOURCES

Propeller P8X32A-40 Microcontroller
Parallax, Inc. | www.parallax.com

WeatherBug Labs API
WeatherBug | http://weather.weatherbug.com/labs.html

W5100 Ethernet controller and WIZ810MJ network module
WIZnet Co. Inc. | www.wiznet.co.kr

by Minas Kalarakis

Web Camera Design

This versatile web camera system can take a picture at a resolution of 640 × 480 or 320 × 240, pan the camera horizontally and vertically, and change its IP and gateway address to match a network. After each photo is divided into 64-byte segments, an Ethernet module transmits the packets over the Internet.

You can use cameras for everything from recording celebrations to building surveillance. One of the most exciting new developments in camera technology is the webcam. At the heart of a webcam is a microcontroller that controls peripheral devices, such as the camera module (camera chip, lens, etc.) and the communications. Due to my interest in cameras, the Internet, and embedded technology, it made sense for me to design my own web camera (see Photo 1).

I built my web camera around a Microchip Technology dsPIC30F4013 16-bit microcontroller, a COMedia C328-7640 serial camera module, a WIZnet WIZ810MJ module, and two standard servos for rotating the camera module (see Figure 1). In this article, I will describe how I did it.

NETWORK MODULE

The WIZ810MJ dictates the way the camera communicates with the other devices in the system (see Figure 2). It is a network module that includes a W5100 TCP/IP hardwired chip (including PHY) and mag-jack (RJ-45 with transformer) with other glue logic.

A network interface card (NIC) must have a unique MAC address. Where can you find a MAC address? You can buy 1,000 to 1,500 MAC addresses from the IEEE, but it can be expensive. Thus, the best option is to use

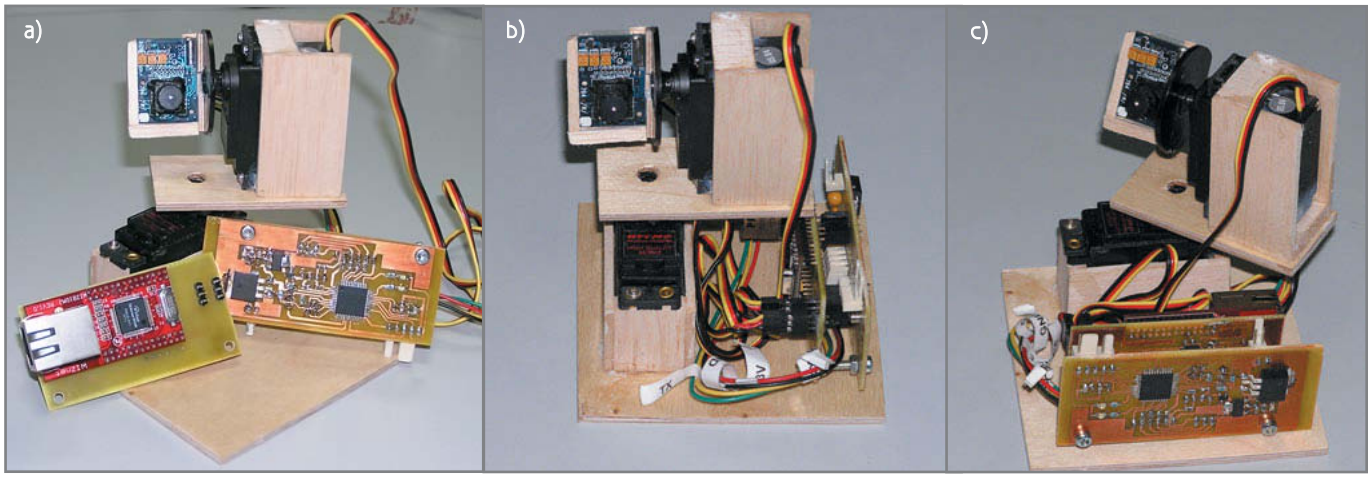


Photo 1a—This is the complete web camera design. **b**—The wiring is fairly simple. **c**—The design features two single-layer boards.

an old Ethernet card. I had one on hand, so I installed it in my PC to get its MAC and then I threw it away. As a result, my camera is recognized as an Intel NIC.

DHCP OR STATIC IP

In addition to a MAC address, the camera needs an IP address. IP addresses can be assigned statically or dynamically. Dynamic assignments are handled by a Dynamic Host Configuration Protocol (DHCP) server. For example, my home network gets IP addresses from the ADSL router's DHCP server, which uses the range of 10.0.0.xx to 10.0.0.137. The router itself has a static IP address of 10.0.0.138. However, if the camera had a dynamically-assigned IP address, none of the other machines would know what that address is and they wouldn't be able to communicate with the camera. Therefore, the camera has a statically-assigned IP address stored in the microcontroller's EEPROM at address 0x7FFC00. The default value for this address is 10.0.0.50 (defined in the source code) and the port is 50000. Of course, this IP address can be changed through the firmware.

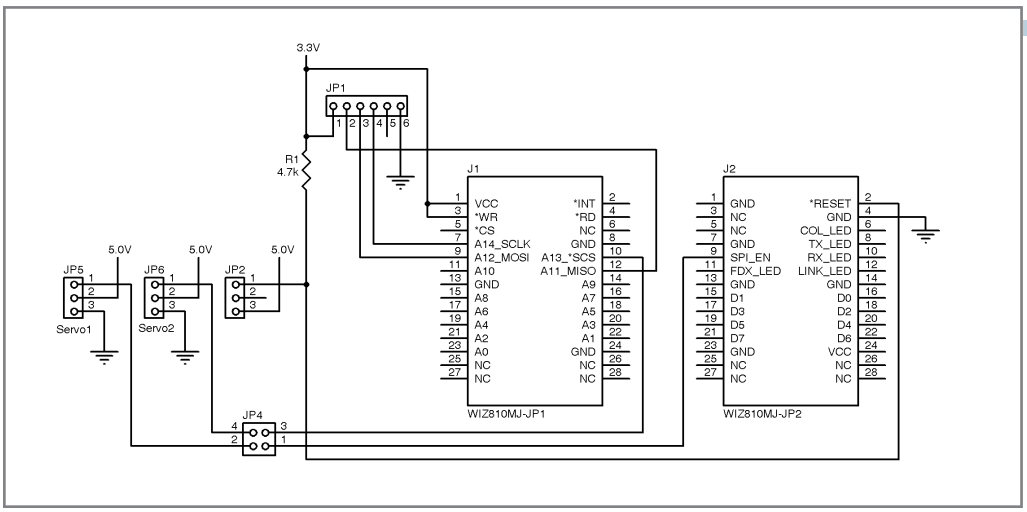


Figure 1—Here you see the WIZnet WIZ810MJ module and two standard servos that are used to rotate the camera module.

Listing 1 is the code that fetches the stored IP address from the EEPROM. Each location in the EEPROM is 16 bits wide and can store 2 bytes. As you can see in Listing 1, the IP address, the SubNet mask and the gateway address are copied to `addr_param[]`, a global variable of type `char`.

Other Functions like `Send_a_UDP_Packet()` can access this array to get the IP address to include with the data that the WIZ810 will send to the host computer. Later in this article, I will describe the function that acquires these parameters and stores them in EEPROM.

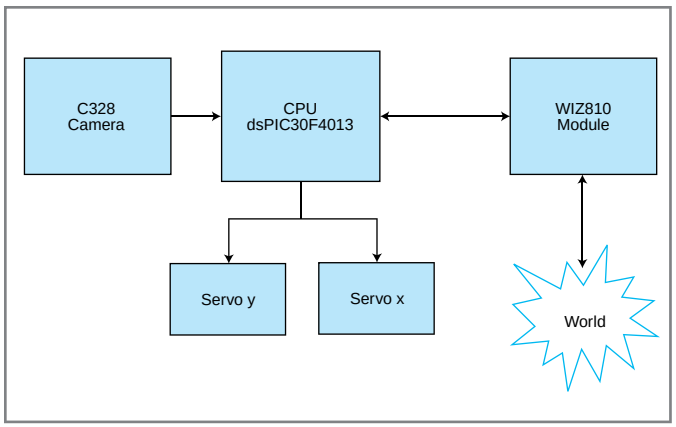


Figure 2—The design is fairly straightforward. A dsPIC30F4013 sits at the center of the design.

TCP OR UDP?

This was my first project using embedded Ethernet. I had to decide between the the transmission control protocol (TCP) or the user datagram protocol (UDP). TCP establishes a connection in advance and delivers the data reliably and in the sequence it was sent. UDP features an unreliable connectionless datagram transmission structure. It processes data without establishing a connection. Therefore, lost or out-of-sequence packets are not hidden from the application. Also, there is no flow control, which means that packets can arrive faster than the recipient can process them.

I transmit photos with this system. So what if I lose one? I can catch the next one.

On one hand, today's computers are fast enough to process multiple applications at the same time. On the other hand, the UDP algorithm, as described in the W5100 datasheet, is simpler than TCP. That's why I chose the UDP protocol.

The WIZnet module can be interfaced via a parallel bus or via SPI. I chose SPI for its low pin count and simplicity. Although the dsPIC30F4013 microcontroller has two serial ports, I may develop future projects with microcontrollers with only one UART. However, UART1 on the '3014 shares pins with the SPI interface, so it is necessary to use a command like `U1MODEbits.ALTIO=1;` to move the UART1 function to alternate pins. I also issued a command for a 200-ms delay because it seemed to need some time to do the job! The SPI port is initialized in 8-bit master mode (see Listing 2).

A Microchip Technology TC2117-33 LDO regulator supplies the WIZ810 with 3.3 V. It draws 146 mA. I should also mention that the WIZ810 draws more current when the cable isn't connected. It gets really hot. All W5100 inputs are 5-V-tolerant, which means that they can be connected directly to the microcontroller. However, for compatibility, I reduced the microprocessor's

Listing 1—IP, Subnet, and Gateway static values are stored in EEPROM starting at location 0x7FFC00. For fast execution, cache all values to the microcontroller memory.

```
for (i=0;i<11;i++){
        Temp=Eeprom_Read(0x7FFC00+i);
        addr_param[i]=(Temp & 0xFF00) >>8;
        addr_param[i+1]=Temp & 0x00FF;
        i++;
}
```

output signal to 3.3 V with a resistor divider (see Figure 3). A 33-Ω resistor is connected in series to eliminate the SDI current. The SPI_EN and the *CS signals are connected to Port B's pins 11 and 12, respectively. The RESET signal is pulled high using a 2.2-kΩ resistor and connected to pin PB10.

CAMERA MODULE

The C328 camera module provides a serial interface (UART) and a JPEG compression engine (see Figure 4). The module consists of three main parts: an OmniVision Technologies OV7640/8 VGA color digital camera chip with an 8-bit YCbCr interface; an OV528 serial bridge, which is an embedded controller chip with a JPEG CODEC that can compress and then transfer image data from the camera chip to external devices; and a program in EEPROM that provides a set of user-friendly commands for interfacing to external host. (This program supports 11 commands for interfacing to the host.)

RESOLUTION

The module can produce pictures in Normal mode (no compression) or Compressed mode (picture compressed with the JPEG algorithm). The maximum resolution in Compressed mode is 640 × 480 pixels. In Normal mode, the resolution is 160 × 120 pixels. For each picture, a total of 57,600 bytes (i.e., 160 × 120 × 3 bytes per pixel) must be transferred to the host.

In practice, the size of a compressed picture file with a resolution of 640 × 480 pixels rarely exceeds a total of 90 KB. For a picture taken inside a room, the file size is about 60 KB. For a half-size picture with a 320 × 240 resolution, the file size will be 30 KB or less. The bytes must be transferred with a slow UART interface at 57,600 bps.

VGA resolution is 640 × 480 pixels with 16 or 256 colors (the display standard for the PC). It was introduced in 1987 with IBM's PS/2 line and was popular in PCs with 14″

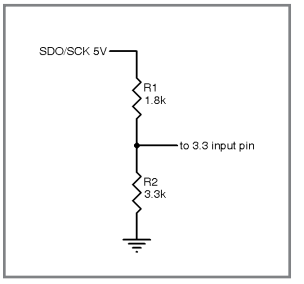


Figure 3—I reduced the microprocessor's output signal to 3.3 V with a resistor divider.

WEST 2009

Conference & Exhibition

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

Conference: September 14 – 18, 2009

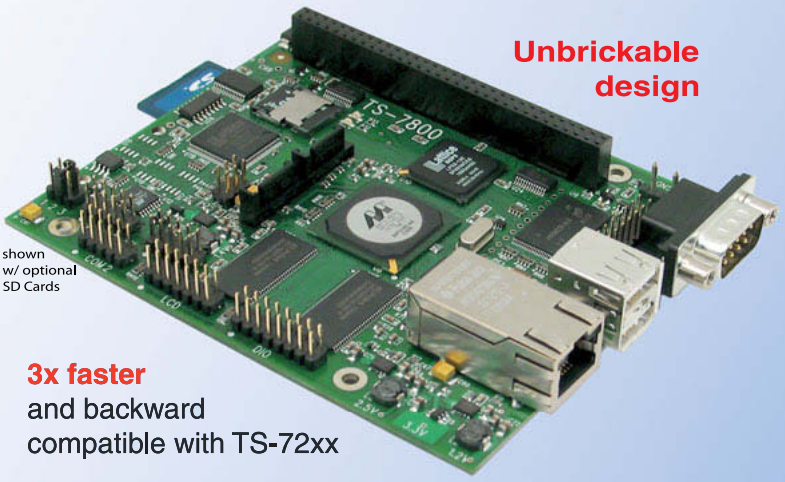
Exhibition: September 15 – 16, 2009

Santa Clara Marriott • Santa Clara, CA

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

Embedded Single Board Computers

High-End Performance with Embedded Ruggedness



shown w/ optional SD Cards

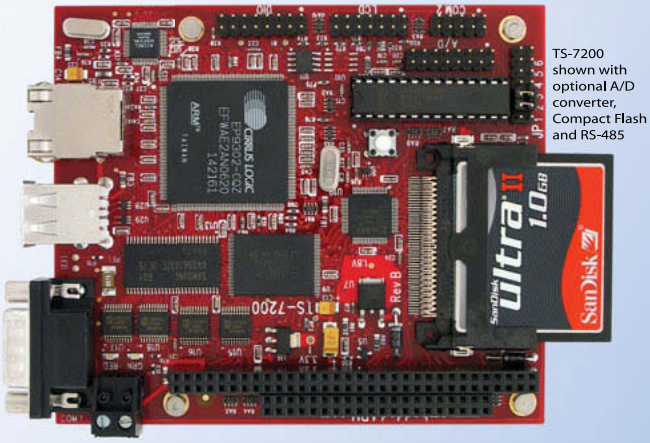
Unbrickable design

3x faster and backward compatible with TS-72xx

TS-7800
500 MHz ARM9

- Low power - 4W@5V — **$229** qty 100
- 128MB DDR RAM
- 512MB high-speed (17MB/sec) onboard Flash — **$269** qty 1
- 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

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 — as low as **$99** qty 100
- Fanless, no heat sink
- SDRAM - up to 128MB — **$129** qty 1
- 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
- SD card option
- VGA video
- LCD ready
- Linux, Real Time extension, Debian

- 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

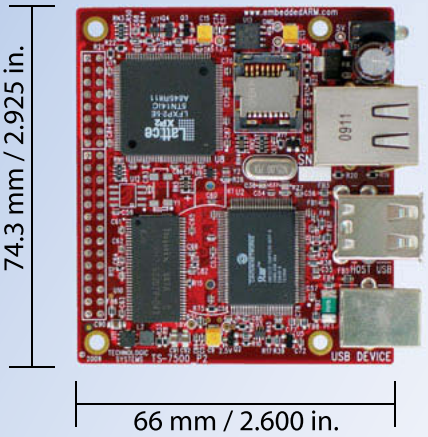
New Products

TS-7500
250 MHz ARM9

NEW!

- 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 < 2 seconds
- Customizable FPGA - 5K LUT
- Power-over-Ethernet ready
- Optional battery backed RTC
- Watchdog Timer
- 8 TTL UART
- 33 DIO, SPI, I^2C

Our Smallest Computer at Our Best Price Point



$84 qty 100

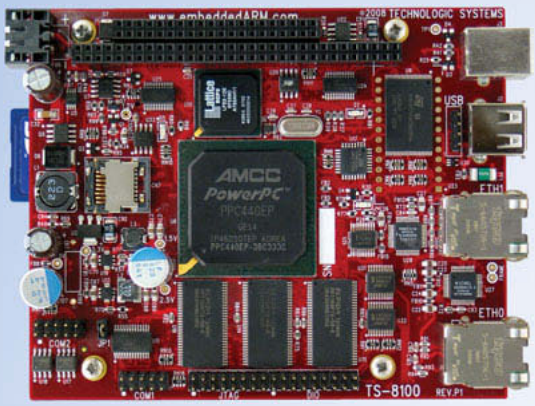
$99 qty 10

TS-8100
400 MHz Dual-Execution PPC

NEW!

- 12K LUT customizable FPGA
- Double-precision FPU
- Multifunctional PC/104 connector
- 128 MB DDR RAM w/ ECC
- 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

50% Faster than 500MHz ARM9 with Floating Point Unit



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

or 15″ monitors. Many new flat panels have a resolution of 1,280 × 1,024 or more. A picture with 640-pixel horizontal resolution covers about only half the monitor.

Modern desktop programming systems have functions that can resize an image. A picture with a 320 × 240 pixel resolution is balanced in quality and file size.

CAMERA INTERFACE

The host (a dsPIC30F4013 microcontroller) must initialize the C328 module after powering it up. Initialization involves transmitting the SYNC command (AA 0D 00 00 00) via the UART until the module sends an acknowledge command (ACK). An ACK command is usually received by the time the SYNC command is sent 60 times. A 10-ms delay must be used between SYNC commands. The best synchronization occurs at 57,600 bps (see Listing 3). At a high speed of 115,200 bps, synchronization can't be achieved. Thus, the best for the host is to power off and on the module. The BSS22 transistor on the PCB acts as a switch to power on and off the module. At the speed of 57,600 bps, the module synchronizes at 60 SYNC commands and never fails. Thus, the firmware doesn't wait to receive the ACK command.

The C328 implements two different communication modes, depending on which command the host sends to get a snapshot picture

Listing 2—In the dsPIC30F4013, UART1 and the SPI share the same bus. Force UART1 to use an alternative bus. The SPI is initialized in 8-bit master mode.

```
Uart1_Init(57600);
U1MODEbits.ALTIO = 1;    // Clear the way for SPI
Delay_ms(200);           // It needs some time to do the job
Spi_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_2,
_SPI_PRESCALE_PRI_1,  _SPI_SS_DISABLE ,
_SPI_DATA_SAMPLE_MIDDLE,_SPI_CLK_IDLE_LOW,_SPI_IDLE_2_ACTIVE);
```

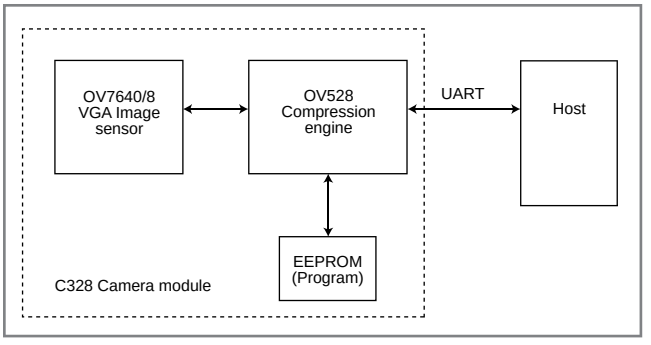


Figure 4—The C328 camera module includes a VGA color digital camera chip and an OV528 serial bridge. A UART is the intermediary between the module and host.

Listing 3—The C328 module must synchronize at the host UART speed. The host must send a special packet several times. An LED connected at pin 13 blinks to indicate this.

```
void camera_connect(){
char i;
   for(i=0;i<70;i++){
         send(0xAA , 0x0D , 0 , 0 , 0 , 0 );  // Sending SYNC packets
         PortCbits.RC13 ^=1;
         delay_ms(10);
   }
   send(0xAA , 0x0E , 0x0D , 0 , 0 , 0 );     // Confirm SYNC with an ACK packet.
   Connected=1;
}
```

(uncompressed or a JPEG picture). In snapshot mode, the host sends a `Get_Picture()` command and the C328 replies with AA A0 01 xx yy zz (3 bytes hold the length of data following these bytes) and all the bytes in the picture's file. This means about 900 KB of data or 160 s for a picture with a resolution of 640 × 480. This mode is unacceptable because 640 × 480 × 3 bytes per pixel = 921,600 bytes × 10 bits/byte = 9,216,000 b/57,600 bps = 160 s per photo, or 2.67 min per photo.

In JPEG mode, the C328 uses the packet method. Before the `Get_Picture()` command, the host issues a command to determine a packet's number of bytes. The default value is 64 bytes long, and the maximum is 512 bytes. After some tests, I found that the best performance was achieved with the default values. Figure 5 illustrates this point, and the `camera_snapshot()` function implements it in code. As you can see, this function requests the picture data. The resolution must be set prior to this function. During the main procedure, the `Get_A_Photo (char resolution)` function is called to set the resolution. But first, it establishes the connection to the camera with the `camera_connect()` function. It sets the resolution with `camera_setup(vgaResolution)` and finally calls the `camera_snapshot()` function to get the data. `camera_snapshot()` is responsible for gathering the packet data and passing them to the array character packet[256]. After a packet is received from the camera, it is acknowledged, and the function passes the data to the WIZ810MJ's socket 0 buffer and does everything required to send it as a UDP packet. It transfers the value of `remote_ip` and remote port to the Socket 0's registers `S0_DIPR` and `S0_DPORT`, respectively. Subsequently, it calculates the start address of data and passes the values to the `S0_TX_WR0` and `S0_TX_WR1` registers. Finally, the host issues a `S0_CR_SEND` command for the data to be sent to the output and clears the send flag to be ready for the next packet. At that point, an enhancement can be made to write the data directly to the socket 0 buffer. To do so, the length of the packet must be set at 512 bytes long (maximum) and the UART speed must be 115 kbps to increase performance. The overall process will increase the frames captured by the host from three to five per minute to three to six per minute. One frame more isn't important at this time.

On the PCB, there are two red LEDs. One is connected on pin RC13 and flashes on SYNC commands. The other is connected on RC14 and flashes once when the camera takes a picture.

The C328 requires 3.3 VDC to work and its I/O is not 5-V-tolerant. Therefore, a resistor divider (R3 and R5) at the dsPIC30F4013's UART2 TX pin is used to convert the 5-V signal to 3.3-V levels. A 33-Ω resistor (R4) was added to the dsPIC30F4013's UART2 Rx pin to

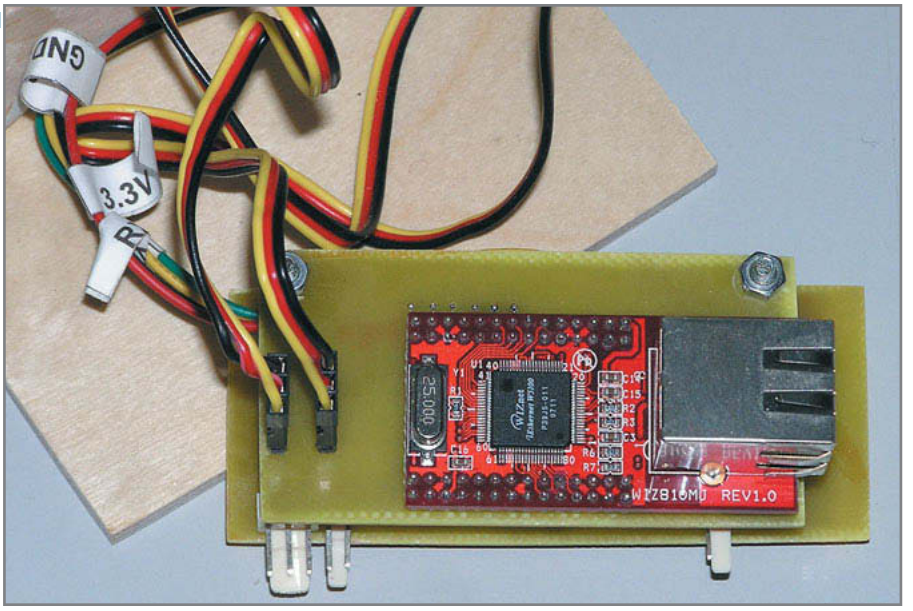


Photo 2—One board holds the dsPIC30F4013. The WIZ810MJ module is mounted on the other.

limit the current.

CAMERA ROTATION

Two standard servos—the second is attached to the shaft of the first servo—rotate the camera horizontally and vertically. The dsPIC30F4013 communicates with servos via pulses. As the host, the dsPIC30F4013 generates a pulse of various lengths approximately every 20 ms.

The pulse's duration applied to the control wire determines the angle of the shaft. This is called pulse width modulation (PWM). The servo expects to see a pulse every 20 ms or so. If the pulse spacing is greater than about 50 ms (manufacturer-dependent), the servo will enter Sleep mode in between pulses. It will move in small steps and the output will be jerky. The off time can vary. This has no adverse effects as long as its value is between approximately 10 to 30 ms. It is only the on time that determines the position of the output arm.

The pulse is normally between 1 and 2 ms long. The length of the pulse is used by the servo to determine the position to which it should rotate. Note that different servos will have different constraints on rotation. However, they all have a neutral position that's always around 1.5 ms (e.g., a Futaba S3003 servo's neutral position is 1,520 µs and its maximum rotation is 1,900 µs). When a pulse is sent to a servo that's less than 1.5 ms, the servo

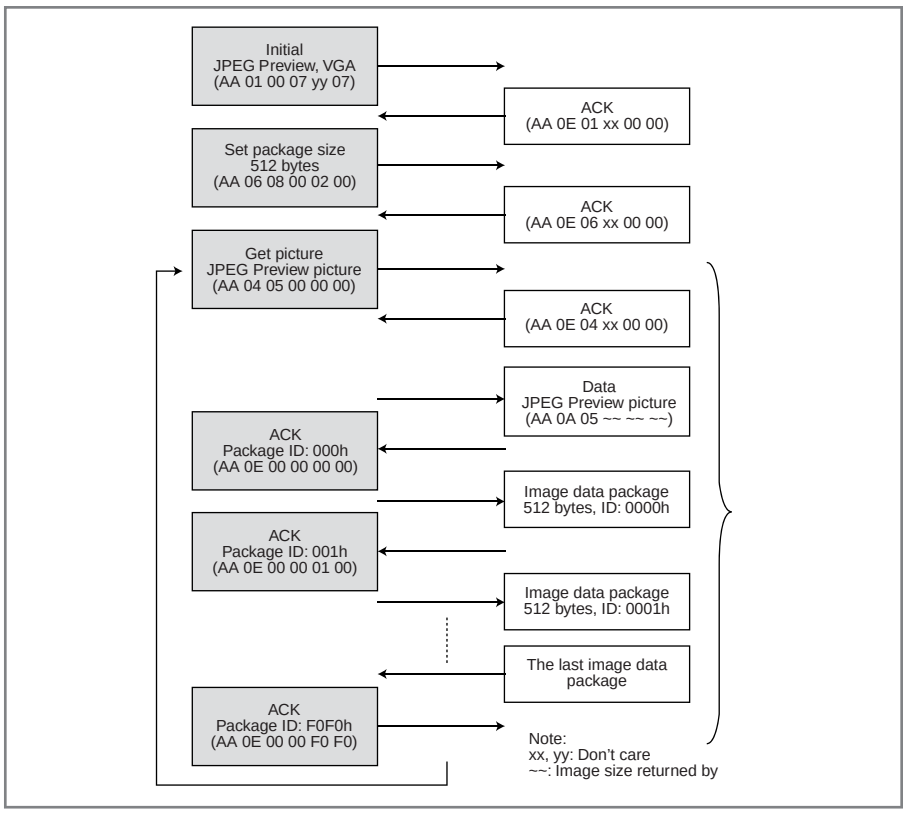


Figure 5—This is the command protocol to get a snapshot. The host is on the left. The host requests the image as packets of a known size. At the end, it acknowledges with a special packet. (Source: COMedia, "C328-7640 User Manual," 2005, www.comedia.com.hk)

rotates its output shaft a number of degrees counterclockwise from the neutral point and holds it there. When the pulse is wider than 1.5 ms, clockwise rotation occurs. Generally, the minimum pulse is about 1 ms wide, and the maximum pulse is 2 ms wide. Because of the hardware (e.g., motor and gears), the servo cannot rotate instantly to the instructed position with one pulse. The host has to issue some pulses to the servo until it reaches the final position.

The nominal supply voltage for the servo is 4.8 to 6.0 V at 7.2 to 8 mA. A Microchip Technology MCP1826S regulator supplies the entire device with 5.0 V at 1,000-mA maximum current. The supply voltage for the servos is settled at 5 V, directly connected to MCP1826S.

The dsPIC30F4013 drives the servos with pulses. This sounds like a good application for the comparator module of the microcontroller. The comparator module is driven by Timer2. The dsPIC has an 8-MHz crystal and the PLL enabled with a multiplier of four, giving a core clock frequency of 32 MHz. Therefore, Timer2 is initialized through the InitTimer2Interrupt() function with the following parameters: dual compare mode, continuous pulses output (OC1CON = 0x0005), rising edge start (OC1R) = 5, falling edge start (OC1RS) = 52. The register PR2 is set at 600. Changing the value of OC1RS affects the "on" time. This can be considered a duty cycle that can be changed at will. The value of 25 corresponds to 0 degrees, and the value of 85 to 180 degrees, with a resolution of 3 degrees per step. Channels 1 and 2 are initialized for the two servos.

HARDWARE

As you can see in Photo 2, the design consists of two single-layer boards. One holds the dsPIC30F4013 and the second is piggy-backed to WIZ810MJ module. The main board is simple. It features a dsPIC30F4013 with an 8-MHz crystal, power supply circuitry, and headers for connecting the board holding the WIZ810MJ (see Figure 6).

A 5-V, 1-A MCP1826S regulator provides power to the system. A TC2117-3.3 linear regulator on the main board provides the 3.3 V

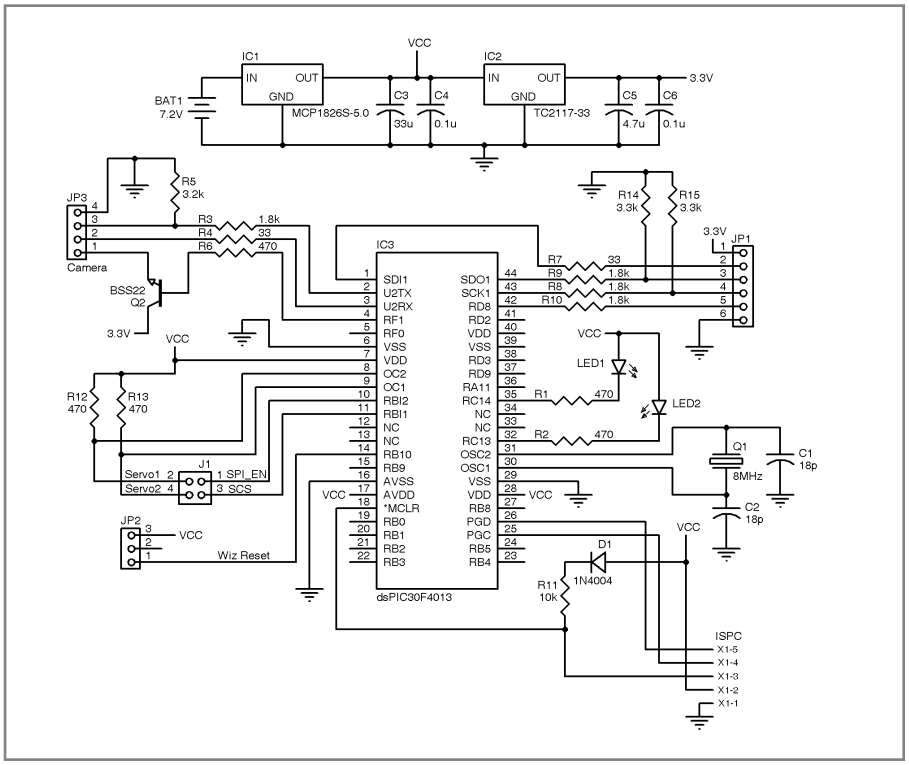


Figure 6—The processor PCB features a dsPIC30F4013, an 8-MHz crystal, power supply circuitry, and headers for connecting to the other PCB.

required by the WIZ810MJ module and the C328 camera. Thus, the WIZ810MJ module and the dsPIC30F4013 are 3.3-/5-V-tolerant, the C328 module is 3.3-V-only so level-shifting circuitry is required. I used level-shifting circuitry for both.

The host communicates with the WIZ810MJ via the SPI bus. It uses the UART port for the C328. A BSS22 transistor acts as a switch to power on and off the C328 camera module.

MAIN BOARD FIRMWARE

The main board's firmware (see Figure 7) was written in the C language. The MikroC compiler was used. The total size of the code is less than 6 KB. MikroC has an evaluation version that works fine with code less than 6 KB, so it is easy to experiment with.

I first read about the WIZ810MJ in Fred Eady's 2007 article "iEthernet Bootcamp: Get Started with the W5100" (*Circuit Cellar* 208). He introduces the W5100 and covers the topic of sockets. In an example, Eady uses the UDP protocol. After I made a decision about the protocol, I started looking for code. The well-written W5100 manual details the process of sending a UDP packet. Searching the Internet, I also found code for Atmel's microprocessors. I keep some header files from that code and the same alias for further reference.

The firmware isn't interrupt-driven. The main function looks for data that have arrived in the WIZ810MJ's Rx buffers by reading the `Sn_RX_RSRx` register. It loops until the arrival of data (see Listing 4). When data arrives, it passes them to a global parameter `Packet[]`. It moves the `buffers` pointer to the new location and writes a 0x04 to `SO_IR` to clear the Receive flag.

Action is taken according to the first byte in the packet. A switch statement takes care of this. There are six different cases, and among them are some that combine actions. For instance, case 53 (ASCII "5") means "Rotate camera then get a photo at 320 × 240 resolution." The PC client can change the web camera's IP Subnet mask value. Case 52 (ASCII "4") takes care of storing the new values to dsPIC30F4013 EEPROM.

I want to bring special attention to the SPI routine. When you try to use the W5100 in SPI mode, even if the *SCS is High, the W5100 (or WIZ810MJ) drives the MISO. To avoid doing so, the functions `wr_wiz_reg(char reg_data, unsigned int reg_addr)` and `rd_wiz_reg(unsigned int reg_addr)` enable SPI_EN and then pull down the *CS signal. They write or read from the module and then they disable SPI_EN and pull up the *CS.

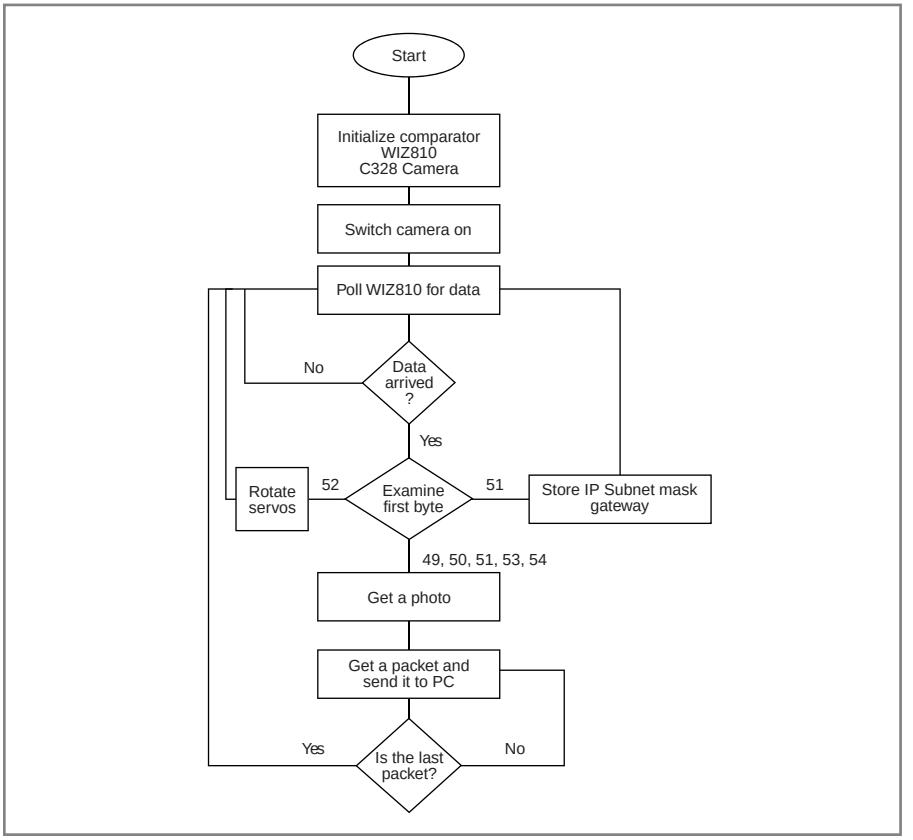


Figure 7—The firmware flow chart is fairly straightforward. It polls the WIZ810 for data. When data arrives, action is taken according to the first received byte.

WEBCAM PROGRAM

Software is required for the desktop PC to communicate with the camera. A PC program was needed to display the photos on its screen. The language is Visual Basic version 6.0 and standard controls were used to enable everyone to experiment with the code.

It all begins with the PC. The client program is responsible for requesting a picture, collecting the packets and checking the photo's integrity, and then displaying it (see Photo 3). In the program's main window, there are two buttons marked 320 × 240 and 640 × 480. After pressing a button, the program sends a UDP packet using the control WinSoc. The packet consists only of 1 byte. This is the character "2" for button 320 × 240 and the character "1" for button 640 × 480. The WinSoc control uses the camera's static IP address and port number to send the packet.

Listing 4—The firmware polls the WIZ810 for data arrived. The Sn_RX_RSRx register holds the size of data arrived. If there is no data, then poll the WIZ810 again.

```
do{
        hi_byte = rd_wiz_reg(Sn_RX_RSR0(0));
        lo_byte = rd_wiz_reg(Sn_RX_RSR1(0));
        get_size = make16(hi_byte,lo_byte);
}while(get_size <=0x0000);     // if no bytes received --> loop
```

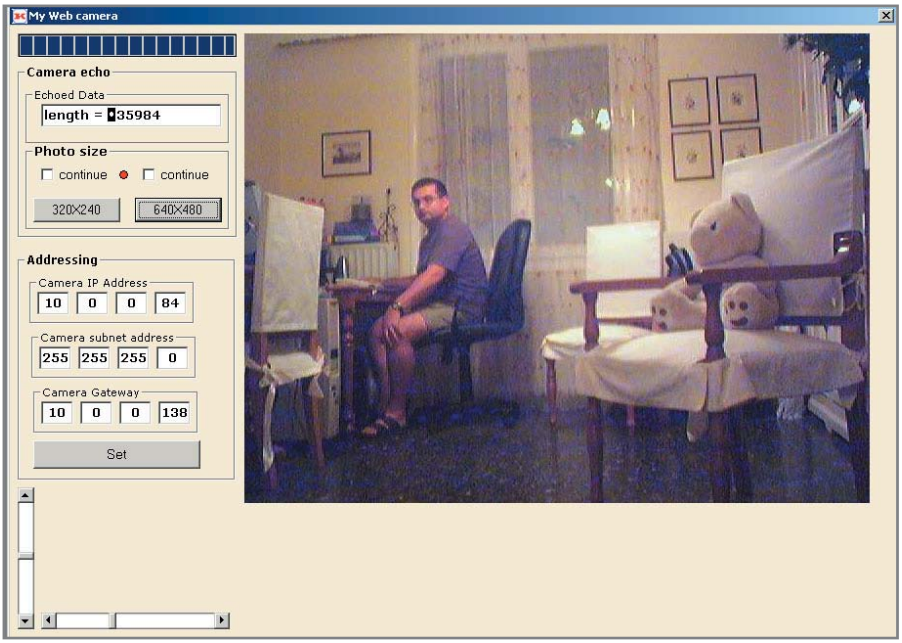


Photo 3—Here you see the client program's main window.

As I already mentioned, action is taken according to the first byte in the packet. A switch case takes care of this. A click on button "320 × 240" sends the character "2" (ASCII 50). The switch case calls the function to get and then send a photo in CCTV resolution.

The WinSoc control listens to the default port, 50000, and collects the data. The WinSoc control's DataArrival event will be raised when data arrives in the default port, 50000. To inform the program how many bytes it has to collect, the first packet includes the logo length=xxxx, where xxxx is the size of the picture. Every packet that arrives adds its bytes on a RAM buffer. When all bytes arrive at the buffer, a `LoadPicture()` function writes them to disk, and then loads the file to a picture control. The picture control is configured to double a picture's width and height. After that, the procedure checks to see if the corresponding "continue" box over the button is checked. If so, it issues a packet with the same character to the camera to get a new photo.

A progress bar on the top shows the progress of the received bytes. There is a Set button in the main window. By clicking this button, you send the camera a packet with all the values shown above. The packet is 13 bytes long. The sliders at the bottom left corner are used to rotate the camera. Every time you move the slider, the slider control fires the change event and a packet with new values are transmitted to the camera.

IMPROVEMENTS

There's more work to be done on the firmware. It isn't interrupt-driven. A new packet can arrive, but it has to wait until an entire photo is sent. This requires a new signal to be added on the PCB to connect the WIZ810MJ interrupt to the microcontroller.

The UART speed should be set to 115 kbps to improve the camera's connection. This requires the firmware to poll the UART RX for the proper answer before moving to the next step. Right now, this design operates with my home ADSL connection at 1024/128 kbps. I can send five frames per minute to my office computer.

Minas Kalarakis (info@kalarakis.gr) holds a B.S. in marine communications from The Naval Marine School of Crete. He is a network administrator and computer technician for The Man Power Organization. Minas's main areas of interest are software and hardware development for embedded systems. In addition his interest in electronics, he enjoys flying RC model aircraft and cycling with his kids.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/228.

RESOURCES

F. Eady, "iEthernet Bootcamp: GetStarted with the W5100," *Circuit Cellar* 208, 2007.

WIZnet, "WIZ810MJ Datasheet," Ver.1.2, 2008, www.wiznet.co.kr/en/pro02.php?&ss[2]=2&page=1&num=23.

SOURCES

COMedia C328-7640 Serial camera module
COMedia | www.comedia.com.hk
Electronics123.com (distributor) | www.electronics123.com

dsPIC30F4013 Microcontroller, MCP1826S regulator, and TC2117-33 regulator
Microchip Technology, Inc. | www.microchip.com

C compiler
mikroElektronika | www.mikroe.com/en/compilers/mikroc/dspic/

OV528 Serial bridge and OV7640/8 VGA Color digital camera chip
OmniVision Technologies, Inc. | www.ovt.com

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

by Kirt Weakman

iMash

An Ethernet-Controlled HERMS

After reading a 2006 *Circuit Cellar* article about the development of a heat exchange recirculating mash system (HERMS), Kirt set out to update his own Ethernet-controlled design. In this article, he describes how upgrading the system involved metal bending, TIG welding, MIG welding, embedded Linux code, VB.NET code, and more.

Homebrewing is an extremely diverse hobby. On one end of the spectrum, you can make beer using simple equipment such as a stock pot and the old stove top. On the other end, you find complex systems utilizing control loops and advanced equipment that are comparable to process-control technologies in industry. I think that is why homebrewing appeals to me. It's an endeavor that can accommodate diverse interests and skill levels.

Beer making involves basically four principal ingredients: water, malted barley, hops, and yeast. The water allows starches within the malted barley to dissolve, thus enabling activated enzymes to convert it into fermentable and non-fermentable sugars. Generally, hops are added to this sugar solution and boiled for a certain period of time. Hops provide bitterness to balance the sweetness, aroma for fragrance, and a preservative. Once cooled to a certain temperature (approximately 68°F), yeast is added. The yeast comprises the single-cell organisms that consume the fermentable sugars and produce CO_2 and alcohol. The beer is then allowed to ferment until it is ready to be transferred to another holding vessel or to be served.

Homebrewers usually categorize the brewing process and use terms like extract, mini-mash, or all-grain. These terms describe the process of converting the malted barley into a

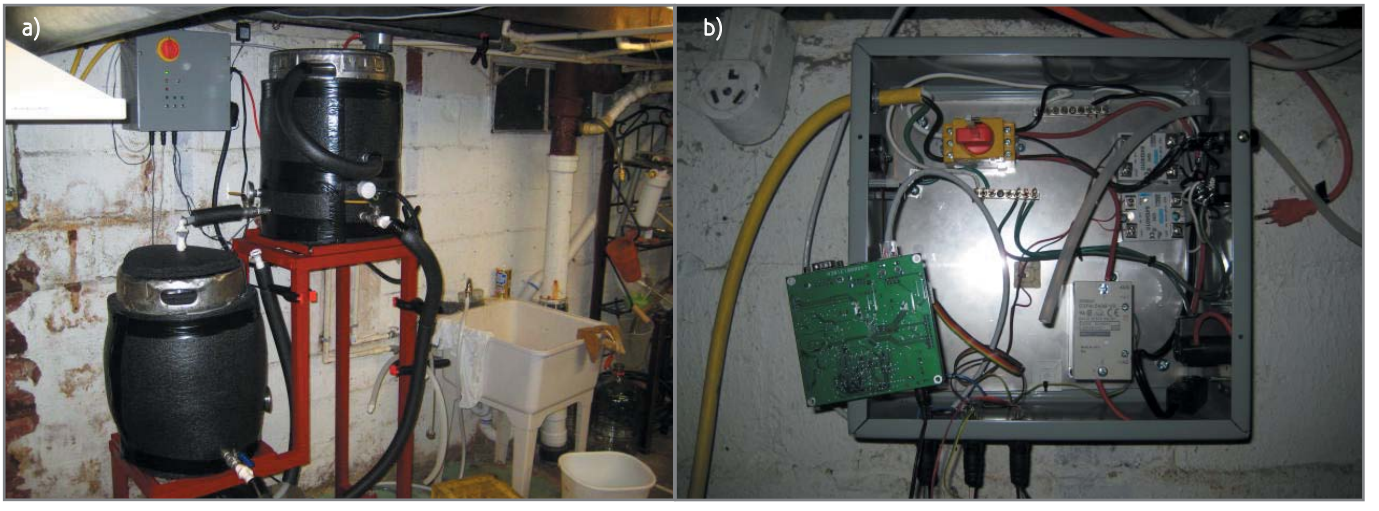


Photo 1a—Check out my homebrewed HERMS. I've used the system to brew two batches of beer: American Hefeweizen and American Amber. **b**—Take a look inside the iMash enclosure. Three solid-state relays along with the SBC are mounted inside. A disconnect switch is integrated into the assembly to manually control the power. An Ethernet cable and quick connectors allow the unit to interface with the system.

fermentable product the yeast can consume. Extract brewing is generally the simplest and requires the least amount of equipment. The malted barley has already undergone the starch conversion process and is packaged in either a liquid or dried form. As an example, these are the metal cans you usually see with the appealing graphics that make you want to brew. Moving up the ladder, mini-mashing is similar to extract brewing, with the exception that a small amount (usually less than 1 lb) of actual malted barley is steeped in the brew pot before the extract is added. This adds some taste complexity to the finished beer. Finally, all-grain is the process in which milled malted barley is mixed with hot water and allowed to set to foster the sugar conversion process. The grain and water (called mash) are then rinsed with more hot water (called sparge water) and transferred to a boiling kettle. At that point, all three processes generally go through the same process to become beer.

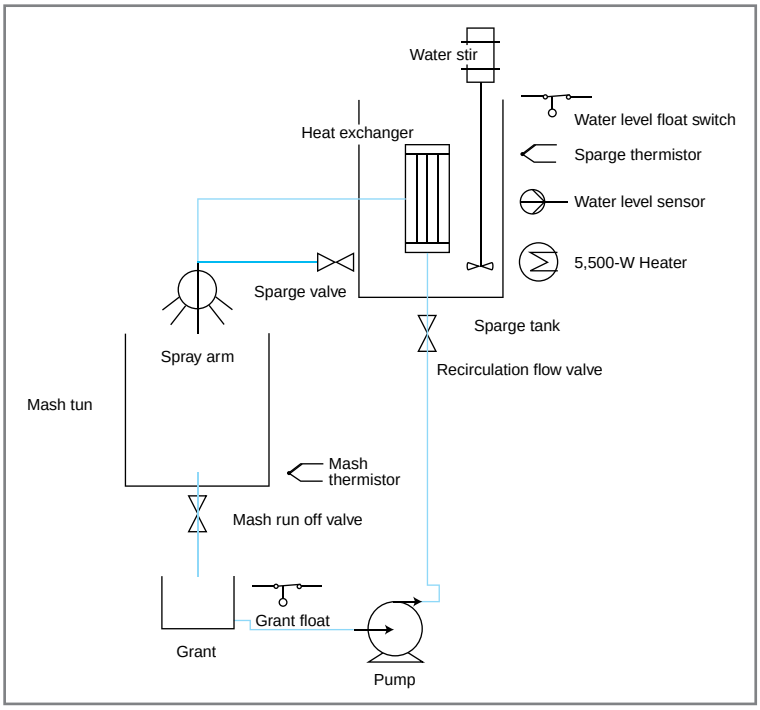


Figure 1—Here you see the system's hardware and sensors. There are three vessels: the sparge tank, the mash tun, and the grant.

I've been brewing for close to 12 years, starting with extract brewing and slowly working my way to all-grain. My old system employed a cooler to hold the water/grain mix during the conversion. This generally worked pretty well, but it had some disadvantages. First, it would drop the temperature by about 4° over the conversion period. Second, there was variation in the target temperature of the water added to the grain. Third, some grains like wheat were not fully modified. This basically meant that to optimize the characteristics of the grain (amount of sugar converted, protein levels, and

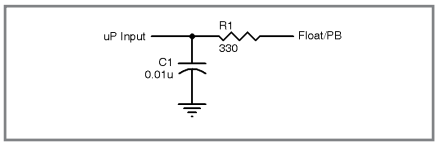


Figure 2—The digital input circuit was used to read float switches and the interface panel switch inputs.

head retention) the grain had to go through several different ranges of temperatures. This was not easy to do in a cooler since I could not heat it directly.

After reading Mark Nesdoly's 2006 article "Home-Brewed HERMS" (*Circuit Cellar* 191) and doing some research on the Internet, I was inspired to update my system (see Photo 1). Nesdoly details the difference between various system architectures and provides some control background on the process. I decided to upgrade my cooler setup to a heat exchange recirculating mash system (HERMS). This control methodology has the capability to actively control

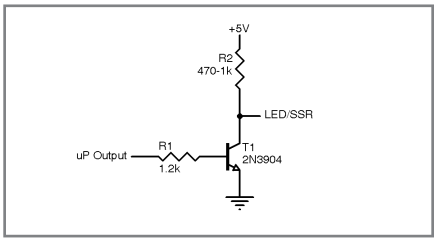


Figure 3—The digital output circuit was used to control various LEDs, SSRs, and buzzer outputs.

the mash temperature, thus addressing the problems I had with the cooler system. I also wanted to focus on simplicity and system robustness. The heart of the control system is a Glomation GESBC-9302E embedded single board computer (SBC) running Embedded Linux. The SBC implemented a network socket communication scheme back to a Visual Basic .NET application for interface and control purposes. Temperature sensing was accomplished by fabricating two thermistor probes that utilized high-precision thermistors in a Wheatstone bridge configuration. Temperature was controlled with a 5,500-W heating element (water heater element available at your local hardware store) along with a small stir motor. Each was controlled by a solid state relay (SSR). A March 809-HS pump and SSR were implemented to recirculate the mash. The water level was monitored in the sparge tank using a Motorola MPXV400X series pressure transducer. Two fluid level floats were used detect liquid levels for the heater and pump logic.

SYSTEM DESIGN

Most HERMS I read about on the Internet and in *Circuit Cellar* were conceptually similar in that they were designed to control the temperature of

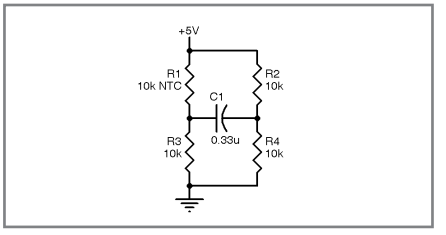


Figure 4—This is a Wheatstone bridge with an RC filter. Two of these were implemented to read temperature (mash thermistor and sparge thermistor).

the recirculating mash. However, there is a plethora of implementations. Some implement gas burners, some use a dedicated vessel for the heat exchanger, and there is a myriad of control implementations (from elegant to scary). My design goals were simplicity, robustness, and cost.

After much iteration, I decided on the system detailed in Figure 1. (An additional schematic is available on the *Circuit Cellar* FTP site.) The sparge tank contains the heat exchanger, heater, water float switch, thermistor, and level sensor. The mash tun contains a thermistor and a false bottom. The latter separates the mash liquid from the grains so the liquid can be extracted and recirculated. A small vessel below the mash tun, called a "grant," holds the liquid for the pump intake. A float is used to determine if the grant is empty so the pump won't run dry. The grant helps eliminate any vacuum from forming between the pump and mash tun. Ideally, we want the grain bed inside the mash tun to form a filter. A vacuum could potentially interfere with this filter.

I designed my system to utilize both gravity and an electric pump to recirculate and transfer water and mash liquid. The benefit is that I can use the pump to recirculate and later transfer the mash liquid to my boil kettle. At the same time, I can use gravity to have the water in the sparge tank rinse the mash. This methodology saves me the addition of another pump and control logic. I wrapped all the vessels with high-temperature insulation. This helps maintain the vessel temperatures to provide tighter temperature control. For convenience, the heater system is totally electric. The main boil kettle is gas fired, but the sparge tank heater is electric. The entire system operates from 240 VAC.

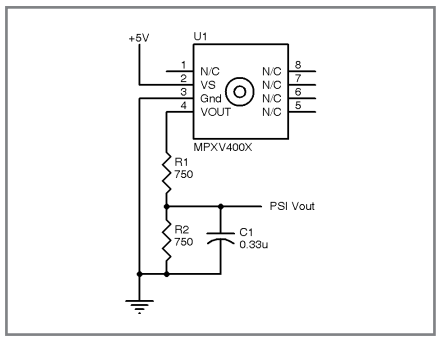


Figure 5—This is the pressure sensor interface circuit. The sensor was implemented to read the water level in the sparge tank.

HARDWARE

When I first started this project, I considered using a general-purpose microcontroller for the system.

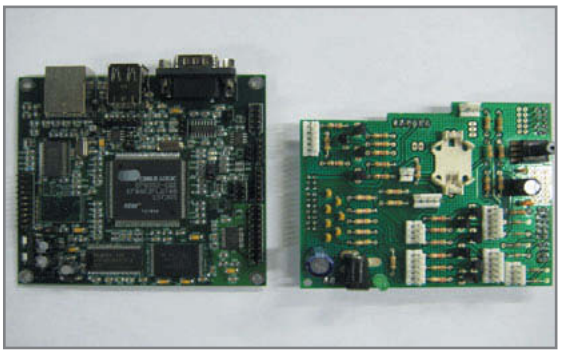


Photo 2—The custom-designed daughterboard sits on top of a Glomation SBC.

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

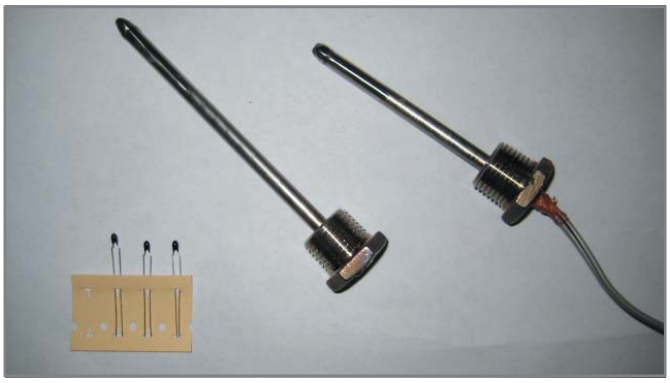


Photo 3—These thermistor assemblies were made by TIG welding stainless steel tubing to 0.50″ threaded and drilled plugs. The thermistors are shown in the bottom-left corner. Wires were soldered along with heat-shrink tubing to form the electrical assembly.

There are numerous free or low-cost programmers and compilers available, and the development process is pretty straight forward. One of my main goals was to be able to log the mash cycle in a Microsoft Excel CSV file. I wanted to be able to program the mash cycle using an Excel worksheet (because documentation is key to successful repeatable brewing). I did not want to use a serial interface, and a Bluetooth interface was more than I wanted to invest. So, I used my home Ethernet because it was there. Any computer in the house can potentially interface with the system.

The GESBC-9302E SBC SBC has many useful features like Ethernet, USB, serial, EEPROM, DIO, and optional data acquisition (analog-to-digital, 14 bit – 8 channels). Additionally, the Cirrus ARM SoC chip on the board has a pretty decent ARM Linux support package implementing the Linux 2.6.24 kernel.

I used Eagle CAD to design a top interface daughter board for the power, DIO, and A/D interface circuitry. The interface circuits are pretty straightforward. The digital input circuit in Figure 2 shows the RC filter interface circuit used to read the float switches and the switch positions on the user interface panel. Digital output circuit in Figure 3 uses a simple NPN transistor to switch various loads such as the SSRs, LEDs, and buzzer. The SBC is populated with a Texas Instruments ADS7871 14-bit SPI ADC. This part is capable of operating in a differential mode, so an RC-filtered Wheatstone bridge circuit provides an adequate interface to achieve high resolution and low noise. A Wheatstone bridge enables the reading of the temperature of the mash and sparge tanks. Figure 4 shows a Wheatstone bridge with an RC filter.

A Freescale pressure transducer connected to one of the open single-ended inputs on the ADC reads the sparge tank's water level. Figure 5 shows the pressure sensor interface circuit. Photo 2 shows how the two boards are interfaced together. The completed assembly produces a modular control in a rather small footprint.

A 40-A SSR and two 25-A SSR (purchased from your friendly Internet auction site) are mounted inside a standard 12×12×6 metal enclosure. A disconnect switch completely shuts off the system when it isn't in use. I did not purposely design in redundant safety mechanisms. I wanted to have the system in a known safe mode and not have to rely on software or other interface logic to sense failed components.

CONSTRUCTION

I am an advocate of having any material that comes into contact with the mash or water be constructed of NFS materials such as copper or stainless steel. These materials are simple to clean and do not leach any chemicals.

The first order of business was to construct the thermistor probes (see Photo 3). I used 0.25″ hollow stainless tube and TIG welded the end closed and then welded the tube to a drilled out 0.50″ plug. The NTC thermistor was then configured and placed inside the tube.

Once complete, I modified my old sparge tank by welding in various 0.50″ couplings and a 1″ threaded nut for the heater. I used 30′ of 0.50″ OD stainless tubing to form the heat exchanger. I opted for stainless because of its cost and the performance. Refer to Photo 4 for a look inside the sparge tank.

Next up, I welded another coupling to the mash tun to house the thermistor. Infused with the welding spirit, I went ahead and MIG welded up a new red frame to showcase the new system.

SOFTWARE & SBC

There are two primary areas of software work, the embedded Linux in the SBC and a VB.NET application on the PC. I architected the system to operate in a client server mode. The SBC was configured as the server and the PC as the client. I chose this because the SBC might be completely stand-alone in the future. The two applications use a predefined network socket at any internal IP address to communicate.

The Cirrus Linux latest release (at the time I'm writing

Photo 4—Take a look at the sparge tank. You can see the heat exchanger, heater, and the sensors. I took this photo after the stir motor was attached (key learning).

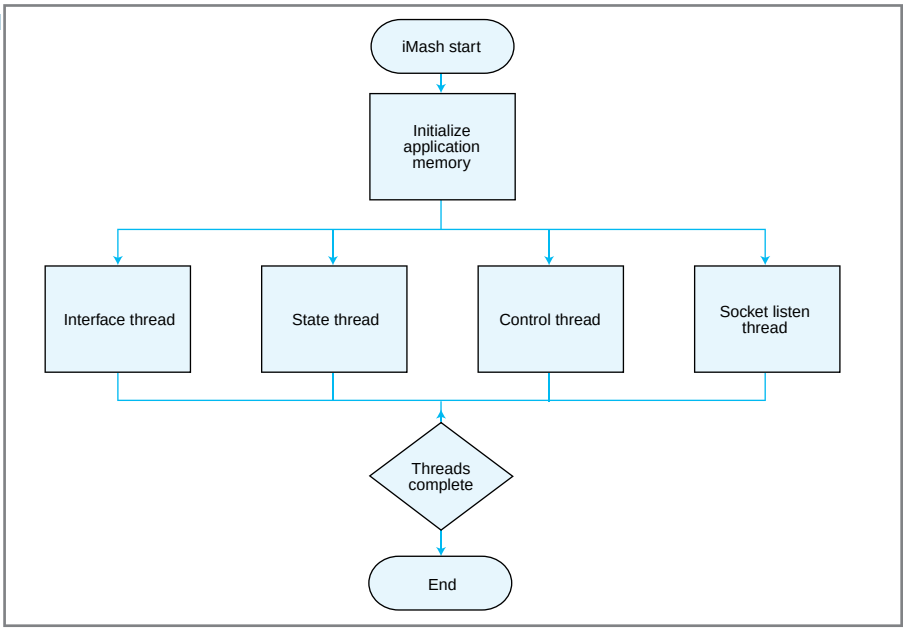


Figure 6—The SBC software architecture includes various threads. Each thread handles a certain aspect of the application.

this article) is Crater 1-0-3. I downloaded the release, built the board, and had it running code in a few hours. Admittedly, I had some prior experience with this board, so using the latest release was pretty painless. I updated some of the build options using the Make menuconfig for the board configuration and the kernel release. I added the support of I²C for the on-board EEPROM and I updated the driver for the on-board RTC. I had to edit the data rate for the serial terminal settings to 57,600 bps. Otherwise, by default, the kernel would send out the boot-up sequences over the terminal at a different data rate than what Busybox used. If you're unfamiliar with Busybox, it's pretty much a bunch of core support applications targeted toward the embedded environment. Some examples would be date, ls, shell utilities, top, tftp, and so on.

Once the Linux operating system was up and running on the SBC, my next step was to start application development. I configured the Eclipse IDE to edit and compile the application code. I was not successful in getting the GNU debugger (GDB) operating. As a result, most of my debugging efforts were focused using `printf` statements and log files. It was not the most efficient debugging environment. I tested some pieces of code by compiling on my Linux Debian desktop before deploying them on the target board. Once the socket interface was functioning, I was able to write pieces of test code using VB.NET.

When the application is launched, it uses p-threading to create four threads. Each thread handles some special aspect of functionality. Figure 6 shows the SBC software architecture and the various threads. The interface thread handles all of the low-level I/O manipulation and the SPI to read the ADC. The state thread manages the overall system state controller, which can put the control in standby, diagnostics, running, and more. It also enables loads like the pump and heater to turn on and off based on the status of the recipe being executed. The controller thread is the heart of the temperature control. It contains the PID algorithm and software PWM control for the heater SSR. The PID algorithm was written using a couple PID implementation white papers as references. I chose to implement the floating library over using fixed-point math because there were no hard timing requirements. The PWM had a 5-s period because the thermal response of the system is rather slow. The PWM is evaluated every 100 ms (5 s/100 ms = 50 control

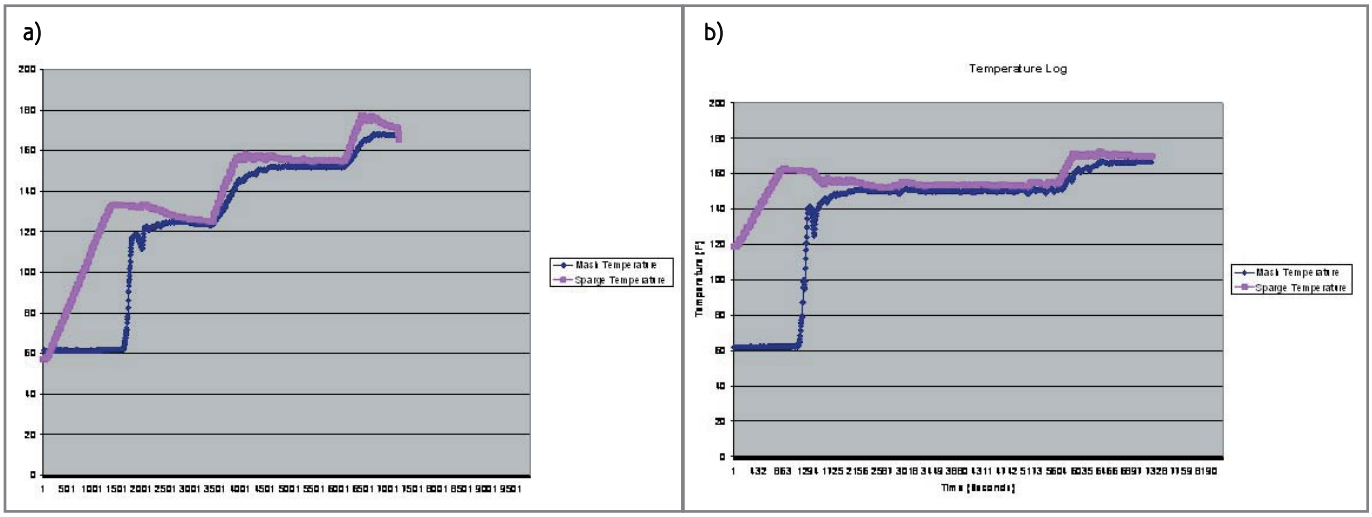


Figure 7a—These are the three temperature steps for the American Hefeweizen. b—These are the two temperature steps used to make the American Amber.

points). The SSR has a specification of 10 Hz as the fastest control frequency. The socket listen thread manages all of the network traffic to communicate with the VB.NET application.

VB.NET

The VB.NET application development was pretty straightforward. If you are used to the "old" VB 6.0 days, there might be some transitioning over to the .NET version. The .NET version is essentially class-based, which at the end of the day makes the code much more structured.

I used VB.NET because I have Visual Studio on my machine. If you don't have Visual Studio, Microsoft has express editions you can download. They're pretty sweet and will enable you to do nearly everything you need to make a Windows application. When I needed a strip charting graph, I discovered the Zed graph open-source project. I configured the graph component to create a nice strip chart that included a lot of cool features like zoom and scale.

TEST RESULTS

I performed the first test just to see if the system would hold water. It almost passed. I found a small fracture in one of the joints and repaired it. Once water tight, I applied power to the entire system and started evaluating.

The first issue came from the sparge tank thermistor. The reading topped out at about 149°F and pretty much stopped. I measured the voltage across the Wheatstone and went back to the datasheets. After a short period of time, I realized the thermistor's look-up table was set up for the wrong thermistor part number. A new table and 5 minutes later I was up and running. The system began heating and everything seemed well, except the bottom of the sparge tank was cold and the top was pretty hot to the touch.

I made another discovery. If you are going to temperature-control a water bath, you need to recirculate or agitate the water to ensure you don't have any thermal pockets. Next thing you know, the sparge tank is undergoing some more TIG welding to support a small AC motor to stir the water. The SBC and VB.NET code was modified with the new stir logic and the entire system was back up and running. This time, success!

A MULTIFACETED PROJECT

Since the initial successful test, I've used the system to make two batches of beer. The first was an American Hefeweizen. The second was an American Amber.

The mash temperature profiles in Figure 7 showcase the system's response. So far, I'm extremely pleased with the results. I have the system dialed in so that I can maintain close to less than 1° to 1.5° of variation through the brewing session. From a process control perspective, the project was a success. The real proof will be in the finished product.

In the future, I might make the system more stand-alone. It is a nice option to be able to download various mash recipes, but I think eventually just a handful will be needed. I could use the USB interface and thumb drive on the SBC to log data and possibly store the recipe profiles. I don't use the USB functionality at this time, although the drivers are all there to support it. I have also considered using either a serial LCD or USB LCD to enhance the system's ability to display data.

This multifaceted project involved metal bending, TIG welding, MIG welding, embedded Linux code, and VB.NET code. I found myself wearing many hats. One hour, I was welding stainless and the next I was back on the keyboard writing interface code. I guess that's why homebrewing appeals to me.

Kirt Weakman (weakmank@netscape.net) earned a BSEET and an MST from Purdue University. He has been designing and writing code for embedded systems for more than 10 years. Kirt is currently a senior engineer at a major appliance manufacturer. He enjoys homebrewing, kayaking, camping with his family, and driving and restoring Volkswagen buses and Beetles.

PROJECT FILES

To download the code and schematic, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/228.

RESOURCES

Atmel Corp., "Discrete PID controller," AVR221, 2006, www.atmel.com/dyn/resources/prod_documents/doc2558.pdf.

Cirrus Logic, ARM Cirrus Linux Support, 2007, http://arm.cirrus.com/files/index.php?path=linux%2Freleases%2Flinux-2.6%2F1.0.3/.

J. Palmer, "How to Brew," 1999, www.howtobrew.com.

C. Valenti, "Implementing a PID Controller Using a PIC18 MCU," AN937, Microchip Technology, 2004, www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en020434.

Zed Graph Project, 2007, http://zedgraph.org/wiki/index.php?title=Main_Page.

SOURCES

GESBC-9302E SBC
Glomation | www.glomationinc.com

MPXV400X Pressure transducer
Motorola, Inc. | www.motorola.com

ADS7871 14-bit SPI ADC
Texas Instruments, Inc. | www.ti.com

by Enoch Hwang

Master Control

Implement an I²C Master Bus Controller in an FPGA

Enoch explains how to implement an I²C master bus controller using an FPGA. He presents the I²C bus protocol, and then describes how to build the controller using VHDL or Verilog code. The design accesses a real-time clock chip via the I²C bus and displays the time on a seven-segment display.

The physical size of ICs has reduced dramatically over the years. The main reason, of course, is attributed to the fact that more and more transistors can be cramped into a smaller space. A less-mentioned reason is because the pins for interconnections between ICs have also decreased both in size and in number. As you probably know, the actual circuitry of the IC is much smaller than the packaging of the IC. The reason that the packaging has to be larger is the larger number of pins needed for the connections to the PCB. One solution for reducing the packaging size is therefore to have fewer pins for the connections. For example, back in the 1980s, the National Semiconductor MM58167B was a popular real-time clock (RTC) chip. It is a dual-in-line package with 24 pins. Of the 24 pins, four (chip select, read, write, and ready) are for interfacing control, five (A0 to A4) are for addressing, and eight (D0 to D7) are for data. So, 17 out of the 24 pins are used just for sending and receiving data/commands between the RTC and the microcontroller. A newer RTC chip, the Epson RTC-58321, still uses 10 out of 16 total pins dedicated for communication between the chip and the microcontroller. In order to reduce the connection pin counts even further, Philips (NXP Semiconductors) developed the I²C protocol, which requires only two lines for communication between two or more chips.

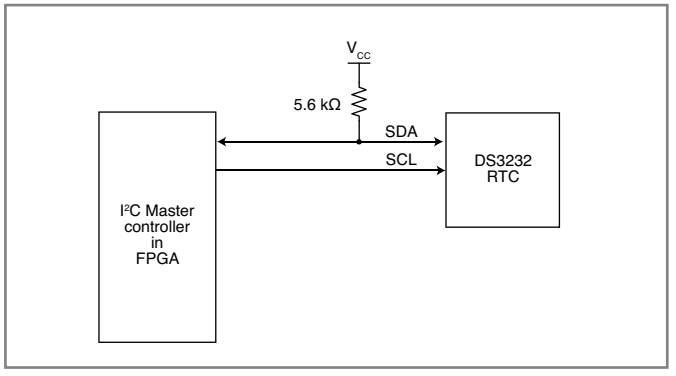


Figure 1—This is the I²C bus system with the I²C master controller implemented in an FPGA and the real-time clock device acting as the slave. There is only one master, so only the SDA line needs to be pulled up with a 5.6-kΩ resistor. The SCL line is controlled by the master controller.

In this article, I will show you how to implement an I²C master bus controller using a field-programmable gate array (FPGA). An FPGA is a chip that can be used to implement any digital logic circuit. I will demonstrate the master controller's operation by having it communicate with an RTC chip, the Maxim DS3232, connected on the I²C bus as a slave. Using the I²C bus as the communication channel, the master controller can send and receive data to and from the slave. In order to show the design of the I²C master bus controller as simply as possible, some of the subtle details in the I²C specifications are not fully implemented. Hence, there might be situations where this master controller might report a false error.

I²C BUS

The inter-integrated circuit (I²C) bus is a simple bidirectional serial bus that supports multiple masters and slaves. It consists of only two lines: a serial bidirectional

data line (SDA) and a serial bidirectional clock line (SCL). Within the I²C bus specifications, two modes are defined: a standard mode with a maximum clock rate of 100 kHz and a fast mode with a maximum clock rate of 400 kHz.

Each device connected to the I²C bus is software-addressable by a unique address, and a simple master/slave relationship exists at all times among the devices. The device that controls the sending and receiving of messages by controlling the bus access is referred to as the master. Devices that are controlled by the master are the slaves. Both the master and the slave can send and receive messages. A device that sends data to the bus is referred to as the transmitter. A device receiving data is referred to as the receiver. More than one master and more than one slave can all coexist on the same I²C bus. However, the bus is always controlled by a single master that's responsible for generating the serial clock (SCL) and controlling the bus access by initiating and terminating a message transfer.

IC TO I²C CONNECTION

As I mentioned earlier, the I²C bus consists of only two lines, SDA and SCL. Both lines are open-drained, and must be pulled up to V_{CC} with a 5.6-kΩ resistor. Regardless of how many devices are connected to the bus, only one pull-up resister is needed per line. Furthermore, the SCL line needs to be pulled up with a resistor only if there will be two or more masters in the system, or when the slave will do clock stretching as a flow-control measure. In the first case, the pull-up resistor on the SCL line is required for arbitration purposes when two or more masters try to initiate a data transfer at the same time. An arbitration procedure has been defined to avoid the chaos that might ensue from such an event. In the second case, a receiver may not be ready to receive data from the transmitter, and so it will hold the SCL line low to "stretch" the clock to delay the transmitter.

As you can see in Figure 1, the example consists of only one master and one slave. To make my master controller as simple as possible to understand, I assumed that the slave will not do any clock stretching. (If the slave does attempt to perform clock stretching, it will sink all of the current that the master can supply on the SCL line, and physical damage may result.) Hence, I did not use a pull-up resister on the SCL line. Note that this implementation of the master controller does not follow the full specifications of the I²C. Nevertheless, after understanding how the controller is designed, you can very easily modify it to follow the full specifications. My I²C master controller is implemented in an Altera Cyclone II FPGA, and the slave is the Maxim DS3232 real-time clock chip. The master is in control of the SCL line at all times.

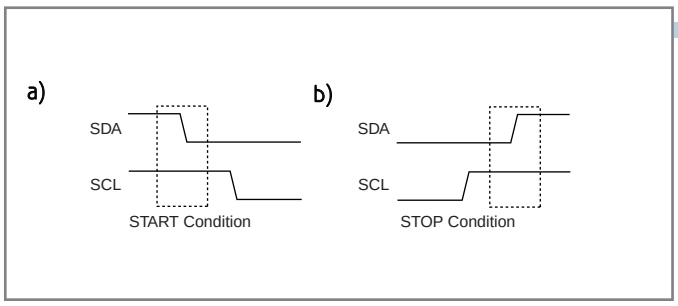


Figure 2—The START (a) and STOP (b) conditions are both initiated by the master. The START condition happens when the SDA line changes from a high to a low while the SCL line is at a high. The STOP condition happens when the SDA line changes from a low to a high while the SCL line is at a high. These are the only two situations where the SDA line can change when SCL is at a high.

I²C PROTOCOL

The I²C bus is idle when both SCL and SDA are at a logic 1 level. The master initiates a data transfer by issuing a START condition, which is a high-to-low transition on the SDA line, while the SCL line is high (see Figure 2a). The bus is considered to be busy after the START condition. After the START condition, a slave address is sent out on the bus by the master. This address is 7 bits long followed by an eighth bit, which is a data direction bit where a 0 indicates a write from the master to the slave and a 1 indicates a read from the slave to the master. The master, which is controlling the SCL line, sends out the bits on the SDA line, one bit per clock cycle of the SCL line, with the most significant bit sent out first. The value on the SDA line can be changed only when the SCL line is at a low.

The slave device—whose address matches the address being sent out by the master—will respond with an acknowledgment bit on the SDA line by

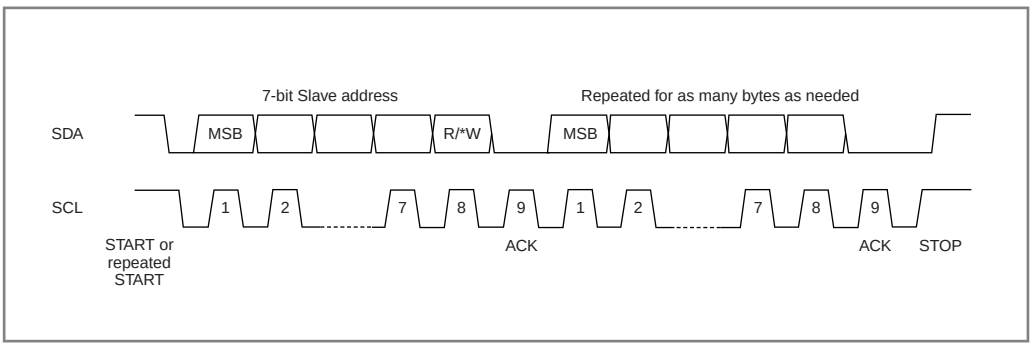


Figure 3—The master initiates data transmission by sending the START condition. For every 8 bits of data transmitted, the receiver sends an acknowledgment during the ninth clock cycle by pulling SDA low. The only exception is when the master-receiver wants to end the data transmission in which case the master (who is the receiver) will not acknowledge by keeping SDA high. Data transmission is terminated by the master sending the STOP condition.

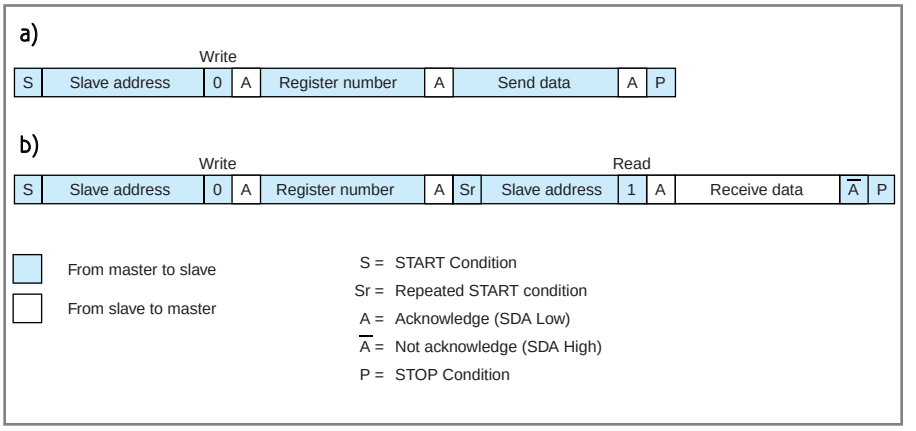


Figure 4a—The master transmits 1 byte of data to the slave. **b**—The master receives 1 byte of data from the slave. In both cases, the master first sends the 7-bit slave address and the write bit, followed by the register number to access. In the master-transmitter scenario **(a)**, the master can immediately send out the data byte since the data direction is still a write. For the master-receiver scenario **(b)**, the master has to do a repeated START and resend the slave address with the read bit in order to change the direction of the data transmission from a write to a read. After this, the master can then receive a byte of data from the slave from the given register number.

pulling the SDA line low during the SCL line's ninth clock cycle (see Figure 3). The direction bit R/*W determines whether the master or the slave will be the transmitter in the subsequent data transmission after the sending of the slave address.

Every byte put on the SDA line for transmission must be 8-bits long with the most significant bit first. Except for the START and STOP conditions, the SDA line must not change when the SCL line is high. The number of bytes that can be transmitted is unrestricted. Each byte has to be followed by an acknowledge bit. The master generates the acknowledge-related clock pulse. The transmitter releases the SDA line (sets it to high impedance) during the acknowledge clock pulse, and the receiver must pull down the SDA line during the acknowledge clock pulse to acknowledge the receipt of the byte. The one exception is when a master-receiver is involved in a transfer. In this case, the master-receiver must signal the end of data to the slave-transmitter by not generating an acknowledgement on the last byte clocked out of the slave.

To signal the end of data transfer, the master sends a STOP condition by pulling the SDA line from low to high, while the SCL line is at a high (see Figure 2b). Instead of sending a STOP condition, the master can send a repeated START condition so that it can change the direction of the data transmission without having to release the bus.

Figure 4a shows the scenario where the master writes 1 byte of

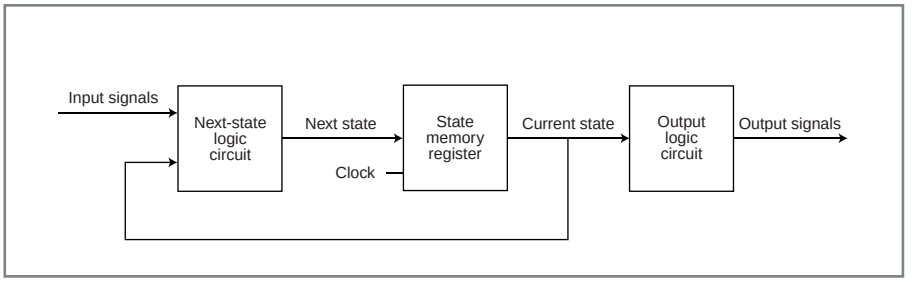


Figure 5—This is a finite state machine (FSM). There are three main components: the state memory to store the FSM's current state, the next-state logic to determine the next state to go to (depending on the current state and the input signals), and the output logic to generate the appropriate output signals from the controller.

We Speak Embedded.

Design Excellence

Robert Brunner
CEO of Ammunition
Renowned Industrial Designer of such product lines as the Apple II, Macintosh, Newton, and PowerBook

Creating Technology Solutions

Tom Scholz
Founder of Scholz Research and Development
Member of the band Boston and inventor of the Rockman headphone guitar amplifier

Driving Corporate Excellence

T.J. Rodgers
Founder, President, Chief Executive Officer & Director of Cypress Semiconductor Corp.

3 Keynotes. 85 Sessions. 75 Speakers.

Training + Education= All the Answers You Need.

ESC Boston is a must-attend event for embedded systems engineers. You can customize your educational experience by selecting from over 85 sessions in 20 tracks specific to your interests. It is the place for you to identify solutions to immediate design challenges and meet in person the solution providers for your next project.

ESC Boston is the place for the embedded community to learn today to design tomorrow. Register now at www.embedded.com/boston

Register Today.

www.embedded.com/boston

Learn today. Design tomorrow.

ESC

Boston • Hynes Convention Center

data to the slave (i.e., the master is the transmitter and the slave is the receiver). The master initiates the data transfer by first issuing the START condition followed by the 7-bit slave address plus the write (0) bit. After receiving an acknowledgment from the slave, the master sends the register number to let the slave know which register the following data is to be written into. Again the slave responds with an acknowledgment. The master then sends the data byte to the slave. After the slave acknowledges the receipt of the data byte, the master sends the STOP condition.

Figure 4b shows the scenario where the master reads 1 byte of data from the slave (i.e., the master is the receiver and the slave is the transmitter). The master initiates the data transfer by first issuing the START condition followed by the 7-bit slave address plus the write (0) bit. Although the master wants to receive a byte, I need to send the register address byte first to let the slave know which register the master wants to read from. After receiving an acknowledgment from the slave, the master sends the register number to let the slave know which register to read from. The slave responds again with an acknowledgment. This time the master has to do a repeated START condition because it needs to change the data direction from a write to a read. The repeated START is followed by the slave address again, but this time with the read (1) bit instead. The slave acknowledges and then sends the data byte from the addressed register to the master. This time, because the master is the receiver, the master has to acknowledge the receipt of the data byte. If the master wants to receive more data bytes from the slave, it sends a 0 to acknowledge it. If the master doesn't want to receive any more bytes, it won't acknowledge by keeping SDA at a high. Finally, the master sends the STOP condition.

Listing 1—This is the VHDL code template for an FSM. The language syntax is not case-sensitive. However, in the example, all keywords are in uppercase, and all user identifiers are in lowercase or a mixture of uppercase and lowercase.

```
-- FSM template
-- Copyright Enoch Hwang 2008

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY I2C_controller IS
PORT (
     Clock, ResetN: IN STD_LOGIC;
     --I2C control output signals
     scl: OUT STD_LOGIC;
     sda: OUT STD_LOGIC;
END I2C_controller;

ARCHITECTURE fsmd OF I2C_controller IS
     SIGNAL state: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN

     PROCESS(Clock, ResetN)
     BEGIN
          IF(ResetN = '0') THEN
               scl <= '1';
               sda <= '1';
               state <= x"00";

          ELSIF(Clock'EVENT AND Clock = '1') THEN
               CASE state IS
               WHEN x"00" =>        -- Idle
                       scl <= '1';  -- SCL = 1
                       sda <= '1';  -- SDA = 1
                       state <= x"01";

               WHEN x"01" =>        -- Start
                       scl <= '1';  -- SCL stays at 1 while
                       sda <= '0';  -- SDA changes from 1 to 0
                       state <= x"02";

               -- remaining states here

               WHEN OTHERS =>
                       scl <= '1';
                       sda <= '1';
               END CASE;
          END IF;
     END PROCESS;

END fsmd;
```

CONTROLLER DESIGN

The trick for implementing the I²C master controller—or any controller for that matter—is to use a finite-state machine (FSM). Manually designing an FSM requires some knowledge of digital logic design. This is beyond the scope of this article. (Refer to the Resources listed at the end of this article if you want to learn more about digital logic design.) However, even without any knowledge in digital logic design, you can still easily implement an FSM by writing VHDL or Verilog code.

VHDL and Verilog are two popular

Listing 2—Fragments of the VHDL code for the I²C master controller show the implementation of the SDA signal. The first line declares the interface signal `sda` as `INOUT` for bidirectional data transfer. The second line declares an internal signal named `sda01`, which will be assigned the logical values of 1 and 0. Next, a conditional assignment statement is used to set the `sda` line to either the high impedance value `'Z'` or the logic value 0 depending on the value of `sda01`. `sda` gets a Z value if `sda01` is a 1; otherwise, `sda` gets a 0 value. The last two lines show the actual logical value of 1 and 0 being assigned to the internal signal `sda01` instead of directly to the interface signal `sda`.

```
...
sda: INOUT STD_LOGIC;
...
SIGNAL sda01: STD_LOGIC; -- internal SDA having values of 0 and 1
...
sda <= 'Z' WHEN sda01 = '1' ELSE '0';  -- convert SDA 0/1 to 0/Z
...
sda01 <= '1';                          -- SDA = Z
...
sda01 <= '0';                          -- SDA = 0
...
```

Listing 3—These fragments of the VHDL code for the I²C master controller show the generation of the SCL signal. A second clock divider process is used to generate a 200-kHz clock. This 200-kHz clock drives the FSM process for controlling the SCL line. In state x"02" in this process, SCL is set to 0. In state x"03", SCL is set to a 1. It takes two states or two cycles to generate one cycle on the SCL line; therefore, the SCL clock speed is at 100 kHz.

```
...
    CLK_50_MHz: IN STD_LOGIC;
    ...
    -- constants for 200kHz clock divider
    -- to get 100kHz for I2C standard; every 2 FSM cycles = 1 I2C cycle
    CONSTANT max200k: INTEGER := 50000000/(100000*2); -- = 250
    SIGNAL clockticks200k: INTEGER RANGE 0 TO max200k;
    SIGNAL CLK_200k_Hz: STD_LOGIC;
    ...
    fsm: PROCESS(CLK_200k_Hz, ResetN)
    ...
           ELSIF(CLK_200k_Hz'EVENT and CLK_200k_Hz = '1') THEN
                  CASE state IS
    ...
                  WHEN x"02" =>
                         SCL <= '0';
                         state <= x"03";
                  WHEN x"03" =>
                         SCL <= '1';
                         IF (bitcount - 1) >= 0 THEN
                                bitcount <= bitcount - 1;
                                state <= x"02";
                         ELSE
                                bitcount <= 7;
                                state <= x"12";
                         END IF;

    clockdivider200k: PROCESS
    BEGIN
           WAIT UNTIL CLK_50_MHz'EVENT and CLK_50_MHz = '1';
           IF clockticks200k < max200k THEN
                  clockticks200k <= clockticks200k + 1;
           ELSE
                  clockticks200k <= 0;
           END IF;
           IF clockticks200k < max200k/2 THEN
                  CLK_200k_Hz <= '0';
           ELSE
                  CLK_200k_Hz <= '1';
           END IF;
    END PROCESS;
```

hardware description languages (HDL) for designing digital circuits. If you know how to write C code, you will quickly feel comfortable with writing VHDL or Verilog code because many software constructs are identical. It is just a matter of learning the new syntax.

Let's now focus on how the I²C master controller is designed using VHDL.

FINITE STATE MACHINE

Before looking at the VHDL code for the I²C controller, I want to cover the topic of finite-state machines. A finite-state machine is a sequential circuit that uses (as the name suggests) a finite number of states to keep track of its history of operations. Based on this history and its current inputs, it determines what to do next. A sequential circuit is one where its outputs are dependent on its history of operation and its current inputs. The FSM's current state is stored in memory. The FSM's operation is simply to traverse from one state to another by changing the memory's content and to output the necessary control signals from the controller in the appropriate state. The FSM determines the next state to go to by considering the current state that it is in and the input signals to it.

As you can see in Figure 5, an FSM consists of three main parts: the state memory for storing the FSM's current state, the next-state logic for determining the FSM's next state to go to, and the output logic for generating the appropriate output signals from the controller for controlling particular devices. The next-state logic is dependent on the current state that the FSM is in and input signals to it. The input signals can be either external signals to the controller or conditional signals generated within the controller circuit itself. Output signal generation is dependent on the FSM's current state. In some FSMs, the output signals are dependent not only on the current state but also on the input signals. The timing for the FSM is controlled by a clock signal. At every clock cycle, the FSM goes to a new

state and generates a new set of output signals.

VHDL CODE

Listing 1 shows a VHDL code template for describing an FSM. The language syntax is not case-sensitive; however, in the example, all keywords are uppercase and all user identifiers are lowercase or a mixture of uppercase and lowercase. The first two lines with the two dashes are comments. The next three lines are standard library files to include. Among other things, these library files define the STD_LOGIC and STD_LOGIC_VECTOR type that are used later on.

The ENTITY section—which begins with the ENTITY keyword and ends with the END I2C_controller line—defines the interface between the module (black box) and the outside. It includes all the necessary input and output signals to the module. In the example, there are the clock and the reset input signals, and the two output signals scl and sda.

The ARCHITECTURE section defines the operations of the module, which in this case is the FSM, and therefore must contain the three parts of an FSM as modeled in Figure 5. The state variable declared using the SIGNAL keyword is the state memory. It is of type STD_LOGIC_VECTOR, which is an 8-bit bit string.

The PROCESS block specifies that whenever there is a change in either of the two signals, Clock and ResetN, the statements inside the block will be executed in sequential order starting with the first line. I have an active-low reset signal as specified in the IF statement that tests for the signal being a 0. When ResetN is asserted (i.e., when ResetN is equal to 0), the module goes into the reset mode and outputs a logic 1 value for both the scl and sda output signals. Furthermore, it assigns state x"00" as the initial state for when the FSM starts. x"00" is the syntax for the two hexadecimal digits 00.

When ResetN is de-asserted, the ELSIF statement is executed. The condition, Clock'EVENT AND Clock = '1', specified inside the ELSIF statement checks for a rising clock edge. So, at every rising clock edge, the FSM goes to a new state and a new set of output signals is generated.

The state that the FSM is currently in is implemented using the CASE statement that tests for the value in the state variable. The CASE statement is the same as the SWITCH statement in C, which is like a nested IF statement.

The different states that the FSM can go to are defined in the different cases using the WHEN keyword. So, at start-up, the FSM goes to state x"00" because the case for WHEN x"00" is true. In this state, the FSM generates a logic 1 signal for the two output signals scl and sda. In this state, it also assigns x"01" to the state variable, which means that the next state the FSM will go to will be state x"01". At the next rising clock edge, when it executes the CASE statement, the state variable now has the value x"01" so it will go to the WHEN x"01" case. In state x"01", the FSM assigns a logic 1 to scl, a logic 0 to sda, and x"02" as

Listing 4—This portion of the VHDL code for the I²C master controller shows the sending of the slave address and the write bit to the RTC.

```
ARCHITECTURE FSMD OF I2C_controller IS
  -- I2C address of the slave + write'
  CONSTANT SlaveAddress_Write: STD_LOGIC_VECTOR(7 DOWNTO
0):="1101000"&'0';
     ...
        WHEN x"00" =>      -- Idle
             -- when idle, both SDA and SCL = 1
             scl <= '1';    -- SCL = 1
             sda01 <= '1';  -- SDA = 1
             state <= x"01";
        WHEN x"01" =>      -- Start
             scl <= '1';    -- SCL stays at 1 while
             sda01 <= '0';  -- SDA changes from 1 to 0
             bitcount <= 7; -- starting bit count
             state <= x"02";
        -- send 7-bit slave address followed by R/W' bit, MSB first
        WHEN x"02" =>
             scl <= '0';
             sda01 <= SlaveAddress_Write(bitcount);
             state <= x"03";
        WHEN x"03" =>
             scl <= '1';
             IF (bitcount - 1) >= 0 THEN
                          bitcount <= bitcount - 1;
                          state <= x"02";
             ELSE
                          bitcount <= 7;
                          state <= x"12";
             END IF;
        -- get acknowledgment' from slave
        WHEN x"12" =>
             scl <= '0';
             sda01 <= '1';
             state <= x"13";
        WHEN x"13" =>
             scl <= '1';
             Ack <= sda;    -- 0 = acknowledge'; error if it is a 1
             IF sda = '1' THEN
                          RegisterAddressOut <= x"13";
                          state <= x"EE"; -- acknowledge error
             ELSE
                          state <= x"20"; -- send register address
             END IF;
```

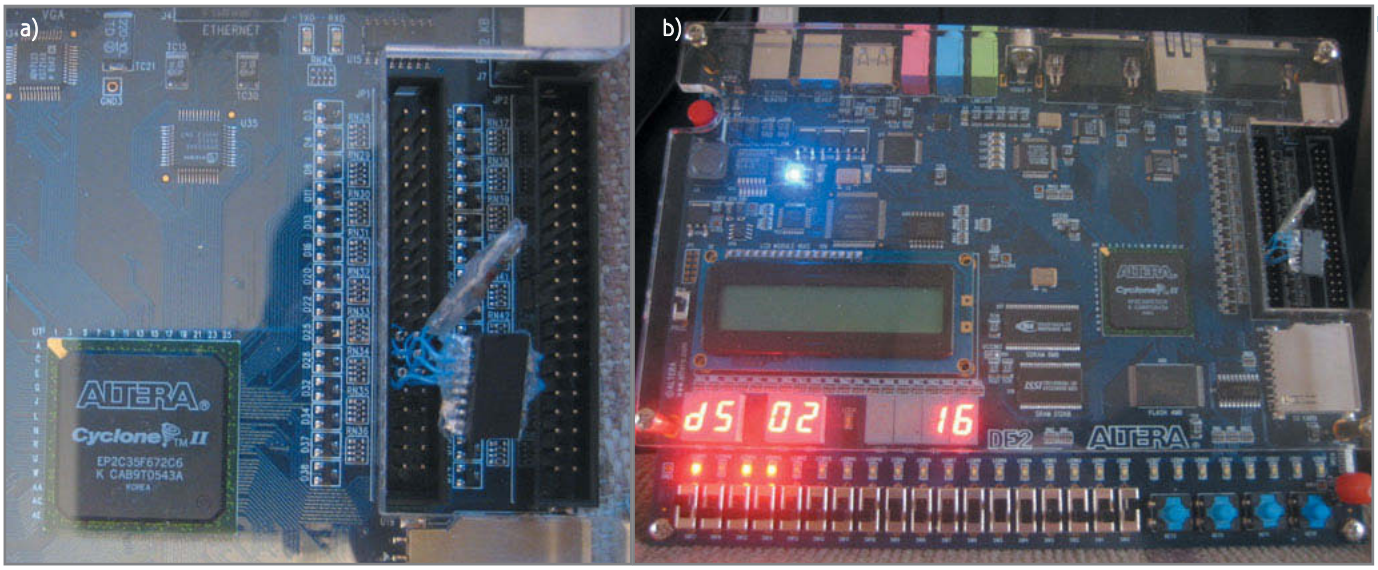


Photo 1a—The DS3232 RTC chip is connected to the I²C master controller implemented inside the Cyclone FPGA via the GPIO pins. In addition to the RTC chip connected to the GPIO is also the 5.6-kΩ pull-up resistor for the SDA line. **b**—The Altera DE2 board with the seven-segment display shows (from left to right): the FSM state d5, which the FSM is currently in; the slave's register 02 that the FSM is reading data from; and the data 16 that is stored in register 02. The right-most eight switches at the bottom of the board are configured for inputting 8-bit data. The next eight switches are for inputting an 8-bit address. The left-most switch is the direction switch for read or write.

the next state to go to.

I²C MASTER CONTROLLER

I am now ready to fill in the details pertaining to my VHDL FSM code template to implement the I²C master controller. Of course, the output signals to generate and the input signals to test for must follow the I²C protocol as described previously. Furthermore, the FSM's clock speed must also satisfy the speed defined in the I²C protocol. The complete code is posted on the *Circuit Cellar* FTP site. Relevant code fragments are shown in Listing 2, Listing 3, and Listing 4.

The implementation of the `sda` signal requires some special attention (see Listing 2). Recall that the SDA line in the I²C bus is bidirectional. In the `ENTITY` declaration, the `sda` signal has to be declared as `INOUT` so that it is capable of doing both input and output. Furthermore, the SDA line in the I²C bus is open-drained and is pulled up by a 5.6-kΩ resistor. So, to output a logic 1 on this line, I need to actually set the line to a high impedance. To get a high impedance, I need to use a tri-state output and assign to it a `'Z'` value.

Refer to the following conditional signal assignment statement:

```
sda <= 'Z' WHEN sda01 = '1' ELSE '0';
```

It assigns a high impedance value `'Z'` to the `sda` line when the internal signal `sda01` has a logic 1 value; otherwise, the `sda` line gets the logic value 0. Hence, to assign the logic value 0 or 1 to the `sda` line, I would instead assign the value 0 or 1 to the internal signal `sda01`, which in turn sets the `sda` line to 0 or Z, respectively.

Listing 3 shows the relevant VHDL code for the generation of the `scl` clock signal. The `scl` signal is generated in the FSM process by going back and forth between two states: the first state sets `scl` to a 0 and the second state sets `scl` to a 1. Therefore, for every `scl` cycle, the FSM must go through two states or two cycles. Hence, to get a 100-kHz speed for the `scl` signal, which is the I²C standard mode maximum clock speed, the FSM clock speed must run two times faster or at 200 kHz. Our primary input clock speed is 50 MHz. I need a clock divider to slow the clock down to 200 kHz. The clock divider process counts the 50-MHz clock ticks from 0 to 250 (i.e., 50,000,000/200,000). To get a 50% duty cycle for the 200-kHz clock, the clock signal CLK_200k_Hz toggles after every 125 counts (i.e., 250/2).

The `fsm` process executes whenever there is a change in the `CLK_200k_Hz` signal. The `ELSIF (CLK_200k_Hz'EVENT and CLK_200k_Hz = '1')` statement says that the `CASE` statement is executed only at the rising edge of the `CLK_200k_Hz` clock. The code fragment shown in Listing 3 shows that the FSM goes back and forth between state `x"02"` and state `x"03"`. In state `x"02"`, `scl` is assigned a 0. In state `x"03"`, `scl` is assigned a 1. The `IF` statement in state `x"03"` tests when to exit this loop and continues to state `x"12"`.

Listing 4 shows the portion of the VHDL code that sends the start condition signal, followed by the slave address of 1101000, plus the write bit of 0 to the RTC slave. The signals to output or input on the `scl` and `sda` lines follow the I²C protocol, as I already mentioned. The initial state, `x"00"`, is the I²C idle state when both the clock and data lines are at a logic 1 level. In the next state, `x"01"`, I send out a start signal by assigning a 0 to `sda01` while `scl` remains at a 1. Recall from the

earlier discussion that sda gets the value 0 when sda01 is a 0. In state x"02", we assign one bit of the slave address to sda01 when scl is at a 0. The variable bitcount is used to index the bit string SlaveAddress_Write, which contains the 7-bit slave address plus the write bit.

In state x"03", sda remains stable while scl is at a 1. States x"02" and x"03" repeat eight times for sending out the 8 bits in SlaveAddress_Write. After sending out the last bit, the FSM goes to state x"12" to get an acknowledgment from the slave. This is accomplished by first setting the sda line to high-impedance 'Z' in state x"12", and then reading the sda line in state x"13" to see if the slave has set it to a 0. In state x"13", if sda is a 0, then the slave has acknowledged the receipt of the address and the communication between the master and the slave can continue. Otherwise, there is an error and the FSM jumps to the error-handling state x"EE". The rest of the data transfer between the master and the slave pretty much follows the same pattern you see in Listing 4.

IMPLEMENTATION

If you have the complete I²C master controller design written in VHDL, you can implement it to actually see it working. To do so, you first must get a hardware description language (HDL) compiler to synthesize the VHDL source code into a netlist.

A free edition of Quartus II is available on Altera's web site. After installing the program, you can start a new project and add our master controller VHDL code to the project. Quartus compiles the VHDL source code to a netlist that can be downloaded onto the FPGA.

For my demonstration, I used the Altera DE2 development board with a Cyclone II FPGA. The board did not have the DS3232 RTC chip, so I wire-wrapped the chip to the general PIO pins on the board (see Photo 1a). There are six wires connecting the RTC chip to the GPIO pins, but only four of them are used for the I²C bus communication: SDL, SCL, VCC, and GND. In addition to the RTC chip connected to the GPIO, there is also the 5.6-kΩ pull-up resistor for the SDA line.

After you successfully compile the master controller VHDL code, you can use Quartus to download the resulting netlist onto the FPGA on the DE2 board. Photo 1b shows the full DE2 board with the I²C master controller implemented inside the Cyclone FPGA and communicating with the RTC. The seven-segment display shows (from left to right) the FSM state d5, register 02 of the slave, and the data 16 being sent from the slave's register 02 to the master. The master controller also allows you to set the address and data switches at the bottom of the board to read or write from and to specific register locations in the RTC chip.

A final note regarding the implementation of this design. You can use a different FPGA prototype board or synthesizer software. The VHDL source code should be general enough so you can use another compiler and compile it for another FPGA chip. If you do, you will need to change the hardware-dependent pin mappings. Furthermore, you should also be able to use another chip instead of the RTC chip that I used as the I²C slave.

In any case, the I²C master controller design I presented should be easy enough for you to understand and modify as necessary. You can also use my FSM as a template for designing other controllers.

Enoch Hwang (ehwang@lasierra.edu) has a Ph.D. in computer science. He is currently a professor of computer science at La Sierra University, Riverside, California. Enoch is interested in embedded microprocessor systems, automation, and robotics.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/228.

RESOURCES

J. Bhasker, *A VHDL Primer*, Prentice Hall Professional Technical Reference, September 1998.

E. Hwang, *Digital Logic and Microprocessor Design with VHDL*, Thomson, 2006, www.lasierra.edu/~ehwang.

Maxim Integrated Products, "DS3232: Extremely Accurate I2C RTC with Integrated Crystal and SRAM," Rev 4, 2008, http://datasheets.maxim-ic.com/en/ds/DS3232.pdf.

Philips Semiconductors (NXP), "The I²C-Bus Specification," Ver. 2.1, 9398 393 40011, 2000, www.nxp.com/acrobat_download/literature/9398/39340011.pdf.

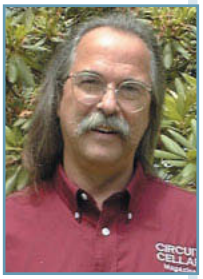
SOURCES

Cyclone II FPGA, DE2 development board, and Quartus II development software
Altera | www.altera.com

DS3232 RTC Chip
Maxim Integrated Produces | www.maxim-ic.com

2009 CIRCUIT CELLAR READER SURVEY & SAMPLE POOL SPONSORS

by Jeff Bachiochi



Embedded Speak

A Text Library for Allophone Translation

Do you need to add text-to-speech capability to a project? If your design requires more than a few canned phrases (strings of allophone codes), you must build up the system's vocabulary. Jeff explains how to develop a text library for real-time translation.

In my May 2005 column, I presented the Magnevation SpeakJet sound and speech generator ("Speech Synthesis with SpeakJet," *Circuit Cellar* 178). If you recall, you can control the complex generator via a serial port. It uses mathematical sound architecture (MSA) to generate complex sounds and speech synthesis. Although you have complete control over its parameters, there are preconfigured DTMF tones, 43 sound effects, and 72 speech elements. Any of these can be invoked by simply sending a single byte from a computer or microcontroller.

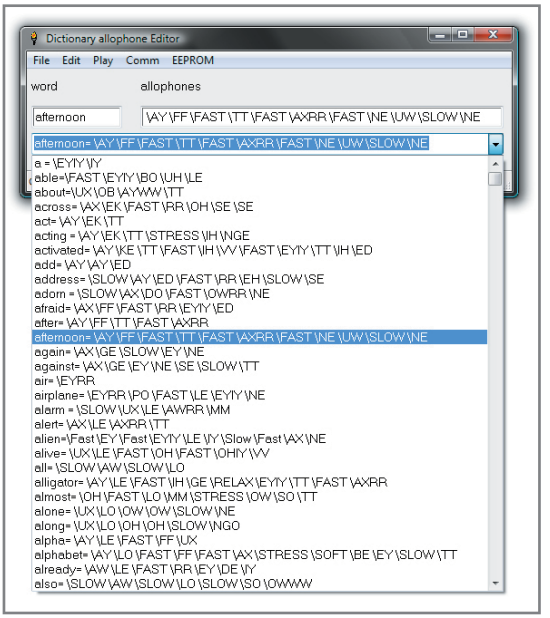


Photo 1—This is the Library Maintenance application written in Liberty Basic. Although you can use this application to add, edit, and remove library entries, its ultimate use is to create an Intel HEX file to program a serial EEPROM for use with this month's embedded text-to-allophone value application.

The Phrase-A-Lator is a companion program available from Magnevation. With this PC application, you can choose a series of allophones to send to the SpeakJet to produce a recognizable word, or pure gibberish, depending on your intentions. This process—replacing the letters of a word with allophones, or a textual representation of specific sounds, and substituting these allophones with byte codes that the SpeakJet will translate into sounds—can be defined as "text to speech." Here, it takes two to tango, so to speak: translation and pronunciation. SpeakJet provides the pronunciation, and you are left to provide the translation.

TRANSLATION

When your project requires the production of a few voice commands or responses, a few canned phrases (strings of allophone codes) are all you need to make use of the SpeakJet's inexpensive speech capability. However, as a project's vocabulary grows, so does the need for flexibility. Real-time translation may be required. As often is the case, there are a number of different ways to get the job done. In this case, you could use a dictionary look-up or rule-based translation. Each of these has its benefits and its drawbacks.

Dictionary look-up is as straightforward as its name implies. Each word to be translated must have an entry in the look-up table. Any word that doesn't have a matching entry in the dictionary will be tossed out (or you could choose to spell the word, say "what," or something else). This can make for some very large tables, especially considering plurals or other forms of the same word. However, each table entry has a predefined

allophone string, so every word can be constructed for accurate pronunciation.

Rule-based translation can be table-oriented as well, but the rules, rather than words, are searched. In English, most rules have exceptions, which actually makes rule-based translation a combination of rules and exceptions.

For this project, I base my translation on a dictionary look-up, which means there must be a way of maintaining the dictionary as well. So, let's start with the dictionary presented by Magnevation for use with their Phrase-A-Lator application. This dictionary has about 1,500 entries and can be easily expanded (with the total size limited to the amount of storage space you have available).

The dictionary is a simple text file consisting of a line of text for each word, and its list of allophones, split by a separation character (the "=" character in this case). Although you can certainly use the Phrase-A-Lator program to maintain the dictionary, the file, in its present text state, requires much more storage space than is necessary. Each allophone can take up to five characters to store as opposed to the single byte value that each allophone represents. Therefore, a good deal of space can be saved by creating a dictionary that holds only those byte values necessary to "speak" the allophones. Plus, the dictionary must be able to be stored in memory, so you'll need to create a library maintenance program anyway.

LIBRARY MAINTENANCE

Library maintenance is a way to add, delete, or alter library entries. In addition, this library must be saved as an Intel HEX file so it can be loaded into a memory device and searched by an embedded microcontroller. This library maintenance program is written in Liberty Basic to handle these functions (see Photo 1). The application will read in a dictionary file (like PhrasALator.DIC) and separate each entry into two arrays of words and allophone strings. You can choose an existing entry from the dictionary and play or edit it, add a new one, or delete an unwanted one. Maybe the key word here is "play." If you connect up the SpeakJet to the PC, you can hear what your translation sounds like (as you would with the Phrase-A-Lator program). When a change is made permanent (by updating the word), the entire file is resorted to assure it remains in alphabetical order. (This is important to the search operation. More on this later.)

Once the dictionary is completed, it should be exported as an Intel HEX file. An Intel HEX file is an ASCII text file with lines of text that follow the Intel HEX file format. Each line in an Intel HEX file contains one HEX record. A record begins with a colon followed by a number of hexadecimal values that include the record length, address, record type, data, and a checksum. The data consists of the actual bytes you are saving. The other record fields are informational. Intel HEX files are often used to transfer a program or data that would be stored in a ROM or EPROM. Most EPROM programmers or emulators will import Intel HEX files.

The data within the HEX file is nothing more than each dictionary entry with a data word preceding it, which is used as a pointer to the next word. This helps speed the dictionary search process by pre-locating the next entry. The first 26 entries are special entry pointers to locate the first word in the dictionary that starts with a particular letter allowing all prior entries to be skipped. As you can see from the letter "z" pointer in Table 1 (the twenty-sixth entry at address 0x0032), the first 25 letters use addresses up to 0x4C94 (19604 decimal). At about 1,500 entries, this is roughly 14 bytes per dictionary entry. All this may not make any sense

Address	Word Pointer Data	Data	entry=allophones
0000	0034		
0002	0416		
0004	060F		
0006	0BBA		
0008	0F12		
000A	121B		
000C	193F		
000E	1BC5		
0010	1EAC		
0012	210E		
0014	21CE		
0016	226B		
0018	2567		
001A	2862		
001C	2A39		
001E	2BA0		
0020	3106		
0022	31AB		
0024	342D		
0026	3F4F		
0028	46AB		
002A	4778		
002C	485E		
002E	4BC9		
0030	4C07		
0032	4C95		
0034	003A	613D9A80	a=<154><128>
003A	46	61626C653D079AAB8A91	able=<7><154><171><138><145>
0046	52	61626F75743D86ADA3BF	about=<134><173><163><191>
...	...	...	...
00416	041D	623DAA8080	b=<170><128><128>

Table 1—The data is stored linearly beginning at address 0x0000. The pointer for the "a" dictionary entries is located at address 0x0000. Notice that it points to address 0x0034, where the pointer to the next "a" dictionary entries is found at 0x003A. Following the next pointer are 4 bytes of data. (The data is shown in ASCII in the last column.) The pointer for the "b" dictionary entries is located at address 0x0002.

ICbank
Gateway to Electronics

MP3P DIY KIT, Do it yourself

(Include Firmware Full source Code, Schematic)

• myPIC

Only
$160 qty 100
$220 qty 1

Powerful feature

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

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

• myWave (MP3 DIY KIT SD card Interface)

Only
$150 qty 100
$200 qty 1

Powerful feature

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

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

• myAudio (MP3 DIY KIT IDE)

Only
$180 qty 100
$220 qty 1

Powerful feature

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

ICbank
Gateway to Electronics

www.icbank.com

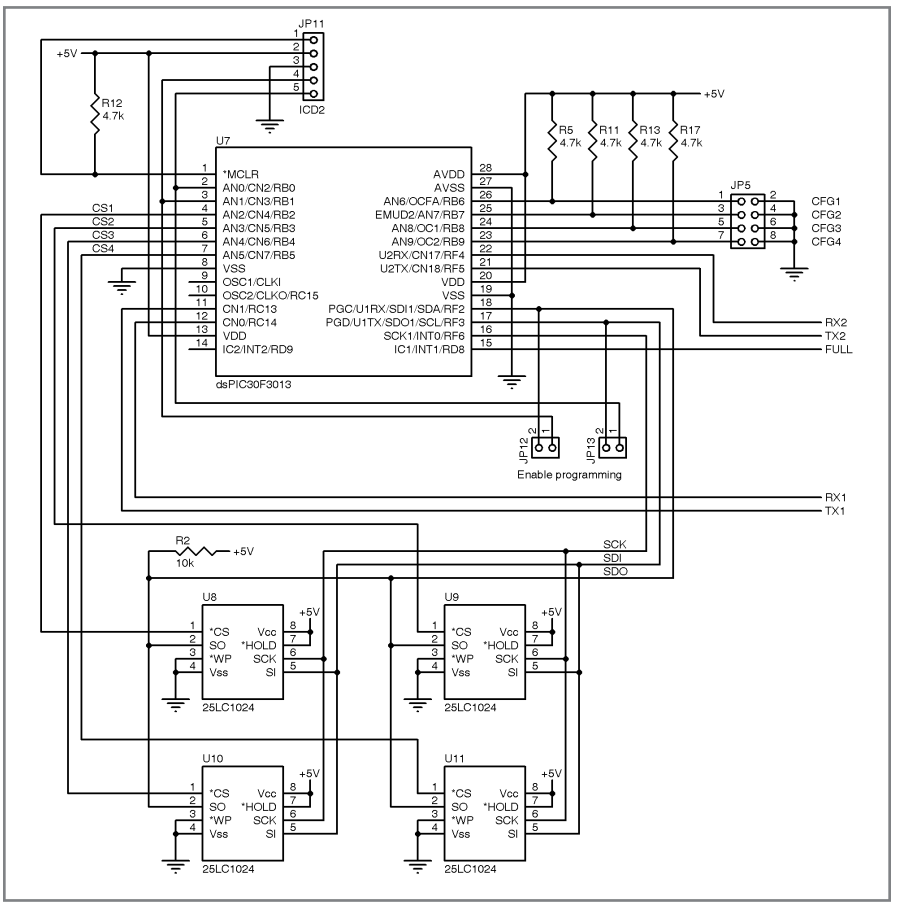


Figure 1—This circuit fragment shows the parts necessary to add text-to-allophone conversion as input to the SpeakJet. Refer to my May 2005 *Circuit Cellar* article for the original circuitry (or visit www.magnevation.com).

unless you understand this project's circuit.

LIBRARIAN

I used a Microchip Technology dsPIC30F3013 for this project because it has two serial ports: one as a text input port and the other as an allophone (value) output port. The text comes from the PC or the embedded project. The allophone values go the SpeakJet. No data actually flows back from the SpeakJet (except for a buffer full signal), so this project could be accomplished with a single port (with the TX and RX connecting to different peripherals). However, I wanted multiple ports to aid in debugging.

An extra PCB from a previous project (that supports the dsPIC) provided me with a platform to start with. The dsPIC30F3013 has 1 KB of internal EEPROM, so I could write some code using the internal EEPROM prior to building up a complete circuit with an external EEPROM. There is enough room internally to hold a small dictionary (approximately 64 entries), if well chosen. You can see that the previously discussed Phrase-A-Lator.DIC file was going to need more room than this. Figure 1 shows the project's circuitry.

Although the internal EEPROM data is accessed differently than data from an external EEPROM, it seemed like a good idea to try out a few ideas quickly. It was a sanity check. After adding letters and numbers to the internal dictionary, I chose a few other words to fill up the available space. This provided a sufficient dictionary with which to test my theories. As it turned out, this was extremely helpful because it enabled me to create the project application before prototyping the complete circuit. But once the initial circuit was operating and providing output that could connect to the SpeakJet, I longed for more. So it was on to a bigger dictionary.

For the external EEPROM, I chose a SPI serial EEPROM because it can be clocked up to 20 MHz. A 1-Mb 25LC1024 can store more than 10 times the present library. Note that the Intel HEX file will have to be modified to add an Extended Linear Address Record when the Library size exceeds 0xFFFF (16 bits). The EEPROM requires an 8-bit read command followed by a 24-bit address to get pointed to the required data. Data then can be received 1 byte at a time. The EEPROM's internal pointer is automatically incremented so that additional sequential bytes can be read without separate requests. With the microcontroller's internal RC oscillator in use (7.37 MHz), a single byte grab of data from the external serial EEPROM takes approximately 30 µs. A real search through all of the 200 "s" entries in the dictionary takes less than 40 ms.

SERIAL I/O

Both UARTs in the dsPIC are set to run at 9,600 bps, which is the default rate for SpeakJet. Each UART has its own TX and RX ring buffers to allow for the continuous reception and transmission of data. After my one-track mind got over the literal meaning of "text to speech," I began to see an advantage associated with characterizing the input. To do this, I created a routine that identifies an input character as belonging to one of five groups: alpha, numeric, control, punctuation, and symbols (see Table 2). This way, actions can be based on groups rather than individual characters.

The input ring buffer is monitored for a punctuation character. When one is received, all characters including this one are transferred to a working buffer. Most of the time, this working buffer will contain a word consisting of alpha characters followed by a space, but it could be another form of punctuation that will be interpreted as the end of a word.

The first letter in the working buffer is down-converted to a number between 0 (a)

Group	Characters
Alpha	a–z and A–Z
Numeric	0–9
Control	0x00-0x1F
Punctuation	<sp>!"#'+,.-:;?
Other	#$%&()*/<>=@[\]^_`{|}~

Table 2—I characterize a character by group to help determine how to handle it.

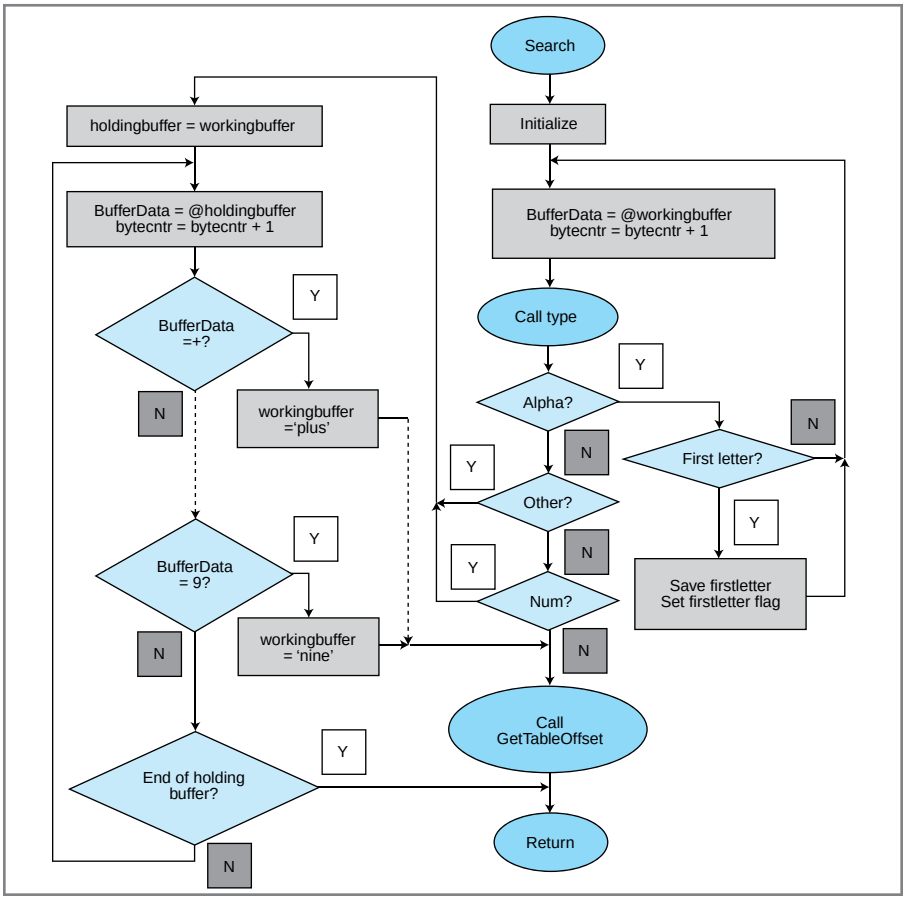


Figure 2—This is where the EEPROM actually gets searched for a match with the text in the working buffer. The entries are sorted, so you can give up, once past the appropriate entry. Otherwise, transmit the associated allophones to the SpeakJet.

and 25 (z). By doubling this number, you come up with an address within the external EEPROM that points to the address in the dictionary where all of the entries for that letter are held. The actual search is nothing more than a big loop sequentially looking through each dictionary entry for word (letter) matches between the buffer and the present dictionary entry (see Figure 2). As long as each letter in the present entry matches the buffer, you continue checking the next letters of dictionary entry. Let's assume all of the letters match and the words are the same length. Then you find a match and continue by transferring the remaining characters in this entry to the output ring buffer, as they are allophone values necessary for this word. These are automatically transmitted to the SpeakJet and pronounced as the matched word or at least what is defined by the dictionary entry. A short silence value is sent after each word to prevent slurred speech.

What happens when the word in the working buffer doesn't match the present entry in the dictionary? If the tested letter is greater than the entry's letter, you can give up on this entry and try the next word entry. If it is less than the entry's letter, you can give up and return with no word (because the entries are sorted). Even when a match exists, they must have identical word lengths or this is not really a match. Again, you can give up on this entry and try the next word entry.

For the most part, any character that is a number or symbol is treated in the exception routine. In this routine, the working buffer is transferred to a holding buffer (see Figure 3). Now each character in the holding buffer can be dealt with separately. For example, if an entry in the holding buffer is the character "9," the text "9" is placed in the working buffer and EEPROM is searched and the allophones are transferred if the dictionary entry is found. The complete holding register is converted one character at a time to handle every exception. All the exception text must be in the dictionary or nothing will be transmitted (word not found).

At this point, I have not added any routines to pronounce numbers in any format other than single digits. It might be nice to add the ability to interpret numbers in groups using hundreds, thousands, millions, dollars/cents, and so on.

dsPIC CIRCUIT FRAGMENT

In my May 2005 article, I explained how to connect the SpeakJet to a Future Technology Devices International FT232 (USB/Serial) device and use it with a PC. The PC was responsible for sending allophone data values to the project's circuitry to get the SpeakJet to, eh, speak. This month's project is designed to go between the FT232 (or some other form of communication interface like RS-232 or your microcontroller) and the SpeakJet. Its input is ASCII text of some kind. The dsPIC30F2013 translates this text into allophones as it matches the text to a dictionary stored in EEPROM. Allophone values are then transmitted to the SpeakJet, thereby relieving the PC or your microcontroller from having to handle this task.

Besides the dsPIC30F2013's two UARTs, this device also supports SPI/I²C serial communications. The specifications for the SPI devices (20-MHz clock) are faster than the I²C device's (maximum 1-MHz clock). And since communications speed is important to search times, the SPI mode is used. The previous discussion showed that the present dictionary required less than 10% of a 25LC1024 EEPROM, but I chose to include the provisions for three additional devices. There is address space for seven additional devices in the 25LC1024's 24-bit address structure without having to increase to a 32-bit address.

I have other ways of programming an EEPROM (other than in-circuit), so you'll notice all the EEPROMs are write protected (via grounding the *WP input on pin 3). You need to tie these high if you want to implement an Intel HEX input application that will program the EEPROM via the input's serial connection to the dsPIC30F2013. You can use one of the unused configuration jumpers to access this function if you want to add that routine to your code.

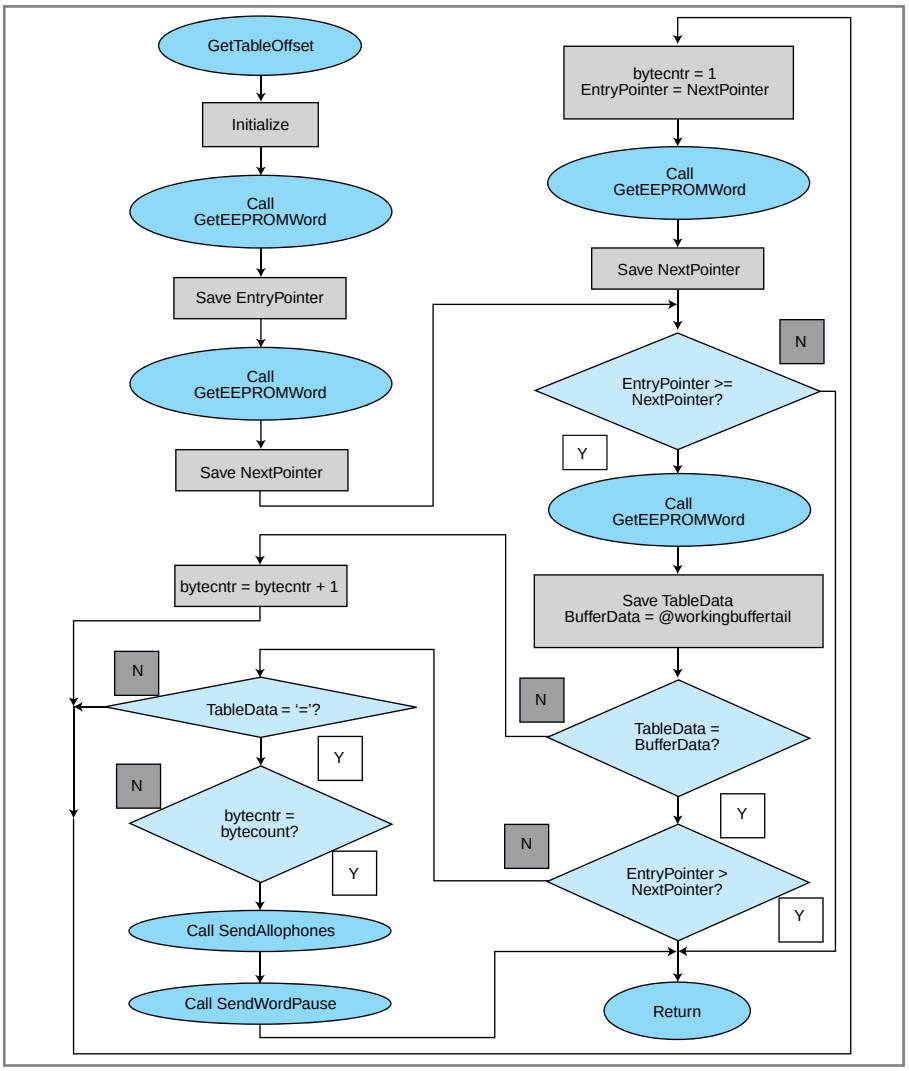


Figure 3—Before searching the EEPROM table, the working buffer may be holding a word or other character. A holding buffer is used if it has been determined that there are numeric or other characters that need translation into text before calling the actual hunt routine `GetTableOffset`.

One note about using dsPIC devices with an in-circuit debugger: Chip peripherals tie up all I/O pins (especially on low-pin-count versions of the device), and the debugger requires a pair of pins to do its job, so the connections for the in-circuit debugger can be redirected to a number of alternate pins to help you avoid interference to these peripherals. Normally, this function is shared with the programming pins, which can't be redirected, for obvious reasons. JP12 and JP13 allow these programming pins to be enabled (connected to the ICD) if you are programming or disabled (disconnected from the ICD) if you are debugging.

TEXT TO SPEECH?

It might be incorrect to label this as a "text-to-speech" project. Speech is much more than just uttering words in a particular sequence. I've treated words here as if they are independent of each other. In reality, they are not. You might say the same word differently depending on its context in a sentence. These words are called homographs. Examples are words like bass, bow, and read—each has an entirely different pronunciation and meaning. Syllable emphasis can be built into look-up dictionaries. But other stresses, hinted at by punctuation, are outside the function of a dictionary.

Prosodics involve variation in syllable length, loudness, pitch, and the formant frequencies of speech sounds. Sentence structure algorithms can be used to determine how these variations might be used to alter the written words based on punctuation. SpeakJet has parameters that can be used to alter an allophone's rate, volume, and pitch. You'll find the stress and relax commands used extensively within the project's dictionary add emphasis within individual words.

Although I'd like the time to investigate rule-based algorithms, it is much more than I can cover in a single column. The best way for me to gauge interest in a subject is to gather feedback from you. Drop me an e-mail and tell me what you want more of (or less of, for that matter).

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/2009/228.

SOURCES

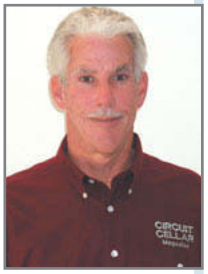
FT232 USB-to-serial UART interface
Future Technology Devices International | www.ftdichip.com

Phrase-A-Lator software and SpeakJet Sound Synthesizer
Magnevation | www.magnevation.com

dsPIC30F3013 DSC and 25LC1024 EEPROM
Microchip Technology, Inc. | www.microchip.com

Liberty Basic
Shoptalk Systems | www.libertybasic.com

by Tom Cantrell



LiOn King

A Look at "Battery-in-a-Chip" Technology

Got Energy? Energy harvesting is all the rage, but like a real harvest you need a place to store the crop. This month, Tom introduces the next advance in thin-film rechargeable lithium battery technology: a battery-in-a-chip.

With the price of oil all over the map, a trip to the local gas station feels like a trip to Vegas. The gas pump is a slot machine: you put your money in, watch the dials spin, and hope today's your lucky day.

It's safe to say that "Green" is the new black and embedded designers must do their part in this new era of energy consciousness. Beyond the headline grabbers, innovations like hybrid vehicles and efficient light bulbs are a myriad of mainstream apps that should go on a digital diet.

Hats off to the silicon wizards for delivering chips that not only do more for less, but also consume less energy doing it. However, these silicon advances put the ball back in the designer's court. It's up to you to figure out innovative applications and clever design techniques that make the most of the energy-saving opportunities.

ENERGY IN A CHIP

Battery technology will play a pivotal role in the Green revolution. For instance, if the question is about the widespread move to

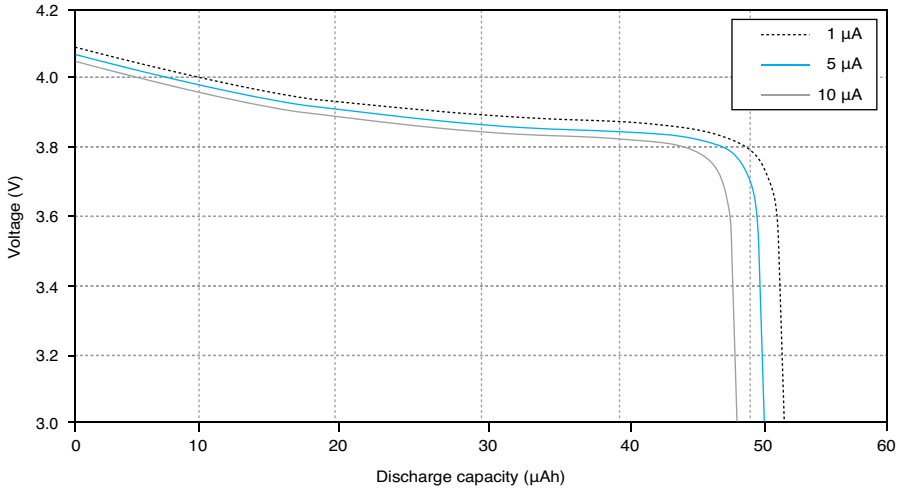


Figure 1—One advantage for lithium batteries, including EnerChips, is a flat discharge curve. Just watch out as you approach the "cliff" since deep discharge isn't good for the battery.

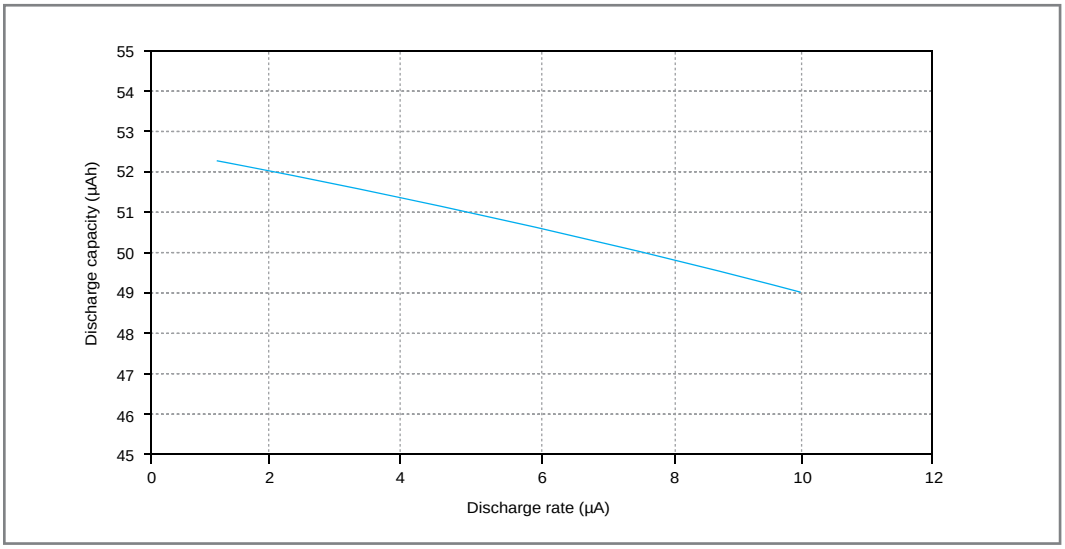


Figure 2—Although capable of relatively high discharge (e.g., 300 µA for 20 ms), note that effective capacity decreases with increasing loads.

electric vehicles, the answer is battery technology that can deliver the range and performance fossil-fuelers are used to.

At the other extreme, Cymbet is taking a "small is beautiful" tack with their "EnerChip" thin-film rechargeable lithium battery technology. In short, as the name implies, it's a battery-in-a-chip. Combined with the latest in nanopower silicon, the EnerChip offers an intriguing option for designers to consider.

Just keep in mind we're not talking about a lot of energy here. So far, the Cymbet batteries top out at 50-µAh capacity (CBC050) with a lesser 12-µAh model (CBC012) also offered, although this is on the order of a thousand times less than the familiar lithium "coin cell." At least the output is a healthy 3.8 V. That's suitable for use in typical (e.g., 3.3-V) designs. Of course, as with any battery technology, you can always lash them together to increase voltage, current, and capacity.

It's a fact of life for all batteries that specs like "50 µAH" and "3.8 V" are overly simplistic and tend to obscure the fact that a myriad of other application factors come into play. The Cymbet batteries are no exception. Let's take a closer look to get a better understanding of how they can serve existing designs or, better yet, enable exciting new ones.

Although on a tinier scale, the Cymbet batteries exhibit the same desirable performance characteristics that have made their larger lithium coin cell cousins so popular. For instance, the voltage discharge curve is as flat as a board, which guarantees virtually full output until the bitter end (see Figure 1). If there's a downside, this means you can't really expect to use the voltage output level as a foolproof indicator of remaining battery life. Power management will have to be smarter than that, which is all the more reason to better understand the specs.

With a throwaway coin cell, using it until it's dead (and then replaced) is standard procedure. By contrast, with the Cymbet rechargeable, you need to be careful not to run it off the cliff shown in Figure 1. There's no simpler way to say it than the CBC050 datasheet does: "Failure to cutoff the discharge voltage at 3.0 V will result in battery performance degradation."

Another benefit of lithium cells is the ability to deliver surprisingly high surge currents—in the case of the CBC050, up to 300 µA. But

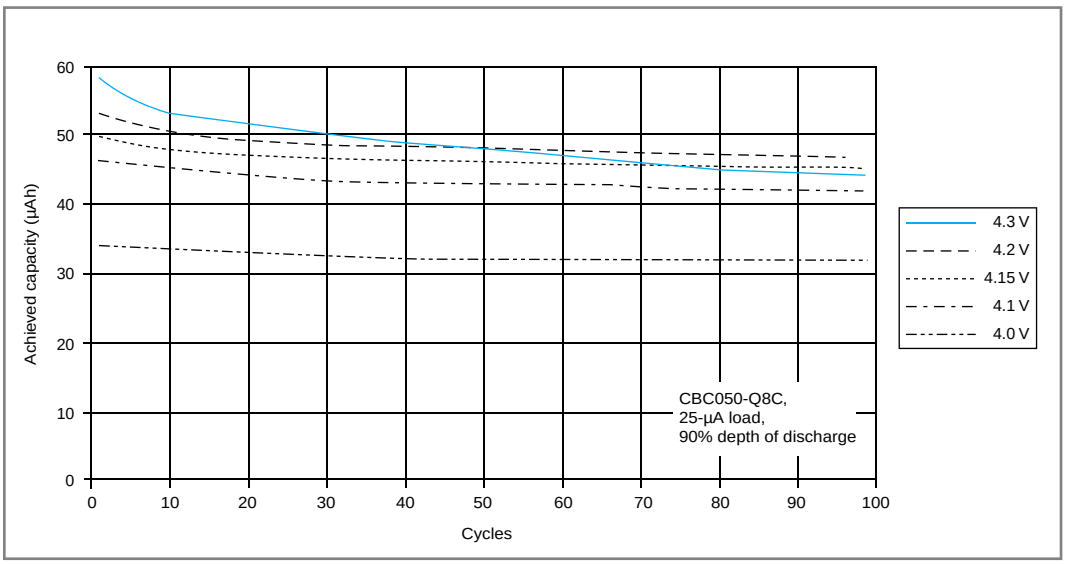


Figure 3—To recharge an EnerChip, you'll need a power supply that delivers exactly 4.1 V—no more, no less. The proper charge voltage both maximizes charge capacity while minimizing "cycle fade."

watch out, because the capacity declines almost linearly with the load as shown in Figure 2. If you think a "50-µAh" lithium cell should be able to run a 300-µA load for 10 minutes (i.e., 50/300, or 1/6 of an hour), you've got quite a surprise coming. Extrapolate the line in Figure 1 to the right and you'll see what I mean. Using a car analogy, drive with a heavy foot on the throttle and your mileage will suffer, so a full tank of gas won't take you as far as if you drive more sedately.

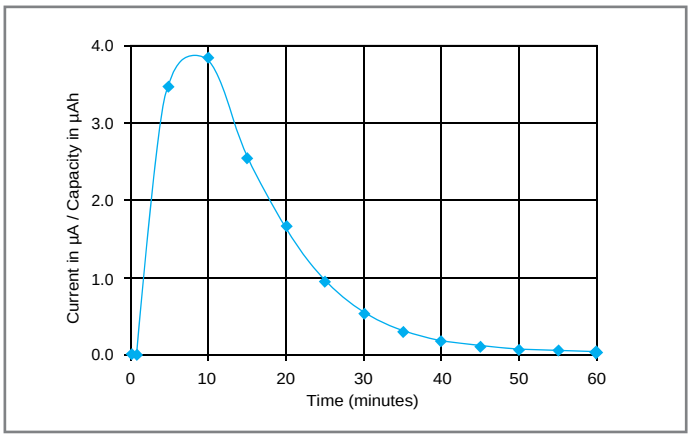


Figure 4—While the charge voltage should be precise (e.g., 4.1 V), not much current is required, just 200-µA peak for the 50-µAh CBC050.

In the old days, NiCad batteries were the all the rage for rechargeable applications. Do you remember the infamous "memory effect"? That referred to the propensity for NiCad batteries to "remember" repeated partial discharge levels and get stuck at a less-than-rated capacity. To counter, users in the know would be careful to "deep discharge" their NiCads to wipe the "memory" clean.

The Cymbet rechargeable lithium technology is somewhat the opposite. It's more like a car (i.e., lead acid) battery in that deep discharge is something you want to avoid because it reduces the number of potential recharge cycles. Ponder the specs and you'll see the effect is by no means trivial. For instance, keep the CBC050 "topped off" by limiting discharge to 10% and you can expect to get a full 5,000 discharge/recharge cycles out of it. But if you routinely run it down to half full (i.e., 50% discharge), that spec drops by a factor of five to 1,000 cycles.

Temperature also plays a role. The aforementioned specs are for 25°C operation. Boost the temperature to 40° and cycle counts are cut in half (i.e., 2,500 and 500 cycles at 10% and 50% discharge, respectively). Also take note of the operating temperature range of –20° to 70°C, a possibly limiting spec in "harsh-environment" applications.

Self-discharge can be a problem for lesser battery technologies. For instance, I have an older digital camera with rechargeable NiMH batteries that I might use once a month to take a picture for this column. Despite being nearly fully charged when I last used it, it is invariably drained when I fire it up a month later.

By contrast, the CBC050 is quite happy to sit idly by for many months or even years. The self-discharge spec comprises two parts. The first is "recoverable" self-discharge. Yes, the capacity will decline over time, but the next recharge will make everything right again with no permanent damage. There's also a "non-recoverable" self-discharge, or "aging," that's more serious in that it represents a permanent loss of capacity.

That all sounds scary until you look at the CBC050's specs. The "recoverable" self-discharge rate is 8% and the "non-recoverable" rate is 2.5%. But that's per year! At a total of 10.5% per year, that means the CBC050 could come to life after almost 10 years in storage, and it would still be quite serviceable (i.e., able to charge back up to 75% of the original rated capacity).

Put all the specs together and you start to get a realistic picture of what a CBC050 can deliver in a particular application. Consider these different application scenarios.

The first has the CBC050 fulfilling the typical role of "battery-backup" for a low-power CMOS chip (i.e., MCU, SRAM, RTC). It's quite well-suited to the task, but faces notable competition from an unlikely source: not another battery, but the so-called "SuperCap" high-value capacitors. But put on your "Green" eyeshade to look closely at a SuperCap datasheet and notice the very high self-discharge spec. In essence, SuperCaps need to be kept on the charger at all times in order to be ready for a call to action. It also means there's energy wasted keeping them topped-up. The difference may not seem like a big deal, but from a holistic "energy consciousness" point of view, the potential self-discharge energy advantage is significant. According to Cymbet, up to one-third the energy spent charging a SuperCap (e.g., 0.2 F) is lost to self-discharge versus a tiny fraction of a percent for an EnerChip.[1]

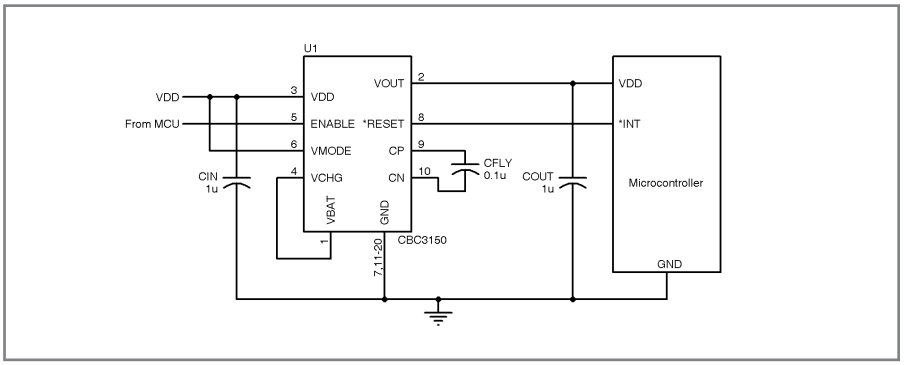


Figure 5—The EnerChip "Charge Controller" parts combine an EnerChip battery and charge control electronics in a single chip.

Safety is something easy to overlook, at least until it jumps up and bites you. Everyone has seen the headlines about exploding laptops and such. Fortunately, Cymbet says that even a dead-short won't lead to unwanted pyrotechnics.[2]

> Cymbet takes their EnerChip technology a major step further with the so-called 'CC' upgrade that combines the battery with power-management logic and packs a complete mini-me Uninterruptible Power Source (UPS) in a chip ...

CHARGE IT

Recharging an EnerChip battery is simple enough. The easiest approach is simply to connect it to a 4.1-V supply and the battery will fully recharge in less than an hour. Alternatively, a two-phase scheme can be used that starts with a constant current (e.g., 50 µA) phase and finishes with a constant voltage (4.1 V) phase. The documentation notes the two-phase approach may be required for future EnerChips; but for these initial batteries, the single-phase constant voltage approach is fast and easy.[3]

The main consideration is that the 4.1-V supply needs to be rather precise—within a few percentage points (see Figure 3). If the voltage is too high (e.g., 4.3 V), the battery will exhibit "cycle fade," in which the capacity declines with each recharge cycle. Too low (e.g., 4.0 V) and the battery won't recharge to full capacity.

The EnerChip internal impedance is high enough that there's no need for extra components to limit the charge current, at least as long as the charging voltage isn't too high (i.e., it should not be greater than 4.3 V). As shown in the typical battery-charging profile (see Figure 4), the charge current peaks at about four times the battery capacity (i.e., 48 µA for the CBC012 and 200 µA for the CBC050).

UPS-LITE

Cymbet takes their EnerChip technology a major step further with the so-called "CC" upgrade that combines the battery with power-management logic and packs a complete mini-me Uninterruptable Power Source (UPS) in a chip that's just slightly larger than the battery alone.

EnerChip CCs are available using either the 50-µAh (CBC3150) or 12-µAh (CBC3112) batteries. They're packaged in a 9 mm × 9 mm 20-pin package, but as shown in Figure 5, there are really only a few pins to deal with.

With a wide 2.5- to 5-V input range, VDD is the primary power source for the EnerChip CC, just as wall power is the primary source for a 120-VAC UPS. VBAT is directly connected to the battery. VOUT is the uninterruptable 3.3-V (typical) output to the load.

An internal charge pump (with an external capacitor connected to the CP and CN pins) generates a precise 4.1 V on the VCHG pin to charge the battery. It's simply a matter of connecting the VCHG and VBAT pins to close the loop. If you're wondering why Cymbet didn't just connect them for you internally, the answer is that a single EnerChip CC can work with up to 10 external EnerChip batteries for applications that need higher current and capacity than the on-chip battery provides. While it doesn't hurt the battery to charge it all the time, needless charging consumes power to keep the charge pump running, so the ENABLE (EN) pin provides an external On/Off switch.

The EnerChip CC automatically handles the switchover between primary (VDD pin) and back-up (VBAT) power delivery, the threshold being set with the VMODE pin. If VMODE is connected to VDD, the threshold is 4.5 V (i.e., suitable for 5-V designs). If VMODE is grounded, the threshold is 3.0 V (i.e., for 3.3-V designs). A third option requires a pair of external resistors, the ratio between them setting the threshold voltage anywhere between 2.5 and 5 V. The RESET* pin is driven low when the EnerChip CC is providing power

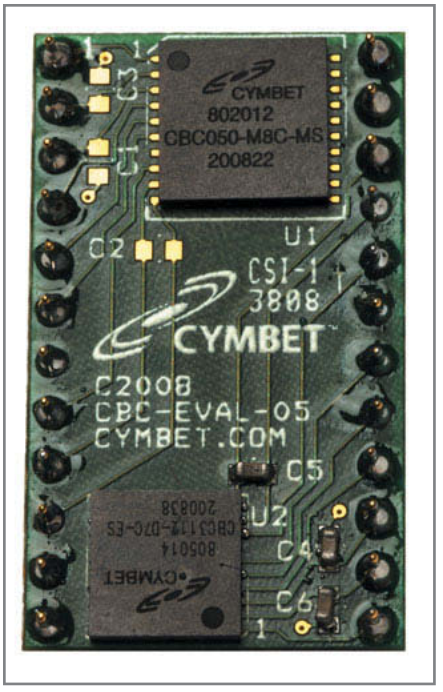


Photo 1—The CBC-EVAL-05 includes both 50-µAh (CBC3150) and 12-µAh (CBC3112) EnerChips. It's easy to switch between one or the other just by rotating the module.

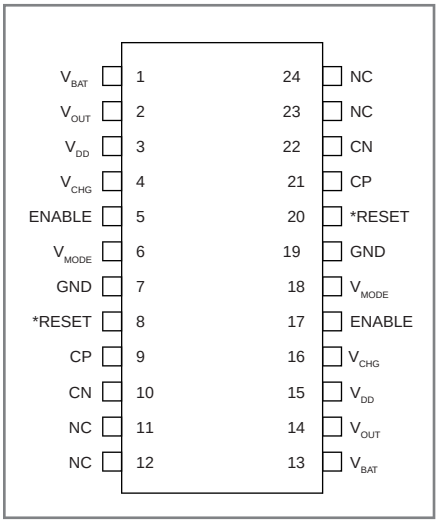


Figure 6—The CBC-EVAL-05 module makes it easy to experiment with Cymbet EnerChip technology.

Photo 2—This design shows how EnerChips and coin cell batteries can work together.[4] The EnerChip keeps the system alive when the coin cell needs to be replaced.

to VOUT from the internal battery.

Battery-protection is also built-in, with the output (VOUT) automatically disconnected from the battery when VBAT falls too low. The combination of precise charge voltage and battery protection maximizes the capacity and number of recharge cycles the battery delivers.

Cymbet offers some handy evaluation modules that make it easy to experiment and prototype with EnerChip batteries and CC controllers. Consider the CBC-EVAL-05 (see Photo 1), which includes both a CBC3112 and a CBC3150 packaged in a 24-pin DIP format. Notice on the pinout (see Figure 6) how the left and right sides of the DIP are mirror images. One side connects to the CBC3112 and the other the CBC3150 so you can test either by simply rotating the module.

The module has the flexibility to support a variety of experiments. For instance, if you're mainly interested in playing with the batteries, but not Cymbet's charge controller, just leave the VCHG pin disconnected from the VBAT pin and have at it.

There's also a hybrid option that has one CC, pardon the pun, in charge of the other's battery (i.e., CBC3150 charge controlling the CBC3112 battery and vice versa). A variation runs both CBCs simultaneously by connecting their VBAT pins with either VCHG pin (not both).

DUST STORM

When comparing Cymbet's EnerChips to existing solutions such as coin cells and SuperCaps, it's all too easy to fall into the "us vs. them" trap. There are no doubt applications where an EnerChip is the clear-cut winner and should replace the earlier devices. But "us vs. them" overlooks the fact that there are a lot of situations where "us and them" can work well together. Let's take a look at some EnerChip-based gadgets and you'll see what I mean.

Photo 2 shows what might seem an unlikely pairing of a coin cell battery and an EnerChip, but it's actually a combination that makes a lot of sense. The application would generally draw from the coin cell, calling on the EnerChip to "bridge" the power-gap when it's time to replace the coin-cell. The EnerChip would allow "in-flight refueling" (i.e., application continues to run) and preserve critical data across battery swaps. For instance, keeping a real-time clock alive with an EnerChip would put an end to the embedded equivalent of the flashing "12:12:12" problem (i.e., devices that lose their minds and need to be re-initialized when you change the battery).

Look no further than Cymbet's "Solar Energy Harvesting" demo kit (CBC-EVAL-08) to see how EnerChips and capacitors can be best buddies too. The kit utilizes a three-tier hierarchy of power generation starting with a solar panel that picks up what energy it can, when it can, from ambient light. The solar panel output feeds a boost converter that steps up the voltage to a useful level (3.5 V). When solar energy is sufficient, it drives the load and charges a pair of CBC050 batteries. If the light fades, the EnerChips take over supplying the load.

So far, so good. The only gotcha being said load had better be pretty small. `Whether powered by the solar panel in bright sun, or running off the EnerChips, we're talking about only tens to hundreds of

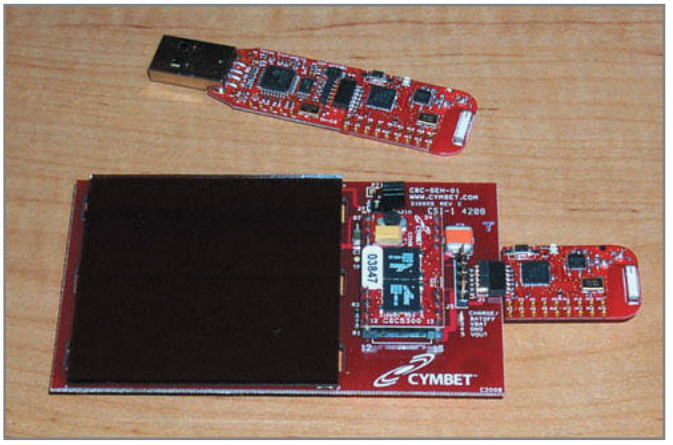


Photo 3—The Texas Instruments eZ430-RF2500 Solar Energy Harvesting kit puts Cymbet EnerChips to work with TI silicon in a "zero-power" wireless sensor application.

microamps on tap.

Here's where our little friend the capacitor comes in. Capacitors may be leaky, but they're also more than willing to give it all they've got in a big bang (i.e., high discharge current). That brings us to the third tier in the power-generation pyramid: a 1,000-µF capacitor. Although hardly a "SuperCap" (real ones are measured in Farads), it can nevertheless deliver a whopping 30-mA discharge for 20 ms, fully 50 times the 600-µA surge the pair of EnerChip batteries can provide. Of course, there's no free lunch. The battery resistance and the capacitor form an RC network that takes a few seconds to recharge.

Is that enough energy to do anything useful? Texas Instruments says so, and to prove it, they've come up with the eZ430-RF2500 Solar Energy Harvesting kit (see Photo 3) that uses the Cymbet Solar Energy Harvester to power a wireless sensor solution based on their MSP430 flash memory MCU and CC2500 802.15.4 radio chips.

The kit comes with the Solar Energy Harvester, a USB adapter that connects to your PC, and a plug-in MCU-plus-radio module for each. The software comprises a simple temperature-sensing application with TI's home-grown SimpliciTI network stack running on the nodes, and a PC-monitoring program that displays the network in action (see Photo 4).

Yes, the application is trivial, but the design implications aren't. This stuff really works! The solar panel was able to power the load and keep the EnerChip charged in moderate lighting conditions, even indoors. But when there wasn't enough light, the EnerChip seamlessly kicked in, able to keep the node on the air for up to 400 additional packets on battery power alone.

TIPS & TRICKS

By now most designers are familiar with the typical low-power design techniques (i.e., sleep mode, powering down unused logic, up/down-shifting the clock rate, and so on). But getting on the energy-harvesting bandwagon requires taking low-power design techniques even further. When you're talking millionths of an amp for a power budget, every little bit adds up.

For example, when was the last time anyone really thought much about all those lowly pull-up resistors littering most designs? Well, think again. Consider the typical 100-kΩ pull-ups inside most MCUs, not just on the I/O lines, but also on the control inputs such as interrupts and reset. The bad news is that a 100-kΩ pull-up at 3.3 V burns 33 µA just sitting there. We always new that, but just didn't care. Now we do.

So, for example, you don't want to leave the pull-ups on your software-scanned matrix keypad enabled all the time. Instead they should only be powered during the active scan. Indeed, where possible (i.e., external pull-ups), use a higher value resistor (e.g., 1 MΩ). But pay close attention to your rise and fall times since chips burn more power during the time an input transitions through the "floating" region between rails.

Similarly, be on the lookout for subtle leakage paths between chips. For instance, an RTC powered by a battery can leak power through its pins to an attached MCU, even if the MCU is powered off. Use diodes and transistors as hose clamps and valves to seal even the tiniest leaks.

The cyclic nature of the capacitor discharge power supply poses all manner of creative challenges for designers. You no longer have the luxury of consuming all the clock cycles you want whenever you want them. Instead your hardware and software design has to deal with the reality that the power supply drives the schedule. Imagine how this complicates an already tricky and timing-sensitive task like wireless networking. The

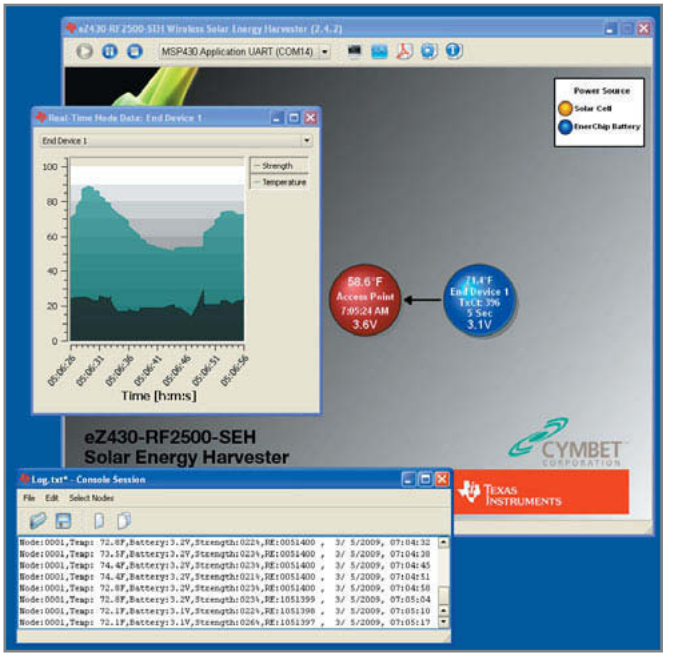


Photo 4—The PC software that comes with the TI kit shows the network in action. Note how the node counts down the number of packets it will be able to send on EnerChip power alone (i.e., in the dark).

Energy Source	Harvested Power
Vibration/Motion	
Human	4 μW/cm²
Industry	100 μW/cm²
Temperature Difference	
Human	25 μW/cm²
Industry	1-10 mW/cm²
Light	
Indoor	10 μW/cm²
Outdoor	10 mW/cm²
RF	
GSM	0.1 μW/cm²
Wi-Fi	0.001 μW/cm²

Table 1—The environment is filled with ambient energy free for the taking.[5] All you have to do is figure out ways to harvest it.

MCU may have to "stairstep" its way through complex procedures one short burst of energy at a time.

With all the starting and stopping, you even need to pay attention to the energy overhead of waking up and shutting down. After all, if your workday was only 10 minutes long, how fast you tie your shoes would suddenly matter a lot.

HARVEST TIME

When you think about it, the environment is filled with huge amounts of energy we can tap (see Table 1). Our old pal Sol(ar) gets most of the headlines, but there are plenty more sources free for the taking. Piezo transducers can capture energy from the vibration of a motor or the shock of a shoe hitting the pavement. Tomorrow's smart wardrobe might literally include "smart clothes" that run off power harvested from the heat of your skin. Or imagine, as Tesla did a century ago, being able to skim power from the RF chatter that bombards us.

We're only at the beginning of the green revolution, and already it's clear that energy harvesting is well beyond the (sunny) "blue sky" hype phase. The technology from Cymbet and TI is clearly viable for some real-world applications today and, with inexorable advances in technology, many more tomorrow. If you want to reap the benefits of energy harvesting, it's time to sow some new and clever designs.

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.

REFERENCES

[1] Cymbet Corp., "Benefits of Using Cymbet EnerChips Instead of Coin-Cells and Super Capacitors," AN-1008, www.cymbet.com/MContentA/pdfs/AN-1008-Replacing-Coin-Cells-and-Supercapacitors.pdf.

[2] Cymbet Corp., "Cymbet Rechargeable Battery and Energy Storage Device," www.cymbet.com/content/applications-faqs.asp.

[3] Cymbet Corp., "Guidelines for charging Cymbet EnerChip Batteries" AN-1002, www.cymbet.com/MContentA/pdfs/AN-1002-Charging-Cymbet-Batteries.pdf.

[4] Cymbet Corp., "Using the Atmel picoPower AVR Microcontroller with Cymbet EnerChips," AN-1014, www.cymbet.com/MContentA/pdfs/AN-1014-Atmel-picoPower-AVR-Demo.pdf.

[5] M. Raju, "Energy Harvesting," Texas Instruments, 2008, http://focus.ti.com/lit/wp/slyy018/slyy018.pdf.

SOURCES

EnerChip CBC-EVAL-05 Evaluation kit
Cymbet | www.cymbet.com

eZ430-RF2500-SEH Solar energy harvesting kit
Texas Instruments, Inc. | www.ti.com

by George Martin

C Start-Up

Get a C Program Up and Running

You've decided to try the C language and have a new design ready to go. Now it's time to get your embedded system up and running.

How do the C language and your compiler take care of getting your C code up and running? I thought this would be an easy article. I figured I'd just search the 'Net and pages of answers would just appear. Or so I thought.

When I started searching with "Starting C program," I got links to information about starting to learn the C programming language. Of course, my search parameters had to be more specific. The best phrase I came up with was "C code start-up." The results were more in line with what I needed. To make a long story short, I came across Derek Jones's "The New C Standard," in which he writes: "Two execution environments are defined: free-standing and hosted. Commentary… Free-standing is often referred to as an embedded system, outside of the C Standard's world."[1]

Hey, what gives? All my embedded code starts up and runs just fine (until I encounter my design and coding errors). The C language is so popular. How can the C (and C++) language ignore embedded designers and applications?

Well, if you think about it a bit, you'll see that "C environments" is a broad statement. If you're writing a C program that runs on a PC to operate on data read from a file, and if the results are presented on the display, that's an entirely different start-up than your typical embedded system. On the PC, C start-up is much more of an operating system issue than a language issue. The documents I list in the References section of this article cover how specific types of embedded systems might start up. (Refer to the "references" at the end of this article.) Let's review.

START-UP REVIEW

Clearly, start-up in Linux is much different than start-up in Texas Instruments Code Composer for the 430, and both differ from starting up the ARM. They do, however, have features in common. The differences in microprocessor architectures dictate differences in start-up.

Some of the more common considerations in starting a C environment have to do with memory and its initialization. There are different memory sections or segments of your program. Let's look at memory initialization and the .bss segment. The memory region contains variables that are to be initialized to 0.[2] These are all the global (or static) variables, except those that are set to a specific value at start-up. The variables that need to be set to a specific value are found in the .data segment. The constants are found in the .rodata segment. ("ro" probably stands for read only.) The code is found in the .code segment (also known as the .text segment). So, initializing of the segments is one of the start-up routine's functions.

Sounds simple. But think about this a bit. Memory locations that can be changed by the program (.bss and .data) need to be located in RAM so that read and write instructions can operate on them. Memory locations that do not need to be changed (.rodata and .text) may be located in RAM, but they could also be

located in EPROM or flash memory. Some embedded systems transfer these segments from flash memory (or disk) to RAM with a bootloader early in the start-up process. I've worked with compilers that generated compressed .rodata and then uncompressed that data in the start-up process. This saves a tremendous amount of space. And if you are loading your code from the EPROM of DISK into RAM, the savings are significant.

Another start-up function is setting up interrupt vectors. When an embedded microprocessor powers up, it fetches instructions from a reset vector location. That reset location along with other vector (general description) or interrupt (more specific description) locations are found in the vector table. On smaller micros, these are eight or 16 vectors, and they are constants located at a fixed memory location. On larger more powerful micros, these vectors (there can be hundreds of them) are in a table that can be changed and even relocated by the program (usually to RAM). Think about it. You may have a vector location for timer interrupt and another for divide by zero.

In the first case (the smaller CPUs), the vector table is defined in assembly language and contains the address of the interrupt routines. So, at compile/link time, all the addresses are known and the table is filled in. In the second case (the larger CPUs), the vector table is relocated and set up in some sort of start-up code. This more complicated setup is usually done in the application code. Each interrupt vector can point to the appropriate interrupt routine when that interrupt is registered. Also, priority levels and other pertinent information are defined. Setting up these vector tables is done at start-up.

We also need to set up the stack and heap. To help understand these concepts, let's look at some code. As a refresher, when you define a variable `INT16 c;`, you are reserving space for that variable. If the definition is in a procedure, then the space is created when the routine is called and destroyed when the routine is exited. When you define a variable `INT16 c = 7;`, you are reserving space for that variable and also setting it to the value 7. If the definition is in a procedure, then the space is created when the routine is called and destroyed when the routine is exited.

STACK & HEAP

How does all this happen? We need to look into the stack.

Two closely related programming concepts that C uses are the stack and heap. From the ARM literature: "The ANSI C library will initialize the stack pointer for the current mode itself. It determines where to place the stack by using one of the following options: If the symbols __heap_base, __heap_limit, __stack_base and __stack_limit are defined, their values will be used to define the top and bottom of the stack and heap."[3]

It is best to first describe how the CPU operates with a stack. In the hardware, when an interrupt occurs, information about the state of the CPU is pushed onto the stack. When that interrupt is completed, the information is popped off the stack. Other information (e.g., registers) may be saved and restored in this interrupt operation. Also in assembly language, there are the `PUSH` and `POP` instructions. Now the stack pointer is set to either the top or bottom of the available RAM. I believe it's more common to be set to the top (a numerical larger value), so let's just focus on that. But be warned: this might be different in your system. The interrupt operation (or the `PUSH` instruction) will save the data at the location pointed to buy the stack pointer and then decrement the stack pointer by the size of the data saved. The `POP` instruction will increment the stack pointer and load the data found at that location. The return from interrupt also increments the stack pointer and restores the data from that incremented location. This is also the method used to reserve space for variables and to pass parameters to a routine.

First, look at reserving space. The `INT16 a = 7;` statement caused the stack pointer to be decremented (reserving space) and then an instruction to store the value 7 at the location pointer to the stack pointer +1 (assuming a 16-bit CPU). Next, consider the call to the `INT16 Sqrt(INT16 A, INT16 B);` procedure. It pushes the A and B values on the stack and also pushes or reserves space for the return value. By looking at the assembly language output, you should be able to see how a variable is defined locally to a routine and how memory is created for the variable and then destroyed as the routine is exited. Look carefully at the assembly code your compiler generates to see how the stack is used.

The stack can't go on forever. I found that C compilers for microprocessors typically have a default stack size. You can change this default value for your specific application. Also, you can enable or disable stack-checking to test to see if the stack has grown past its defined size. Be careful. With stack-checking enabled, the code runs much slower.

Next let's look at the heap. In "Essential C," Nick Parlante writes: "C gives programmers the standard sort of facilities to allocate and de-allocate dynamic heap memory. A word of warning: writing programs which manage their heap memory is notoriously difficult. This partly explains the great popularity of languages such as Java and Perl, which handle heap management automatically. These languages take over a task that has proven to be extremely difficult for the programmer. As a result Perl and Java programs run a little more slowly, but they contain far fewer bugs. (For a detailed discussion of heap memory, refer to http://cslibrary.stanford.edu/102/, Pointers and Memory.) C provides access to the heap features through library functions that any C code can call. The prototypes for these functions are in the file <stdlib.h>, so any code that wants to call these must #include that header file. The three functions of interest are: `void* malloc(size_t size);`, which requests a contiguous block of memory of the given size in the heap; `void free(void* block);`, where the mirror

image of `malloc() - free` takes a pointer to a heap block earlier allocated by `malloc()` and returns that block to the heap for re-use; and `void* realloc(void* block, size_t size);`, which takes an existing heap block and tries to relocate it to a heap block of the given size that may be larger or smaller than the original size of the block."[4]

The heap is a memory region that is below the stack (in our example CPU). It is defined by its boundaries __heap_base and __heap_limit. The C language provides support for your code by allocating some heap memory (`malloc()`), using it, and then releasing that heap memory for other routines to use (`free()`). This can be very useful if you need to reserve lots of memory for one time (or some time) usage and then don't need that memory for a while. There is not a heap pointer in most CPUs like there is a stack pointer. Separate routines manage the heap region. You will find as you look through the references that heap usage and heap management is a difficult process. Memory leaks are a real danger. I usually allocate only one time an amount of memory large enough to do the job. And then I free that memory on exit. This very conservative approach seems to work for me.

In ending this part of the discussion, note that there can be other regions of memory such as battery-backed-up memory or EEPROM memory. Perhaps some compilers will give you the tools to deal with this at start-up. But that's at the black-belt level and we won't go there just yet.

FILES

Three files from a working project are posted on the *Circuit Cellar* FTP site. The files ncrt0.a30, sect30.inc, and NCRT0.LST are for a Renesas Technology MC16 CPU. In ncrt0.a30, you will find all the definitions of the various memory segments. Also, you'll see custom instructions defined. (Remember all caps are #defines.) In NCRT0.LST, you'll see the assembly language that is expanded from these defines. This is the actual code that is executed to set up the memory segments.

In the file sect30.inc, you'll find the vector area. I set up three vectors; the environment set up the others. One is for a timer interrupt and two are for serial port interrupts. Renesas has thousands of microcontrollers and each come in different flavors. You see that there are vector tables for this microcontroller in a 60-pin and then in an 80-pin package. I started putting my vectors in both areas. Sort of defensive programming.

What if the timer was set up before I defined the package (60 versus 80 pins)? I was new to the CPU and the development environment and didn't want to be surprised or constrained in the order I initialized the hardware.

Well, so what now? Let's write some code and look at project-specific start-up and initialization. I'll talk about the Texas Instruments MSP430 and get into the classic `for(;;)` loop.

CODE & INITIALIZATION

Main.c is the main program loop that you will find in a typical embedded system. The file starts out with a header for information purposes. It's just good practice to identify your code. I always add the copyright notices even if that might not legally secure the copyright. But if you don't mention the copyright, I'm sure you never get any protection.

Next, you'll come across the `include` of the file msp430x12x2.h. That file is specific to the CPU. It sets up all the defines that you will need to access the CPU's hardware. Here is where you get access to registers such as the timer and UART. That file comes with the compiler.

Then you'll find the type definitions of `INT8` and `INT16`. If you are a regular reader, you'll recognize these as how we keep our heads screwed on straight. The C keyword `int` changes meaning as you change micros. An `int` becomes different sizes on different micros. Our keywords of `INT8` and `INT16` do not change meanings but are translated into the appropriate constructs for our usage. We've captured (hidden) that translation in the `type` definitions.

Next you'll find the procedures that initialize the hardware (`void InitHardware(void);`), initialize the software (`void InitSoftware(void);`), and then the `for(;;)` loop. This `for(;;)` caused the code to execute here forever. It's an embedded system after all.

The `main()` is entered after start-up (and you now know all about the start-up process). The first step is to disable interrupts. Then the hardware is initialized and then the software is initialized after the interrupts are enabled. The `_DINT();` and `_EINT();` instructions are not in the C language. They are macros that translate to the assembly language instructions that disable and enable interrupts in the CPU. Clearly, these are implementation- and compiler-dependent. After initializing my system, I send out a message on the serial port and enter the `for(;;)` loop. This is the classic method for creating a program that never ends.

If you look close, you will see the `_DINT()` called twice. I do this to be certain interrupts are disabled. Each CPU instruction is actually broken into several segments. Even the single clock instructions look at the interrupt input at a specific point during the instruction. On some microcontrollers, it's possible for the interrupt to come along at a critical time so that the interrupt in the CPU disabled but the interrupt is executed. And then the return from the interrupt reenables the interrupts. Two of the disable interrupt instructions take care of this situation. Hey, those of you who are paying attention have just moved up to your gray belt. Not exactly a black belt yet.

In main.c, I first initialize the hardware and then the software. This seems to be the best way, but perhaps you have a really good reason to reverse the order. In `InitHardware()`, I take care of all the raw hardware settings. If a bit in the CPU needs to be set/cleared to define a specific hardware characteristic. You find it in `InitHardware()`.

Let's consider a UART. In `InitHardware()`, I would set up the CPU registers that control the UART parameters like the clock source, the baud rate, the number of data/stop bits, the parity, and the handshake. I might also read the receive register and make sure any error flags are cleared. In `InitSoftware()`, I would again flush out any data and reset any error flags. But I would probably make these operations a procedure and make that procedure available to the code in general. That way it would be easier to recover from UART errors.

The `InitSoftware()` procedures would initialize items such as variables, any states in state machines, pointers, and databases. If indicators such as an LED error indicator need to be on for testing or off for normal operation, I would do that here. I suppose you could have a `StartupTest()` procedure that you run after the system is stable.

UP & RUNNING

Well, I hope this gave you some insight into how to get your embedded system up and running. If you've been reading this series, you've seen references to free (as in free beer) tools. Low-cost ($50) and even free evaluation boards are also available. So, let's get started using the C language. If you have any topics that you would like to have me cover, please let me know.

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

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/228.

REFERENCES

[1] D. Jones, "The New C Standard," 2008, http://c0x.coding-guidelines.com/5.1.2.pdf.

[2] Wikipedia, BSS Section, http://en.wikipedia.org/wiki/Block_Started_by_Symbol.

[3] ARM, Documentation Homepage, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka3696.html.

[4] N. Parlante, "Essential C," http://cslibrary.stanford.edu/101/EssentialC.pdf.

RESOURCES

K. Boldyshev, "Startup State of a Linux/i386 ELF Binary," 1999-2000, http://asm.sourceforge.net/articles/startup.html.

Rowley Associates, "Startup Code," Rowley Associates Online Documentation, www.rowley.co.uk/documentation/arm_1_7/index.htm?http://www.rowley.co.uk/documentation/arm_1_7/arm_crt0.htm.

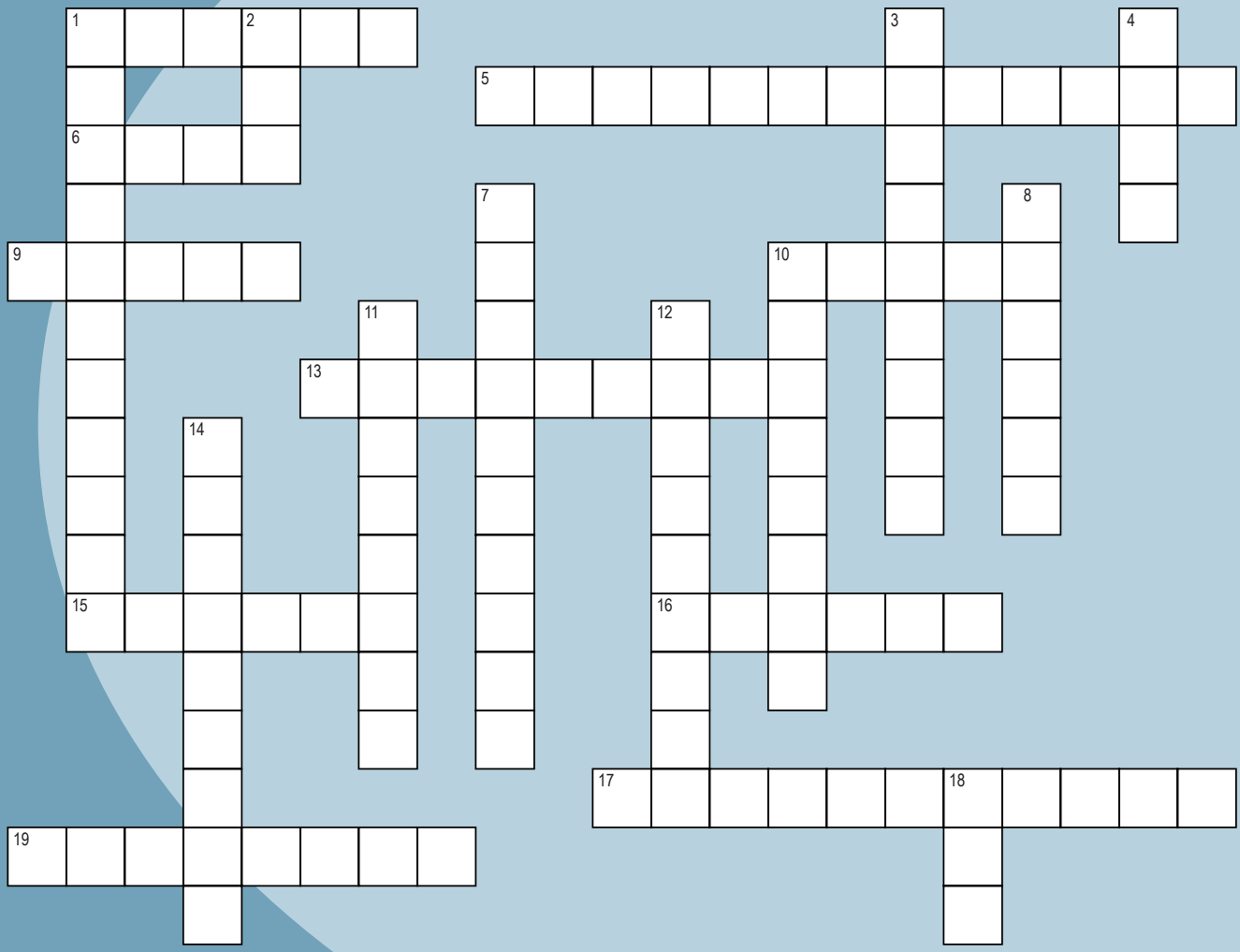
Texas Instruments, "Code Composer Essentials V 3: Modified or Custom Startup-Code," TI E2E Community (Beta), https://community.ti.com/forums/p/286/762.aspx.

SOURCES

M16C Microcontroller
Renesas Technology | www.renesas.com

MSP430 Microcontroller
Texas Instruments, Inc. | www.ti.com

CROSSWORD



Down

1. *Circuit Cellar* headquarters
2. Secure protocol
3. $ax^2 + bx + c = 0$
4. Standards Institute; Washington, DC
7. .PS
8. Links two LANs
10. Negative subatomic particle
11. F, Cl, Br, I, At
12. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610
14. Mn
18. Wire sizes

Across

1. Touching contacts, electricity flows
5. Unidirectional electric current (two words)
6. No value
9. 0.000000000000001
10. "FE" is a framing what?
13. Ground (two words)
15. Switch with a lever
16. _____ guidance computer; real-time flight info
17. mrad
19. Light waves; 10^{-10} meter

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

IDEA BOX

THE DIRECTORY OF PRODUCTS AND SERVICES

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

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

INDEX OF ADVERTISERS

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

PREVIEW of August Issue 229

Theme: **Embedded Development**

Get Started With Embedded Development (Part 1): "Bare Metal" Implementations and "CircleOS" Apps

Cable Tracer Design (Part 1): Underground Cable Detection Made Simple

Infrared Radiation Measurement: FFT Double Beam Infrared Spectrophotometer

---

THE DARKER SIDE **Power Analysis Primer:** From Power Line Measurements to PFC

ABOVE THE GROUND PLANE **A Blast for the Past:** High-Voltage DC Dosimeter Charger

FROM THE BENCH **Threat Level Indication System:** Implement a Simple USB-to-Parallel FIFO Interface

SILICON UPDATE **Thin Is In:** High-Profile Energy in a Low-Profile Package

PRIORITY INTERRUPT

by Steve Ciarcia, Founder and Editorial Director

The Critter Chronicles

It all started with the mulch, but I'm told you grass guys have similar problems. I won't reiterate my rationale for spreading 60 yards of mulch, but let's just say that the best part for me is not having a lawn at all (long story). I just spread the mulch in the spring and then listen to all my friends talk about dying in the 90° heat while they are mowing lawns every Saturday.

The bad news about spreading all this mulch for 20 years is that it has ultimately become a composted smorgasbord for all sorts of underground animal life. I used to wonder why we seemed to have three times as many robin nests than the neighbors until I realized that we're probably growing more earthworms and grubs than a bait factory. Unfortunately, the downside of creating this massive food source is that the local mole population thinks they have died and gone to their version of a Las Vegas buffet.

I don't know whether they are moles, voles, or whatever, but the result is the same—tunnels! There aren't just a few tunnels, mind you. Unabated, a month of mole mayhem is a backyard that looks like the New York subway system after a major earthquake. Over the years, I've tried a number of remedies: mechanical traps, battery-operated vibrators, "mole pellets," grub killers, and other things. After reading that certain moles avoid castor oil like the plague (Who knows what kind of stuff my particular moles or voles don't like?), last year I bought a gallon of pure castor oil—talk about an expensive deterrent! I just had to mix it in a soapy "carrier solution" (easier said than done), spray it on the affected area, and then apply a thousand gallons of water to soak it down to "mole level." Needless to say, it was about as effective as everything else I've tried.

I don't like admitting defeat, but I have a 4-oz nemesis that has definitely been winning the war so far. I've considered a massive overkill application of grub killers and assorted chemical-warfare agents, but I do care about the environment, and I don't want to poison all the birds as a consequence. Besides, I think these guys are attracted by the earthworms rather than the grubs, and since our domestic water supply comes from a well, chemically eradicating all life in my yard might accidentally include me as well.

The one tiny glimmer of hope in the "Critter Chronicles" was that the critters didn't seem to like heavy machinery driving across their latest dig sites. Whenever I spent an afternoon in the backyard with the tractor or some other motorized vehicle, it seemed to take the moles a day or two before they ventured back. "Ah hah! Perhaps there is a simple engineering solution rather than using WMD," I thought. "Apparently, the critters don't like vibration (at least from 2-ton tractors anyway), but the degree of vibration must be important too."

I bought a bunch of those wimpy battery-operated vibrators a number of years back, but they were worthless. Certainly it was because there wasn't enough vibration to be of consequence. I decided to fix that. I had some quick thoughts about sinking a dozen 5′ steel rods in the yard with 0.5-horsepower, paint-shaker motors welded to them, but it would have been pretty ugly. Fortunately, sanity prevailed, and it was off to Home Depot to get parts for a more cost-effective alternative.

Big box stores attract all kinds. Unfortunately, because engineers are a minority population, the Home Depot staff hasn't experienced enough of us to avoid hitting the Panic button when we start selecting out-of-the-ordinary project materials. There I was in the plumbing aisle cutting 8″ pieces of plastic pipe fitted with closed-end caps and piling them in the cart as three staffers walked down the aisle toward me. One of them cautiously asked, "Sir, are you planning on filling those pipes with something flammable?"

Immediately, I realized that these guys were wondering if I was some kind of misguided terrorist who perhaps didn't realize the difference between 350-PSI plastic and 4000-PSIG cast iron pipe. I just smiled and said, "Nope, I'm making mole bombs!" To make a long story short, the terror on their faces evaporated when I explained that I was using the pipes for a less nefarious purpose. I said each pipe would contain a powerful 12-V DC motor with an off-center metal weight attached to the shaft—basically a big vibrator—and would be buried in the yard.

Like everything else I do these days, this project ended up to be more than I originally anticipated. Buried vibrators with enough power to constitute acceptable overkill take a bit of current at 12 V. Ultimately, the three sections of the yard where mole bombs would be tested required separate heavy-duty power supplies and heavy-gauge wiring out to each "field." In addition, since I didn't want the moles to get used to it, I installed a central controller that was wired to the three field controllers and programmed to randomly select on and off times for the vibrators.

Who knows if it works? At this point, I have nine vibrators buried and six more planned. It's been too short a time to see if there has been any affect at all or if this is just another big expensive boondoggle. If that's the case, I think my next attempt will involve using ground-penetrating radar and a harpoon. ;-)

steve.ciarcia@circuitcellar.com

Sweet!

Introducing the MiniCore™ Series of Networking Modules

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

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

MiniCore Module Development Kits

From **$49** Limited time offer.

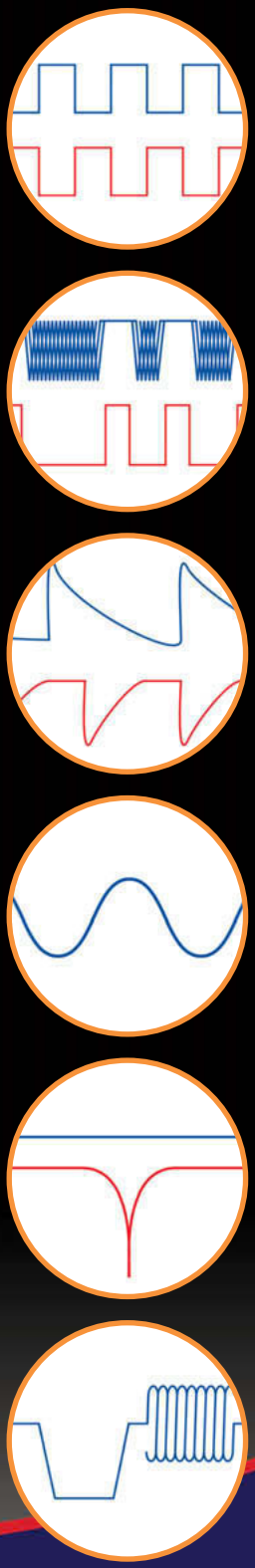
Buy now at: **1.888.411.7228** • **rabbitwirelesskits.com**

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

RABBIT™

09009

And you thought it could only do 8 things at once...

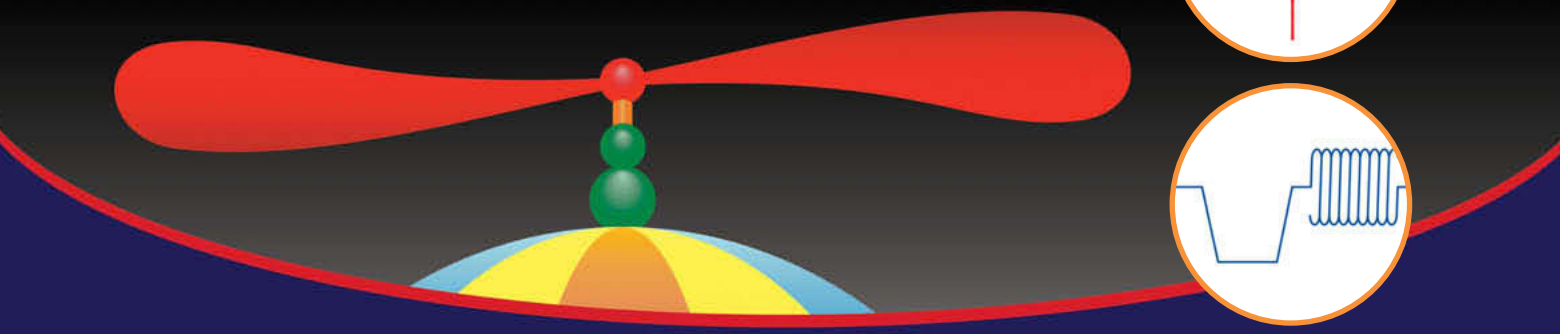


Each of the multiprocessor **Propeller chip's** eight cogs has two built-in **counter modules**. These **configurable state machines** can be set to 32 different modes to perform repetitive tasks while the cog continues to execute code. Use counter modules to:

- **Generate square waves and clock signals up to 128 MHz**
- **Measure pulse and decay durations**
- **Generate precise pulses and PWM signals**
- **Measure signal characteristics such as frequency and duty cycle**
- **Track I/O pin states with 22 logic modes**
- **Perform sigma-delta A/D and duty-modulated D/A conversion**
- **Synthesize custom signals including audio effects**
- **Produce NTSC, PAL, and VGA signals when used in conjunction with each cog's Video Generator hardware**

With the Propeller chip's counter modules, integrating multiple repetitive parallel processes into time-sensitive deterministic applications is suddenly simplified.

For prebuilt Propeller code objects, tutorials, and example applications, go to **www.parallax.com/go/counters**



Get more information at **www.parallax.com/propeller** or call our Sales Department toll-free at 888-512-1024 (Mon-Fri, 7 a.m. - 5 p.m., PDT).

PARALLAX INC

www.parallax.com

CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

BONUS ARTICLE

by Thomas Alldread

NimbleSig III

A New and Improved DDS RF Generator

The NimbleSig DDS generator was unveiled in a 2007 *Circuit Cellar* article. The "NimbleSig III" is the newest version of the design. Thomas describes the updated features and firmware.

Much to my delight, my original NimbleSig direct digital synthesis (DDS) RF generator design won Second Prize in the Luminary Micro Design Stellaris 2006 contest. *Circuit Cellar* subsequently published an article I wrote about the project and design process ("NimbleSig: A Compact DDS RF Signal Generator," *Circuit Cellar* 208, 2007). Since that time, I have updated the design, which I now call "NimbleSig III" (see Photo 1). In this article, I'll bring you up to speed on the project's current status, and I'll describe the improvements I made to the original NimbleSig design.

UPDATED FUNCTIONALITY

A significant improvement over the original design is that the NimbleSig III includes a relatively new Analog Devices AD9958 DDS IC, which incorporates two DDS engines. These engines produce a pair of RF generator outputs. The two DDS engines can operate independently on different frequencies or they can be phased-locked together on the same frequency. When locked together, the relative phase offset between the two generators can be accurately specified. Additionally, I increased the original NimbleSig's maximum frequency of 160 MHz to 200 MHz.

I use the fast interrupt feature of an NXP Semiconductors LPC2138 microcontroller, along with an assembler code service routine, for the higher-rate modulation. As you can see in columns A, B, and C in Photo 2, the modulation linearity outperforms most legacy generators. The amplitude steps are much finer for the higher-modulation rates than what's obtained with the original NimbleSig design. Photo 2 (D1) shows an extremely deep Bessel zero carrier null. This reflects the purity of the FM modulation and the accuracy of the deviation setting. The spectrum analyzer display in Photo 2 (D2)—which spans 0 to 200 MHz—shows the typical spectrum purity of the output. The amplitude and frequency of the spurs vary with operating frequency. They typically remain more than 50 dB down from the –10-dBm output level.

MPU AND HARDWARE

The NimbleSig III's MPU is an NXP Semiconductors LPC2138 ARM 7 series microcontroller. It provides 512 KB of program memory space that's roomy enough to provide

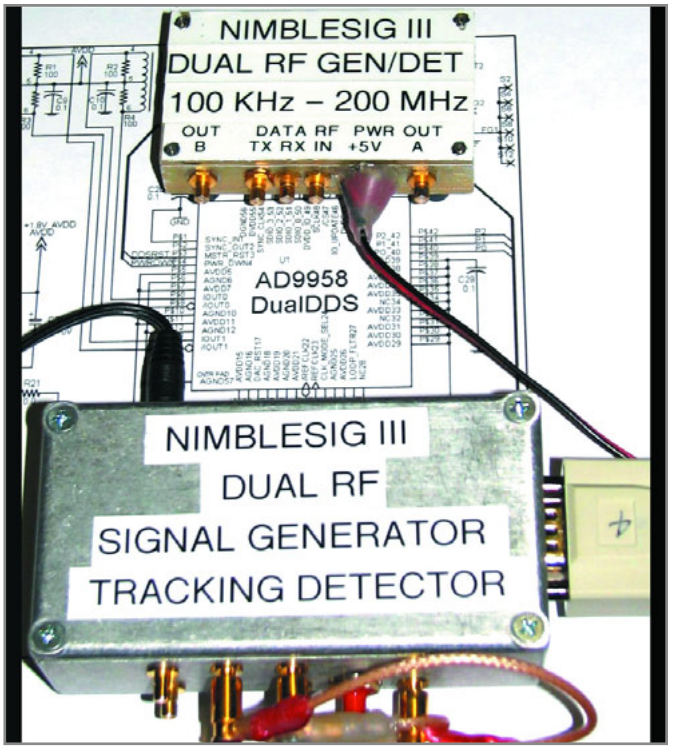


Photo 1—The NimbleSig III is an up-to-date version of my compact DDS RF signal generator. Here you see two different modules.

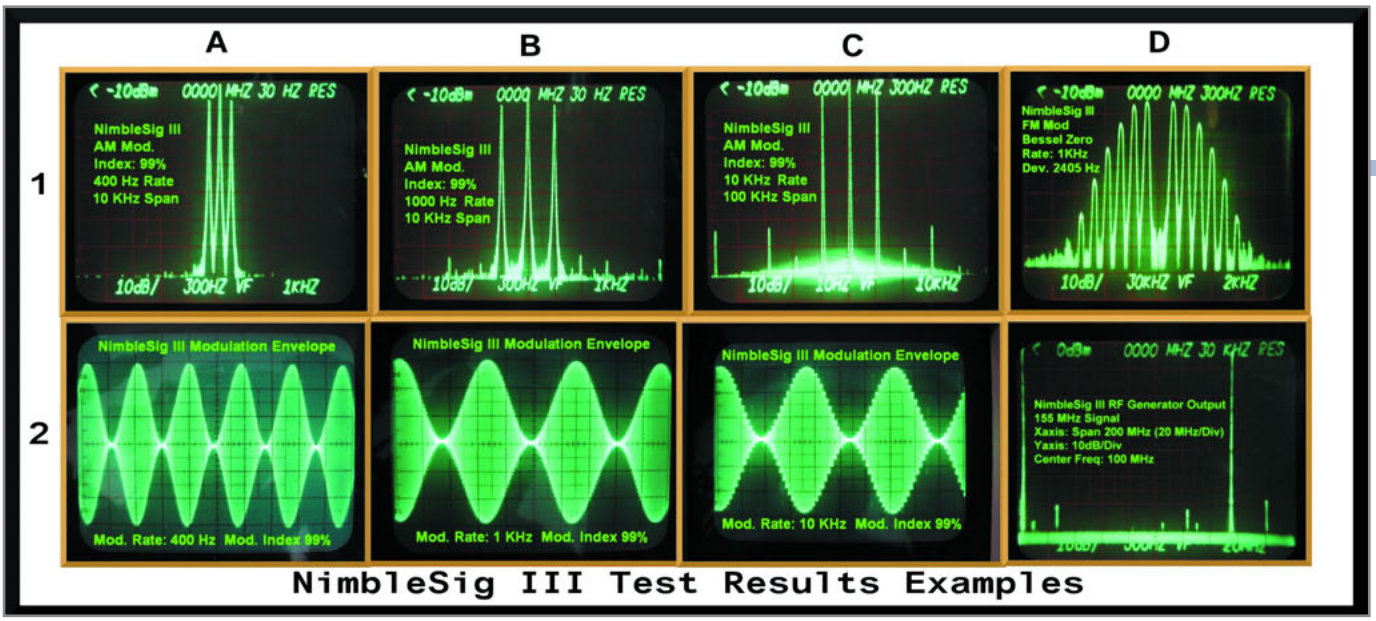


Photo 2—Take a look at the NimbleSig III's modulation performance and spectrum purity. These are examples of test results.

an improved plain-language user interface with a lot of Help files. An external 64-KB EEPROM is now provided for the nonvolatile storage of initialization variables and calibration data. I also chose a different device for the linear regulators, which use an SMD package that's much easier to work with.

As you can see in Figure 1, a 25-MHz internal TCXO provides the MPU clock and optionally provides the reference for the DDS. The full TCXO output of 3.3 V_{PP} is reduced to the required DDS input level of 1.4 V_{PP} by a voltage divider when the TCXO is used for the frequency reference. An external reference may be injected by replacing the voltage divider with a termination resistor. The DDS operates with a 500-MHz reference by appropriately multiplying the reference signal frequency with an internal PLL. Similarly, a PLL within the 50-MHz MPU internally multiplies the TCXO frequency by two.

The DDS outputs feed a pair of seven-pole elliptic filters

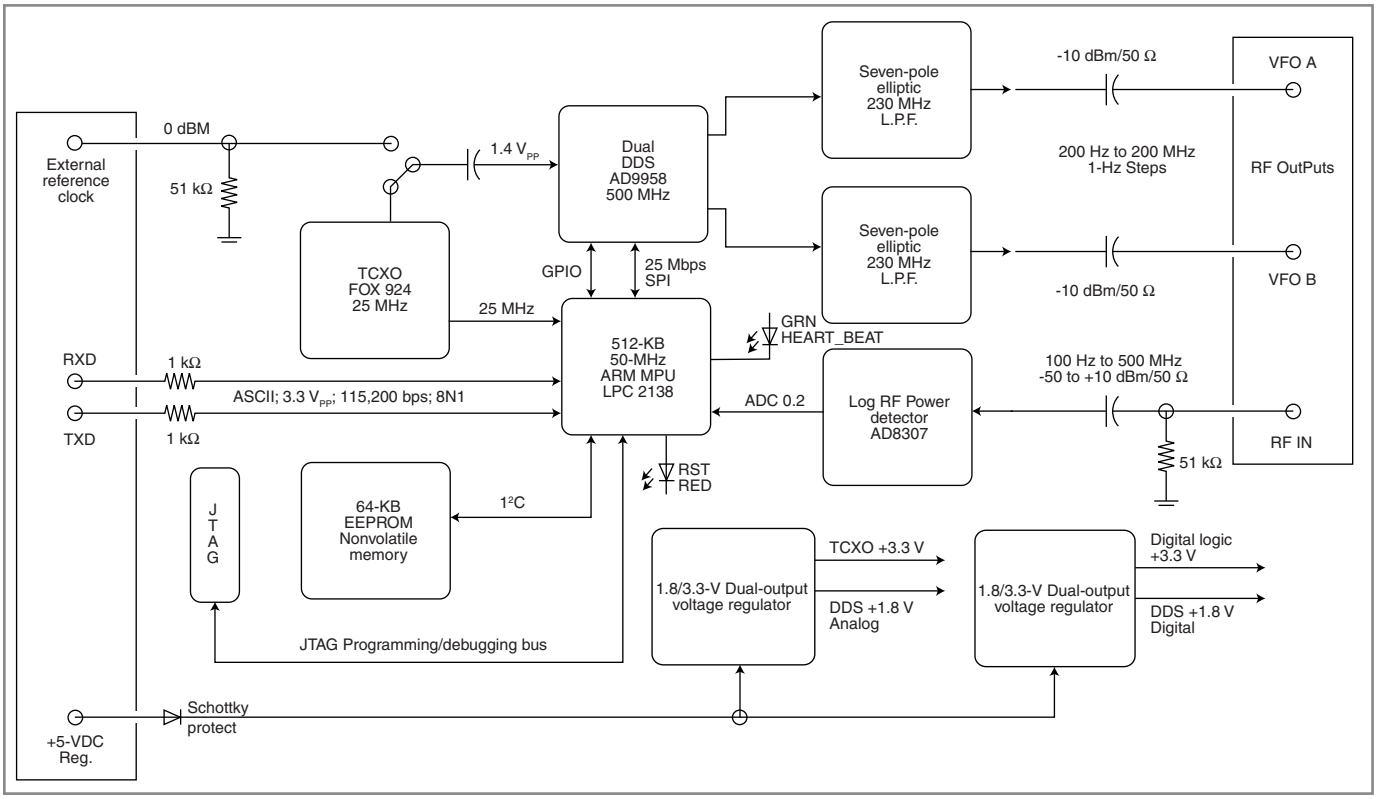


Figure 1—I built the NimbleSig III around an NXP Semiconductors LPC2138 microcontroller.

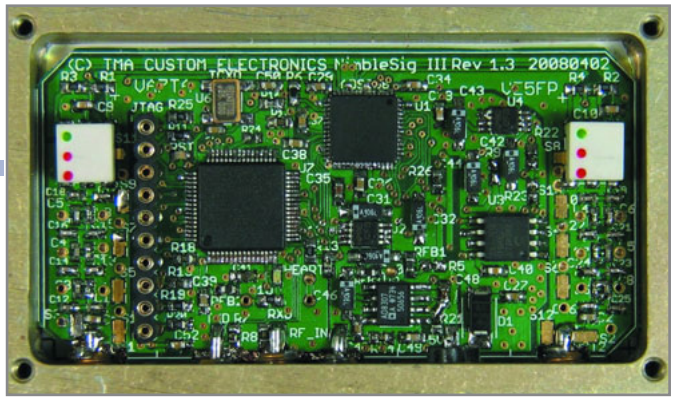


Photo 3—This is a top view of the populated PCB.

constructed from tiny 0402 SMD air core inductors and NPO capacitors. Shunt capacitor pairs are used between sections to minimize the effects of stray inductance. Design information and test results for the filters are posted on my web page (www3.telus.net/ta/NimbleSig%20III/NS3_230%20MHz%20LowPassFilter/index.html).

A Schottky rectifier provides reverse-polarity protection. Separate linear regulators are provided for the TCXO 3.3-V, DDS analog 1.8-V, DDS digital 1.8-V, and 3.3-V digital power buses. Photo 3 shows the component layout.

FIRMWARE

I expanded the firmware from the original 16 KB size up to about 100 KB. The new firmware provides more than 130 commands that harness the capabilities of the NimbleSig III module as follows:

— Frequency Setting 200 kHz to 200 MHz – 1 Hz steps
— Relative Phase Offset 0 to 360 degrees – 22 milliDegree steps
— Modulation Rate 1 Hz to 20 kHz – 1 Hz steps
— AM Modulation Depth 1 to 99% – 1% steps
— FM Deviation 1 Hz to 100 kHz – 1 Hz steps
— Output Level Reduction 0 to 10 dB – 0.1 dB steps
— Power Level Meter measurement modes - single sample, 128 averaged, 1024 averaged, minimum and peak
— Power Level Meter Calibration – detector dynamic range tracking
— Power Level Meter Calibration – frequency response 200 kHz to 500 MHz
— A and B Signal Generator – 10 dBm output level frequency response correction – 200 kHz to 200 MHz
— DDS register value modification commands
— Calibration table screen dumps
— Help pages that list the commands and provide syntax
— Control of the human interface verbosity level

The majority of the new firmware is written in C within the "CrossWorks for ARM" IDE by Rowley Associates. CrossWorks, in turn, uses the GNU GCC compiler. I became interested in the CrossWorks for ARM product because Rowley offers a low-cost, noncommercial license for this fully functional, fully supported professional IDE. The NimbleSig III's current code size is around 100 KB, so it far exceeds the crippling limitations of most trial-version IDE platforms, which are typically expensive to license.

The NimbleSig III software uses some proprietary code from Rowley that can't be released in source code format under the terms of the license. Thus, the source code is not available. Although I retain the copyrights for the NimbleSig III program, the current beta release of the object code is available for noncommercial use by hobbyists. You may download the memory image file in Intel hex code format from my web site.

BARE PC BOARD

At the time I write this, I have a limited supply of bare NimbleSig III prototype PC boards. The current boards require four rather minor modifications and do not have solder-mask protection over the vias, which I left bare for test point access. Go to my web site for detailed views and information about the bare boards and needed modifications.

Please note that good surface-mount construction tools, a good microscope (or equivalent), and well-developed skills in surface-mount construction are needed to successfully populate the NimbleSig III board. Some of the ICs have fine-pitched contact spacings and small 0402/0603-size discrete components are used.

Although I am personally very pleased with the NimbleSig III's performance, with consideration for the usual need for liability protection, I must emphasize that there are no implied warranties or guarantees associated with the use of my NimbleSig III prototype design for any application. Additional schematics, parts lists, test results, and more are available on my web site.

Author's note: I wish to express my gratitude to Dr. James A.R. Koehler for introducing me to surface-mount construction and for his significant contribution. I also wish to thank Analog Devices, for their innovative line of fine IC products, and Rowley Associates, for their wonderful IDE and excellent customer support. Additional information about this new design was published by the ARRL in a series that spans the first three 2009 issues of QEX.

Thomas Alldread (nimblesig@telus.net) graduated from the CREI's Telecommunications Engineering Technology program and received certification from MANSCETT. After working in the telecommunications industry as a technician, instructor, and transmission standards engineering specialist, Thomas worked for Bell Canada as a long-distance facilities management advisor in Saudi Arabia. Now retired, Thomas lives with his wife on picturesque Vancouver Island, where he spends most of his time pursuing interests such as designing surface-mount electronics, hobby farming, RVing, digital photography, and amateur radio.

Resources

Analog Devices, "Two-Channel 500 MSPS DDS with 10-Bit DACs," D9958, 2008, www.analog.com/en/rfif-components/direct-digital-synthesis-dds/ad9958/products/product.html.

American Radio Relay League, *QEX*, www.arrl.org/qex/.

Sources

AD9958 DDS IC
Analog Devices, Inc. | www.analog.com

LPC2138 Microcontroller
NXP Semiconductors | www.nxp.com

CrossWorks for ARM
Rowley Associates | www.rowley.co.uk

by Peter McCollum

Sound Synthesis Made Simple

A Multi-MIPS Music Box

Peter recently entered the world of simple music synthesis. He built a compact electronic sound-generating box. The design features DC-to-DC converter circuitry, an EEPROM containing sequences of notes, 18 LEDs, and a microcontroller and audio circuitry.

My daughter has a collection of music boxes and the mechanisms that they contain. She has several of the standard boxes that you wind up to play a single song. But we recently came across a more exotic type of music box in an antique store. It had a single drum that was longer than normal with a large number of pins. After playing one song, the mechanism shifted the drum sideways slightly so that a different set of notes played. This music box was too expensive for our budget, but it occurred to me that I could build an "equivalent" design using some familiar parts: microcontrollers, op-amps, and other electronic devices! And so a project idea was born: Build an electronic device that could emulate the features of a multiple-song music box mechanism. The wind-up motor would be replaced by a battery and some DC-to-DC converter circuitry. The drum that defines the song (or songs) would be an EEPROM containing sequences of notes. The mechanical governor that allows it to play at a constant speed—plus the tines of the "comb"—would be replaced by a microcontroller and audio circuitry. Because a mechanical version would have moving parts that could be watched, I decided that a row of 18 LEDs was needed to give some visual effects. (Common music box mechanisms play 18 notes, or 1.5 octaves on the standard scale.)

Another purpose of this project was to explore the digital-to-analog converter (DAC) peripheral on Microchip Technology's dsPIC33FJ*xxx*GP8*xx* series of microcontrollers. The DAC supports two channels of 16-bit data at a rate of up to 100,000 samples per second. This project seemed like a great excuse to learn about a new peripheral (see Photo 1)!

WIND-UP MOTOR POWER

My plan was to use a fairly small battery pack and avoid the inconveniences of linear regulators. The MPU—a Microchip Technology dsPIC33FJ64GP804—requires 3.3 V, yet the audio components I had chosen work best on 5 V. The solution was to use a pair of Microchip MCP1253 DC-to-DC charge pump converters, one of each configured to provide the two required voltages (see Figure 1). These chips come in a small eight-lead MSOP package and do not require an inductor for operation. The devices accept an input voltage of 2.1 to 5.5 V—and so they operate in either a "buck" or "boost" mode—as needed. This wide input range enables the device to keep operating normally even when the battery is mostly depleted. For a battery pack, I chose a bank of three AAA cells to provide a nominal 4.5-V input.

NOTE GENERATION

To produce a musical note through a DAC, the first step was to define the basic waveform numerically as a sequence of numbers that represent exactly one full cycle. I call this the "wave table." I chose to represent a cycle with eight values: four for the positive half-cycle and four for the negative. If these eight values are made available to the DAC at a fixed rate, the output is a tone that has a frequency of one-eighth the DAC's input rate. To represent an approximation of a sine wave, the values were: [0.0, 0.707, 1.0, 0.707, 0.0, –0.707, –1.0, –0.707]. The DAC in the MPU works with 16-bit values, so the numbers I actually used were these signed integers: [0, 1414, 2000, 1414, 0, –1414, –2000, –1414].

Once I could produce a tone of a certain frequency, the next step was to define a series of possible tones that represented 18 adjacent notes on the standard musical scale. Notes that are one octave apart differ in frequency by a factor of two, and any two adjacent notes differ by a factor of the twelfth root of two. In my scheme for controlling the tone-sample timing, the pitch of a note was really defined by its period, rather than its frequency. So, a partial listing of the notes [C, C#, D, D#, E, ...] was represented by period values of [2100, 1990, 1870, 1770,

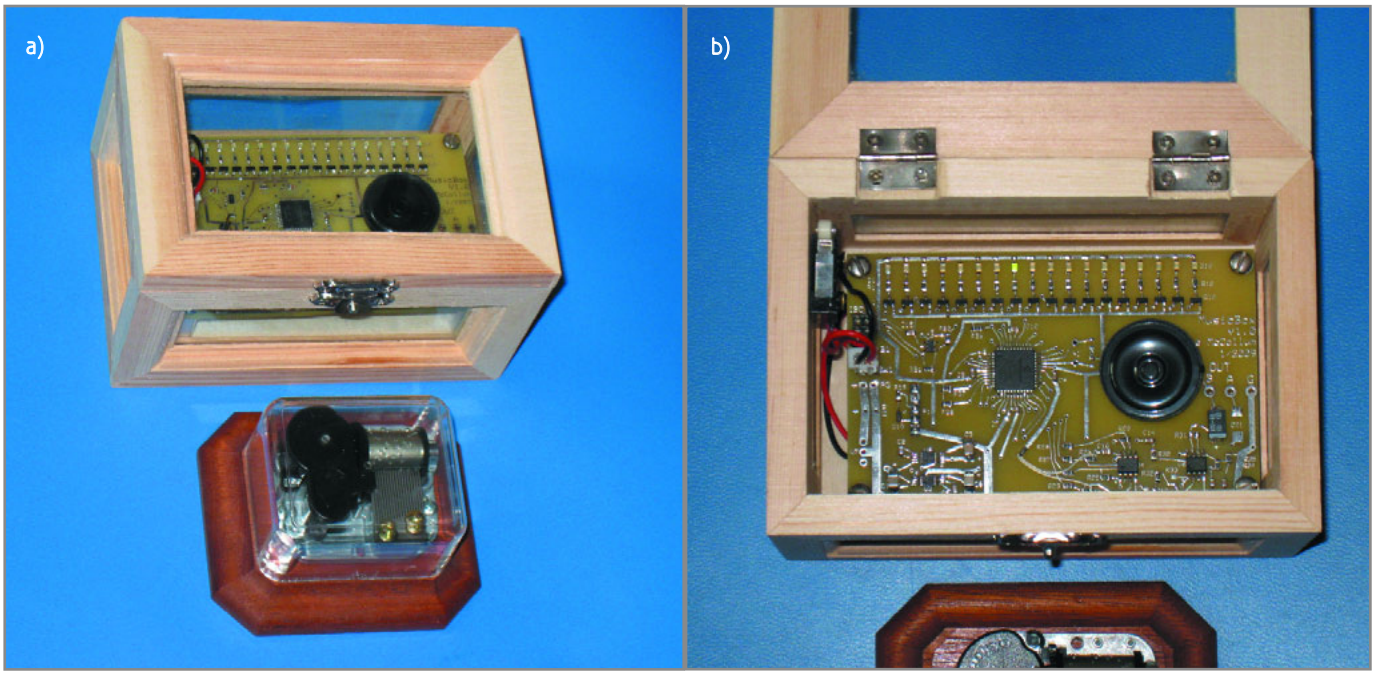


Photo 1a—This is the electronic music box, and its mechanical counterpart. My version is mounted in a small box from a craft store. **b**—On the left edge of the box, I mounted a switch that powers up the board when the lid is opened. An array of green SMT LEDs is across the top edge of the board, with a small speaker on the right. The three-cell battery pack is under the PCB.

1670, ...]. In the MPU, any given note can be played by assigning the appropriate period value to an MPU timer, and each time the timer expires, the next value in the wave table (described in the previous paragraph) is output via the DAC. Therefore, in the code, only a single value assignment was required to play any of the available notes. Also, it is fairly

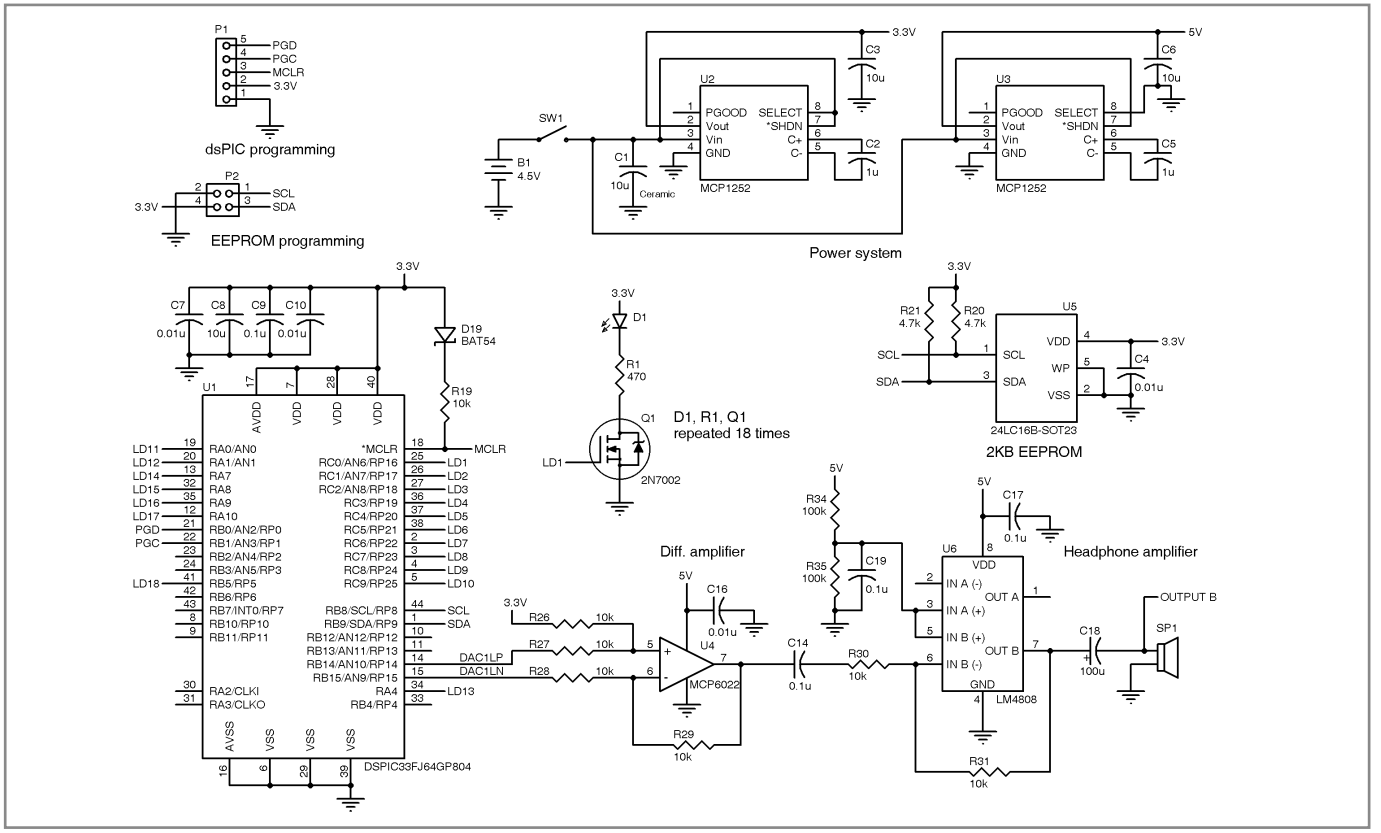


Figure 1—The four main portions of the circuit are the MPU, power system, EEPROM, and audio section. The audio section was adapted from a suggested design in the dsPIC datasheet. My PCB design includes the additional components for a second audio channel so that the board could be used in another application that requires stereo sound.

easy to play multiple notes in parallel (polyphony), if multiple timers are used.

One more feature was necessary in order to produce an output tone that sounded roughly like the plucking of a tine in a music box: an amplitude envelope. Without applying an amplitude envelope, the output is just a sequence of "beeps" that has no personality at all and sounds rather robotic. The solution was to control the volume of each tone so that it started out loud but decayed to zero in a short time. In a traditional music synthesizer, an ADSR (Attack, Decay, Sustain, Release) envelope generator is used. But I simplified the model to be just an instantaneous attack followed by a linear decay. I believe I can improve the music box emulation's sound quality by implementing a more complex ADSR system. I encourage you to experiment in this area.

SONG SEQUENCING

My Music Box system has the capability to play 18 different notes. The final step in the design was to provide sequences of notes that represent songs. The board includes a Microchip Technology 24LC16B 2-KB EEPROM for storing not only song information, but also the aforementioned wave table and note period values. The MPU communicates with the EEPROM via an I^2C bus. I used a Total Phase Aardvark I^2C/SPI host adapter to easily program song data into the EEPROM. The PCB design includes a four-pin header that mates directly with the Aardvark device. Total Phase's free Flash Center software package was used for reading and writing EEPROM and flash memory devices.

Playing a song in a recognizable fashion requires both a sequence of notes and the timing of the notes. In the case of a music box, the length of the actual notes is fixed, but the time between the notes varies. In the EEPROM, I stored sequences of bytes that represented either notes or time delays. For notes, the byte value was simply the number of the note, from 0 to 17. But for time-delay bytes, the high-order bit was set, and the remainder of the byte represented a relative time delay value.

The software reads each byte, checks the high-order bit, and then plays either a new note or a new time delay. Using this system, the EEPROM has room for seven songs of up to 127 steps each, where a "step" is either a note or a time delay. One additional byte indicates the length of the song.

The EEPROM is also used to record which song plays in a nonvolatile manner. This is so that the next song plays each time the Music Box's lid is opened according to the order that they are stored. An interesting variation would be to choose the next song randomly. By using the EEPROM to store a state variable, I could easily implement a pseudo-random sequence. Or can you think of a way to produce a truly random value from a device that always powers up in the same state? I'll leave it up to you to consider.

A FINAL "NOTE"

I hope you've enjoyed this little excursion into the world of simple music synthesis. You can also use this board as a handy development platform for exploring the DAC peripheral available on certain dsPIC33 series MPUs. Feel free to e-mail me your questions.

Peter McCollum (saipan59@Q.com) holds a BSEE from Colorado State University and has been working in the computer industry since 1981. He currently works as a firmware engineer focused on MPU applications in storage systems. As a hobby, Peter designs and builds projects that range from microcontroller devices to vacuum-tube radios.

PROJECT FILES

To download the code and a project video, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/228.

SOURCES

dsPIC33FJ64GP804 Microcontroller, MCP1253 converters, and 24LC16B EEPROM
Microchip Technology, Inc. | www.microchip.com

Aardvark I^2C/SPI Adapter and Flash Center software
Total Phase | www.totalphase.com