

www.circuitcellar.com

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

DATA ACQUISITION

Intelligent Power Supply

High-Performance
Motor Control

Protect Your Embedded
Designs

I-Zip Data Acquisition



Announcing a complete hardware and software solution from NetBurner

The NetBurner MOD5234

ETHERNET CORE MODULE with eTPU



Only **\$99**
Qty. 100

NetBurner MOD5234 Ethernet Core Module Features

INDUSTRIAL TEMPERATURE RANGE

| -40°C to +85°C

PERFORMANCE AND MEMORY

| 32-Bit CPU | Freescale ColdFire MCF5234 147 Mhz
| 2MB Flash Memory | 8MB SDRAM

DEVICE CONNECTIVITY

| 10/100 Ethernet | 3 UARTs | 16-channel eTPU | I²C | SPI | CAN
| 47 Digital I/O | 16-bit Data Bus | SD/MMC Flash Card Support

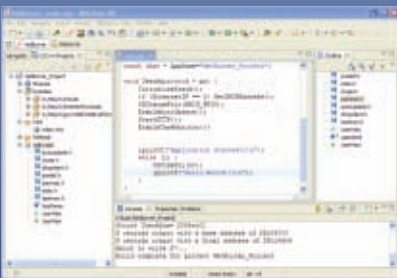
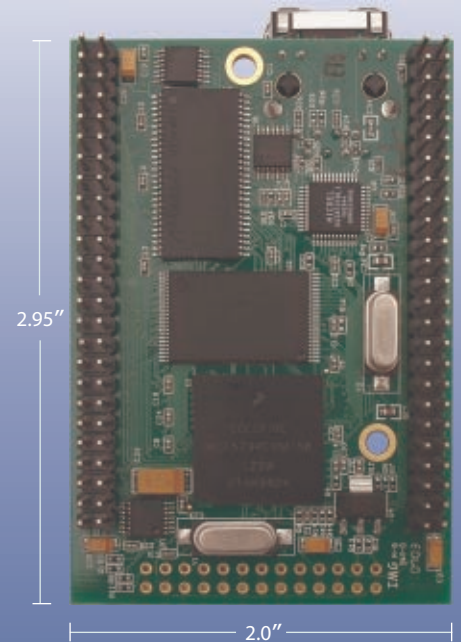
Customize with NetBurner's Royalty-free Software Suite

DEVELOPMENT SOFTWARE

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

COMMUNICATION SOFTWARE

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



All hardware and software is included with the

NetBurner MOD5234 Development Kit for only \$299!

The Development Kit features NetBurner's Eclipse, an enterprise level professional IDE that provides editing, downloading and debugging in one environment.

Order the MOD5234 Development Kit: Product No. NNDK-MOD5234-KIT



Product No. | MOD5234-100IR
Information and Sales | sales@netburner.com
Web | www.netburner.com
Telephone | 1-800-695-6828



Real Mixed-Signal Programmability.



Get PSoC®. Because change happens.

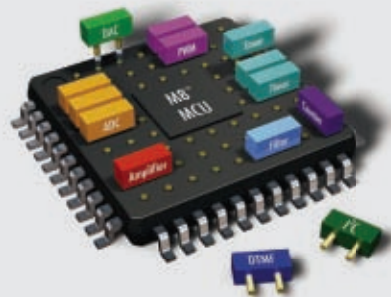
PSoC flexibility enables changes anytime: at concept, through production, in the field. Specifications change constantly. Yet pressures to differentiate, minimize costs, and speed time-to-market remain the same. To stay ahead of the curve, you need flexibility, programmability, and scalability. PSoC's unique programmable architecture delivers this and more. Futureproof your design; make PSoC your agent of change.

PSoC delivers:

- The configurability of an FPGA, the mixed-signal integration of an ASIC, and the familiarity of an MCU.
- Reusable IP, compatible device families and variable resource options ensure you can optimize design efforts and accommodate changes.
- The industry's first visual embedded design tool, PSoC Express™, speeds design time, enabling you to generate a complete design without writing a single line of code.

GET STARTED WITH PSoC NOW.

- Download our "Change Happens" White Paper and get 50% off a PSoC development kit: www.cypress.com/changepaper
- Download free PSoC Express™ visual embedded software: www.cypress.com/changesoft
- Request free PSoC device samples: www.cypress.com/changechip
- Free online training: www.cypress.com/changetrain
- Purchase PSoC development tools: www.cypress.com/changetools



PSoC includes programmable analog and digital blocks, a fast MCU, flash and SRAM memory, all in a compact package (as small as 3mm x 3mm).





Link Instruments

PC-Based Test Equipment

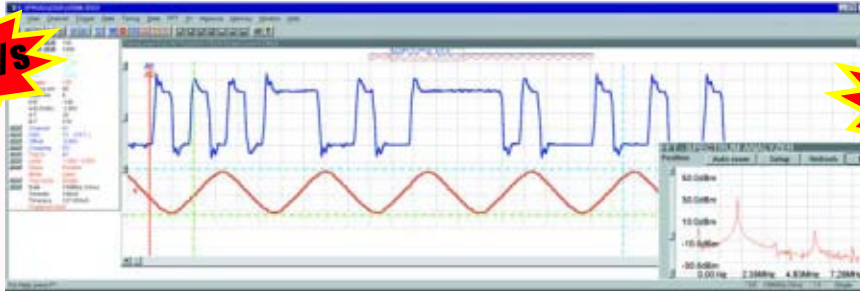
Digital Oscilloscopes



- 2 Channel Digital Oscilloscope
- **500 MSa/s** max single shot rate
- 1Mpt sample memory
250 MSa/S (Dual channel) 512 Kpts
500 MSa/S (Single channel) 1 Mpts
- Only 9 oz and 7" x 3.5" x 1.5"
- Portable and Battery powered
- USB 2.0
- FFT Spectrum Analyzer

DSO-8202 (200MSa,128K) **\$799**
 DSO-8502 (500MSa,1MPT) **\$950**

500MSa/s

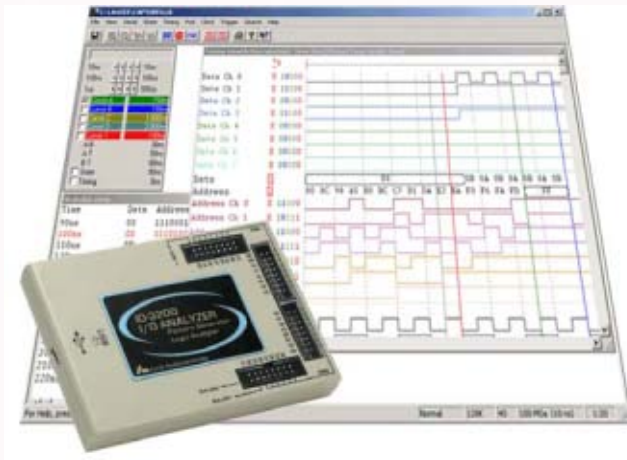


1Mpts

Logic Analyzer & Pattern Generator

NEW!

Portable
3.2" x 3" x 0.65"
USB 2.0 Powered



- Logic Analyzer (32 channels)
- Pattern Generator (up to 32 channels)
- up to 400 MSa/s
- Variable Threshold
- 2 External Clocks
- SPI output and disassembly
- I²C output and disassembly
- up to 2Msamples/ch

IO-3208A **\$750**
 IO-3232A **\$899**
 IO-3232B **\$1399**



Link Instruments (973) 808-8990

www.Linkins4.com

U-VERSION VFD

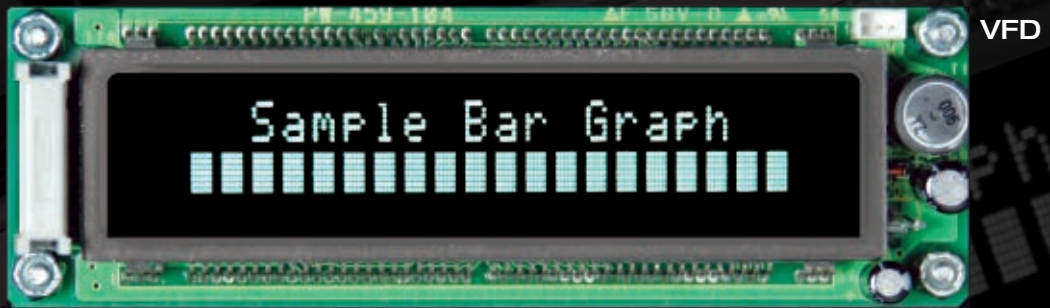
Noritake

Drop-in Replacement for LCD Modules

The new family of U-Version modules is compact, low power and lower in cost with a custom designed chip. This new VFD technology has an 8 and 4 bit parallel interface and enables the replacement of LCD's with Noritake U-Version VFD modules.



LCD



VFD

**UPGRADE
NOW...**

- Brightest Display
- Widest Viewing Angles
- Widest Temperature Range
- Fits Right In
- No Programming Change
- Low Cost

www.noritake-elec.com/53

TASK MANAGER

Blogs & Video

During the past year, I've received numerous e-mails from readers who've begun using two inexpensive technologies to chronicle and show off their projects: blogs and video. Every other week or so, I receive an e-mail from a reader who points me to a web site or video file that showcases an interesting new project. A great example is Miguel Sanchez's vertical plotter project, which you can view at www.youtube.com/watch?v=VmB14M78CWU.

Blogging about your projects and presenting your peers with short videos are effective ways of sharing your ideas and receiving useful feedback about your designs. I encourage you to give it a try, and when you do, be sure to send us an e-mail with a link to your work. We'd love to have a look. If your project intrigues us, we might approach you about the possibility of writing an article for the magazine.

This month, in addition to our usual batch of interesting columns, we're featuring seven great articles that will keep you in design mode until the new year.

On page 10, Alexander and Jordan Popov describe an intelligent power supply setup for an embedded system. Read this article before you begin testing your next design.

Dale Wheat's PIC-an-LCD design is a PIC-based serial LCD controller (p. 18). As you'll see, the chip is programmed in C and can work with most alphanumeric LCD modules.

In "High-Performance Motor Controller," Daniel Ramirez explains how he uses a USB interface to issue high-level commands to his motor controller from his laptop (p. 26). It's your turn to give it a try.

Are you ready for some 1-Wire solutions? In the second part of his "1-Wire in the Real World" series, Steve Hendrix finishes explaining how he built a 1-Wire master (p. 42). Now you can put a 1-Wire solution in play even if it's in a "hostile" environment.

On page 52, Aubrey Kagan begins a new series of articles about the art of protecting an embedded design. This month, he focuses on the topics of power supplies, inputs, and ground.

Once you're confident that you can protect your designs, check out Stuart Ball's article entitled "Pulse Generation" (p. 58). He encourages you to try a continuous rotary knob in your next design instead of a keypad or buttons.

Wrapping up the feature article section of the magazine is Thiadmer Riemersma's article about embedded scripting (p. 62). He explains how to harness the power of the Pawn scripting language. As you'll see, it's possible to extend your firmware without changing it.

Happy reading.

Remember: if you try your hand at a design that's similar to any of those covered in this issue, be sure to write about the experience and shoot a short video. We want to see what you're up to!

One last note. Keep in mind that the submission deadline for the Microchip 16-Bit Embedded Control Design Contest is October 16, 2007. That means you have about a month and a half to finish up your designs and submit your entries. With \$15,000 in total cash prizes up for grabs, this is sure to be a highly competitive contest. Good luck!

cj@circuitcellar.com



CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

CHIEF FINANCIAL OFFICER
Jeannette Ciarcia

MANAGING EDITOR
C.J. Abate

MEDIA CONSULTANT
Dan Rodrigues

WEST COAST EDITOR
Tom Cantrell

CUSTOMER SERVICE
Debbie Lavoie

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

CONTROLLER
Jeff Yanco

ART DIRECTOR
KC Prescott

NEW PRODUCTS EDITOR
John Gorsky

GRAPHIC DESIGNER
Mary (Turek) Sobuta

STAFF ENGINEER
John Gorsky

PROJECT EDITORS
Steve Bedford
Ken Davidson
David Tweed

ASSOCIATE EDITOR
Jesse Smolin

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.

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

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

Entire contents copyright © 2007 by Circuit Cellar, Incorporated. All rights reserved. Circuit Cellar is a registered trademark of Circuit Cellar, Inc. Reproduction of this publication in whole or in part without written consent from Circuit Cellar Inc. is prohibited.

NKK[®]
SWITCHES

PROFITABLE PROJECTS

TAKE YOUR DESIGN PROJECTS INTO FULL BLOOM

Take the time to stop and smell the profitability. NKK's SmartSwitch™ series of LCD programmable switches and displays are available in a virtual bouquet of colors and dynamic graphics making it both simple and cost-effective, for your inspired designs to blossom. Better, more efficient designs and processes will improve your bottom line and help your projects flourish.



IS Dev Kit-2

SMARTSWITCH[™]

ACTIVATIONS MADE SIMPLE

Get Your **FREE**
3D CAD Model at:
<http://cc.nkksmartswitch.com>

1.877.2BUYNKK (228-9655)

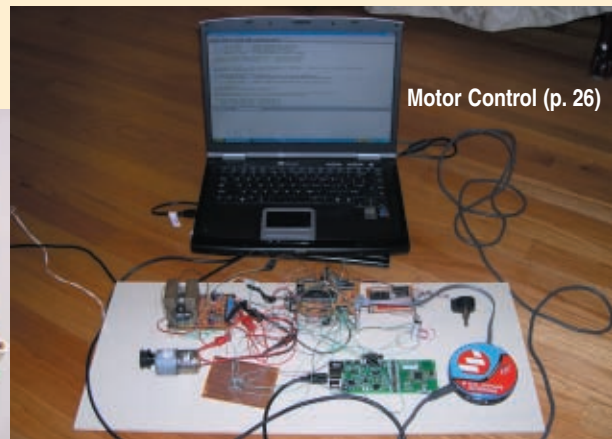
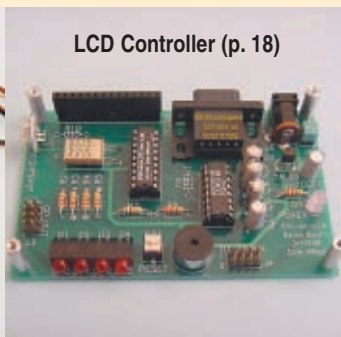
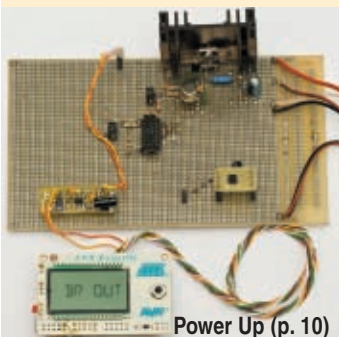
Power your designs with NKK.
Visit us online to:

- **ORDER** Dev Kits to drive your creativity
- **DOWNLOAD** software to expedite your design process
- **PROTOTYPE** and test new design ideas cost effectively

September 2007: Data Acquisition

FEATURES

- 10 **Smart Power**
An Intelligent Power Supply for Embedded Systems
Alexander Popov & Jordan Popov
- 18 **PIC-an-LCD**
A Character-Based Serial LCD Controller
Dale Wheat
- 26 **High-Performance Motor Controller**
Daniel Ramirez
- 42 **1-Wire in the Real World (Part 2)**
The Solutions
Steve Hendrix
- 52 **Resilience in Embedded Designs (Part 1)**
Power Supply, Inputs, and Ground
Aubrey Kagan
- 58 **Pulse Generation**
Encoder Interfacing to Microcontrollers
Stuart Ball
- 62 **Embedded Scripting**
Thiadmer Riemersma



COLUMNS

- 70 **LESSONS FROM THE TRENCHES**
String Theory
George Martin
- 74 **FROM THE BENCH**
I-Zip Dashboard
Jeff Bachiochi
- 80 **SILICON UPDATE**
Game On
Tom Cantrell



DEPARTMENTS

- 4 **TASK MANAGER**
Blogs & Video
C.J. Abate
- 8 **NEW PRODUCT NEWS**
edited by *John Gorsky*
- 93 **CROSSWORD**
- 94 **INDEX OF ADVERTISERS**
October Preview
- 96 **PRIORITY INTERRUPT**
Chronic Subscription Overdose
Steve Ciarcia



complete solution for

Remote Access Control



Atmel® provides a complete, secure RAC solution based on AES, the Advanced Encryption Standard. Customize your own design with our RAC development tools and free application note.

RAC system

RAC stands for Remote Access Control and is the remote control of various applications. It is often referred to as Remote Keyless Entry and Remote Wireless Entry. A typical Remote Access Control system consists of one receiver and one or more transmitter units. Since parts of these systems are portable they require battery operation. RAC systems operate on licence free ISM bands, and security and power consumption are the two most distinct challenges.

picoPower™ technology

- Off the shelf high performance AVR® microcontrollers
- Increased battery life time
- Reduced need for external components
- Full flexibility with only one microcontroller

Security

- Based on the proven AES algorithm
- Encrypted transmissions
- Unique transmitter IDs
- Blocks unauthorized access
- Defends reply or prediction attacks

Atmel's RAC solution – your solution!

Our high performance AVR microcontrollers with picoPower™ technology reduce the need for expensive components in your design. By using AES for encryption you don't have to think about licences, patents or other hidden costs. Even the code is available for free, just like AVR Studio®, the free development environment.

With our proven high quality tools, Atmel has everything you need for easy, customizable RAC development.



Find more info at: www.atmel.com/AVRman



USB MIXED-SIGNAL SCOPE AND WAVEFORM GENERATOR

The BS100U is the newest member of BitScope's popular family of PC-based mixed-signal oscilloscopes. Like all BitScopes, it has an analog input bandwidth of 100 MHz and supports real-time simultaneous analog and logic capture up to 40 Msps. Unique to the BS100U is its optoisolation, decoupling it from the PC. You can ground reference it independently. USB drop-outs due to ground loops or glitches, when looking at high-power electrical or automotive systems, are never a problem. Another feature to make its debut in the BS100U is a powerful DSP-based, flash-memory-programmable waveform generator. Operating independently of the scope's capture engine, it enables complex waveforms to be synthesized con-

currently with waveform capture. The BS100U has four inputs feeding two analog channels plus eight concurrent logic channels, ± 5 V of adjustable external trigger input, a calibration output, and low-power modes for extended use on battery power in the field.

The premium BitScope DSO software package is included with the BS100U for a complete set of integrated virtual instruments on Windows or Linux PCs. Standard functions include mixed-signal and digital storage scopes, a logic analyzer, a baseband spectrum analyzer, an X-Y phase plotter, and an integrated data recorder. With the BS100U, the DSO introduces 2-Gsps equivalent time sampling with phase coherent full-speed dual-channel capture for HF eye diagrams, ISI and modulation analysis, a multiband spectrum analyzer for RF and narrow-band signal analysis, and sophisticated transfer function analysis applications using the built-in waveform generator.

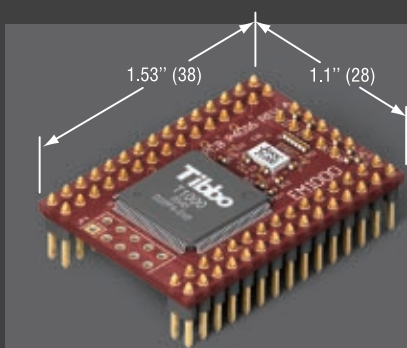
The BS100U costs \$495.

BitScope Designs
www.bitscope.com



Tibbo
TECHNOLOGY

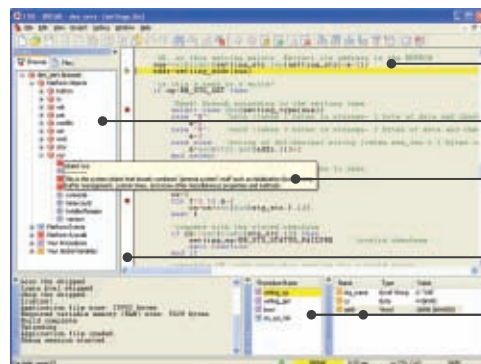
Build your next automation project around our EM1000 BASIC-programmable Embedded Module



- 50 MIPS CPU
- 100BaseT Ethernet port
- 512K flash disk
- 4x high-speed UARTs
- High-speed parallel slave port
- Real-time clock with backup power
- 49x general-purpose I/O lines
- Development kit available (EM1000-SK)

- Programmable – in BASIC!
- Optimized for real-time applications
- Rich object set
- Built-in webserver
- Event-driven operation
- Sophisticated development environment supports cross-debugging (no ICE needed)

Code and debug your Tibbo BASIC application using Tibbo Integrated Development Environment (TIDE) software



- Write in familiar BASIC language
- Inspect objects, procedures, and variables
- Code faster with auto-completion and code hints
- Set breakpoints, execute step-by-step, etc.
- Monitor the state of variables and stack

web: www.tibbo.com email: sales@tibbo.com

NEW PRODUCT NEWS

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

14-BIT, 25-MSPS ADC FOR HIGH-TEMPERATURE APPLICATIONS

The LTC2246H is a 25-MspS, 14-bit ADC for high-temperature, high-reliability data acquisition systems. The LTC2246H, guaranteed for operation across the -40° to 125°C temperature range, satisfies the high temperature demands of automotive and military applications. Additionally, the LTC2246H comes in a leaded package, enabling easy inspection of solder joints in manufacturing. This high-speed ADC is targeted for collision avoidance radar automotive systems.

The LTC2246H, which consumes only 75 mW, provides an excellent 74.5-dB signal-to-noise ratio and 90-dBc spurious free dynamic range baseband performance. The LTC2246H is packaged in a small 5 mm \times 5 mm TQFP package with integrated bypass capacitors requiring only a small number of external components for more compact, cost-effective designs.

For lower resolution requirements, Linear Technology is also offering the pin-compatible, 12-bit, 25-MspS LTC2226H. The H-grade (-40° to 125°C) LTC2226H and LTC2246H are \$10.80 and \$18, respectively, each in 1,000-piece quantities.

Linear Technology Corp.
www.linear.com



Reliable, Easy-to-use Embedded RF Modules

and the **Wireless Expertise** to connect the pieces of your data system

Featured RF Protocols	Range (power output)	RF Data Rate
Self-powered	up to 150' (5 mW)	120,000 bps
ZigBee	up to 1 mile (100 mW)	250,000 bps
Bluetooth	up to 300' (100 mW)	3 Mbps
WiFi 802.11	up to 1,000' (1 Watt)	54 Mbps
900 MHz ISM License-free	up to 40 miles (1 Watt)	9,600 - 115,200 bps

RoHS compliant

30+ Years of RF Experience
Call & find which RF solution best fits your application

AD HOC ELECTRONICS

AdHoc101.com

Reliable OEM RF Modules + Custom Engineering (801) 225-2226

CIRCUIT BOARDS

Manufactured Domestically Since 1984

As low as...

\$9.95

each!

Two Boards
Two Layers
Two Masks
One Legend

Unmasked boards for even less!

e-pcb.com



Smart Power

An Intelligent Power Supply for Embedded Systems

This “intelligent” power supply was designed specifically for testing embedded systems. In addition to its numerous useful features, the ATmega169-based power supply can provide any voltage from 0 to 5 V with 10-bit resolution.

As digital integrated circuits evolve, their power supply requirements also change. When I started experimenting with digital ICs, the 74 TTL family was popular and easy to acquire. All the boards were powered by a single 5-V power supply. I made my first step into the embedded world with an i8031 microcontroller with the inevitable '373 latch for addresses, external EPROM, SRAM, and a MAX232. The system was powered by a common 5-V power supply. The power supply was not a component that required attention—the same design was applicable for most digital designs.

As technologies evolve, transistors inside ICs become smaller and require lower voltages. The major focus for an embedded system moves from speed to lower power consumption and more battery-powered devices are developed. To efficiently use the capacity of a battery, a microcontroller must use the same voltage or about the same voltage provided by a battery. That is why there are a variety of voltage requirements for embedded microcontrollers (from below 1 V to more than 5 V). An embedded system developer can no longer rely only on the good old 5-V power supply to feed the new chips.

As an embedded systems developer, I always need a good power supply for my embedded projects. The Atmel AVR Design Contest 2006 was the perfect opportunity to shoot two rabbits with one bullet. I decided to participate in the contest, which I always enjoy, and build a useful power supply. This article gives me the chance to present the design to the embedded community.

My first idea was to build a digitally

controlled power supply that could provide voltages down to 0 V to feed most single-supply embedded microcontrollers. But my thoughts changed after playing with the Atmel AVR Butterfly module that was shipped to the contestants. It offers a nice user interface and menu system, and it has enough resources to build a more complicated device. After thinking things over, I came to a few conclusions.

First, like every good power supply, I wanted mine to have short-circuit and overcurrent protection. This is essential for safety and reliability. I wanted the protection to be self-resetting because it can be annoying to replace fuses. Second, I wanted the display to be used to show the voltage and the current or power consumed. By monitoring the current drawn and power consumed, I knew I could directly measure the power efficiency of the system under test. Third, I wanted the device to be able to test the target system for sensitivity to major power supply problems, such as a brown-out, slow-rising power, periodical voltage drops, and noise on the power lines.

With the help of my father Jordan

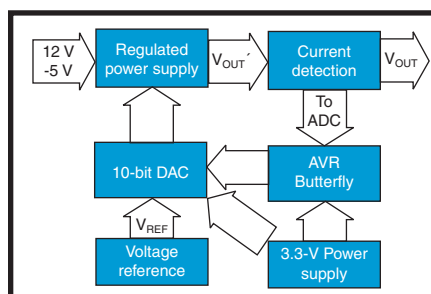


Figure 1—As you can see, the power supply is pretty simple thanks to the Butterfly module.

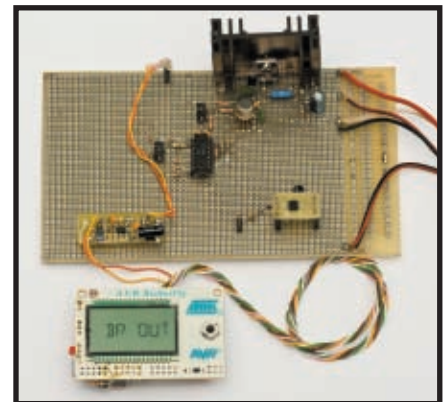


Photo 1—The prototype of the intelligent power supply did a great job proving the concept, testing the software, and evaluating the results. The finished product—which has a more compact PCB design, a 110/220-V PSU and cord, and a proper (grounded) case—features the Butterfly on its front panel.

(especially with respect to the analog part of the circuit), I built an intelligent power supply. The working prototype is shown in Photo 1. You’ll find it easy to build a similar system. The system’s main components are depicted in Figure 1.

HARDWARE

The system is powered by two voltages. The main voltage is 12 V, and it must provide enough current for the target load, the Butterfly (through a 3.3-V linear regulator), and the other components. The second voltage, -5 V, is used as a negative supply voltage to the op-amps.

The main block is the regulated power supply, which was built with a linear regulator. A 10-bit DAC controls the voltage. The output of this block flows through the current-detection block and then to the output connectors of the power supply.

As you can see in Figure 2, the system features a National Semiconductor LM723 voltage regulator. It has a temperature-stabilized, low-noise voltage reference. In addition to short circuit protection, it can provide output voltages down to 0 V.

The LM723 requires special handling for low output voltages. For output voltages greater than 2–3 V, the V₋ pin can be connected directly to ground. But for voltages down to 0 V (or even further), the V₋ pin should be connected to a negative voltage of at least -0.4 V. There are several ways to produce this voltage. One method is to convert positive to negative voltage with a switching capacitive inverter. But note that this method can introduce noise. The LM723's voltage reference is relative to the V₋ voltage. That is why it is important that V₋ be stable and noise free. Thus, another method is used (see Figure 3). V_{REF} = 1.28 V is produced by U2A, R19, R5, and R6 from the LM723's reference voltage. This is inverted to VM256 = -2.56 V by the op-amp U2B, R1, and R2. This also works as negative feedback to V_{REF}, partially compensating for its temperature coefficient and stabilizing

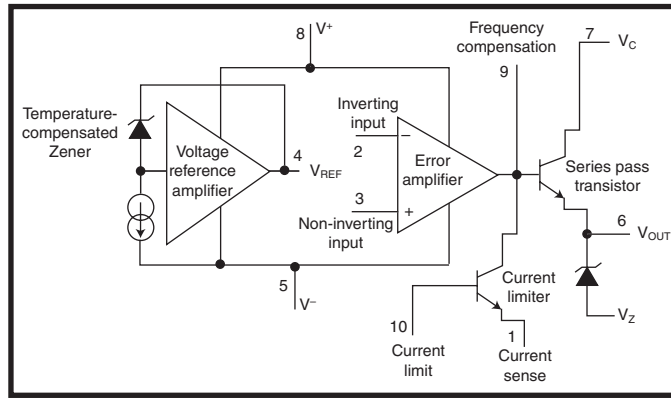


Figure 2—The LM723 linear voltage regulator is a reliable component. It contains the major analog blocks needed to build a stable, regulated power supply with current protection.

the voltage reference even more.

Microchip Technology's TC1321 DAC (U3) is connected to the LM723's IN₊ pin to set the output voltage. The TC1321 was chosen for its 10 bits of resolution, 2.7–5.5 V single-supply operation, good integral and differential linearity, and an output voltage offset of less than 8 mV. The DAC is controlled by the CPU inside the Butterfly via an I²C interface.

The reference voltage for the DAC is V_{REF} = 1.28 V. It's produced from the reference voltage of U4 in the way described above.

The DAC's output is filtered through a simple low-pass filter built with R7 and C5. Its purpose is to smooth the output voltage and filter out the sam-

pling frequency on the DAC's output.

Many electronic devices can't withstand a small reverse voltage in the power supply. That's why a voltage-offset correction is provided (R20, R9, R10, R18, and U2D) in the voltage-feedback circuit. This eliminates the possibility of a negative output voltage during startup (when the DAC is zeroed). There are two reasons why this offset is not eliminated in the soft-

ware with a constant added to the number written in the DAC. First, the offset can be positive, and there may be a need for a negative constant. This won't work because the DAC works only with unsigned values. As for the second reason, note that in software, the correction is discrete and there can be an offset of up to one-half of it left.

The diode D1 provides additional safety to the powered electronic circuits by not letting the output voltage become lower than -0.7 V. On the power supply's output, there is the usual capacitor (C7) with the unusual value of only 1 μF! You would typically expect something like 1,000 μF here, but had this been used, the output response would have been slowed down and no high-speed control from the CPU would have been possible. A small capacitor is still needed to prevent the circuit from self-oscillating.

The transistor Q1 is used to provide more output current than the LM723 can source. The power supply is linear and it converts voltages, but since the current is the same for input and output, Q1 has to dissipate the excessive power (i.e., $P = (V_{IN} - V_{OUT}) \times I_{OUT}$). That is why it should be mounted on a large heatsink. Its area is calculated for the worst-case scenario (i.e., $V_{OUT} = 0 \text{ V}$, $I_{OUT} = I_{MAX}$, $\theta_{AMB} = \theta_{MAX}$). If the dissipated power is less than 5 W, a 65-to-90-W transistor in a TO220 case can be used. When the power is between 10 and 15 W, a Darlington transistor in a TO247 case rated for more than 100 W should be used. The area needed for the heatsink can be seen in Table 1.

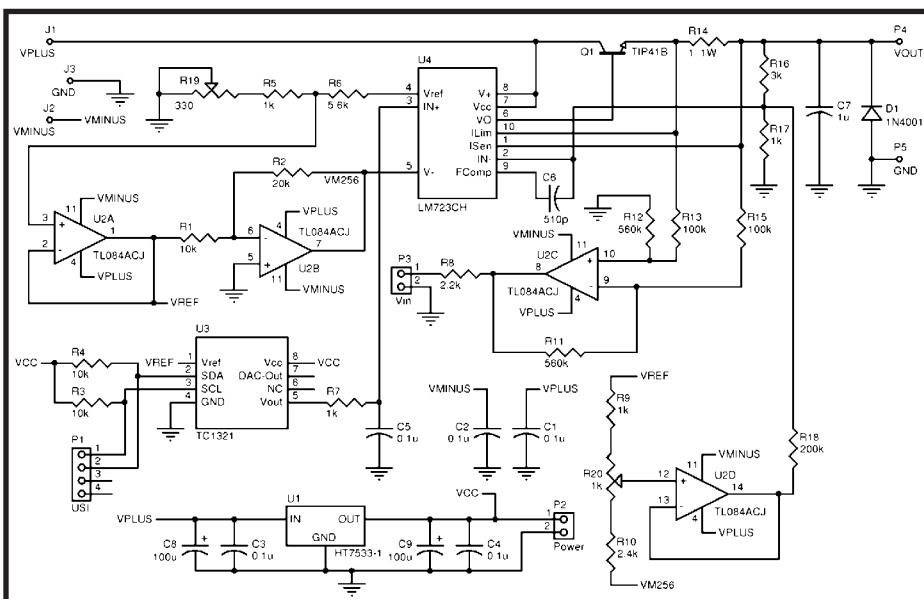


Figure 3—The schematic of the intelligent power supply shows that no exotic components are used. The device can be easily reproduced by almost any embedded enthusiast. The Butterfly shouldn't be a problem to obtain. The connections to it are via headers P1 (USI), P2 (power), and P3 (V_{IN}).

$P_C = V_{PLUS} \times I_{MAX}$ W	2	5	10	15	
$\theta_{MAX} = 45^\circ\text{C}$	S, cm ²	35	100	220	400
$\theta_{MAX} = 113^\circ\text{F}$	S, inch ²	5.4	16	34	62
$\theta_{MAX} = 60^\circ\text{C}$	S, cm ²	50	125	300	600
$\theta_{MAX} = 140^\circ\text{F}$	S, inch ²	7.8	20	47	93

Table 1—The area of the heatsink required for transistor Q1 depends on the maximum current drawn, maximum ambient temperature expected, and V_{PLUS} . The casing of the device can also be used as a heatsink if it's made of metal.

The resistor R14 has two functions. It sets the LM723's current threshold. The protection is triggered when the voltage across R14 reaches 0.65 V. It is also used to measure the output current. The common-mode output voltage is suppressed by the differential amplifier U2C and only the voltage drop across the resistor R14 is amplified. The voltage is proportional to the output current in the ratio 1 V/1 A. The voltage is amplified to about 5.6 V per amp to fit the range of the Butterfly's V_{IN} measurement circuit, which has a 6:1 divider in the input. Any offset is eliminated in the software because it is not as critical as the output voltage.

The intelligent part of the device is the Butterfly. It has a good user interface with a five-way joystick and a six-character alphanumeric LCD. Many other peripherals are connected to the micro-controller, such as a piezo speaker, DataFlash, an NTC thermistor, a light sensor, and an RS-232 lever shifter. All of the peripherals can be used to extend the device's functionality in the future. Currently, the LCD, joystick, JTAG, ADC, USI (for the I²C driving the DAC), and one red LED are used. The LED was not originally on the board, but it was easy to add. Its cathode was soldered to pin 3 on the USI header, which is unused in the I²C communication, and its anode was connected to the positive power of the Butterfly board through a 220-Ω resistor. The LED illuminates when the current protection is triggered.

The DAC (U3) and the Butterfly module are powered by a linear power supply that uses Holtek Semiconductor's HT7533-1 voltage regulator. This low-dropout regulator is reliable and has no risk of oscillation or high-voltage output in low-current mode, which is sometimes seen in low-dropout regulators.

The absence of hum and noise is

important for the accuracy of this power supply. All the power grounds should be routed with different thick tracks to a single point on the board. The signal grounds should also be connected to this point.

SOFTWARE

Martin Thomas's GCC port of the demo code for the Butterfly is used as a base for the software (see Figure 4). Atmel's driver for the USI has been ported to GCC and software timeouts have been introduced to eliminate freezing if errors occur on the I²C bus.

The CPU frequency is changed to 8 MHz because floating-point arithmetic is used. In addition, all the power-saving features are removed because they are not needed in a power supply project.

The ATmega169V's ADC is configured to generate an interrupt on every conversion. Current protection is a priority over any other process in the soft-

ware. ADC measurements are averaged in packets of eight to eliminate spikes and errors that could lead to false-over-current detection. The constants for matching the ADC code to the current were calculated after measuring several points of the characteristic and it proved to be linear with just a small offset.

All the test cases are generated through a programmable linear interpolating engine with as many points as needed. Every point consists of a time interval and a voltage to be reached at the end of the interval. This way, many forms can be generated, the most useful of which are already programmed: brown-out, slow-rise, slow-fall, etc. The points and parameters for each program may be easily configured with several arrays in the program memory.

CALIBRATION

The power supply needs to be cali-

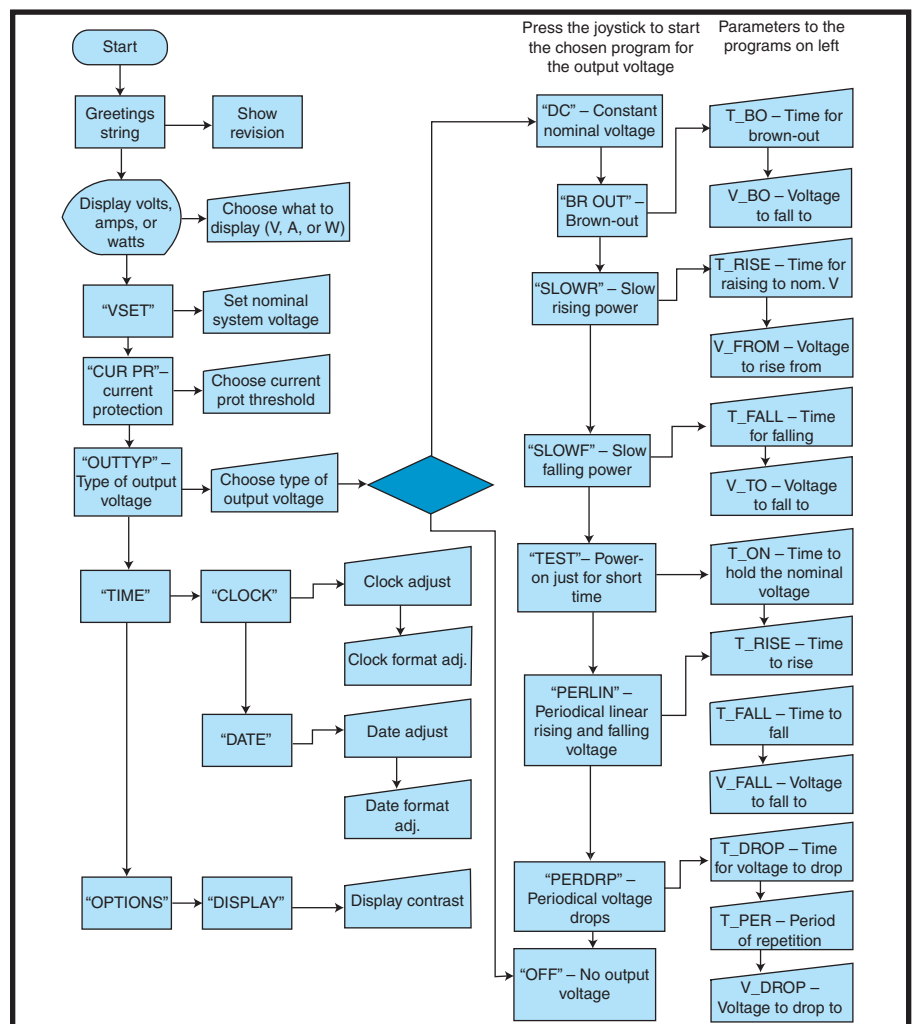


Figure 4—The power supply can be fully configured and used through the Butterfly's comfortable user interface.

WIZnet iEthernet

2007

DESIGN CONTEST

**CIRCUIT
CELLAR®**

THE MAGAZINE FOR COMPUTER APPLICATIONS

Put your creativity to the test!

(Sep. 20 ~ Jan. 31)

Join the Ethernet Revolution!

Circuit Cellar magazine is pleased to bring you the WIZnet iEthernet Design Contest 2007. Now you can easily add Ethernet capability to your embedded project and win fame and fortune in the process. WIZnet's W5100 hardwired TCP/IP

Ethernet controller will be the key to your contest success. This 3-in-1 chip solution brings TCP/IP implementation without an OS. Both MAC and PHY are embedded.

Show us how you use the impressive W5100 to usher your embedded project into the Ethernet revolution. Your creativity and design skills could win you a share of \$15,000 in cash prizes and recognition in Circuit Cellar magazine.

**For details, visit
www.circuitcellar.com/wiznet.**

WIZnet

www.wiznet.co.kr
www.ewiznet.com

brated to be as accurate and useful as a test device. Calibration involves hardware and software, and it requires a few cycles of recompiling and reprogramming the Butterfly. This is needed only once after assembly. The voltage reference is stable enough in both temperature and time, and no drift is expected.

Don't connect the Butterfly before checking if everything is properly wired and behaving normally in the rest of the circuit. The first step after checking the circuit is to power it on and measure the voltages VPLUS (12 V), VMINUS (-5 V), VCC (3.3 V), VREF (1.28 V), and VM256 (-2.56 V). The values may be somewhere near these. There is no need to be exact. Next, measure the reference voltage VREF and adjust it to exactly 1.28 V with the trimmer R19. After turning off the power, connect the Butterfly with the three connectors to the other part of the schematic. Then turn on the power.

Next, you must set the default values in the corresponding files for a few constants. In the file DAC.h :

```
#define V2CODECONST (200.0)
```

In the file ADC.h :

```
#define CODE2CURCONST (2.5e-3)
/* current in amperes */
#define CURCODEOFFSET 0
```

After compiling the code and flashing the Butterfly, the output voltage should be set to zero via the menu system. With the trimmer R20, the output voltage should be adjusted to 0 V.

To calculate V2CODECONST (the code value for $V_{OUT} = 1$ V), enter some voltage V_{SET} (e.g., 4 V). Measure V_{OUT}

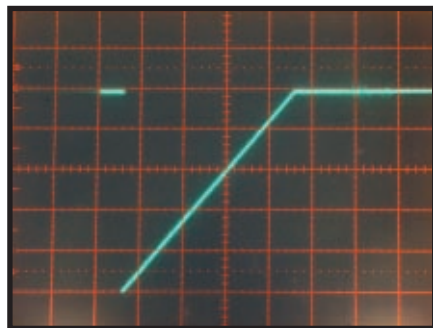


Photo 3—Slowly rising voltage can be real trouble for circuits without low-voltage detection. This test should always be performed.

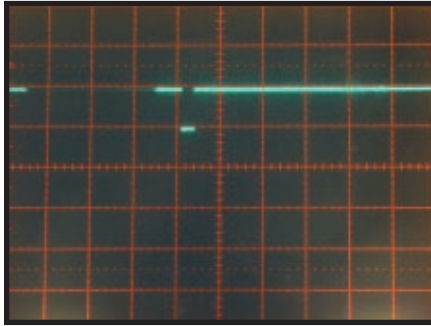


Photo 2—Brown-out test: How long and how deep of a voltage drop could the circuit sustain without resetting itself?

and calculate the real constant:

$$V2CODECONST = V2CODECONST_{DEFAULT} \times \frac{V_{SET}}{V_{OUT}} \quad [1]$$

With an open circuit for the output (zero current consumption), let the device display the current and write it down (I_{ODISP}). Calculate the CURCODEOFFSET as follows:

$$CURCODEOFFSET = \frac{I_{ODISP}}{CODE2CURCONST_{DEFAULT}} \quad [2]$$

The next step is to program the Butterfly with the new constants. To calculate the CODE2CURCONST value, set a predefined voltage V_{SET} (e.g., 5 V) and connect a known accurate resistor R_{LOAD} with enough power and with a value of about 47Ω (to sink about 100 mA of current). Check the displayed value on the LCD (I_{DISP}). The constant can be calculated as follows:

$$CODE2CURCONST = \frac{I_{LOAD}}{code} = \frac{\left(\frac{V_{SET}}{R_{LOAD}} \right)}{\left(\frac{I_{DISP}}{CODE2CURCONST_{DEFAULT}} \right)} \quad [3]$$

Change this constant in the ADC.h file and recompile the program, then flash the AVR. The calibration is done!

VOLTAGE PRESETS

Now that the power supply has been brought to life, it's time to see its real strength, the capability of voltage presets to simulate different problems in the power supply to test the behavior of the system being powered.

The first preset is used to supply a constant voltage to the connected

device. Despite being the simplest of all presets, it does have the features common to all of them: regulated voltage down to 0 V, programmable current protection, and self-restoring short-circuit protection. You can observe the output current and power on the LCD. These features distinguish the power supply from the common solution of the embedded systems enthusiast, a simple linear voltage regulator powered from a mains-connected adapter.

The real value that the power supply offers is its ability to simulate the undesirable conditions of the embedded system's voltage supply.

BROWN-OUT

One of the most common problems every embedded system has to deal with is short-voltage drop (brown-outs), which is usually caused by commutating large loads to the mains power lines (see Photo 2). Almost every microcontroller has a built-in brown-out detection circuit that provides an internal reset when the supply voltage drops below a certain threshold. Often, there is a need for a system-wide reset, which is where stand-alone brown-out detection ICs are used. The preset is configured with two parameters: T_{BO} (the duration) and V_{BO} (the voltage to fall to during brown-out condition).

SLOWLY RISING VOLTAGE

The condition usually happens while the power supply is powered up (see Photo 3). It can be real trouble because integrated circuits start to operate on lower voltages than specified, but their behavior is unpredictable and unstable. One of the scenarios is that a self-programming microcontroller (every one having a

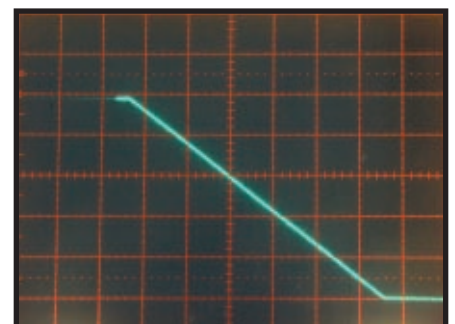


Photo 4—This is a test for the behavior of the circuit after the power supply is switched off.



M16C- The broadest platform with true code and pin compatibility

42 to 144pin, 24K to 1M Byte Flash & 1K to 48K Byte RAM

Renesas Technology

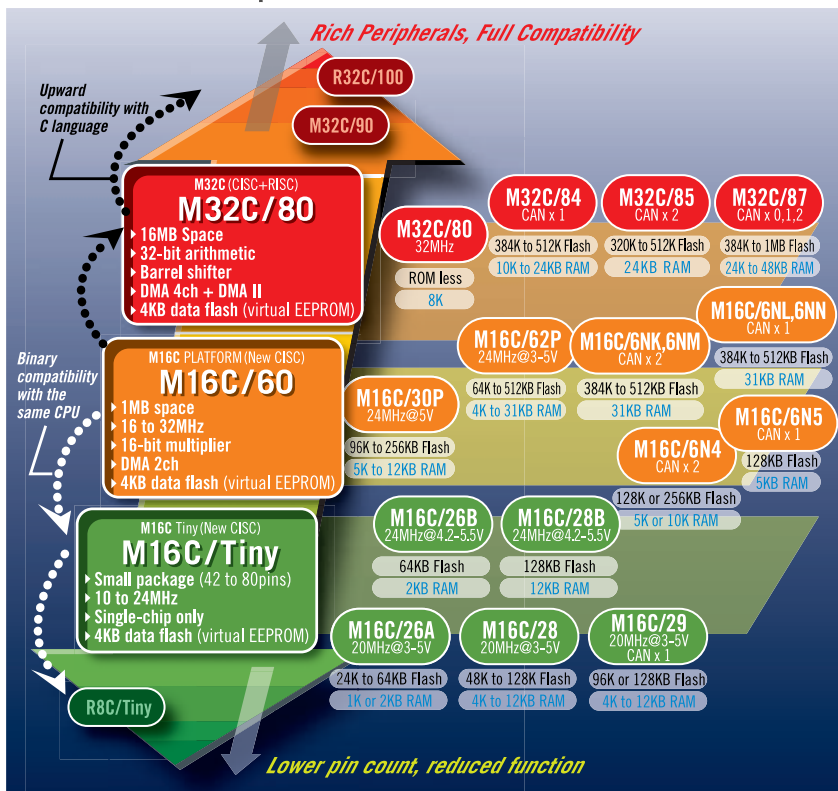
No.1* supplier of microcontrollers in the world

presents a wide range of M16C microcontrollers. M16C is the only fully code-compatible platform in the industry that addresses the entire 8-bit through 32-bit price/ performance application space.

Your application can range between 24K Bytes and 1M Bytes of code size, with between 42pins and 144pins of package size, while keeping the same code base and development tools.

The consistency and compatibility of the M16C Platform enables you to reduce your development time while still allowing the flexibility to adapt to changing system requirements.

M16C Product Roadmap



*Source: Gartner Dataquest (April 2006) "2005 Worldwide Microcontroller Vendor Revenue" GJ06333

HOT Products M16C/29 Group

M16C/29 Block Diagram

M16C/60 CPU Core 20MHz	PLL	TimerA 5ch
	On-Chip Osc	TimerA 3ch
	RTC 32KHz	Three-phase Motor Cont.
Multiplier	WDT	
Interrupt	CRC	ROM Correction
RAM Max 12KB	10-bit A/D Max 27ch	SIO/UART 3ch
ROM Max 128KB MASK/Flash with Protection	IC/OC timer 8ch	SIO 2ch
	LVD	Multimaster I ² C
	DMAC	CAN2.0B 1ch

Top Reasons To Select M16C

- Broad Platform** - Wide range of selection; 24KB to 1MB on-board Flash, 42pin to 144pin
- Compatibility** - Pin and Code compatibility across the platform enables you to upgrade/downgrade MCU in same package without changing the board design and peripherals.
- Powerful** - High speed interrupts (M32C: 0.61u sec in 32MHz), optimized instructions for 1 cycle operations, Hi-speed hardware multiplier.
- Versatile** - Specialized on-chip peripherals; CAN, LIN, 3ph-PWM, DMA, etc
- Efficient** - Multiple clock sources and multiple power saving modes
- Quiet** - Built-in noise cancellation circuits provides excellent EM/EMS characteristics that outperform IEEE standard spec
- Easy** - Same Tool Chain for evaluation and development across the board
- Reliable and secured** - Trusted flash and built-in fail safe features such as oscillation-stop detection circuit, protect registers, enhanced WDT, etc.



Get Started Today -

Go online and register to be eligible for a FREE Starter Kit
www.america.renesas.com/ReachM16C/d



Visit us at ESC Boston!
Booth #1209

Embedded Systems CONFERENCE BOSTON

Renesas Technology Corp.

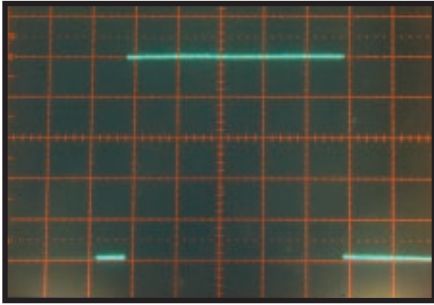


Photo 5—This is a test for smell and smoke from the circuit. With this power supply, it should not go that badly due to the short time and the current protection, but some components can get warm.

bootloader) incorrectly executes instructions and messes up its non-volatile or flash memory.

The preset is configured with two parameters: T_RISE , the time for rising to normal voltage, and V_FROM , the voltage to rise from.

SLOWLY FALLING VOLTAGE

Regardless of the power supply circuit used in an embedded system, when the power turns off, the output filter capacitor discharges through the circuit and the voltage slowly falls (see Photo 4).

This is more common than the rising voltage condition during power-on. The effects to the circuit are similar and the measures taken to eliminate the risks during power-on should work here too. Nevertheless, it's always better to test, not to assume, so this preset will be helpful to see how the circuit behaves after it is switched off.

The parameters are: T_FALL , the time for falling from default voltage to V_TO , voltage level.

SHORT TIME TEST

The preset can be used for testing a circuit for overrated current consumption (see Photo 5). The power is supplied to the device for only a short time. If there is a problem due to a short circuit or miswiring, the current protection will trigger. The preset is also useful for discovering problems leading to high current consumption but are not short circuits. One common problem is connecting the electrolytic capacitor in a wrong polarity. This does not always trigger the current protection, and after some seconds, the capacitor explodes! That is why the circuit

should be powered for a short time and the current consumed should be observed.

There is just one parameter for this preset: the time the power is turned on.

PERIODICAL LINEAR VOLTAGE

The preset can be used for different purposes. With a small period, such as the one in Photo 6, the tested circuit input capacitors can be evaluated. Do they provide enough filtering to the hum from the mains power net or to the other periodical disturbances? Since the waveform of the voltage is triangular, not sinusoidal, the introduced hum has more harmonics, providing a better test to the filtering part of the circuit than a pure sinusoidal signal.

Another application of the preset is testing the linear voltage regulator's behavior in the entire operating range of input voltages. Some LDOs may become unstable under some conditions (input voltage and load) or stop stabilizing the input voltage, feeding it to the output! This can destroy all the integrated circuits powered by the LDO. That is why a test for voltage stability should be performed. The preset is useful for doing it.

Since it is a more complex waveform than the previous presets, this one is configured with more parameters: T_RISE , the duration of the voltage rising, T_FALL , the duration of the voltage falling, and V_FALL , the voltage to fall to.

PERIODICAL VOLTAGE DROPS

The waveform generated by this preset has even more harmonics than the periodical linear one (see Photo 7). Thus, it is a test for the power supply filtering of the entire circuit, especially useful for noise-sensitive audio amplifiers, such as those for microphones or guitars.

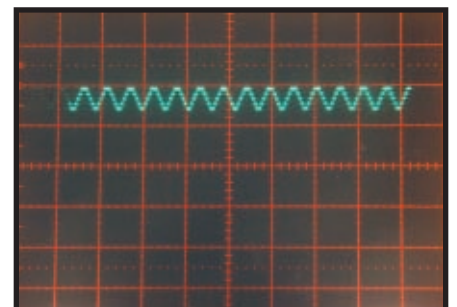


Photo 6—Periodical linear voltage can be used to test the filtering efficiency of the input capacitors.

We Listen. Think. And Create.

Distributed I/O

Digital I/O

Serial I/O

Industrial Computing

HMI

SeaLINK Ethernet serial servers are the fastest, most reliable way to connect serial devices to your network.



SeaLINK Ethernet Serial Servers Offer:

- 1, 2, 4, 8, and 16-Port Models
- RS-232, RS-422, RS-485, and Optically Isolated Versions
- Included Software Enables Virtual COM Port Operation
- Easy Installation and Configuration
- DIN Rail or Table Mount Design
- Extended Temperature Option Available

FOCUS
On Success
Call Today!

SEALEVEL

sealevel.com > sales@sealevel.com > 864.843.4343

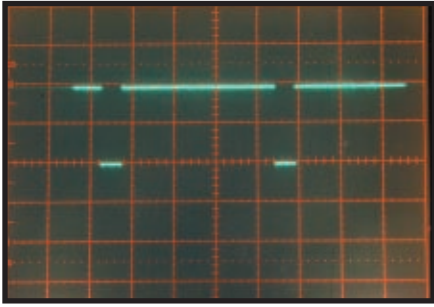


Photo 7—This is a test for the power supply filtering of the entire circuit. It is especially useful for noise-sensitive audio amplifiers, such as those used for microphones and guitars.

A popular module in embedded systems is a GSM/GPRS modem, sometimes used for voice communication. Due to the time-division multiplexing in the GSM network, the modem drains current unequally, but periodically in time. The frequency is about 216 Hz and can be very annoying. It is heard when a GSM phone is placed near an FM radio tuner or active PC speakers. The peak current consumed by the modem is about 2 A, which causes voltage drops in the traces, thus introducing noise across power supply lines. Noise is easily transferred to the sensitive audio circuits if they are not routed properly. To test how the circuit reacts to this, the periodical voltage drops preset can be used, configuring it with parameters to simulate an active GSM modem. By altering the voltage drop, the power supply rejection ratio of the audio amplifiers can be evaluated.

The preset is configured with three parameters: T_DROP, the voltage drop duration, T_PER, the period of repetition, and V_DROP, the voltage to drop to.

FUTURE DEVELOPMENT

You'll find this power supply useful, especially if you put it in a case and include a nice front panel. If you need additional features, keep in mind that the feature-rich Butterfly provides many possibilities for expanding the device. You can easily add the features you need.

One useful expansion is to use RS-232 communication to the PC for configuring and displaying results of measurement. You can send a snapshot of the current drawn during startup via the RS-232 for display and analysis. You can also use the clock and data functions from the original AVR Butterfly demonstration code.

The calibration process can be made

easier, avoiding reprogramming the device several times, by storing the constants in E²PROM and providing a menu system for their adjustment. The EEPROM can also be used to store the last used configurations of the presets. ■

Alexander Popov (sasho@popovbrothers.com) has an M.E. in Communication Technologies from the Technical University of Sofia, Bulgaria, and a Specialist's degree from ELSYS high school. He is currently an embedded systems developer at SmartCom, Bulgaria, working as a subcontractor for IBM.

Jordan Popov holds an M.E. in Computer Systems from the Technical University of Sofia. He has more than 20 inventions to his name. Jordan runs a small business that develops and manufactures custom electronic products. You may reach him at jordan@popovbrothers.com.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2007/206.

RESOURCES

Microchip Technology, "TC1321: 10-Bit Digital-to-Analog Converter With Two-Wire Interface," DS21387B, 2002.

National Semiconductor, "LM723/723C Voltage Regulator," DS008563, 1999.

M. Thomas, "AVR-Projects," 2007, www.siwawi.arubi.uni-kl.de/avr_projects.

SOURCES

ATmega169V and Butterfly

Atmel Corp.
www.atmel.com

HT7533-1 Voltage regulator

Holtek Semiconductor, Inc.
www.holtek.com

TC1321 DAC

Microchip Technology, Inc.
www.microchip.com

LM723 Voltage regulator

National Semiconductor Corp.
www.national.com

GCC Compiler

WinAVR
<http://winavr.sourceforge.net>

Intelligent Display Engine

IntelliLCD

- 7" Wide TFT Color Touch Screen
- Supports TrueType Windows Fonts
- RS232 Interface
- 800 x 480 resolutions
- 260k Colors
- SD Card Support
- BMP File Support
- LED Backlight
- USB PC sync, keyboard, mouse
- Display Commands: Line, Circle, Box, Print..
- Control Commands: Button, Slider, CheckBox...

\$399
qty 1

COMFILE

TECHNOLOGY

www.comfiletech.com
 Toll-Free: 1.888.928.2562

PIC-an-LCD

A Character-Based Serial LCD Controller

Dale describes his useful PIC-an-LCD device, which is a character-based serial LCD controller. Based on a Microchip Technology PIC16C621, the LCD controller chip is programmed in C language and works with most alphanumeric LCD modules. It accepts normal or inverted serial data and connects to either TTL or RS-232-level signals with a single resistor.

The PIC-an-LCD is an LCD controller chip with many useful features (see Photo 1). It is based on the Microchip Technology PIC16C621 microcontroller and programmed in C language. Originally a closed and proprietary design, I now place it, and all of its supporting documentation, in the public domain. In this article, I'll describe the inner workings of the device and give some background on its development.

HISTORY

The PIC-an-LCD serial LCD controller chip was suggested to me by my friend, Billy Gage, of BG Micro. He sells an interesting assortment of new and surplus electronics, including many LCDs based on the Hitachi HD44780 driver chipset. After discovering the joy of controlling LCDs with Microchip PIC microcontrollers, I set about to make a device that was simple to use and low in cost yet had many useful and interesting features. Thus, the PIC-an-LCD was born. Billy gets the credit for the name.

The PIC-an-LCD is now coming up on its tenth anniversary. Thousands of these chips have been sold all over the world, and creative folk are still coming up with fascinating uses for them. The February 1999 issue of *Popular Electronics* magazine ran an article by Carl J. Berquist titled "Liquid Crystal Displays—The Easy Way," as the cover story, with a life-size photo of

the humble PIC-an-LCD chip in the middle of the page.

FEATURES

The PIC-an-LCD works with any alphanumeric LCD module based on the Hitachi HD44780, the Samsung KS0066U LCD controller, or an equivalent driver chipset. It has a TTL serial input that operates at 2,400 or 9,600 bps. The specific communication parameters are 8 data bits, 1 start bit, 1 stop bit, no parity bits, and no hardware or software handshaking. The serial input also has versatile "autopolarity" smarts that enable it to accept normal or inverted serial data. This enables it to connect to either TTL or RS-232-level signals using a single resistor. Four general-purpose outputs and a bell output are also provided. A level shifter and a signal inversion circuit are not required.

The PIC-an-LCD has a large number of software features. Refer to Table 1 to learn more about the PIC-

an-LCD's device pins and their respective functions.

OLD-SCHOOL DEVELOPMENT

Way back in the ye olde twentieth century, embedded development was not the walk in the park that it is today. Before the advent of reprogrammable flash memory-based microcontrollers, the development cycle consisted of writing code, cursing and coaxing until the code compiled, and then looking in the EPROM eraser for a clean part to program. If no parts were ready, you cranked up the timer, started up the ultraviolet lamp, and went for a walk for 10 or 20 minutes. Once toasty parts were available from the oven, you took one and put it in the EPROM programmer and burned your code into the part. Then, you took that part, plugged it into your prototype, tested your new code, and repeated as needed. Once your code was as perfect as humanly possible, you bought a tube (or 20) of one-time programmable (OTP) parts and very carefully blasted them with your code. More cursing, coaxing, and crossed fingers were indicated.

Cursing and coaxing remain, but these days, the part (singular) usually stays in the prototype. But I digress.

SPENDING MONEY

There was also a lot of money changing hands back in the good old days. The windowed EPROM versions of the OTP parts cost \$10 or more

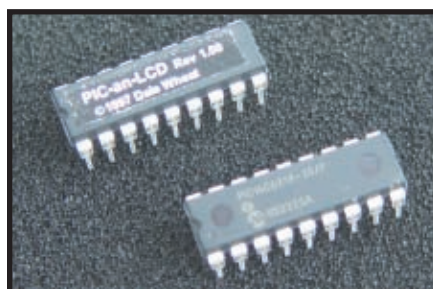


Photo 1—The PIC-an-LCD is a serial LCD controller based on the Microchip PIC16C621 and programmed in C.

each, and you needed at least two or three, so you could be erasing while the other one was programmed or tested. This overlap gave the thoughtful programmer a moment to reflect on the progress being made, which often resulted in the allocation of more funds for more windowed parts. I had four before it was over.

The device programmer was also a good way of ridding yourself of excess fundage. There were prototyping programmers available for under \$100, but a real production programmer started in the \$500 range and just kept going. I used a private-label version of Needham's EMP-20, which plugged into the parallel port and had a DOS driver program. It has since been discontinued and replaced with the EMP-21, which sports a USB connection and a Microsoft Windows interface. I can shed a wee tear about this, but I have to be comforted by the fact that my EMP-20 is still running and programming PIC-an-LCD chips to this day. I also have to tip my hat to the good people at Needham's for their excellent customer service. What company these days has a human being answer the phone on the first ring and has that same person answer your technical question completely and professionally? So, it's a case of money well spent, as I have been able to use the device programmer for lots of other projects. Oh, and don't forget that you need to buy a good EPROM eraser. There are cheap ones available, but they take a lot longer to erase a chip completely, so it's worth it to get a decent one.

MORE MONEY

Let's spend more money: I had taken a quick look at the PIC assembly language and decided against it. I was not "afraid" of assembler because I had learned to program that way eons ago on an obscure National Semiconductor part, then the popular Z80 from Zilog, followed by the IBM PC. The PIC architecture just seemed odd to me, with all the bank switching, strangely named registers, and instruction mnemonics. Perhaps I didn't give it a fair chance. They seem to be doing quite well, at last look. The PIC-an-LCD was my first and my last commercial PIC application, as I subsequently

PIC-an-LCD device pin descriptions		
Pin	Name	Description
1	GP2	General-purpose output 2
2	GP3	General-purpose output 3
3	GP4	General-purpose output 4 / Bell
4	-MCLR	-MCLR (reset)
5	GND	Ground
6	SDI	Serial data in
7	RS	LCD register select (LCD_RS)
8	RW	LCD read/write control (LCD_RW)
9	E	LCD Enable (LCD_E)
10	DB4	LCD Data bit 4 (LSB)
11	DB5	LCD Data bit 5
12	DB6	LCD Data bit 6
13	DB7	LCD Data bit 7 (MSB)
14	VCC	V _{cc} (5 VDC at 10 mA max)
15	OSC2	OSC2 crystal
16	OSC1	OSC1 crystal
17	GP0	General-purpose output 0
18	GP1	General-purpose output 1

Table 1—The PIC-an-LCD chip is housed in an 18-pin plastic DIP package.

was drawn to the Atmel AVR side of town. The point is that I chose to go with a higher-level language for this project, specifically C. At that time there was only one commercially available compiler that was in my price range, and that was the PIC C compiler from Custom Computer Services (CCS). So, there went another \$99 for the version I wanted. But again, I have to admit that it was money well spent because the CCS compiler had many features that made embedded programming relatively painless, some of which I have yet to see in any of the free or open-source compilers available today.

WRITING THE CODE

The firmware developed slowly. The first and most obvious task was to select a chip with just enough, but not too much, capability. Microchip makes a dizzying array of products in the PIC line. This just keeps growing because they seem to be reluctant to ever discontinue a part. I narrowed the field down to the PIC16C62x family, which had the best set of required features at a competitive price. I ordered a few of each of the windowed versions of the '620, '621, and '622 variants of the part. The main difference was the amount of program memory available. I did not have a good feel for how big the PIC-an-LCD firmware would turn out to be, so I wanted to be pre-

pared. Like the story of Goldilocks and the Three Bears, I eventually found one that was just right. It turned out to be the PIC16C621. Microchip has subsequently performed a die-shrink on this part and released it as the PIC16C621A, which works interchangeably in this application, and costs even less.

WHY YES, I SPEAK LCD

The first step in the application design was to get the PIC to talk to the LCD. This was where I found out, for sure, that datasheets lie. Most datasheets from LCD manufacturers simply copy, verbatim, the information provided by the driver chipset manufacturers. Somewhere along the line, some errors were introduced and have been happily and faithfully reproduced to this day. Nevertheless, I was able to get the firmware to talk reliably to the LCD. Indulge me a bit and allow me to go into a little bit of detail about what it takes to interface a microcontroller to a garden-variety LCD module. There are three main areas that require your attention.

STEP 1 – POWER

The first area is to provide the proper voltages to the LCD module. Most alphanumeric LCD modules require a regulated 5-V power supply, but draw no more than 2 or 3 mA. It is odd that when more black pixels are displayed, less current is required. Some newer LCD modules will operate at 3.3 V, but the vast majority of units available in the surplus channel are the 5-V species. One nice thing about this type of LCD is that almost every model has the same pinout. There are a few exceptions, which I will detail shortly. Refer to Figure 1 for a typical LCD layout.

Pin 1 is always ground. It is often marked on the LCD module with a miniscule "1" in the silkscreen layer, usually accompanied by a corresponding "14" at the other end of the connector. If the pin is not actually numbered as such, it will often have a square pad or some other indication that makes it unique among the other pins. On the dual-row connectors, pins 1 and 2 are usually marked. Note that this numbering scheme may seem "backwards" because it supposes a header or connector is installed on the top of

the module, which would very often stand taller than the bezel of the LCD unit itself, and interfere with flush mounting of the LCD in an enclosure.

Pin 2 is always V_{CC} . This is normally 5 V. As mentioned before, the supply current requirement is nominally low and this is one of the most attractive features of the LCD. Stable operation requires a regulated power supply and one that can hold the supply voltage within 5%. If power and ground lines are accidentally swapped, as will be the case if you don't pay close attention to the

true pin numberings on the dual-row connectors, the LCD driver chips will grow quite warm and draw an unreasonable amount of current. Thankfully, most LCDs are not immediately destroyed in this manner, but extended exposure to this reverse condition will eventually cause permanent damage to the unit. Please note that even the most momentary of reversed power to the PIC16C621 will result in instant yet invisible destruction of the device. This trick can be performed by placing the chip in its socket backwards as the

power and ground pins straddle the middle of the part. You have been warned.

The best way to test proper power supply connections is to measure the amount of current flowing in the circuit. Anything more than 5 or 10 mA, except on very large displays, is cause for concern. If less than 1 mA of current is measured, then it is highly likely that a proper connection has not been made to the correct terminals.

Pin 3 is V_{EE} or the voltage used to drive the LCD segments themselves. V_{EE} is adjusted to control the contrast or viewing angle of the display. This is where things get weird. Most LCD modules have a limited temperature range, which is normally 0° to 50°C. That's fine for indoor applications, but in the real world, temperatures sometimes fall below freezing, even here in Texas. To compensate for such temperature extremes, extended temperature fluid is used between the sheets of glass that comprise the LCD module. Extended temperature LCDs can require a much wider range of V_{EE} . I have found that many normal temperature range LCDs work well with V_{EE} at or very near ground, with 0.3 V being the most common. This can be provided by simply connecting pin 3 to ground via a 330- Ω resistor. If this doesn't work, try connecting pin 3 directly to ground. If that doesn't work, you'll need to set up a voltage divider between V_{CC} and ground. This can be either two resistors of a known ratio or a potentiometer in the 20-k Ω range. If this still doesn't work, you'll need a negative voltage supply of up to -8 V. You'll also know that you most likely have an extended temperature range LCD.

Test the voltage to pin 3 by connecting power and ground to pins 2 and 1, respectively. With no other pins connected, the LCD will begin operating but will not be properly initialized, no matter what the datasheet says about it being able to initialize itself. I have yet to see that happen in the wild. What you should see is a line of solid black blocks. If this is a two-line LCD, then only the top line is normally black. If it's a four-line LCD, then the first and third lines will be black. If you see no black blocks, it's almost certain that the contrast is not properly set. Try the various voltages

For 3 \$51 PCBs
FREE Layout Software!
FREE Schematic Software!

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

expresspcb.com

ETHERNET REVOLUTION !!

W5100 is '3 in 1' chip solution
for embedded Internet

= **TCP/IP Core** + **MAC** + **PHY**

3 in 1



Serial-to-Ethernet **WIZ100SR**



- Support serial speed max 230Kbps
- 10/100Mbps Ethernet and Auto-MD
- Provide *Configuration Tool program to control Serral device at your taste
(*With this program, users can config all serial parameters through Ethernet)
- Plug-in type module having pin header (12pins x 2)
- Size : 50mm x 30mm x 12mm (W x L x H)

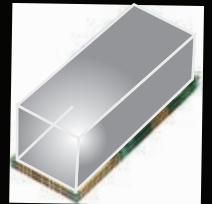


Serial-to-Ethernet **WIZ110SR**

- Support WIZ100SR
- In addition, include the DB9 and RJ45 I/F for user's convenience

All-in-One Network Module **WIZ800MJ**

Coming Soon~



WIZnet 2007
iEthernet
DESIGN CONTEST

Put your creativity to the test!

(Sep. 20 ~ Jan. 31)

**CIRCUIT
CELLAR**
THE MAGAZINE FOR COMPUTER APPLICATIONS

WIZnet

www.wiznet.co.kr
www.ewiznet.com

at pin 3 to get the display looking good and at a comfortable angle from where you are observing it.

STEP 2 – INITIALIZATION

Once you've observed the row of black blocks, and only then, are you ready to move on to item number two, which is device initialization. This is performed by the PIC-an-LCD automatically. It knows the magic numbers and the proper sequence and pacing to send them, which differ slightly from any of the

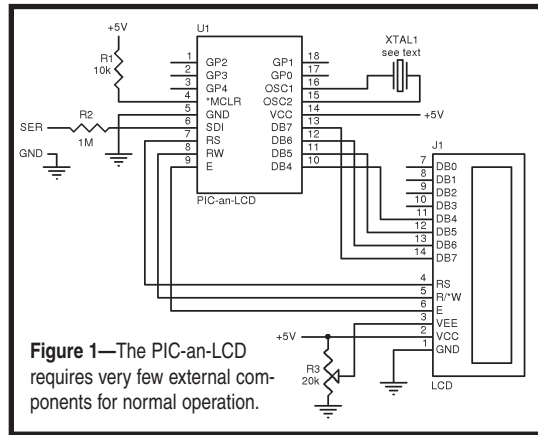


Figure 1—The PIC-an-LCD requires very few external components for normal operation.

datasheets I've encountered. Timing is the most critical element in this process. When you start writing your own LCD initialization routines, do yourself a favor and start out really slow. This is the number one cause of improperly initialized devices. Another tricky aspect of debugging initialization code is that the LCD tends to stay initialized even during short power cycles. So, while you think you're completely resetting the system by turning it off briefly and turning it back on, the

LCD is still merrily ticking along like nothing happened. This masks subtle bugs that sneak into your code, such as reducing some of the timing parameters.

Now that the LCD is properly initialized, you may see a single block cursor, a blinking underline cursor, or nothing at all. This behavior is defined as part of the initialization phase. I always like to see positive proof of correct behavior, so the first few times after I cobble together some LCD code I like to leave a big, rude, blinking block cursor flashing at me. I also decided that would be the best option for all the PIC-an-LCD chips out there, so their new owners could see that something was happening. Once I've decided on a particular application for the LCD project, I normally turn off the cursor, except for user-input functions.

STEP 3 – DATA & COMMANDS

The third item concerns the sending of data and commands to the LCD module. Timing, as always, is critical. Data sent to the data register generally gets written to the display memory buffer. For example, if you write the ASCII code for an exclamation point (0x21) to the data register, an exclamation point appears where the cursor was, and the cursor is advanced one space to the right. You can send an instruction to the instruction register and expect it to be carried out, but be aware that the controller chips are relatively slow when compared to most microcontrollers. It can take many milliseconds to execute some instructions, especially if it involves clearing the screen, which involves a multitude of writes.

There are a lot of assumptions in the last paragraph. Instead of the dis-

Fighting against your PCB-Design Software?

Here's something that will spare your time and your budget!

Boards designed under EAGLE are found in patient monitoring equipment, chip cards, electric razors, hearing aids, automobiles and industrial controllers. They are as small as a thumbnail or as large as a PC motherboard. They are developed in one-man businesses or in large industrial companies. EAGLE is being used in many of the top companies. The crucial reason for selecting EAGLE is not usually the very favorable price, but rather the ease of use. On top of that comes the outstanding level of support, which at CadSoft is always free of charge, and is available without restriction to every customer. These are the real cost killers!

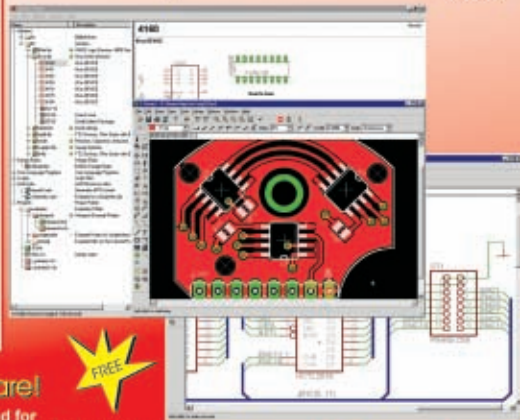


EAGLE 4.1

Schematic Capture • Board Layout
Autorouter

for Windows®
Linux®
Mac®

Now available for Mac!



Version 4.1 Highlights

- ▶ Powerful library management: e.g. move devices between libraries, base library for packages, generate package variants from other libraries.
- ▶ Dynamic ratsnest during routing process.
- ▶ Copy function in schematic.
- ▶ Rotate components in 0.1-degree steps.
- ▶ Blind & buried vias and pads with off-center drill.
- ▶ User-defined background color.
- ▶ Miter function for (rounded) tracks.
- ▶ Smash for groups.
- ▶ Measure distances between arbitrary points.
- ▶ Choose alternative raster on-the-fly with Alt-key.

EAGLE 4.1 Light is Freeware!

You can use EAGLE Light for testing and for non-commercial applications without charge. The Freeware Version is restricted to boards up to half Eurocard format, with a maximum of two signal layers and one schematic sheet. All other features correspond to those of the Professional Version. Download it from our Internet Site or order our free CD.

If you decide in favor of the Commercial Light Version, you also get the reference manual and a license for commercial applications. The Standard Version is suitable for boards in Eurocard format with up to 4 signal layers (max. 99 schematic sheets). The Professional Version has no such limitations.

<http://www.CadSoftUSA.com>
800-858-8355

Prices	Light	Standard	Professional
Layout		199\$	399\$
Layout + Schematic		398\$	798\$
Layout + Autorouter		398\$	798\$
Layout + Schematic + Autorouter	49\$	597\$	1197\$

Pay the difference for Upgrades

CadSoft Computer, Inc., 801 S. Federal Highway, Delray Beach, FL 33483
Hotline (561) 274-8355, Fax (561) 274-8218, E-Mail: info@cadsoftusa.com

play memory buffer, you can tell the LCD controller to point to the custom character generator RAM (CGRAM) section. Writing data to the data register sends dot patterns to the CGRAM, eight per character, with a total of eight custom characters available. These are mapped to locations 0 to 7, and duplicated in locations 8 to 15, which normally correspond to the otherwise unprintable ASCII

control codes. The PIC-an-LCD uses a couple of the custom character slots to create characters common to most PCs but missing from the standard LCD character generator ROM, specifically the backslash “\” and the tilde “~.” Also, the expected automatic advancement of the cursor is a programmable option. It can go backwards or nowhere, depending on how the unit is initialized. Even the number of visible lines on the LCD must be programmed because the same controller chipset is used for many different configurations of LCDs.

The mapping of the characters on the LCD is not contiguous. All LCD modules of this flavor have 80 bytes of display RAM, which is configured as two lines of 40 characters each. The addresses of the first bytes of each row, however, are 64 bytes apart. That leaves a 24-byte gap from the end of line one to the beginning of line two.

The PIC-an-LCD handles noncontiguous mapping fairly well, but not perfectly. It introduces the concepts of “carriage return” and “line feed,” for example, which exist in the typewriter/teletype output device model, but not in the native LCD functionality.

So, by now we have the LCD module properly connected to the appropriate power supply and we have adjusted the contrast for good viewing. I can't emphasize enough how important it is to do this before trying to send commands and data to the LCD. Now it is time to connect some more wires to the LCD, but this time they come from the microcontroller.

SHORT(ER) BUS

The HD44780 interface uses a parallel data bus and three control lines.

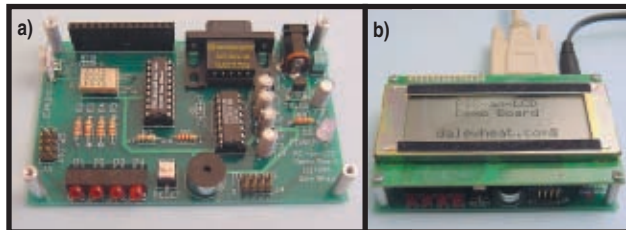


Photo 2a—The PIC-an-LCD demo board is shown with the LCD module removed so you can see its components. I crammed everything but the kitchen sink onto the board and still made it fit within the form factor of a standard four-line LCD module. The PIC-an-LCD chip is the one on the left. **2b**—The PIC-an-LCD demo board is shown completely assembled. Once upon a time, these 4 x 20 LCD modules were everywhere to be found in the surplus channel and quite attractively priced.

The data bus is normally 8 bits wide. The internal registers and memory locations of the HD44780 are also 8 bits wide. A very nice feature that was added to the HD44780 interface was the ability to send 8-bit data and instructions in two steps, using only 4 data bits. This reduces the number of dedicated interface lines. This is the method used by the PIC-an-LCD. Only the upper 4 bits (D4, D5, D6, and D7) of the LCD module's data bus are connected to the PIC-an-LCD chip.

The control lines are very simple. The

first one to consider is the register select, or RS line, which is on pin 4 on most LCD modules. When this line is a logic one, the data bus is connected to the data register. Think of this register as the gateway to the display data and custom character generator memory arrays. When the register select line is a logic zero, the data bus is connected to the instruction register or the status register, depending on the state of the

read/write, or R/W line.

The read/write line, which is on pin 5, controls the direction of the data flow on the data bus. When low, data is written from the data bus to the destination within the controller chip indicated by the register select line. When the line is high, data is read out of the controller chip and output on the data bus. Many simpler LCD interfaces omit the control line and use only writes to control the LCD. This method is perfectly valid, but it tends to run a little slower because open-loop wait states must be inserted after every write.

Going Wireless is Easy!

Wireless Mesh Networking

NEW!



Bluetooth

with Dip-Switch options



USB
ZigBee™
Stick

LEMOS

INTERNATIONAL

Multi-Channel
900 MHz Wireless
RF Modules



Call Toll Free 1-866-345-3667 or email at sales@lemosint.com

www.lemosint.com

Being able to read back from the status register will enable the microcontroller to know exactly when the HD44780 is ready for more data and instructions.

The next and last control line is the enable line, or just E. It is a positive-going pulse that clocks the data in or out, depending on the state of the other two lines. Note that the pulse generally has a minimum active state time of 450 ns, but it can vary from device to device. This is by far the most common timing pitfall of working with LCDs. Modern microcontrollers can raise and lower individual output lines in much less time, and a pulse that's too skinny just doesn't get the job done. The enable line is always on pin 6.

LCD INITIALIZATION IN C

The first thing that needs to be done each and every time the PIC-an-LCD is powered up is initialize the LCD module. The complete listing is posted on the *Circuit Cellar* FTP site.

One of the handy features of the CCS C compiler is the way it handles bit fields. Many limited implementations of the C language, especially for embedded applications, omit bit-field capabilities. Since most input and output with microcontrollers consists of bit twiddling on some level, the ability to use structures of bit fields cleans up the source code. The CCS C compiler also allows input and output registers to be mapped using structure definitions. That spares the programmer from a lot of shifting, masking, and tedious bit counting. The LCD interface uses four data lines and three control lines, for a total of seven lines. The lines are all mapped to a single I/O port, but can be referred to individually as `lcd.rs`, `lcd.rw`, `lcd.e`, and `lcd.data`, which represent four data lines, respectively.

Although the PIC16C621 lacks a true external data bus, each of the general-purpose input and output lines can be reconfigured on the fly. This enables a single pin to be an output one moment and an input the next. For the initialization of the LCD module, we can get by with outputs only. This is accomplished with the `set_tris_b(LCD_WRITE)` statement, `LCD_WRITE` being a constant defined using the `lcd_pin_map` structure. Functions that need to read from the LCD module can reverse the direction of

the port pins using `LCD_READ` as an argument to `set_tris_b()`. A 15-ms delay is generated to give the LCD module time to "wake up." Now, the tricky part: convince the HD44780 to operate in 4-bit mode, even though it wakes up in 8-bit mode. This is done by repeatedly sending a partial command to set the interface length with odd timing spaces. Once that's done, send the command to set (again) the 4-bit data mode, as well as the two-line mode and the 5×7 font mode. These are the most common settings. They are summed up in the instruction code `0x2c`. Next, clear the screen with the `0x01` instruction and then wrap up the initialization with the instruction that sets up a block cursor, `0x0d`.

SERIAL INPUT

The CCS C compiler also provides a library for performing serial input and output, even in the absence of a UART. Notice that the serial input line also happens to be the interrupt pin. Depending on the state of the serial input line at power-up, the program sets up either inverted or noninverted routines to accept incoming serial data. This enables the PIC-an-LCD to be connected directly to any RS-232 device with only a resistor. No signal inversion circuitry is required. It will also work if it is connected directly to another microcontroller, like a BASIC Stamp, another PIC, or whatever you can cobble up. The interrupt is configured to be either rising- or falling-edge triggered, and that sets the serial input routine in motion. A character is received and stored in a circular buffer.

The PIC16C621 requires an external crystal for normal operation. The PIC-an-LCD code assumes either a color-burst crystal with a frequency of 3.579545 MHz, which results in an incoming data rate of 2,400 bps, or four times that frequency (14.31818 MHz) to obtain 9,600 bps. When I originally designed the PIC-an-LCD, using the PIC16C621, the available maximum clock rates were 4 MHz and 20 MHz. The new PIC16C621A offers those clock speeds and even a 40-MHz version. Theoretically you could bump the data rate up to 19,200 bps, but I haven't tested this.

HANDLING DATA & COMMANDS

The main loop waits for a character

to show up in the buffer and then sends it to the `lcd_putc()` routine, which decides what to do with it. The `lcd_putc()` routine is where all the features are mapped. It is a single, large `switch()` statement. First, all the command cases are checked. If the received character is not a command, it is assumed to be printable data. I remapped the custom characters from their native positions of 0 to 7 (repeated at 8 to 15) to 128 to 136 to avoid conflicts with the ASCII control characters.

The simplest command cases are those that can be handled by sending a single instruction to the LCD module. These include the home cursor, cursor left, cursor right, clear screen, shift left, shift right, and special cases to properly display the backslash "\" and tilde "~" characters, which are not normally present.

The next simplest command cases are the ones that simply wait for another character to arrive and then do something with that data. These include the cursor address, send LCD command, send LCD data, set number of lines, set general-purpose output lines, print signed decimal number, print unsigned decimal number, and set cursor display options.

This leaves the most complex commands, which generally require a function call to process. The simplest of these are the commands to save and restore the cursor position. The `bell()` function toggles the bell output at 1 kHz for 0.1 s, assuming a 14.31818-MHz crystal.

While the LCD module supports the equivalent of a nondestructive backspace command with the cursor left command, I wanted to include a destructive backspace as well, because this is what most people expect to happen when they press the backspace key. It's trivial to do. First, a nondestructive backspace command is issued. This backs up the cursor one space. Then, a space character is printed, but since this moves the cursor back to where it was originally, another nondestructive backspace command is issued. The horizontal tab command moves the cursor to the next tab position, which is presently fixed at multiples of four.

The remaining functions, which include vertical tab, carriage return, and line feed are more sophisticated in that they have to take varying LCD geome-

tries into account. Even though all LCD modules think they have two lines of 40 characters each, they are often packaged in many other configurations and mapped accordingly. This can provide some confusing behaviors to the uninitiated.

CONCLUSION

This concludes the whirlwind tour of the source code. If you're actually looking at the original code, you can see some things have been commented out. One trick I learned about the CCS C compiler, or at least the version I was using 10 years ago, was that if you left out the very last break statement in a `switch()` statement, you would save a word of program code. I also had to leave out the print hexadecimal number function, in order to be able to leave in the signed and unsigned decimal number functions. I ran out of room. All told, the final version uses 2,044 of the 2,048 bytes available in the program memory.

There's a noncritical but slightly annoying bug in the code. Can you find it? If you're a real code tweaker, you might want to up the size of the serial input buffer. The original PIC16C621 chip had 80 bytes of RAM, 16 of which could be used for the serial input buffer. The new PIC16C621A has 96 bytes, so why not use them to increase the size of the serial input buffer from 16 to 32 bytes? Because the serial input routine is software based, it eats up a lot of foreground time receiving characters. That leaves very little time to actually process them. A bigger buffer will help postpone the inevitable buffer overruns.

BUILDING A PIC-AN-LCD CIRCUIT

Figure 1 shows a minimal PIC-an-LCD example circuit. A step-by-step guide to building a PIC-an-LCD circuit is also detailed in the user's manual. This circuit can easily be built on a solderless breadboard.

Another example circuit is the official PIC-an-LCD demo board (see Photos 2a and 2b). This was produced and sold as a kit and as an assembled unit. It was my attempt at using all of the features of the PIC-an-LCD chip. This was also my first commercially produced PCB design. Not knowing any better, I just laid out the components and traces using a Gerber file editor, without the benefit of a

schematic. I was rather pleased that it required no feed-throughs, also known as vias. It seems that I learned the basics of PCB layout and design about two decades too early, when every single drill hit cost money, and single-sided was the way to go, if possible.

SUCCESS!

The PIC-an-LCD user's manual is available for download on my web site <http://dalewheat.com>. It has all the information needed to hook up the PIC-an-LCD in a typical application circuit. The PIC-an-LCD condensed datasheet is also available on my web site. It has all the main points, such as pin descriptions, a command summary, and typical LCD connection, all on a single page.

The PIC-an-LCD was my first commercially successful design, and I'd like to thank Billy Gage of BG Micro for making it possible. I now give it away to the world to use as it sees fit. ☺

Dale Wheat (dalewheat.com) is a full-time freelance writer in the Dallas area. He works primarily with embedded systems and shiny things that blink or beep. Dale is married and the father of two adult children.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2007/206.

SOURCES

PIC-an-LCD

BG Micro
www.bgmicro.com

C Compiler

Custom Computer Services, Inc.
www.ccsinfo.com

HD44780 Driver chipset

Hitachi
www.hitachi.com

PIC16C621 Microcontroller


Microchip Technology, Inc.
www.microchip.com

EMP-20 and EMP-21 Programmers

Needham's Electronics, Inc.
www.needhams.com

KS0066U LCD Controller

Samsung
www.samsung.com

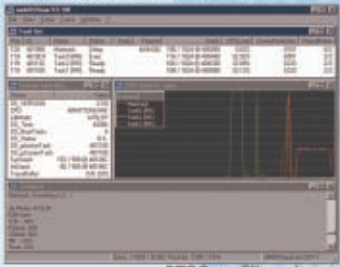


Eval Versions Available

RTOS (embOS®)

+++ 8/16/32 bits +++

- Object Or Source Code
- Small Footprint
- Preemptive Multitasking
- Zero Interrupt Latency
- Easy to Use Start Project
- Profiling Support Included




- RTOS profiling viewer

GUI (emWin®)

+++ 8/16/32 bits +++

- ANSI "C" Source Code
- 2D Graphic Library
- Window Manager/Widgets
- PC Simulation Included
- RTOS Independent
- Font Converter
- Image Converter



- car dashboard example

FILE SYSTEM (emFile)


+++ 8/16/32 bits +++

- ANSI "C" Source Code
- MS-DOS/MS-Windows Compatible
- FAT12, FAT16, FAT32 Support
- Non FAT File System Available
- RTOS Independent

JTAG (J-Link™)

+++ ARM7/9 Cortex M3 +++

- Fast 720 kb/s Download
- USB to JTAG



phone: 978-874-0299

www.segger.com

High-Performance Motor Controller

With a USB interface, Daniel can issue high-level commands to his motor controller from his laptop. This enables his laptop to process algorithms while an embedded controller handles the rest of the work. Motor control has never been easier.

Unlike PCs, which have a high-performance peripheral controller interface (PCI) bus, laptops have not been widely used to control embedded hardware because of their inherent I/O limitations. However, recent developments in embedded Ethernet appliances and the wide acceptance of USB-based devices for both laptops and PCs now make it possible for design systems to harness the incredible processing speeds (gigahertz) of today's laptops. For high-end industrial applications, embedded Ethernet is available with connections to embedded devices via NetBurner, Rabbit Semiconductor, and similar boards. But these solutions are usually more costly than those based on USB boards. Embedded Ethernet requires an embedded TCP/IP stack and more involved Ethernet software development compared to the USB solution proposed in this article.

The USB motor control system is shown in the block diagram in Figure 1. The laptop acts as the host and a Microchip Technology PIC18F4450 USB microcontroller is the USB peripheral. The 16-bit Microchip dsPIC30F4011 motor controller board and the dual H-bridge board handle motor control. The two-microcontroller approach provides the flexibility of a USB interface combined with the processing power of a laptop. This includes using the various laptop development and analysis tools, such as Excel and MATLAB and software

development tools, such as Visual C++. The block diagram shows how the PIC18F4450 acts as a gateway to communicate with the dsPIC30F4011, using the SPI and USB interfaces.

Don't let the "USB" keyword give you the impression that this interface is difficult to use in applications with a lot of human interface description (HID) programming and USB endpoints (buffers), which require a lot of technical reading. Microchip provides all the necessary drivers to enable you to access the USB port and treat it as a serial port (COM3) on the laptop using a Windows virtual communication port (VCP). When all you really want to do is control a DC motor with a laptop using your favorite application development language (e.g., Ada, Basic, C, C++, or JAVA), the well-written Microchip application note AN956 gives you all the necessary information to use the USB

port in this manner. For more detailed information on USB specifications and the HID interface, refer to Jan Axelson's *USB Complete: Everything You Need to Develop Custom USB Peripherals*. It provides what you need to develop advanced HID-based USB applications.

Although I could have used Visual C++ or Visual Basic for the host motor controller application running on the laptop, I chose the Ada95 language because of its excellent reputation as a reliable embedded software development language for safety-critical applications. These include such diverse applications as aircraft systems, medical applications, the NASA International Space Station (Canadarm), and the European Space Agency's Ariane rocket.

SENSOR PLATFORM

The USB motor controller was designed to drive some of my home security and robotics projects, including the tilt/pan sensor platform that I made from Vex parts. A Radio Shack SW-P-WOC Swann Communications NightHawk wireless security camera is mounted on it (see Photo 1). The Vex robotics design system is manufactured by Innovation First. It is similar to other construction sets, such as the original A.C. Gilbert and Meccano erector sets that enabled you to build all kinds of robots, props, and other devices. The design system originally sold at Radio Shack

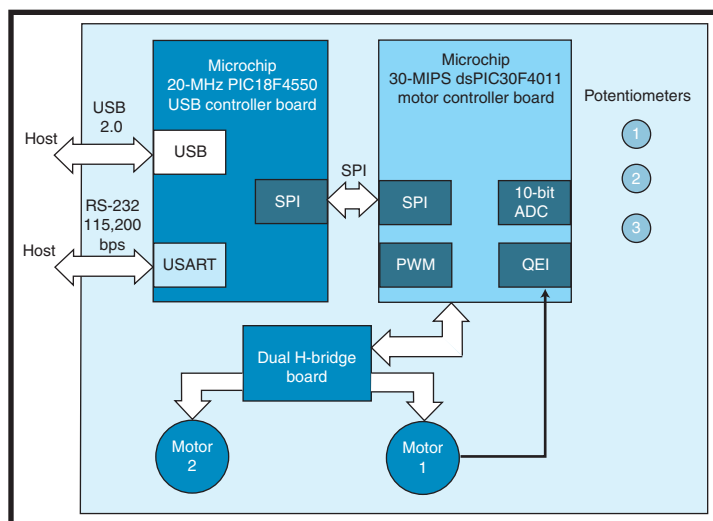


Figure 1—The block diagram of the USB motor controller system shows all the major components, including the laptop, the USB controller board, the DC motor controller board, and the H-bridge motor driver board.

stores, but it is now available only on the Internet directly from Innovation First.

Dean Kaman (inventor of the Segway) initially inspired and supported For Inspiration and Recognition of Science and Technology (FIRST) competitions using the Vex system. The system is currently used in FIRST competitions at high schools all over the U.S., and with Dean's continued support, it is now an integral part of Rhode Island's high school robotics curriculum.

To test the USB motor controller, I connected the USB motor controller's PWM outputs to the two PWM inputs from the dual H-bridge board, which directly drives the two 5-V geared HD DC motors. The platform frame was made from Vex system components, which I purchased at Radio Shack, along with gears and belts salvaged from old surplus equipment.

The sensor platform has two very powerful HD geared DC motors, each with 100 counts-per-revolution (CPR) optical encoders attached to their respective input shafts. One of the

motors is used to pan the sensor platform 0° to 360°, while the other motor is used to tilt the sensor platform ±90°. My goal was to be able to position the platform to within ±1° resolution. The encoders attached to the motors should help attain these specifications under laptop control using the USB bus.

The first motor pans the sensor platform using the timing belt. The second motor, which is attached to the platform itself, tilts the sensor. While this may seem like a very simple example, the USB motor controller also allows you to control many types of motors, such as coreless, BLDC, and three-phase motors using sophisticated PID control. Microchip provides all the necessary application notes and firmware for that.

AN ELEGANT SOLUTION

One of the neat features of Microchip's USB-based serial port, application note AN956, is the fact that practically any PC- or laptop-based serial application will work using a VCP

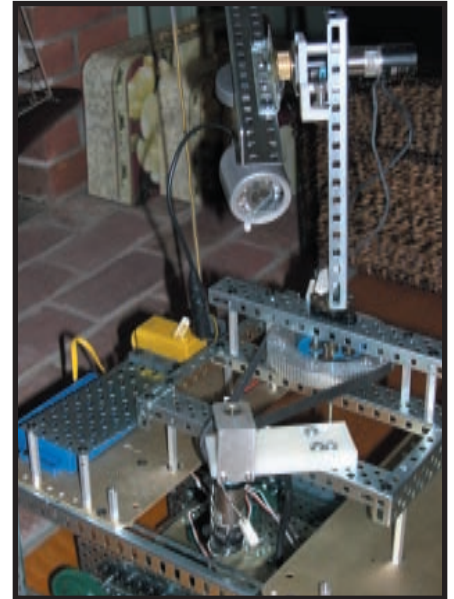


Photo 1—This is the tilt/pan sensor platform with the NightHawk wireless security camera mounted on it.

with no HID programming required. This feature is what makes the board act as a gateway to embedded controller applications, such as the USB motor controller. It is an excellent feature

since the current trend in laptop designs is to leave out serial ports, including the parallel port, in favor of a USB port. The USB port can also supply the 5-V power supply to an embedded project, as long as the power requirements are modest (around 100 mA).

It is important to note that the Microchip USB driver was developed on the PIC18F4550 USB controller as a state machine, so only polling or interrupt processing may take place in the user's application, but no blocking statements can be used or the driver will hang the controller.

The PIC18F4550 board is a very well-designed, low-cost board that uses a 20-MHz oscillator to generate the 48-MHz clock required for USB communications using its on-chip PLL. The board derives its power from the USB bus, so no external power supply is required.

You would think that I worked for Microchip, with the praise I have for their products and accompanying software tools, which my motor controller design relies on. The truth is I don't live anywhere

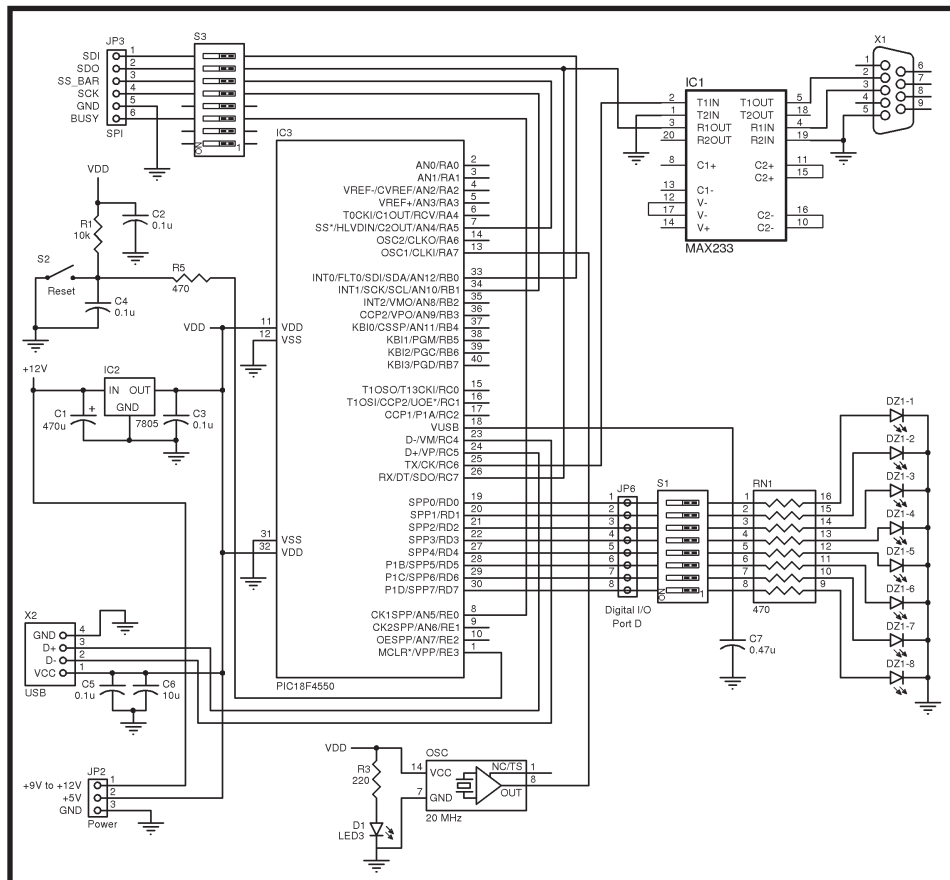


Figure 2—The DIY PIC18F4550 USB controller board schematic shows how to make the USB interface between the laptop and the main motor controller board.

**Add SD
in a FLASH!**

with our

Flash Card Library

- ✓ Easily exchange data between your product and a PC
- ✓ Create, Append and Delete PC compatible files
- ✓ Add GIGABYTEs of onboard or removable storage to your product
- ✓ Compatible with over 20 card styles SD, MMC, MicroSD, TransFlash, ...
- ✓ No OS, RTOS or interrupts are needed
- ✓ Works with low cost Microcontrollers
- ✓ Support for FAT16 and FAT32 file systems
- ✓ Full C sources easily ports to many C compilers

Full directory support **NEW**

Flash Card Adapter Board

The flash card adapter board contains all the basic circuitry needed to add PC-compatible flash card memory to virtually any microcontroller application, using as few as four I/O pins. It also includes a 3.3V regulator, built-in level shifting, and sockets for both SD/MMC and MicroSD/TransFlash memory cards.



PIC Development Board

The development board includes a built-in demo application and all the necessary hardware to run each of the example programs included with the SFCLIB (sold separately). It includes sockets for SD/MMC and MicroSD/TransFlash memory cards. The board also has an MPLAB ICD 2 In-Circuit Debugger jack for easy programming and debugging, and a large prototyping area for adding your own circuitry.



**EFFICIENT
COMPUTER SYSTEMS**

Offering MEGA solutions for the embedded MICRO world.

For more information see our website www.ECS87.com/sfclib
603-776-3151 • 5 Emerson Drive Center Barnstead, NH 03225

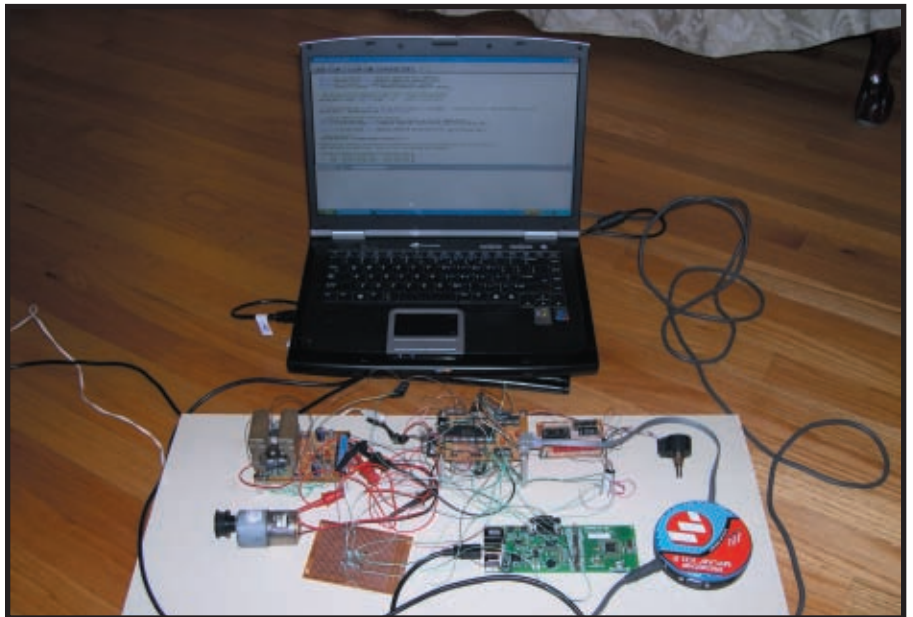


Photo 2—The PIC18F4550 USB controller board acts as a gateway between a laptop and a dsPIC30F4011 motor controller.

near enough to the beautiful state of Arizona to be able to work there (as much as I like visiting the Grand Canyon, Sedona, and the large crater).

DIY USB 2.0 BOARD

As you can see, Microchip has taken a lot of the USB drudgery away from the DM163025 full-speed USB demo board, which includes all the necessary USB hardware and the software drivers for Windows XP. Although I purchased a couple of these boards for convenience, I was able to reuse some of my older PIC18F452 prototype boards. I just substituted the PIC18F4550 with minimal changes for my own DIY USB controller (see Figure 2).

In addition to the USB interface, this board also supports a USART, an SPI, and an I²C interface. I opted to use the SPI to communicate with the dsPIC30F4011 motor controller because of the higher data rates possible using an 8-bit SPI, although the USART and I²C interfaces can also be used for this purpose.

I purchased a PIC18F4550 USB controller board directly from Microchip (see Photo 2). It came with the USB cable, USB connector, and provided all the necessary firmware and USB drivers to handle the

USB 2.0 interface to the laptop. In future designs, I will include it as part of the design since it is very easy to do and will lower the cost. The board was very easy to set up and use. Microchip provides a complete schematic of the board for readers who would prefer to build it from scratch. In fact, to make the construction easier, you can purchase the board (see Figure 2).

A TALE OF TWO PICS

Now you know how the laptop talks to the USB controller and acts as a communications gateway between the laptop and the embedded dsPIC30F4011 motor controller. Inter-processor communication between the two controllers is handled via the SPI



Photo 3—The RH-5A-5502 geared DC motor is a very powerful low-voltage (5 V) geared motor that also has a 100-CPR optical encoder attached to the motor's shaft.

Ultra-Small MCUs, But That's Only Half The Story

256–2304 bytes RAM

10–12 Bit DAC
smarTClock

32 kB Flash
100 MIPS

2% Oscillator

8–24 Bit ADC
500 ksp/s

Up to 6 channel PWM

16x16 MAC

4-Channel, 16-Bit Timer

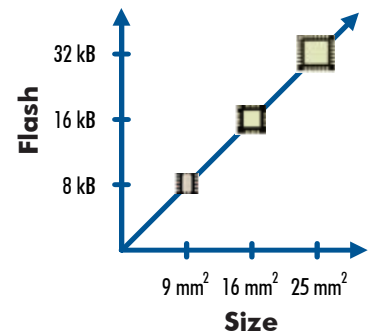
UART, SMBus, SPI

Temp Sensor

Comparators

Fully Loaded, Small Form Factor MCUs

Silicon Laboratories' mixed-signal MCUs combine a high-speed 8051 CPU, Flash memory and best-in-class analog peripherals in ultra-small packages allowing designers to reduce component count while improving system performance. These highly-integrated, feature-rich devices are optimized for space-constrained, low-power, cost-sensitive applications.



Low-Cost, Professional Tools

Complete development kits (including in-system debug and integrated development environment) make development quick and easy. Low-cost evaluation kits and reference designs simplify evaluation and accelerate time-to-market.



Highest Functional Density

- 8–24 Bit ADC
- 2–32 kB Flash Memory
- 25–100 MIPS 8051 CPU

Product details: www.silabs.com/SmallMCU


SILICON LABS

www.silabs.com

Where you can get all your semi's!

	Mouser®	Newark®	Catalog Distributor X	Jameco®
Altera		✓		✓
Analog Devices		✓	✓	✓
Atmel Semiconductor	✓		✓	✓
Avago Technology	✓	✓	✓	✓
Cypress Semiconductor		✓		✓
Diodes Inc.	✓			✓
Fairchild Semiconductor	✓	✓		✓
Freescale Semiconductor	✓	✓	✓	✓
Infineon Technology			✓	✓
Integrated Devices		✓	✓	✓
Intel Corporation		✓	✓	✓
Intersil	✓	✓	✓	✓
Lattice Semiconductor	✓	✓	✓	✓
Linear Technology				✓
Lite-On	✓			✓
Maxim Semiconductor		✓	✓	✓
Micron Technology				✓
Microsemi				✓
National Semiconductor		✓	✓	✓
NEC Corporation	✓			✓
Philips Semi			✓	✓
Renesas Technology				✓
Sharp Microelectronics	✓			✓
ST Micro	✓	✓	✓	✓
Texas Instruments	✓	✓	✓	✓
Toshiba				✓
TOTAL	12	14	14	26

More semi lines. That's the Jameco advantage.

Now get the broadest selection of semiconductor lines from Jameco! In addition to finding all the semi's you want in one location, engineers have also found that Jameco offers the highest in-stock availability, the lowest prices guaranteed and same-day shipping, making Jameco the new choice for electronic components. Visit www.Jameco.com or call 1-800-831-4242 to place an order or request a free catalog.

JAMECO
ELECTRONICS

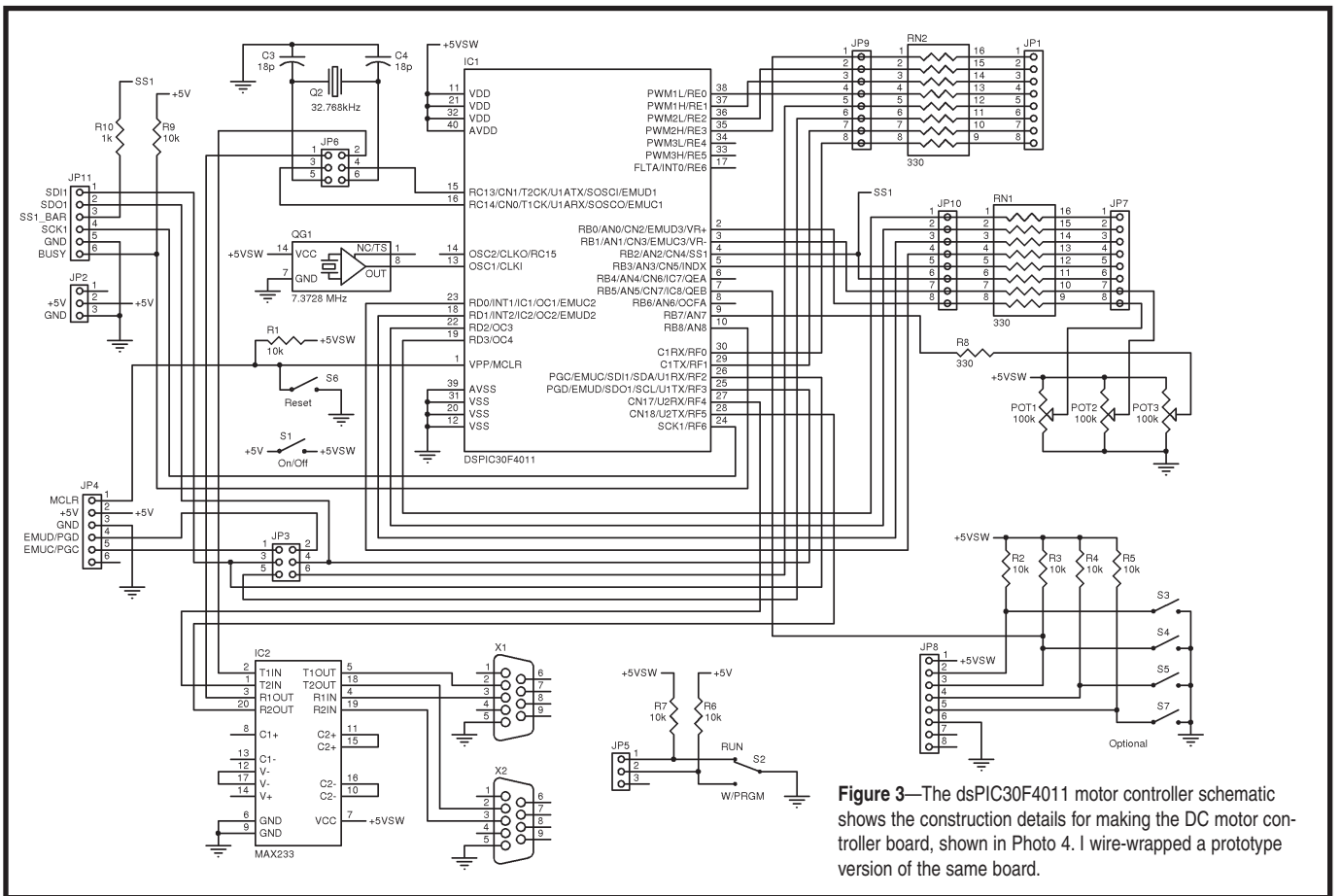


Figure 3—The dsPIC30F4011 motor controller schematic shows the construction details for making the DC motor controller board, shown in Photo 4. I wire-wrapped a prototype version of the same board.

Propeller™ Starter Kit



The **Propeller Starter Kit** includes everything you need to get started with the Propeller microcontroller. Kit includes the Propeller Demo Board, the Propeller Manual, software on CD, a power supply, and a USB cable. The Propeller Demo Board includes a built-in Propeller chip (P8X32A-Q44), EEPROM and 5 MHz crystal pre-wired to connectors for interfacing to devices such as a mouse, keyboard, TV or VGA monitor and speakers. Together with the many objects from the Propeller Object Library you can see some of the impressive tasks the Propeller can perform within minutes.

Propeller Starter Kit; #32300; \$149.95

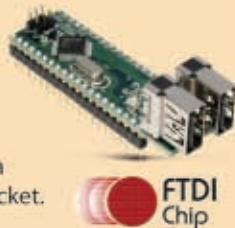
Order the **Propeller Starter Kit** at <http://www.parallax.com/propeller> or call our Sales Department toll-free at 888-512-1024 (Mon-Fri, 7am-5pm, PDT).

Propeller, Parallax and the Parallax logo are trademarks of Parallax Inc.



VDIP2 USB Host Controller Module

MCU-to-USB host controller development module supplied on a PCB; designed to fit a 40-pin DIP socket.



mouser.com/ftdi/a

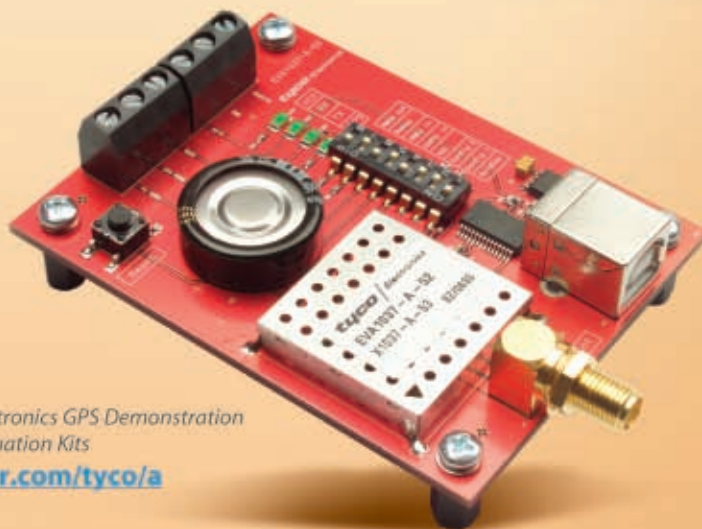
MatchPort™ Embedded Device Server

Enables the building of wireless networking into any electronic device with a serial interface.



mouser.com/lantronix/a

Embedded Products for the Latest Technologies



Tyco Electronics GPS Demonstration and Evaluation Kits

mouser.com/tyco/a

GPS Receiver Module

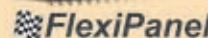
GPS receiver module with accurate positioning, small footprint, and ultra-low power consumption.



mouser.com/tyco/a

Toothpick 2.0™

PIC MCU module with integral Bluetooth® radio and Toothpick™ services firmware layer.



mouser.com/flexipanel/a

Universal Socket Module

Flexible, comm-port architecture providing cellular, Ethernet, PSTN or Wi-Fi® network access with interchangeable modules.



mouser.com/multitechsystems/a

- The ONLY New Catalog Every 90 Days
- NEWEST Products & Technologies
- Over 845,000 Products Online
- More Than 330 Manufacturers
- No Minimum Order
- Fast Delivery, Same-day Shipping



a tti company

*The Newest Products
For Your Newest Designs*

mouser.com (800) 346-6873

bus (see Figure 2). The PIC18F4550 now takes the role of SPI master and communicates with the dsPIC30F4011 motor controller, which functions as an SPI slave via the 8-bit SPI bus.

The PIC18F4550's I/O pins are used for SPI communication (RB0/SDI, RC7/SDO, RA5/SS_BAR, RB1/SCK, and RE0/BUSY). The corresponding motor controller dsPIC30F4011's pins used for SPI include RF3/SDO, RF2/SDI, SS1_BAR/RB2, RF6/SCK1, and BUSY/RB8, as shown in the connection diagram. Just make sure that the SDO outputs connect to the corresponding SDI inputs from each controller.

I found it necessary to insert a 1-k Ω resistor in series with the SS_BAR (slave select) line in order to obtain reliable SPI communication between the two controllers. In addition, I added a busy line that is polled by the firmware and is used to indicate when a motor command has been processed.

Functions from Microchip's SPI C libraries are used to read and write data or outgoing data over the SPI bus. The PIC18F4550 is configured as the

SPI master, while the dsPIC30F4011 is configured as the SPI slave peripheral, using an interrupt service routine (ISR). These routines wait for an SPI interrupt generated when a byte of data arrives and are used to store data in an SPI Rx buffer. A complete message of 64 bytes is processed as a motor message.

MOTOR CONTROLS

I plan to add three potentiometers to the motor controller board, so in addition to reading the latest encoder counts, the firmware will enable the user to read up to three analog inputs connected to three potentiometers, which act as motor controls (see Figure 3). These potentiometers should make it easier for the oper-

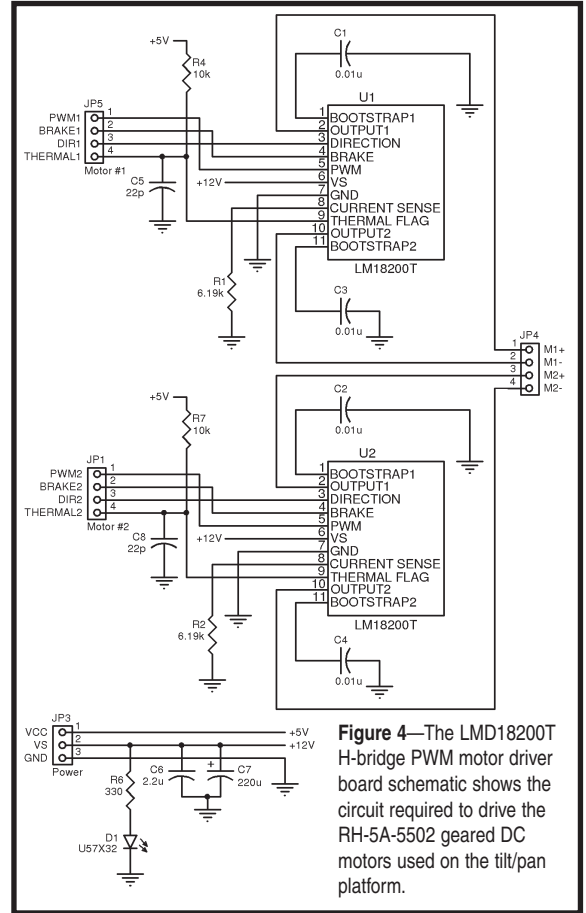


Figure 4—The LMD18200T H-bridge PWM motor driver board schematic shows the circuit required to drive the RH-5A-5502 geared DC motors used on the tilt/pan platform.

PCB-POOL®

SERVICING YOUR COMPLETE PROTOTYPE NEEDS

Price Example: 16 Sq-Inches (double sided pth)

2 Days: \$ 90.00

8 Days: \$ 22.50

Standard PCB-Pool Service
SIMPLY SEND YOUR FILES AND ORDER ONLINE!

New Service: WATCH "ur" PCB®

Save vital time on design errors in advance of receiving your Prototype. View high resolution photographic images of your PCB during each production stage. Be one step ahead, use our realtime PCB monitoring service.

WWW.PCB-POOL.COM

Tollfree USA : 1877 390 8541
sales@beta-layout.com

Visit us at
ESC Boston
Booth #321

DOWNLOAD OUR FREE PCB SOFTWARE
www.free-pcb-software.com

Industry Quality
LEAD FREE
Pb, Sn, Ag, Cu, Ni, Au

ROHS / WEEE
conform

TARGET
PROTEUS
PreTel
Electronics WORKBENCH
orcad
Serfing Layout
Easy-PC
ELMS
GraphCode
R

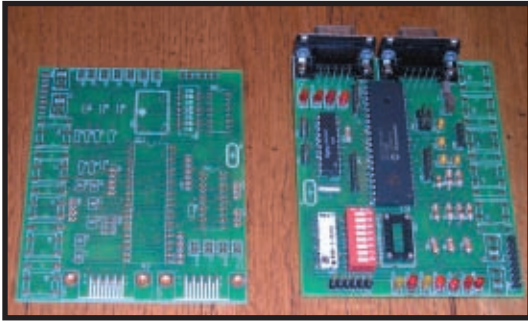


Photo 4—This photo shows the double-sided motor controller PCB that I recently completed. It works just like the wire-wrapped version, but it is easier to build.

ator to adjust the motor position, velocity, direction, and acceleration during Operational and Calibration modes.

These controls will help reduce the time spent during the calibration “fine tuning” process, which by nature, requires much iteration. They enable the user to modify the KP, KI, and KD constants as needed in order to bring the “plant” under PID control.

The three potentiometers are an optional feature that require software to be written, which I plan to do in the future.

DC-GEARED MOTORS

The RH-5A-5502 geared DC motor comes attached with a 100-CPS encoder with an ultraflat gear head and zero backlash (see Photo 3). The motor’s features make it ideal for mirror-smooth motions required for pointing devices, such as security cameras, web cams, and distance-measuring devices. They were salvaged from surplus scanner equipment that I found at a local electronics surplus store. The rated torque for each Harmonic Drive RH-5A-5502 is 0.29 Nm, the maximum speed of rotation is 110 RPM, and the rated speed of rotation is 55 RPM.

QUAD ENCODER INTERFACE

The next step towards obtaining precise motor speed or position control is to use some kind of sensor feedback that indicates the motor’s actual speed and direction. The sensor usually used for this purpose is called an optical encoder (see Photo 3). Other types of encoders exist, such as rotary encoders and magnetic Hall-effect encoders, in addition to variable resistors, which can provide a resistance or voltage value proportional to the number of turns it has made. These encoders are connected to the shaft of the motor whose speed and direction is being monitored. The most common encoders are absolute encoders and relative encoders. Absolute encoders provide the actual position and direction while relative encoders provide the number of counts and direction.

MOTOR CONTROLLER BOARD

The dsPIC30F4011-based motor controller board, shown in Photo 2, is the heart of this project. It drives the complementary PWM outputs, as required, to control the motor’s speed and direction and also read the optical encoder via the QEI interface. In addition, it also reads three potentiometers connected to three analog channels (AN0–AN2) from the 10-bit ADC. It also has two UARTS, two SPI ports, and one PC port. After examining the block diagram, you can see how the two microcontrollers com-

Proteus Design Suite 7

Make Your Job Easier & Faster!

- Design Your Schematic
- Simulate Your Design
- Debug Your Source Code
- Make Your PCB

NEW in Proteus Version 7:

- * **New User Interface**
 - Easy to Use - Conventional Mouse Buttons
 - Context Menus - More information Available to the User
 - More Information on Screen to Assist User
- * **New Design Explorer**
 - Cross Probing, from Schematic to PCB
 - Spreadsheet like view of the design parts & Netlist
 - Parts list view shows the design by sheets, parts & pins
 - Drilling down to Parts Pins and nets
- * **New Tools**
 - New Industrial Grade 4 Channel Oscilloscope
 - Logic Analyzer Instruments
 - I²C and SPI Debugger Tools, Master/Slave Mode
 - 3D Visualization of PCB Layout
 - Print 3D View of PCB
 - Add your parts in 3D to Lib
 - 3D Output in .3ds Format

**World's FIRST Schematic Based
USB Device Simulation Tool!!**

R4 Systems Inc.
www.r4systems.com
Toll Free 866.499.8184 info@r4systems.com

municate via the 8-bit SPI bus. The 16-bit IC is truly an exceptional product. The on-chip DSP makes this microcontroller ideal for number crunching PID control loops and general floating-point number crunching using IEEE 754 float-ing-point support.

The board is shown in Figure 3. It uses a standard DIP 40-pin IC, which makes the construction easier than the current SMT TQFP form factor. Wire-wrap techniques were used to make the original working prototype.

PCB DESIGN

The double-sided 3" x 4" PCB was not ready in time for this article, but you can see the progress I've made in Photo 4. Advanced Circuits made three boards for \$99. They did a great job and were a pleasure to work with.

As you can see, it's a high-quality board. The design and PCB layout was done using Cadsoft's Eagle CAD, which I highly recommend. It is possible to make a more compact USB motor controller board that includes the USB controller and the motor controller, but I have not had a chance to do so.

MOTOR DRIVER BOARD

The dsPIC30F4011 uses two PWM channels to drive the DC motors used on the tilt/pan platform using a 10-bit PWM

motor control channel to control the duty and frequency of the motors. The motor speed is increased or decreased by varying the PWM signal pulse width. The PWM output from the dsPIC30F4011 directly drives a dual-LMD18200T PWM H-bridge motor driver board that can handle up to 50 VDC and currents as high as 3.5 A (see Photo 2).

The H-bridge circuit in Figure 4 was used to build this board. I used heavy gauge wire-wrap, although it can be built using standard PCB techniques and components using "thick" traces with wide pads recommended for power signals that connect directly to the DC motors. This will ensure a very compact DC motor controller. A complete H-bridge circuit is required for each DC motor that will be used to control up to a maximum of two DC motors. The H-bridge circuits may be placed on the same PCB as long as there is room for heatsinks (attached to each LMD18200T H-bridge driver IC).

POWER SUPPLY

The USB cable supplies enough power for both controller boards, but I used a separate 5-V power supply for the dsPIC30F4011, the dual H-bridge board, and the 5-VDC motors. The motor supply is actually a rechargeable 6-V sealed lead acid (SLA) battery. Using higher-voltage 12- and 24-V motors

would require additional changes to the dual-H-bridge motor driver board.

MOTOR MESSAGES

In order to communicate effectively between the 8-bit PIC18F4550, the 16-bit dsPIC30F4011, and the 32-bit laptop processors, I defined the simple message data structures in Listing 1 in Ada95 and Listing 2 in Microchip C30 C. A checksum field and some simple checksum logic was added to validate messages sent and received.

Both files need to have data structures that are functionally identical to guarantee that the message gets across without data corruption; otherwise, the motor controller will ignore the message if the checksum sent does not agree with the checksum received. This feature should help the USB motor controller deal with the electrical noise generated by the motor's carbon brushes.

Each message is limited to 64 bytes to match the 64-byte USB endpoint buffer used in the Microchip USB drivers.

TEST MESSAGES

The USB motor controller has various hardware and software components that require testing before the controller becomes operational. They include the host GNAT Ada95 test application, which runs on the laptop.

The motor test application, running on the laptop, issues open-loop motor command messages to the USB motor controller. The messages gradually ramp the motor from rest to its fastest speed of 0 to 1,023 for the PWM duty cycle in order to slowly accelerate the motor from rest to its top speed. Then, it sends a reverse ramp 1,023 down to 0 to slowly decelerate the motor to a full stop. The test is repeated continuously until the application is terminated or another test is selected.

A motor-direction

Listing 1—This is an example of the GNAT Ada95 message data definitions required to communicate between the Ada application running on the laptop and the embedded Microchip USB application that runs on the PIC18F4550.

```

-- Message Motor buffer type
type MOTOR_BUFFER_TYPE is array (1 .. MAX_MOTORS) of win32.ULONG;

_*****
_*      Dmm,xxxx - Send PWM duty command message range xxxx (0000..FFFF) to motor
_*      mm range (00..01). This command is used to vary
_*      the motor speed and direction.
_*****
type DUTY_COMMAND_TYPE is
record

  Field_1 : Win32.WORD;           -- Start of message field                +1
  MessageID : Win32.WORD;        -- Message ID range (0000 - FFFF) Hex          +1
  Command : Win32.WORD;          -- Duty Command to SPI Slave                  +1
                                     -- (see enumeration declaration above.)
  NumberOfMotors : Win32.WORD;    -- Total number of motors range (0..MAX_MOTORS) +1
  Duty : MOTOR_BUFFER_TYPE;      -- Duty Cycle range (00000000 - FFFFFFFF) Hex +12
  EncoderCounts : MOTOR_BUFFER_TYPE; -- Encoder counts for each motor
                                     -- range (00000000 - FFFFFFFF) Hex          +12
  DirectionFlags : Win32.WORD;    -- Direction bits for each motor (CW=0, CCW=1) +1
  StatusFlags : Win32.WORD;      -- Status bits                                +1
  Filler : PADDING (1 .. 4);     -- Padding (total of 34 words)                +2
                                     --                                           ---
                                     --                                           32 Words
end record;

```


test slowly changes the motor direction from clockwise (CW) to counterclockwise (CCW) and repeats, causing the motor to oscillate forward and backward. This test is repeated continuously until the application is terminated or another test is selected.

The final test issues commands to tilt and pan the motors to a specific orientation by panning the sensor platform 0° to 360°, using the pan motor and tilting the platform ±90°, using the tilt motor with the 100-CPR optical encoders, and using 4× decoding to ensure accuracy.

PID CONTROL

The 16-bit (30-MIPS) dsPIC30F4011 is quite capable of running a proportional integral derivative (PID) control loop on its own using the on-chip digital signal processing (DSP) hardware, as the Microchip application notes show. Fred Martin's *Robotic Explorations: A Hands-On Introduction to Engineering* has a simplified PID example that provides insight into open loop, closed loop, and PID control of DC motors for robotics applications.

I also wanted to run PID loops from the laptop or PC host via the USB interface because my laptop is capable of 1.7-GHz processing speeds. It also provided me with the ability to experiment with advanced fuzzy logic control. In fact, Microchip has a reference design for a fuzzy logic-based air flow controller.

As for motor controller applications, Microchip came through again with a few PID-related application notes that can be directly flashed onto the board using the ICD2. The tachometer application in particular provides the user the ability to measure a motor's RPM, speed, and position. The high-level PID and fuzzy logic motor commands are received from the laptop as motor control messages.

Position, speed, velocity, and acceleration commands can be executed using closed-loop control, leaving the motor controller to maintain the PID control loops by flashing Microchip's PID algorithms for various kinds of motors directly to the dsPIC30F4011 motor controller using the ICD2 debugger/programmer. In order to make the tuning process easier, it is recommended that motors be select-

Listing 2—This listing shows how the corresponding C messages are defined that enable communication between the Ada application and the dsPIC30F4011 microcontroller via the PIC18F4550 microcontroller.

```

//*****
/* Dmm,xxxx - Send PWM duty command message range xxxx *
/* (0000..FFFF) to motor mm range (00..01). This command is *
/* used to vary the motor speed and direction. *
//*****
typedef struct duty_command_type
{
    word Field_1; // Start of message field

    word MessageID; // Message ID range (0000 - FFFF) Hex

    word Command; // Duty Command to SPI Slave (see enumeration
                  // declaration above.)

    word NumberOfMotors; // Total number of motors range
                        // (0..MAX_MOTORS)

    word Duty[MAX_MOTORS]; // Duty Cycle range (0000 - FFFF) Hex

    word EncoderCounts[MAX_MOTORS]; // Encoder counts for each motor
                                    // range (0000 - FFFF) Hex

    word DirectionFlags; // Direction bits for each motor (CW=0, CCW=1)

    word StatusFlags; // Status bits

    word Filler[17]; // Padding (total of 34 words)
} DUTY_COMMAND_TYPE;

```

ed with similar specifications to those used in the Microchip reference designs.

SOFTWARE DEVELOPMENT

The mandated Ada language was originally developed during the early '80s, starting with Ada83, a standard language to replace the legacy COBOL and FORTRAN languages and promote code reuse across various U.S. government agencies. Unfortunately, the language has fallen out of favor in the U.S. due in part to the new commercial off-the-shelf (COTS) mandate that leaves only C, C++, and JAVA for development because they are the languages widely used in commercial software development these days.

The latest language release is Ada95, although there is Ada2005 work going on in Europe where Ada is still widely used. Surprisingly, another area where Ada95 is being used is for programming the massively parallel super computers used in research and astronomy as a replacement for FORTRAN.

The GNAT Ada95 tool suite is freely available on the Internet. It provides all the necessary tools to generate robust Ada-based applications using the Windows XP environment in a similar manner to C# and Visual C++. A simple

motor test control loop listing that varies the motor duty cycle by issuing a ramp 0 to 1,023 to the motor is posted on the *Circuit Cellar* FTP site.

FIRMWARE

The embedded software is supplied in Microchip hex and Intel format for both the PIC18F4550 and the dsPIC30F4011 boards. The tool required to build and program the motor controller is the Microchip ICD2, which is available for around \$120. Also needed is the MPLAB IDE, which contains an in-circuit serial programmer (ICSP), a simulator, and an in-circuit debugger (ICD to flash and debug the applications).

The PIC18F4550's firmware was developed using the student edition of the PIC18 C compiler. The dsPIC30F4011's firmware was developed using a 60-day demo of C30 C. Both applications used MPLAB and the in-circuit debugger (ICD2) to flash memory and debug the applications. The C source code for both controllers is available on the *Circuit Cellar* FTP site. You may also want to download the dsPIC30 C compiler (60-day demo), which is needed only if you plan to customize or modify the C sources because I provide the necessary hex files.

On the PIC18F4550, you can see the section of code that handles the USB message received from the laptop and routes it to the SPI port, which then directs it to the dsPIC30F4011. The source code provides all the necessary communications and motor message processing required to make the boards function as an integrated USB motor controller.

The firmware application is written in Microchip C30 C on the dsPIC30F4011. Again, only the main loop is shown on the dsPIC30F4011, computing the message checksum, checking its validity, and processing the message.

MOTOR TEST BENCH APPS

The motor test bench application is the firmware from Microchip application note GS002 that is flashed on a dsPIC30F4011 using the ICD2. It enables you to measure a motor's position or angles, speed (tachometer), and direction (forward or backward) using the encoder counts generated by the QE1 peripheral from a dsPIC30F4011 motor controller board. Using my own versions of the

firmware, I have been able to successfully control DC motors in Open Loop mode using the PWM peripheral and return encoder counts using the QE1.

Ada95 HOST APPLICATION

The host application runs on the laptop and sends motor control messages to the USB motor controller using the COM3 serial port on my laptop, but it may differ on other machines. It is a work in progress using the Ada95 with the message data structures mentioned earlier. This application integrates the PIC18F4550 with the dsPIC30F4011. The application can be translated to any other computer language that supports the PC's serial ports (COM3 and COM4).

The Ada95 host application runs from the laptop as either a standalone Windows application *.exe or directly from the GNAT Ada95 IDE. (The motor control loop listing is posted on the *Circuit Cellar* FTP site.) The application uses other Ada packages that are not shown in the listing, but they are included with the source code.

The host motor controller applica-


tion issues PWM commands that move the motor from rest to its highest speed and then back down to rest by issuing a ramp 0 to 1,023 and then issuing a ramp 1,023 back to 0.

I also provide an example of an Ada95 exception handler that is used to handle run-time errors due to various factors. It includes an exception handler for bad user input and for the motor commands that exceed their limits using Ada constraint (range) checking.

This is an excellent feature where each Ada variable is checked against its initial minimum, maximum, and nominal ranges and also checks its type. A constraint error will be generated when the value exceeds 1,023 or is less than 0. In this case, the exception handler limits the value and displays a warning message to the operator.

TEST RESULTS

Connecting one or two DC motors with attached encoders to the JP1 and JP2 encoder headers tests the completed hardware. First, the motor PWM frequency and direction (forward and



CIRCUIT CELLAR®

back issues available as

**Searchable Archives
on CD-ROM**

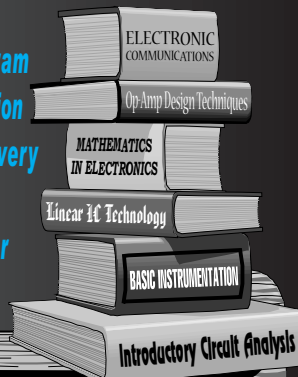
NOW SHIPPING:

- CD-ROM #11 2006 Issues 186-197
- CD-ROM #10 2005 Issues 174-185
- CD-ROM #9 2004 Issues 162-173

Order Online:
www.circuitcellar.com
or call 860.875.2199

PROFESSORS

The Circuit Cellar college program puts quality engineering information in the hands of your students every month. Sign up now to get Circuit Cellar distributed to your class this semester.



To update your professor account or to find out more about our college program, visit www.circuitcellar.com/products/collegeprogram/

backward) is specified and PWM commands are sent to the selected motor. It should start running at the speed selected. The motor ID value is required to select the motor being commanded. When I powered up the hardware, I saw the motors speed up, slow down, and stop as I expected. In addition, I sent the DC motor controller messages to change the PWM frequency (period) and messages to return each motor's state. These all worked fine in my application.

LOOKING AHEAD

As you can see, the USB interface provides a convenient gateway to the embedded USB motor controller hardware that enables a laptop or PC to issue high-level commands in order to harness the incredible gigahertz processing available in today's PCs and laptops using Microchip's elegant USB serial port solution. Now the laptop can be used for processing the algorithms, while the embedded controller handles the timing, reads the sensors, and generates the necessary PWM waveforms. GNAT and Ada95 can be used for developing the host application on the laptop, while the Microchip PIC18 C and dsPIC C30 C compilers are used exclusively to develop the embedded firmware.

Another idea that I am exploring (for the closed-loop PID control of two or more DC motors at the same time) is to use the USB and SPI interfaces via a PIC18F4550-based motor controller. I could then use it to connect a PC or laptop to two or more dsPIC30F4011 motor controller boards networked via the SPI bus and run multiple instances of rehosted PID closed-loop control using Windows NT, XP, or Linux using GNAT Ada, C++, and JAVA, which support multitasking or multithreading. The data is passed from the PC or laptop to the DC motor controllers via the USB bus using the motor messages. The motor ID is used to select which motor to send commands to and also to receive encoder counts feedback.

You can use the USB motor controller as a test-bed for learning about the latest PID and fuzzy control of DC motors. This article scratches only the surface of using a laptop to control external devices. In a future article, I

might go into more detail about using the Ada95 as an alternative development language for controlling embedded devices from a PC or laptop with emphasis on software safety. ☑

Daniel Ramirez is a senior software engineer at Raytheon with over 15 years of experience working on real-time embedded systems. He has a B.S. in Computer Science and an M.S. in Engineering from Northeastern University. His hobbies include watching old movies, antiques, travel, golf, photography, and Vex robotics.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2007/206.

RESOURCES

J. Barnes, *Programming in Ada: Plus an Overview of Ada 9X*, Addison-Wesley Publishing Company, 1994.

R. Graham, "FAQ on PID Controller Tuning," The College of New Jersey, www.tcnj.edu/~rgraham/PID-tuning.html.

G. Lucas, "Using a PID-based Technique for Competitive Odometry and Dead-Reckoning," *Encoder: The Newsletter for the Seattle Robotics Society*, www.seattlerobotics.org/encoder/200108/using_a_pid.html.

Microchip Technology, Inc., "PIC18F2455/2550/4455/4550 Data Sheet," DS39632D, 2007.

———, "dsPIC30F4011/4012 Data Sheet," DS70135E, 2007, ww1.microchip.com/downloads/en/DeviceDoc/70135E.pdf.

———, "AN956: Migrating Applications to USB from RS-232 UART with Minimal Impact on PC Software," DS00956B, 2004, ww1.microchip.com/downloads/en/AppNotes/00956b.pdf.

———, "AN600: Air Flow Control Using Fuzzy Logic," DS00600B, 1997, ww1.microchip.com/downloads/en/AppNotes/00600b.pdf.

———, "AN901: Using the dsPIC30F for Sensorless BLDC Control," DS00901A, 2004, ww1.microchip.com/downloads/en/AppNotes/Sensorless

[%20BLDC%2000901a.pdf](#).

———, "AN908: Using the dsPIC30F for Vector Control of an ACIM," DS00908A, 2004, ww1.microchip.com/downloads/en/AppNotes/ACIM%20Vector%20Control%2000908a.pdf.

———, "AN957: Sensorless BLDC Motor Control Using dsPIC30F2010," DS00957A, 2004, ww1.microchip.com/downloads/en/AppNotes/BLDC%20MC%2000957a.pdf.

———, "AN984: An Introduction to AC Induction Motor Control Using the dsPIC30F MCU," DS00984A, 2005, ww1.microchip.com/downloads/en/AppNotes/AC%20Induction%20Motor%2000984a.pdf.

———, "AN992: Sensorless BLDC Motor Control Using dsPIC30F2010," DS0092A, 2005, ww1.microchip.com/downloads/en/AppNotes/00992A.pdf.

———, "GS002: Measuring Speed and Position with the QEI Module," DS93002A, 2005, ww1.microchip.com/downloads/en/devicedoc/93002A.pdf.

———, "PICDEM FS USB Demonstration Board User's Guide," DS51526A, 2004, ww1.microchip.com/downloads/en/devicedoc/51526a.pdf.

SOURCES

Eagle Light Edition
Cadsoft Computer
www.cadsoftusa.com

RH-5A-5502 DC Servo system
Harmonic Drive
www.harmonicdrive.net

Vex robotics design system
Innovation First, Inc.
www.innovationfirst.com

DM163025 PICDEM USB demo board, dsPIC30F4011 DSC, PIC18F2455, PIC18F452, and PIC18F4550 microcontrollers
Microchip Technology, Inc.
www.microchip.com

LMD18200T Motor driver board
National Semiconductor Corp.
www.national.com

SW-P-WOC NightHawk camera
Swann Communications, Inc.
www.swannsecurity.com



Go Wireless Go Rabbit®

Check out our new Wi-Fi and ZigBee® core modules — the latest additions to our pin-compatible family.

RCM4510W
ZigBee RabbitCore®

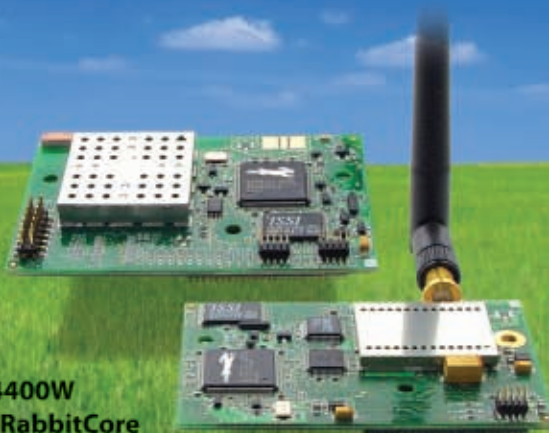
\$72 (qty. 100)

Wireless Development Kits

Only \$199 (reg. \$299)

RCM4400W
Wi-Fi RabbitCore

\$99 (qty. 100)



Applications

- Industrial Control
- Remote Terminal Unit (RTU)
- Building Automation
- Data Acquisition

RCM4500W Family

- ZigBee/802.15.4
- 49 GPIO
- 6 Serial Ports

RCM4400W Family

- Wi-Fi/802.11
- 35 GPIO
- 6 Serial Ports



2900 Spafford Street, Davis, CA 95618 Tel 530.757.8400

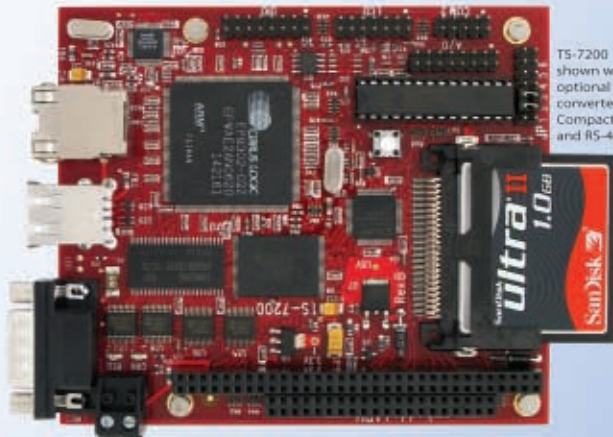
Order Your Kit Today

Development kits include everything to start your development, Dynamic C® software, wireless RabbitCore module, development board, programming cables and accessories.

www.rabbitkits-wireless.com

PC/104 Single Board Computers

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



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

- options include:
onboard temperature sensor, A/D Converter 8 channel 12 bit, Extended Temperature, Battery Backed Real Time Clock, USB Flash 256 M (with ARM Tool Chain), USB WiFi

200 MHz ARM9
Power as low as 1/4 Watt

- 5 boards, over 2000 configurations as low as
\$99
qty 100
- Fanless, no heat sink **\$129**
qty 1
- SDRAM - up to 128MB
- Flash - up to 128MB onboard
- 10/100 Ethernet - up to 2
- DIO lines - up to 55
- 2 USB ports
- COM ports- up to 10
- Programmable FPGAs
- Linux, Real Time extension, NetBSD
- SD card option
- VGA video

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



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

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

133 MHz 586

- 5 boards in series as low as
\$229
qty 100
- Power as low as 800mA **\$259**
qty 1
- Fanless, no heat sink
- SDRAM - up to 64MB
- COM Ports - up to 4 ports
- Ethernet Ports
- DIO Channels - up to 40
- PCMCIA II adaptor
- Compact Flash adaptor
- USB Ports (Except on TS-5300)

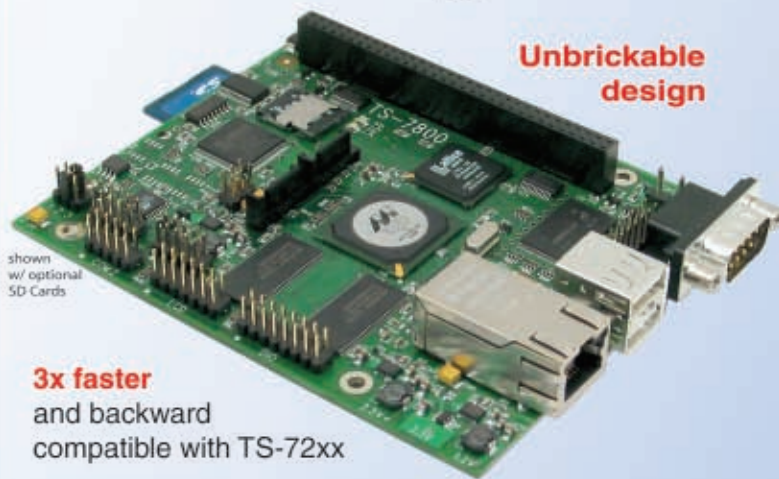
- Over 20 years in business
- Open Source Vision
- Never discontinued a product
- Engineers on Tech Support
- Custom configurations and designs w/ excellent pricing and turn-around time
- Most products stocked and available for next day shipping

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

New Products and PC/104 Peripherals

NEW!

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
- 12,000 LUT user-programmable FPGA
- Internal PCI Bus, PC/104 connector
- 2 USB 2.0 480 Mbps
- Gigabit ethernet
- 2 SD sockets
- 10 serial ports
- 110 GPIO
- 5 10-bit ADC
- 2 SATA ports
- Sleep mode uses 200 microamps
- Boots Linux in < 2 seconds
- Linux 2.6 and Debian by default

NEW!

TS-POE100
Power-Over-Ethernet

- 2.4 Amp 5V isolated power (12W)
- PC/104 peripheral w/ 10/100 Ethernet
- Sleep Mode & Wake-on-LAN with PoE



\$99
qty 1

board shown
with optional
16 bit PC/104
connectr and
2nd RS-485

NEW!

**TS-PLC Programmable
Logic Controller**

- Use standalone or cluster via wireless or RS-485
- 7 analog/digital I/Os
- opt. 500 meter wireless
- Program in ladder logic or C
- Web interface
- 4.7-30VDC or USB power input
- Rugged aluminum enclosure
3.6" x 3.1" x 1.2"

\$89
qty 1



see our website for more boards and option details

 **Technologic**
SYSTEMS

We use our stuff.

Visit our TS-7200 powered website at
www.embeddedARM.com

1-Wire in the Real World (Part 2)

The Solutions

Steve continues explaining how he turned a laboratory prototype into an electrically powered ice protection system for aircraft. He describes how he built a 1-Wire master and then covers some of the application structures that take full advantage of the 1-Wire bus.

Last month, I discussed some of the challenges associated with taking a laboratory prototype into the real world and turning it into what is now the ThermaWing system, which is an electric ice protection system for small to medium aircraft. This month, I'll put the pieces together and describe how to build a 1-Wire master by making the hardware and firmware play nicely. Then, I'll look at some of the application structures that build on that foundation to take full advantage of the 1-Wire bus (and the available slave chips) to address many of the application challenges.

FIRMWARE TECHNIQUES

Before I dive into the code, I will explain some coding techniques so you can understand the listings. I use slightly unconventional techniques that make for more readable and maintainable code. The techniques also make it easier to get it right the first time.

I've learned to write my firmware (and all of my design documentation) with the "cement mixer syndrome" in mind. That comes from a former boss who insisted that I always document my work just in case I get hit by a cement mixer and somebody else has to continue onward. As a consultant, I see the value in this approach every time I'm called in to solve a problem where the original developers are long gone. I sometimes get a call to do the next revision of a product that I designed for a company two acquisitions and a couple of years ago. The value of clear documentation becomes obvious at that point as well!

I have a few standard definitions that I automatically insert into either

a header file or directly into the C source, depending on the size of the project. C language defaults the common variable types to signed, but my need for signed variables in embedded designs is fairly rare. First, I create shorthand definitions for unsigned variables by prefixing a lowercase *u*, yielding *uchar*, *uint*, and *ulong* types. My *lengthof* macro is safer than *sizeof* to determine index limits for arrays, and it is lower maintenance than declaring a macro or constant for each size. It also works on multiply-dimensional arrays, arrays of structures, and so on. If I declare `ulong A [4] [7] [123] [12]`, then `lengthof (A [1] [2])` returns 123, the correct number of entries for the next index.

In any project involving communication between two devices, you will need to pick variables and structures apart into bytes. The *lobyte*, *hibyte*, *loword*, and *hiword* macros provide an intuitive and portable way to extract pieces of variables up to 32 bits long. As a bonus, most compilers allow them on

the left-hand side of an assignment. To adapt these macros when changing between Little Endian and Big Endian processors, you need only to swap the 0 and 1 subscripts. The *offsetin* macro uses the compiler to derive the exact memory offset to locate a particular item within a structure in a way that is guaranteed by the language to be consistent with whatever address assignment the compiler has generated. The macro is highly useful when you need to generate a numeric EEPROM address to be passed to *read* and *write* routines.

To allow temporarily suspending interrupts during a critical code section, the *SuspendInts* and *RestoreInts* macros provide a consistent way to save and restore the interrupt state and help keep the bookkeeping straight. Note the unmatched opening brace in *SuspendInts*. The opening brace enables you to declare a local auto variable where you save the interrupt context and forces the compiler to help you match it with the similarly unmatched closing brace in *RestoreInts*. You can circumvent

Listing 1—Here are some 1-Wire family codes. Note the use of a C language *enum* construction rather than a more conventional series of *#define* declarations.

```
typedef enum
{
    fcDS2401 = 0x01,
    fcDS2423 = 0x1D,
    fcDS2409 = 0x1F,
    fcDS2450 = 0x20,
    fcDS2438 = 0x26,
    fcDS18B20 = 0x28,
    fcDS2751 = 0x51,
    fcUnknown = 0xFF,
}
FamilyCodeT;
```


Listing 2—Check out the 1-Wire bit-level routines. Four macros handle all the microcontroller-specific manipulations. The ReadBit and WriteBit routines control the timing with very short delay loops that must be adjusted for clock speed and processor efficiency. Above this level all of the code is completely portable.

```
// Low-level 1-Wire Access
#define OneWireSet()      {OneWire = 1; OneWireTris = 0;}
#define OneWireClr()     {OneWire = 0; OneWireTris = 0;}
#define OneWireFloat()   {OneWireTris = 1;}
#define OneWirePullup()  {OneWireSet (); OneWireFloat ();}
// OneWire Bus Write a bit
// This routine specifically uses only the LSB of B
void OneWireWriteBit (uchar B)
{
uchar i; // used for delay loops; adj as needed for processor & clock rate
OneWireSet ();
SuspendInts;
OneWireClr ();
if (B & 1)
    {
    for (i = 5; i; -i);
    OneWireSet ();
    for (i = 55; i; -i);
    }
else
    {
    for (i = 55; i; -i);
    OneWireSet ();
    for (i = 5; i; -i);
    }
OneWireFloat ();
RestoreInts;
}
// OneWire Bus Read a bit
uchar OneWireReadBit (void)
{
uchar i; // used for delay loops; adj as needed for processor & clock rate
uchar Result;
OneWireSet (); // Ensure bus recovery time
SuspendInts; // Timing is critical here
OneWireClr (); // Leading edge of bit time
for (i = 2; i; -i); // Delay
OneWirePullup (); // Assist the passive pullup
for (i = 8; i; -i); // Delay
Result = OneWire; // Sample state of bit
OneWireTris = 0; OneWire = 0; OneWireTris = 1; // sample time marker
// useful for troubleshooting
for (i = 42; i; -i); // Wait for end of read slot
OneWirePullup ();
RestoreInts; // End of critical section
return Result;
}
// OneWire Bus Reset
// Return true in case of an error (no devices, or bus shorted)
bool OneWireReset (void)
{
uint i; // used for delay loops; adj as needed for processor & clock rate
uchar LastLow;
OneWireSet (); // Drive bus high to start
for (i = 240; i; -i); // Delay
SuspendInts; // Critical timing
OneWireClr (); // Send out reset pulse
for (i = 160; i; -i); // Delay
OneWirePullup (); // Recover to high state
for (i = 160; i; -i) // Watch for presence pulse
    if (OneWire == 0) break;
RestoreInts; // Done with critical timing
if (!i) // Handle exception case
    {
    printf ("No 1-wire devices\n\r");
    }
}

```

(Continued)

the protection by branching out of the construction in the middle, but being a good structured programmer, I wouldn't do such a thing, would I?

The C language enum declaration is underused. I use it in the downloadable 1-Wire code to provide a number of useful identifiers. For instance, Listing 1 shows how I declare the family codes, which comprise a portion of each 1-Wire serial number. Other declarations in the downloadable code provide readable identifiers for each of the 1-Wire ROM commands and function commands. I abhor the all-too-common lists of #define declarations to provide readable identifiers for port assignments, hardware registers, and so on.

When I declare an enumerated type that might be used as an array index or I have to iterate over the type or check limits, I declare the enum with an extra member specifying the number of valid entries, like I do with the Branches member of this enum that supports iteration:

```
typedef enum {brMain, brAux, Branches}
BranchT;
```

I can then iterate over the group with a construction (i.e., `i = 0; i < Branches; ++i`) and the compiler keeps everything neatly sorted out, even if I add or remove items in the enum.

Finally, I have a pet peeve against those who declare arrays using a plural name for the identifier. Although it's natural to call it "Devices" at the point of declaration because the programmer is thinking of a collection of information about all the "Devices," the declaration appears once, whereas the name is used many times in the code. At those numerous points of use, I am almost always referring to a single device, such as Device [3], which is then readily pronounced "Device 3." That is much closer to the real meaning than "Devices 3," which doesn't make grammatical sense. I reserve the plural form of the identifier for the count of how many entries are present in the array, most often when the number of valid entries changes dynamically (the lengthof macro takes care of the static case). Thus, I can iterate across

the array (`i = 0; i < Devices; ++i`). This consistency in variable naming helps keep everything straight.

LOW-LEVEL 1-WIRE ROUTINES

Listing 2 shows my implementation of the low-level routines to read or write a single bit on the 1-Wire bus. All code listings in this article are for the HI-TECH Software PICC-18 compiler, but they also port directly to other readily available embedded C compilers. If you are compiling for a Big Endian processor, such as one from Freescale Semiconductor, change the declarations for high/low byte/word (as noted in the comments) and check the compiler options to be sure that an enum will compile to a 1-byte object. There is no embedded assembly language, although the lowest-level routines include a couple of timing loops that must be adjusted for a specific clock rate and processor. Above the bit level, everything is completely generic.

Note that I cannot publish the complete application code. As a work for hire, it belongs to my client. The code I show here reflects my toolkit-level routines that I incorporated into the “work for hire” in the interest of saving my client time and money.

Sending a bit from the master to a slave is straightforward and matches the datasheet description. That is, for a “1” bit, drive the line low for at least 1 μ s and drive it active high within 15 μ s. For a “0” bit, drive the line active low for at least 60 μ s and then active high. In each case, you must observe the minimum datasheet time with the line high before driving the line low again to start the next bit.

To receive a bit, however, I deviate slightly from the datasheet recommendations and overcome the reflection and noise problems I described last month. The master drives the line low for 2 μ s to generate the start pulse of a read bit timeslot. It then drives the bus high for 1 μ s and finally releases the line to a floating state by setting the pin as an input. If the slave returns a “0” bit, the bus is momentarily in contention and at an intermediate (invalid) voltage, but neither the master nor a slave looks at the bus during this interval (see Photo 1). The slave is driving the

Listing 2—Continued.

```
        return 1;
    }
    // Wait for all devices to end presence pulse, with debounce
    for (i = LastLow = 500; i && i + 20 > LastLow; -i)
    {
        if (OneWire == 0) LastLow = i;
    }
    OneWirePullup (); // Clean finish
    if (!i) // Handle another exception case
    {
        printf (“1-wire bus stuck low\n\r”);
        return 1;
    }
    return 0; // Reset completed okay
}
```

line low during this time. As soon as the master releases the line, the active slave vigorously returns the bus to a valid low state, which remains stable throughout the master’s sampling time.

If, on the other hand, the slave returns a “1” bit, the master has already stabilized the line at a high-level voltage (see Photo 2). When the bus is released, it is already high and stable long before the master samples the state. You can gain the rapid settling time of a bipolar driver while still retaining the ability to share the bus among multiple units. Simply use a passive pull-up and active-low drivers, with only the master providing an active-high drive at exactly the right

times and never being fooled by a line reflection or noise.

This active pull-up method will corrupt a bus reset. Therefore, it is not usable during a reset. Many units can listen on the bus during a reset. If the master attempts to drive the line high and then releases it while a slave is still asserting the line low, other slaves may see the signal as the start of a read bit slot and things get completely wrapped around the axle. So, for reset, you must rely completely on the passive bus pull-up, resulting in the waveform shown in Photo 3. The slower bus-rise time causes no problem for reset because the reset opera-

Listing 3—Here are some byte transfers on the 1-Wire bus. Bytes are transferred LSB first. The Write routine shifts bits out of its LSB, while the Read routine enters bits into the MSB, shifting the previously read bits right as each new bit arrives.

```
////////////////////////////////////
// OneWire Write Byte
void OneWireWrite (uchar B)
{
    uchar i;
    for (i = 8; i; -i)
    {
        OneWireWriteBit (B);
        B >>= 1;
    }
}

////////////////////////////////////
// OneWire Read Byte
uchar OneWireRead (void)
{
    uchar i;
    uchar Result;
    for (i = 8; i; -i)
    {
        Result >>= 1;
        if (OneWireReadBit ()) Result |= 0x80;
    }
    return Result;
}
```

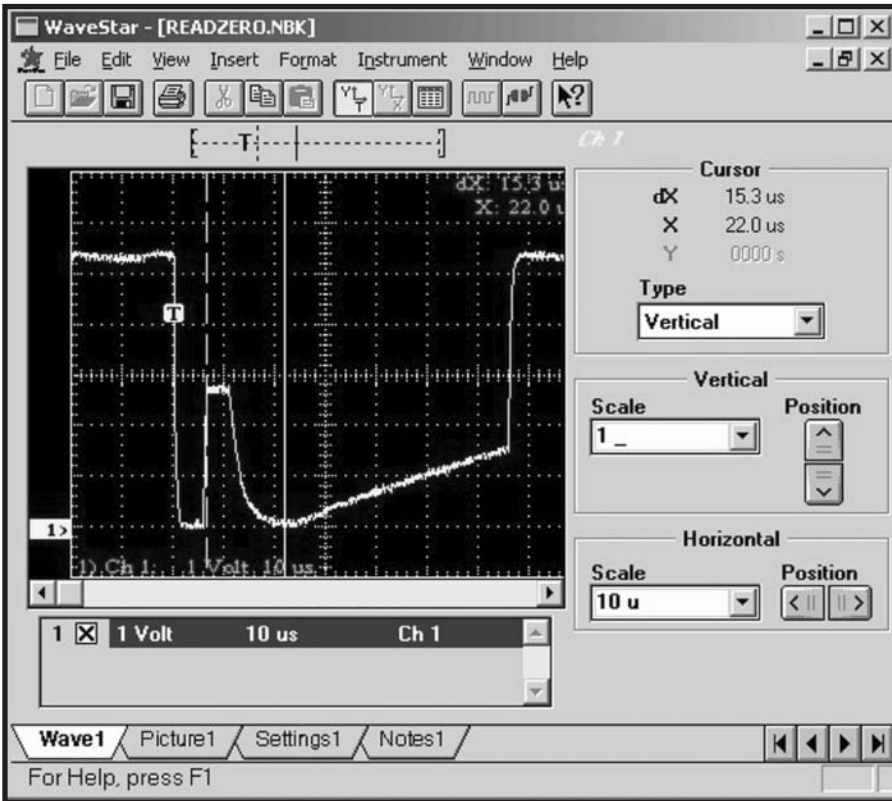



Photo 1—Note the bus contention for 6 μ s as the master provides an active pull-up. The slave returns the line to a solid “zero” at the sampling time marked by the right cursor, and the slow rise thereafter is provided by the weak passive pull-up. The bus is finally returned to a solid “one” condition as the master terminates the bit slot with an active pull-up.

tion is much rarer and longer than the normal bit slots.

BYTES AND MESSAGES

With the foundation in place, you can send a bit of either value and read a bit returned by a slave. Because the 1-Wire protocol performs almost all transfers as 8-bit bytes (least-significant bit first), the extension to reading and writing bytes is straightforward (see Listing 3). Messages are built of bytes, with some examples of specific message types shown in the code available on the *Circuit Cellar* FTP site.

ENUMERATION

The 1-Wire search algorithm is not as straightforward. All 1-Wire slaves have a unique 64-bit address and they all support the 1-Wire search algorithm. Because 1-Wire was designed to support the unannounced appearance and disappearance of units that may or may not have ever been seen previously, you need a way to search for an unknown device. Even if (and that’s a highly unlikely “if”) you could test 1 million addresses per second, a brute-force

search of the 64-bit address space would take almost 600 kiloyears. That’s a trifle longer than most of my designs will tolerate! The clever 1-Wire search algorithm supports searching the entire address space in such a way that completely enumerating all devices present on the bus can be done in less than 25 ms per attached device. The algorithm is described in the datasheets. More subtle points aren’t always obvious.

The heart of the 1-Wire search algorithm, along with the nitty-gritty exception cases that Murphy’s Law always throws in my face in the real world, are shown in Listing 4. In a nutshell, the master broadcasts a command to all units on the bus, announcing that you are searching for addresses. The master then provides two read slots. Each device replies with its first address bit in the first read slot, and the complement in the second read slot. Because the bus is passive high and active low, the zeros always win out in case there are two devices with different address bits. If the first read slot returns a zero, there is some device present and active on

PRESTO In-System USB Programmer



Very fast and flexible in-system programmer for Microchip & Atmel MCUs, serial EEPROMs & Flash PROMs, devices with JTAG (CPLDs, MCUs, FPGAs, FPGA PROMs, ...)

SIGMA - unique logic analyzer
USB modules - parallel & serial

ASIX ASIX, Prague, Czech Rep.
info@asix-tools.com

www.asix-tools.com

PIC-SERVO

MOTION CONTROL

MOTION CONTROLLERS FOR BRUSH, BRUSHLESS AND STEPPER MOTORS.

- controller chips
- controller boards

www.picservo.com
JEFFREY KERR, LLC

bob

basic overlay board

Introducing:
Vector graphics
Custom Font Capability
SPI or RS-232 Interface

Meet bob-4

Video display generator

www.decadenet.com

DECADE ENGINEERING
503-743-3194 Turner, OR, USA

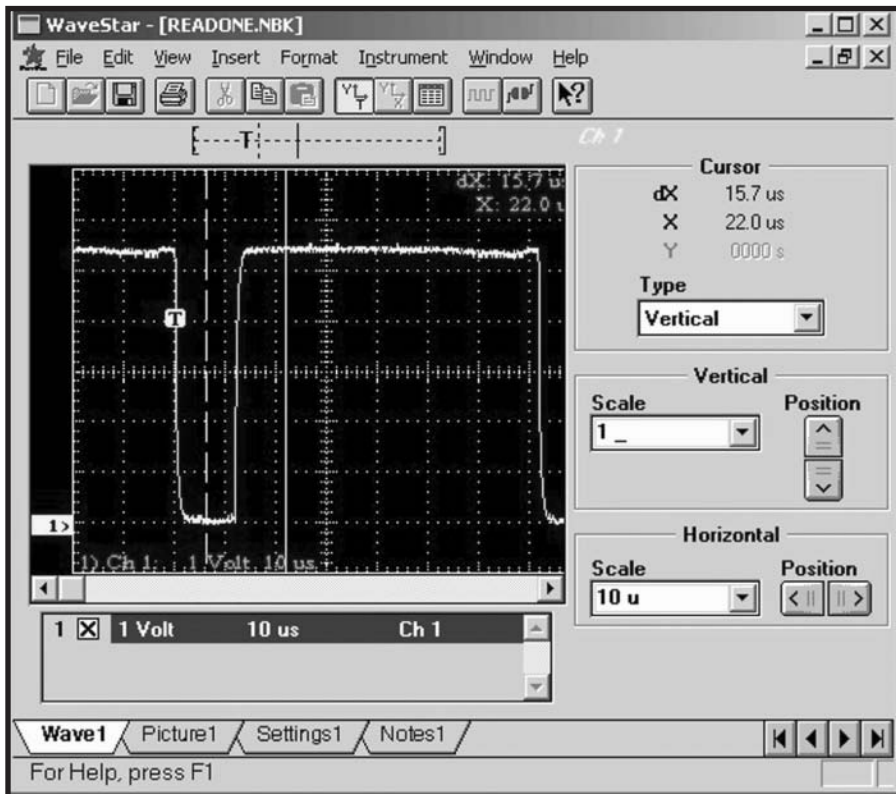


Photo 2—The active pull-up provided by the master guarantees a clean reading at the sampling point shown by the cursor.

the bus whose address has a zero in this position. If the second read slot (the complement of the address bit) returns a zero, an active device has a one in this bit position. If you look at the two bits another way, the master can detect four separate cases: 00 means at least one active device has a “zero,” and another has a “one” in this bit position. 01 means all active devices have a “zero” in this bit position. 10 means all active devices have a “one” in this bit position. 11 means no active devices are present on the bus (an error condition unless a device has been disconnected from the bus in the middle of the search).

The master then answers the 25 cent (2-bit) sequence by writing either a “zero” bit or a “one” bit to the bus. All slaves whose address matches the written bit remain active. All others go to sleep and do not further participate in the search. The master then issues two more read slots to search on the next address bit, repeating for each of the 64 bits. At the end of the search, the master has discovered the complete 64-bit address of one device on the bus.

The complete downloadable code implements the search with a Reset

Search routine, which establishes the initial conditions, and a SearchNext routine, which returns one complete address each time it is called. Although

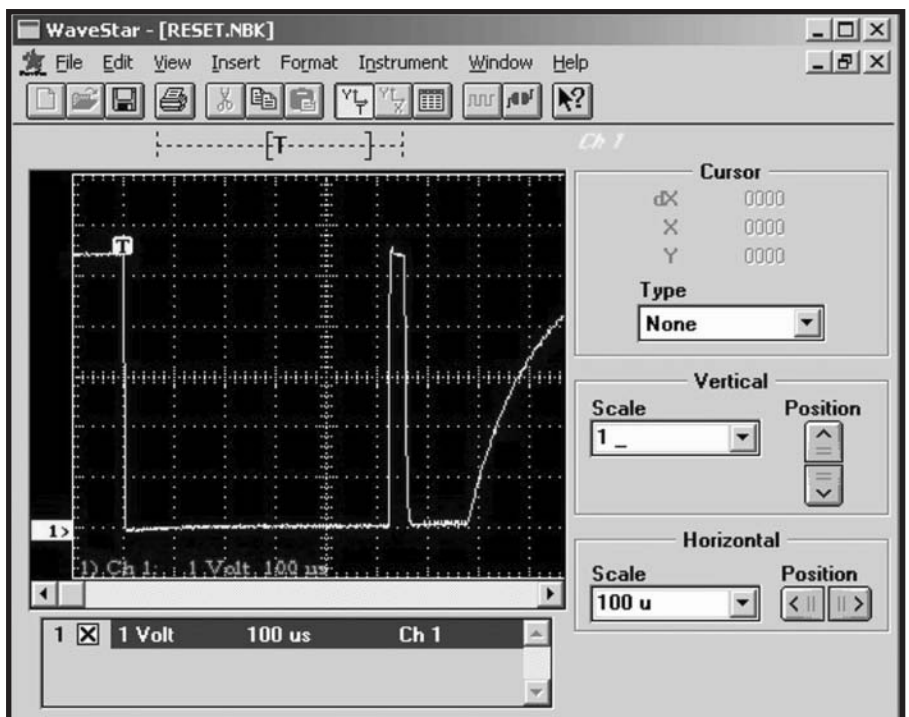


Photo 3—This is a reset waveform. Note the nice square 480- μ s minimum reset pulse from the master, terminated by an active pull-up. This is followed by the presence pulse from one or more slaves with the slow-rising, trailing edge provided by the weak passive pull-up.

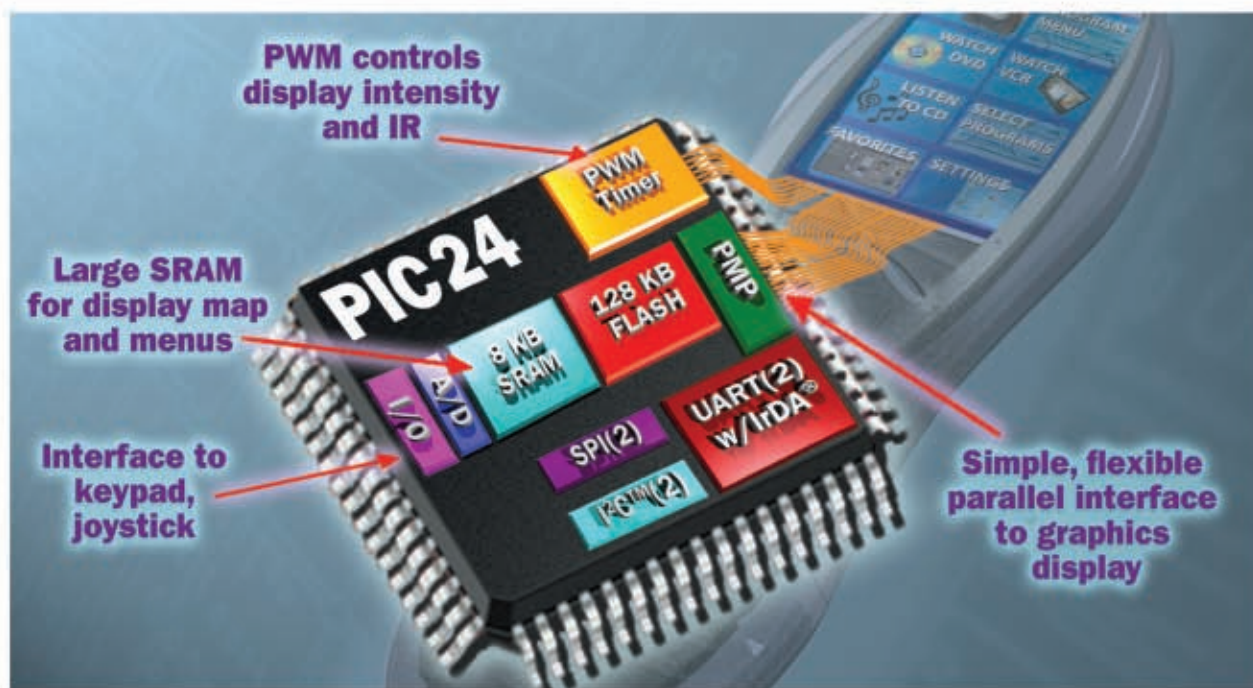
the algorithm is simple in principal, it’s dastardly difficult to get exactly right. The difficulty for me was in the bookkeeping required to ensure that I traversed every valid branch of the address tree without either getting stuck repeatedly traversing the same address or missing a branch.

After bench-testing the routines with a limited sample set of slave devices, I spent several days troubleshooting the wiring in the test aircraft before discovering that adding a device over here could disrupt the traversal in such a way that another device over there would disappear from the search. The clue that finally pinpointed the search problem was that attaching unit X would always cause unit Y to disappear, even when they were on separate buses. (The application implementation used four microcontroller port pins to provide four electrically independent buses.)

DISTRIBUTED CONFIGURATION

As I stated in my introduction, a key issue favoring the 1-Wire bus is a desire to avoid extra software-certification efforts to support different configurations. Therefore, I tried as much as possible to localize any information that

Control Your Remote with Low-Cost 16-bit MCUs



Whether your design is a home theater remote or any other embedded application, the PIC24F delivers the peripherals, performance, development tools and software you need.

With 16 MIPS performance and an extensive peripheral set, Microchip's PIC24F microcontrollers are highly cost-effective solutions.

GET STARTED IN 3 EASY STEPS!

- 1. Learn all about our 16-bit products in only 20 minutes!** Our *FREE* 16-bit web seminars highlight our 16-bit architecture and comprehensive support suite that includes low-cost development tools, a variety of software libraries (many free), application notes, reference designs and more!
- 2. Take advantage of *FREE* resources for download!** Our MPLAB® IDE Integrated Development Environment and a full-featured trial of the MPLAB C30 C Compiler from our web site gets you up and running fast!
- 3. Get hands-on technical training for as low as \$49!**



Attend a half- or full-day 16-bit course at a Microchip **Regional Training Center** to take your designs to the next level! Attendees also receive deep discounts on related development tools!

Purchase and program your 16-bit PIC24F devices and related development tools at...

microchip
DIRECT
www.microchipdirect.com

Device	Pins	Flash (KB)	PIC24F Family Features
PIC24FJ64GA006	64	64	8 KB RAM
PIC24FJ64GA008	80	64	Parallel Master Port
PIC24FJ64GA010	100	64	5 - 16-bit Timers
PIC24FJ96GA006	64	96	5 - Output Compare/PWM
PIC24FJ96GA008	80	96	5 - Input Captures
PIC24FJ96GA010	100	96	Real-Time Clock/Calendar
PIC24FJ128GA006	64	128	2 - UART with IrDA® and LIN Protocols
PIC24FJ128GA008	80	128	2 - SPI, 2 - I ² C™
PIC24FJ128GA010	100	128	10-bit ADC, 16 Channels
			2 Analog Comparators

Visit our web site for more information about additional 16-bit devices with higher performance and added features like DSP and enter our 16-bit Embedded Design Contest today!



MICROCHIP

www.microchip.com/PIC24

would be specific to a particular model.

The system design was generic to virtually all small aircraft, but there were small differences that I would have to accommodate for different aircraft makes and models. System limitations combined with the proprietary ice-removal algorithm to dictate the constraints on the minimum and maximum size of any one heater. Any aircraft model has a total amount of ice-protection area, as well as mechanical constraints on how that area can be serviced. For instance, a landing light can't be covered up by a heater, which means one aircraft may require six, nine, or up to 14 separate heaters for bigger aircraft. The heater characteristics dictate that the resistance of a heater will be different than another with different dimensions. The system measures the heater resistance to detect shorts, opens, or damage to the heater, so the software needs to know the nominal resistance of each heater for comparison. All the configuration information could be stored in the central controller's nonvolatile memo-

Listing 4—The core of the algorithm finds a single attached device by serial number. The routine iterates across the 64 bits of the address using a switch statement to sort out the various conditions possible with zero, one, or many slaves present on the bus and participating in the search.

```

////////////////////////////////////
// Search next device on the 1-wire bus
// Start from SearchSerial and search for next higher address
// Leave SearchSerial set to the found address
void SearchNext (void)
{
uchar Bit;
uchar ZeroDecision;
uchar Result;
ZeroDecision = 0xFF;
if (OneWireReset ())
{
ResetSearch ();
SearchComplete = 1;
return;
}
OneWireWrite (owSearchRom);
for (Bit = 0; Bit < 8 * sizeof SearchSerial; ++Bit)
{
////////// begin interrupt-blocked section //////////
SuspendInts;
Result = OneWireReadBit (); Result <<= 1;
Result += OneWireReadBit ();
switch (Result)
{
case 0: // both bits zero
// differing address bits at this position
if (LastZeroDecision < 8 * sizeof SearchSerial &&
Bit < LastZeroDecision)
{
if (!SearchBit (Bit)) ZeroDecision = Bit;
OneWireWriteBit (SearchBit (Bit));
}
}
}
}

```

(Continued)

COMPACT EMBEDDED SERVER



- x86 200MHz CPU
- 128MB SDRAM On Board
- 512MB CompactFlash™
- 10/100 Base-T Ethernet
- Reliable (No CPU Fan or Disk Drive)
- Two RS-232 & Two USB 1.1 Ports
- Optional Wireless LAN & Hard Drive
- Optional On Board Audio
- Dimensions: 4.5 x 4.5 x 1.375" (115 x 115 x 35mm)



2.6 KERNEL

Compact SIB
(Server-In-a-Box)
Starting at \$170.00
Quantity 1.

- EMAC Linux 2.6 Kernel
- Menu Drive Configuration Utility
- Eclipse Development Environment
- HTTP and FTP Servers
- PPP Dial In/Out Server & Client
- Telnet Server

Since 1985
OVER
22
YEARS OF
SINGLE BOARD
SOLUTIONS

EMAC, inc.
EQUIPMENT MONITOR AND CONTROL

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

EzPCB
Professional PCB Supplier

**High Quality
Competitive Price**

URL: www.EzPCB.com

Email: sales@ezpcb.com Tel: +86 139 1002 1704
HDI Up To 50 Layers, 2.5mil T/C, 0.1mm Hole Size
Other Products: Stencils, Keypads, Frontpanels,
Flex PCB, Enclosures, Turnkey Services

BitScope USB Mixed Signal Oscilloscope

Analog + Digital

Digital Storage Oscilloscope

- ✓ Dual Channel Digital Scope with industry standard probes or POD connected analog inputs. Fully opto-isolated.

Mixed Signal Oscilloscope

- ✓ Capture and display analog and logic signals together with sophisticated cross-triggers for precise analog/logic timing.

Multi-Band Spectrum Analyzer

- ✓ Display analog waveforms and their spectra simultaneously. Base-band or RF displays with variable bandwidth control.

Multi-Channel Logic Analyzer

- ✓ Eight logic/trigger channels with event capture to 25nS.

DSP Waveform Generator

- ✓ Built-in flash programmable DSP based function generator. Operates concurrently with waveform and logic capture.

Mixed Signal Data Recorder

- ✓ Record to disk anything BitScope can capture. Supports on-screen waveform replay and export.

User Programmable Tools and Drivers

- ✓ Use supplied drivers and interfaces to build custom test and measurement and data acquisition solutions.

Inventing the future requires a lot of test gear...

...or a BitScope



BS100U Mixed Signal Storage Scope & Analyzer

Innovations in modern electronics engineering are leading the new wave of inventions that promise clean and energy efficient technologies that will change the way we live.

It's a sophisticated world mixing digital logic, complex analog signals and high speed events. To make sense of it all you need to see exactly what's going on in real-time.

BS100U combines analog and digital capture and analysis in one cost effective test and measurement package to give you the tools you need to navigate this exciting new frontier.



Standard 1M/20pF BNC inputs



Smart POD Connector

Opto-isolated USB 2.0

12VDC with low power modes

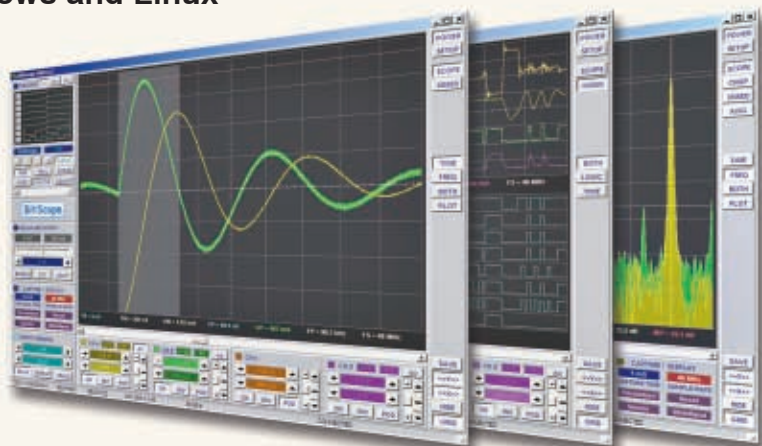
BitScope DSO Software for Windows and Linux

BS100U includes BitScope DSO the fast and intuitive multichannel test and measurement software for your PC or notebook.

Capture deep buffer one-shots, display waveforms and spectra real-time or capture mixed signal data to disk. Comprehensive integration means you can view analog and logic signals in many different ways all at the click of a button.

The software may also be used stand-alone to share data with colleagues, students or customers.

Waveforms may be exported as portable image files or live captures replayed on another PC as if a BS100U was locally connected.



www.bitscope.com

ry, but that would mean the controller could not be generic across all models of aircraft. Instead, I chose to place the configuration information in the nonvolatile memory of a Maxim Integrated Products DS2751 multichemistry battery fuel gauge, which is used to sense heater temperature, because it would be permanently attached to a particular heater.

Listing 5 is an excerpt from the code posted on the *Circuit Cellar* FTP site. The code enables you to use a high-level language construction, such as a C language structure, as an overlay on the memory space in a DS2751. The DS2751 includes a block of special function registers as well as two banks of uncommitted EEPROM and some static RAM. The typedef for DS2751RegT enables you to lay out the internal register arrangement as a C language structure. The structure can then be used as a member of a larger structure that may include an array of bytes covering EEPROM or other structure members defining specific allocations of variables. Thus, with care, I can declare structures and regular C language variables within the memory

Listing 4—Continued.

```

    }
    else if (Bit == LastZeroDecision)
    {
        OneWireWriteBit (1);
        SearchBitSet (Bit);
    }
    else // Bit > LastZeroDecision
    {
        ZeroDecision = Bit;
        OneWireWriteBit (0);
        SearchBitClr (Bit);
    }
    break;
case 1: // all active devices have 0 in this bit position
    OneWireWriteBit (0);
    SearchBitClr (Bit);
    break;
case 2: // all active devices have 1 in this bit position
    OneWireWriteBit (1);
    SearchBitSet (Bit);
    break;
case 3: // there are no active devices on the bus (error)
    printf ("\n\rNo dev at bit %d\n\r", Bit);
    ResetSearch ();
    SearchComplete = 1;
    Bit = 0xFE;
    break;
}
////////// end interrupt-blocked section//////////
RestoreInts;
}
if (ZeroDecision == 0xFF) SearchComplete = 1;
LastZeroDecision = ZeroDecision;
printf ("\n\r");
ShowOneWireSerial (&SearchSerial);
}

```

ZigBee™ Certified
Wireless Modules

XBee™ and XBee-PRO™

ZigBee™ Mesh Networks

- Range up to one mile
- Worldwide approved 2.4 GHz
- Modules start at \$19

Order your development kit online today! from \$129

MaxStream™
A Digi International® Brand

Toll-free 866-765-9885
www.zigbeemodule.com/cc

Wireless Control Application Kit

900 MHz and 2.4 GHz RF Control

- Low power LP3500 Single-Board Computer
- Two MaxStream® license-free RF Modules, FCC approved
- Software libraries and samples
- ModBus and PPP support for serving web pages

Complete Application Kit
\$599 2.4 GHz Version
\$499 900 MHz Version

RABBIT™ Semiconductor
 Order Online At rabbitappkits.com

07070

Listing 5—This is a C language construction for the DS2751 registers. This structure was extended in the final application to include configuration information stored in the DS2751's EEPROM. The structure declaration supports the use of the `offsetin` macro, described above, to generate a numeric address for issuance in a 1-Wire command or to read or write within the DS2751 address space.

```
typedef struct
{
    uchar   Reserved00;
    uchar   Status;
    uchar   Reserved02;
    uchar   Reserved03;
    uchar   Reserved04;
    uchar   Reserved05;
    uchar   Reserved06;
    uchar   Eeprom;
    uchar   SpecialFeature;
    uchar   Reserved09;
    uchar   Reserved0A;
    uchar   Reserved0B;
    int     Voltage ; // units of 152.588  $\mu$ V ( $\pm 5$  V FS)
    int     Current ; // units of 1.953125  $\mu$ V ( $\pm 64$  mV FS)
    int     CurrAccum; // units of 6.25  $\mu$ VH ( $\pm 204.8$  mVH FS)
    int     Reserved12;
    int     Reserved14;
    int     Reserved16;
    int     Temperature; // units of 1/256  $^{\circ}$ C ( $\pm 128^{\circ}$ C FS)
    int     Reserved1A;
    int     Reserved1C;
    int     Reserved1E;
}
DS2751RegT;
```

map of such an external device.

The primary caveats to this approach are that the compiler issues no warning if your structure exceeds the available memory or if it is misaligned with the actual registers, and that different compilers may use different physical memory sizes for the same variable types. All the compilers I've used on small embedded projects, however, treat a `char` as 8 bits, an `int` as 16 bits, and a `long` as 32 bits. Where a variable is larger than 1 byte, the programmer also needs to be aware of the allocation strategy used by the compiler to create multibyte variables. Such a variable can have the least significant byte at the lowest memory address or the most significant byte at the lowest memory address. These arrangements are commonly referred to as Little Endian or Big Endian. Processor architecture guides but rarely mandates the choice. Compilers targeting Freescale Semiconductor processors typically use Big Endian, while compilers targeting Microchip Technology processors usually use Little Endian.

As a side note, the technique is highly useful any time I need to allocate addresses in a separate address space. I can use this method to allocate

space for variables and clean, readable access to those variables. One example is EEPROM that is either at a specified offset within the main address space (as on the Freescale Semiconductor microcontrollers) or that lives in a completely independent address space (as on the Microchip microcontrollers). Another is memory-mapped or even I/O-mapped registers. Also note that my "offsetin" macro provides a portable method of converting a structure member's name into a numeric address offset to be passed to the 1-Wire bus, to an EEPROM access routine, or as a numeric port address.

The device system application includes an expanded version of the DS2751's structure covering the nonvolatile memory included in the chip. Additional fields are included there to specify the resistance, maximum power, actuation sequence, manufacture date, and other information pertinent to the switching of the particular heater. One oddball configuration item is the total number of heaters expected in the system. Unlike the other parameters that are specific to one heater, this parameter applies to the system as a whole and not to any one heater. I decided to store the expected number of heaters in every switch and to verify, as the system is ini-

tialized, that all modules agree on the expected number of heaters. Any disagreement clearly indicates a problem, whether it's an installation error or corrupted memory. As long as all the switches agree on the number of switches that should be present, the controller can signal a fault if the configured number of heaters doesn't appear during the search of the 1-Wire bus.

IMPROVEMENTS

The 1-Wire communication protocol seems to lack a reputation for reliability. With proper techniques in the master end, however, it can be made quite reliable even on long wires in a hostile electrical environment. The added benefits of a guaranteed (by Maxim Integrated Products) unique serial number for traceability and distributed configuration information storage made 1-Wire an ideal fit for this application. Someday I'll get one of those "round tuits" and be able to implement a few of the dozens of 1-Wire projects that I keep thinking of as I read the newsgroups! ☎

Steve Hendrix, P.E., CFIAI, lives with his wife Kathy and five children, ages three through 21, next to Cuyahoga Valley National Park in Ohio. When he isn't busy hiking with them in the Park, leading Boy Scouts, foster parenting, or ringing handbells, he runs Hx Engineering, LLC. Steve has designed software and hardware for various projects: Space Station Freedom, vibration analysis, IEEE-488 bus hardware, biomedical sensing (including wireless EEG, EKG, and EMG recorders), industrial instrumentation, and phototherapy equipment.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2007/206.

SOURCES

PICC-18 Compiler
HI-TECH Software
www.htsoft.com

DS2751 Multichemistry battery fuel gauge
Maxim Integrated Products, Inc.
www.maxim-ic.com

Resilience in Embedded Designs (Part 1)

Power Supply, Inputs, and Ground

In this series of articles, Aubrey describes techniques for minimizing the chances of problems occurring during the embedded design process. He also outlines the major causes of system failure, such as power supply noise and electromagnetic interference.

As Carl Sagan might have said, there are millions and millions of embedded systems in use today. If you consider how many of them suffer errors or faults, the percentages would be infinitesimal. Yet, every embedded designer has experienced a situation during which a microcomputer has stopped working. This can be permanent (fault), which requires hardware repairs, or transient (error), which is cured by resetting or power-cycling the system. In this series of articles, I will describe ways to approach embedded design that minimize the chance of problems (a fault or error) and allow for graceful recoveries when faults occur.

During development, any detected failure should be interpreted as a design flaw. The design should then be investigated and modified to alleviate the problem. Systems should be tested under all possible circumstances, with all combinations of inputs, like a normal development process. The techniques I'll describe here are not intended to be used as cures for these problems. They should be implemented as a means of protecting your system against the unforeseen.

What are the possible sources of the unforeseen? Unfortunately, one of the main culprits is the designer himself, especially when he has not catered to a particular set of circumstances because of either myopia or poor specifications. This kind of fault is usually repeatable, or at least occurs frequently, and requires a design fix. Power supply noise, earth loops and earth "bounce," electromagnetic noise, and RF can also be the source of system failure.

The last and least significant of the

possible sources of unforeseen error is radiation, which is caused by either an emission of a subatomic particle within the epoxy of the IC or external sources. The particles alter the state of some memory or register bits, which can cause the microcomputer's execution path to be erroneously altered.

The degree to which you need to protect your system depends largely on the consequences of system failure. Creating a pleasing LED display for home use hardly needs the same degree of vigilance as the antilock braking system of a car. Consider the following ideas and suggestions in that context. When

designing, try to consider the worst possible set of circumstances and attempt as many tests as possible, especially in the actual working environment.

POWER SUPPLY

The European Union's CE certification (EN61000 in particular) has forced all of us to consider input protection (although we always should have done this) to protect against damage to our equipment and a temporary malfunction. Two of the possible approaches to protect against overvoltage include using a metal oxide varistor (MOV) or a transient voltage suppressor (TVS) at the input. When an MOV is activated as a result of overvoltage, it shorts to ground. The TVS clamps to a fixed voltage. Either way, there is no current limit, which can result in burnt tracks and wires and other component failures. When considering a power supply to a board, the easiest solution is to put a fuse in series with the supply line. Lower input currents could use series resistors to limit the current, but I will defer this discussion to the protection of inputs in general. There are other surge-suppression techniques like the gas discharge arrestor. Each device has its pros and cons and they are occasionally used together.

When using a DC supply, it is a good idea to protect the input to your board against reverse voltage, even though it may only be a one-time event. This could be as simple as a series diode (see Figure 1a). Consider the power that this diode will dissipate and the volt drop that occurs. If this is a problem, the circuit in Figure 1b is a solution because the TVS will forward bias if there is an

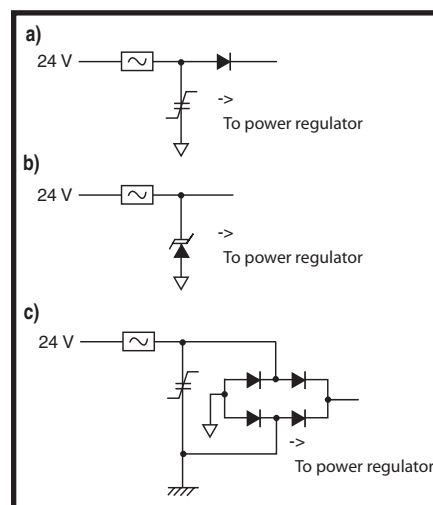


Figure 1—In the first (a) and third (c) diagrams, an MOV will short to ground if a voltage surge is seen. This prevents the overvoltage from being applied to the rest of the electronics. In the second diagram (b), a TVS is used, which clamps the voltage with the same result. **a**—A reverse current, due to an applied reverse voltage, is prevented by the diode. **b**—The reverse voltage is short circuited by the TVS and will blow the fuse. **c**—A voltage of either polarity may be used because it will be rectified by the diode bridge.

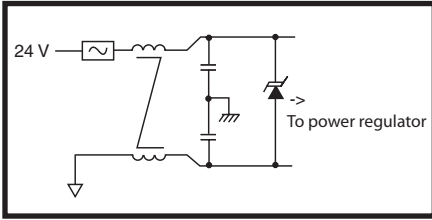


Figure 2—An input choke can reduce input noise and provide overvoltage protection. This is necessary only in the most extreme conditions or to meet regulatory approval. In smaller projects, you can resort to simpler techniques, such as an MOV or TVS, and perhaps a single choke or a simple bead.

erroneous connection without the drawback of a series diode drop. Make sure there is current limiting. A diode bridge allows for a supply of either polarity, but there is a potential drawback that I will cover in the ground-loop discussion later (see Figure 1c). Make sure you have a clean supply to minimize problems in the rest of the circuitry. In extreme cases, you can filter through a coil at the input with capacitors to the chassis/earth ground (see Figure 2). As obvious as it seems, many people do not design their power supply regulators for worst-case conditions. First, there is the input-voltage variation. If the input voltage falls too low, the output voltage can be rather unpredictable. Too high, and the circuit can fail. This is especially true with an AC-rectified supply.

Second, power dissipation can also be an issue. If the heatsinking of the supply components is insufficient, intermittent shutdowns can occur because the electronics heat up and fail, then cool and recover. Even at fairly low currents, a linear regulator can get hot. (Refer to the “Heat Dissipation” section in ResilienceReferences.pdf, which is posted on the *Circuit Cellar* FTP site). Elevated temperature is not your friend. Properties of any component are affected by temperature. The reliability of electronic devices can be compromised by heat, especially electrolytic capacitors, which can warm up because they are usually close to the power electronics. The mean time between failure (MTBF) of an electrolytic capacitor is halved for every rise of 10°C.

Avoid long traces if possible and ensure the tracks are wide enough to carry the maximum current. Digital circuits tend to have surge demands when switching and each device should be

decoupled with a ceramic 0.1- μ F capacitor for every IC, and a 1- μ F tantalum (or similar) for every four to five ICs, all placed as close as possible to the associated IC. Analog circuits can and will oscillate without similar decoupling.

PROTECTION

When designing protection for circuitry, you may need to consider protection for one-time events. I have seen a spec that called for protection in case the electrician installing the RS-485 network cable accidentally connected it to 120 VAC. It also may not occur to you that pulling cables through a conduit can create significant static electricity. It dissipates into your circuit when first connected and is never an issue again. (Refer to the “Static Discharge from Wire Installation” section in ResilienceReferences.pdf.)

When the voltage on an input is greater or less than the supplies of the device, a current will normally flow into or out of the device’s input. Permanent damage is likely to occur, unless the resulting current is limited. As covered in the “Power Supply” section, you need to consider surge protection not only to meet regulatory approval, but also to protect the product. Two of the principle ways of doing this are by simply placing an MOV or a TVS at the input. Many components are said to meet the CE requirements. Some integrated circuits even include surge suppression to meet the standard. It is one thing to design for the specifications, but from my experience, reality tends to be a little harsher. Repeated stressing of an MOV results in the degradation of the part until it finally fails open circuit. The TVS, on the other hand, will fail short circuit. I don’t know which is worse. The former enables the unit to continue operating unprotected until it is destroyed by a voltage surge. The latter immediately crashes the system, loading the power supply or the signal source, and possibly damaging them. Fortunately, there are ways to protect these devices, ironic as it may seem, to protect the protection.

When considering energy dissipation, size is everything. Don’t expect an MOV in a 1206 package to be able to dissipate the same amount of energy as an S09 package. Bigger is better. You can limit the current that the

surge suppressor is passing by using a current-limiting resistor (see Figure 3).

There are several factors to consider when selecting the resistor. You need to remember the power that will be dissipated (again the size issue). You also need to consider the technology of the resistor. For high-voltage surges, most regular resistor technologies will simply act as a short circuit. The old-style carbon composition resistors are ideal for this purpose, but newer technologies exist. (Refer to the “Surge Suppression Resistors” section in ResilienceReferences.pdf on the *Circuit Cellar* FTP site.)

Other possible techniques include using a PTC resistor instead of a regular resistor (see Figure 3). Initially, the impedance is low, enabling normal operation. When a surge occurs, the device heats up as increased current passes through it. That results in a resistance increase and limits the current. Of course, the resistor must be rated for the surge voltage and the surge suppressor. The PTC resistor must be capable of dissipating the energy. The PTC resistor could be replaced by a device from Tyco Electronics’s Poly-switch range. It effectively goes open-circuit on overcurrent and then resets.

You could also use an NTC resistor in series with a fuse together with the surge suppression. The resistor limits the initial surge current. As it heats up, the resistance drops, causing a higher current in a positive feedback approach. As the current gets beyond the fuse rating, the fuse blows and protects the circuit with the complication of having to diagnose a fuse fault and replace it (see Figure 4).

There are several other devices (and new ones all the time) that could be used

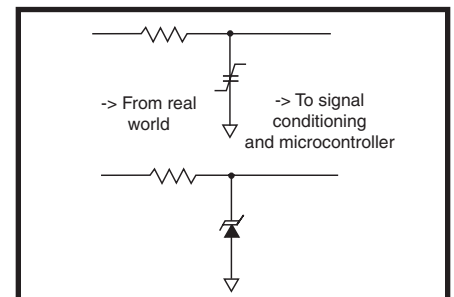


Figure 3—When an MOV or a TVS starts to conduct, very high currents can result. This can lead to the failure of the device or the wiring and tracks in the circuit. One of the techniques to protect the device is to use a current-limiting resistor (as shown). However, resistor technology is also important.

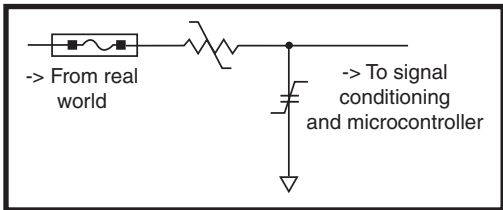


Figure 4—Protecting an MOV using an NTC resistor and fuse will result in more severe surges blowing the fuse. Lesser surges will be clamped without disrupting operation, which will require a fuse replacement.

in these forms of protection. The Analog Devices ADG465 is like the Polyswitch in its approach and the SP720 from Littelfuse has multiple TVSs in a single IC package to protect several signal lines.

OVERVOLTAGE PROTECTION

Overvoltage protection is different from surge protection because the elevated voltage is normally maintained for much longer periods and may be the continuous condition. It is also intended to protect the inputs against misconnections, such as inverted voltage or 120 VAC connected to a 5-V input. Many industrial systems operate at 24 VDC, while the electronics operates at 5 VDC. In this circumstance, we can clamp the voltage at the input to the electronics and limit the current by an input resistor. Two such possibilities are shown in Figure 5.

The Zener diode clamps the positive input voltage to 4.7 and 0.7 V for a negative input voltage. The resistor must be sized for the correct power dissipation. If you are catering for surge as well as overvoltage, consider using a carbon composition resistor here, and remember that the clamp time of a Zener is slower than the TVS, so the Zener could be replaced with a TVS.

A second alternative is to use two Schottky diodes. The diode to V_{CC} will clamp the positive input voltage to $V_{CC} + 0.3$ V, while the diode to ground will clamp a negative input voltage to -0.3 V. The remarks for the resistor specifications remain the same as in the previous paragraph. Typical electronic specifications limit the input voltage to between $V_{CC} + 0.6$ V and $V_{SS} - 0.6$ V; hence the use of Schottky technology to ensure the input remains within the spec. The 0.6-V levels are because there are internal diodes (parasitic and intentional), which can be forward biased. If there is

no current limit, the device can suffer permanent damage. You can often use just these internal diodes with sufficient series resistance to limit the current.

These techniques are true for both analog and digital inputs. Schottky diodes, however, can have a high leakage current that deteriorates with temperature. The input circuitry of some ADCs can interact with this leakage current to provide errors. The A/D inputs may not be designed for high-value series resistors because they can affect the settling time or contribute to voltage droop, so pay attention to the datasheets. Forewarned is forearmed.

Overvoltage level translation can be achieved by using comparators, optocouplers, magnetic coupling, and capacitive coupling. These techniques are also associated with electrical isolation, which I will get to later.

EMI FILTER

You can also protect against RF interference by using an EMI filter on inputs (and outputs if necessary). These normally consist of some form of inductance followed by a capacitor to ground and/or to earth as well. Obviously, the inductor provides high AC impedance and the capacitor shorts whatever gets through to ground. Less effective, although often sufficient, is an RC low-pass filter where the coil is replaced by a resistor. Enclosing the PCB in a shielded metal housing can magically resolve an unexplained effect.

INPUTS

High-voltage inputs (greater than approximately 200 V) can actually find a path to signal return as a result of impurities and conductance of the board. This results in a leakage current and an error is measured. A guard band (normally earthed) around the track can cure the problem.

On an associated note, be careful to define inputs on all CMOS circuitry. Floating inputs can lead to some bizarre effects, including excessive current consumption and even catastrophic failure in some cases. This also applies to microcontrollers. Define unused I/O pins as outputs and make sure that unused inputs have

pull-up resistors.

GROUND LOOPS

It's very common in industrial environments to have different ground voltages between electronic systems. The first aspect I would like to deal with is the classic ground potential difference from one part of the factory floor to another. As a result of the difference in ground potential over even a few feet, many designers insist on using isolation whenever possible. But, there are some alternative techniques. There are times when isolation is essential, even if it is not a safety issue, because there can be a difference of tens of volts in the earth potential over longer distances.

The second aspect is as a result of misunderstanding or poor specifications. As an example, the building automation industry runs on 24 VAC with one side of the transformer connected to ground. A standard DC voltage-generation technique is to full-wave rectify this AC voltage. The resulting ground on the PCB is a diode drop above system ground. Any circuitry brought onto the board must be aware of that fact. For instance, many thermocouple junctions are grounded. If the board assumes one side of the thermocouple is connected to common, you have an immediate earth loop that will "smoke" the bridge and possibly other components. There are a number of solutions. The easiest is to work with half-wave rectification, but you can also use differential voltage techniques or opti-

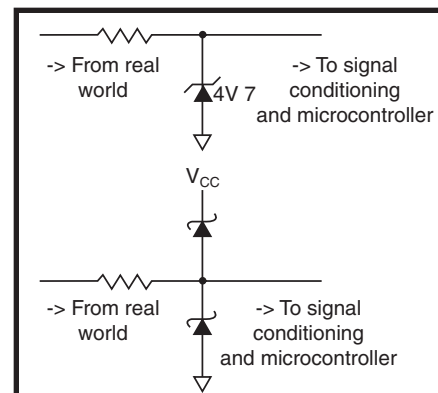


Figure 5—Clamping voltage inputs can be achieved in two ways. The tolerance of the Zener diode voltage must be considered since a 10% part could result in a voltage of higher than 5 V. The Zener diode protects against a negative input voltage. Diodes or Schottky diodes can be used to clamp the input voltage, but they may also have drawbacks. Take care not to force too much current into V_{CC} because some voltage regulators cannot sink current.



SAVE \$300
Introductory Offer

DENSER CODE
better performance

PROPSOC[®]
for
with OMNISCIENT CODE GENERATION[™]

HI-TECH PRO for PSoC ANSI C COMPILER, from the company that has, for over two decades, delivered the industry's most reliable embedded software development tools, comes with:

OMNISCIENT CODE GENERATION

Extracts information from multiple source files simultaneously, allowing more intelligent state-of-the-art code generation that:

- Optimizes the size of each pointer variable in your code based on its usage;
- Eliminates contention for the PSoC index register;
- Eliminates the need for many non-standard C qualifiers and compiler options;
- Produces more efficient interrupt context switching code;
- Automatically analyzes user assembly and object code files; and
- Customizes the functionality of the `printf` library function.

DENSER CODE, BETTER PERFORMANCE

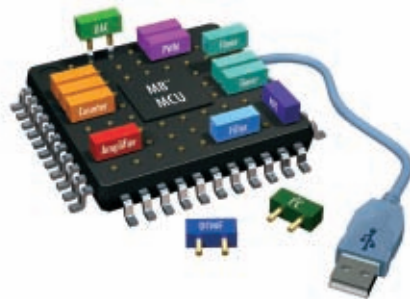
Code compiled with HI-TECH PRO for PSoC can deliver up to 50% denser code than its competitor.

FREE 45 DAY EVALUATION

A fully functional 45 day trial version of HI-TECH PRO for PSoC can be downloaded, free of charge, at www.htsoft.com/portal/ccpsoc.

HI-TECH PRO FOR PSoC PACKAGE INCLUDES:

- PSoC Designer[™] - Cypress Semiconductor Corporation's free IDE for PSoC;
- HI-TECH PRIORITY ACCESS[™] - 12 months access to updates and technical support; and
- HI-TECH SATISFACTION GUARANTEE - a hassle free 30 day money back guarantee.



PSoC[®] is a registered trademark of Cypress Semiconductor Corporation.

PSoC Designer[™] is a trademark of Cypress Semiconductor Corporation.

HI-TECH
SOFTWARE
HI-TECH Software LLC

6600 Silacci Way Gilroy, CA 95020 USA

Ph: 800 735 5715 Web: <http://www.htsoft.com>

cal/magnetic/capacitive isolation.

OPTOISOLATORS

For digital I/O, perhaps the most common method of achieving isolation involves using optical isolators. In addition to the isolation, however, the optoisolator can be used to translate voltage levels and provide surge suppression, even if the input and output grounds are common. At times, optocouplers are used to isolate AC voltages or are required to withstand reverse applied voltages. This

can be treated with a reverse diode across an LED of an optocoupler.

There are several issues to consider. As an input, you need to be concerned about surge/overvoltage damage to the LED. When configured as a system output, the transistor is also susceptible to surges and short circuits. It does not have much in the way of drive current, so it will need buffering. I don't think this is news to you. However, there is a "gotcha" that requires careful reading of the datasheet and attention to the subsequent design.

In creating a versatile interface, I often choose devices with high gain and then use low output currents. Optocouplers can suffer from an effect called "dark current," where there is a leakage current that appears at the output. The current increases with an increase in ambient temperature and soon you can have false inputs.

There are also analog optoisolators such as the Clare LOC110 linear optocoupler. In order to improve the linearity of the device, they normally require additional op-amps to drive them, so the protection aspect is lost because the input amplifiers will need protection.



5th International System-on-Chip (SoC)

Conference & Exhibit

November 7 & 8, 2007

Radisson Hotel Newport Beach, California

CPU/DSP Cores, Multicore, Embedded Memory, Semiconductor Advances, Low-Power Issues, Chip/System Architectures, New Design & Development Tools, EDA-related Issues, Multimedia IPs, High-Speed Interfaces, NoC, Challenges with Analog Design, SoC/ASIC/ASSP Design issues, FPGA & Foundry, and Much, Much More . . .

www.SoCconference.com

***The Most Targeted & Informative
System-on-Chip (SoC)
Conference & Exhibit
Event of the Year!
Don't Miss Out!***

**For Information and Questions,
Please Contact SoC Conference Organizing Committee:
SoC@savantcompany.com
949-851-1714**

www.SavantCompany.com

OTHER ISOLATION TECHNIQUES

There are three other techniques used in isolation. Magnetic isolation is normally used for analog isolation while the capacitive approach, in addition to the magnetic one, is used for digital interfacing. The capacitive approach is becoming more popular because it allows for faster transmission rates. The third approach uses relays. Some exotic options include iCoupler (magnetic) isolators from Analog Devices and high-speed isolators from NVE.

SIGNALS & VOLTAGE

It is very common in electronics for a signal to ride on top of a constant common-mode voltage. This voltage can be the result of the powering of a sensor (as in a Wheatstone bridge) or as a result of the ground differentials.

In analog circuitry, there are two input configurations that can be used. The first is called a difference amplifier and the second is an instrumentation amplifier (IA).

A difference amplifier has unequal input impedances, but its inherent design often allows for input voltages outside of the supply voltage. Some specialized products take this to an extreme and enable you to avoid isolation. Texas Instruments's INA148 can handle a common-mode voltage of 200 V.

The IA is normally a precision device, and you can expect it to be priced accordingly. You could make your own from two or three op-amps, but the noise performance would be significantly worse than the commercial product, unless you have extremely good matching of the resistors.

Some electronic systems are not referenced to ground and are floating. To interconnect between systems, you need to establish a common reference unless you use some form of isolation. Sometimes all this needs is a resistor of several kilohms between the grounds if no current is passing. At other times, you will need the isolation techniques discussed above.

The RS-422/485 specification includes a provision for common-mode voltage differences between communication nodes of 12 to -7 VDC. One man's input is another's output, so the topic of a general RS-485 interface serves as a good segue into next month's discussion about RS-485, resilient outputs, and watchdogs. Looking even further ahead, I will discuss resilient software in Part 3 of this series.

This article has touched on many details; unfortunately, there is not enough magazine space to discuss them fully. Most of the material is available on the Internet. Be sure to read through the ResilienceReferences.pdf file on the *Circuit Cellar* FTP site. It includes a list of these documents and a few additional comments. ☺

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

PROJECT FILES

To download ResilienceReferences.pdf, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2007/206.

RESOURCES

L. Chang, "Cable Discharge Event," AN202012, National Semiconductor Corp., 2006.

T. Lun, "Designing for Board Level Elec-

tromagnetic Compatibility," AN2321, Freescale Semiconductors, Inc., 2005.

B. Perrin, "I/O For Embedded Controllers," Parts 1 & 2, *Circuit Cellar Online* 110 and 111, 1999.

R. Valentine, "Protection Techniques Ensure μ C Reliability in Power Control Circuits," *EDN Magazine*, 1996.

T. Williamson, "Designing Microcontroller Systems for Electrically Noisy Environments," AP125, Intel Corp., 1996.

SOURCES

ADG465 Single-channel protector
Analog Devices, Inc.
www.analog.com

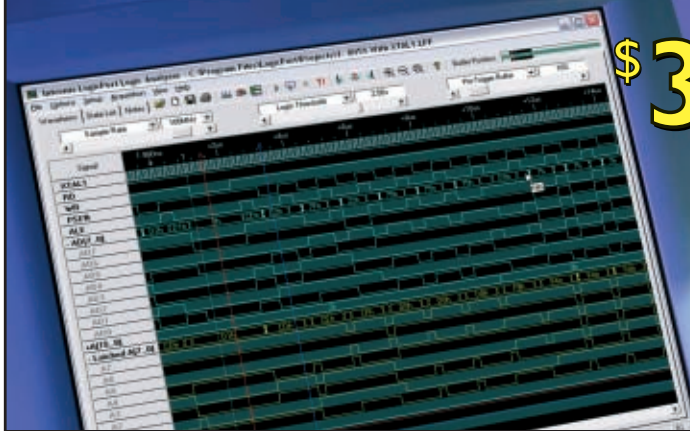
LOC110 Linear optocoupler
Clare, Inc.
www.clare.com

SP720 Diode array
Littelfuse
www.littelfuse.com

INA148 Common-mode amplifier
Texas Instruments, Inc.
www.ti.com

500 MHz Logic Analyzer

\$389



Professional Features – Exceptional Price

34 Channels sampled at 500 MHz
Sophisticated Multi-level Triggering
Transitional Sampling / Timing and State

**Connect this indispensable tool to your PC's
USB 1.1 or 2.0 port and watch it pay for itself within hours!**

- 500 MHz Sampling / Timing Mode (Internal clock)
- 200 MHz Sampling / State Mode (External clock)
- Multi-level Triggering on Edge, Pattern, Event Count, Group Magnitude/Range, Duration etc.
- Real-Time Hardware Sample Compression
- Qualified (Gated) State Mode Sampling
- Interpreters for I²C, SPI and RS232
- Integrated 300 MHz Frequency Counter
- +6V to -6V Adjustable Logic Threshold supports virtually all logic families
- Full version of software free to download
- Mictor adapter available

www.pcTestInstruments.com

Visit our website for screenshots, specifications and to download the easy-to-use software.

Intronix Test Instruments, Inc.
Tel: (602) 493-0674 • Fax: (602) 493-2258
www.pcTestInstruments.com



Pulse Generation

Encoder Interfacing to Microcontrollers

Try a continuous rotary knob in your next design instead of keypads or buttons. With an optical encoder and an ATmega8515 microcontroller, a digital output is produced in Stuart's system.

In the digital world, we are used to using keypads and the increase and decrease buttons in our designs because they work well and are easy to implement. But sometimes the best user interface is just a round knob. This is the case when you have to adjust a value and watch it change on a display, such as an oscilloscope, or when a continuous feel is needed.

In some cases, you can add a simple potentiometer to your design to either directly control something (such as the volume of an audio amplifier) or read the value of a potentiometer with an ADC and interpret it with software. But, a potentiometer has limitations: it can turn only one revolution, in high-resolution applications it is difficult to set exactly the same value twice, and the resistive element tends to wear out with continuous use.

There are a lot of cases where adding a continuous rotary knob, which could be read by a microprocessor, would enhance the design. One way to add a rotary knob to your design is to use an optical encoder. This is a potentiometer with a shaft for the attachment of a knob and producing a digital output when rotated. Most encoders do not have mechanical stops and can be rotated continuously.

Optical encoders were expensive in the past; however, the price has come down, especially those made for consumer electronics, such as Grayhill series 62P encoders. They are available for less than \$10 through distributors such as DigiKey. The 62P series encoders produce 16 steps per revolu-

tion and include a mechanical detent for each step.

RANGE VERSUS RESOLUTION

One immediate problem with rotary encoders is simultaneously handling high resolution and wide range (see Figure 1). This project, a pulse generator, uses an Atmel ATmega8515 microcontroller to generate a continuous pulse output with a varying frequency and duty cycle. One of the ATmega8515's timers creates the output waveform. The frequency and duty cycle of the output are controlled by 16-bit registers, so they have a range up to 65,535.

Two registers are used to generate the pulse output. One controls the frequency and one controls the period that the output is low (i.e., the duty cycle of the waveform). To get high resolution, you want one click of the encoder to produce one increment or decrement of the period or frequency registers. However, this is an impractical way to cover the full range, because you would have to rotate the knob 4,096 times to go from one extreme to the other. (To avoid confusion, in this article, "period" applies to the duty cycle adjustment, not the overall period of the waveform.)

Ideally, the encoder would have a large step size when you want to rapidly go from one range to the other, and a small step size to zero in on the exact values needed. You could have multiple encoder wheels, one for each step rate, but the circuit is both cheaper and more compact if a single encoder can be used.

The 62P series encoders are available with an integrated push button. Rotating the knob produces a digital output waveform. Pushing the knob closes a momentary switch. My example uses the switch to select the range. Encoder mode so one rotation step (one detent) increases or decreases the timer values by 10. Pushing the knob a second time increases the value to 100. Pushing the knob again increases it to 1,000, and the fourth push returns it to a step size of one.

Using the switch in this way enables you to vary the frequency/period over the entire available range with just five full rotations of the knob in 1,000-step mode. However, you can select a single step of 68 ns in 1-step mode. The circuit includes LEDs to indicate the 10-, 100-, and 1,000-step modes so you know what to expect when the knob is turned.

With this circuit, it is possible to adjust the period register to a value greater than the frequency register, which would produce no output. In that case, the software forces the period register to equal the frequency register minus three so the output does not disappear completely. The calculation is made each time an encoder change is detected.

The circuit includes push button switches to select between frequency and period, and LEDs to indicate each. In Frequency mode, rotating the knob adjusts the frequency. In Period mode, the knob adjusts the period of the low portion of the waveform. The encoder knob has several modes of operation

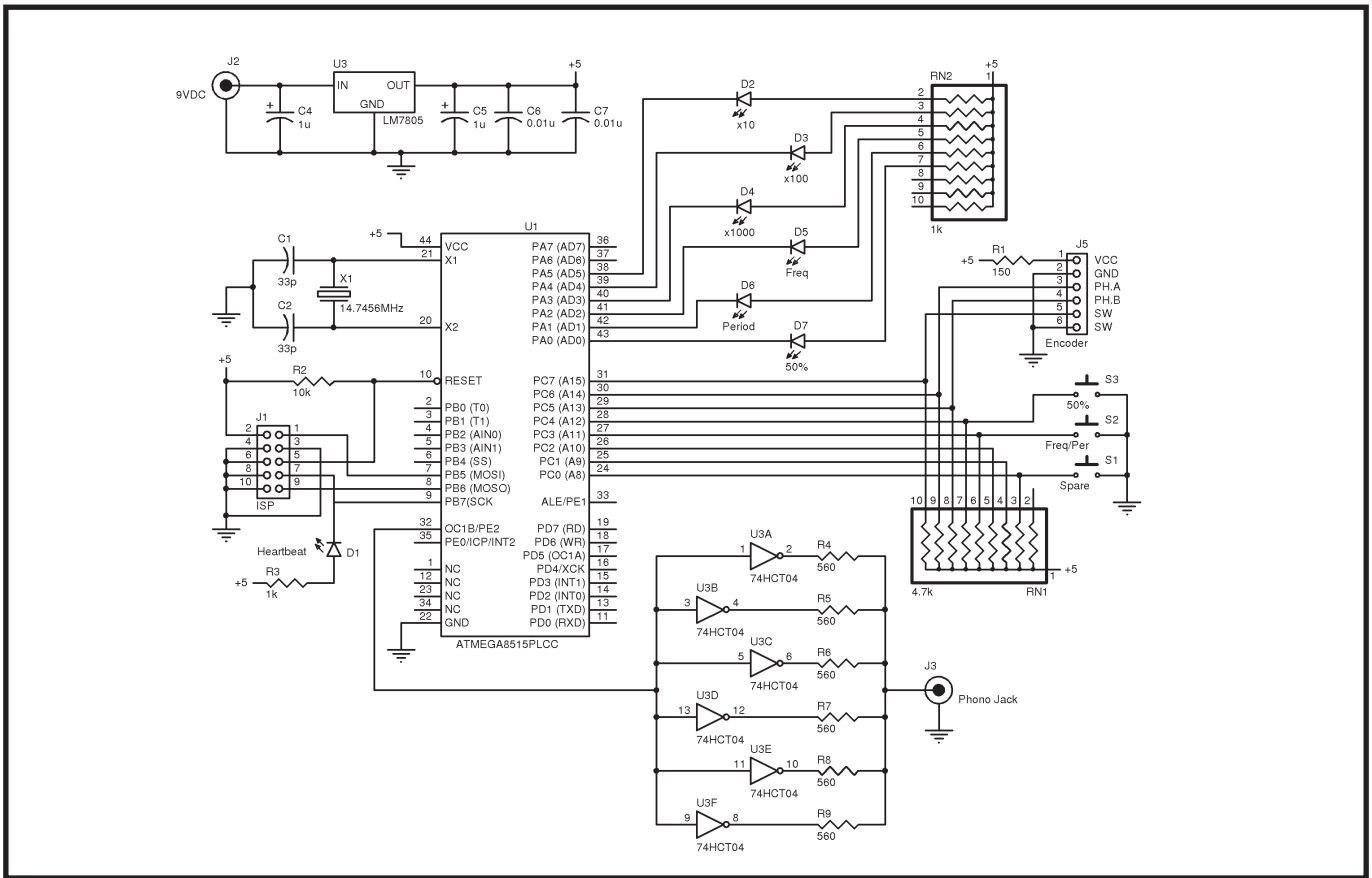


Figure 1—The pulse generator circuit is based on an Atmel ATmega8515 microcontroller.

because it has the $\times 1$, $\times 10$, $\times 100$, and $\times 1,000$ step sizes in both Frequency and Period modes.

The circuit also has a switch (and an associated indicator LED) to select a 50% duty cycle. In this mode, the period is fixed at 50% of the frequency and the knob can only be used to adjust the frequency. The $\times 10$, $\times 100$, and $\times 1,000$ modes are still available, but they apply only to frequency adjustment. The Frequency/Period switch does not change the encoder functionality in this mode. The new period value is calculated each time an encoder change is detected.

OTHER IMPLEMENTATIONS

An encoder without an integral switch could be used in the circuit by using an additional push button switch to change modes of operation. Another enhancement could include remembering the step size of the Frequency and Period modes. This way, if the frequency was adjusted using $\times 10$ steps and the period was adjusted using $\times 1$ steps, switching back and

forth between Period and Frequency modes would also switch the step size to the last value used for the respective mode. That feature is not included in the software, but it is certainly possible to add.

Another way to implement the range resolution is to have a “shift” key for each step size. The keys would be held down while rotating the encoder knob to adjust the value. The advantage of this is very fast switching back to $\times 1$ mode. The disadvantages are the need for extra switches and the need for two-handed operation.

Another enhancement that could be added is to have the period (duty cycle) adjustment be proportional to the frequency. So, the $\times 10$, $\times 100$, and $\times 1,000$ settings would be a percentage (say, 1%, 5%, and 10%) of the frequency register value in Period Adjustment mode. The larger rate settings (particularly the $\times 1,000$ setting) would allow for finer adjustment at higher frequency values.

USER CONSIDERATIONS

In many cases you have some fac-

tors to consider when using a continuous adjustment of this type. One example is the period/frequency adjustment of the example circuit. What do you do when the user tries to adjust the period to a greater value than the frequency register contains? In the case of the waveform generator, the software limits the period to ensure a valid output, even if it isn't exactly what the user expected. There are other examples. Do you let the user spin past the end of the range and wrap around to the beginning, or do you stop advancing the values when the end of the range is reached? Do you need to limit voltage, current, motor speed, or other parameters to a safe value?

Avoiding user confusion is also a factor. Say you limit the user to the end of the range so it can't wrap around by continuing to spin the knob. The current frequency register value is 65,000, and the user turns the knob clockwise while in $\times 1,000$ mode. The next step would be 66,000, which would wrap around to 464 in a 16-bit register. However, since a wraparound is not

Channel B	Channel A	Direction
Rising	1	Forward
Falling	0	Forward
0	Rising	Forward
1	Falling	Forward
Rising	0	Reverse
Falling	1	Reverse
1	Rising	Reverse
0	Falling	Reverse

Table 1—An example of quadrature encoding using two lines to indicate rotation and direction.

allowed, the software limits the value to 65,535. The user then turns the knob back one detent. The new value is 64,535 not 64,000. Does this confuse the user? In the case of the waveform generator, probably not. In other cases, it may.

In a timing-based circuit, can you turn the knob faster than the waveform can be adjusted? If you can, strange outputs can result. Another potential drawback is the chance of making a register adjustment while the timer is wrapping around. In the example circuit, the only impact of this is a possible delay while the timer counts up to the new value again. In some other applications, this must be prevented by synchronizing timer updates with timer rollovers.

OTHER APPLICATIONS

This article has focused on using an encoder to implement a pulse generator because it is a simple example that illustrates the concepts. Other applications for a variable-range selection or step size are: one knob to select both the band and frequency in a radio or other RF device, normal and fine tuning for a radio or transmitter, a single knob to select an input or output signal source and to set the level, sweeping quickly across a displayed waveform and then zeroing in on a particular area, and selecting a file or data block from a list and then zooming in to view a particular location.

THE CIRCUIT

Although the circuit used here is intended to illustrate the encoder range and resolution concepts rather than be used as a construction project, it may be useful for you to understand the circuit itself. The circuit is based on the ATmega8515 and a Grayhill 62P22-L6

encoder. The ATmega8515 is an 8-bit, flash memory-based microcontroller with in-circuit programming capability. The important features for this purpose are the internal 16-bit timers.

Timer1 of the ATmega8515 has a PWM mode of operation with two 16-bit registers. One register, OCR1A, sets the number of clocks before the timer restarts and counting starts over (the number of clocks that constitute a full cycle of output). The other register, OCR1B, sets the number of clocks that the OCR1B output remains low at the beginning of the output cycle. By adjusting the OCR1A value, the frequency of the output waveform can be controlled, and the duty cycle can be controlled by adjusting the OCR1B register. The software reads the encoder knob and other switches to determine how to adjust the two registers.

The 62P22-L6 encoder is designed for PCB mounting and has a 0.1875" diameter shaft. As mentioned earlier, the encoder produces 16 pulses per revolution and includes a push button switch. The encoder uses optical components to detect shaft rotation, so the output is glitch free, eliminating the need for debouncing circuitry or software. The encoder requires an external, current-limiting resistor (R1 in Figure 1) to supply current to the internal optical emitters. The optical transistors, which produce the Phase A and Phase B outputs, require external pull-ups.

The encoder produces a quadrature output. This means that two signal lines are used to determine when the shaft has been turned and in which direction. In the forward direction, phase A goes high first, then phase B goes high, then phase A goes low, and then Phase B goes low. In the reverse direction, B leads A (goes high first followed by phase A). By comparing the last state of phase A and B to the new state of the new signals, the software can determine which way the user turned the knob. Since only one phase changes at a time, there is no problem with ambiguity when reading the phase signals. Table 1 shows how the bits change for quadrature encoding and how the changes are interpreted.

The circuit uses LEDs to indicate when the encoder is in Period or Fre-

Forward motion (returns 1)		
Chan B, A	Changes to	
0, 1	1, 1	
1, 0	0, 0	
0, 0	0, 1	
1, 1	1, 0	
Reverse motion (returns 2)		
Chan B, A	Changes to	
0, 0	1, 0	
1, 1	0, 1	
1, 0	1, 1	
0, 1	0, 0	

Table 2—The software looks for specific state changes to detect motion. Note that there are 16 possible values for old and new states, but only eight are valid. The remaining eight indicate no change to the encoder state or are illegal transitions.

quency mode, 50% mode, and the encoder step size. The Port A pins could also be connected to a standard (14-pin) character LCD, such as those produced by Lumex. Of course, the software must then be adapted to use an LCD.

The circuit uses a standard 2.1 × 5.5 mm power jack to accept power from a 9-VDC wall-mount supply. J1 is the ISP connector for downloading code to the microcontroller. LED D1 blinks about once per second to indicate that the microcontroller is operating correctly.

The circuit uses a 14.7450-MHz crystal so the serial port will operate at standard data rates. However, since the serial port is not used in this design, almost any frequency will work. The output frequency and period step sizes, however, are tied to the crystal frequency. S1 to S3 are standard push button switches. S1 is a spare switch that is not used in this design.

THE SOFTWARE

Software for the project is written in WinAVR, which uses the GNU GCC open-source C compiler to target Atmel microcontrollers. The source is broken into two modules, the main program (pulsegen.c) and the encoder processing code (encoderprocess.c). No interrupts are used in the code.

The main program initializes the timers and variables, sets up the I/O ports, and then starts a continuous loop. The loop checks for overflow of the 8-bit T0 timer, which occurs about every 18 ms. The timebase is used to blink the heartbeat LED and debounce

Offset (hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Value	0	1	2	0	2	0	0	1	1	0	0	2	0	2	1	0

Table 3—This lookup table is used to detect motion by comparing old/new encoder values. A value of 0 means no change, 1 means forward motion, and 2 means reverse motion.

switch closures. The main loop also checks for switch closures and encoder changes and it updates the LED status based on the current state.

Encoder processing consists of comparing the stored bits from the last encoder sample to the new encoder sample. If the encoder has changed states, a positive value is returned to the main loop for processing. A value of 1 means the encoder was turned clockwise. A value of 2 means the encoder was turned counterclockwise. A value of 0 is returned if the encoder quadrature bits did not change or if the change was an unrecognized value.

The encoder comparison is performed by creating a byte that consists of 0000xxyy, where xx is the previous encoder value and yy is the new value. This is used to index into an array that determines what value is returned. Table 2 shows the valid encoder state changes.

The transitions in Table 2 translate into a 16-value array in the software. Array values are the values returned from the encoder processing code based on the old/new phase bit values (see Table 3). The array values of zero are places where there is no change (bits 2 and 3 = bits 0 and 1) or where the transition is illegal, indicating a missed step.

If a positive value is returned from the encoder comparison, the main code will increment or decrement the frequency or period values based on the state of the frequency/period indicator and the value of the 50% indicator. The period cannot be adjusted when in 50% mode, because it is forced to be half of the frequency.

The increment and decrement routines are passed the step size (1, 10, 100, or 1,000) and will increase or decrease the appropriate register by the step size. If the calculated values result in the period value being larger than the frequency value, the period is forced to be just less than the frequency to ensure that an output is still produced.

This is purely a decision for operator convenience. There is no reason why the output can't be allowed to turn off if you set a frequency value that is too small or a period

value that is too large.

Although the push button switches are debounced using the 18-Hz T0 rollover, the encoder quadrature signals are not debounced and are sampled every pass through the main loop. This provides a sufficiently high sampling rate to ensure no encoder transitions are missed. As I mentioned earlier, debouncing is not needed for the quadrature signals because the encoder's optical circuitry produces glitch-free outputs.

Adding a rotary control knob using a quadrature encoder is straightforward, and there are ways to handle the resolution/range trade-off. With the decreasing cost of encoders intended for consumer electronics applications, you can use these encoders in any places where they will make life easier for the user. ☒

Stuart Ball, P.E., is an electrical engineer with 20 years of experience. He is the author of three books about embedded systems, all published by Elsevier/Newnes. Stuart is an engineering manager at Seagate Technologies, LLC, in Longmont, CO.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2007/206.

RESOURCE

Grayhill, Inc., "Optical Encoders: Series 62P: Low Cost, PC Mount," 970, 2003, <http://lgrws01.grayhill.com/web/pdf/Bltn970.pdf>.

SOURCES

ATmega8515 Microcontroller
Atmel Corp.
www.atmel.com

62P22-L6 Encoder
Grayhill, Inc.
www.grayhill.com

ANDRE LAMOTHE'S
XGAMESTATION
DO-IT-YOURSELF VIDEO GAME SYSTEM

Inspired by the Atari 2600, Apple II & Commodore 64!



SX52 @ 80 MIPS

INCLUDES:

- Assembled XGS Micro Edition Unit!
- Complete Development Kit!
- Tools, Demos & Utilities!
- eBook on Designing the XGS Console!
- Cables and Power Supply Included!




WWW.XGAMESTATION.COM
(925) 736-2098 SUPPORT@NURVE.NET

USB/ETHERNET DAQ

LabJack UE9



Available now for only ...
\$429 qty 1

USB/Ethernet Data Acquisition & Control

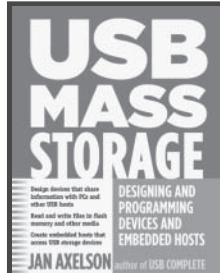
- * USB 2.0/1.1 and Ethernet
- * 14 analog inputs (12- to 16-bit)
- * Stream input data up to 50 kHz
- * Use with C, VB, LabVIEW, etc.
- * Includes DAQFactory Express
- * Operates from -40 to +85 deg C
- * 2 analog outputs (12-bit)
- * 23 digital I/O
- * Up to 2 counters (32-bit)
- * Up to 6 timers
- * Approx. 3" x 7" x 1"

LabJack Corporation, Colorado, USA
info@labjack.com, (303) 942-0228

www.labjack.com

ADD MASS STORAGE TO YOUR DESIGNS

USB MASS STORAGE



USB Mass Storage
Designing and Programming Devices and Embedded Hosts
Jan Axelson

ISBN 1-931448-04-3 \$29.95
Lakeview Research LLC www.Lvr.com

From the author of USB Complete

Embedded Scripting

With the Pawn scripting language, less is more. As Thiadmer explains, the language requires so little from its embedded host system that it can be added to an NXP LPC2106 microcontroller with few resources. Now you can extend your firmware without changing it.

In this article, I will present Pawn, which is a scripting language that targets embedded systems. I will also describe an implementation of the Pawn virtual machine on an NXP Semiconductors LPC2106 microcontroller.

Pawn is open-source software, but it has a zlib/libpng license that explicitly allows for commercial use. Refer to Figure 1 for an overview of the “conceptual blocks” that I will discuss.

SCRIPTING LANGUAGE

When I started as an engineer, many electronic devices were hardwired, using cascades of logic ICs or even analog chips. But the move to CPU-controlled devices was beginning. The press wrote that these microcontrollers were going to make our machines “intelligent,” but the real drive behind this move was, of course, the reduction of cost, increased functionality, and further miniaturization—all at the same time.

The increased functionality of contemporary devices almost always leads to the desire or necessity to configure the device. Many networked devices present a web browser interface for basic configuration. Older devices have well-known user interfaces with a handful of tiny push buttons that have double or triple functions. (Daylight savings time has just ended as I write this. I have lost count of how many ways there are to set a clock.)

The tricky thing is deciding what must be configurable, and how. You have to foresee how the customer will be using the apparatus. Imagine, for example, a climate control system: heating, cooling, and ventilation. Some users may want to have different settings in various rooms. Others may want to enter an elaborate schedule in order to reduce energy waste. Others may want PIN code security or the ability to keep a log of the average temperature over certain periods of time. A hotel may want to connect a climate control system to its booking system because there is no point in heating or cooling a vacant room. A laboratory may want to connect it to a security system. Whatever you think of, someone will come up with a scenario that your hardware can handle but your firmware can't support.

When a configuration system grows out of its proportions, it is time to look for an alternative—a scripting system, and ideally a scripting system that does not require more memory or CPU cycles than you can spare. A

scripting system can do everything that static configurations can do, but it adds dynamics. A script is an extension of your firmware, and it may change how the device behaves or interfaces with other devices. Obviously, you could change the firmware instead, but you'd do so at the risk of arriving at a multitude of customer-specific versions of the firmware, all of which have to be maintained. Trust me, I have been there. With scripts running in a “sandbox,” there is a separation between the host application (the firmware) and the customized module. You may enable the customer to “script” your device without risking breaking the essential functionality of the device.

There are many scripting languages, so you may wonder why another language is needed. The answer is that the design for a quick and light virtual machine prompted me to make changes to the programming language that I used as a model. Pawn started as a subset of C, and it still has roughly the same operator set and instruction set as C. The virtual

machine requires little from its host (the firmware or application that embeds the virtual machine). It does not use dynamic memory or a file/console I/O interface, and it does not need to run in a separate thread. For example, the virtual machine runs well on a Texas Instruments MSP430F1611 microcontroller, with 48 KB

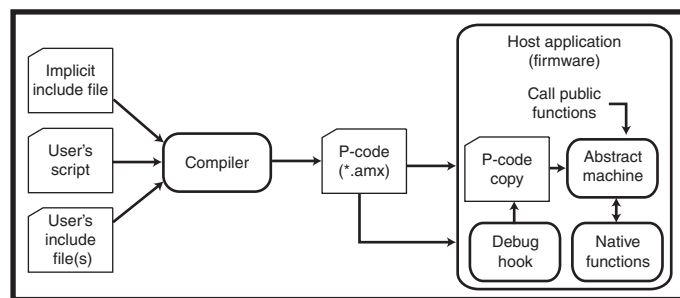


Figure 1—This is an overview of the scripting system. The Pawn compiler runs on a PC or workstation and builds the P-code file from a set of source files. The P-code file must then be moved into the firmware, possibly via a debugger hook, for the abstract machine to execute it.

Listing 1—These are the steps needed to initialize and run a script: initialize the virtual machine, register native functions, and call the `exec` function. The error handling makes it look more complex than it really is.

```
extern const AMX_NATIVE_INFO my_Natives[];
extern unsigned long bytecodes[];

AMX amx;
cell ret;
int err;

memset(&amx, 0, sizeof amx);
err = amx_Init(&amx, bytecodes);
if (err == AMX_ERR_NONE) {
    err = amx_Register(&amx, my_Natives, -1);
    if (err == AMX_ERR_NONE) {
        err = amx_Exec(&amx, &ret, AMX_EXEC_MAIN);
        if (err == AMX_ERR_NONE)
            HandleReturnCode(ret);
    }
}
if (err != AMX_ERR_NONE)
    HandleError(err);
```

of ROM and 10 KB of RAM, alongside the main firmware of the appliance (a security device in this case) and the μ C/OS-II multitasking kernel. There are not many scripting languages that run in such an environment. I made it easier on myself. I used an NXP LPC2106, which is a Keil ARM7TDMI microcontroller with 128 KB of ROM and 64 KB of RAM (plenty of memory).

VIRTUAL MACHINE

Since the more interesting part of the Pawn scripting system is the virtual machine, I will start there. Pawn is a bytecode-compiled language, and the virtual machine is a bytecode interpreter. Incidentally, the Pawn manual uses the term P-code or pseudocode instead of bytecode, but that is merely because I am from the old school. The two mean the same thing. For reasons of alignment, each opcode takes the size of the basic data unit, a “cell.” Typically, Pawn is configured to be a 32-bit language (even when running on an 8-bit controller), and a cell is 4 bytes. The virtual machine is in portable C source and, to a large extent, in a single-source file. With that said, there are many configuration options to tweak the virtual machine to your requirements or preferences.

Adding the virtual machine to your project requires a few steps. First, include the source code for the Pawn virtual machine in your other source files. This one is easy because the virtual machine should compile out of the box. The second step is to call (from

your firmware) the functions to initialize and “run” the virtual machine. This should not be too hard either. The third step is to add “native” functions to your firmware. Native functions are the interface between the script and your firmware. Without native functions, a script cannot reach out of its sandbox. It can burn a few CPU cycles, if you have them in excess, but that is about it. Although you may be able to use one or more of the native function libraries written by me and others, you will likely need to write a handful of native functions that are specific to your device or firmware. One last thing: you also need to find a way to transfer the bytecode of the script to the device (e.g., via a serial line or using a memory card of some kind). If you are using a serial line, one option is to implement a debugger hook in your firmware. The Pawn source-level debugger has commands to transfer files. It also has the common breakpoint, data inspection, and stepwise execution commands, of course.

The snippet of code in Listing 1 shows the essential code for running a compiled script. The listing assumes that an array called `bytecode` is declared elsewhere and that it is filled with the compiled script. Furthermore, the array must be large enough to contain the data and the stack for the script. I will omit how to fill this array for the moment. This snippet declares a variable with the type `AMX` and initializes it by first clearing it and

then calling `amx_Init`. While the virtual machine exists, this variable contains the complete state of all pseudo-registers of the virtual machine. If you want to have two virtual machines running two scripts, you essentially need only create and initialize a second `AMX` variable. The requirement to clear the `AMX` structure before initializing it is a bit of a leftover from the past, but it still has a purpose in enabling you to split the code from the data/stack. “AMX” stands for abstract machine executor. “Abstract machine” was the name for virtual machines in the era that bytecode was still called P-code.

After initializing the virtual machine, register the native functions to it. You can do this in a single step or in multiple steps. In Listing 1, there is only one table with native functions. I will return to how to implement native functions later. For a quick peek, refer to Listing 2. Initializing a script and registering native functions are one-time steps. When you run a script multiple times, there is no need to repeat these.

Finally, `amx_Exec` runs the script and does not return until it completes. In Listing 1, the script starts at the “main” entry point. Like in C language, this is a function called `main` in the script. A script may have alternative entry points. In that case you can select the entry point to call. The primary use of multiple entry points in a script is in an event-driven setup, where each kind of event “fires” a particular function in the script. In Pawn, these entry points are called “public” functions. In a sense, public functions are the opposite of native functions. The host application calls into the script through a public function and a script calls back into the firmware through a native function.

The Pawn virtual machine is very configurable. On many systems, it will compile with the default configuration, mostly because the virtual machine is quite self-contained. It requires only a few string routines from the standard library. On smaller systems, you may need to add a few macro definitions on the compiler command line. Table 1 lists the options I used for this LPC2106-based project. Other configuration macros are listed in the Pawn documentation.

At a high level, this completes the embedding of the Pawn virtual machine in your own code, with the exception of a few holes that I have jumped over and need to fill in. The functions to handle errors and to handle any return value from the script must also be written by you. Fortunately, if you have configured the Pawn compiler well, most problems will be flagged by the compiler, so you can simplify the error reporting on the device. For example, `amx_Register` fails if the script contains references to native functions that the firmware does not provide. Since the Pawn compiler requires the declaration of all native functions, this can only happen if the user has a Pawn compiler setup that does not match the target firmware.

NATIVE FUNCTIONS

On to native functions. Listing 2 gives implementations for three simple, native functions. These functions allow you to toggle some of the general-purpose I/O lines and to configure these as inputs or outputs. It is an arbitrary choice to limit the range of I/O pins between 0 and 15.

It has more to do with the particular development board that I used for testing the code in this article than with a particular limitation of the microcontroller or the Pawn system.

All native functions have the same definition. The first argument is a pointer to the virtual machine state and the second parameter holds all arguments that the script passed to the native function. The `params` function argument is a pointer directly into the stack of the abstract machine. The native function can access the fields directly, without needing to copy/extract arguments out of the virtual machine. The first element in the `params` array is the number of bytes passed to the native function. Functions that accept a variable argument list can use this to find out how many arguments were passed. The `my_Natives` table is at the end of Listing 2. It was also referred to in Listing 1. For convenience, this table ends with a "null" entry, so you can instruct `amx_Register` to count the number of elements in the table, instead of doing it yourself.

I used a development board that I saw in *Elektor Electronics*.^[1] The board has the first 16 I/O pins connected to a row of LEDs. By toggling the I/O pins on and off, the Pawn script in Listing 3 creates a "walking light." I cheated. The script uses a native function (`delay`) that is not in Listing 2. This is just a simple function that waits the number of milliseconds passed in a parameter. The native function declarations at the top of Listing 3 would normally be in an include file. At first sight, this code may not look much like C because semicolons are optional (in most cases). Parentheses around the function arguments are optional in a few situations as well. It is perfectly well allowed to add these optional elements and make the code look more like C (see Listing 4).

Fine, but how do you get the script into the microcontroller's memory so that it can execute it? Well, you must find that out yourself because it depends on which interfaces you have in your device. If your device uses a CompactFlash or an SD/MMC card, you can store the compiled script on

Easy Embedded Linux

\$169
Qty 1



16MB FLASH / 32MB RAM

200Mhz Arm9 CPU

16 Digital I/O

Watchdog

10/100 Ethernet

Battery backed Clock/Calendar

Audio In/Out
2 USB
2 Serial Ports

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

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

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

JK microsystems

Got Serial, Need Network?



Bluetooth
Qty 1
\$175



Ethernet
Qty 1
\$99



Wireless
Qty 1
\$199

Volume Discounts Available



gridconnect.

www.gridconnect.com
+1 800 975-4743

USBee DX

Oscilloscope/Logic Analyzer

2 Analog and 16 Digital Signals
Up to 24Mps, 100+ Million Samples
Dual 8-bit ADC, +/-10V inputs

Bus Decoding

Click and Drag Instant Decode of Bus Transactions
I2C, SPI, ASYNC, CAN, USB Low and Full Speed
I2S, SM Bus, 1-Wire, PS/2

Digital Voltmeter

2-Channel, +/-10V, 8-bit ADC, Logging

Data Logger

2 Analog and 16 Digital Signals Plus Timestamp

Digital Signal Generator

16 Digital Outputs, Up to 24Mps
Playback of Logic Analyzer Traces

Pulse Width Modulator

16 User Controlled PWM Channels

Frequency Counter

16 Channel Counter to 24MHz

Frequency Generator

Generates Sets of Common Frequencies

I2C Controller

Control I2C Devices Using Simple Text Scripts

Remote Controller

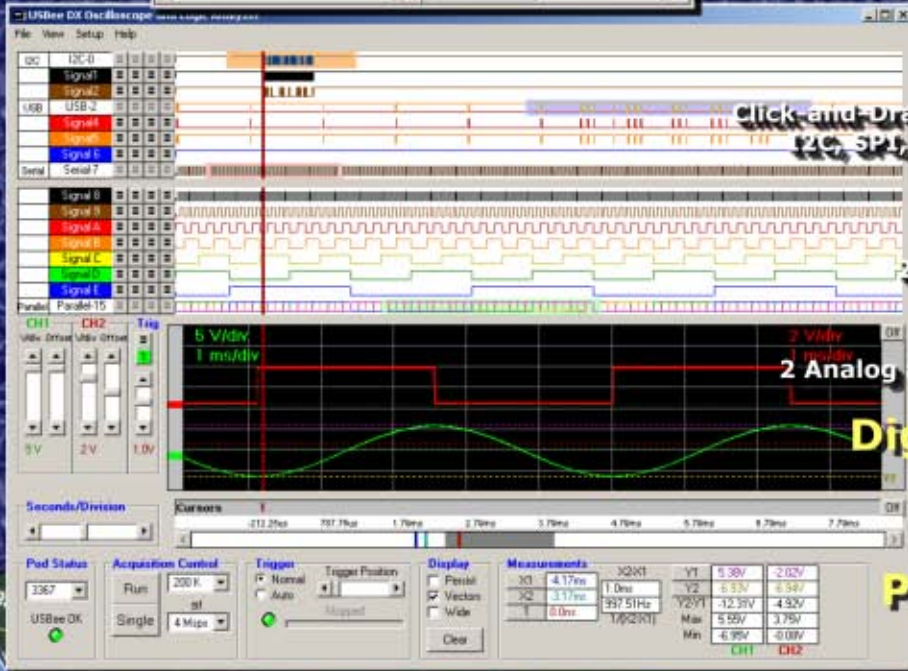
Easily Control Your Hardware Using Your PC

Pulse Counter

16 Channel Pulse Counter With Gating Control

plus the USBee Toolbuilder Source code and Library

Create Your Own Applications to Control the USBee DX



Actual Size

Data Extractors

Optional Software Modules for the USBee AX-Pro and USBee DX

Continuous Real-Time Embedded Bus Data Streaming

Store Bus Data to Disk or Send to Your Custom Application

Capture and Process Entire Test Sequences

Parallel, Serial, I2C, USB, ASYNC, CAN, SMBus, SPI, I2S, 1-Wire

USBee.com (951) 693-3065
support@usbee.com



*The premier conference and exhibition for PCB
engineering, design and manufacture professionals*

PCB DESIGN CONFERENCE EAST

Conference: October 21-26, 2007
Exhibition: October 23-24, 2007
Durham Marriott at the Civic Center
Durham, NC

View the conference
program online at
www.pcbeast.com

Register online now for your FREE Exhibits
Pass with Priority Code E07CC097

Association Sponsor:



Media Sponsors:

PRINTED CIRCUIT
DESIGN & MANUFACTURE

**CIRCUITS
ASSEMBLY**

You'll enjoy free admission to the:

- Two-day exhibition
- Tabletop Showcase
- Keynote Address
- 13 FREE technical sessions
- Wine and Cheese Mixer
- Backyard BBQ and Game Night

Visit www.pcbeast.com for details.

Listing 2—Implementing native functions is fairly straightforward. Native functions have a standard definition and they access parameters directly in the stack of the virtual machine. Only when handling arrays or values passed by reference do you have to translate addresses between “physical” and “virtual.”

```
#define MAX_IOPIN    16

cell AMX_NATIVE_CALL n_configpin(AMX *amx, const cell *params)
{
    int mask = (1 << params[1]);
    if (params[1] >= 0 && params[1] < MAX_IOPIN) {
        switch (params[2]) {
            case 0:                /* input */
                GPIO_IODIR &= ~mask;
                return 1;
            case 1:                /* output */
                GPIO_IODIR |= mask;
                return 1;
        }
    }
    return 0;
}

cell AMX_NATIVE_CALL n_setpin(AMX *amx, const cell *params)
{
    int mask = (1 << params[1]);
    if (params[1] >= 0 && params[1] < MAX_IOPIN) {
        switch (params[2]) {
            case 0:
                GPIO_IOCLR = mask;
                return 1;
            case 1:
                GPIO_IOSET = mask;
                return 1;
        }
    }
    return 0;
}

cell AMX_NATIVE_CALL n_getpin(AMX *amx, const cell *params)
{
    int mask = (1 << params[1]);
    if (params[1] >= 0 && params[1] < MAX_IOPIN)
        return (GPIO_IOPIN & mask) != 0;
    return 0;
}

const AMX_NATIVE_INFO my_Natives[] = {
    { "configpin", n_configpin },
    { "setpin",    n_setpin    },
    { "getpin",   n_getpin    },
    { NULL, NULL }
};
```

that card and load it. Alternatively, if you have a suitable area of nonvolatile memory, you can transfer the compiled script to that area. For the transfer, you can use the Pawn debugger at one end, so that you have to implement only the receiver in the firmware. I posted a suitable “receiver end” that stores the transferred script in one of the 16 sectors of flash ROM on the LPC2106 on the *Circuit Cellar* FTP site. For a first quick test, however, I usually transform the output of the Pawn compiler to a C array with the “xd” utility. Then, I include the C array into the source

code and adjust Listing 1 to copy the array into the bytecodes array before calling `amx_Init`.

Before storing the script in a location where the microcontroller can reach it, you should compile the Pawn source to bytecode. The Pawn compiler has some defaults that may not be suitable for small systems. For instance, it reserves a 16-KB stack. To adjust the stack size, you can use the option `-S` on the command line of the Pawn compiler. Better yet, you can store it in a file called `pawn.cfg`, which you put in the same directory where

the compiler resides. The compiler will use this as the default. If you also store the declarations of all native and public functions and constants in the `default.inc` file, you will not have to explicitly include any “header” files in your scripts because the Pawn compiler processes `default.inc` before parsing your script.

PAWN SPECIFICS

I have put the cart before the horse by jumping at the embedding of the virtual machine in firmware before telling you exactly what the Pawn language brings to your firmware. Pawn is a derivative of C, but it is reduced and simplified at a few points. The biggest deviation from C is that Pawn lacks a typing mechanism. All variables are 32-bit integer values in Pawn. The rationale behind it is that a Pawn script would mostly manipulate “objects” in the firmware, and it would do that through native functions. Compare it to dealing with files in C. The file is a resource in the operating system, and your C program manipulates it through library functions like `fopen`, `fread`, and so on. The C program identifies an open file with a file pointer, an opaque handle. Pawn scripts should also identify resources of the device or of the firmware via opaque handles. To get some static checking, Pawn has a “tagging” mechanism, which you might see as a typing system where all types are restricted to fit into 32 bits.

The single-base type is called a “cell” in Pawn. I described it as a 32-bit integer because most implementations use a 32-bit cell. By redefining operators, you can store fixed-point or floating-point values in Pawn. User-defined operators are similar to overloaded operators in a language like C++. This design allows for some flexibility. On platforms lacking floating-point support, you can still use fractional values in Pawn by adding a fixed-point module. It comes at a cost, however, because user-defined operators have the overhead of a function call.

You will often want to associate a script, or part of a script, with an event (e.g., an alarm that goes off, a level that was reached, or a switch that was toggled). In other words, scripts tend to

Command	Description
AMX_NODYNALOAD	When compiling the Pawn virtual machine for Microsoft Windows or Linux, the virtual machine can load libraries with native functions as DLLs or shared libraries. (This is not applicable to most embedded systems.) If your firmware runs on embedded Linux or Windows CE, you may need to add this macro to disable dynamic loading.
AMX_ANSIONLY	Pawn has support for "wide characters" by default, but it requires a few wide character string functions from the run-time library to support this. If you only intend to use 8-bit characters, you can remove this dependency.
AMX_NO_NATIVEINFO	This definition removes the function <code>amx_NativeInfo</code> via conditional compilation. This function is rarely needed, and it is the only function with static data. After removing the function, the virtual machine is fully reentrant.

Table 1—Here are a handful of useful options for building the virtual machine. The Pawn documentation lists several more.

be event-driven. With multiple entry points in the form of public functions, Pawn enables you to run a different part of the script for each kind of event. The advantage of doing it this way, rather than running a different script for every event, is that you can share variables and states between the events.

The word "states" did not drop by accident. Pawn supports finite state machines ("automata") in the language. It is my experience that many event-driven programs end up with a few states. You can, of course, implement states in an ad hoc way with a variable and a few `switch` statements, but with language support, the compiler checks the automaton model. Pawn extends the standard state machines by introducing state-local variables—variables that have the scope (and life) of a set of states. When data must be shared across states, state-local variables can replace global variables.

I posted a somewhat expanded version of Listings 1 to 3, plus some "driver" code for a few internal peripherals, such as the real-time clock and the RS-232, on *Circuit Cellar's* FTP site. A debugger hook, which uses the in-application programming (IAP) interface of the LPC2106 to store the script in a flash memory ROM sector, is also posted on the site. I compiled the source code with GNU GCC. When using a different compiler, you may need to make minor adjustments to the IAP module. The other modules should compile without problems. The GNU GCC team apparently feels that the relative precedence levels of the logical "and" and logical "or" operators are ambigu-

ous or ill-understood by programmers. When compiling `amx.c` with all warnings enabled, GCC issues a list of warnings where it suggests to add parentheses around sub-expressions. You can make these go away with the GCC command-line option `-Wno-parentheses`.

PAWN IN REVIEW

Pawn's strong point is in the interface with the host application. There is little overhead in calling a native function, so this interface is fast. Arguments can be passed by value or by reference, and all arguments may have

default values. (This is not restricted to the last arguments in the argument list.) In the function call, you can indicate the arguments by position or by name. Using named arguments is convenient for functions with many arguments, especially if you leave most arguments at their default values.

The weak points of Pawn are strings and the typing system, or the lack of a typing system. Pawn knows cells and arrays of cells. It does not have structures or classes. The conceptual idea of Pawn is that the host application builds and maintains objects or resources on behalf of the script and passes the script an opaque handle. In addition, array operations are more flexible in Pawn than in C, so in some contexts, you may simulate light-weight structures with arrays. That aside, the lack of support for structured data is probably the weakest point of Pawn.

Support for strings in Pawn is equally primitive, as in C. Strings are zero-terminated arrays with a fixed maximum length. If you have to do a lot of string processing in your device, you may want to make a dynamic string library in the firmware and let the Pawn scripts access these via native functions. This

Listing 3—A simple script that makes a running light at the 16 LEDs connected to the first 16 I/O pins of the Elektor ARMee development board.

```
enum {
    Input,
    Output,
}
native bool: configpin(pin, type)
native bool: setpin(pin, value)
native getpin(pin)
native delay(ms)

main()
{
    const first_pin = 0
    const last_pin = 15
    new pin

    for (pin = first_pin; pin <= last_pin; pin++)
        configpin pin, Output

    pin = first_pin
    for ( ;; )
    {
        setpin pin, 1
        delay 100
        setpin pin, 0
        if (++pin > last_pin)
            pin = first_pin
    }
}
```

Listing 4—This is the “main” function in Listing 3 in C-style syntax. As you can see, Pawn can look like C.

```
main()
{
    const first_pin = 0;
    const last_pin = 15;
    new pin;

    for (pin = first_pin; pin <= last_pin; pin++)
        configpin(pin, Output);

    pin = first_pin;
    for ( ;; )
    {
        setpin(pin, 1);
        delay(100);
        setpin(pin, 0);
        if (++pin > last_pin)
            pin = first_pin;
    }
}
```

is roughly the same as the approach used by the `std::string` class in C++. For applications and devices with an international audience, a bonus is that Pawn knows both “packed” strings with 1 byte per character and “unpacked” strings where a character takes a 32-bit cell. Therefore, an unpacked string can store Unicode characters or an even larger character set in UCS-4 encoding. (Unicode has a limit of approximately 50,000 characters for all languages in the world, which some feel is too small.)^[2] In many functions, you can use these interchangeably, without needing to first transform one kind of string into the other kind. The Pawn compiler accepts source code in UTF-8 encoding, and you can use Unicode characters in string literals.

GET STARTED

This article only scratches the surface of many of the aspects involved in adding Pawn to a system. I have barely discussed the language, and there is more to calling public functions and making native functions. The documentation for Pawn is largely split into two guides. The “language guide” is what the script writer needs. It contains a tutorial and the definition of the Pawn language, as well as appendices covering the compiler command-line syntax and explanations for error messages. You need the “implementer’s guide” for guidance about making native functions, passing arguments to public functions, and

handling errors. The nature of embedded systems is very diverse, which is why I usually recommend to first get Pawn running on a standard PC or workstation. Then, move it to the embedded world after familiarizing yourself with it. The implementer’s guide gives explicit building examples for Microsoft Windows and Linux using a variety of compilers.

Although I mentioned that there are a lot of “little languages” like Pawn already, I have not referred to any of them. I will make an exception for BCPL, created by Martin Richards in 1966, because Pawn owes a lot to it.^[3] First, BCPL indirectly inspired C. Pawn takes its general syntax, its operator set, and most of its instructions from C. Second, BCPL implemented some concepts in 1966 that we now view as “modern,” such as optional semicolons at the end of an instruction. Pawn’s semantic rules for optional semicolons are identical to those in BCPL. The expression syntax like “5 <= a <= 10”—which gives the logical result of variable “a” being between 5 and 10—is copied from BCPL too. Oh, and the idea to make a compiler produce object code for a virtual machine so only the virtual machine needs to be ported to a new hardware platform is ubiquitous today, but BCPL is reportedly the first system to take this approach. BCPL is an interesting language and system, and it is still very much alive. Thank you, Mr. Richards. 📧

Thiadmer Riemersma writes system software and embedded software for his company, CompuPhase, which is located in the Netherlands. He holds a B.Sc. in Precision Mechanics, which comes in handy for the construction of prototypes. In his spare time (and sometimes during company time), Thiadmer maintains the Pawn scripting language and toolset. You may reach him at thiadmer@compuphase.com.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2007/206.

REFERENCES

- [1] T. Dixon, “LPC210x ‘ARMee’ Development Board,” *Elektor Electronics*, April 2005.
- [2] N. Goundry, “Why Unicode Won’t Work on the Internet: Linguistic, Political, and Technical Limitations,” Hastings Research, Inc., 2001, www.hastingsresearch.com/net/04-unicode-limitations.shtml.
- [3] M. Richards, “BCPL,” University of Cambridge, www.cl.cam.ac.uk/~mr10/BCPL.html.

RESOURCES

NXP Semiconductors, “LPC2104/2105/2106: Single-chip 32-bit Microcontrollers; 128 KB ISP/IAP flash with 16/32/64 KB RAM,” LPC-2104_2105_2106_6, 2006, www.nxp.com/pip/LPC2106.html#datasheet.

J. Walker, “Extended Dump and Load Utility,” www.fourmilab.ch/xd/.

SOURCES

Pawn toolkit

ITB CompuPhase
www.compuphase.com

ARM7TDMI RISC Processor

Keil
www.keil.com

LPC2106 Microcontroller

NXP Semiconductors
www.nxp.com

MSP430F1611 Microcontroller

Texas Instruments, Inc.
www.ti.com



String Theory

George explores how C language supports strings. He wrote a program using C that runs on a PC and converts an input file containing PCB assembly information into a file that can be used by a Philips pick-and-place machine.

Sorry, I am not going to be discussing theoretical physics. I hope you're not too disappointed. As you've learned C, you've moved from an embedded single-chip solution with a Texas Instruments MSP430 to a much more powerful and fully integrated chip: Freescale Semiconductor's Coldfire family of processors. With the smaller devices, it's unlikely that you would need all the support that C has to offer for string processing. And, the examples you worked with sidestepped much of the string issues. Well, this month you will explore how C supports strings.

You will explore strings by writing a program using C that runs on a PC and converts an input file containing PCB assembly information into a file that can be used by a Philips pick-and-place machine. This is a real program, and you can use this design approach for any file-conversion utility that you might need.

First, some background on this project. I recently designed a PCB that had many circuits repeated several times each. The parts list consisted of about 130 different part numbers, but the PCB contained more than 1,200 placements. Also, the customer had five or 10 different variations as to which of the 1,200 parts to place. The large amount of information and the variations were sinking the contract manufacturer in data.

DESIGN

I used Protel DXP to capture the design and generate the schematics. The output was a net list file, which was sent to the company that did the artwork design. They also used Protel (but an earlier version). They returned a design file that

was the PCB artwork. I could load that file into my version of Protel and work with it. From the Protel database, you could generate coordinates for each component placed. The coordinates were in a comma-separated value (CSV) file format. And, this file could be loaded into an Excel spreadsheet. The X,Y coordinates could be manipulated to rotate and translate all the placements. The order for picking could also be changed. In general, the contract manufacturer could easily work with this Excel file.

But wait, it's even more complicated. The specific pick-and-place machines used could hold many tape-and-reel parts or many tube parts, but not lots of both. Some parts could be mechanically centered using the pick jaws while others need to be optically inspected and registered before placement. So, the parts were divided among two machines. The first machine placed the resistors and capacitors (tape and reel parts) that needed only mechanical centering, while the second machine placed the ICs that needed optical checking before placement. By dividing the work in this manner, two machines were working on the build and the placement rate was doubled.

The only other detail I need to describe is that each pick-and-place machine has multiple heads. So, you need to specify which pick heads to use. Also, all the like parts (say the 10-k Ω resistors) are located at the same pick location. So, you need to specify where the pick location (source) of the part is as well as the place (destination) location.

SOFTWARE

For this file-conversion project, I used

Borland's C++ BuilderX. I still use Borland Turbo C++ 3.0, but I thought it was time to step up to a more modern toolset. I'm not sure where or how I got my copy, but cost was probably a deciding factor. It looks like you can download free versions (www.turboexplorer.com/homepage). A high cost should not stop you from trying C on your PC.

Now, note that this program we're creating is not a windows GUI solution. The IDE development environment is windows-based, but the application you are going to write opens a DOS window, runs, and then closes that window. Nothing fancy here, just trying to keep it a straightforward coding project.

PARAMETERS

I will start by talking about command-line parameters. In a DOS window, have you ever entered a command with options after that command? Perhaps "dir /od," which lists the directory contents in chronological order or "copy file1.ext file2.ext." C supports passing to a program any parameters contained on the line when the program was activated. The convention looks like this:

```
int main(int argc, char *argv[]);
```

`argc` is the number of parameters. `argv` is an array of pointers to each of those parameters. `argv[0]` is a string containing the program's name if that's available. As a review, `argv[argc]` is a null pointer because there are `argc-1` parameters passed to the program. And `argv[1]` is a pointer to a text string that contains the

Listing 1—This is how I extracted the command line parameters.

```
for (i = 1; i < argc; i++) {
// printf("argc %d = %s\n\r",i,argv[i]);
  if (strncmp("IN:", argv[i], 3) == 0) {
    strcpy(InFileName,&argv[i][3]);
    printf("InFileName = %s\n\r",InFileName);
    Debug = 0;
  }
  else if(strncmp("OUT:", argv[i], 4) == 0) {
    strcpy(OutFileName,&argv[i][4]);
    printf("OutFileName = %s\n\r",OutFileName);
    Debug = 0;
  }
  else if(strncmp("DBG:", argv[i], 4) == 0) {
    printf("Debug Mode On.\n\r");
    Debug = 1;
  }
}
```

program location and name. If `argc = 1`, then the first parameter is at index = 0 (program name) and `argv[1]` is a NULL pointer.

`main()` is supposed to return an integer reporting the status of the program. `EXIT_SUCCESS` will be a `#define` that you can find in the header file `XXXX.h`.

The program supports three command-line parameters. They are "IN:" (the input filename) "OUT:" (the output filename), and "DBG:" (Debug mode is on). In lines 157 to 173, in the `PlcCvrtMain.c` file posted on the *Circuit Cellar* FTP site, you can see how I extract the command-line parameters (see Listing 1).

Note that I use the `DBG` command line to force input and output file names and set a variable `debug = 1`. This is very useful when printing out values while debugging. It can easily be removed for "production."

The loop looks at each of the arguments in turn. Then, for each argument, it tests to see if it is one of the supported names. This test is done with the C function `strncmp`.

A character-by-character comparison

is performed:

```
#include <string.h>
int strncmp(const char *str1,
           const char *str2, size_t count);
```

The `strncmp()` function compares at most `count` characters of `str1` and `str2`. The return value is as follows: less than zero means `str1` is less than `str2`; equal to zero means `str1` is equal to `str2`; and greater than zero means `str1` is greater than `str2`. If there are less than `count` characters in either string, then the comparison stops after the first null termination.^[1]

I used `strncmp` instead of `strcmp`, so I could be sure the strings matched exactly.

Note the commented out line `printf("argc %d = %s\n\r", i, argv[i]);`. This line was used to help debug the command line parameters. It uses C's `printf()` function (www.cppreference.com/stdio/printf.html for one explanation). `Printf()` sends a formatted string to the standard output device (`stdout`). Embedded systems probably don't have a standard output device. While

Listing 2—This is how I open the output file for writing.

```
// Open File for output
fp_output = fopen(OutFileName, "w"); // Write in Text mode
if (fp_output == NULL) { // could not open the output file
    printf("Output File Not Opened.\n\r");
    return(2); // error
}
printf("Output File Successfully Opened.\n\r");
```

**MSP430
CPU12
ARM
AVR**

Easy to Use!
Low Cost!
State of the Art!

C
compilers

starting at
\$199!

high powered
development,
garage powered
prices!

Fully Functional
Demos!

Plus:
AVR, ARM & HCS12 Hardware Kits

www.imagecraft.com
info@imagecraft.com
(650) 493-9326 • FAX (650) 493-9329

**Embedded DSP Hands-On
Workshops**

Avoid confusing datasheets.

Rapidly learn to program, design
and debug your application.

Reinforce course concepts with
included development tools.


Let us help you accelerate your
design process. Visit:
www.dsp-workshops.com

Best Home LED

PAR30
7W
625 Lumens

- Newest technology
- Brightest, most efficient LEDs
- Energy saving
- Long life
- Cool operation

www.besthomeLEDlighting.com
for a wide selection of LED bulbs,
low prices and free shipping!



aimtec
Your Power Partner

60 watt AC-DC Converter
One of the smallest footprints in the world

AME60-Z series offers:

- Universal Input voltages of 90~260
- Output voltages of 5, 12, 15, 24V
- I/O isolation 3000VAC

Over 3,400 searchable Aimtec products available at **DComponents (802) 752-4321**

DComponents.com

Affordable Test Gear

Clean used test equipment at reasonable prices

- Meters
- Oscilloscopes
- Power Supplies
- Logic Analyzers
- Function Generators
- Laboratory Equipment

Personalized service adapted to the special needs of small businesses and individual developers.

Your satisfaction is our top priority.

AffordableTestGear.com



Advertising Calendar

November Issue #208
Analog Techniques
Space Close: Sept. 12

December Issue #209
Graphics & Video
Space Close: Oct. 10

Call: Shannon Barraclough
(860) 872-3064
Shannon@circuitcellar.com

on a PC running DOS, the DOS window is the standard output device. When an embedded system has an output, I use the `sprintf()` function. That is the same as `printf()`, but the output is placed in a buffer and can be sent to the port supported in the hardware. Two very useful functions indeed.

The actual extraction of each parameter is done in the following:

```
strcpy(InFileName, &argv[i][3]);
```

The statement copies the characters starting at `argv[i][3]` (just past the string "IN:") until the null character is reached into `InFileName`. And `InFileName` is an array of characters large enough to hold my parameters. What if someone enters a parameter that's larger than the `InFileName` array? Well, this program performs erratically. By the way, that's a technique used by a number of the security attacks in an attempt to gain access to a system. Note that the code prints successful parameters extracted from the command line with the `printf` function.

The next operation the main loop attempts is to open the input file for reading and the output file for writing.

COMPILER

The C compiler I'm using has file I/O routines. These can be very compiler and environment dependent. That's where portability becomes a major headache for developers. Writing one piece of code that runs on Windows, Apple, and Linux machines is difficult at best. The `fopen` command in Listing 2 attempts to open the output file for writing. If successful, a pointer to the stream associated with that file is returned. If an error occurs the null pointer is returned. I next check for a success file opening, and if there is an error, I report it and exit the program. If successful, I also report it and continue.

What is the stream associated with that file? Well, I suspect it's a struc-

Function	Command
Load module	InstallMyModule()
Open device	OpenMyModule()
Read device	ReadMyModule()
Write device	WriteMyModule()
Close device	CloseMyModule()
Remove module	RemoveMyModule()

Table 1—These are device driver events and their associated interfacing functions.

ture containing pointers to functions used to manipulate the device. You can look up how to write device drivers in Linux or Unix to see one implementation of streams. You'll see tables that look like Table 1. The routines in the table are the routines available for device manipulation and what the stream pointer would point to.

CONVERSION

The first thing needed for the output file is a standard heading. This heading is the same for all boards processed, and it consists of a sequence of commands. I wrote that heading data to the output file with the statements in Listing 3.

The `sprintf(outbuf, "PCBNAME=DS1315 \n");` statement copies the string delimited by the quotation marks into the output buffer (`outbuf`), while `fputs(outbuf, fp_output);` sends that output buffer (`outbuf`) to the output file (`fp_output`) stream. I could have used an array of strings and a pointer or index passing each string to the output file, different approaches. In this approach, the data is part of the code while, as an array, the data is separated and perhaps easier to support different header information. I could do that as another command line parameter.

Note that the `\n` character is a special character. It's the new line character. There are several of these special or nonprintable characters that will be useful in your embedded systems work.

After the header information, I process each line in the input file and make up an

Listing 3—This is a partial listing of the code used to write the standard header information to the output file.

```
sprintf(outbuf, "PCBNAME=DS1315 \n");
(void) fputs(outbuf, fp_output);
sprintf(outbuf, "&B.COMMENT=/ \n");
(void) fputs(outbuf, fp_output);
```


associated line in the output file. The next step is to read the input file line by line and separate each line into the different fields. Lines 17 through 58 describe the fields and their contents. Each field is separated by space characters or end of line. Not all fields are used, but all are parsed. All of this starts on line 269. After clearing out the array `inbuf[]`, the call to `fgets` reads data from the input file until a new line character is read or until `IN_BUF_SIZE-1` characters are read. I defined `IN_BUF_SIZE` to be 300, so we should never read a line longer than 300 characters. If data is read, pass that data to `int ExtractFields(char*source)`. And that procedure nulls out the fields and calls `int GetNextField(char *source, int *pos, char *Fld)`. `GetNextField` accepts a starting position and skips spaces until either data is found or no more data is remaining. Its return value signifies which is the case.

When you've reached the end of the input line, process the input fields. The first field is either a "Y" or an "N" character. If it's a "Y," the input line is processed (line 283 in the `PlcCvrtMain.c` file). Processing consists of writing to the `outbuf` the formatted string from line 290, appending to that line a value based on `Field[9]` (refer to line 293 in the `PlcCvrtMain.c` file posted on the *Circuit Cellar* FTP), and then the last `Field[10]` (line 300 in the `PlcCvrtMain.c` file). I don't have the room to describe all the formatting options available in the `printf()` or `sprintf()` functions, but they are well documented in the literature and on the web.

Notice all input fields are read, but not all are sent to the output file. Also notice how straightforward it would be to change the function of this program if the contents of the fields were to change.

The last step in the conversion process is to exit the main procedure when there is no more data to process. Line 312 in the `PlcCvrtMain.c` file does a return with a parameter of 0. This is useful if you wrote a batch file to process several input files at once. And error returns could be handled in that batch file. Don't you just love DOS?

MOVING ON

Well, did you notice that I did not go into a detailed explanation of each line

of C code as I presented this work? I hope that as you read each of these articles your working knowledge of the C language is growing. If you're just jumping into this series of articles, I suggest you start from the beginning as a review.

Three files are available on the *Circuit Cellar* FTP site. `PlcCvrtMain.c` is the only C file in this project. `TestIn.txt` is a test input data file. `TestOut.txt` is the output of this program.

Next time I'll focus on flash, flash, and more flash. ☒

George Martin (gmartin@circuitcellar.com) 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). George's 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 information. George is a nationally ranked revolver shooter.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2007/206.

REFERENCE

- [1] N. Kohl, "Standard C String and Character: `strncmp`," *C/C++ Reference*, www.cppreference.com/std/string/strncmp.html.

SOURCES

Protel DXP electronic design package
Altium
www.altium.com

C++ BuilderX IDE and Turbo C++ 3.0 IDE
Borland Software Corp.
www.borland.com

ColdFire microprocessor family
Freescale Semiconductor, Inc.
www.freescale.com

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





Robot Kits
Line followers
Robot arms
Hexapods
Chassis



Mechanical Components
Gearboxes, servos
Wheels, ball casters



Motion Control
Servo controllers
Motor controllers



Robot Controllers

voltage regulator
power LED (green)
red user LED (on PD1)
ATmega48/168 microcontroller
20 MHz clock
dual H-bridge
trimmer pot (on ADC7)
programming connector

Solder Paste Stencils

Use our low-cost solder paste stencils to quickly assemble your surface-mount designs.

From \$25.



Custom Laser Cutting

From \$35



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

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



I-Zip Dashboard

Jeff applies what you learned last month to design a display for his electric bicycle. The system uses a LIN bus to present real-time information about your driving habits and fuel efficiency. The system displays information such as voltage, current, slope, distance, and speed data.

What would Henry Ford have to say about the advances in the automobile industry if he were alive today? While environmental impact was not at the heart of Ford's River Rouge Plant (a self-sufficient automobile factory), Ford's plan could have been success-

ful even if it had had a green foundation. Not that America would be any closer to becoming the big green transportation machine. Although change might seem driven by customer demand, it's an excruciatingly slow process.

The oil companies have us over the proverbial barrel. We are not in a position to totally stop buying gas-guzzling vehicles. Without an alternative, we are at the whim of (sometimes daily) rising pump prices. Can we blame the automakers and oil companies for want-

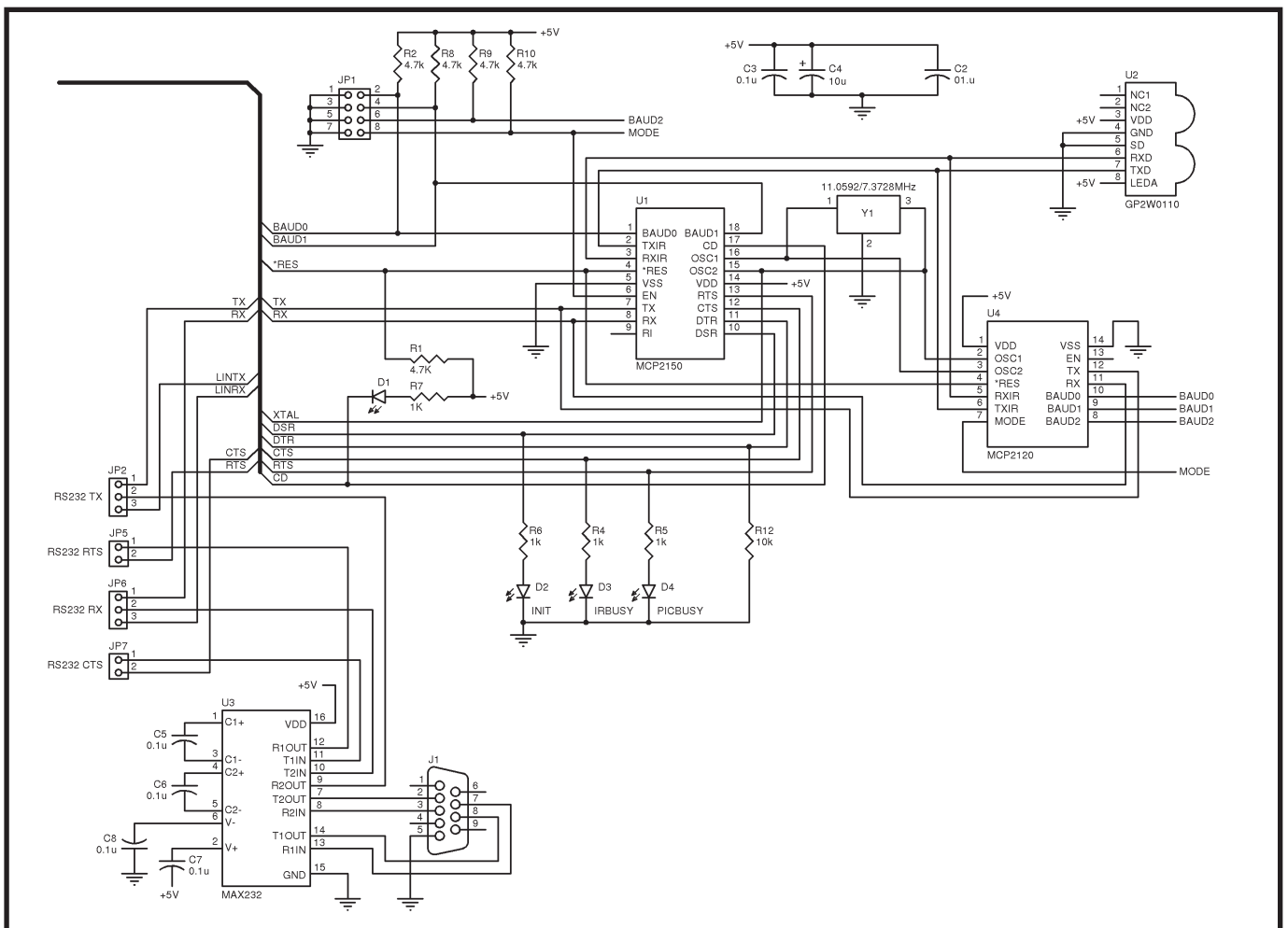


Figure 1—The circuitry shown here adds an RS-232 interface to help debug the application. Also, support for an IrDA link (using a Microchip MCP2150 or an MCP2120) has been added for future experimentation.

ing to continue making profits while they jockey for control of our transportation future? After all, our country is based on capitalism.

I don't pretend to have the answer. As U.S. citizens, we elect people to make decisions for us (hopefully in our best interests). Although you may vote, voter turnout is disgustingly low in the U.S. Have you ever written or called any of your local or state representatives? Even if you voted for "the other guy," you still have a right—no, make that a responsibility—to be a part of the decision process. While I don't like suggesting that our silence puts a stamp of approval on bad decisions with little constituent input, a lack of direction allows political action committees (PACs) to have a great impact.^[1] While a PAC is not necessarily a bad thing, each has its own agenda and may not be lobbying for your best interests. So, help direct your country, or be satisfied living in theirs.

Consumers have encouraged automobile designers to provide increased safety protection and a more comfortable internal environment. Many vehicles provide feedback. In addition to providing speed and distance data, some vehicles provide real-time data about your driving habits and how they are affecting fuel efficiency. You see that when you put the pedal to the metal, mpg suffers. The best fuel economy is achieved when you treat the gas pedal as though there is a raw egg between it and your lead foot. Compass or GPS circuitry can inform you of the

direction you are travelling, pinpoint your position, or even tell you how to get to the nearest Chinese restaurant. To many, a vehicle is a home away from home, and as such, most want their car to have many of those comforts. They include improved climate control, a super audio or even video entertainment center, and adaptable suspension and braking systems.

Now that I've spent a number of columns designing the basics for my electric bicycle project, I want to do something with the sensor data that has been networked. Last month, I described how to use a Palm device to monitor internal registers in a microprocessor with NS Basic. It's time to apply it to a project that will display data passing on the local interconnect network (LIN) in real time: a dashboard display, if you will, using the tools previously discussed.

WHERE'S THE DATA FROM?

Up to this point, I've covered measuring battery voltage (24-V gel cells), electric motor current (brushed DC motor), collecting data with a USB thumb drive interface, and measuring the slope of the roadway (accelerometer) and travel distance (optical encoders). In "Local Interconnect Network," I discussed the LIN bus (*Circuit Cellar* 201, 2007). Automotive manufacturers are moving to

include networks to simplify wiring harnesses and ultimately decrease the weight of vehicles. To keep my mission relative to this technology, I chose the LIN bus as the communication medium for the electric vehicle (bicycle) project. The protocol for the LIN bus uses a format that requests or provides data packets initiated by a master node. A LIN message consists of a header that provides the packet structure and identifies who should respond. The slave node completes the frame with data and a checksum. Because the master node is "in charge," it must schedule frame traffic. In this project, the master is also a slave node (node 0). A second slope and distance module is node 1, and

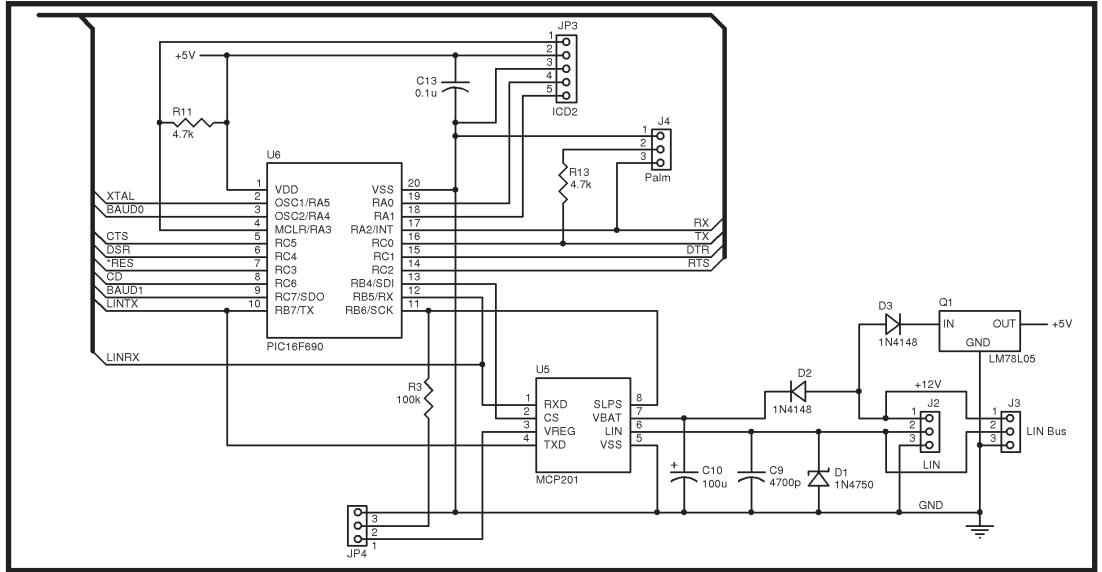


Figure 2—LIN bus monitoring is handled by the Microchip PIC16F690 through the MCP201. The bit-banged serial to the Palm Visor attaches via J4.



Photo 1—This screenshot of the I-Zip Dashboard shows all of the gauges and idiot lights active. Although my Visor is monochrome, newer handhelds have full-color screens.

Function	3-Byte data	xx-Hexadecimal value
Voltage	"Vxx"	00-FF (volts × 10)
Amperes	"Axx"	00-FF (amps × 10)
Slope	"Sxx"	00-7F (slope), MSBit = 1 (–slope)
Odometer	"Oxx"	00-FF (link count)
Pedals	"Pxx"	00-FF (link count)

Table 1—Monitored data is sent to the Palm display once per second. The data is formatted into an ASCII string for readability.

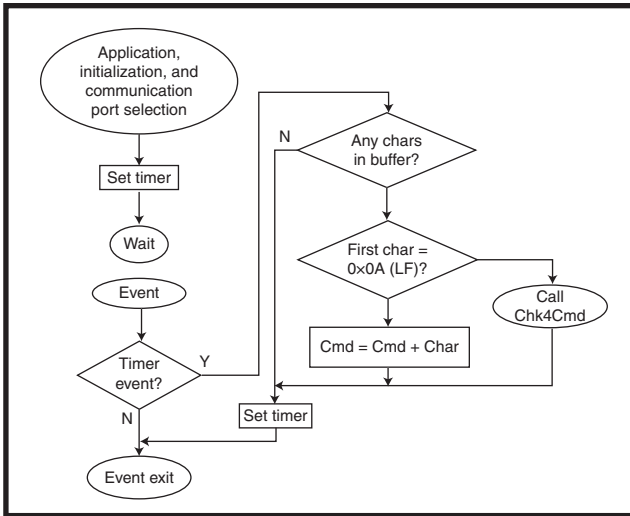


Figure 3—The “I-Zip Dashboard” is a single-form application filled with objects. After the application’s form has been initialized, no action takes place until an event occurs. If the TIMER was the cause of the event code being executed (it could be something else, such as a button press), the serial buffer is checked for data. If data exists, the first character in the buffer determines what will be done.

the project’s bus monitor module is node 2.

The first scheduled frame requests data from node 0. The second frame requests data from node 1. The schedule calls for the frames to be repeated every 1 s. Node 0 provides the volts and amps data, and node 1 provides slope and odometer data. This month’s project is a silent node. By design, it offers no data. It merely serves as a LIN bus monitor and data repeater for the Palm Visor display. I designed the node with a microcontroller to referee between the LIN data and the ASCII data I wanted to transfer to the Palm. Could the Palm handle the LIN data directly? The Palm doesn’t have the LIN-supported EUART hardware, which is found in some microcontrollers, so I wasn’t willing to investigate a potential dead end with an impending deadline approaching. Besides, passing ASCII data makes debugging the Palm interface a lot friendlier.

Using a microcontroller in the design of this module also gave me a chance to add other hardware I wanted to eventually experiment with (see Figures 1 and 2). You can see the design has a couple of IrDA devices on board, as well as an RS-232 interface opening up a number of options, such as an IrDA to RS-232 or Palm link.

The module will need the LIN rou-

tinues to collect data. The requirements for the project include listening only to the LIN bus and sending data to the Palm device so the code is simplified.

Everything is based on LIN bus activity. The LIN master polls slave modules 0 and 1 every second. Slave module 0 responds with the volts and amps data. Slave module 1 responds with slope, odometer, and pedals data. Slave module 2 listens to the data traffic. Upon receiving the data coming from slave module 1, slave module 2 has all five pieces of data and formats it for output to the Palm device (presumably this happens at the same rate, once every second, per the LIN schedule). The data is reformatted as an ASCII string so it can be easily viewed using any terminal program (see Table 1). This makes debugging much easier since you can read or print what is being transmitted.

THE DASHBOARD

The whole idea was to use something that was inexpensive and easy to program. We are all encouraged to be “green” by making use of stuff that is collecting dust, too good to throw out, but no longer cutting edge. My Palm Visor makes a great user I/O device. Programming special applications with NS Basic is easy and affordable. I started this project by designing what I wanted my dashboard to look

like. There are a number of objects on the form in Photo 1. Other than the button labeled “Trip Reset” and a normally invisible text field with the word “Test,” each gauge is an object created in its own little window. These windows are modular and can easily be moved around (to reconfigure the display).

Since each window has its own identity, commands used while a window is “in focus” are referenced in the upper-left corner of that window. You can reposition the window without having to worry about what’s happening within the window. I use most of the graphic commands in various windows, including text, line drawing, and bitmaps. The speedometer and gas tank gauges are bar graphs made by drawing open rectangles to represent the graph and closed rectangles to show an amount. Text below the graph labels its function.

Two windows that can display (or hide) bitmap images are in the lower left. I created a couple of small images with Microsoft Paint. They were designed to act like idiot lights. A battery icon is displayed when the electric motor supplies power to the motor. The pedals icon is displayed

like. There are a number of objects on the form in Photo 1. Other than the button labeled “Trip Reset” and a normally invisible text field with the word “Test,” each gauge is an object created in its own little window. These windows are modular and can easily be moved around (to reconfigure the display).

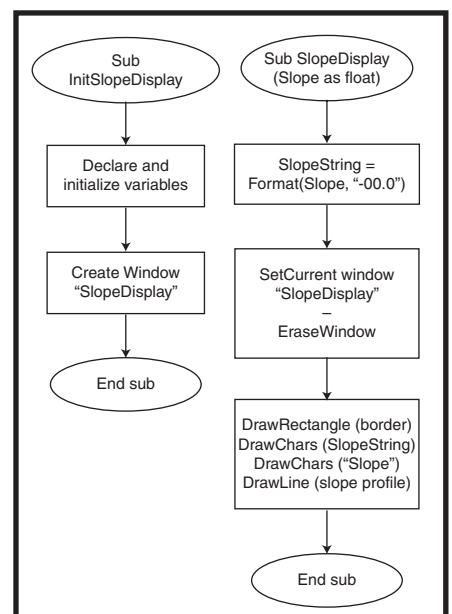


Figure 4—Every “gauge” is treated as a separate object (graphic window) on the application’s form. Each object must be initialized (created) when the form is opened (left-hand flow called from initialization). The TIMER event can call individual objects (i.e., this slope-display subroutine) to update the display (right-hand flow called from Check4Cmd).

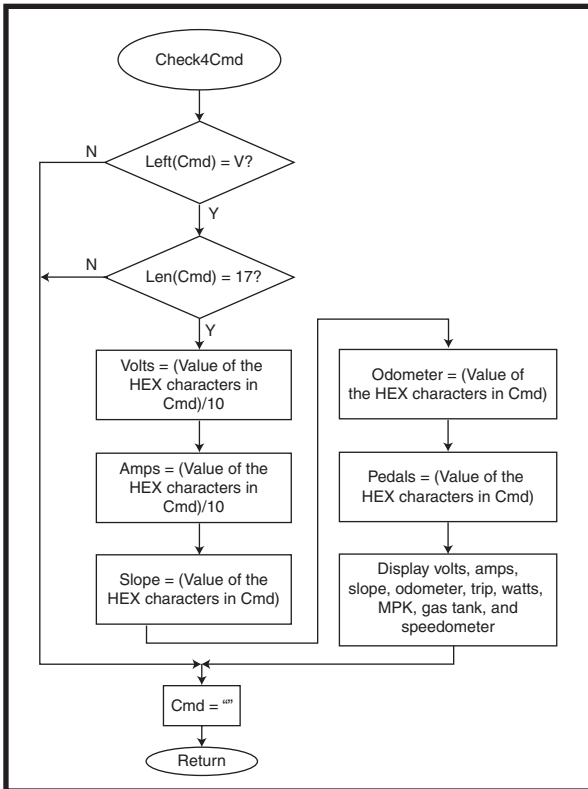


Figure 5—When the first character of string Cmd matches one of the commands, the HEX data following the command character is evaluated. If the new value (for the variable) has changed, a display routine is called to update the display.

when you apply force to the pedals. The endearing term “idiot light” comes from a single light on an automobile’s dashboard, which indicates a serious (but vague) engine issue.

One gauge most vehicles do not offer is an indication of slope. Since it plays a major part in this project, I wanted to not only indicate the measured slope, but also provide a visual representation of it by drawing the slope as a varying horizon. This is the only data that is presented as a signed value. Therefore, the slope is displayed as uphill (when positive) and downhill (when negative).

The remaining gauges are mainly text displays (except for drawn borders). With only five pieces of information (volts, amps, slope, odometer, and pedals), this application must calculate data for a number of gauges. The odometer and trip gauges are the only cumulative display of data in this project. I haven’t included the instructions to “remember” these values in an associated file. There may be other things I want to save in this applica-

tion (like a log of all the received commands), so I’ll table this issue for now. In the meantime, the cumulative values revert to zero if the application is exited.

Here are a couple of other areas of interest. The gas tank is based on the battery voltage. If you remember, lead-acid cells are considered discharged at a voltage of 1.75 V per cell (i.e., 21 V = 1.75 V × 12 cells) and fully charged at 2.3 V per cell (i.e., 27.6 V no load). With a single byte used to indicate battery voltage, the highest voltage represented is 25.5 V, with a nominal operating voltage of approximately 25.2 V. I used two constants to represent the “full” voltage and “empty” voltage of the gas tank. The actual voltage decay curve of

the battery is fairly linear (except at the extremes), so the gauge level is a reflection of the voltage measurement within the two limits. I could have added an idiot light with a picture of a gas can to indicate a low warning, but I chose not to.

The odometer and pedal values also have constants associated with them. Because the data merely counts the number of chain links passing the sensors, a constant is needed to convert the counts into actual distance. Each constant is determined by the chain’s pitch, the rear gear’s tooth count, and the rear tire’s circumference:

$$\frac{\text{Distance}}{\text{Count}} = \frac{\text{Circumference}}{\text{Teeth}}$$

$$\frac{(\text{Odometer}) \text{ Distance}}{\text{Count}} = \frac{96''}{60} = 1.6''$$

$$\frac{(\text{Pedals}) \text{ Distance}}{\text{Count}} = \frac{96''}{20} = 4.8''$$

While the constants applied to the counts give inches of travel (the actual values are saved in the variables odometer and pedals), that value is not displayed. The gauge uses a more

standard miles (and tenths) display. This means that the odometer must be converted from inches to feet, miles, and tenths by another calculation in the display routine:

$$\frac{(\text{Odometer}) \text{ Distance}}{\text{Count}} = \frac{1.6''}{12'' \text{ (per foot)}} = 0.133'$$

$$\frac{(\text{Pedals}) \text{ Distance}}{\text{Count}} = \frac{4.8''}{12'' \text{ (per foot)}} = 0.4'$$

$$\text{Tenths (of a mile)} = \frac{\text{Odometer}}{\text{(feet per mile)}} \div \text{tenths}$$

$$\text{Tenths (of a mile)} = \frac{\text{Odometer}}{\frac{5,280}{10}} = \frac{\text{Odometer}}{528}$$

Electric motor power is easily calculated using volts × amps. The watts gauge is real-time power, not accumulated usage. My older caravan had an information console that showed actual MPG usage. This was informative for revealing driving techniques that increase and decrease MPG. I added a miles per kilowatt-hour (MPK) gauge as an indication of costs based on distance and watt-seconds. This is based on the odometer value and the watts value:

$$\text{MPK} = \frac{\text{Distance (miles)}}{\text{Power (kilowatts)}}$$

$$\text{MPK} = \frac{\frac{\text{Odometer}}{\text{feet per mile}}}{\frac{\text{Watts}}{1,000}}$$

$$\text{MPK} = \frac{\frac{\text{Odometer}}{5,280}}{\frac{\text{Watts}}{1,000}} = \frac{\text{Odometer}}{\text{Watts} \times 52.8}$$

Of course, if you use only the pedals, this equation attempts to calculate an infinite number and creates a divide-by-zero error as the watts = 0. Some checking here is done to display a special value for MPK (999.99 in this instance).

SERIAL COMMUNICATION

NS Basic provides a simple programming language that lets you write and download an executable program to any handheld using the Palm OS. With NS Basic, I can collect, manipulate, and display data with the Palm’s LCD. For this project, the application boils down to one rou-

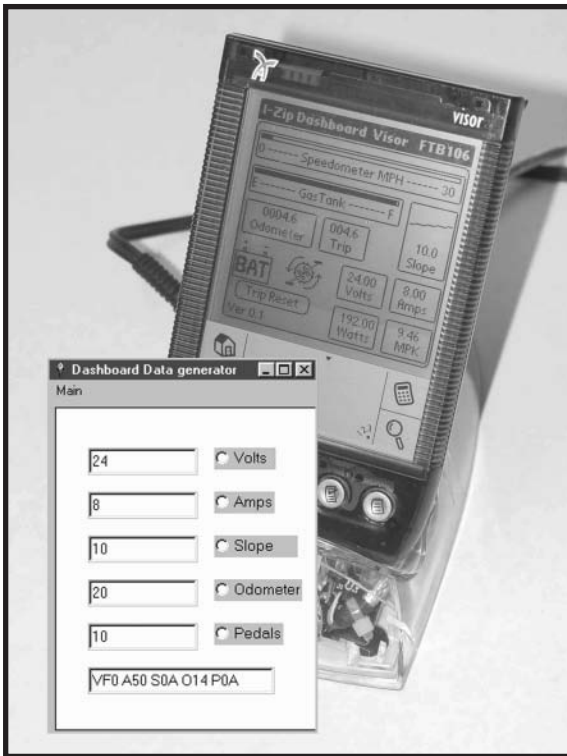


Photo 2—I developed a small Liberty Basic program (inset) for my PC that sends a string of data to the Palm through the serial port. With this program, I can set any of the five data values to help debug the NS Basic program running on the Palm.

tine that retrieves and displays data. The latest version of NS Basic includes a “SerialEvent” function. Unfortunately, my old Handspring Visor does not support this function, so I must use a timer and periodically check for characters received in the serial buffer.

Take a look at Figures 3, 4, and 5 to get a feel for the program flow in this application. Characters are removed from the serial input buffer on a FIFO basis. The character should be a command (capital letter), a hexadecimal character, a carriage return <cr>, or a line feed <lf>. If the character is not an <lf>, then the character is added to the Cmd string. If the character is an <lf>, the Cmd string is evaluated for data based on its length (which should be 17 characters) and the first character (which should be V). Separate routines extract the five values from the Cmd string and update the appropriate variables. For instance, if the beginning of the Cmd string equals VFF, the value 255 is used to update the variable volts: volts = value/10, or in this case volts = 25.5.

This application receives data, reformatted by the associated LIN module, and updates its application’s variables. A change in the variable’s value requires not only a change to that specific gauge on the display, but also to the other objects that may be dependent on that data (i.e., watts = volts × amps). It is up to you to decide what information should be updated upon changed data.

IS THIS IT?

This project will be taking a break here. Now that the weather is warm, I’m taking the time to get the last modules mounted and the whole system fully operational. I’ll wrap this up after I put some miles on the bike. Then I can give a report on how well it

functions and what improvements might have been made.

Using a Palm (like my Visor) for user I/O makes a lot of sense. It will be difficult to come up with a less expensive touchscreen interface that is so easily programmed. While newer Palm devices offer a plethora of interface possibilities (i.e., IR, USB, and Bluetooth), the old standard three-wire serial is still the easiest to use. Sometimes I find it necessary to write a small application like the “Dashboard Data Generator” just to help with the debugging process (see Photo 2).

Although I opted to use a single screen for the entire dashboard, I did give some thought to a dashboard that acts like a slideshow and displays each full-screen gauge for a few seconds (in round robin fashion). Also, with the processing power of the Palm device, values could be presented in various formats (i.e., English and metric).

Since the Palm device has the potential to send data back to the system, the display could become more integrated if slave module 2 (this

month’s project) was designed to do more than just listen on the LIN bus. The module might pass requests back from the display. A request might be a simple task like turning the lights on and off or something more intensive, such as implementing “cruise control.” Hmm, cruise control on a bicycle, that would be overkill, don’t you think? But wait, the design is not meant just for a bicycle. The bike just happened to be a handy (and inexpensive) platform to get started with. All of this is applicable to bigger and more useful transportation vehicles. After a little experience with this design, I’ll be setting my sights on finding a road-worthy platform capable of passing the necessary Department of Motor Vehicles inspections for registration. 📧

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

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2007/206.

REFERENCE

- [1] The Leadership Institute, “Business and Association PAC Study,” 2005, <http://pacstudy.leadershipinstitute.org>.

SOURCES

MCP201 LIN Transceiver, MCP2120 developer board, MCP2150 developer board, and PIC16F690 flash-based microcontroller
Microchip Technology, Inc.
www.microchip.com

NS Basic Programming Language
NS BASIC Corp.
www.nsbasic.com

Handheld Visor
Palm, Inc.
www.palm.com

Only 4 Steps...

...are required to generate efficient, reliable applications with the μ Vision IDE and development tools from Keil.

Step 1. Select Microcontroller and Specify Target Hardware

Use the Keil Device Database (www.keil.com/dd) to find the optimum microcontroller for your application.

In μ Vision, select the microcontroller to pre-configure tools and obtain CPU startup code.

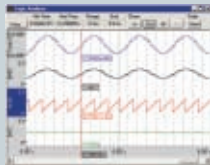
Step 2. Configure the Device and Create Application Code

The μ Vision Configuration Wizard helps you tailor startup code to match your target hardware and application requirements.

Extensive program examples and project templates help you jump-start your designs.

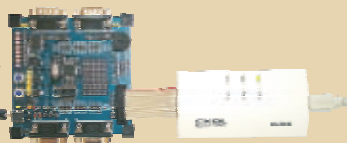
Step 3. Verify Program Execution with Device Simulation

High-speed simulation enables testing before hardware is available and helps you with features like instruction trace, code coverage, and logic analysis.

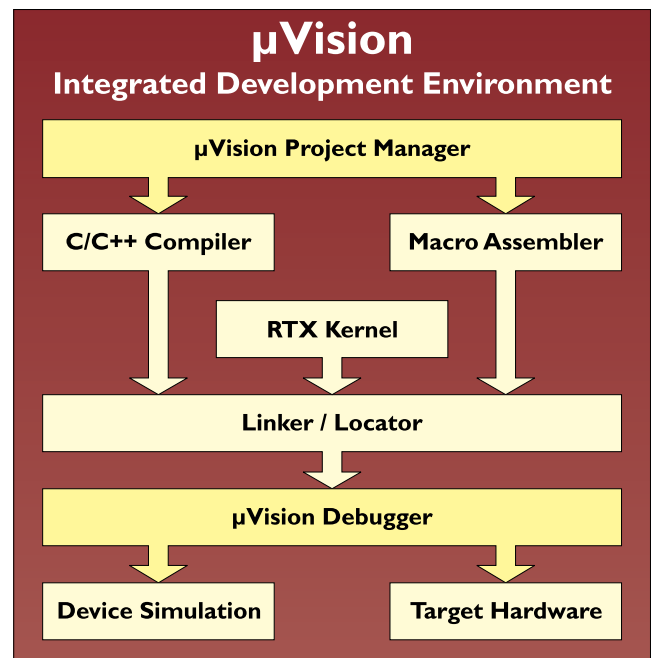


Step 4. Download to Flash and Test Application

Once your application is runs in simulation, use the Keil ULINK USB-JTAG Adapter for Flash programming and final application testing.



Keil Microcontroller Development Tools help you create embedded applications quickly and accurately. Keil tools are easy to learn and use, yet powerful enough for the most demanding microcontroller projects.



Components of Keil Microcontroller Development Kits

Keil makes C compilers, macro assemblers, real-time kernels, debuggers, simulators, evaluation boards, and emulators.

Over 1,200 MCU devices are supported for:

- **8-bit** - 8051 and extended 8051 variants
- **16-bit** - C166, XC166, and ST10
- **32-bit** - ARM7, ARM9, and Cortex-M3

Download an evaluation version from www.keil.com/demo



Game On

When the coding gets tough, the tough get coding. And the rest of us should be glad they do. Recently, Tom turned to game expert André LaMothe to see how to teach new-school hardware some old-school programming tricks. Are you game?

Let me start this month by saying I'm not really into playing video games. Nothing personal, just a matter of other priorities competing for my increasingly nonexistent spare time.

That being said, I have nothing but respect for the wizards who make miracles happen under the hood. Video games have always pushed the limits of designers' ingenuity to meet players' seemingly insatiable demand for faster and more realistic thrills and spills.

As someone who grew up in an era of computing limits, I appreciate that game designers seem to be the last of a shrinking pool of those who really understand what's going on inside the box and aren't afraid to get their hands dirty optimizing every bit and cycle.

Which brings us to this month's topic. Let's get into some old-school gaming on some very new-school hardware.

8 HEADS BETTER THAN 1

Of course, when I'm talking about "new-school" hardware, I'm talking about the multicore chips that are all the rage.

Keep in mind that the concept of "parallel processing" isn't new. Practically since the dawn of computing, designers have wondered if there was a way to combine "n" of the computers they've got in hand already rather than struggling to come up with a new "n" times more powerful single CPU.

Nevertheless, until recently, the latter path was generally chosen as the one of least resistance. Sure, parallel

processing sounds good in principle, but in the meantime, let's just add more cache and crank up the clock rate.

Taking the easy way out worked for a long time, but now it's falling out of favor. The era of the big-iron single processor uber-chip is coming to an end in the face of critical "hot-spot" power issues and declining ROI for baroque architectural trinkets like out-of-order execution and speculation.

So, finally after all these years, parallel processing here we come, ready or not. Since most folks fall into the latter category, my inbox is filled with announcements for conference papers like "How To Think Algorithmically In Parallel"^[1] and "Concurrent Programming for Modern Architectures."^[2]

About a year ago, I covered the Propeller chip from Parallax ("Turning the Core-ner," *Circuit Cellar* 193, 2006). With eight 20-MIPs processors crammed onto a low-priced chip (\$12.95 quantity one), it's the first multicore for the masses. This month, let's take a look at one intriguing, and dare I say fun, Propeller-based design, André LaMothe's aptly named Hydra (see Photo 1).

I would have written this column a few months ago, but I must confess it took me a while to get through LaMothe's 800-plus page book, *Game Programming for the Propeller-Powered Hydra*, which goes well beyond the typical user guide or datasheet. The entire kit, including the book, the Hydra SBC, and a bunch of add-ons (e.g., keyboard, mouse, gamepad, cables, etc.) is \$199.95. Or you can get the book and accompanying CD (many megabytes of Propeller code) separately for \$39.95.

Let me say at the outset that, despite the title, LaMothe's book isn't just for gamers. Although many of the chapters and code examples have a game-centric slant, much of what's discussed is generally applicable and serves as a worthy supplement to Parallax's own Propeller documentation. But, even if you're not into games, take a moment

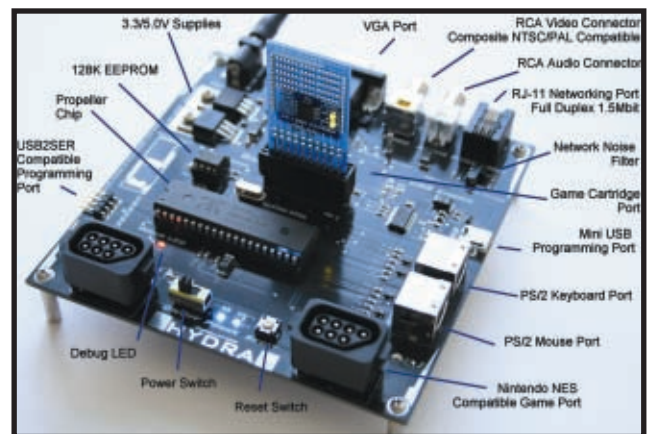


Photo 1—Based on the multiheaded Propeller chip, the aptly named Hydra SBC is long on I/O interfaces, but short on extra chips. The idea is that with enough MIPS on tap, in this case 160 peak MIPS for the 8 x 20 MIPS Propeller, software can replace silicon.

At FTDI, you'll find more than just our acclaimed range of USB Interface Chips...

We also have a wide range of competitively priced USB interface modules and cables ideal for implementing one-off designs through to full mass production runs. Our on-line shop accepts payment by credit card - no minimum order restrictions apply

TTL-232R USB to TTL Serial I/F Cable

\$23.11



- Easy solution for attaching 5v / 3.3v MCU to USB
- Fully 5v compatible I/O signal levels
- 3.3v I/O version also available
- TTL UART interface
- FTDI VCP or D2XX Drivers available
- 1.8m long 6 way cable
- 6 pin SIL pin socket (0.1in pitch)
- FT232RQ embedded into USB type "A" plug
- Data transfer rates from 300 baud to 3M baud
- Optional RTS/CTS or X-ON/X-OFF handshaking

MM232R Miniature USB Module

\$20.82



- Micro Miniature USB Module
- FT232RQ USB UART
- 0.1in Pitch Pinout
- TXD, RXD, RTS#, CTS# UART Interface Pins
- Communication from 300 baud to 3M baud
- Four configurable GPIO Pins including clock output
- USB Self / Bus powered options
- 3.3v / 5v I/O signal level options

UM232R / UM245R DIL Modules

\$23.11



- 24 Pin DIL format USB Modules
- FT232RL USB UART (UM232R)
- FT245RL USB FIFO (UM245R)
- Turned pins fit standard 24 pin i.c. socket
- USB Self / Bus powered options
- 3.3v / 5v I/O signal level options
- Full set of UART Interface Pins (UM232R)
- All multi-function CBUS GPIO Pins available (UM232R)
- Power Enable control available (UM245R)

VMUSIC2

\$43.41



- Not only can you easily add your USB flash disk to your product but you can also play back MP3 and other popular digital music from your USB
- Uses FTDI's USB host controller Vinculum VNC1L
- Single 5V supply input
- USB 'A' type socket
- Stereo 3.5mm headphone jack socket for audio playback
- Jumper selectable UART or SPI interfaces
- VMUSIC2 is Pb-free and RoHS compliant

VDRIVE2

\$28.32

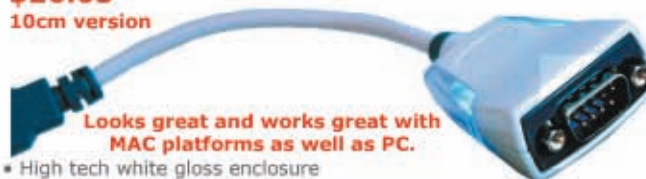


- An easy solution for adding a USB flash disk to an existing product
- Easy front panel mounting
- Uses FTDI's USB host controller Vinculum VNC1L
- Only 4 signal lines and 5V power supply plus ground are needed
- 2 mm (0.08") pitch 8 pin connector
- 8-way header interconnect cable provided
- Jumper selectable UART or SPI interfaces
- VDRIVE2 is Pb-free and RoHS compliant

US232R-10 Premium USB - RS232 Cable

\$26.03

10cm version



Looks great and works great with MAC platforms as well as PC.

- High tech white gloss enclosure
- Blue side-lit LED TX and RX traffic indicators
- Gold Plated USB and DB9 connectors
- Supplied in retail packaging with driver CD
- Communication rates from 300 baud to 1M baud
- 10cm cable length (1m version available at \$24.98)

VDIP1

\$28.32



- The VDIP1 module is an MCU to embedded USB host controller VNC1L
- VDIP1 is supplied with pre-loaded Vinculum VDAC firmware
- USB 'A' type socket to interface with USB peripheral devices
- Single 5V supply input
- Jumper selectable UART, parallel FIFO or SPI MCU interfaces
- Auxillary 3.3 V/ 200mA power output to external logic
- Designed to fit into a 24 pin DIP socket
- VDIP1 Pb-free and RoHS compliant

UC232R-10 Economy USB RS232 Cable

\$21.86

10cm version



- Matt finish, nicely sculpted white plastic enclosure
- Supplied loose packed in an anti-static bag
- Wide range of drivers downloadable from FTDI web site
- Communication rates from 300 baud to 230k baud
- 10cm cable length (1m available to special request only)

Future Technology Devices International Limited

7235 NW Evergreen Parkway Suite 600

Hillsboro, OR 97124-5803

USA

Tel: +1 (503) 547-0988

E-Mail : us.shop@ftdichip.com

Web : www.ftdichip.com



Volume Discounts Available

Over 50 different USB Interface products in stock at www.ftdichip.com/chipshop.htm



Photo 2—Check out this '50s version of a “media center,” including what some feel was the first “computer” (albeit analog) “video” (albeit using an oscilloscope rather than a CRT) game. In 1958, William Higinbotham of Brookhaven National Laboratories crafted his Pong-like “Tennis for Two” game.^[9]

to read the first chapter (which is Chapter “0” naturally), an informative and entertaining review of the history of video games going all the way back to (and even predating) Pong (see Photo 2).

READ AND HEED

The book is divided into three major parts. After a bit of housekeeping in the form of software installation and a “Quick Start” board checkout, the first part covers the design of the Hydra SBC itself with a chapter devoted

to games, keep in mind that the techniques described are useful for any application with game-like requirements (i.e., real-time, audio, video, etc.).

A fundamental premise of the Propeller chip is that general-purpose I/O pins coupled with fast and clever software is the way to go versus the traditional approach of dedicated I/O functions hardwired in silicon. That premise is certainly apparent with the Hydra design. Take a close look at the board and what you will find is a lot of

ed to each subsystem and interface. The second part, a full 250 pages, is devoted to “Propeller Chip Architecture and Programming,” including plenty of programming examples. Only the last part of the book, “Game Programming on the Hydra,” is truly gaming-centric. And even if you aren’t into


I/O connectors with very little beyond the Propeller chip itself driving them. The main exception is the USB interface, where one of the popular FTDI USB-to-serial adapter chips provides a link to the standard Propeller develop-



Photo 3—Get down tonight and shake your booty with one of the many circa '70s demo games that's included with the Hydra kit. It's more than fun and games though, since the included source code and explanatory text yield insights into clever programming techniques that can equally serve “real” applications.

FlashPro430 GangPro430


USB Flash Programmers for Texas Instruments' MSP430 microcontrollers.



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

- ✓ 60 kB Flash can be programmed in about 2 seconds
- ✓ supports JTAG, Spy-BI-Wire and BSL interfaces
- ✓ can assign unique serial numbers
- ✓ up to eight programmers can be connected to one PC and program target devices simultaneously

New: FlashPro-CC and GangPro-CC
USB Flash Programmers for CC series devices (ChipCon) from TI



One PC and 8 programmers


Programmers

USB-FPA-1 JTAG / SBW or BSL MSP430Fxx

USB-FPA-2 MSP430Fxx

USB-FPA-3 MSP430Fxx

USB-FPA-8 MSP430Fxx



www.elprotronic.com

WIRELESS MADE SIMPLE

BRING YOUR PRODUCT QUICKLY AND LEGALLY TO MARKET

RF Modules

Low-Cost TX & RX Modules

Multi-Channel Modules

Long-Range Modules

Add INSTANT wireless analog / digital capability to your product.

OEM Products

Handheld TXs

Keyfob TXs

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

Function Modules

Antennas

Specialty

GPS

Embedded Chips

Permanent Mount

From ceramic chips to gain Yagis, keyless entry to WiFi.

Whips

Gain Antennas

Magnetic Base



800-736-6677

159 Ort Lane · Merlin, OR 97532

www.linxtechnologies.com

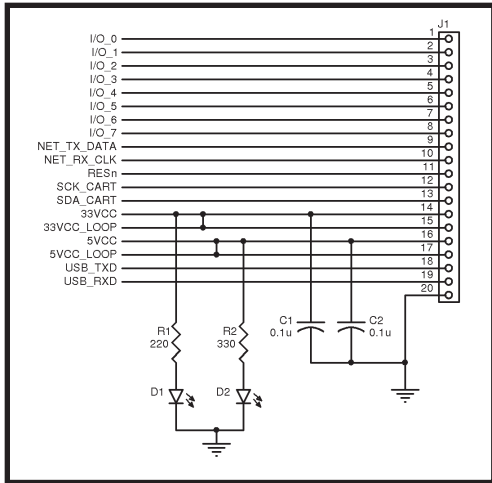


Figure 1—The Hydra SBC has a simple “cartridge” expansion connector and the kit includes one preprogrammed with a game and one bare for experimenting with your own add-ons.

ment tools (i.e., IDE, assembler, and Spin language) running on a PC.

There’s also a 128-KB serial EEPROM on the board that can boot up the Propeller chip for stand-alone (i.e., PC not connected) operation. The current version of the Propeller chip uses only the first 32 KB, but software could use the rest for application-specific purposes, such as nonvolatile settings, calibration, data logging, etc. In addition, there’s a connector that accepts small plug-in modules for memory or I/O recalling the “cartridge” plug-ins of vintage video games.

Speaking of which, the “Quick Start” section in Chapter 1 has you plug in a cartridge included in the kit that’s preprogrammed with a circa ’70s “breakout-like” game (see Photo 3). Put on some disco music and relive the days when companies like Atari and Magnavox were household names. There’s also a generic prototyping cartridge with holes on 0.1” centers for crafting your own add-ons. Late news: I just found out there’s also a memory expansion cartridge (with 128 KB of EEPROM and 512 KB of SRAM) in the works.

After a very brief introduction to the Propeller chip and IDE, Chapter 1 culminates with the installation of the FTDI USB chip virtual serial port drivers and the Parallax tools on your PC (both are included on the CD that comes with the Hydra). The final pre-flight check is to use the IDE to com-

pile and download an “Alien Invaders-like” game to verify that the software tools and PC link are good to go. Now, with the ability to drive the Propeller chip from the on-board EEPROM, a plug-in “cartridge,” or your PC, you’re ready to rumble.

The next hundred pages describe the functional blocks, with a chapter devoted to each one. Some of these chapters are pretty short. After all, it’s hard for even the most enthusiastic wordsmith to say much about a reset switch or debug LED.

The chapters describing the more muscular features are where the book really shines.

LaMothe manages to convey a lot of information in a way that is both short and sweet. For instance, the chapter on NTSC video is only a dozen or so pages and an easy, breezy read. Yet it manages to fully convey the essence of what’s actually a rather complicated subject (in the words of the author, “...one of the greatest hacks in electronics history”). The same goes for the chapters on VGA and audio. Of course, I learned all this stuff before—many times. I’ll keep LaMothe’s book handy for the next time I have to learn it again (and again).

Many of the Hydra I/O functions (including audio, video, keyboard, and mouse) are the same as their counterparts on the Parallax development board, which I covered in my earlier column. Let’s spend a few moments on the additions that are unique to Hydra.

As shown in Figure 1, the aforementioned “cartridge” connector provides a handy way to experiment with your own add-ons. Both 3.3- and 5-V power are supplied to the board, as well as “loop” connections (i.e., 3VCC_LOOP and 5VCC_LOOP), which allow Propeller software to determine that a cartridge has been inserted. For talking to the cartridge, you’ve got a choice of using a clock serial connection (as the 128-KB EEPROM car-

tridge does) and/or eight parallel I/O lines. Note that the latter are shared with the VGA port, so you can use them for I/O or VGA, but not both at the same time.

You’ve also got access to the FTDI chip (USB_TXD, USB_RXD) and something called “Hydra Net” (NET_TX_DATA, NET_RX_CLK). The latter is a scheme that LaMothe came up with on his own to connect Hydras to each other. Nothing too formal about it, really just a couple of I/O lines that you can drive as you see fit with software. For example, you could choose to implement a full-duplex UART or a half-duplex clock serial link. The Hydra Net port includes network termination circuitry, so even when using standard RJ-11 connectors and phone wires, Hydra Net works at 256 kbps up to 100 m away, and even higher speeds (e.g., 2.5 Mbps) for short distances.

A game is nothing without a gamepad, so in addition to a keyboard and mouse, the kit also includes a Super Nintendo Entertainment System (NES) controller. The Hydra board has two of the oddball NES connectors so you can plug an extra controller in if you want to go head-to-head with your friends. After playtime is over, the gamepad also makes for a handy general-purpose input gadget and it’s easy to talk to since the interface is just a shift register (see Figure 2).

HEAD SPINNING

The second part of the book, “Propeller Chip Architecture and Programming,” is probably the most

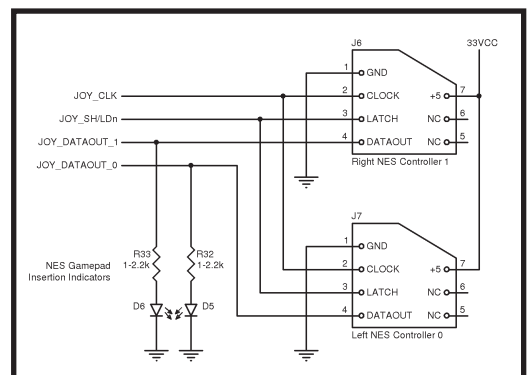


Figure 2—The hardware interface to an NES gamepad is simple; it’s just a three-pin (clock, data, shift/load) shift register. A gamepad is a handy low-cost generic input device not necessarily limited to just games.

generally applicable. Starting with an overview of the architecture, each register and assembly language instruction is described both in words and via simple examples. Going beyond this usual fare, LaMothe shows no fear in tackling some of the more esoteric capabilities of the chip.

For instance, the Propeller chip doesn't have the hardware multiplier you typically find on a chip these days. Of course, one option is the old school "shift & add" software multiply routine using the example code provided in the book. But, as with many other features of the chip, Propeller offers a thinking-outside-the-box option in the form of logarithm-based math.

You probably know that logarithms are a handy way to express large numbers with small ones. For instance, the base 10 logarithm of one million is six (i.e., $10^6 = \text{one million}$). It turns out logarithms have another cool feature, turning big-ticket multiplies and divides into simple adds and subtracts. In other words:

$$\text{Log}(A \times B) = \text{Log} A + \text{Log} B$$

$$\text{Log}\left(\frac{A}{B}\right) = \text{Log} A - \text{Log} B \quad [1]$$

Let's try it, using the example LaMothe provides in his explanation of the technique. Say you want to multiply 100 times 1,000. The (base 10) log of 100 is 2 (i.e., $10^2 = 100$) and the log of 1,000 is 3 (i.e., $10^3 = 1,000$). So, multiplying 100 times 1,000 is almost as simple as adding 2 and 3. I say "almost" because you'll presumably want to convert the result back to a regular number. Doing so calls for a reverse "antilog" (exponentiation) function (i.e., converting 10^5 to the final answer, namely 100,000).

Of course, performing all these log and antilog calculations isn't a cake-

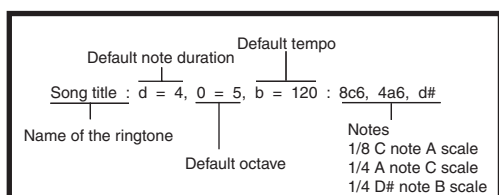


Figure 3—Take advantage of standard RTTTL ringtones and the kit's "RTTTL Jukebox" code to give your embedded application an audio makeover.

Listing 1—Sometimes a relatively small amount of memory can go a long way, especially given the inexorable decline in cost/bit. Inside the Propeller, 4 KB are used for a sine lookup table, eliminating slow trig calculations in favor of a fast table lookup.

```

' Get sine/cosine
'
' quadrant:      1          2          3          4
' angle:         $0000..$07FF $0800..$0FFF $1000..$17FF $1800..$1FFF
' table index:   $0000..$07FF $0800..$0001 $0000..$07FF $0800..$0001
' mirror:       +offset   -offset   +offset   -offset
' flip:         +sample   +sample   -sample   -sample
'
' on entry: sin[12..0] holds angle (0 to just under 360 degrees)
' on exit:  sin holds signed value ranging from $0000FFFF (1) to
           $FFFF0001 (-1)
'
getcos add    sin,sin_90      ' for cosine, add 90 degrees
getsin test   sin,sin_90     wc ' get quadrant 2|4 into c
        test   sin,sin_180   wz ' get quadrant 3|4 into nz
        negc   sin,sin        ' if quadrant 2|4, negate offset
        or    sin,sin_table   ' or in sin table address >> 1
        shl   sin,#1         ' shift left to get final word address
        rdword sin,sin        ' read word sample from $E000 to $F000
        negnz sin,sin        ' if quadrant 3|4, negate sample
getsin_ret
getcos_ret
'
'
sin_90 long $0800
sin_180 long $1000
sin_table long $E000 >> 1 'sine table base shifted right
'
sin long 0

```

walk, but Propeller has a trick up its sleeve in the form of log and antilog lookup tables built into on-chip ROM. It's true that the exact recipe for cooking up some calculations using the tables is pretty complicated and non-intuitive. Fortunately, LaMothe gives a most excellent and detailed explanation culminating in listings of the Propeller assembly language routines that use the tables to convert (log) and restore (antilog) 32-bit integers. FYI: the pair of routines combined takes about 2 μ s with the Propeller running at 80 MHz. While DSPs have nothing to fear, that's certainly not shabby. Oh yeah, don't forget that's for just one of the on-chip processors. There's nothing at all to prevent running the routines concurrently on all eight processors at the same time, cutting the effective execution time by a factor of eight (i.e., 250 ns).

Similarly, Propeller has a built-in sine lookup table and LaMothe provides a routine that uses it to deliver decently accurate conversions (resolution better than 0.05°) in as little as 500 ns (at 80 MHz) (see Listing 1). Once again, that's effectively 66.6 ns per conversion

(i.e., 500 ns/8) if you run it on all eight processors at once. The scheme delivers the sine of 0° to 90° with 16-bit accuracy and you can easily determine the result for other quadrants (e.g., 90–180, 180–270, and 270–360) by mirroring and flipping the lookup table in software. As well, trigonometric identities can be used for related calculations (e.g., cosine = sine (x + 90)).

Next, LaMothe turns his attention to Spin, the unique interpreted language Parallax has devised for the Propeller chip. Like the earlier discussion of the chip's assembly language, this section is very helpful because it provides detailed explanations and example code snippets for each of the major language features.

He even dips a toe into the murky waters of multiprocessing and parallel programming, with helpful direction to references as well as suitable precautionary disclaimers.^[4] A typical author might just say the subject is complicated and could prove difficult to understand. LaMothe puts it a bit more bluntly when he says that you may "be crushed by enormous gravitational anomalies."

Finally, it's all pulled together with a



Photo 4—Keep your eyes on the road and your hands upon the wheel. “X-Racer” (by Hydra Demo Coders Team members Jay T. Cook and Remi Veilleux) spins up the Propeller with fancy 3-D graphics techniques, such as ray casting and sprite scaling. Not bad for a chip that does everything (graphics, video, audio, gamepad, and the game itself) with just 64 KB of memory.

series of programming examples that put the pedal to the metal with key Propeller features, such as the built-in video shifters and high-speed timers. These examples demonstrate the use of various libraries crafted by Parallax (e.g., TV, VGA, and graphics) as well as LaMothe’s own routines for sound generation and gamepad interfacing. The examples progress from the simplest plot a pixel routine and head onward and upward from there with line, triangle, polygon, and text examples.

THEN PLAY ON

The final part of the book, “Game Programming on the Hydra,” is understandably the most gaming-centric. Those of you who, like me, aren’t really into games might anticipate this section wouldn’t be useful or interesting. And it’s true some of the more arcane aspects (e.g., sprite editors) have little to do with generic embedded computing.

However, even if it is not especially relevant it is all darn interesting! For instance, in the chapter on “Sound Design for Games,” I learned about RTTTL, which stands for Ring Tones Text Transfer Language. Nokia invented this simple and efficient ASCII format for defining ringtones and one of the Hydra demos included with the kit is an RTTTL jukebox (see Figure 3). This got me thinking that embedded systems in general could be served by a trend towards higher fidelity and even user-programmable sound effects. For sure, I’d like a “ringtone” for our household clothes dryer instead of the harsh industrial-grade klaxon that could wake the

dead. The days of a plain (not to mention obnoxious) beeper or buzzer may, and should, be numbered.

I was further captivated by the chapter titled “AI, Physics Modeling and Collision Detection.” Sounds pretty dry, but thanks to his fun and friendly writing style (the chapter is subtitled “A Crash Course”), LaMothe manages to cover a lot of territory from matrix processing, to state machines, to genetic algorithms somehow keeping it all connected and coherent. It’s almost as though you learn a lot without feeling like you’re “working” hard enough.

At least read the chapter “Introduction to Game Development.” Once again, hidden behind the gaming facade is some welcome and retro-refreshing insight on the process of creating and coding complicated applications. Some may poo-poo all this as talk about “toys,” but as LaMothe points out, does anybody ever remember getting a “blue screen of death” or “please wait” on a video game? Fact of the matter is that game designers deserve respect for their ability to deliver rock-solid real-time code that wrings every MIPS to be had, and then some, from the hardware.

Remember, all 800-plus pages of the book are backed by many megs of example code, tools, and documentation on the accompanying CD. And yes, I did somehow manage to find time to play—oops, I mean “work”—with all the cool games, demos, and utilities in the book’s culminating “Hydra Demo Showcase” (see Photo 4).

GAME OVER

The idea of combining multiple processors to gang up on an application isn’t new. But, what is new is that the time has come to actually do it. That won’t be easy. Of course, we’ll rely on the tools (compilers, OSs, etc.) to try to find and exploit parallelism, but the tools can only do so much “automagically.”

Thus, expect more along the lines of the aforementioned conference tutorials designed to teach old programmers new parallel tricks. I imagine it won’t be long before a “Parallel Processing for Dummies” book hits the shelves

(subtitle: “A Step-By-Step Guide to Non-Step-By-Step Thinking”).

My own feeling is that reeducating the programming masses may not only be difficult, but perhaps actually unnecessary. Maybe a better approach is to rely on wizards that “get it” to do the heavy lifting (i.e., deal with the hard-core parallel stuff) and let civilian “step-by-step” programmers take advantage of their work.

And so it is with Propeller. Even though it’s a mini-me multicore, getting the most out of Propeller calls for getting way under the hood with clever programming. And that’s just what folks like André LaMothe, the Hydra Demo Coder Team, and everyone contributing to the “Object” Repository on the Parallax web site are doing.

The best way to start solving the parallel processing “problem”? Take advantage of the work done by those already solving it. ☒

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] The 34th International Symposium on Computer Architecture, San Diego, CA, June 9 to 13, 2007, www.cse.ucsd.edu/isca2007.
- [2] ACM SIGPLAN 2007 Conference: Programming, Language, Design, and Implementation, San Diego, CA, June 10 to 13, 2007, <http://ties.ucsd.edu/PLDI/index.shtml>.
- [3] Brookhaven National Laboratory, “The First Video Game,” www.bnl.gov/bnlweb/history/higinbotham.asp.
- [4] B. Barney, “Introduction to Parallel Computing,” Lawrence Livermore National Laboratory, 2006, www.llnl.gov/computing/tutorials/parallel_comp.

SOURCE


32360 Hydra game development kit
Parallax, Inc.
www.parallax.com

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. Send your disk and digital submission sheet to: IDEA BOX, Circuit Cellar, 4 Park Street, Vernon, CT 06066 or e-mail adcop@circuitcellar.com. For more information call Shannon Barraclough at (860) 875-2199.

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



phyCORE[®] OEMable Single Board Computers

XScale: PXA270, PXA255

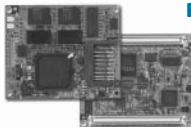
ARM: LPC3180 (ARM9); LPC22xx, LPC229x, AT91 (ARM7)

PowerPC: MPC5554, MPC5200B, MPC565, MPC555

ColdFire: MCF5485 **x86:** Elan SC520

CAN
C166/XC16x/ST10/8051

Blackfin: BF537



Faster-to-Market: Save time by integrating a PHYTEC Single Board Computer Module into your target circuitry.

Make-or-Buy: Why make your own when you can buy PHYTEC off-shelf solutions, cost-effective to 1000s units/year?

Integrated Support Services: Let PHYTEC assist you in the design of your end product: from tools and RTOSes to production. Our hardware is bundled with leading compilers (Keil, IAR, CodeWarrior), RTOSes (WinCE, Linux) and debuggers.

Immediate Support: Talk to PHYTEC technical staff with every call. No waiting for answers.

Your OEM solution: With 20 years design, production, and integration experience, PHYTEC is your OEM partner.

PHYTEC America, LLC ■ 203 Parfitt Way, SW, G100 ■ Bainbridge Island, WA 98110 USA
www.phytec.com ■ (800) 278-9913 ■ www.phytec.com

USB

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

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

Absolutely NO driver software development required!

www.dlpdesign.com



NEW

RS232 to TCP/IP

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



Free 30 day evals at www.taltech.com

Low Cost PGM/DEMO Kits



USB-DONGLE

The **USB-DONGLE** and **Derivative Boards** allows quick and easy ICP/ISP programming of many popular microcontroller families such as the LPC9xx, ARM7 LPC2xxx and 89V52X2 from NXP Semiconductors. The kit includes a Virtual COM Port Driver that allows hex files to be downloaded and programmed using Flash Magic or other common utilities. The unit also provides a low cost platform for testing or prototyping of simple microcontroller based designs (blink an LED, measure ICC, test a Timer or PCA output). Low cost **Derivative Boards** are available for many different microcontrollers from NXP. Please consult our website for details.

Derivative Boards



LPC9103 HVSON10



ARM7 LPC2103 LQFP48

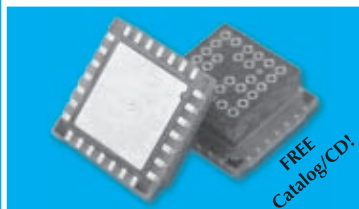
XA Development Kits, I2C, MDIO & SPI tools in stock, visit www.teamfdi.com for details.



www.teamfdi.com
VISA/MC/Amex

Future Designs, Inc.
2702 Triana Blvd
Huntsville, AL 35805
(256) 883-1240
Fax (256) 883-1241

CONNECT TO SMT PADS




FREE Catalog/CD!

- BGA
- QFN (MLF)
- QFP
- SOIC
- PLCC

IC SMT Pads Become Connection Interface with SMT Emulation

Industry's widest selection for SMT Pattern Interconnect Technology allows Test, Prototype, Package Conversion, Emulation

Quick Custom Designs for all package types



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

\$189



Tiny Linux controller

16MB fast SDRAM	10/100 Ethernet
64MHz Coldfire MCU	3 serial ports (RS232)
4.5 MB flash memory	1 CAN port
SD card socket (to 2GB)	LCD/KPD port

Eclipse/CDT dev. env. Free serial debugger

55 I/O pins & capabilities to burn...

www.steroidmicros.com

www.can232.com

Only \$108 €89


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

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

Only \$154 €129

www.canusb.com


Instant LCD



inst.LCD-002 - \$189

- Versatile Programmable Module
 - Serial/PC/USB/Parallel Interface
 - AVR/BASIC Stamp/VB Compatible
 - Onboard Flash Bitmap Memory
 - Downloadable TTF Fonts
- 2.7" or 5.6" TOUCH Color TFT LCD
 - 240x160 or 320x240 resolutions
 - Transflective w/ LED Frontlight (2.7")
 - Transmissive w/ 350 nit backlight (5.6")
 - 512 colors or 65,535 colors

EARTH.LCD.COM We Make **LCDS** Work.™



Loadstar SENSORS

Introduces
 Capacitive Load Sensors with
 True USB connectivity

Integrated signal conditioning
 Digital USB or Analog 0-5 V output
 Accuracies - 0.25% to 0.025% of FS
 Rugged stainless steel construction
 Temperature compensated
 Easy mounting features built in

www.loadstarsensors.com
 650.938.4282 | info@loadstarsensors.com

MYLYDIA, INC.

Layout Gerber,
 Prototype Making
QUICK TURN
 PCB & Turnkey
 @
 The Best Prices

Tel: 800-695-9342
Sales@mylydia.com
WWW.MYLYDIA.COM

Solve complex signal acquisition problems...



- positioning & control
- environmental
- acceleration
- transients
- pressure
- vibration
- sonar
- GPS
- Linux Driver
- Guaranteed in stock
- Customization available
- 16-bit analog inputs and outputs
- Million sample FIFO eliminates interrupts
- Wide analog input and output ranges
- -40°C to +85°C Standard

www.stx104.com
Apex Embedded Systems
sales@stx104.com • 608-256-0767 x24



Hands-on Training

for Rabbit Developers

Learn More at
Rabbit-U.com

07114

CHINA PCB SUPPLIER

DIRECT SALE FROM CHINA
PROTOTYPE TO PRODUCTION

instant online quote
 shopping cart ordering system
 China competitive prices
 free Electrically test



web: <http://www.pcbcart.com>
 email: sales@pcboard.com
 Tel: +86-571-87012818
 Addr: No. 2 Huxiang Road,
 Hangzhou, P.R. China

WWW.PCBCART.COM

Link Instruments

Digital Oscilloscope

500 MSa/s

- 2 Channel Digital Oscilloscope
- 500 MSa/s max single shot rate
- 250 MSa/S (Dual channel) 512 Kpts
- 500 MSa/S (Single channel) 1 Mpts
- Advanced Triggering
- Portable and Battery powered
- Only 9 oz and 7" x 3.5" x 1.5"
- FFT Spectrum Analyzer
- USB 2.0

\$950



www.LinkInstruments.com 973-808-8990

I²C/SMBus



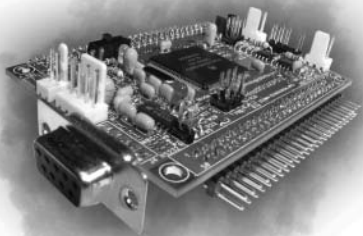
- Bus Monitors
- Protocol Analyzers
- Host Adapters
- Multiplexers
- Battery Applications
- Software Tools

MCC
 Micro Computer Control

I²C is a trademark of Philips Corporation

www.mcc-us.com

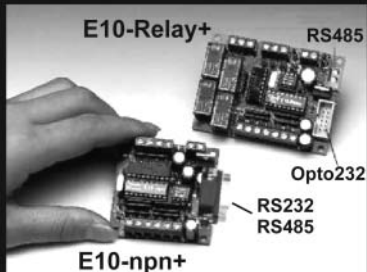
Start designing with
Freescale's
advanced S12X family!



Toll-free: 1-877-963-8996
www.technologicalarts.com

The \$69 PLC

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



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



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

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

USB Host Stack

- USB 2.0
- EHCI, OHCI, UHCI, ISP116x, ISP1362, ISP176x, ARM, ColdFire
- Hub, Mass Storage, Modem, Mouse, Keyboard, Printer, Serial Converter
- Low Cost, Royalty-Free
- Full Source Code
- Standalone or RTOS
- Device and OTG Available



www.smxrtos.com/usb

Modules + uClinux™



OEM Pricing
Starts at \$85^{ea}
(qty 500 pc)

- 32-Bit Core Processors
- Various Processors Supported
- Extended Temp Ratings
- RoHS Compliant
- up to 32 Mbyte SDRAM
- up to 16 Mbyte Flash
- MAC/PHY/Transformers
- RS232 Transceivers
- I/O, SPI, UARTs
- Feature Rich Bootloader
- Complete Kits and BSPs
- Full OS and GNU Tools

home
automation

industrial
controls

machine
interface

networked
sensor

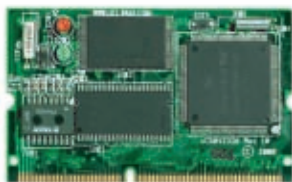
data
acquisition

Arcturus

empower embedded.

tel +1.866.733.8647

www.ArcturusNetworks.com



DUAL ETHERNET + USB
- MCF5272 - ColdFire

ETHERNET + DSP
- TMS320VC5471 - ARM7

ETHERNET + CAN
- MCF5282 - ColdFire

ETHERNET + LCD
- 68VZ328 - 68K

building
systems

signal
processing

media
end-points

network
appliances

protocol
conversion

ColdFire® - ARM® - 68K



Serving the Electronic industry for over 25 years with New and Surplus Electronic Parts, Kits, LCD's, LED's, Audio Parts, Meters, Switches, Power Supplies, Fans, Relays, Tools, etc.... Come On Down and Visit Our Internet Store at www.bgmicro.com, or you can call our toll free number for a live person.

We'll be Expecting you!

WWW.BGMICRO.COM

1-800-276-2206

PC/104 PERIPHERALS

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

ADIO-104 *This Module Does it all!*

- 16 Analog Inputs
- 8 Analog Outputs
- 24 Digital I/O lines
- 5V Power
- Pulse Accumulator
- Open-Drain Outputs

DIO96-104

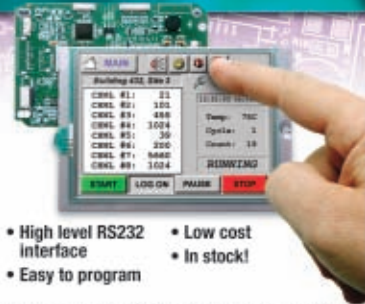
- 96 Bi-Directional Digital Channels
- Input, Output and Strobed I/O functions
- Uses Familiar 82C55A Chips

XIO-R08

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

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

Add a color touch interface to your embedded product!



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

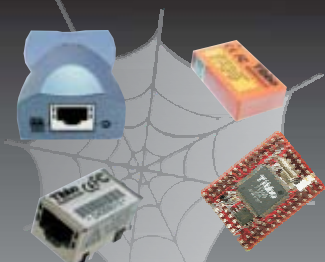
Add color graphics to any 8/16 bit embedded system. Easy, fast and flexible. Up and running in hours!

REACH
TECHNOLOGY INC.

www.reachtech.com • 510-770-1417

842 Boggs Avenue • Fremont, CA 94539

Net Modules



programmable
revolutionary

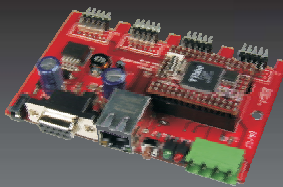
SAVE 5% enter coupon code CCI at checkout

ABACOMdirect.com



tel: +1(416) 236 3858
fax: +1(416) 236 8866

32io SERVER



control & monitor
anything anywhere
over the internet

SAVE 5% enter coupon code CCI at checkout

ABACOMdirect.com



tel: +1(416) 236 3858
fax: +1(416) 236 8866

Wireless I/O

extend your digital I/O's
over 100's of feet up to
tens of miles - in both
directions - with the
16IO-SSRT RF transceiver
modules



SAVE 5% enter coupon code CCI at checkout

ABACOMdirect.com



tel: +1(416) 236 3858
fax: +1(416) 236 8866

250mW FHSS Embedded Radio Module (900 MHz)

www.radiotronics.com

Wi.232FHSS-250-R
from

RADIOTRONIX

True UART to antenna solution

- Frequency Hopping Spread Spectrum (FHSS)
- Features Analog Devices ADF7020 Transceiver IC
- Features Silicon Labs C8051F311 Microprocessor
- +23.5dBm (250mW) output power
- 153.6kb/s maximum RF data rate
- Automatic Gain Control (AGC)
- Both Analog & Digital RSSI
- 4 Variable power steps
- MAC addressing mode
- Small size 1.2" x 1.2"
- FCC certified version available

LONG RANGE!

Development Kits Available!
(RK-Wi.232FHSS-250-FCC-R)

ORDER HERE!

COMPONENTS
Designing in Electronic Components

www.dcomponents.com 1-802-752-4321

586-Generation Industrial Control

586Drive+P300

Ethernet/TCP, 24-bit ADC, DAC, HV I/O, CF, 300+ HV I/O with screw terminals.



- 586 Drive (control board) + P300 (expansion board)
- AMD SC520 processor, program in C/C++
- 4 RS232/485, ADC, DAC, Solenoid Drivers, OPTO
- CompactFlash and FAT16 file system support
- Hardware TCP/IP stack for 100M Base-T Ethernet

\$250 in OEM quantities

50+ Low Cost Controllers with ADC, DAC, solenoid drivers, relays, PC-104, CF, LCD, DSP motion control, 10 UARTs, 300 I/Os. Custom board design. Save time and money.



1724 Picasso Ave., Suite A, Davis, CA 95616 USA

Tel: 530-758-0180 • Fax: 530-758-0181

www.tern.com • sales@tern.com



ValueCAN

The High Value
Tool For
Controller
Area
Network

- USB to CAN
- Simple software analyzer included
- DLL with examples for custom applications
- PC isolated from CAN
- 100% bandwidth at 500Kb

Only
\$295



www.intrepidcs.com

CAN-4-USB FX

USB to CAN Interface
USB 2.0 Hi-Speed 480Mbps!

Other companies may claim USB 2.0 but if it isn't Hi-Speed it is only 12Mbps.

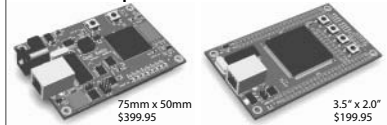
2007 marks our 11th year of selling CAN interfaces in over 30 countries!

\$249
USD.



Zanthic Technologies Inc.
403-878-2202
www.zanthic.com

High-speed USB 2.0 + Xilinx FPGA Complete Software API



XEM3010-1500P:

- 1,500,000-gate FPGA
- Programmable PLL
- Over 110 I/Os
- 32 MB SDRAM
- Configuration PROM
- 0.8-mm expansion
- \$399.95 / Qty 1
- \$349.95 / Qty 10

XEM3001:

- 400,000-gate FPGA
- Programmable PLL
- Over 80 I/Os
- 0.1" expansion headers
- Business-card size
- \$199.95 / Qty 1
- \$174.95 / Qty 10

FrontPanel Software API:

- Windows XP, Mac OS X, Linux • C/C++, Python, Java
- Configure FPGA and communicate with your design
- The easiest way to integrate USB into your product
- Use for image capture, control, test equipment, etc.
- Up to 38 MB/s transfer rate!

5% off with Coupon Code: CKTCLR73*

*Valid for first order only. Void 30-days after issue publication.

Opal Kelly Visit us online at:
www.opalkelly.com

The dsPIC®/PIC24® MCU CCS C Compiler is now available for purchase!

PCD Compiler Built in Functions:

- Standard C Pre-processors
- Math Libraries
- Interrupt Handlers
- Discrete I/O
- Constants in ROM
- 32, 48 and 64-bit Float
- ICSP and In-Circuit Debugging
- Linking to Microchip Libraries
- int1, int8, int16, int32, int48 & int64

Add-on • Upgrade • Pricing • Information
www.ccsinfo.com/dspicc
262.522.6500 x35
PCD IDE \$350



PIC/dsPIC® is a registered trademark of Microchip Technology Inc. in the U.S. and other countries.

FAT 12/16/32 FILE SYSTEM

- DOS/Windows Compatible
- USB Flash Disk
- USB Floppy & Hard Disk
- SD/MMC
- CompactFlash, ATA/IDE
- DiskOnChip
- NAND & NOR Flash
- 10KB RAM / 25KB ROM typical
- Low Cost, No Royalty
- Full Source Code

www.smxrtos.com

Micro Digital Inc
RTOS Innovators

800.366.2491 sales@smxrtos.com

SpectraPLUS 5.0 Audio Spectrum Analysis

Features

Sound Card based I/O
FFT sizes to 1048576pts, 1/96 Octave
Up to 24 bit, 200kHz sampling rates
3-D Surface and Spectrogram
Digital Filtering, Signal Generation
THD, IMD, SNR, Transfer Functions
DDE, Macros, Data Logging,
Vibration Analysis, Acoustic Tools

FREE 30 day trial!

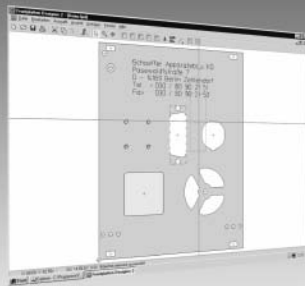
www.spectraplus.com

PHS

Pioneer Hill Software
360 697-3472 voice
pioneer@telebyte.com

Front Panels?

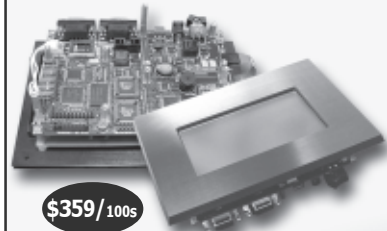
Download the free Front Panel Designer
to design your front panels in minutes



Order your front panels online
and receive them just in time

www.frontpanelexpress.com

QScreen™ - Low Cost Versatile Instrument Controller



\$359/100s

- Touchscreen Operated GUI
- 128x240 CCFL- Backlit Graphic Display
- Programmable in C or Forth
- Hundreds of Screens, Buttons, & Menus
- A/D and Two RS232/485 ports
- 8 Timer-Controlled Digital I/O Lines
- Up to 1 MB Flash, 512KB RAM
- Wide Selection of Plug-in I/O

Mosaic Industries Inc.
tel: 510-790-1255 fax: 510-790-0925
www.mosaic-industries.com

**Serial • Ethernet
Web Server**

**Simple, DTE, DCE
No SW Changes**



**\$99.⁹⁵
Qty 1**

+1 630-245-1445
Naperville, Illinois USA
www.gridconnect.com




ANYONE
Can Now Easily
Hand Solder Surface-
Mount Components!
**Even A 10
Year Old!**

www.schmartboard.com



**Weather
Instruments**
for PCs



www.agelectronica.com



**1-6 Layers Flex PCB
High Quality
1-2 Weeks Delivery
Start at Only \$299**



Custom Membrane Keyboards
PCB/FPC Backed, Built-in LED
Start at \$245

EZPCB www.ezpcb.com
sales@ezpcb.com

**Flashlite
186**



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

**\$69
QTY 1**

**\$99
Development
System**

Development kit includes:

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

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

JK microsystems

**ALL
ELECTRONICS**
CORPORATION

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

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

**CUSTOM MEMBRANE
KEYBOARDS / SWITCHES**



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

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

PRINTED CIRCUIT BOARDS

**QUALITY PRODUCT
FAST DELIVERY
COMPETITIVE PRICING**



- Aluminum Backed PCB
- Single & Double sided
- SMOBC/RoHS
- LPI mask
- Through hole or SMT
- Nickel & Gold Plating
- Routing or scoring
- Electrical Testing
- Artwork or CAD data
- Fast Quotes
- Flex Circuits

SPECIAL OFFER:
10 pcs (3days) 1 or 2 layers \$249
10 pcs (5days) 4 layers \$695
(up to 30sq. in. ea.) includes tooling, artwork, L.P.I. mask & legend

PULSAR, INC

9901 W. Pacific Ave. Franklin Park, IL 60131 • Phone 847.233.0012
Fax 847.233.0013 • www.pulsar-inc.com • sales@pulsar-inc.com



Full Speed It writes your USB Code!

NEW! HIDmaker FS for Full Speed FLASH PIC18F4550

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

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

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



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

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

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

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

HIDmaker FS Combo: Only \$599.95

DOWNLOAD the HIDmaker FS Test Drive today!

www.TraceSystemsInc.com

301-262-0300

WIRELESS RS-232

WCSC (Willies Computer Software Co)



- Extremely easy to install and use
- Distances up to 100 meters (329 feet)
- Bluetooth Serial Port Profile

- Communicates with Bluetooth enabled PDAs, Laptops, Smartphones, & Computers
- No programming or special software needed

Other Products

- Professional serial communication libraries & development tools.
- PCI, PCMCIA, USB, Universal PCI, & ISA multiport RS232, Rs422, & Rs485 cards, Bluetooth/USB Dongles

<http://CircuitCellar.wcscnet.com>
sales@wcscnet.com (281)360-4232

TStik!™

Rugged TINI Java™ Module with 10/100 BaseT



TStik is a ruggedized TINI400 chipset in the familiar SIMM72 form factor. Upgrade most DSTINI1 (TINI390) systems or use our new TILT socket boards (TILT Pro is shown above).

TStik is about \$100, and sockets start at under \$60.

SYSTRONIX®

full details at www.TStik.com

STEPPER SYSTEM BRICKS!



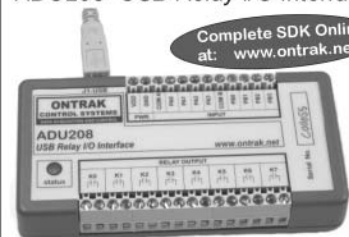
Looking for a compact, rugged OEM drive in the 42 V / 3.5 A per phase range? Our **CY-41C** and **CY-42C** are built for extra reliability. \$96 / \$99 each in 100 quantity.

CyberPak
COMPANY, INC.

800-328-3938 • www.cyberpakco.com

USB Data Acquisition

ADU208 - USB Relay I/O Interface



Complete SDK Online at: www.ontrak.net

FEATURES

- 8, 5-AMP relay outputs
- 8, ISOLATED digital inputs
- Port Powered, Aux 5VDC output \$189.00 QTY 1

Other Models:

- ADU200- 4 Channel Version with RS232 \$139.00
- ADR218- Solid-State Version 8 Channel \$225.00
- ADU100- 3 CH, 16-Bit ISOLATED Analog Inputs, PGA, 4 digital I/O, RS232 and 5 AMP Relay Output \$199.00

ONTRAK CONTROL SYSTEMS INC.

PH: (705) 671-2652 Fax: (705) 671-6127

www.ontrak.net

Order online at:
www.melabs.com

microEngineering Labs, Inc.

Development Tools for PIC® Microcontrollers

Phone: (719) 520-5323

Fax: (719) 520-1867

Box 60039

Colorado Springs, CO 80960

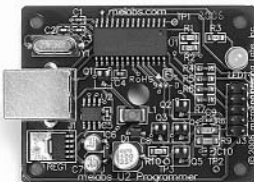
USB Programmer for PIC® MCUs

\$89.95
(as shown)

RoHS Compliant

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

Includes Windows 98, Me, NT, 2K, and XP Software



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



EPIC™

Parallel Port Programmer starting at \$59.95

Serial Port Programmer starting at \$79.95

LAB-X Experimenter Boards



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

Pricing from \$79.95 to \$349.95

PICPROTO™ Prototyping Boards



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

Pricing from \$8.95 to \$19.95

BASIC Compilers for PICmicro®



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

PICBASIC™ Compiler \$99.95

BASIC Stamp 1 Compatible
Supports most 14-bit Core PICs
Built-In Serial Comm Commands

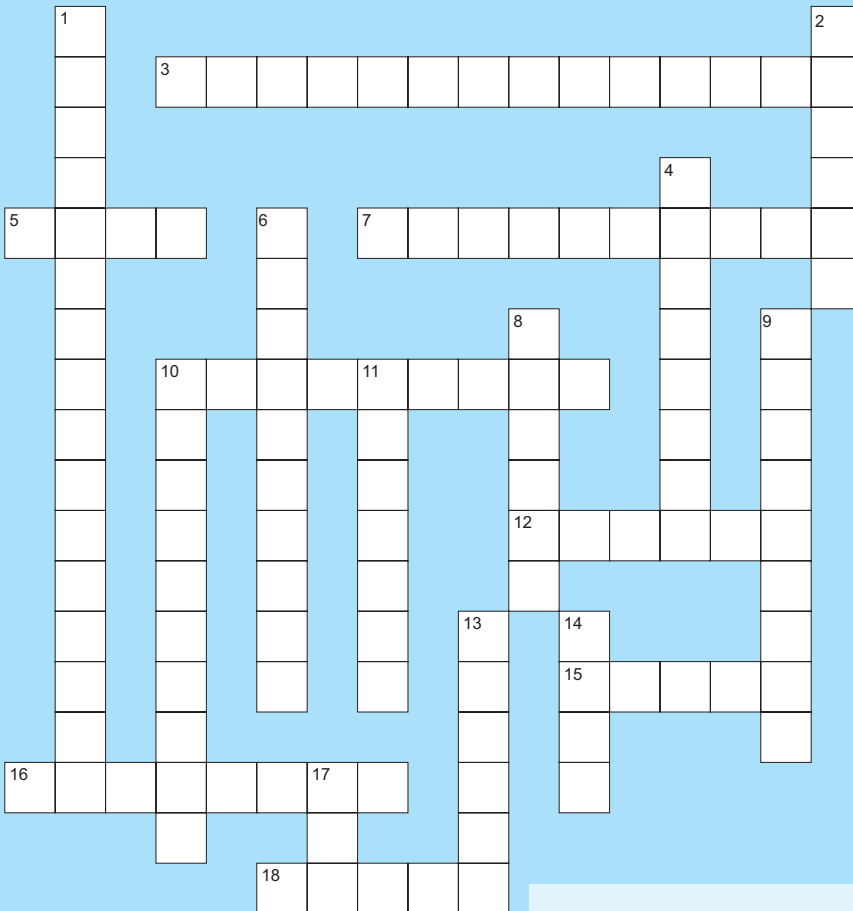
PICBASIC PRO™ Compiler \$249.95

Supports Microchip PIC10, PIC12, PIC14, PIC16, PIC17, and PIC18 microcontrollers
Direct Access to Internal Registers
Supports In-Line Assembly Language
Interrupts in PICBASIC and Assembly
Built-In USB, I2C, RS-232 and More
Source Level Debugging

See our full range of products, including books, accessories, and components at:

www.melabs.com

CROSSWORD



Across

3. A circuit with an output that is proportional to the derivative of the input. There are two types of these circuits, active and passive.
5. A computer socket that links one device with another
7. A semiconductor diode that detects light and generates electrical current
10. A wireless communication specification for PCs and other digital devices
12. A pen-like instrument used with a touch-screen
15. In a battery or direct-current source, it is the negative terminal. In a passive load, it is the positive terminal.
16. A binary format of code unique to Java programs
18. The area where data and objects are stored and held to be processed

Down

1. Electricity generated in response to applied mechanical strain. If the material is not short-circuited, the applied strain induces a voltage across it.
2. A depression on a planet or asteroid that's formed by the impact of a smaller object
4. Relating to stars or constellations
6. The outline of a program without language-specific syntax and real programming statements. Although it cannot be executed, it can enable a programmer to concentrate on the algorithms without having to worry about the details of a specific language.
8. Use this type of testing to determine a system's stability
9. A document that describes the characteristics of a component
10. This program's only job is to load other software to enable an operating system to start
11. A strong flow of particles or fluid
13. An electronic document, often given with software, which contains additional information about troubleshooting installation problems and last-minute changes to the software
14. The American inventor (1938–present) who helped create the TCP. In 1972 he moved to DARPA, and in October of that year, he demonstrated the ARPANET by connecting 40 different computers at the International Computer Communication Conference.
17. A technology that enables you to use virtual prototypes in place of physical prototypes

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

INDEX OF ADVERTISERS

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

Page		Page		Page		Page	
91	AAG Electronica, LLC	82	Elprotronic	87	Loadstar Sensors, Inc.	87	Rabbit Semiconductor
45	ASIX s.r.o.	9	e-PCB	87	MCC (Micro Computer Control)	89	Radiotronix – DComponents
89	Abacom Technologies	20	ExpressPCB	50	Maxstream	89	Reach Technology, Inc.
9	Ad Hoc Electronics	48, 91	ezPCB	88, 90	Micro Digital, Inc.	15, 95	Renesas Technology
72	Affordable Test Gear	86	FDI-Future Designs, Inc.	47	Microchip	56	SoC Conference
72	Aimtec – DComponents	81	FTDI Chip	92	microEngineering Labs, Inc.	91	Schmartboard
91	All Electronics Corp.	90	Front Panel Express, LLC	90	Mosaic Industries, Inc.	89	Scidyne
87	Apex Embedded Systems	87	General Circuits, Inc.	32	Mouser Electronics	16	Sealevel Systems
88	Arcturus Networks	64, 91	Grid Connect	87	Mylydia, Inc.	25	SEgger Microcontroller Systems LLC
7	Atmel	55	HI-TECH Software LLC	5	NKK Switches	29	Silicon Laboratories, Inc.
89	BG Micro	71	IMAGEcraft	C2	NetBurner	92	Systronix
71	BestHomeLEDLighting	86	Intec Automation, Inc.	3	Noritake Co., Inc.	86	TAL Technologies
49	Bitscope Designs	90	Intrepid Control Systems	61	Nurve Networks LLC	C3	Tech Tools
65	CWAV	57	Intronix Test Instruments, Inc.	92	Ontrak Control Systems	40, 41	Technologic Systems
22	CadSoft Computer, Inc.	86	Ironwood Electronics	90	Opal Kelly Inc.	88	Technological Arts
17	Comfile Technology, Inc.	64, 91	JK microsystems, Inc.	66	PCB East Design Conf.	89	Tern, Inc.
90	Custom Computer Services, Inc.	30	Jameco	33	PCB-Pool	8	Tibbo Technology, Inc.
92	CyberPak Company, Inc.	45	Jeffrey Kerr, LLC	C4, 31	Parallax, Inc.	92	Trace Systems, Inc.
1	Cypress Microsystems, Inc.	79	Keil Software	86	Phytec America LLC	88	Triangle Research Int'l, Inc.
86	DLP Design	61	LabJack Corp.	91	Picofab, Inc.	92	WCSC (Willies Computer Software Co.)
71	DSP Workshops	61	Lakeview Research	90	Pioneer Hill Software	21	Wiznet
45	Decade Engineering	87	Lawicel AB	73	Pololu Corp.	13	Wiznet iEthernet Design Contest 2007
48	EMAC, Inc.	23	Lemos International	91	Pulsar, Inc.	90	Zanthic Technologies, Inc.
87	Earth Computer Technologies	2, 87	Link Instruments	34	R4		
28	Efficient Computer Systems	82	Linx Technologies	39, 50	Rabbit Semiconductor		

Preview of October Issue 207 Theme: Signal Processing

Signal Recovery: Restore the Attenuated Portion of an AC-Coupled Signal

Embedded Speech: Add Audio Capabilities to Small Applications

Simple Ethernet

The Monarch Butterfly: Build a Barometric Altimeter

Wireless Thermostat System

Resilience in Embedded Designs (Part 2): RS-485, Voltage Supervisors, Watchdogs, & Outputs

THE DARKER SIDE No Fear with FIR: Put a FIR Filter to Work

ABOVE THE GROUND PLANE Hearing Clearly: Hardware

FROM THE BENCH RoboGrowth

SILICON UPDATE | Sense, Therefore I Am

ATTENTION ADVERTISERS

November Issue 208 Deadlines

Space Close: Sept. 12
Material Close: Sept. 18

Theme:
Analog Techniques

Call Shannon Barraclough
now to reserve your space!

860.875.2199

e-mail: shannon@circuitcellar.com

Educators and Students:
Register Today

to Receive Exciting
Microcontroller Resources!

RENEASAS
UNIVERSITY
PROGRAM



Complete Development Kits



Renesas Starter Kits provide a USB-powered, MCU-based system board with in-circuit debugger/flash memory programmer. They include a CD containing our integrated development environment with toolchain, plus documentation, example firmware and interesting projects.

- ▶ **Free for Educators:** Register at the Renesas University website to receive ten free Starter Kits per semester. In return, we request the submission of material that enriches Renesas University; i.e., code, student projects, technical papers, embedded control designs, etc.
- ▶ **Low cost for Students:** If actively enrolled in an educational institution, a Starter Kit can be purchased at a very low cost after registering at the Renesas University website.

Visit us at ESC Boston!
Booth #1209

Embedded
Systems CONFERENCE
BOSTON

Renesas — the #1 supplier of microcontrollers in the world — is launching Renesas University, an exciting educational program that gives educators a way to teach microcontroller (MCU) technology using a modern architecture and professional-grade tools. It also offers many valuable resources that help students learn about MCUs and how they can be applied in significant embedded system designs.

#1MCU
REACH
FURTHER

Renesas is a worldwide leader in:

- ▶ Microcontrollers
- ▶ Embedded flash microcontrollers
- ▶ MCUs in car navigation systems
- ▶ Power amplifiers for GSM phones
- ▶ LCD controllers for color mobile displays

The Renesas University program nurtures an online community where educators and students come together to share ideas, address technical issues and discuss microcontroller topics. It is characterized by:

Publish ▶ Renesas actively encourages academics and students to publish microcontroller-related papers. We provide assistance in publishing course material and microcontroller related books.

Toolchain ▶ The Renesas integrated development environment with toolchain is the commercial version of our development tools – with full C compiler, assembler, linker, and debugger. It is not a typical capability-reduced “educational” version. The only limitation is a 64KB code size after 60 days of use.

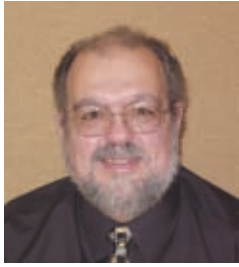
Modern ▶ Renesas microcontrollers utilize a modern architecture designed specifically for C and other high-level languages. Our devices handle the most demanding applications of today and tomorrow.

For more information on Renesas University and how to enroll, please visit www.renesasuniversity.com or email: University@rta.renesas.com



© 2007 Renesas Technology America, Inc. Renesas Technology America, Inc. is a wholly owned subsidiary of Renesas Technology Corp.

Everywhere you imagine. **RENEASAS**



PRIORITY INTERRUPT

by Steve Ciarcia, Founder and Editorial Director

Chronic Subscription Overdose

As I've mentioned before, I've been trying to clean up my act regarding energy consumption rather than just looking at solar PV as a solution to overconsumption. At the very least, I've been looking to see where all this power is being used so that I can economize where applicable. Last month, I commented on my efforts at reducing some of the energy I consume for lighting by adding more CFLs and new AC-powered LEDs.

The irony of specifically looking at my energy costs is that it got me thinking about how I'm spending so much other money living in our communications-saturated world. The reality may be that a few bucks saved on my electric bill are just a drop in the bucket compared to the hidden expenses of all the other communication and information-related products and services I use. Certainly, environmental considerations are high on the list of reasons for installing a home PV electric system, but, if we're comparing apples to apples about schemes for saving money each month, and for about \$60,000 out-of-pocket less, I could save just as much as generating my own electricity if I merely threw out my cell phone or ripped the DIRECTV antenna off the roof. How many more subscription fees do I have that offer equivalent savings?

Yeah, I know that on face value it's a ridiculous comparison, but it got me thinking about subscription expenses in general—and, I have a confession to make. Before explicitly looking into it for this editorial, I doubt I could tell you exactly what they were with any accuracy.

It's sort of like your computer system. There was a time when you bought a computer and installed programs that did exactly what you wanted. The programs you bought were discrete and operated independently with no extra charges. Today, they call it feature enhancement, but the end result is a plethora of interactive and interdependent bloatware, installed in the tiniest handheld to the largest desktop, that all seem to require constant maintenance and endless communication-company subscription payments to actually work as described. In fact, I almost lost my cool when the Verizon dealer told me that after paying \$350 for a new RAZR phone (when they first came out) I'd also have to subscribe to Verizon's \$5/month cell phone e-mail service if I wanted to actually see the pictures anywhere except on the phone! You've got to be kidding.

Absurdities like this hide a greater problem in today's high-tech environment—chronic subscription overdose (CSO). Virtually everything we do these days involves some monthly or yearly subscription fee. Corporate America has discovered that the recurring revenues generated from small subscription “pin pricks” create large total revenues but, more importantly, a contract-secured customer base. It's only because we add these obligations incrementally and each seems to be just another small fee that we don't consider them sinister. Add up all the communications-related services and subscriptions and you will find a classic case of CSO.

I think I lead a rather low-profile existence in our media-overkill world. Everyone seems to have as much text messaging, streaming TV, data and picture exchange, and expensive cell-delivered life fulfillment as battery capacity allows. I just want to make a simple telephone call and don't bother me with the other stuff—obviously no expensive extravagances there.

Still, I'm a very bad boy when it comes to all the rest of my communication purchases and it was shocking to add it all up. While some might find it curious that I have no online subscriptions at all, the 30 print magazine and four newspaper subscriptions still add up to \$1,100 a year. Combine that with four landlines at home, one at the cottage, and two full-time fixed-IP-address DSL lines and you get the picture. In fact, because I have so many uploading devices, I pay \$90/month for high-speed DSL at home (which still seems like a snail's pace compared to what I'd really like to have).

When I look at the communication bills, I admit that my real communication vices are television and satellite subscriptions. Even though I've never purchased a pay-per-view program, my DIRECTV bill is \$145/month. Satellite TV or not, I also have cable TV at another thousand dollars a year. And, while I don't walk around with an MP3 appendage like many people these days, I do seem to have satellite radio in the house and in every car where I am. They would have all been XM, but due to exclusive satellite radio deals with some car brands, I get to pay for both XM and Sirius to the tune of \$676 a year! To add insult to injury, literally speaking, if I want the BMW Assist (similar to OnStar) to call for help when I rear-end somebody while trying to tune Sirius via my iDrive, it's another \$240 a year, and it's only for that car!

We've all become accustomed to these gadgets and services, but it takes adding them up to realize an insidious case of CSO. In my situation all these little pin pricks added up to \$10,098 a year, and this is probably still low compared to many of you. Obviously, any real solution requires a lifestyle change, which in my case, I assure you, will be a hard sell. In the meantime, and more importantly, I guess this exercise points out that no one will ever believe that I'm installing a PV system to save money on my electricity.

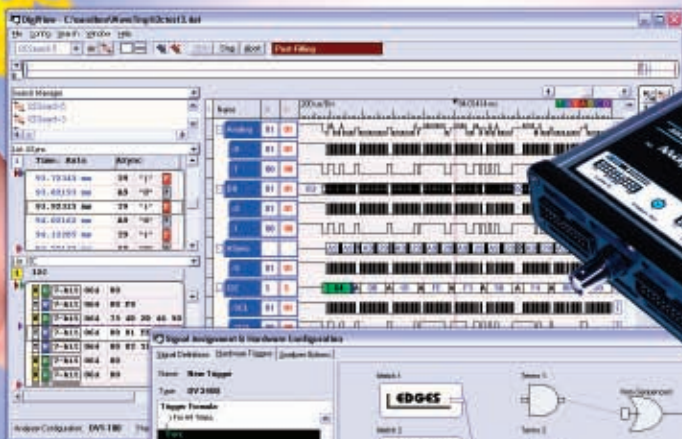
steve.ciarcia@circuitcellar.com

PC Based Logic Analyzers

NEW Model DV3400

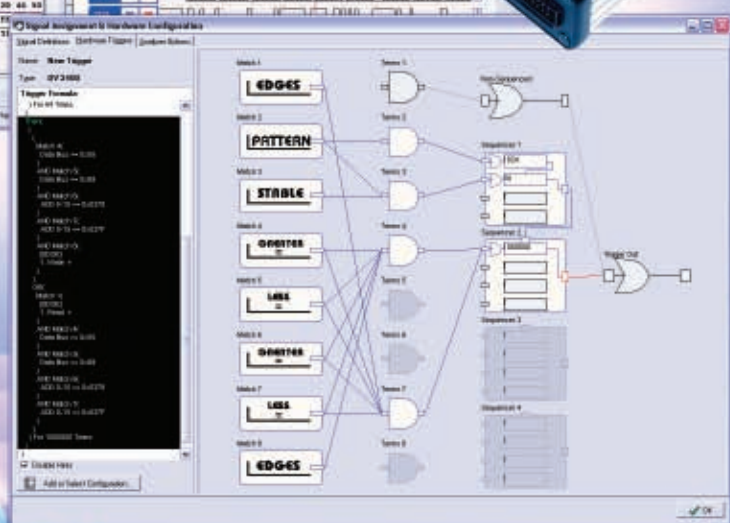
from \$499.00

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



Professional Hardware Capture + Software Analysis

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



Very flexible, "easy to configure" Advanced triggering.

PICmicro[®] MCU Programmer

Multi-Function, In-Circuit & Gang Operation

only \$199.00



See our Memory Emulators

from \$179.00



TechTools

www.tech-tools.com

(972) 272-9392 • sales@tech-tools.com



Why the Propeller Works

by Chip Gracey, creator of the Propeller chip and President of Parallax Inc.

I am the person who designed, debugged, tuned, and tested the Propeller chip. This project took eight years of my time, plus two years of a layout engineer's time. An excruciating amount of attention went into every aspect of the Propeller's implementation and testing, and I allowed no compromises.

The Propeller was an entirely "full-custom" effort. Every polygon of the Propeller's mask artwork was made here at Parallax. We designed our own logic, RAMs, ROMs, PLLs, band-gap references, oscillators, and even ESD-hardened I/O pads. All these structures were first fabricated on test chips and then thoroughly evaluated, often resulting in design changes. This yielded an ideal set of known-good blocks, which could be confidently applied to the overall design. Then, the whole chip was fabricated and subsequently tested at many levels. This allowed us to fix any problems resulting from integration and to fine-tune the clocking systems that are key to the Propeller's low power consumption. The final chip, which is the only version we've ever sold, is the third iteration of this whole-chip process and has no known problems.

The Propeller was given the kind of thorough design treatment that almost no other chips receive today. It used to be that every chip was full-custom, and all of its transistors and wiring were designed by hand, for the point of application. As semiconductor technology shrunk, though, the prevailing design methodology shifted away from this kind of specialization, toward generalization and abstraction, so that designs of greater complexity could be practically realized. The modern design methodology centers around hardware description languages, IP block reuse, and the automated placement and interconnection of potentially billions of gates. The end silicon result is invariably an incomprehensible rat's nest of wiring, standard cells, and IP blocks, usually none of which were designed by the engineers applying them. This methodology is certainly a boon for very complex designs, but it has become the standard approach for designing almost any chip containing logic today. For chips, the old method means small dies, high speed, low power, and exacting performance, whereas the modern method tends to generate bigger dies, lower speed, more heat, and sometimes bugs from IP which you have no control over. I'm sure you get the idea.

Through an exceptional effort, we made the Propeller as electrically robust and efficient as we could. Yet, the core quality of the Propeller really resides in its architecture. The architecture is what took the first six years of development to iron out, and the architecture is what engages people. All the effort that went into the silicon implementation was to ensure that this core quality was ideally housed.

The story of the Propeller's architecture would be a book in itself, but to get an idea of the plot, you can visit the Propeller discussion forum (<http://forums.parallax.com>) and witness the excitement of people doing things they never thought possible. They are finding the Propeller to be a great vehicle for invention and discovery, as well as the means to realize complex embedded systems that are not possible with any other chip. A few forum members have even said that the Propeller has drawn them back into software and electronics after long absences.

The Propeller is tough, reliable, and low-power. It's no illusion, and no accident.

We plan on a very long sales life for the Propeller and we have no intention of diluting the concept with many slight variants, for which you'd inevitably be getting end-of-life notices after a few years. This is good news for customers, because they are the ones who are going to be making investments in programming that will, in sum, dwarf the energy that we spent developing the Propeller. We made a platform that is, hopefully, deserving of their coming efforts.

Sincerely,

Chip Gracey
President
Parallax, Inc.



Learn more at www.parallax.com/propeller