

PROJECT: Digital Dip Meter
LOCATION: United States
PAGE: 20

Q&A: EE Design & Research
LOCATION: Canada
PAGE: 41

DESIGN TIPS: Fault-Tree Analysis
LOCATION: Canada
PAGE: 46

INSIGHT: Working with Arduino
LOCATION: United States
PAGE: 49

CIRCUIT CELLAR

THE WORLD'S SOURCE FOR EMBEDDED ELECTRONICS ENGINEERING INFORMATION

FEBRUARY 2013

ISSUE 271

WIRELESS COMMUNICATIONS

QR Coding Explained

Standing Waves 101

MCU-Based 3-D Paint Program

Camera Controller Front-Panel Board

Robust Design with Linux

PLUS

The OSH Revolution

// A Model for the Future
// Why Open Source = Efficiency
// Design Collaboration
// And More



circuitcellar.com

Easy-to-use Serial to Ethernet Solutions

➤ SBL2e 100

2-Port 3.3V TTL serial device support with analog to digital converters & digital I/O
Part No. SBL2E-100IR: \$21.95 Qty. 1K



➤ SBL2e CHIP (80-pin LQFP)

2-Port 3.3V TTL serial device support with analog to digital converters & digital I/O
Part No. SBL2ECHIP-236IR: \$12.50 Qty. 1K



For as low as
\$12⁵⁰

➤ SBL2e X

1-Port RS-232 serial device support
Part No. SBL2EX-100IR: \$79.00 Qty. 1K



➤ SBL2e XA

1-Port RS-232 and 3.3V TTL serial device support with analog to digital converters & digital I/O
Part No. SBL2EXA-100IR: \$79.00 Qty. 1K



Access serial, analog, or digital I/O data over Ethernet!



The goal:

Control, configure, or monitor digital I/O, analog inputs, or a serial device using Ethernet



The method:

Connect serial port, digital I/O, or analog input to a SBL2e device



The result:

Access device from the Internet or a local area network (LAN)

NetBurner SBL2e Devices network-enable serial, analog, or digital I/O devices out of the box - no programming or development is required. The included virtual COM port Windows utility enables you to easily access serial devices located on a remote NetBurner device as if they were plugged into your own computer. The hardware is pre-programmed to convert your serial data to Ethernet, enabling communication with the serial device over a network or the Internet.

Need a custom solution?

NetBurner Development Kits are available to customize any aspect of operation including web pages, data filtering, or custom network applications.

- **Development Kit for SBL2e**
Part No. NNDK-SBL2E-KIT: \$299.00



www.mouser.com

The Newest Products for Your Newest Designs®



ORE.

mouser.com

The widest selection
of the newest products.



Authorized distributor of semiconductors
and electronic components for design engineers.



MOUSER
ELECTRONICS

Got Range?

As with wireless connectivity, when it comes to your engineering skills, range matters. The more you know about a variety of applicable topics, the more you'll profit in your professional and personal engineering-related endeavors. Thus, it makes sense to educate yourself on a continual basis on the widest range of topics you can. It can be a daunting task. But no worries. We're here to help. In this issue, we feature articles on topics as seemingly diverse as wireless technology to embedded programming to open-source development. Let's take a closer look.

Consider starting with Catarina Mota and Marcin Jakubowski's Tech the Future essay, "Open-Source Hardware for the Efficient Economy" (p. 80). They are thoughtful visionaries at the forefront of a global open-source hardware project. You'll find their work exciting and inspirational.

On page 20, Stuart Ball describes the process of designing a digital dip meter. It's a go-to tool for checking a device's resonant frequency, or you can use it as a signal source to tune receivers. Ball used a microcontroller to digitize the dip meter's display.

Interested in 3-D technology? William Meyers and Guo Jie Chin's 3-D Paint project (p. 26) is a complete hardware and software package that uses free space as a canvas and enables you to draw in 3-D by measuring ultrasonic delays. They used a PC and MATLAB to capture movements and return them in real time.

This month we're running the third article in Richard Lord's series, "Digital Camera Controller" (p. 32). He covers the process of building a generic front-panel controller for the Photo-Pal flash-trigger camera controller project.

Turn to page 37 for the fifth article in Bob Japenga's series on concurrency in embedded systems. He covers the portable operating system interface (POSIX), mutex, semaphores, and more.

Check out the interview on page 41 for insight into the interests and work of electrical engineer and graduate student Colin O'Flynn. He describes some of his previous work, as well as his Binary Explorer Board, which he designed in 2012.

In *Circuit Cellar* 270, George Novacek tackled the topic of failure mode and criticality analysis (FMECA). This month he focuses on fault-tree analysis (p. 46).

Arduino is clearly one of the hottest design platforms around. But how can you use it in a professional-level design? Check out Ed Nisley's "Arduino Survival Guide" (p. 49).

Standing waves are notoriously difficult to understand. Fortunately, Robert Lacoste prepared an article on the topic that covers an experimental platform and measurements (p. 54).

This month's article from the archives relates directly to the issue's wireless technology theme. On page 60 is Roy Franz's 2003 article about his WiFi Snifi design, which can locate wireless networks and then display "captured" packet information.

If you like this issue's cover, you'll have to check out Jeff Bachiochi's article on QR coding (p. 68). He provides an excellent analysis of the technology from a pro engineer's point of view.

cj@circuitcellar.com



CIRCUIT CELLAR®

THE WORLD'S SOURCE FOR EMBEDDED ELECTRONICS ENGINEERING INFORMATION

EDITORIAL CALENDAR

ISSUE	THEME
270 January	Embedded Applications
271 February	Wireless Communications
272 March	Robotics
273 April	Embedded Programming
274 May	Measurement & Sensors
275 June	Communications
276 July	Internet & Connectivity
277 August	Embedded Development
278 September	Data Acquisition
279 October	Signal Processing
280 November	Analog Techniques
281 December	Programmable Logic

Analog Techniques: Projects and components dealing with analog signal acquisition and generation (e.g., EMI/RF reduction, high-speed signal integrity, signal conditioning, A/D and D/A converters, and analog programmable logic)

Communications: Projects that deal with computer networking, human-to-human interaction, human-to-computer interaction, and electronic information sharing (e.g., speech recognition, data transmission, Ethernet, USB, I²C, and SPI)

Data Acquisition: Projects, technologies, and algorithms for real-world data gathering and monitoring (e.g., peripheral interfaces, sensors, sensor networks, signal conditioning, ADCs/DACs, data analysis, and post-processing)

Embedded Applications: Projects that feature embedded controllers and MCU-based system design (e.g., automotive applications, test equipment, simulators, consumer electronics, real-time control, and low-power techniques)

Embedded Development: Tools and techniques used to develop new hardware or software (e.g., prototyping and simulation, emulators, development tools, programming languages, HDL, RTOSes, debugging tools, and useful tips)

Embedded Programming: The software used in embedded applications (e.g., programming languages, RTOSes, file systems, protocols, embedded Linux, and algorithms)

Internet & Connectivity: Applications that deal with connectivity and Internet-enabled systems (e.g., networking chips, protocol stacks, device servers, and physical layer interfaces)

Measurement & Sensors: Projects and technologies that deal with sensors, interfaces, and actuators (e.g., one-wire sensors, MEMS sensors, and sensor interface techniques)

Programmable Logic: Projects that utilize FPGAs, PLDs, and other programmable logic chips (e.g., dynamic reconfiguration, memory, and HDLs)

Robotics: Projects about robot systems, devices capable of repeating motion sequences, and MCU-based motor control designs (e.g., mobile robots, motor drives, proximity sensing, power control, navigation, and accelerometers)

Signal Processing: Projects and technology related to the real-time processing of signals (e.g., DSP chips, signal conditioning, ADCs/DACs, filters, and comparisons of RISC, DSP, VLIW, etc.)

Wireless Communications: Technology and methods for going wireless (e.g., radio modems, Wi-Fi/IEEE 802.11x, Bluetooth, ZigBee/IEEE 802.15.4, cellular, infrared/IrDA, and MCU-based wireless security applications)

UPCOMING IN CIRCUIT CELLAR

FEATURES

Energy-Monitoring System, by Dean Boman

Markov Music Box, by Bruce Land

DIY Rotational Inverted Pendulum, by Nelson Epp

COLUMNS

Arduino Survival Guide: Analog I/O, by Ed Nisley

Microcontroller-Based Morse Coding, by Jeff Bachiochi

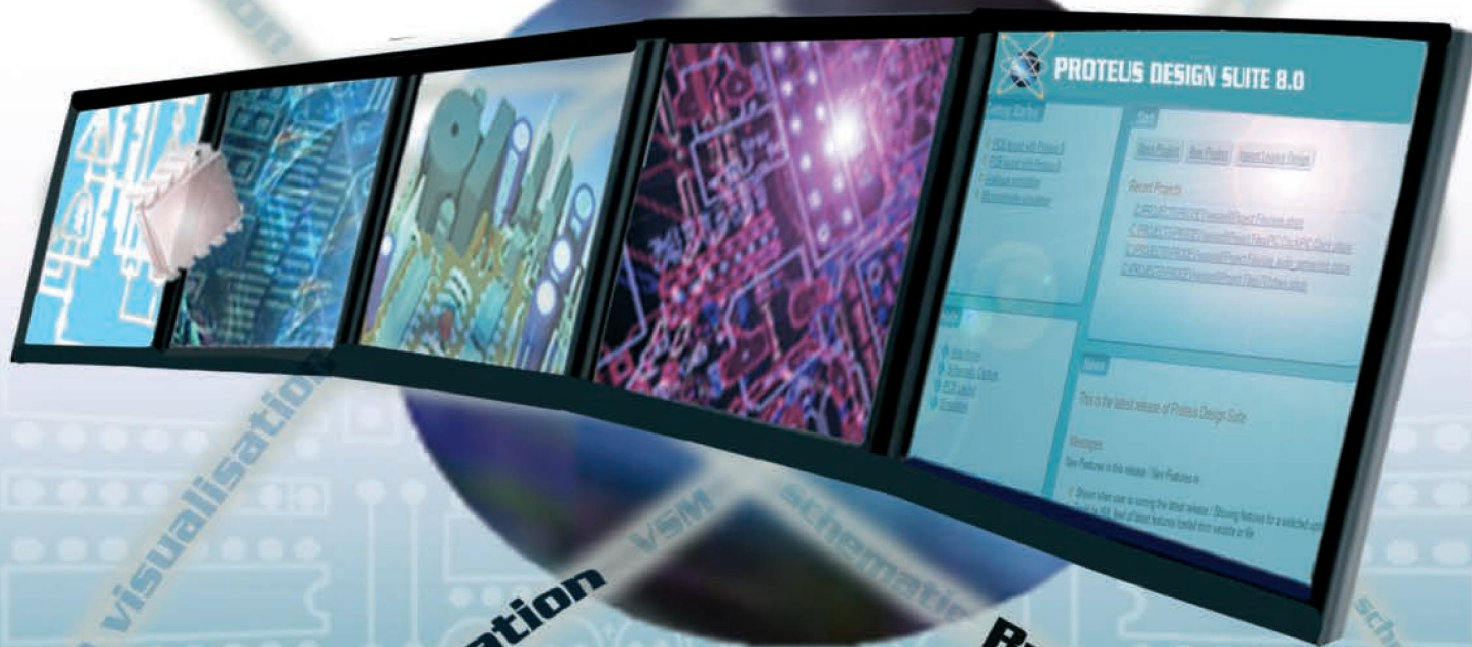
Quality and Reliability in Design, by George Novacek

Chip Biometrics, by Patrick Schaumont

CAD CONNECTED

PCB Layout

Microprocessor Simulation



3D visualisation

3D visualisation

Built-in IDE

schematic

PROTEUS DESIGN SUITE VERSION 8

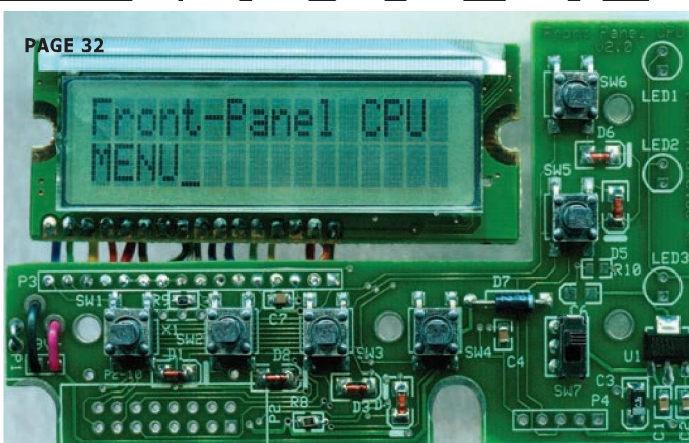
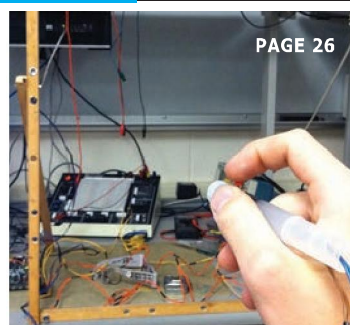
Featuring a brand new application framework, common parts database, live netlist and 3D visualisation, a built in debugging environment and a WYSIWYG Bill of Materials module, Proteus 8 is our most integrated and easy to use design system ever. Other features include:

- Hardware Accelerated Performance.
- Unique Thru-View™ Board Transparency.
- Over 35k Schematic & PCB library parts.
- Integrated Shape Based Auto-router.
- Flexible Design Rule Management.
- Polygonal and Split Power Plane Support.
- Board Autoplacement & Gateswap Optimiser.
- Direct CAD/CAM, ODB++, IDF & PDF Output.
- Integrated 3D Viewer with 3DS and DXF export.
- Mixed Mode SPICE Simulation Engine.
- Co-Simulation of PIC, AVR, 8051 and ARM MCUs.
- Direct Technical Support at no additional cost.

labcenter  www.labcenter.com
Electronics

Visit our website or
phone 866 499-8184
for more details

Labcenter Electronics North America, 411 Queen St. Newmarket, Ont. Canada L3Y 2G9.
Email: info@labcenter-electronics.com Tel 905.898.0665 Fax 905.898.0683



02 EDITOR'S LETTER Got Range?

By C. J. Abate

10 NEW PRODUCTS

17 MEMBER PROFILE

18 CLIENT PROFILE

19 TEST YOUR EQ

20 Build a Digital Dip Meter

By Stuart Ball

26 3-D Paint

A Complete Hardware and Software Package

By William Myers and Guo Jie Chin

32 Digital Camera Controller (Part 3)

Build a Generic Front-Panel Board

By Richard Lord

37 EMBEDDED IN THIN SLICES Concurrency in Embedded Systems (Part 5)

Designing Robust Systems with Linux

By Bob Japenga

41 QUESTIONS & ANSWERS Engineering & "Pure" Research

An Interview with Colin O'Flynn

By Nan Price

46 THE CONSUMMATE ENGINEER Fault-Tree Analysis

By George Novacek

49 ABOVE THE GROUND PLANE Arduino Survival Guide

Digital I/O

By Ed Nisley

54 THE DARKER SIDE Introduction to Standing Waves

By Robert Lacoste

60 FROM THE ARCHIVES The Wi-Fi Snifi

Sniffing In and Out of Wireless Networks

By Roy Franz

(Circuit Cellar 157, August 2003)

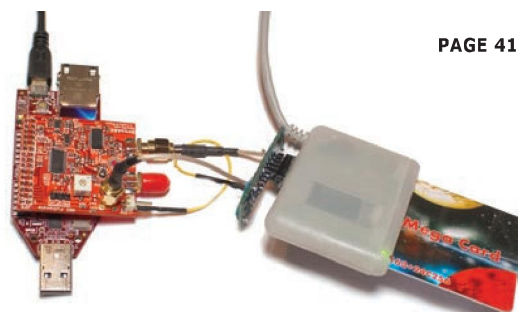
68 FROM THE BENCH QR Coding for Engineers

By Jeff Bachiochi

76 CROSSWORD

80 TECH THE FUTURE Open-Source Hardware for the Efficient Economy

By Catarina Mota and Marcin Jakubowski



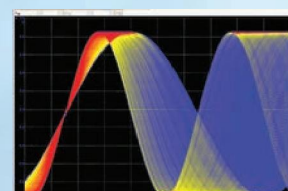
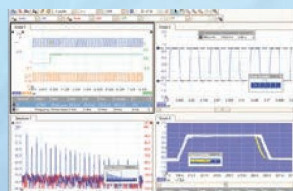
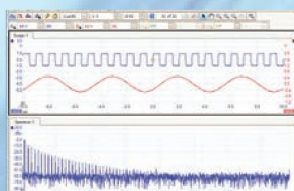
PicoScope®

PC OSCILLOSCOPES



PicoScope 2100 Series	PicoScope 2200 Series	PicoScope 2205 MSO	PicoScope 3200 Series
			
1 Channel 10 to 25 MHz Bandwidth 50 to 100 MS/s Sampling 8 bits Resolution (12 bits enhanced) 8 to 24 kS Buffer memory Price from \$206	2 Channels + AWG 10 to 200 MHz Bandwidth 100 MS/s to 1 GS/s Sampling 8 bits Resolution (12 bits enhanced) 8 to 40 kS Buffer memory Price from \$262	2 Analogue, 16 Digital Channels + AWG 25 MHz Bandwidth 200 MS/s Sampling 8 bits Resolution 48 kS Buffer memory Price from \$658	2 Channels + EXT and AWG 60 to 200 MHz Bandwidth 500 MS/s Sampling 8 bits Resolution (12 bits enhanced) 4 to 128 MS Buffer memory Price from \$658
PicoScope 3400 Series	PicoScope 4000 Series	PicoScope 6400 Series	PicoScope 9200 Series
			
NEW 4 Channels + EXT and AWG 60 to 200 MHz Bandwidth 1 GS/s Sampling 8 bits Resolution (12 bits enhanced) 4 to 128 MS Buffer memory Price from \$988	2 Channels + EXT and AWG, or 4 Channels 20 to 100 MHz Bandwidth 80 to 250 MS/s Sampling 12 bits Resolution (16 bits enhanced) 32 MS Buffer memory Price from \$823	UPDATED 2012 4 + External trigger and AWG or Func. Generator 250 to 500 MHz Bandwidth 5 GS/s Sampling 8 bits Resolution (12 bits enhanced) 128 MS to 1 GS Buffer memory Price from \$3300	2 Channels 12 GHz Bandwidth 5 TS/s (equivalent) Sampling 16 bits Resolution 4 kS Buffer memory Price from \$9892

Software includes: Measurements, Spectrum analyzer, Full SDK, Advanced triggers, Color persistence, Serial decoding (CAN, LIN, RS232, I²C, FlexRay, SPI), Masks, Math channels, all as standard. **FREE UPDATES**



www.picotech.com/PCO498

CALL TOLL FREE: 800 591 2796
















THE TEAM

FOUNDER:	Steve Ciarcia	PROJECT EDITORS:	Ken Davidson, David Tweed
EDITOR-IN-CHIEF:	C. J. Abate	PUBLISHER:	Hugo Van haecke
ASSOCIATE EDITOR:	Nan Price	ASSOCIATE PUBLISHER:	Shannon Barraclough
CONTRIBUTING EDITORS:	Jeff Bachiochi, Bob Japenga, Robert Lacoste, George Martin, Ed Nisley, George Novacek, Patrick Schaumont	ART DIRECTOR:	KC Prescott
		CONTROLLER:	Jeff Yanco
		CUSTOMER SERVICE:	Debbie Lavoie
		ADVERTISING COORDINATOR:	Kim Hopkins

THE NETWORK



OUR INTERNATIONAL TEAMS

	United Kingdom Wisse Hettinga +31 (0)46 4389428 w.hettinga@elektor.com		Spain Eduardo Corral +34 91 101 93 85 e.corral@elektor.es		India Sunil D. Malekar +91 9833168815 ts@elektor.in
	USA Hugo Van haecke +1 860 875 2199 h.vanhaecke@elektor.com		Italy Maurizio del Corso +39 2.66504755 m.delcorso@inware.it		Russia Nataliya Melnikova 8 10 7 (965) 395 33 36 nataliya-m-larionova@yandex.ru
	Germany Ferdinand te Walvaart +49 (0)241 88 909-0 f.tewalvaart@elektor.de		Sweden Wisse Hettinga +31 (0)46 4389428 w.hettinga@elektor.com		Turkey Zeynep Köksal +90 532 277 48 26 zkoks@beti.com.tr
	France Denis Meyer +31 (0)46 4389435 d.meyer@elektor.fr		Brazil João Martins +351214131600 joao.martins@editorialbolina.com		South Africa Johan Dijk +27 78 2330 694 / +31 6 109 31 926 J.Dijk@elektor.com
	Netherlands Harry Baggen +31 (0)46 4389429 h.baggen@elektor.nl		Portugal João Martins +351214131600 joao.martins@editorialbolina.com		China Cees Baay +86 (0)21 6445 2811 CeesBaay@gmail.com

Issue 271 February 2013

ISSN 1528-0608

CIRCUIT CELLAR® (ISSN 1528-0608) is published monthly by Circuit Cellar Incorporated, 111 Founders Plaza, Suite 300, East Hartford, CT 06108. Periodical rates paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate USA and possessions \$50, Canada \$65, Foreign/ROW \$75. All subscription orders payable in U.S. funds only via Visa, MasterCard, international postal money order, or check drawn on U.S. bank.

Cover photography by Chris Rakoczy—www.rakoczyphoto.com

Subscriptions

Circuit Cellar, P.O. Box 462256, Escondido, CA 92046
E-mail: circuitcellar@pcspublink.com
Phone: 800.269.6301, Internet: circuitcellar.com
Address Changes/Problems: circuitcellar@pcspublink.com

Postmaster: Send address changes to Circuit Cellar, P.O. Box 462256, Escondido, CA 92046.

US Advertising

Strategic Media Marketing, Inc.
2 Main Street, Gloucester, MA 01930 USA
Phone: 978.281.7708, Fax: 978.281.7706, E-mail: peter@smmarketing.us
Internet: circuitcellar.com
Advertising rates and terms available on request.

New Products: New Products, Circuit Cellar, 111 Founders Plaza, Suite 300, East Hartford, CT 06108, E-mail: newproducts@circuitcellar.com

MEMBERSHIP COUNTER

We
now have

265646

members
in

83

countries.

Not a member yet?

Sign up at circuitcellar.com

SUPPORTING COMPANIES

All Electronics Corp.	77	ExpressPCB.	59	Microchip Technology, Inc.	13
AP Circuits	17	EzPCB.	47	Microengineering Labs, Inc.	78
APEC	29	Flexipanel, Ltd.	79	Mosaic Industries, Inc.	78
Apox Controls, LLC.	77	FTDI Chip.	C3	Mouser Electronics, Inc.	1
ARM.	25, 48	Grid Connect, Inc.	47	NetBurner	C2
Artila Technologies, Inc.	77	Humandata, Ltd.	63	Pico Technology, Ltd.	5
Beta Layout, Ltd.	31	Imagineering, Inc.	C4	Pololu Corp.	39
BusBoard Prototype Systems.	77	Ironwood Electronics	78	Reach Technology, Inc.	78
Circuit Cellar 25 th Anniversary USB	57	Jeffrey Kerr, LLC.	65	Rigol Technologies	19
Cleverscope.	31	Labcenter Electronics	3	Saelig Co., Inc.	11
Comfile Technology	15	Lakeview Research	65	Technologic Systems	8, 9
CTIA Wireless	23	Logical Devices	79	Tern, Inc.	79
Custom Computer Services	78	Maxbotix, Inc.	77	Triangle Research International, Inc.	79
Elektor	72,73	MCC, Micro Computer Control	77		
EMAC, Inc.	17	Mental Automation	78		

Not a supporting company yet?

Contact Peter Wostrel (peter@smmarketing.us, Phone 978.281.7708, Fax 978.281.7706)
to reserve your own space for the next issue of our member's magazine.

Head Office

Circuit Cellar, Inc. 111 Founders Plaza, Suite 300, East Hartford, CT 06108
Phone: 860.289.0800

Copyright Notice

Entire contents copyright © 2013 by Circuit Cellar, Inc. 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.

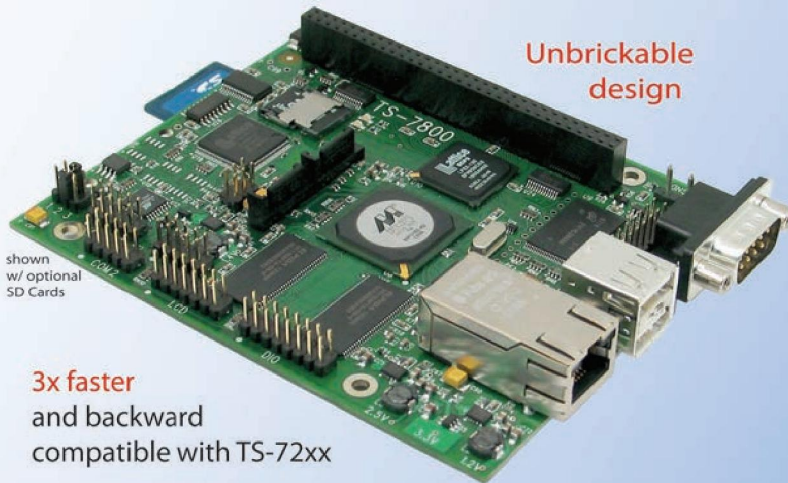
Disclaimer

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.

Embedded Systems

High-End Performance with Embedded Ruggedness



Unbrickable
design

3x faster
and backward
compatible with TS-72xx

shown
w/ optional
SD Cards

TS-7800 500MHz ARM9

- Low power - 4W@5V
- 128MB DDR RAM
- 512MB high-speed (17MB/sec) onboard Flash
- 12K LUT customizable FPGA
- Internal PCI Bus, PC/104 connector
- 2 host USB 2.0 480 Mbps
- Gigabit ethernet
- 10 serial ports
- 5 ADC (10-bit)
- Sleep mode uses 200 microamps
- Boots Linux 2.6 in 0.7 seconds
- Linux 2.6 and Debian by default

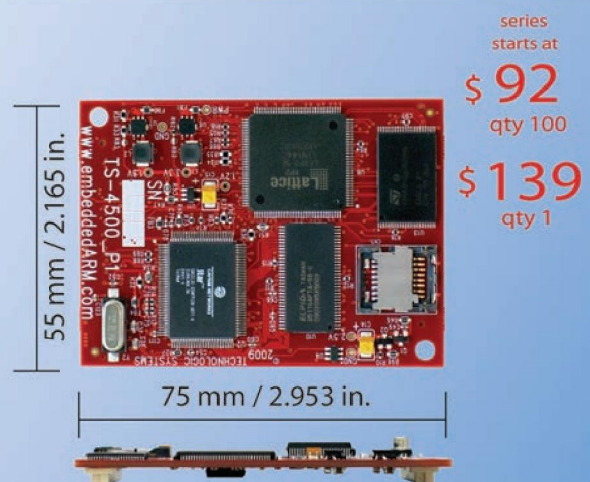
\$229
qty 100

\$269
qty 1

TS-SOCKET Macrocontrollers Jump Start Your Embedded System Design

TS-SOCKET Macrocontrollers are CPU core modules that securely connect to a baseboard using the TS-SOCKET connector standard. COTS baseboards are available or design a baseboard for a custom solution with drastically reduced design time and complexity. Start your embedded system around a TS-SOCKET Macrocontroller to reduce your overall project risk and accelerate time to market. Current TS-SOCKET products include:

- TS-4200: Atmel ARM9 with super low power
- TS-4300: 600MHz ARM9 and 25K LUT FPGA
- TS-4500: Cavium ARM9 at very low cost
- TS-4700: 800MHz Marvell ARM with video
- TS-4800: 800MHz Freescale iMX515 with video
- Several COTS baseboards for evaluation & development



- Dual 100-pin connectors
- Secure connection w/ mounting holes
- Common pin-out interface
- Low profile w/ 6mm spacing

series
starts at
\$ 92
qty 100

\$ 139
qty 1

- Over 25 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

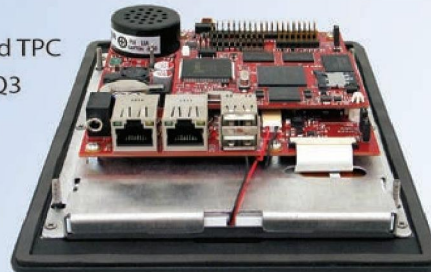
Touch Panel Computers 800MHz with Video Acceleration

- Resistive touchscreen, LED backlit display
- Gasketed construction
- Tough powder coated finish
- Fanless operation from -20°C to +70°C
- 800MHz ARM CPU
- 256MB RAM, 256MB SLC XNAND Drive
- MicroSD slot
- 5K LUT programmable FPGA
- Dual Ethernet, USB ports
- CAN, RS-232 ports, RS-485
- Mono speaker on PCB, stereo audio jack
- SPI, DIO



Fully enclosed TPC
available Q3

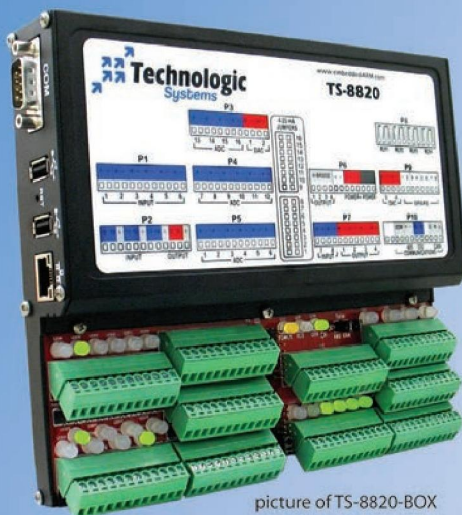
NEW!



series
starts at

\$415
qty 100

\$479
qty 1



picture of TS-8820-BOX

NEW!

series
starts at

\$199
qty 100

\$229
qty 1

Technologic Systems now offers three powerful computers targeting industrial process control. Implement an intelligent automation system at low cost with a minimal number of components.

Industrial Controllers Powerful, Rugged, Affordable

- 250MHz (ARM9) or 800MHz (ARM9 or Cortex-A8) CPU
- Fast startup (under 3 seconds)
- Fanless operation from -20°C to +70°C
- User-programmable opencore FPGA
- Program in Ladder Logic or C
- Debian Linux
- Modbus support
- PoE capable 10/100 Ethernet, USB 2.0 Host Ports
- Industrial screw-down connectors
- Opto-Isolated DIO, Digital Counters, Quadrature
- Up to 46 DIO with PWM
- Opto-Isolation available for RS-232, RS-485 and CAN
- DIN mount option



We use our stuff.

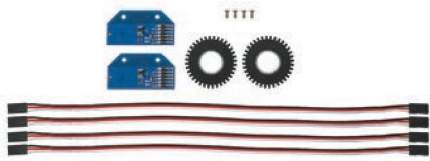
Visit our TS-7800 powered website at

www.embeddedARM.com



ROBOTIC WHEEL ENCODER & AMBIENT LIGHT PROXIMITY SENSOR

The **36-Position Quadrature Encoder Set** provides rotational feedback for robot wheels. The set was specifically designed for Parallax's Motor Mount and Wheel Kit, which is included with the Eddie and MadeUSA robotic platforms. The kit also can be used with custom robots or mechanical systems with 0.5" axles.

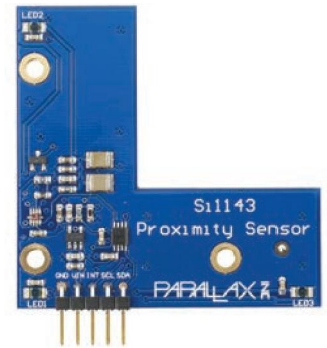


The encoder set provides two out-of-phase outputs from within a single sensor assembly. Its 36-position encoder disks, which resolve to 144 positions with the quadrature sensor output, are incised to grip 0.5" diameter axles. Key features include low power consumption, dual-channel outputs that provide speed and directional information, and a six-pin, single-row header that

accommodates a four- or six-wire interface.

The **Si1143 Proximity Sensor** is well suited for noncontact gesture recognition in microcontroller applications. Gestures in the up, down, left, right, and center directions can be detected by measuring infrared light levels from the three on-board IR LEDs.

The Si143 measures visible and IR ambient light levels, providing a range of operation from darkness to full sunlight. The sensor's easy-to-use interface is compatible with any microcontroller. Its standard 0.1" header pins enable the sensor to conveniently connect to breadboard or through-hole projects.



The 36-Position Quadrature Encoder Set and the Si1143 Proximity Sensor both cost **\$29.99**.

Parallax, Inc.
www.parallax.com

HIGH-RESOLUTION PWM UNIT FOR DIGITAL POWER CONVERSION

The **XMC4400**, **XMC4200**, and **XMC4100** Cortex-based microcontrollers offer a high-resolution PWM unit. Devices in the XMC4000 microcontroller family use ARM Cortex M4 processors.

With a 150-ps PWM resolution, the XMC4400, XMC4200, and XMC4100 microcontrollers are well suited for digital power conversion in inverters, switching and uninterruptible power supplies (UPS), and other applications including I/O automation units, user interfaces (HMI), and logging and control systems.

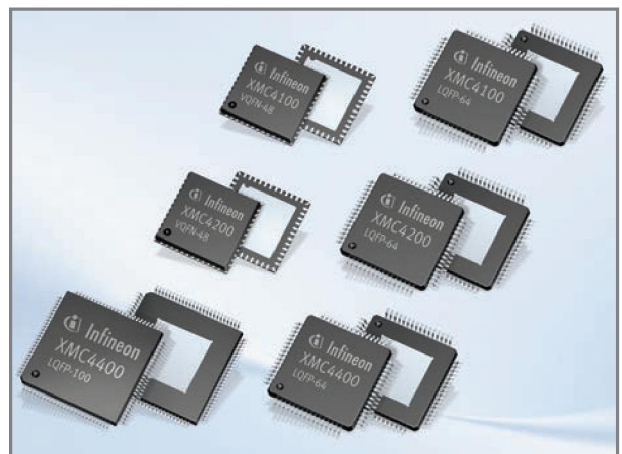
Like Infineon's XMC4500 microcontrollers, the XMC4400, XMC4200, and XMC4100 series offer powerful application-optimized peripherals, fast and robust embedded flash technology, an extended -40°C-to-125°C temperature range, and tools for automatic code generation.

The XMC4000 family includes four series: XMC4500, XMC4400, XMC4200, and XMC4100. The microcontroller families differ mainly in core frequency, memory capacity, peripheral functions, and number of I/Os. The XMC4400, XMC4200, and XMC4100 series have a powerful CPU subsystem with 120 MHz or 80 MHz, DSP functionality, a floating-point unit, and fast flash memory (512, 256 or 128 KB). They feature a 22-ns read time and error-correction code and SRAMs up to 80 KB. The microcontrollers' peripherals include high-speed 12-bit ADCs and DACs and integrated delta-sigma demodulator modules (XMC4400). Communication is provided by Ethernet MAC (XMC4400), USB 2.0, CAN interfaces, and serial communication channels, which can be individually software-configured as UART, SPI, Quad-SPI, I²S, or I²C. The microcontrollers also provide a touch interface and an LED matrix display.

The XMC4400, XMC4200, and XMC4100 are supported with the DAVE 3 integrated development platform, which enables convenient, fast, and application-orientated software development. Third-party tools can be used to extend the Eclipse-based environment with free GNU compiler and debugger. DAVE 3 also supports automatic code generation based on predefined software components (i.e., the "DAVE Apps"). The DAVE Apps are configured in a user-friendly way via the graphical user interface. DAVE 3 ensures industrial application developers can use the XMC4000 microcontrollers' functionality with little programming effort. The generated code can be compiled and debugged directly in DAVE 3 or imported into third-party tools for further processing (currently Altium, ARM, Atollic, IAR Systems, and Rowley).

Contact Infineon for pricing.

Infineon Technologies
www.infineon.com



NEW PRODUCTS

DIGITAL POTENTIOMETERS WITH HIGH BANDWIDTH & LOW RESISTANCE TOLERANCE

The **AD514x** and **AD512x** series of nonvolatile single-, dual-, and quad-channel digital potentiometers (digiPOTs) feature a $\pm 1\%$ resistance tolerance to improve component matching in industrial and communication control systems. The 11 digiPOTs in the AD514x and AD512x series achieve a high 3-MHz bandwidth, which enables fast system response time.

The nonvolatile digiPOT series meet a range of system-level requirements in 256- or 128-TAP, SPI or I²C interfaces, leaded and leadless packaging, all of which feature 4-kV ESD protection. The devices offer a low temperature coefficient performance over a -40°C -to- 125°C temperature range.

The AD514x and AD512x digiPOTs are available in a 3-mm \times 3-mm LFCSP package option for board savings. Contact Analog Devices for pricing.

Analog Devices, Inc.
www.analog.com

Non-Volatile digiPOTs for Industrial and Communications Control Systems

11 Generics, 34 Products:

- ◆ Single, Dual or Quad
- ◆ 128 or 256 Tap ◆ SPI or I²C Interface

REAL-TIME CLOCK FOR AUDIO AND CONTROL DEVICES

The **Barix Real-Time Clock (RTC)** accessory helps ensure audio and control devices continue operating uninterrupted during network failures. The devices help keep mission-critical operations for broadcast radio, streaming media, building automation, and other applications on time.

The self-sustaining reference clock plugs into any device with an RS-232 serial port, including Barix IP audio and control products. The Barix RTC maintains time, even when unpowered, for years. This enables the RTC to provide time information immediately after a device startup, even without a network-based time reference.

The Barix RTC enables devices to work offline without network connection, playing out audio messages and time-sensitive content on time. Similarly, broadcasters streaming syndicated programs with local network IDs, jingles, ads, and promotions can trigger scheduled events without affecting their on-air content.

The RTC can be used by IP control devices to gain independence from network time references, continuing to switch lights and boilers on and off if the network fails. This ensures energy-saving techniques for schools, businesses, and other facilities continue without disruption.

Contact Barix for pricing.

Barix
www.barix.com



BEST SCOPE SELECTION & lowest prices!

25MHz Scope

Remarkable low cost 25MHz 2 channel plus trigger USB bench scope with 8 inch full color LCD display. Spectrum analysis and autoscale functions.

PDS5022S \$279



iPhone Scope

5MHz mixed signal oscilloscope adapter for the iPhone, iPad and iPod Touch! A FREE iMSO-104 app is available for download from Apple App Store.

iMSO-104 \$297.99

100MHz Scope

High-end 100MHz 16Sa/s 2-channel benchscope with 1MSa memory and USB memory port. Includes a FREE scope carry case! New low price!

DS1102E \$399



60MHz Scope

60MHz 2 channel digital scope with a 500 MSa/s sample rate, 10MSa memory & 8" color TFT-LCD screen.

SDS6062 \$399

World's Smallest

The world's smallest MSO! This DIP-sized 200kHz 2 channel scope includes a spectrum analyzer and arbitrary waveform generator. It measures only 1 x 1.6 inches in size!

Xprotolab \$49



100MHz MSO

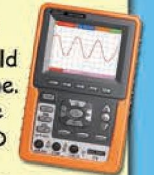
2 channel 100MSa/s oscilloscope and 8-ch logic analyzer USB 2.0 and 4M samples storage per channel with sophisticated triggering and math functions.

CS328A \$1359

Handheld 20MHz

Fast, accurate handheld 20MHz 1-ch oscilloscope. - 100 M/S sample rate - 3.5 in. color TFT-LCD - 6 hour battery life Inc. rugged impact-resistant case

HDS1021M \$299.95



More selections at:

www.saelig.com

Follow us on:



Saelig
unique electronics



NEW PRODUCTS

NEW TOOLS FOR WIRELESS CONNECTION TO ZigBee

The **A2530x24xxx** series, Anaren's new family of Anaren Integrated Radio (AIR) modules, are specifically designed to help OEMs develop products that wirelessly communicate in compliance with the ZigBee standard. Based on the Texas Instruments (TI) CC2530 low-power RF system-on-a-chip (SoC), which operates using TI's Z-Stack firmware, the family of AIR modules is bundled with AIR Support for ZigBee, which includes time-saving AIR-ZNP firmware (including more than 30 code examples), precertification to applicable global, regulatory standards, and development tools (e.g., Anaren's BoosterPack for TI MSP430 and Stellaris LaunchPad development kits).

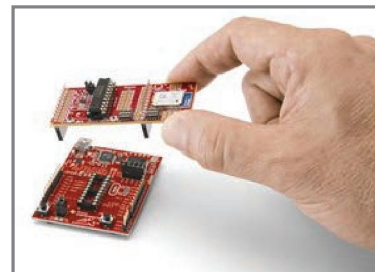
The A2530x24xxx devices require minimal RF engineering and ZigBee experience. They are easy to program for a shortened design cycle. The devices are available with an integral or connectorized antenna and a tiny, 2.5-mm × 11-mm × 19-mm standardized footprint. The devices are pre-certified to FCC/IC and compliant with ETSI. There is a choice of range-extender or non-range extender modules. The A2530x24xxx devices' additional features include a 2.4-GHz IEEE 802.15.4-compliant RF transceiver (TI's CC2530), a wide 2.2-to-3.6-V input voltage range, and excellent receiver sensitivity and robustness to interference (−95 dBm average).

Anaren has also introduced a BoosterPack featuring its new family of modules. The **CC2530 BoosterPack Kit** helps OEM engineers develop wireless applications using a TI LaunchPad for MSP430 or a Stellaris microcontroller. The BoosterPack provides "out-of-the-box" wireless connectivity to easily develop applications based on the ZigBee standard. It also includes AIR-ZNP firmware solution (based on TI's Z-Stack).

The kit includes three A2530E24A AIR Module BoosterPacks for connection to TI's MSP430 or Stellaris's LaunchPad development kit (LaunchPad not included). Each BoosterPack includes an on-board MSP430G2553IN20 Value Line microcontroller, pre-flashed with Anaren's AIR-ZNP firmware (based on TI's Z-Stack for the ZigBee standard).

Contact Anaren for pricing.

Anaren, Inc.
www.anaren.com



ENERGY-HARVESTING DISCOVERY KIT

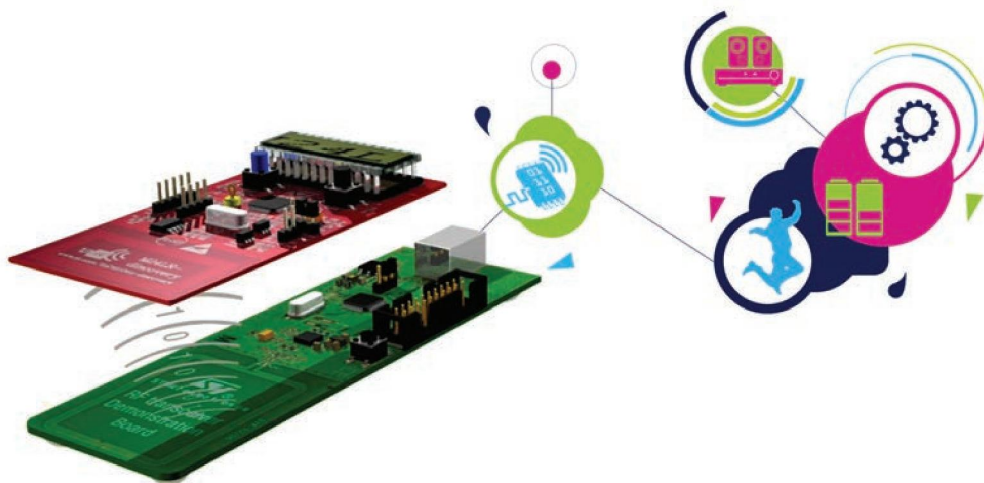
The **M24LR Discovery Kit** helps you design battery-free electronic applications that can exchange data with ISO15693-compatible NFC-enabled smartphones or radio-frequency identification (RFID) reader-writers. The kits help create and integrate energy-autonomous data collection, asset tracking, or diagnostics capabilities in applications, including phone and tablet accessories, computer peripherals, electronic shelf labels, home appliances, industrial automation, sensing and monitoring systems, and personal healthcare products.

With a combination of industry-standard serial bus (I²C) and contactless RF interfaces, the M24LR EEPROM memory is capable of communicating with host systems "over-the-wire" or "over-the-air." The M24LR's RF interface can convert ambient radio waves emitted by RFID reader-writers and NFC phones or tablets into energy to power its circuits and enable complete battery-free operation.

The M24LR Discovery Kit includes an RF transceiver board with a 13.56-MHz multiprotocol RFID/NFC transceiver (CR95HF) driven by an STM32 32-bit microcontroller, which powers and wirelessly communicates with a battery-less board. This board includes ST's dual-interface EEPROM memory IC (M24LR), an ultra-low-power 8-bit microcontroller (STM8L), and a temperature sensor (STTS75).

The M24LR Discovery Kit costs **\$17.50**.

STMicroelectronics
www.st.com

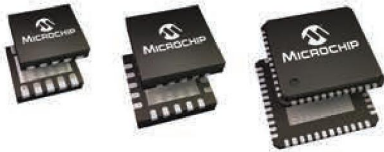


NEW PRODUCTS

We've got you covered!



8 | 16 | 32 bit
PIC® MICROCONTROLLERS



Feeling locked in?

Forced architecture and other compromises should not constrain your design. With Microchip's 8-, 16- and 32-bit solutions, tools and compilers, you won't have to.

The choice is easy when you think about it.

Visit Microchip.com/pic today.



DIVERSITY WITHOUT COMPROMISE

MICROCHIP.COM/PIC

DIVERSITY WITHOUT COMPROMISE

MICROCHIP.COM/PIC

E-FIELD-BASED 3-D GESTURE CONTROLLER

The configurable **MGC3130** is an electrical-field (E-field)-based 3-D gesture controller, providing low-power, precise, fast, and robust hand-position tracking with free-space gesture recognition. The controller features Microchip's **GestIC** technology, which enables intuitive, gesture-based, non-contact user interfaces for many end products.

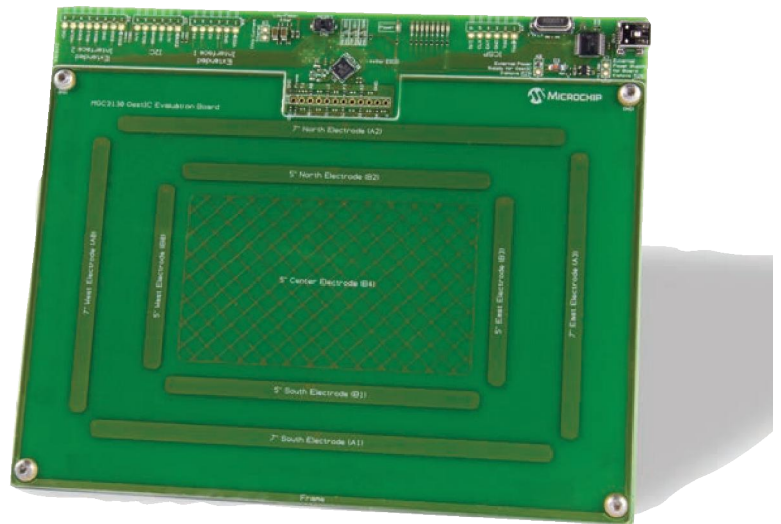
The MGC3130 includes 150-dpi, mouse-like resolution and a 200-Hz sampling rate to sense fast hand and finger motions. It has a super-low-noise analog front-end for high-accuracy interpretation of electrode sensor inputs. The controller's configurable Auto Wake-Up on Approach at 150- μ W current consumption enables always-on 3-D gesture sensing in power-constrained mobile applications.

The MGC3130's additional features include automated self calibration, 32-bit digital signal processing for real-time processing of x/y/z positional data, integrated flash memory to easily upgrade deployed products, and a 70-to-130-kHz E-field with frequency hopping to eliminate RF interference.

GestIC technology achieves high gesture-recognition rates through Colibri Suite, which is a library of 3-D gestures for hands and fingers that is preprogrammed into the MGC3130. The Colibri Suite combines a stochastic Hidden Markov model (HMM) and x/y/z hand-position vectors to provide recognized 3-D hand and finger gestures. Examples include Wake-Up on Approach, Position Tracking, Flick Gestures, Circle Gestures, and Symbol Gestures to perform functions (e.g., on/off, open application, point, click, zoom, scroll, free-space mouseover, etc.). The chip also provides prefiltered electrode signals for additional functionality.

GestIC technology uses thin sensing electrodes made of any conductive material, such as PCB traces or a touch sensor's indium tin oxide (ITO) coating, to enable invisible integration behind a device's housing. In addition, the technology provides 100% surface coverage, eliminating "angle-of-view" blind spots. With a detection range of up to 15 cm, the MGC3130 is well suited for products used in close proximity for direct user-to-device interaction.

In addition, Microchip's **Sabrewing MGC3130** single-zone evaluation kit enables development with the MGC3130 by providing a 5" or 7" selectable electrode size. The kit comes with the AUREA graphical user interface (GUI), which is available for a free download at www.microchip.com/get/DST9. The GUI enables designers to easily match their system commands to Microchip's Colibri Suite. The evaluation kit costs **\$169**. The MGC3130, featuring GestIC technology, is available in a 5-mm \times 5-mm, 28-pin QFN package. The controllers cost **\$2.26** each.



Microchip Technology, Inc.
www.microchip.com

CONTROLLER FOR SMOOTH SERVOMOTOR RESPONSE

The **SMC-01** is a manual servomotor controller for a single servomotor. The controller performs via an on-board potentiometer. A Microchip Technology PIC12F683 microcontroller is at the heart of the SMC-01. The potentiometer connects to the microcontroller to proportionally control the servomotor's rotation.

The servomotor's shaft is capable of responding as fast and as far as the potentiometer knob is rotated. A universal three-pin header enables easy connection to the servo motors. They are simply plugged into the board. The circuit is controlled by an inexpensive, eight-pin microcontroller and powered by a 9-V battery.

The unit can be purchased as a kit or fully assembled. The SMC-01 kit costs **\$24.95**. The fully assembled SMC-01A costs **\$34.95**.

Images SI, Inc.
www.imagesco.com



NEW PRODUCTS



Windows CE 6.0 Touch Controller

CUWIN

The CUWIN is a series of Windows CE touch controllers that are more cost-effective than a PC, but with more features than an HMI touch screen. Create sophisticated applications with C++ or any .Net language.

533MHz ARM CPU
128MB SDRAM & Flash
SD Card Support
Ethernet, RS-232/485
USB, Audio Out
Windows Embedded CE 6.0

The CUPC is a series of industrial touch panels with all the features of a modern PC for the most feature-rich user experience.

ATOM N270 1.6GHz CPU
2GB RAM
320GB HDD
SD Card Support
Color Display (1024 x 768)
RS-232, Ethernet
USB, Audio Out
Windows XP/XP Pro

CUPC

Industrial Touch Panel PC with ATOM Processor



C-Programmable Modular Industrial Controller

MOACON

The MOACON is a modular, C programmable, ARM-based automation controller designed for industrial environments.

Choose from a diverse, feature-rich selection of modules including:

- Digital I/O
- Analog I/O
- RS-232/485
- Motor Control
- Ethernet
- High-speed Counter & PWM

BASIC with LADDER LOGIC CONTROLLER



only \$29

New

Integrated CUBLOC Controller and I/O Board

CB210

The CB210 is an inexpensive, integrated CUBLOC controller and I/O board programmable in both BASIC and Ladder Logic.

I/O Ports x20	10-bit A/D x6
PWM x3	RS-232 x1
80KB Program Memory	3K Data Memory

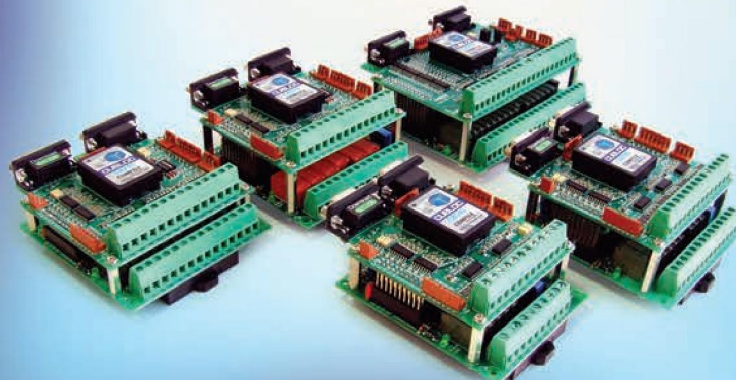
The CUSB is a series of compact, CUBLOC-integrated, industrial I/O boards programmable in both BASIC and Ladder Logic.

CUBLOC CB280 Core Module

- 80KB Program Memory
- 3KB Data Memory
- DC 24V Inputs x9~16
- A/D Inputs x2~6
- Relay Outputs x6~16
- PWM x2~6
- High-speed Counters x0~2
- RS-232/485 x1~2

CUSB

Integrated Industrial Controllers



COMFILE

TECHNOLOGY

1175 Chess Dr., Suite F, FOSTER CITY, CA 94404
call : 1-888-9CUBLOC (toll free)
1-888-928-2562
email : sales@comfiletech.com

www.comfiletech.com

DIGITAL CLASS D AUDIO DEVELOPMENT FOR 32-BIT EMBEDDED DESIGNS

The **Class D ToolStick** kit is a cost-effective USB-based evaluation kit that enables developers to add digital Class D audio capabilities to 32-bit embedded designs based on Silicon Labs's SiM3U1xx Precision32 microcontrollers. The kit helps developers upgrade basic "buzzer/beeper" alert sounds used in personal medical devices, fitness equipment, high-end toys, small appliances, and other consumer electronics products to sophisticated voice prompts, music, sound clips, and streaming audio.

The SiM3U1xx microcontrollers include the following: a 300-mA, high-drive I/O that can directly drive a small speaker; a crystal-less USB transceiver compatible with the USB audio interface; two 250-ksps, 12-bit ADCs; and an I²S receiver that supports audio streaming from a PC, a portable music player, or a range of I²S-enabled audio devices. The only external components required to drive Class D audio from SiM3U1xx microcontrollers are inexpensive inductors, some capacitors, and ferrite beads.

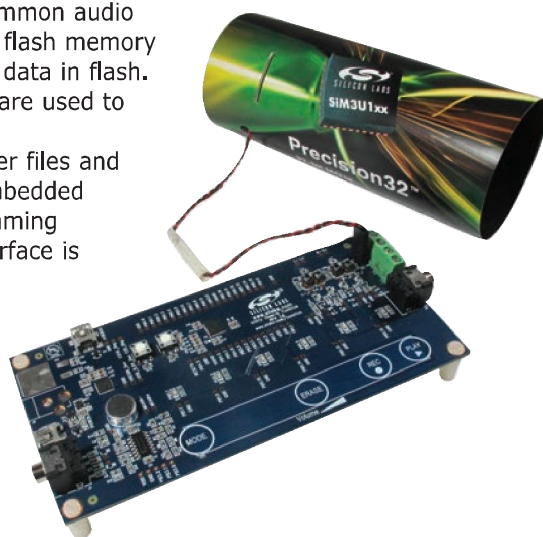
You can use the ToolStick to add capacitive-touch buttons and sliders to 32-bit embedded systems. The SiM3U1xx microcontrollers' high-drive I/Os with PWM can be used to directly drive other components (e.g., small motors), without using separate power field-effect transistors (FETs).

The Class D ToolStick kit is powered from USB using the SiM3U1xx microcontroller's internal 5-V regulator. The board uses a simple speaker to play music from a stereo jack, a computer, or a recorded message. The ToolStick provides four modes of operation. The microcontroller's on-chip ADCs are used to sample data from a portable music player or USB audio streaming from a PC. The kit uses a common audio compression algorithm to play prerecorded sound clips stored from on-chip flash memory and it uses an audio-compression algorithm as a voice recorder that stores data in flash. Capacitive-touch buttons and control volume with a capacitive-touch slider are used to handle mode transition.

The Class D ToolStick evaluation kit comes complete with hardware Gerber files and software, which helps streamline the process of adding Class D audio to embedded applications. The ToolStick features a built-in USB-based debugger/programming interface and accessible pins for easy prototyping. The ToolStick debug interface is fully operational with Silicon Laboratories's complimentary Precision32 IDE, compiler, AppBuilder crossbar configuration software, and Keil toolchains.

The Class D ToolStick evaluation kit includes full source code and implements a Class D amplifier demonstration using a small-footprint, 40-pin, 6-mm × 6-mm package SiM3U1xx microcontroller. The kit costs **\$35**.

Silicon Laboratories, Inc.
www.silabs.com



MCUs WITH INTEGRATED USB ENABLE HIGH-SPEED CHARGING

The **RL78/G1C** group of 16-bit microcontrollers conforms to the USB Battery Charging Specification, Revision 1.2. The microcontrollers provide USB host/function interfaces, which support full-speed and low-speed USB communication. The USB interface is used between PCs and PC peripheral equipment and also as a general-purpose interface for consumer and industrial equipment in applications such as smartphone accessories, portable healthcare devices, AV accessories, and game and industrial equipment.

The microcontrollers' battery-charging function enables high-speed charging up to 1.5 A. They also integrate a 1% accurate high-speed oscillator.

The RL78/G1C microcontroller group is the first RL78 microcontroller family that includes USB host/function and can be easily deployed in existing systems utilizing other RL78 family products. The integration of USB functionality alongside RL78's smart peripherals enables them to achieve 71 μ A/MHz in full operation and 0.23 μ A in Stop mode (RAM retained). The first series within the RL78/G1C microcontroller group features 32-to-48-pin devices in packages as small as 5 mm × 5 mm, up to 32 KB of flash, and 5.5 KB of on-board RAM.

Samples of the RL78/G1C group of microcontrollers are available. Prices vary by USB peripheral functions and USB host functions including packages and number of pins. For example, the 32-pin LQPF package R5F10KBCAFP device with USB peripheral function costs **\$1** per unit, in 10,000-unit quantities.

Renesas Electronics Corp.
www.renesas.com



NEW PRODUCTS

Alexander Pozhitkov

Member Name: Dr. Alexander Pozhitkov

Location: Seattle, WA

Education: MS in Chemistry, Moscow State University,
PhD in Genetics and Bioinformatics, University of Cologne,
Germany

Occupation: Research scientist

Member Status: He has been a subscriber for a year.

Technical Interests: Alex is interested in low-level hardware programming and high-voltage electronics, including vacuum tubes.

Most Recent Embedded Tech-Related Acquisition: He recently received a single-board fanless PC with a solid-state hard drive as a gift.

Current Projects: Alex is further developing the NakedCPU platform he wrote about in his two-part article series, "The NakedCPU," (*Circuit Cellar* 259–260, 2012).



Thoughts on the Future of Embedded Technology:

Alex says he's worried that embedded solutions are becoming less transparent. He remembers working with one system that had several DVDs of examples and libraries but it didn't have a comprehensive guide to the system's architecture. "As a researcher and someone who wants to get to the bottom of things, such a situation is frustrating. This is certainly my personal researcher's view. I am not commenting on the application side of increasingly complicated embedded systems."



AP CIRCUITS
PCB Fabrication Since 1984

As low as...

\$9.95
each!

Two Boards
Two Layers
Two Masks
One Legend

Unmasked boards ship next day!

www.apcircuits.com



System on Module SoM-3517

- TI ARM Cortex-A8 600 MHZ Fanless Processor
- Up to 512 MB of DDR2 SDRAM
- Up to 1GB of NAND Flash
- Up to 2GB of eMMC Flash
- 2 High Speed USB 1.1/2.0 Host ports
- 1 High Speed USB 2.0 OTG port
- 4 Serial Ports, 2 I2C and 2 SPI ports
- Processor Bus Expansion
- 10/100 BaseT/Fast Ethernet
- CAN 2.0 B Controller
- Neon Vector Floating Point Unit
- 24-bit DSTN/TFT LCD Interface
- 2D/3D Accelerated Video w/ Resistive Touch
- Small, 200 pin SODIMM form factor (2.66 x 2.375")



The SoM-3517 uses the same small SODIMM form-factor utilized by other EMAC SoM modules and is the ideal processor engine for your next design. All of the ARM processor core is included on this tiny board including: Flash, Memory, Serial Ports, Ethernet, SPI, I2C, I2S Audio, CAN 2.0B, PWMs, Timer/Counters, A/D, Digital I/O lines, Video, Clock/Calendar, and more. The SoM-3517M additionally provides a math coprocessor, and 2D/3D accelerated video with image scaling/rotation. Like other modules in EMAC's SoM product line, the SoM-3517 is designed to plug into a custom or off-the-shelf carrier board containing all the connectors and any additional I/O components that may be required. The SoM approach provides the flexibility of a fully customized product at a greatly reduced cost. Contact EMAC for pricing & further information.

<http://www.emacinc.com/som/som3517.htm>

Since 1985
OVER
28
YEARS OF
SINGLE BOARD
SOLUTIONS

EMAC, inc.
EQUIPMENT MONITOR AND CONTROL

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

Anaren, Inc.

Client: Anaren, Inc. (www.anaren.com/AIR)

Location: 6635 Kirkville Road, Syracuse, NY 13057

Contact: AIR@anaren.com

Embedded Products/Services: Anaren Integrated Radio (AIR) modules are tiny, surface-mountable, pre-certified transceiver modules based on Texas Instruments's (TI) low-power RF chips—and they are designed to ease development of wireless functionality for non-RF savvy OEMs. More information is available at www.anaren.com/AIR.

Product Information: AIR modules reduce time and costs associated with adding wireless capability to a wide range of electronic products, given that they are easy-to-implement (e.g., small format, SMT technology), pre-certified to FCC, IC, or ESTI (saving many certification expenses), and supported by a range of development tools (e.g., the new CC2530 BoosterPack kit for development of ZigBee standard applications using TI's MSP430 or Stellaris LaunchPad kits).



Exclusive Offer: *Circuit Cellar* readers in the US, Canada, and Europe can enter to win a free CC2530 BoosterPack kit for ZigBee applications (\$100 value, plus there will be no charge for shipping this kit). To participate, send an e-mail to AIR@anaren.com before midnight (EST) March 31, 2013, with "2013 *Circuit Cellar* offer" in the subject line. The e-mail must include: Name, Title, Company, Company Website, Complete Shipping Address, and Telephone Number (for shipping confirmation). More details on the kit's content and benefits are available at www.anaren.com/air/cc2530-boosterpack-kit. No purchase necessary. Void where prohibited.



Circuit Cellar prides itself on presenting readers with information about innovative companies, organizations, products, and services relating to embedded technologies. This space is where Circuit Cellar enables clients to present readers useful information, special deals, and more.

Get PUBLISHED. Get NOTICED. Get PAID.

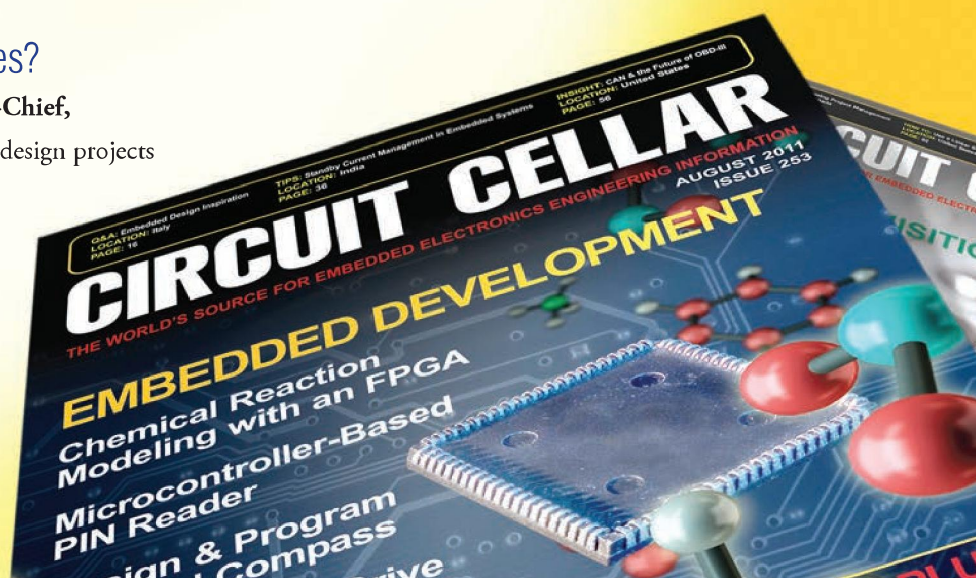
Circuit Cellar feature articles are contributed by professional engineers, academics, and students from around the globe. Each month, the editorial staff reviews dozens of article proposals and submissions. Only the best make it into the pages of this internationally respected magazine.

Do you have what it takes?

Contact C. J. Abate, Editor-in-Chief, today to discuss the embedded design projects and programming applications you've been working on and your article could be featured in an upcoming issue of *Circuit Cellar* magazine.

editor@circuitcellar.com

CIRCUIT CELLAR



TEST YOUR EQ

CONTRIBUTED BY
DAVID TWEED

Answer 1—Yes, given a few basic capabilities, it is possible to implement multiple levels of interrupt priority in software. The basic requirements are that it must be possible to reen-able interrupts from within an interrupt service routine (ISR) and that the different interrupt sources can be individually masked.

Answer 2—In normal operation, all the interrupt sources are enabled, along with the processor's global-interrupt mask.

When an interrupt occurs, the global interrupt mask is disabled and the "master" ISR is entered. This code must (quickly) determine which interrupt occurred, disable that interrupt and all lower-priority interrupts at their sources, then reen-able the global-interrupt mask before jumping to the ISR for that interrupt. This can often be facilitated by precomputing a table of interrupt masks for each priority level.

Answer 3—For one thing, the start-up latency of all the ISRs is

**Answers for
Issue 270**

increased by the time spent in the "master" ISR. This can be a problem in time-critical systems. This scheme enables interrupts to be nested, so the stack must be large enough to handle the worst-case nesting of ISRs, on top of the worst-case nesting of non-interrupt subroutine calls.

Finally, it is very tricky to do this in anything other than Assembly language. If you want to use a high-level language, you'll need to be intimately familiar with the language's run-time library and how it handles interrupts and reen-ability, in general.

Answer 4—Yes, on most such processors, you can execute a subroutine call to a "return from interrupt" instruction while still in the master ISR, which will then return to the master ISR, but with interrupts enabled.

Check to see whether the "return from interrupt" affects any other processor state (e.g., popping a status word from the stack) and pre-prepare the stack accordingly.

Also, beware that another interrupt could occur immediately thereafter, and make sure the master ISR is reentrant beyond that point.

Contributed by David Tweed

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

NEW! DS2000 Digital Oscilloscopes



Starting at
\$839

COMPARE & SAVE
vs Tektronix TDS2000C

Our latest Best-in-Class Scope offers an unmatched feature/value package!

Now there's a 2 channel scope with a dynamic feature-set and an extremely low noise floor to help you capture smaller signals... starting at only \$839! That's the newest member of our market-redefining family of oscilloscopes, the DS2000! This fast and versatile instrument covers frequencies up to 200 MHz with two channels and has a wide vertical range (500uV/div~10V/div) for the best view of your smallest signals. Add a 2 GSa/s max. sample rate and 14 Mpts. of memory depth to its user-friendly interface and 8 inch WVGA display and you've got a scope that delivers unparalleled performance!

RIGOL
Beyond Measure

Check out the Best Value in Scopes, call 877-4-RIGOL-1 or visit RigolScope.com

Build a Digital Dip Meter

A dip meter is a handy piece of test equipment typically used to check a device's resonant frequency or as a signal source to tune receivers. This article describes how to use a microcontroller to digitize a dip meter's display and explores how to maintain functionality when converting a design from analog to digital.

A dip meter, originally known as a grid-dip meter, is a piece of test equipment that goes back to the early days of amateur radio. It consists of a tunable RF oscillator with the tuning coil exposed outside the chassis. When the coil is brought near a resonant circuit (e.g., an inductor-capacitor tank circuit) and when the oscillator is tuned to the same frequency as the resonant circuit, the circuit will draw energy from the oscillator. This results in a dip of a meter that is in the oscillator's grid circuit (in an older vacuum-tube design). The coils plug into the dip meter so different coils may be used for different frequency bands.

The dip meter's coil is part of a tuning circuit consisting of the coil and a variable capacitor. The variable capacitor is connected to a dial on the front of the chassis that is marked with the oscillator's frequencies. When the meter "dips," indicating the dip meter oscillator is at the same frequency as the resonant circuit, you can read the frequency from the dial.

In a more modern dip meter, a transistor is used instead of a

vacuum tube and the meter is typically in the gate circuit (if the transistor is a FET) or the base or collector (if the transistor is a bipolar type). **Figure 1** shows a typical schematic for a transistor dip meter using an MPF102 FET. The meter is in the FET's gate circuit.

Although the schematic shows the meter's negative side connected to the gate circuit and the positive side grounded, it is not wired backward. Due to this circuit's self-biasing nature, the

gate voltage is negative while the circuit is oscillating. The oscillator sine signal appears at the gate, but it is shifted below ground so the average (DC) level is negative. C5, the capacitor across potentiometer RV1, filters out the RF leaving the negative DC value applied across the meter.

A dip meter's typical purpose is to check the resonant frequency of an LC tank circuit, crystal, or antenna. Dip meters can be used as signal sources for tuning receivers. I have even used one as a local oscillator for an RF mixer circuit.

Dip meters are simple to build and use. Thousands have been bought, constructed from kits, or built from scratch by amateur radio operators over the years. Fewer amateurs build their own equipment now and many of the dip meter's uses have been taken over by other equipment (e.g., inexpensive frequency counters). In addition, many circuits that dip meters used to measure (e.g., LC tanks) are now implemented with ceramic filters or other methods.

Why discuss an obsolete tool for amateur radio in a magazine that isn't necessarily oriented to radio amateurs? The answer is

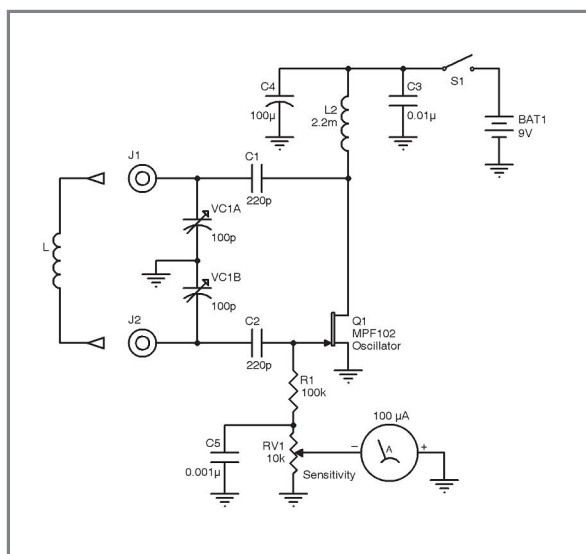


Figure 1—Here is a typical analog dip meter circuit's schematic. The 100- μ A meter in the gate circuit dips when the oscillator is tuned to the same frequency as the circuit it is measuring.

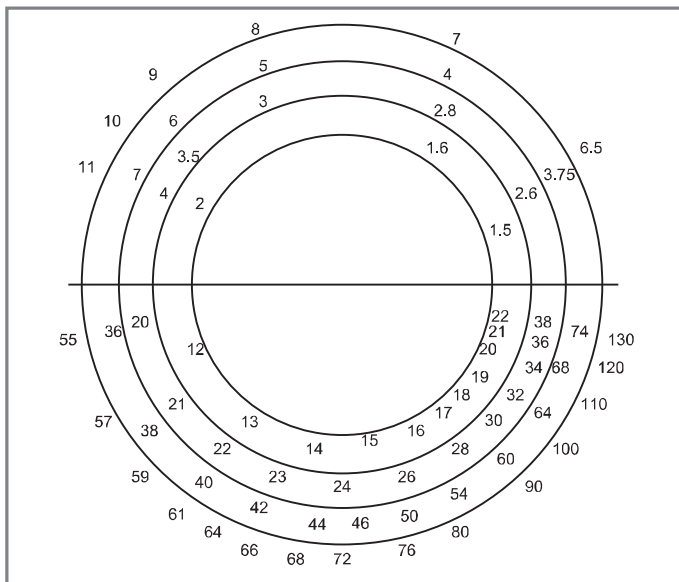


Figure 2—This is a dial from an old homemade analog dip meter. Note the compression of the frequency bands to the end of the dial with the higher frequencies.

straightforward: because this project uses a microcontroller to digitize a dip meter's frequency display, which solves several problems to using dip meters. Consequently, this article provides an instructive look at what is needed to convert a design from analog to digital while retaining the original functionality. It also demonstrates some issues and obstacles that can be solved by adapting an older analog design to digital.

THE OLD WAY OF DOING THINGS

Although the dip meter is a simple circuit with a simple concept, it does suffer from some drawbacks. I will discuss each of these in detail.

Frequency pulling: This is one of the most problematic issues in dip meter use. When the dip meter's coil is brought near a resonant circuit, the resonant circuit draws energy from the oscillator. It also pulls the oscillator frequency. In some cases, the resonant circuit can pull the oscillator by several megahertz. There are some ways around this. Usually, you try to couple the oscillator to the resonant circuit as loosely as possible by increasing the distance between the oscillator coil and the resonant circuit's coil. This reduces but does not eliminate the issue, and it reduces the dip meter's sensitivity so the dip is less pronounced and harder to see.

Even with minimal coupling, there is some frequency shift. And, since the frequency markings on the dip meter dial are fixed, you do not know how much the frequency has been shifted.

Bandspacing: A dip meter's dial is usually circular or semi-circular. [Figure 2](#) shows a typical dial from a homemade meter. There are only so many bands that can be shown on the dial. So, if there is room for eight bands on the dial, then only eight coils can be made and only eight frequency ranges can be used. The typical frequency range using a common variable capacitor is about 2:1, so for a dip meter that is going to operate from 1 to 100 MHz, you might have seven

bands like this:

1–2 MHz, 2–4 MHz, 4–8 MHz, 8–16 MHz, 16–32 MHz, 32–64 MHz, 64–128 MHz

If you were building a dip meter like this, you would actually put in a little overlap. So, the second band might be 1.9 to 3.8 MHz so it overlaps the first band, the third band would be 3.6 to 7.2 MHz, and so on. But either way, if you wanted to have expanded tuning over the range of, say, 4 to 4.5 MHz, you would need another frequency band on the dial. You would quickly run out of dial space if you did this for more than one or two frequency bands.

Dial compression: Because of the way variable tuning capacitors work, the frequencies tend to be compressed at one end of the dial. [Figure 2](#) shows that the first third of the frequency span takes up half the dial and the remaining two thirds are compressed into the last half of the dial. This makes the higher frequencies harder to read.

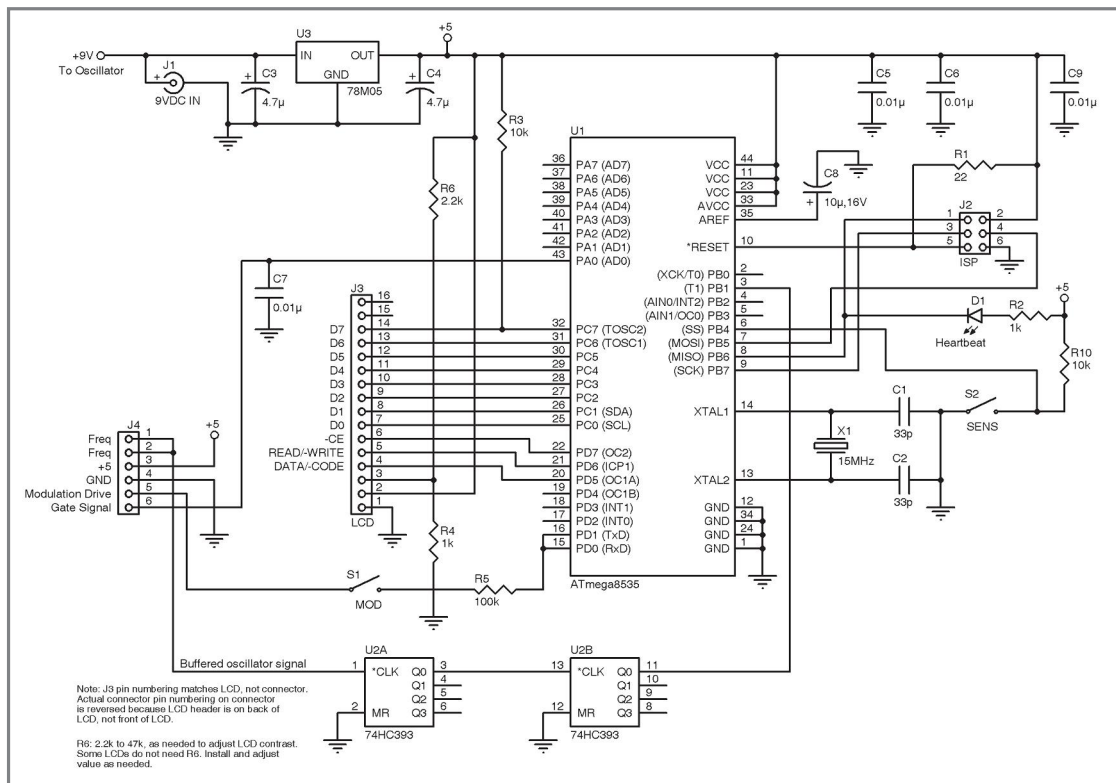
Dial resolution: It can be difficult to read the frequency precisely due to the inability to show more than a limited number of lines on the dial and the parallax of reading the frequency from a tuning dial connected to the variable capacitor's shaft.

Calibration: This is not a problem for a purchased meter, but if you were building a meter from scratch, you would have to calibrate the dial. This means you need a relatively



Photo 1—This digital dip meter prototype shows an external LC circuit and the various tuning coils. The block of wood is used so the test circuit will be the same height as the dip meter coil.

Figure 3—The circuit uses an Atmel ATmega8535 microcontroller in a PLCC-44 package.



accurate receiver, oscilloscope, or frequency counter to calibrate each band.

NEW SOLUTIONS

This project's circuit solves most of those problems by using an LCD to digitally display the frequency. I will now detail how each of the problems are resolved.

Frequency pulling: The resonant circuit will still pull the frequency, but since the LCD directly reads the actual frequency, you know exactly how much. When I tested the prototype, I was surprised at how much a crystal or resonant LC circuit would pull the oscillator frequency.

Bandspacing: Since there is no dial, there is no limitation on the number of coils. Consequently, you can have coils for much smaller bandwidths that will enable more precise frequency tuning. In the prototype circuit, I bridged the coil for 6.7 to 15.5 MHz with a 56-pF capacitor and the frequency span became 4.8 to 6.3 MHz. So, the new tuning range is 1.3:1 instead of 2.3:1.

Dial compression, dial resolution, and calibration: The digital meter has no dial, so there are no compression or resolution issues and no calibration is needed. It is true that the compression effect is still there. The last half of the

tuning arc is still the last two thirds of the tuning range. But the frequency can be directly read, so this is more a problem of manual dexterity than accuracy.

CIRCUIT COMPLEXITY

These improvements do come at a cost, of course. Some of the drawbacks are detailed below.

Increased circuit complexity: Instead of a simple, single-transistor circuit, the circuit has four transistors, an additional regulator, a microcontroller, a crystal, and assorted other parts. In addition, the microcontroller firmware must be developed.

Increased design complexity: The original circuit can operate to about 200 MHz if carefully constructed. The only limiting factor is the transistor itself and the circuit layout. For the digital design, the maximum frequency can be limited by the oscillator transistor, the microcontroller's divider, and the buffer amplifier. In the prototype, the divider IC limits the maximum operating frequency to about 100 MHz.

CONSTRUCTION

Photo 1 shows the digital dip meter. The schematic is shown in Figure 3 and Figure 4. The circuit is built on three prototype boards so everything fits in the

case. You could build it on one board, but you might find it difficult to get the circuit to fit neatly into the case since the LCD and the tuning capacitor are at different heights.

All components are standard 0.25-W resistors, 10% or 20% capacitors at 25 or 50 V. The transistors were TO-92. The Texas Instruments (TI) UA78L05 fixed-voltage IC voltage regulator was a TO-92. The TI UA7805 was a surface-mount power package used for voltage regulators and transistors, but a TO-220 would also work. Electrolytic capacitors can be radial or axial types. Of course, you could make a PCB and use all surface-mount components to shrink the circuit size.

The digital dip meter features an Atmel ATmega8535 microcontroller, which is readily available, inexpensive, and has an on-chip ADC for the meter function. It also has a counter with an external count input, which is essential for frequency counting. Aside from the requirement for an external count input, almost any microcontroller can be used (although some would obviously need an external ADC). The prototype's microcontroller was in a plastic leaded chip carrier (PLCC) package so it could be easily socketed.

The display is an 8 x 2 LCD. The top



Mobility makes You smarter.

However you fill in the blank, CTIA 2013™ is for those serious about wireless.

Presented by CTIA–The Wireless Association®, the proven authority leading the mobile movement since its inception. CTIA 2013 will continue to illuminate the future by showcasing the leaders, ideas and experiences that are transforming our dynamic industry.

CTIA2013™

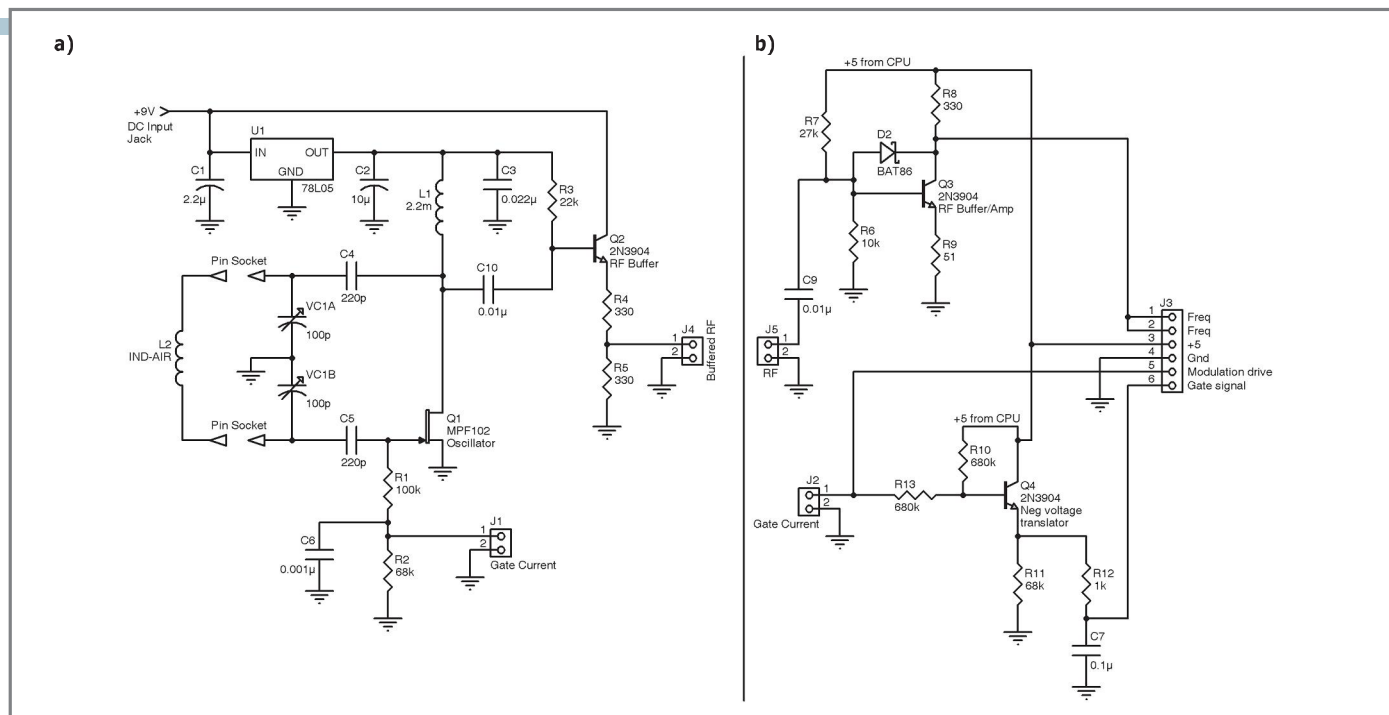
THE Mobile Marketplace

May 21-23, 2013

Sands Expo & Convention Center Las Vegas, NV

CTIA2013.com

Prepare for tomorrow. Get smarter. Think big.



line displays the frequency (four digits) and the bottom line shows the analog gate current (really the voltage) amplitude. The analog meter function uses custom characters programmed into the LCD memory to simulate an analog display. Using this technique, an eight-character LCD can display up to 40 analog values. I described this technique in my article "Analog Bar Graph Display" (*Circuit Cellar* 183, 2005).

On the oscillator board, transistor Q1 is the oscillator, which is the same as in the analog design. Q2 buffers the RF signal so the digital front end doesn't load the oscillator signal.

On the buffer board, Q1 is a negative voltage translator. The voltage at the oscillator transistor's gate is negative so it has to be translated to a positive voltage for the microcontroller ADC. Q2 buffers and normalizes the RF signal. Note that diode D2 on the buffer board is a Schottky diode. This is important to keep Q3 from saturating. Without D2, or if D2 is a normal silicon diode, Q3 will saturate and the operating frequency will be severely limited.

The buffered RF signal from the buffer board connects to U2 on the processor board. U2 is an NXP Semiconductors 74HC393 CMOS device that is wired to divide the incoming frequency by 32. This enables operation to about 100 MHz. The divider is needed because the microcontroller can only accept a frequency input up to about 3.75 MHz.

The microcontroller accepts the divided frequency output into one of its counter inputs and counts the frequency by periodically capturing and sampling the count. I chose the 15-MHz crystal to make the math easy when converting counts to megahertz.

The buffered, level-shifted voltage at the oscillator gate drives the analog input on ADC0. The gate voltage's negative value must be translated and scaled to match the microcontroller ADC's analog input range. The scaling circuit is fairly simple because the display is viewed by the user, so precise

scaling is not needed. Some analog circuit conversions require more complicated level shifting and scaling circuits to bring an analog signal into an acceptable range.

Switch S2 is read by the microcontroller and used to adjust the analog meter's sensitivity so dips can be more easily seen when the gate voltage's value is low. Because the oscillator gate goes less negative when it is in resonance with an external circuit, the "dip" is actually a "peak" in this circuit (i.e., the meter voltage jumps up, not down, when resonance with an external circuit is reached).

The software senses S2's value (high or low) and either divides the ADC output by 2 or by 4 before displaying the analog value on the LCD. The ADC is configured to use the microcontroller's internal 2.5-V reference.

Switch S1 provides an audio modulation output to the oscillator via the gate circuit when closed. The modulation signal is generated on the asynchronous serial output pin by the microcontroller. The microcontroller continuously sends a fixed-serial byte to the asynchronous serial transmitter to generate a fixed tone. This enables the dip meter to be used as a modulated signal source for checking receivers.

Coils are connected to the circuit using a pair of sockets from a pin-and-socket connector, soldered to the oscillator board, and passed through two holes drilled in the case. Banana jacks or pin jacks can also be used.

The prototype's coils are simply off-the-shelf inductors mounted on perfboards for strength. Ordinarily, they would be covered with heatshrink tubing. I left them uncovered so the construction would be obvious. You could also hand-wind coils using wood dowels or plastic forms as cores.

The tuning capacitor is not critical. It needs to be a two-section variable capacitor with a shaft on which a knob can be attached. This type of capacitor is not as common as it once was, but you can still find them on the Internet. I used

an air-variable capacitor. You could also use the type of poly capacitors that come from a transistor radio. The schematic shows the capacitor with two identical 100-pF sections. You could also use capacitors with two 270-pF sections or other values. The two sections do not need to be identical values; 135- and 100-pF sections would work. You could go down to something such as 50-pF per section, but you would need more coils to cover the same frequency range. The circuit runs from an external "wall wart," 9-VDC, 300-mA power supply.

TESTING THE CIRCUIT

The circuit is fairly easy to test. You simply plug in a coil, apply power, and verify that the oscillator frequency is displayed and the analog value is showing on the display's bottom line. As you tune across the frequency band, the analog voltage will vary.

You can test the dip functionality by connecting an inductor and capacitor in parallel and checking for a dip. Photo 1 shows an old choke from my junkbox in parallel with a ceramic capacitor. When you tune through their resonant frequency, there will be a small jump in the analog reading.

You can also check the circuit by connecting a crystal's leads with a two-turn link of wire then placing the wire link near the dip meter coil. However, crystals have very sharp tuning, so you must tune very slowly to see the peak. Once you tune the circuit to resonance, you can see how much the resonant circuit pulls the oscillator frequency by moving the dip meter away from the resonant circuit and watching the frequency change.

GOING FURTHER

You can improve the circuit. You can replace U2 with a front-end counter. This would enable the circuit to operate at the oscillator's full range.

You could replace switch S1 with a potentiometer connected to analog input ADC1 so the sensitivity is continuously variable, as on the analog dip meter. The potentiometer would just provide a varying voltage to the microcontroller that would be used to determine the analog signal's scaling

factor. Or, you could use a potentiometer to vary the analog voltage to the microcontroller ADC input and no software scaling would be needed. You could replace R5 on the oscillator board with a 300- or 500-Ω potentiometer.

A 16-character LCD would enable better resolution of the analog "meter"

value by enabling 80 discrete analog voltages to be displayed instead of 40. But that would come at the expense of a larger case.

This project demonstrated some interesting design considerations. It also showed the advantages of converting an analog circuit like this to digital monitoring and control. 📺

Stuart Ball is a registered professional engineer with a BSEE and an MBA. He has more than 30 years of experience in electronics design. He is currently a principal engineer at Seagate Technologies.

RESOURCE

S. Ball, "Analog Bar Graph Display," *Circuit Cellar* 183, 2005.

SOURCES

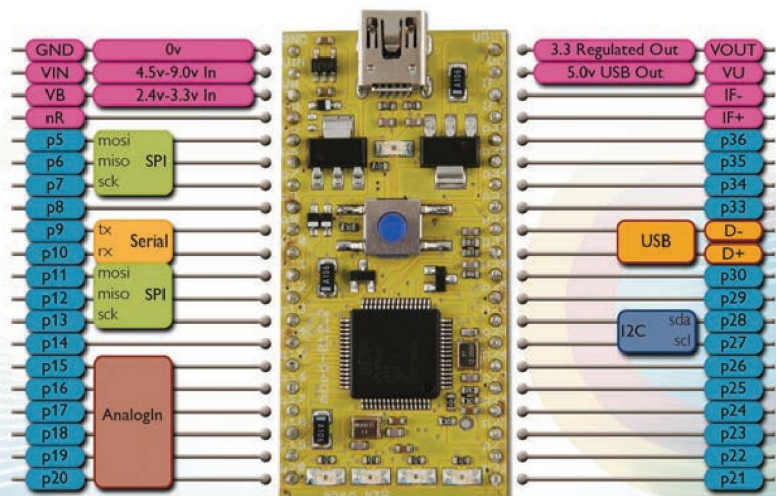
ATmega8535 Microcontroller
Atmel Corp. | www.atmel.com

74HC393 CMOS Device
NXP Semiconductors | www.nxp.com

UA78L05 and TI UA7805 Fixed-voltage IC voltage regulators
Texas Instruments, Inc. | www.ti.com

mbed

mbed NXP LPC1114U24 Microcontroller



Rapid prototyping for USB Devices, Battery Powered designs and 32-bit ARM® Cortex™-M0 applications

<http://mbd.org>

3-D Paint

A Complete Hardware and Software Package

3-D Paint is a complete hardware and software package that uses free space as a canvas and enables you to draw in 3-D by measuring ultrasonic delays. A standard PC running MATLAB captures your movements and returns them in real time.

With the goal of designing an exciting system for a design course at Cornell University, we built a 3-D paint program comprising hardware, a microcontroller, and a PC running MathWorks's MATLAB computing environment. All three modules must interact to enable an artist to wave a pen in the air and see his movements translated in real-time to various projections on the PC (see [Photo 1](#)). We wanted to design something that relied on many aspects of our education, including: physics, mathematics, A/D circuits, signal processing, microcontroller programming, peripheral communication, and high-level coding. In this article we'll explain the design's hardware and software and their implementation details. We'll also discuss design tradeoffs and engineering challenges we encountered in the project's development.

HIGH-LEVEL OVERVIEW

The design's foundation relies on the fact that the speed of sound is constant in a given medium. Under everyday conditions, the speed of sound in air is 340.29 mps. This means there is a one-to-one mapping between time and distance for sound propagation. By constructing a pen equipped with an ultrasonic transmitter and a stand with ultrasonic receivers, we could calculate their separating displacement by emitting a sound pulse from the pen and recording the time delay until it was received by the stand. Using three uniquely positioned receivers on the stand and collecting three distinct displacements, we could stipulate the position of the artist's pen relative to the stand in 3-D space. [Figure 1](#) shows our resulting coordinate system, including the

stand design and receiver positioning.

To map the three displacement measurements to trilaterated x, y, z coordinates, we derived the following relationships:

$$z = \frac{d_1^2 - d_3^2 + l^2}{2l}$$
$$y = \frac{d_1^2 - d_2^2 + l^2}{2l}$$
$$x = \mathfrak{R}(\sqrt{d_1^2 - y^2 - z^2})$$

With this physical and mathematical means of determining 3-D coordinates, we turned our attention toward the actual infrastructure that would enable the capturing and computation of sound delays.

The microcontroller's primary function was to facilitate the rapid acquisition of time delays between the transmitter and the three receivers. We tasked this job to a microcontroller because of its inherent ability to interface with analog hardware and communicate with higher-level machines (e.g., PCs).

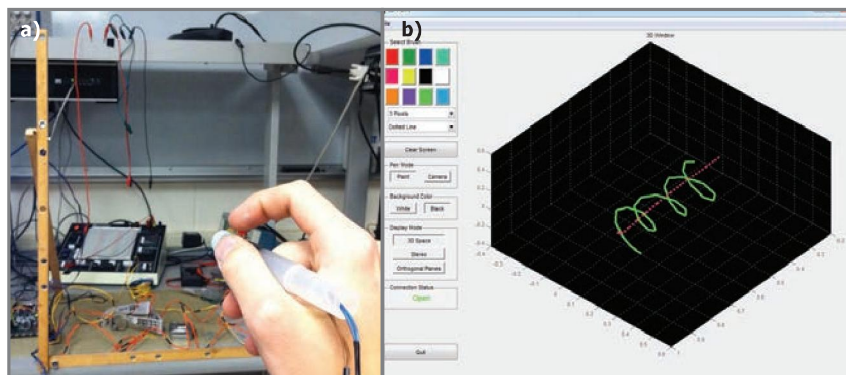


Photo 1—The 3-D paint program includes hardware, a microcontroller, and a PC running MATLAB. **a**—Here is the interface from the artist's point of view. **b**—This is the 3-D Paint project's MATLAB GUI.

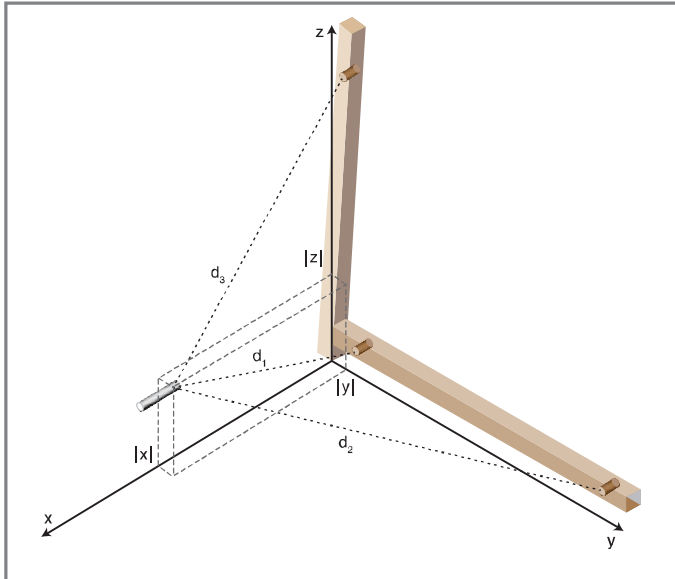


Figure 1—This is the mathematical coordinate system for trilateration and stand/receiver design.

The hardware was broken up into two components: emission and detection. On the emission side, we needed hardware that would provide a sufficiently strong signal to drive our ultrasonic transmitters. On the detection side, we needed appropriate hardware amplifiers that could take the small signal from a receiver and trigger the microcontroller when a sound pulse was detected.

MATLAB was where the bulk of the artist interaction was centralized. It was used to produce a fully functional GUI so the artist could change various parameters about the paintbrush (e.g., color, etc.) and receive visual feedback as to what was being drawn. Furthermore, MATLAB also needed to interpret the pen position in a “camera” mode in addition to a “paint” mode, which would enable the pen to function as a camera to view the drawing. [Figure 2](#) shows a high-level diagram of the complete system.

HARDWARE

When designing the receiver stand, we mounted all three receivers in one plane facing the design space so the pen could always be simultaneously oriented toward each receiver. Combined with a careful selection of wide-directionality ultrasonic Rx/Tx pairs, this enabled us to trigger all receivers with a single transmission pulse from the pen. Positioning the receivers 47 cm from the stand origin produced a happy medium that was wide enough for good sensitivity, yet compact enough that Rx/Tx directionality was not a concern.



Photo 2—The pen, which features a mounted push button, was made from an emergency whistle.

We constructed the pen from an emergency whistle (see [Photo 2](#)). We drilled a hole on top of the whistle to mount a single-pole, single-throw push button (to signal when the

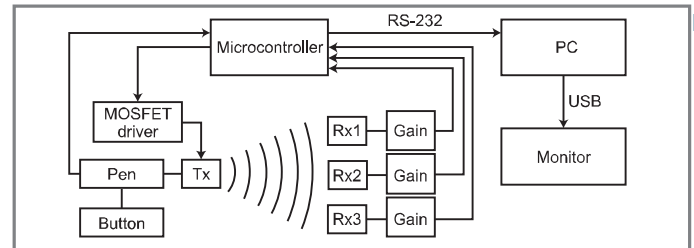


Figure 2—The complete system includes a microcontroller, a PC, a monitor, a MOSFET driver, a pen, and a button.

artist wanted to draw). We drilled another hole on the front to mount the ultrasonic transmitter.

We bought two Jameco ultrasonic Rx/Tx pairs and planned to use one of the transmitters as a receiver. The Rx/Tx pairs were sharply resonant at 40 kHz. This was well suited for use with our Atmel ATmega644 microcontroller, which could easily generate the necessary signal to drive the pairs. The pairs had a -6-dB point for a 60° beam angle, which was sufficiently spread for the project.

We needed to drive the ultrasonic transmitter with a powerful signal to produce the strongest response possible on the receive side. We used a Microchip Technology TC4428 MOSFET driver, which provided a simple and effective design. We used a single driver chip with both inverting and noninverting internal drivers because we were generating a square wave and because the microcontroller’s 40-kHz output (0–5 V) was perfectly suited for logic lows and highs. The noninverting driver would output V_{CC} if it saw a voltage greater than 0.8 V on the input and ground otherwise. The inverting driver swapped this mapping. Setting V_{CC} to 9 V (so the device could be operated using a 9-V battery) and running our 40-kHz output from the ATmega644 to both the inverting and noninverting inputs, we could generate an 18-V swing from a single-power source.

On the receive side, we found the Texas Instruments INA129 instrumentation amplifier was ideal due to its larger operating voltage (± 18 V), fast slew rate (4 V/ μ s), high single-stage gain (10,000), and high common-mode rejection ratio (CMRR), which was especially important given that the receive and transmit circuitry was co-located.

Each inamp’s output was wired up to a respective pin on the microcontroller. This gain gave our received signal a peak-to-peak swing of approximately 6 V, which was ideal for triggering external interrupts on the ATmega644. [Photo 3](#) shows our final soldered implementation of the analog circuitry.

SOFTWARE

We implemented the software across two platforms: the ATmega644 and MATLAB. The code for the microcontroller operated within a `while(1)`

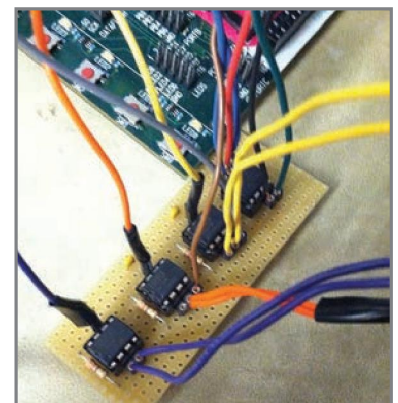


Photo 3—This is the soldered Rx/Tx circuit implementation.

loop after some basic initialization procedures. Initialization procedures included setting up UART communication (we used 38,400 bps to speed up communication), configuring I/O pins, initializing timers, setting up interrupts, and initializing variables.

The code required us to use all three ATmega644 timers. We used Timer0 to establish a 0.5-ms time base from which we could dispatch various actions. Timer1 was used as a means to calculate the delay in the cycles needed for a pulse to be detected. Because we wanted to time as accurately as possible, Timer1 was counting at the microcontroller's full clock speed (16 MHz). We utilized Timer2 to generate the 40-kHz square wave required by the ultrasonic Tx/Rx pair.

We wrote five interrupts for the code operation. The first was the Timer1 overflow interrupt. This interrupt was triggered every time Timer1 (an 8-bit counter) overflowed. Because 8 bits was not nearly enough space to represent a typical delay time in cycles (they were generally about 10,000 cycles), we had the overflow interrupt increment a variable by 256 every function call so we could track the true cycle count. The second interrupt ran whenever we received a new character on the serial port. We used this to tell the program MATLAB had sent a command and the ATmega644 would need to respond. The other three interrupts were hardware interrupts on `int0`, `int1`, and `int2`. Each ultrasonic Rx was wired up to a given pin after gain, so the interrupt would fire when the pulse was received.

The `while(1)` loop first updated the pen button's state, which was fully debounced, every 20 ms. Next, the code checked whether or not it was time to emit a new pulse. We set the inter-pulse period to 20 ms to ensure any reflections from previous pulses would have plenty of time to die out before we began a new measurement. We also only emitted a pulse for 0.5 ms. This was all that was necessary to properly trigger the receivers, and limiting the pulse length helped keep the environment quiet for each pulse. The loop also checked for a potential timeout. If all three receivers had not been triggered within 7.5 ms, the system would reinitialize everything, throw out the bad data, and emit another

pulse. This guaranteed that any captured data came from one unique pulse.

The loop also checked whether or not all three receivers had triggered. If they had, it entered them into the signal-processing buffer. We converged on a two-stage processing scheme for this project comprising a sliding window median filter and a sliding window mean filter. The median filter was a crucial first step because there could be very large outliers compared to the true delay, and a simple averager would have enabled these outliers to have a drastic effect on processed data. To implement the filter, we created an index to an array that would cyclically drop samples into the buffer. This made it so we didn't need to move any values around in the array when a new sample came in. Next, we needed to sort the data, which was accomplished via the C standard library function `qsort`. Because we needed to maintain the buffer order for proper windowing, we had to copy the data to a temporary array, sort it, then extract the median by simply looking at the middle array index. We found a median length of five produced excellent stability and responsiveness.

Next, the median filter outputs were fed into a sliding-window averager. The mean filter cyclically buffered new values in exactly the same fashion as the median filter. Whenever a new sample arrived, the code would sum the array values and divide out by the array length. This final value was what was then sent up to MATLAB in the UART data packet. We found a mean length of 10 again produced excellent stability and responsiveness.

The last thing the loop checked for was whether or not MATLAB had sent a flag indicating that it was ready for a new sample. The protocol we implemented dictated that MATLAB would send "y" if it was prepared for a new packet. This is the way MATLAB is best suited to receive data. Simply pouring packets onto the UART was neither appropriate nor reliable.

We used the stock GUIDE GUI development environment to create a GUI in MATLAB. It enabled us to graphically draw out where we wanted each component and then write backend code to support it.

The GUI drew the x, y, z coordinate data in six separate ways. "3-D Space" showed a 3-D projection of the data (here the "camera" mode would enable the user to use the pen to control the drawing's viewing angle). "Stereoscopic" showed two plots (one for the left eye and one for the right eye) slightly offset in azimuth so a human could cross his eyes and see the drawing pop out (similar to a Magic Eye stereogram). "Anaglyph" showed two overlaid projections of the drawing in red and blue for use with 3-D glasses. Lastly, "Orthogonal Projections" showed three 2-D projections of the drawing. Additionally, the MATLAB GUI enabled the artist to select the brush color, size, type, pen mode, background color, and display mode.

MATLAB initialized by setting up the serial object to talk with the ATmega644, creating various global variables that are passed around the GUI (e.g., brush color), and configuring various particulars about the GUI (e.g., plot color and axis size). It also created an important timer object. We set up how often we wanted it to execute (every 50 ms in our case) among other parameters and programmed a callback function. The vast majority of our code resided within this timer callback function.

The callback first sent a "y" to the ATmega644 to let it know it wanted data. Then it waited to receive the data. When we received the packet, we unpacked it by assigning the three delays and a button state to variables used within MATLAB.

When we read in the cycle delays, we subtracted a constant number of delays to make up for the INA129's rise time. MATLAB would next process the data from cycle delays into distance in meters. If the program was in Paint mode, it would buffer the distance into a buffer specifically for Paint mode. Because MATLAB pulled values from the ATmega644 at a rate not equal to the rate they were produced on the microcontroller, breaks in the data were reintroduced. To mitigate this, we introduced a small sliding window mean filter exactly like the one on the ATmega644. In the case of Paint mode, we used a three-length window so it wouldn't detract from responsiveness. Next, we used the prior equations to trilaterate the filter

APEC® 2013

March 17–21, 2013

Long Beach Convention Center

Long Beach, CA

THE PREMIER
GLOBAL EVENT
IN POWER
ELECTRONICS™

Visit the APEC 2013
web site for the latest
information!

www.apec-conf.org

SPONSORED BY



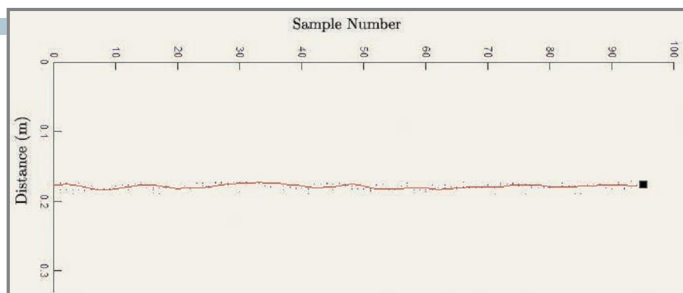


Figure 3—This plot shows the single-receiver accuracy.

delays into x , y , z coordinates, which produced a linear interpolation between a point and its predecessor.

One of the MATLAB program's critical features was to display the current pen position cursor on all the plots. Otherwise, the artist wouldn't know where the pen was relative to the rest of the drawing before making a new stroke. Thus, at this point in the timer function execution MATLAB updated all the cursor values. Next, if the button associated with a given packet was pressed, all the plots were updated with the new data point.

If the program was instead in Camera mode, then the incoming distance measurements were placed in a separate mean-filtering buffer. The reason for separate buffers was that the Camera mode could afford a bit more delay but had to be much more stable compared to the Draw mode. In Camera mode, the x , y , z coordinates were simply used to specify the camera's location. (The camera always points at the central point in the design space.)

Each of the GUI's toggle buttons and drop-down menus had a corresponding callback function. In each callback function, the appropriate global variable was set by whatever the artist selected for use elsewhere in the program.

The biggest upside to using MATLAB was the extensive infrastructure it already contained for producing output (e.g., 3-D plots, exporting images, rotating plots, etc.). The biggest downside was that this same extensive overhead was not designed for fast real-time use. It didn't run quite as efficiently as we had liked, and further iterations of this design would likely include rebuilding the PC program in a more flexible language for real-time applications.

OPERATION & IMPROVEMENTS

The device operated well within a reasonable design space. We limited the axes length (and thus implied operational range) to roughly 1 m. Within this region, our raw data was fairly accurate, so, post filtering, we saw good results. [Figure 3](#) shows the pen held at a constant distance from a single receiver. The raw data points are plotted along with a filtered curve.

For this test, we held the pen 17.42 cm from the transmitter. MATLAB reported a 17.45-cm mean-filtered distance, which was an acceptable error for our purposes. As witnessed from the plot, the data deviated very little from this average.

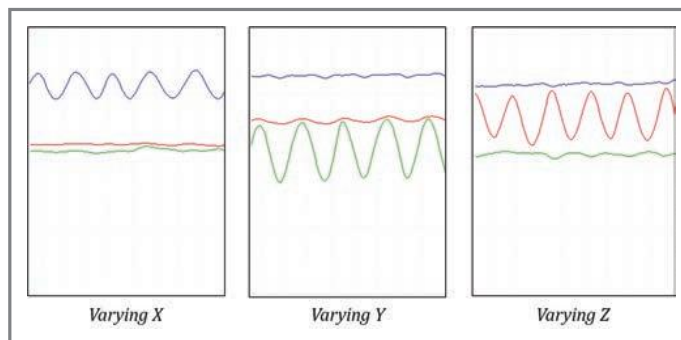


Figure 4—These x , y , and z variations show the orthogonal coordinate isolation achieved.

The hallmark test for proper functionality was watching how our trilateration equations interpreted pen movements in real time. Initially, we faced a large amount of coupling between the x , y , and z coordinates, with each coordinate unexpectedly following when another was varied. We corrected for this by manually calibrating the aforementioned slew offset.


[Figure 4](#) shows the isolation quality we managed to achieve. For each panel, the pen was displaced in a different orthogonal direction and, as expected, only one coordinate sinusoidally moved with the pen input in each.

3-D Paint is currently able to execute with a 0.5-s time lag between the artist moving the pen and the stroke being portrayed in the MATLAB GUI. The main bottleneck occurs in the MATLAB code, as MATLAB is not well suited for real-time operation. Fortunately, shifting some of the signal processing onto the ATmega644 helped to incrementally improve the speed.

We were satisfied with how our implementation turned out. During final testing and calibration, we felt that the system worked well enough to expose our poor artistic skills, which represented the true bottleneck in terms of the quality of images produced. Using supports to direct the pen, we found that highly accurate information was captured and displayed by the system. Furthermore, the entire design was only \$47.24.

To improve our current design, we would like to implement the oscillator on the pen instead of having it be driven from the microcontroller unit. This, together with Bluetooth communication, would enable us to make the device completely wireless, thus enhancing artist freedom and mobility. Finally, we would not use MATLAB to process information and would

instead generate a graphical artist interface in a more flexible language. MATLAB does not work well for real-time operation. This resulted in a noticeable time lag in our current device's operation.

We would like to see what a genuine artist could do with this type of device. We say this not only because we are interested in seeing how finely the system can interpret someone with artistic mastery, but also because we are curious as to how someone could exploit a 3-D canvas to create new kinds of drawings, hopefully much more sophisticated than the example shown in [Figure 5](#). 

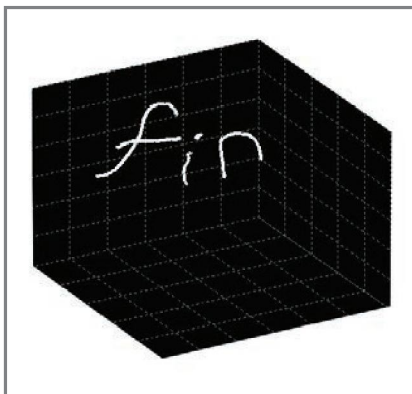


Figure 5—You can use the 3-D device to create this type of drawing.

William Myers (wgm37@cornell.edu) graduated summa cum laude from Cornell University's ECE program in May 2012. He is a Financial Software Developer at Bloomberg in NYC. In true ECE spirit, he has reveled in the medley of topics the major embodied by indulging in everything from lasers to MEMS and Assembly to jQuery. When he is not sinking his teeth into a Tolstoy novel or reading an esoteric typography blog, you could find him running along the East River or rock climbing in a converted Brooklyn newspaper warehouse.

Guo Jie Chin (chinguojie@gmail.com) is a senior majoring in ECE and Economics at Cornell University. Having interned at a couple of banks, he will probably be working at one next year. He dreams of one day bringing the benefits of finance and technology to the developing world. In his free time, he enjoys sleeping, drawing, watching movies, and jogging through the wilderness in upstate New York.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2013/271.

RESOURCES

Craig's Area, "Atmel Ultrasonic Ranger," <http://craigsarea.com>.

K. Gluck and D. DeTomaso, "Ultramouse3D," ECE 4760 Designing with Microcontrollers Final Projects, Cornell

University School of Electrical and Computer Engineering, ECE 4760 Designing with Microcontrollers Final Projects, http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2009/kwg8_dmd54/kwg8_dmd54/index.html.

B. Land, "Using Graphics," BioNB 441, Cornell University, <https://instruct1.cit.cornell.edu/courses/bionb441/Graphics>.

——, "Debounce Demo," Cornell University, <http://people.ece.cornell.edu/land/courses/ece4760/labs/s2012/lab2/code/debounceGCC644.c>.

M. Lindner, "From 3-D Plot (Stereoscopy)," MATLAB Central, File Exchange, MathWorks, Inc., www.mathworks.com/matlabcentral/fileexchange/27283-3d-plot-stereoscopy/content/stereo_plot.m.

SOURCES

ATmega644 Microcontroller
Atmel Corp. | www.atmel.com

MATLAB
MathWorks, Inc. | www.mathworks.com

TC4428 MOSFET Driver
Microchip Technology, Inc. | www.microchip.com

INA129 Instrumentation amplifier
Texas Instruments, Inc. | www.ti.com



PCB-POOL
Beta LAYOUT

THE ORIGINAL SINCE 1998

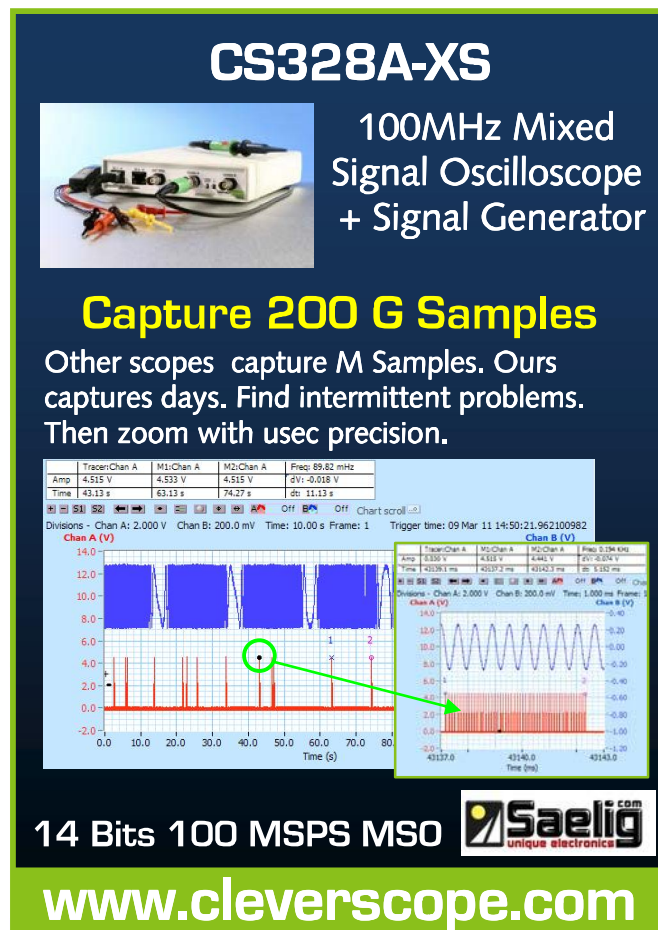
FREE Stencil
with every prototype order

EAGLE order button
pcb-pool.com/download-button
20% off! on your first order

Call Tyler: 1 707 447 7744
sales@pcb-pool.us

PCB-POOL® is a registered trademark of
Beta LAYOUT

www.pcb-pool.com



CS328A-XS
100MHz Mixed
Signal Oscilloscope
+ Signal Generator

Capture 200 G Samples
Other scopes capture M Samples. Ours captures days. Find intermittent problems. Then zoom with usec precision.

Trace/Chan A	M1:Chan A	M2:Chan A	Freq: 69.82 mHz
Amp: 4.515 V	4.533 V	4.515 V	dV: -0.018 V
Time: 43.13 s	63.13 s	74.27 s	dt: 11.13 s

Divisions: Chan A: 2.000 V Chan B: 200.0 mV Time: 10.00 s Frame: 1 Trigger time: 09 Mar 11 14:50:21.962100982

14 Bits 100 MSPS MSO

Saelig
unique electronics

www.cleverscope.com

Digital Camera Controller (Part 3)

Build a Generic Front-Panel Board

The first parts of this article series introduced the microcontroller-based Photo-Pal and described its hardware and construction as well as its code, user interface, and timing. Many elements of the Photo-Pal's design can be used to create a generic front-panel board for use with numerous other applications.

This article series describes the Photo-Pal, which is a microcontroller-based electronic flash-trigger camera controller. The second part of this article series, "Digital Camera Controller (Part 2): Code, User Interface, and Timing," (*Circuit Cellar* 268, 2012), described how an 8-bit microcontroller and a two-line character display can be used to implement a versatile user interface with software-defined push-button keys. As the Photo-Pal camera controller design evolved, it became apparent that this user-interface concept and associated hardware could be applied to many other projects. In some applications, such as Photo-Pal, this user-interface core would have enough available I/O pins and leftover microprocessor power that the entire project could be implemented with the addition of only a few simple parts. For other projects, this implementation could become a front-panel human interface that could communicate with other microcontrollers and computers or with a busy DSP that couldn't handle the extra interface overhead.

With this more generic concept in mind, I decided to design a PCB that would implement a fully functional user interface. My goal was to simplify construction of future projects by limiting the hand-wired portion to the components specific to that project.

I had used hand-wired boards mounted in a Hammond Manufacturing standard plastic case to build several battery-operated devices, including the Photo-Pal.

This pocket-size case includes a battery compartment and is big enough for a standard 2 × 16 character display and a surrounding circuit board. It is also deep enough to hold a second board with the application-specific hardware. For several of my future projects, it made sense to design a generic controller circuit board that would easily fit into this plastic case along with the display.

I designed an "L" shaped circuit board that wraps around the bottom and right edges of the display so the four software-defined push buttons are placed below the display's bottom edge with the "Increase" and "Decrease" push buttons at the display's right edge (see [Photo 1a](#)). The board also includes three LEDs, the power switch, a 5-V regulator, a 16-pin application connector for the power, and I/O connections to the application board. It is designed to mount directly onto the mounting bosses of the Hammond plastic case. Photo 1b shows the board's back side.

MICROPROCESSOR DECISIONS

When I built the original Photo-Pal controller, I reached into the microcontroller drawer of my existing parts collection and grabbed a Microchip Technology PIC16F873A CMOS flash-based microcontroller. This microcontroller had the functionality I needed, but it turned out the 4,096 instruction limit provided barely enough code space. Microchip has now upgraded the PIC16F87x family to the newer pin-compatible PIC16F88x family, which offers lower power consumption and several new features, including the ability to use an internal 8-MHz oscillator. For the generic controller, it made more sense to use the PIC16F886 part that has the same features as the PIC16F883

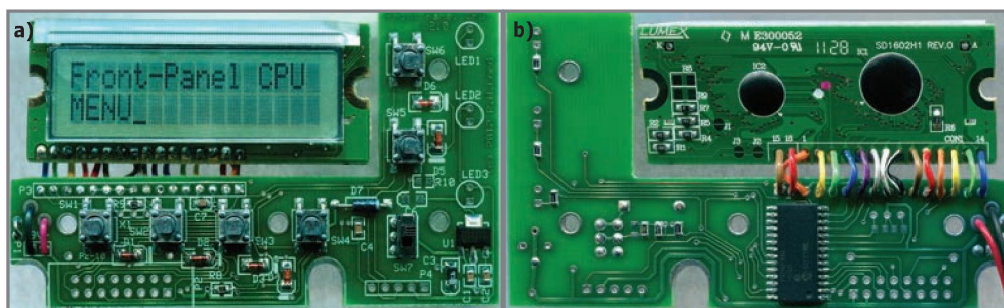


Photo 1—The circuit board wraps around the display's bottom and right edges. **a**—Four push buttons are located below the display's bottom edge. The "Increase" and "Decrease" push buttons are at the display's right edge. **b**—This is the board's back side.

but has twice as much code space (8,192 instructions).

PIN CONFIGURATION

One of the difficulties I encountered was determining which pins to commit to the front-panel functionality and which pins would be made available for the application it would control. Each PIC16F886 I/O port pin can be configured to provide additional dedicated functions (e.g., an analog input or a serial data output). Various applications might need to use one or more of these special pin functions, so hard wiring a pin to the display meant the other specialized functions wouldn't be available for the application's use.

In the original Photo-Pal—which I described in "Digital Camera Controller (Part 1): Hardware and Construction" (*Circuit Cellar* 267, 2012)—I chose the microcontroller pins with the idea of organizing the Photo-Pal signals into logical function groups that would simplify programming. I didn't worry about other potential functions a pin might be used for in other applications. For the generic front-panel design, the basic starting design was derived from the schematic that appeared as Figure 3 in Part 1 of this series. Some pin assignments were rearranged to make better availability of the microcontroller's special features. A new application connector was added to provide access to the application I/O pins, replacing the connection that had served the switches and LEDs of the old two-board version.

If there had been enough extra room on the circuit board, each of the microcontroller's I/O pins could have gone through a jumper before reaching their destinations. This way, the wiring could be rearranged to gain access to different pin functions if needed for a particular application. However, with a tight board layout, this was not an option. For the generic front-panel board to be useful, I needed to carefully analyze what microcontroller features would be lost for each choice of an internally dedicated pin.

With the LCD configured to work using a 4-bit data bus, nine microcontroller pins are needed to implement the front panel's essential circuitry. These are the four data lines (DB4–DB7) and three control lines (E, R/W, and RS) of the LCD and two lines (SW1:4-enable and

SW5:6-enable) needed to read the six front-panel push buttons. In addition, there are three optional LEDs and an optional analog input used to measure the battery voltage. If all the options are included, 13 of the microcontroller's port pins will be dedicated to the board's own hardware and are thus unavailable for the user's application. The question becomes which of the pins to use for the dedicated board hardware and which to provide for the user application.

In the original Photo-Pal design, it seemed logical to keep the four LCD data lines (DB4–DB7) grouped together in either the upper or the lower four pins of the same port. I was using the comparator inputs at RA2–RA3 for the trigger circuitry and RA6–RA7 were potentially unavailable, so that ruled out Port-A. Five pins in Port-C (RC3–RC7) were where the USART, SPI, and I²C capabilities were made available, so that port was eliminated. In Port-B, pin RB0 was dedicated to providing a hardware-activated interrupt that could be useful, so RB4–RB7 seemed like the only reasonable choice for the LCD data lines.

My first attempt at a PCB followed the original Photo-Pal design a little too closely, using RC3 and RC4 as the low-tru enables for reading the push-button switch banks. Unfortunately, I realized that this prevented the use of the SPI and I²C ports by a user application after I had produced the first PCBs. This was a serious oversight, because many applications would make extensive use of these interfaces. While there were a few projects where I could use these boards, they didn't meet the criteria for a more generic design. I needed a better way to help me decide which pins to dedicate to the on-board function.

PIN SELECTION

Table 1 is a worksheet I developed to help me select which pins to use. The upper part of the worksheet is a table of the PIC16F886 microcontroller's 24 available I/O pins grouped into the three I/O ports shown in the first column with the functions provided at each pin shown in the second column. The third column is to be filled in with the tentative assignments of the pins to the signals needed for the dedicated on-board functions. In the fourth column, the remaining pins are

then assigned to the application connector. The earlier PIC16F87x family used pins XTAL1 and XTAL2 for the crystal or ceramic resonator that was needed to supply the microcontroller's operating clock. The newer PIC6F88x family offers the option of an internal 8-MHz oscillator, making one or both of these pins potentially available as port RA6 and RA7. However, for some applications, a more stable clock source may be desirable, so it is a good idea not to assume that they are always going to be used for I/O.

As this worksheet evolved, I also added function tables to the bottom of the worksheet that list each of the categories of special I/O functionality the PIC16F886 supports and which I/O pins are associated with various configurations of that feature. Note that the pin usage depends on the particular configuration being supported. For example, if you want to use a single analog comparator with an internally generated reference, then only RA3 (C1IN+) or RA2 (C2IN+) would be used as an analog input and the other pins would be available for other uses. However, if you want access to all inputs and outputs of both comparators, pins RA0–RA5 would all be dedicated to the internal comparators' operation and unavailable for any other use in that configuration. (Refer to the PIC16F88x family datasheet for the details of each special function and the I/O pins used by the various configurations of each special function.) As the worksheet's upper table is used to tentatively assign port pins to the front panel's dedicated hardware, shading in the respective port pins as they appear in the function tables at the worksheet's bottom helps reveal which microprocessor functions are made unavailable to the application as a consequence.

Using an early version of the pin-assignment worksheet, I decided to move the two push button read-enable outputs over to RA4–RA5. I reasoned that RA6–RA7 might need to be used to support an external clock source. It also seemed logical to conclude that most common applications using the microcontroller's comparators wouldn't need external access to the outputs via the C1out and C2out configuration. The comparator outputs are internally available to be read from a hardware register and can be used to generate an interrupt. If I needed the comparator

PIC16F886 Microcontroller		Generic Front-Panel Board		Photo-Pal	
Port	Function	Dedicated pins	Application pins	As an app to generic FP	
RA0	RA0/AN0/ULPWU/C12in0-	(V _{BATT})	(User P2 – Pin 4)	V _{BATT}	
RA1	RA1/AN1/C12in1-		User P2 – Pin 5	(not connected)	
RA2	RA2/AN2/V _{REF-} /CV _{REF} /C2in+		User P2 – Pin 6	TRIG_SWITCH	
RA3	RA3/AN3/V _{REF+} /C1in+		User P2 – Pin 7	SOUND_TRIG	
RA4	RA4/T0CKI/C1out	SW1:4 –enable	(XXXX)	SW1:4 -enable	
RA5	RA5/AN4-SS/C2out	SW5:6 –enable	(XXXX)	SW5:6 -enable	
RA6	RA6/OSC2/CLKout	(xtal2)	(jumper option)	(XTAL2)	
RA7	RA7/OSC1/CLKIN	(xtal1)	(jumper option)	(XTAL1)	
RB0	RB0/AN12/INT		User P2 – Pin 8	ARM_SW_in	
RB1	RB1/AN10/P1C/C12in3-	(–LED1/LCD contrast)	(jumper option)	Trig-LED	
RB2	RB2/AN8/P1B	(–LED2)	(jumper option)	Shutter-LED	
RB3	RB3/AN9/PGM/C12in2-	(–LED3)	(User P2 – Pin 11)	Flash-LED	
RB4	RB4/AN11/P1D	LCD-DB4/SW4	(XXXX)	LCD-DB4/SW4	
RB5	RB5/AN13-–T1G	LCD-DB5/SW3	(XXXX)	LCD-DB5/SW3	
RB6	RB6/ICSPCLK	LCD-DB6/SW2:6/PGC	(XXXX)	LCD-DB6/SW2:6/PGC	
RB7	RB7/ICSPDAT	LCD-DB7/SW1:5/PGD	(XXXX)	LCD-DB7/SW1:5/PGD	
RC0	RC0/T1OSO/T1CKI	LCD-E	(XXXX)	LCD-E	
RC1	RC1/T1OSI/CCP2/PWM2	LCD-RW	(XXXX)	LCD-RW	
RC2	RC2/P1A/CCP1/PWM1	LCD-RS	(XXXX)	LCD-RS	
RC3	RC3/SCK/SCL		User P2 – Pin 12	MIC-PWR-out	
RC4	RC4/SDI/SDA		User P2 – Pin 13	SHUTTER-0-out	
RC5	RC5/SDO		User P2 – Pin 14	SHUTTER-1-out	
RC6	RC6/Tx/CK		User P2 – Pin 15	FLASH-0-out	
RC7	RC7/Rx/DT		User P2 – Pin 16	FLASH-1-out	
Analog Comparator		Timer 0		USART	
RA0	C12in0-	RA4	T0CKI	RC6	TX/CK
RA1	C12in1-			RC7	RX/DT
RA2	CV _{REF} /C2in+	Timer 1			
RA3	C1in+	RB5	–T1G	SPI	
RA4	C1out	RC0	T1OSO/T1CKI	RC3	SCK
RA5	C2out	RC1	T1OSI	RC4	SDI
RB1	C12in3-	CCP		RC5	SDO
RB3	C12in2-	RC2	CCP1	RA5	–SS
		RC1	CCP2		
ADC		PWM		I ² C	
RA0	AN0	RC1	PWM2	RC3	SCL
RA1	AN1	RC2	PWM1/P1A	RC4	SDA
RA2	AN2/V _{REF-}	RB2	P1B		
RA3	AN3/V _{REF+}	RB1	P1C		
RA5	AN4	RB4	P1D		
RB2	AN8				
RB3	AN9				
RB1	AN10				
RB4	AN11				
RB0	AN12				
RB5	AN13				
Key					
				Available for user application	
				Optionally available	
				Dedicated to front panel	

Table 1—This is the front-panel controller board's microcontroller pin-out worksheet.

outputs, I'd be better off using an external comparator in the application hardware. Using RA4 for the front-panel switches also meant I couldn't externally clock Timer T0, but I didn't see this as a major

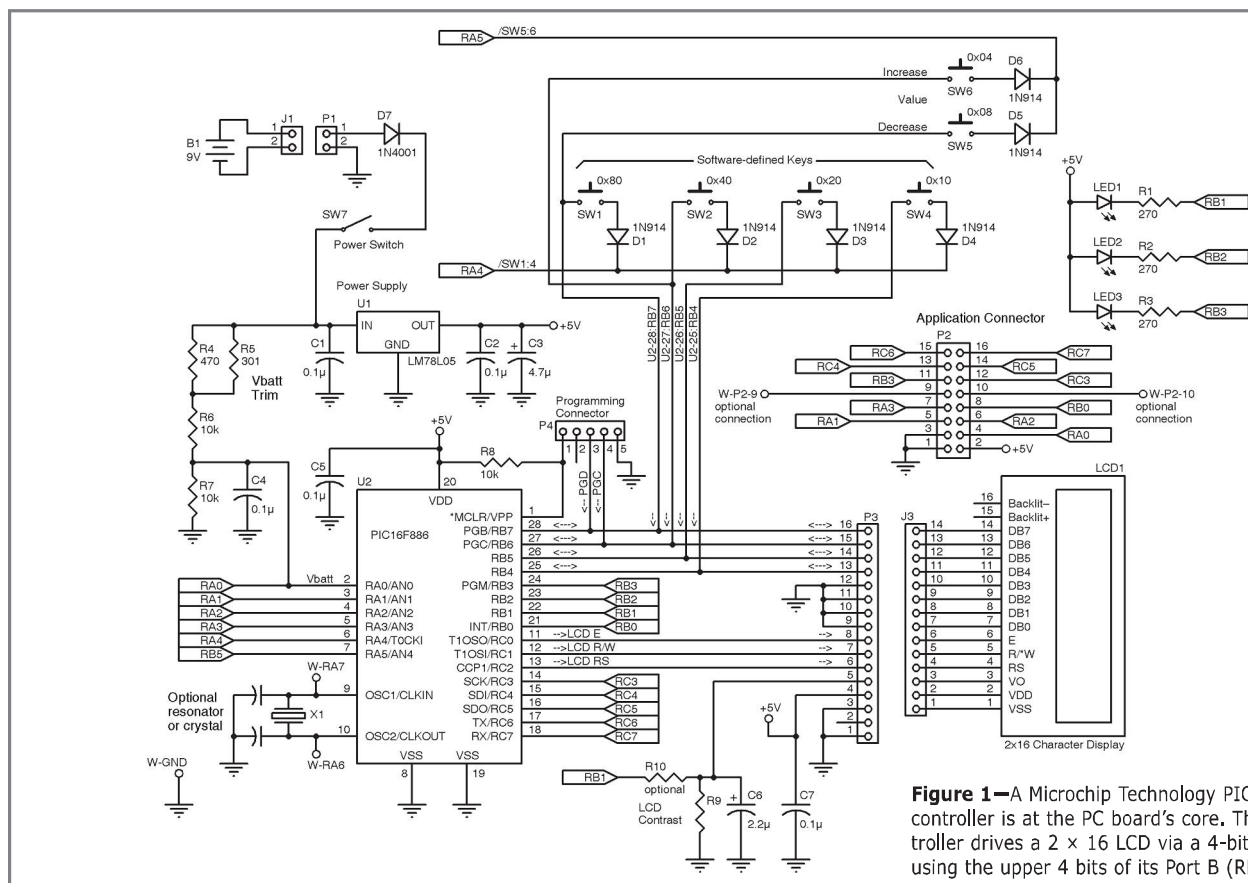
issue since T0 was already being used to provide the timing for the front-panel code. The remaining pin, RA5, also can be the "Slave Select" (-SS) input for the SPI function, but I felt I could live without this

fairly unusual configuration.

One of the interesting things about writing an article is that, to explain the design to the reader, you have to dig a bit deeper into how you arrived at that way of doing things. I put the LCD control lines on Port-C because the LCD data lines (RB4-RB7) serve as both inputs and outputs to be able to both read from and write to the display. In my earlier experience with the PIC16F87x family, after much frustration I finally figured out that the read-modify-write instructions (e.g., bit-set and bit-clear) didn't always work well when applied to I/O ports, where the read was of the pins' physical state, not the data that was in the port's output latch. I learned the hard way that the output latch was disabled when the port pin was being used for something other than an output. Consequently, the bit manipulation instructions were not reading from what they would write to. The evolving workaround for this was first to move the control lines to Port-C and to later have the bit-set or bit-clear instruction modify a software "shadow register," which was then copied into the port's output latch. However, the earlier initial experience had made me cautious about sharing the bi-directional data lines with the LCD's control lines in the same port, so I didn't consider whether the use of a Port-B shadow register would have enabled me to put the entire LCD on Port-B.

My head-slapping moment came from redrawing my pencil sketch of the map for using I/O pins. Filling in the function tables beneath the port assignment table in Table 1 while writing this article, I suddenly realized the legacy-based decision to put the three LCD control signals (E, R/W, and RS) on RC0-RC2 was not as good a choice as I had previously thought. Looking more

closely at the function tables made me see that the dedicated use of these pins would completely prevent the use of the microcontroller's Capture-Compare feature and would limit the usefulness of Timer T1



and the PWM output. If I instead had placed the control signals on RB1–RB3 and moved the optional LED outputs to RC0–RC2, then I would have been able to make full use of the features available on Port-C by selectively removing one or more LEDs from the circuit board or by moving its current-limiting resistor to connect it to another I/O pin. However, by the time I realized this, I had already committed the design to a second PC board using the schematic shown in [Figure 1](#). The design description and operation is essentially the same as Figure 3 in Part 1 of this series. The significant difference is the addition of connector P2, which supplies power and ground and connections to 11 of the microcontroller pins to the user's application. An additional two pins of the application connector are connected to jumper pads that can be optionally connected to RA6–RA7 or RB1–RB2.

A CONTRASTING DECISION

One of the annoyances of using an LCD is that the display's contrast is often not ideal. It can be set to a fixed value by grounding the LCD's V_{EE} pin through a fixed resistor, but the contrast varies with the temperature and the viewing angle, so the resistor value is selected to achieve a workable compromise. Supplementing the fixed resistor with a potentiometer can give you control of the contrast, but it isn't always easy to provide this control on a compact device. While choosing I/O pins for the PC board, it occurred to me that it should be possible to make use of the PWM feature built into the microcontroller to create a variable voltage output that could be applied to the V_{EE} pin through a resistor divider. This option is available by adding resistor R10 to the board.

PCB LAYOUT

The front-panel board's size and shape were dictated by the available space in the plastic case I had selected and the need to share that space with the display. I also wanted to use the existing mounting bosses in the case. In looking at the board's shape and the location of the four mounting holes, it became clear that I would need to use surface-mount parts to have enough room for the microcontroller. Through-hole push buttons would have used up any area that could have been available for the microcontroller, which also had to be surface mounted to ensure there was room for the push buttons. Experience had taught me that if I worked carefully, I could solder the 0.05" pitch pins of the microcontroller's SOIC-28 version. I could also manage 0806-resistor size parts. I concluded that anything much smaller was beyond my skill.

Next was the board layout issue. Now that I am retired, I don't have access to expensive schematic capture and layout tools at work. However, I had noticed one of the advertisers in *Circuit Cellar* offers free downloads of schematic and layout tools that can be used for two-layer boards. The "gotcha" is that the tools produce proprietary output files that can only be used to buy your PC boards from that company. However, for small quantities of boards for trying out ideas, its board prices are reasonable. If you come up with something really good you later want to mass produce, you might have to do the layout all over again to get competitive pricing.

I downloaded and installed the tools and found them surprisingly intuitive, quick to learn, and easy to use. There's no "automatic routing" of the PC traces, but I go back to the days of using red and blue tape and black component pads stuck to acetate

sheets to layout boards, so I'm willing to do some of the work myself for relatively simple board layouts. Without too much difficulty, I created a board layout that would work for the generic front panel. Since the L-shaped board would have left a big area that would have been wasted, I filled the unused area with the layout of another board for a different project, but that's a tale for another time. Suffice it to say, I didn't waste the space.

I received the three promised circuit boards a few days after placing the order. After carefully sawing away the other "filler" board and routing out the notches needed for the front-panel board to fit into the plastic case, I soldered in the components, downloaded the front-panel program, and was pleased to see the board worked as intended.

MORE PROCESSING HORSEPOWER

The PIC16F886 I chose for the generic front panel is adequate for most applications, but there are situations where it would be nice to have more capability. The internal clock speed is 8 MHz (though it can be externally clocked to 20 MHz) and the hardware stack is only eight levels deep. Microchip offers another family of 8-bit microcontrollers (the PIC18x series) with a more complex instruction set and a number of enhancements over the basic PIC16x parts. These include a deeper return stack (31 levels versus eight), PUSH and POP instructions, table read and write instructions that make it easier to handle look-up tables and text strings, an 8 × 8 bit hardware multiply, a two-level interrupt structure, and an additional 16-bit timer. The internal clock can be phase locked to 32 MHz providing four times the performance of the PIC16F886. For those who only program in higher-level languages, the larger stack and more flexible instruction set make the PIC18x family more suitable for C-programming.

Fortunately, there are several parts in the PIC18F2x family that have the same pin configuration as the PIC16F886 and can replace it on the front-panel board without any hardware modifications. The high-end of the pin-compatible parts is the PIC18F2523, which has 16,384 instructions of code space, 1,536 bytes of RAM, and a 12-bit ADC. It is only a couple of dollars more than the PIC16F886, so one might ask why bother with the less capable part in the first place? The simplest answer is legacy. If you already have a lot of PIC16x code to draw from, you might want to stay with what you have been using. However, it is nice to know the generic board can be upgraded to a much more powerful part.

CO-PROCESSING

Perhaps, even the PIC18F2523's availability on the generic front-panel board still isn't sufficient to meet your needs. Maybe you need 30 I/O pins instead of 10, or perhaps your application calls for a blazingly fast DSP. The generic front-panel board can easily communicate with another microcontroller by using either the microcontroller's TX and RX pins on RC6–RC7 with the internal USART function or by using the I²C function on pins RC3–RC5. Microcontrollers and DSPs have become so inexpensive that it is easy to add a coprocessor to the application board for very little extra cost. A 40-pin microcontroller can provide a lot of additional I/O and a DSP can do a lot of signal processing without being burdened by the user interface task that is being handled by the front-panel controller.

USING THE GENERIC FRONT-PANEL BOARD

The generic front-panel board has been designed to provide a simple, easy-to-use human interface with software-defined keys and a microcontroller that can also implement the user's code for many applications. With a dozen I/O pins that can also be used for up to eight analog inputs to the internal ADC, access to two fast analog comparators, two PWM outputs, and a SPI port available on the application connector, the front-panel board can provide most of the functionality needed for many applications with very little additional hardware. The on-board USART can be used with parts such as the Microchip MCP2200 USB-to-UART serial converter to provide USB connectivity. With the addition of an RS-232 or RS-485 interface, the board can become the remote interface for any device with which it can communicate. When even more functionality is needed, the USART and I²C ports can be used to communicate with a coprocessor. By removing the voltage regulator, the board can be powered through the application connector from the USB port or through the connection from a host computer.

LEARNING THROUGH DESIGN

This series of articles has shown how a design for a specific application can lead to a better understanding of a how to use a simple 8-bit microcontroller to implement a powerful user interface by incorporating software-defined push buttons with a simple character display, as described in Part 2 of the series. That has led to the design of a generic front-panel board that can be used for many other applications. Along the way, I learned how to implement simple timers, auto-repeat keys, and a parameter entry method that easily handles both large and small numbers. I explored the ins and outs of menu-driven interfaces and came up with tools (e.g., the worksheet) to assign functions to I/O pins. Hopefully, you've been inspired to apply some of these ideas to your own projects. In the process, I've learned a lot, not only by designing and building both Photo-Pal and the generic front-panel controller, but also by digging deeper into my thought processes to write about them. 📖

Richard Lord (rhlord@comcast.net) holds a BS in Electrical Engineering and an MS in Biomedical Engineering. During his career, he has designed digital electronics for an aerospace company and several telecommunication test equipment manufacturers. Working as a consultant in the 1980s, Richard designed several medical pulmonary test instruments and the electronics for an autonomous underwater robot.

RESOURCES

R. Lord, "Digital Camera Controller (Part 1): Hardware and Construction," *Circuit Cellar* 267, 2012.

——, "Digital Camera Controller (Part 2): Code, User Interface, and Timing," *Circuit Cellar* 268, 2012.

SOURCES

PIC16F873A, PIC18F2523, PIC16F87x, PIC16F88x, PIC16F886, and PIC18x Microcontrollers and MCP2200 USB-to-UART serial converter
Microchip Technology, Inc. | www.microchip.com



Concurrency in Embedded Systems (Part 5)

Designing Robust Systems with Linux

This is the fifth in a multi-part article series examining concurrency in embedded systems. This article discusses some other ways embedded Linux helps you design robust systems with concurrency.

Part 4 of this article series discussed Linux threads and processes. This article will focus on the first forms of inter-process communication (IPC): mutexs, semaphores, and shared memory. As I mentioned in my last article, remember, we are taking this in thin slices. The goal of this article series is to introduce you to the range of tools available in Linux to assist you in designing robust embedded systems with Linux. In my last article, I mentioned Michael Kerrisk's *The Linux Programming Interface*. This month, I want to connect you to the portable operating system interface (POSIX) standard webpage to find more about all types of IPC. (See the sidebar for more information about the POSIX standard for shared memory.)

SHARED MEMORY

In my last article, we looked at the memory model Linux uses and how memory is *not* shared between processes by design. Thus, one process cannot share data with another without some system mechanism. Linux provides three mechanisms for designating certain memory as shared and enables you to access that memory: the historical System V UNIX shared-memory model, shared file mapping supported by POSIX, and the POSIX shared-memory model. This article will examine the two POSIX mechanisms. Generally, we don't recommend using the legacy interfaces when a POSIX interface is available.

Functionally, these two mechanisms are very similar. Since POSIX uses the term "object" to refer to a memory region that can be shared between processes, for the purposes of this article, I will call files created using shared file mapping "nonvolatile shared-memory objects" and the standard shared-memory objects "volatile shared-memory objects."

Let's look at each of these two mechanisms.

VOLATILE VERSUS NONVOLATILE SHARED-MEMORY OBJECTS

In simple terms, both methods enable different processes to directly access shared memory without system calls (i.e., fast). The nonvolatile mechanism enables the memory to be "sticky" across power outages. When power is lost, a volatile shared-memory object's contents are lost. The caveat with nonvolatile objects for real-time embedded systems designers is the question of when the nonvolatile object's data is committed to the underlying file. The kernel will commit the data to file at its leisure. As a designer, POSIX enables us to commit on demand (using `msync()`), which is a mixed blessing. It frees the designer to access nonvolatile memory across processes but places the burden on the designer to know when to commit the mapped

WHAT IS POSIX?

We haven't talked about the portable operating system interface (POSIX) in this column. It is worthy of an article all by itself, but for now, suffice it to say that it defines an operating system (OS) application programming interface (API) that provides a standard interface for applications written across multiple OSes. The POSIX standard was formalized by the Institute of Electrical and Electronics Engineers (IEEE) and is also known as IEEE-STD 1003. Theoretically, you could write a POSIX-compliant application for Linux and switch to QNX or VxWorks with no change to your code. Let me know if you would like an article or two on POSIX.

memory to disk.

Unfortunately, the APIs for using these two mechanisms are not identical. In my opinion, there should be a flag that makes a shared-memory object nonvolatile so they can be interchangeably used by the designer. Very often, we design systems where certain data's nonvolatility requirements change over the product's life cycle. But these differences persist and we must work around them.

CONFIGURING VOLATILE SHARED-MEMORY OBJECTS

Shared-memory objects can be thought of as volatile files (or files kept on a RAM drive) keeping with the Linux virtual memory

paradigm. To configure a volatile shared-memory object, the programmer must perform three steps. The first step is to create a shared object much like you create a file. Open the object by name using the mode and flags just like you open a file. The object can have read-only or read-write access based on the flags. The POSIX system call `shm_open()` is used to perform this function. It returns a volatile shared-memory object handle. The second step is to use `ftruncate()` to set the object's size. Just like a file, the size can be later expanded or contracted. The final step is to use the `mmap()` system call to map all or part of the object into the process' virtual memory. Optionally, the file handle can be closed after the `mmap` call.

CONFIGURING NONVOLATILE SHARED-MEMORY OBJECTS

Similar steps are performed to create a nonvolatile shared-memory object. In this case, you can use the standard `open()` commands (instead of `shm_open`) to create a standard file. Then, like a volatile object, you use the `mmap()` system call to map all or part of the file's contents to a memory region. The `mmap` call sets the file size. As with the volatile object, the file can be closed at this point.

Once the objects are created using one of these two methods, both object types can be identically used by referencing them as memory. The objects look like any other memory object in the process' virtual address space.

PROS & CONS OF USING SHARED-MEMORY OBJECTS

Although shared memory is the fastest means of sharing memory across processes, there is still measurable overhead. As mentioned in my last article, we measured 50 μ s per access to a nonvolatile shared-memory object on one 600-MHz ARM9. If you are going to have significant shared memory between concurrent tasks, you would be better off doing this between threads and eliminating the overhead.

Also, remember, just because the OS provides this feature, it does not provide synchronized access to the shared region. You must use something like a semaphore or mutex to guarantee the data's integrity in these shared regions.

SEMAPHORES

As with shared memory, Linux provides both System V semaphores and POSIX semaphores. As before, we will examine only POSIX semaphores.

POSIX defines two types of semaphores: named and unnamed. Named semaphores are handled between processes with a `sem_open()` system call. Unnamed semaphores share a memory address. With processes, this can be accomplished with a volatile shared-memory object. With threads, this can be accomplished with a global variable. The methods of initializing and destroying these two types differ, but everything else is operationally the same.

HOW SEMAPHORES WORK

A semaphore is a simple integer that can be incremented and decremented but can never fall below zero. The `sem_post()` function increments while the `sem_wait()` function decrements

Listing 1—This is an example of semaphore code.

```
// globals
sem_t semi;
unsigned long long counter; /* shared object */

int main()
{
    pthread_t counter_thread;
    pthread_t reader_thread;

    sem_init(&semi, 0, 1); // semaphore is "named" semi
                          // semi is local
                          // initialize semi to 1

    pthread_create (&counter_thread,
                   NULL,
                   (void *) &incrementer,
                   NULL);
    pthread_create (&reader_thread,
                   NULL,
                   (void *) &reader,
                   NULL);

    while (1 == 1)
    {
        sleep (10);
    }
}

void reader ( void * )
{
    while (1 == 1)
    {
        sem_wait(&semi); // decrement
        // start of critical section
        printf("Current Counter = %llu\n", counter);
        // end of critical section
        sem_post(&semi); // increment
        sleep(1);
    }
}

void incrementer ( void * )
{
    while (1 == 1)
    {
        sem_wait(&semi); // decrement
        // start of critical section
        counter++;
        // end of critical section
        sem_post(&semi); // increment
        nanosleep(100000); // Sleep for 100 microseconds
    }
}
```


the semaphore. If the integer is greater than 0, the `sem_wait` immediately returns. If the integer is 0, the task will wait and relinquish control to the scheduler until the semaphore rises above zero, then it will decrement the integer.

USING SEMAPHORES IN CONCURRENT DESIGNS

Let's look at a simple example using unnamed semaphores to illustrate how you would use a semaphore to provide the necessary synchronization with shared-memory objects (in this case shared between threads).

Assume you have a 64-bit counter that is incremented by one thread, read by another, and stored in nonvolatile shared memory. Assume the underlying hardware only supports 32-bit integer arithmetic. (Note: We are not building this robust, we are not checking the `sem_init` or `pthread_create` returns to keep the example simple.)

The Linux C code segment in [Listing 1](#) shows how this works. The main process creates two threads and a semaphore and then goes to sleep. The counter thread increments a 64-bit counter. Remember, since this is a nonatomic operation, without the semaphore, it could be interrupted midstream and produce incoherent results in the reader thread. The reader thread merely prints the counter value to the standard output device.

With the semaphore in place, the reader thread will always print coherent data. If the reader attempts to print the counter while the counter is being incremented, the semaphore will be set to 0 and the reader thread will block at the `sem_wait()` until the incrementer function has updated the counter.

Two other nice enhancements for the embedded designer are the `sem_trywait()` and the `sem_timedwait()`, which do exactly what you'd expect them to do. With the `sem_timedwait`, you can protect your thread against an unruly thread without waiting forever for the semaphore. You can use the `sem_trywait()` to control the blocking.

MUTEXES

POSIX supports mutexes (which stands for mutual exclusivity), which is another mechanism for providing synchronization for concurrency. Like semaphores, mutexes can ensure atomic access to any shared resource. Unlike semaphores, which can have an integer number of values, a mutex, by default, is binary in nature and can only be locked or unlocked. Let's see how you create and use mutexes.

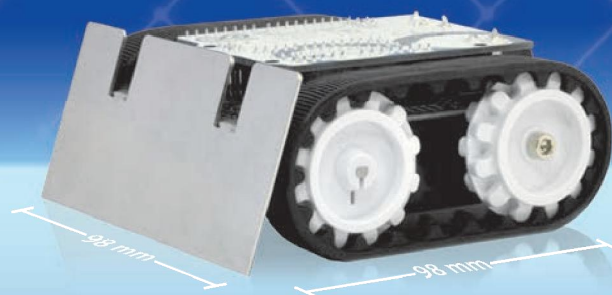
CREATING AND USING MUTEXES

A default-style mutex is simply made by creating a variable of type `pthread_mutex_t` and initializing it with `PTHREAD_MUTEX_INITIALIZER`. It can be as simple as:

```
pthread_mutex_t MyMutex = PTHREAD_MUTEX_INITIALIZER;
```

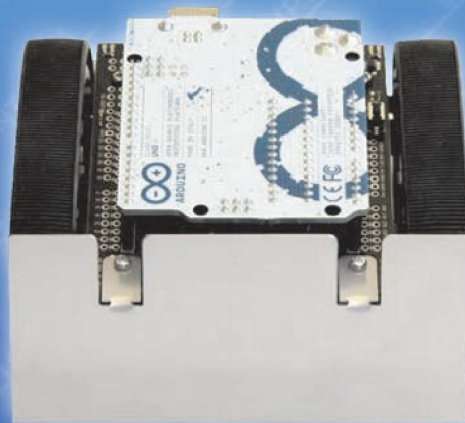
A mutex can then be locked (`pthread_mutex_lock`) and unlocked (`pthread_mutex_unlock`) to achieve the necessary synchronization. Hopefully, you can see that these could be used in the previous code example by replacing the `sem_wait` with the `pthread_mutex_lock` and the `sem_post` with the `pthread_mutex_unlock`.

Get a little pushy.



Zumo

Small enough for Mini-Sumo;
flexible enough to make it your own.



Put your Arduino or compatible controller on the right tracks with the Zumo chassis and Arduino shield! The Zumo is a small, tracked robot platform that works with a variety of micro metal gearmotors to allow for a customizable combination of torque and speed. Add a Zumo shield, which includes a dual motor driver, buzzer, and three-axis accelerometer and compass, to make an Arduino-controlled robot that can really throw its weight around!

Pololu

Robotics & Electronics

Learn more at www.pololu.com/zumo

Listing 2—When using a mutex, nested layers of critical code sections are unnecessary.

```
HighLevelFunction()
{
    // Do some work
    // Enter Critical Section with Semaphore
    //(sem_wait) or Mutex lock
    AccessSharedResource();
    // Do some work
    LowLevelFunction();
    // Exit Critical Section with Semaphore post
    //(sem_post) or Mutex unlock
    // Do some work
}

LowLevelFunction()
{
    // Do some work
    // Enter Critical Section with Semaphore
    //(sem_wait) or Mutex lock
    AccessSharedResource();
    // Exit Critical Section with Semaphore post
    //(sem_post) or Mutex unlock
    // Do some work
}
```

WHERE MUTEXES WORK BETTER THAN SEMAPHORES

There are at least two cases where mutexes work better than semaphores. Many times, we create several functions in a concurrent design that access the same shared resources. With a mutex, you need not worry about nested layers of critical sections of code. Listing 2 illustrates how this works.

With a semaphore, you would be locked out forever in the low-level function when called by the high-level function. With the mutex, you have options to easily solve this. Instead of using the default behavior, POSIX has two other types of mutexes. For `PTHREAD_MUTEX_ERRORCHECK` mutexes, the lock would fail and you would know you need not unlock the mutex. For `PTHREAD_MUTEX_RECURSIVE` mutexes, the lock maintains a counter so you can enter a critical section a second time with no problems (as in our low-level function). Unlocking merely decrements the counter and doesn't actually unlock the mutex.

The second advantage to using a mutex is that the mutex can only be unlocked by the thread with which it was locked. A way to remember this feature is not to think of lock and unlock but owned and available. If it is owned (i.e., locked), only the owner can make it available (i.e., unlocked). This feature helps create a much more robust design because you are forced to think in a more structured manner.

WHERE SEMAPHORES WORK BETTER THAN MUTEXES

If you have a resource such as a finite number of open-file handles and you want to gracefully handle times when you run out of file handles (not just abort your program), semaphores do the trick. These file handles are allocated and dynamically used for a time when the file is open and released when closed. In this case, a semaphore works better than a mutex. If you initialize the semaphore to the number of open file handles when it is created, your

thread can simply perform a `sem_wait` when it wants to obtain a file handle. If there is at least one available, the code does not block. If all of the open file handles are exhausted, the code will block until one becomes available. This is much more robust than aborting the program under this condition.

ALTERNATIVE OPTIONS

All locks and semaphores incur a resource penalty. However, understanding these options is critical in designing embedded systems with concurrency. If you are fortunate enough to be using a toolchain that supports the latest C standard (C11) there are a number of other concurrency controls built in. But that is for another day, since we only take things in thin slices. ☒

Bob Japenga has been designing embedded systems since 1973. In 1988, along with his best friend, he started MicroTools, which specializes in creating a variety of real-time embedded systems. With a combined embedded systems experience base of more than 200 years, they love to tackle impossible problems together. Bob has been awarded 11 patents in many areas of embedded systems and motion control. You can reach him at rjapenga@microtoolsinc.com.

RESOURCES

M. Kerisk, *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*, No Starch Press, 2010.

The Open Group, "The Open Group Base Specifications Issue 7," 2008, <http://pubs.opengroup.org/onlinepubs/9699919799>.

NEED-TO-KNOW INFO

Knowledge is power. In the computer applications industry, informed engineers and programmers don't just survive, they *thrive* and *excel*. For more need-to-know information about some of the topics covered in this article, the *Circuit Cellar* editorial staff recommends the following content:

Concurrency in Embedded Systems

Part 1: An Introduction to Concurrency and Common Pitfalls

Part 2: Atomicity and TOCTTOU

Part 3: Avoiding Concurrency Problems

Part 4: Introducing Linux and Concurrency

by Bob Japenga

Circuit Cellar 263, 265, 267, and 269, 2012

The first two parts of this article series introduce concurrency in embedded systems and discuss two common concurrency design problems. The third and fourth parts of the series examine generic ways to avoid some of these problems and describe how embedded Linux can help you design robust systems with concurrency.

Topics: Concurrency, Atomicity, TOCTTOU, Linux

Go to *Circuit Cellar's* webshop to find this article series and more: www.cc-webshop.com

QUESTIONS & ANSWERS



Engineering and “Pure” Research

An Interview with Colin O'Flynn

Colin O'Flynn is an electrical engineer and graduate student at Dalhousie University who is studying cryptographic systems. He has experience with software programming, RF layout, and PCB and FPGA design. Here Colin provides some background information about his introduction to designing with electronics, his involvement with 802.15.4 wireless communications, and his current projects, including a programmable logic board.—Nan Price, Associate Editor

NAN: Where are you located?

COLIN: I'm currently living in Halifax, Nova Scotia, Canada. I'm originally from Hamilton, Ontario, Canada, and had been living in Edinburgh, Scotland for almost two years before I moved to Halifax.

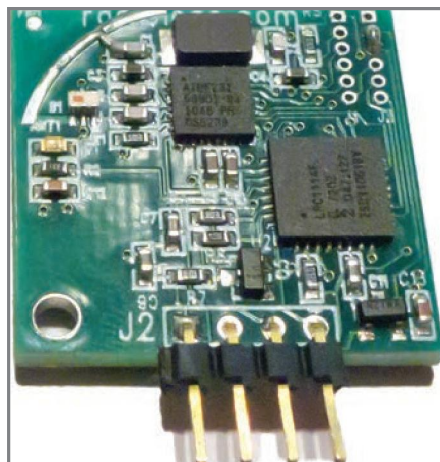
NAN: How did you become interested in electronics?

COLIN: Like many people in this area, I did start at a very young age. If I had to pin one event as the starting of my life-long interest in electronics, it was getting one of those “20-in-1” kits from RadioShack as a present. My parents always encouraged my interest in electronics, but as they were a commercial airline pilot and a chartered accountant, it wasn't the case of them initially pushing me in the same direction they started!

My dad found me a few small “learn-to-solder” kits, which I enjoyed. At age 8, I assembled my first real kit, the LED-Tric Christmas tree featured in the December 1994 issue of *Popular Electronics*. My parents have kept bringing that tree out as a Christmas decoration every year since, and it still works.

Besides my parents, I also had help from local people interested in electronics and became friends with many of the local electronics store owners. I spent

many hours building projects from magazines like *Electronics Now*, *Popular Electronics*, *Circuit Cellar*, and the various Forrest M. Mims III books. I find it interesting to see the recent surge in “maker” culture. It's something that has really been going on for years. Growing up, there wasn't such a thing as maker spaces, but there were local people with interesting workshops who would share projects. It's great to see this a little more mainstream now, as it means more opportunities for people to get involved at any stage of their life in this fascinating world.

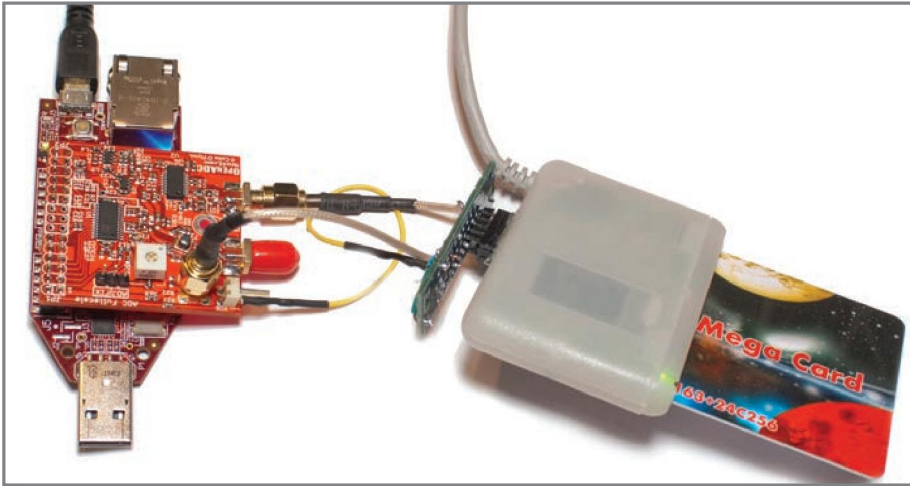


Colin's recent 802.15.4 work involves RadioBlocks, which are small, wireless modules that use the “SimpleMesh” open-source mesh networking software.

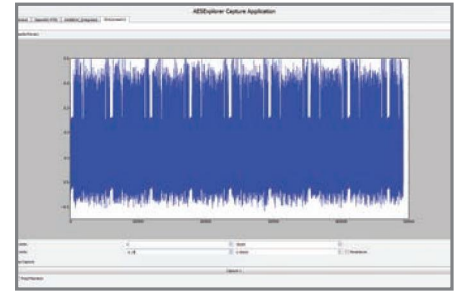
NAN: What is your current occupation? Are you still consulting for projects related to 802.15.4 wireless communications?

COLIN: I'm currently a graduate student at Dalhousie University pursuing a PhD. I decided to go back to school for the chance to do more “pure” research. It's also fun to have access to a range of tools I wouldn't otherwise get—the lab I sit in has an anechoic chamber, for example. And we have most of the latest versions of high-end software like MATLAB (including most of the add-ons), 3-D electromagnetic antenna simulation software, FPGA design software, and so forth.

I'm only loosely involved in 802.15.4 projects for now, and not actively following the latest developments and standards. Having said that, a friend of mine has gotten involved in creating small, wireless modules called RadioBlocks. They use an IEEE 802.15.4 radio combined with a small ARM Cortex-M0 microcontroller. They use an open-source mesh networking software we created called SimpleMesh, so most of my recent work on 802.15.4 has been around this project. The mesh software is designed to do the basic job of sending a block of data to another node, and otherwise staying out of the way. I previously did a



You can attack a microcontroller-based smart card using a commercial reader combined with Colin's OpenADC board on an FPGA development kit.



This screen-capture software shows different computations the smart card performs directly from the power trace.

lot of work using IPv6 on such small sensor networks, but haven't been active in that area lately.

At Dalhousie, I'm working on the area of side-channel analysis of cryptographic systems, specifically power analysis. This area has a simple idea: if you have a microcontroller or other embedded controller, it typically has some internal data bus. When those data lines switch state, it takes power. But the power actually depends on the data. Imagine a databus switching from all 1s to all 0s in a clock cycle, compared to staying at all 1s. Likewise, different operations, such as a MUL compared to a LDI, have different power signatures. If you measure the current consumption on each clock cycle, you can learn something about the data being processed, and then often the secret key. Practically speaking, you can measure this current even with an electromagnetic probe, so you don't need to physically modify the circuit board.

I gave a presentation at Black Hat Abu Dhabi in December 2012 about some of this work. If you are interested, the slides and white paper are available online at Blackhat.com, or from my personal website NewAE.com. You can see the photo above showing an example of attacking a microcontroller-based smart card. The capture software might look something like where you can see different computations the card is performing directly from the

power trace. In this case, each burst is a round of the AES-128 computation.

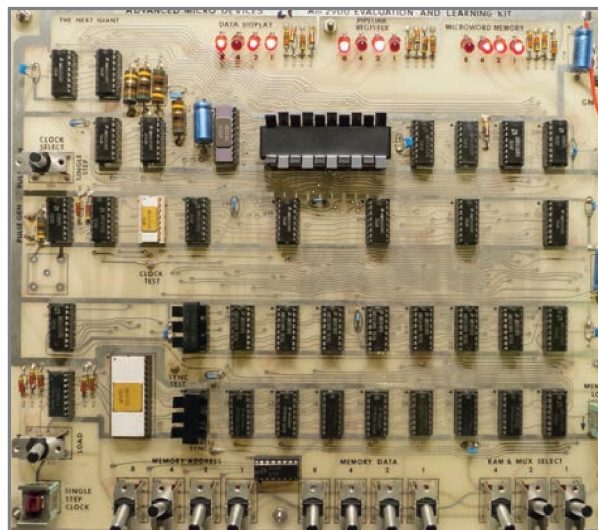
NAN: Many of your projects include Atmel microcontrollers. Why Atmel?

COLIN: It's no secret I've been a big fan of Atmel's AVR microcontroller, but it wasn't my first. I don't know the exact lineage of my microcontroller work, but one of the first things I learned on was an AMD 2900 Evaluation and Learning Kit. A local electronics store happened to have it in stock. They had gotten it from someone cleaning out old inventory, as even at that time it was old. I added heatsinks, as the several amps it drew when powered with 5 V made a lot of those chips very hot. And, of course, you had to keep the entire board powered up

if you didn't want to lose your program you'd been manually entering. From there, I moved onto a Z80 trainer board, which let you program with a hex-entry keypad, and eventually I moved onto programming it from the computer. I designed a Z80 computer board but never built it—I still have the piece of transparency with the taped out PCB design and photosensitive PCB on which I was to expose it. That's more than 10 years old now, so I suspect the chemicals in it have degraded a little!

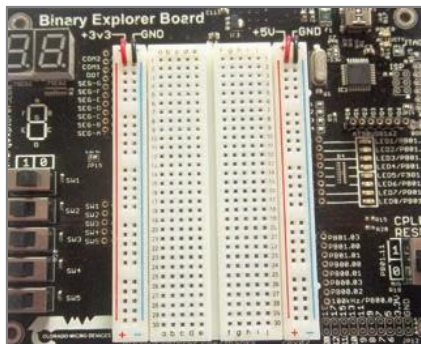
I forget exactly why I picked up the AVR, but I had one of the first AVR's released, Atmel's AT90S1200, which I programmed in Assembly. After Assembly, I programmed them in BASIC (using MCS Electronics's BASCOM-AVR), going as far to write a neural network in BASCOM-AVR. Even today, I think BASIC gets a bad rap. It was almost the original "Arduino" environment, as you could drop down LCD drivers, ADC, and so forth without ever knowing much about how it

worked, and with a really intuitive feel. I moved onto C sometime later, and used C almost exclusively for embedded development since. For some time, I was fairly involved in the tools used in the AVR world, such as WinAVR. Atmel donated a considerable amount of equipment to me, as at the time I was a high school student using these devices for science fair projects. I think that's a great example of how such corporate donations pay off. I've almost exclusively used AVR processors since I am so familiar with them because of that. In addition, as a student with little money but lots of time, I was happy to spend hours each day on AVRfreaks.net or working on open-



The AMD 2900 Evaluation and Learning Kit was one of the first microcontroller products with which Colin worked.

This is Colin's Binary Explorer board. The back side hides a small complex programmable logic device, which is programmed with the USB-AVR shown in the upper right corner. He designed this to go along with some creative-common licensed material for a university course he taught.



Who says engineers don't have fun? This was made for some of the documentation for Colin's Binary Explorer board.

source tools. While Atmel probably ended up giving me around \$3,000 worth of tools, I'm sure the value of work I performed for free in terms of open-source tool contributions or forum posts would be worth many times this.

A funny story around all this work: In undergrad, we used the Atmel AVR microcontrollers. During one of the first labs they distributed a tutorial on how to set up the WinAVR tools and compile your first program. As it turned out, this guide was something I wrote years prior and had posted to the WinAVR website. Sufficient to say, I did OK in that class.

NAN: Tell us about NewAE.com. What kind of information is available on the site?

COLIN: I've run NewAE.com since 2001, although it's not really designed to be the type of website one checks for new content daily. If I've spent some time solving a problem that I think other people could use, I'll put a post up. Sometimes this is a complete project, such as my IEEE 802.15.4 sniffer. Sometimes it's just a small post, such as how to set up the AVR USB keyboard for 5-V operation, which wasn't described in the manual. I also use it for keeping copies of any published papers or presentations.

I've more recently been posting some ongoing research to the site, including blog posts with ongoing projects, rather than just waiting until it's completely finished! In that vein, I started a YouTube channel with some technical videos (www.youtube.com/user/colinpofoynn). A big collection of these are from when I taught a digital logic course and recorded all my presentations from that.

My content spans a huge range of topics—everything from showing my students how to get screen captures, to a demonstration of my soldering station, to recordings of my academic paper presentations. I don't like duplicating work. I'll only go to the effort of making a video or website post if I really couldn't find the information elsewhere. Because of this, I don't have one specific topic you could expect to learn about. I've never been aiming to be like EEVBlog!

NAN: You wrote "It's a SNAP: A Flexible Communications Protocol" (Circuit Cellar 139, 2002) more than 10 years ago. Do you still use SNAP in any of your current projects?

COLIN: I have to admit that I haven't used SNAP in probably eight years! Of course now, when needing to network devices, I'm more likely to turn to a wireless standard.

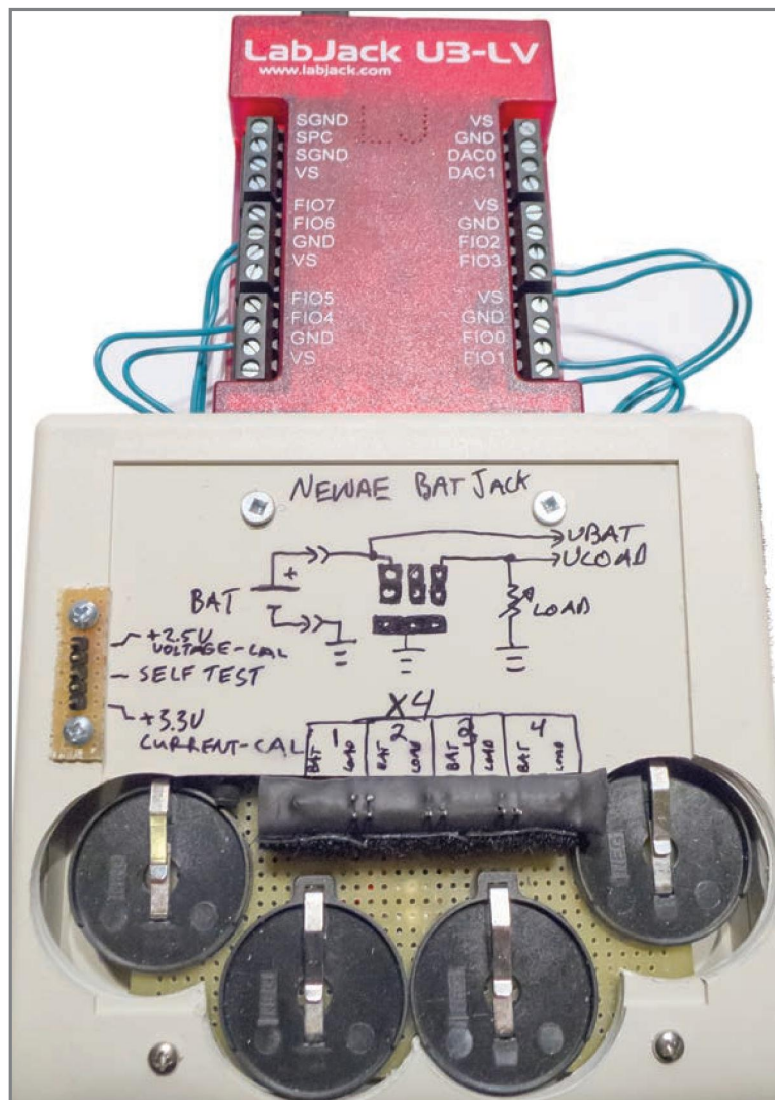
NAN: Your article "Open-Source AVR Development" (Circuit Cellar 196, 2006) provides an introduction to the AVR-GCC toolchain for AVR microcontrollers. The article references the Cygwin project and Sourceforge's WinAVR project. How do these components work in the design?

COLIN: The Cygwin project is still something I use regularly, as it lets you run a variety of Unix-like tools on Windows. The Linux command line is extraordinarily powerful, and it makes it simple to access things like C compilers, text parsing utilities, and scripting tools. With Cygwin, one can have a Linux-like experience under Windows, which I used in that article to build some of the tools you are developing for AVR. By comparison, WinAVR is just a number of prebuilt tools for the AVR development. While it's more work to build your own tools, sometimes you require special features that were not available in the premade tools.

NAN: Atmel products have played a starring role in several articles you have published in Circuit Cellar. For example, an AT90S4433 microcontroller was featured in "It's a SNAP: A Flexible Communications Protocol" (Circuit Cellar 139, 2002), an ATmega88 AVR RISC microcontroller was featured in "Digital Video in an Embedded System" (issue 184, 2005), an AT45DB041 DataFlash and an ATmega88 microcontroller were featured in "Open-Source AVR Development" (issue 187, 2006), and an AT90USBKEY demonstration board was featured in "Advanced USB Design Debugging" (issue 241, 2010). Why Atmel microcontrollers/boards? What do you prefer about these products?

COLIN: As I mentioned before, I have a long history with Atmel products. Because of this, I already have the debug toolchains for their chips and can get projects up very quickly.

When picking boards or products, one of the most important considerations for me is that readers can buy it easily. For me, this means I can get it at DigiKey (and I'll check Farnell for our UK friends). Part of this comes from being in Canada, where DigiKey was one of the first distributors offering cheap and fast shipping to Canada.



One of Colin's ongoing projects is a LabJack-based battery tester designed to simulate a small wireless node.

NAN: Are you currently working on or planning any microprocessor-based projects?

COLIN: My current big project is something I designed over the summer of 2012. It's called the Binary Explorer Board and is something I used when teaching a course in digital logic at Dalhousie University. I needed a simple, programmable logic board and nothing I could find was exactly right. In particular, I needed something with an integrated programmer, several switches and LEDs, and an integrated breadboard. The students needed to be able to use the breadboard without the CPLD to learn about discretely packaged parts. All the CPLD-based trainers I found didn't have *exactly* what I wanted in this regard.

The embedded part is the USB interface using an Atmel AT90USB162 microcontroller, although I plan on later upgrading that to an XMEGA for lower cost and more code room. The firmware is powered by Dean Camera's excellent open-source USB library called LUFA (www.fourwalledcubicle.com/LUFA.php). This firmware lets students program the CPLD on the board easily over USB. But the cool thing is you can go

even further and use the device as a generic programmer for other AVR or CPLDs/FPGAs. For example, you can mount an AVR on the breadboard, connect it to the USB interface, and program that through the Arduino IDE. The entire board would retail for \$35 in single-unit quantity, so it's cheaper than most textbooks. I'm working on making it a real product with Colorado Micro Devices right now.

The design environment is the standard Xilinx tool-chain, although I've made a number of predefined projects to make it simple enough for students with zero previous design experience to use. The idea is to get students familiar with the real tools they might see in the industry. Around this project, it's interesting to note I choose a Xilinx CPLD because of my familiarity with Xilinx devices and design tools. This familiarity comes from years ago when Xilinx donated to me a part for a project I was working on. Now throngs of students will be exposed to Xilinx devices, all because Xilinx was willing to donate some parts to a student.

There is always an assortment of half-finished projects, too. I started designing a battery tester, which could simulate characteristics you'd typically see when driving small wireless nodes from coin-cell batteries. I started planning on using an AVR USB microcontroller and doing all the data logging myself. I then found this LabJack device, which simplified my life a lot, as they had basically a generic USB-based logging/control module.

NAN: What do you consider to be the "next big thing" in the embedded design industry?

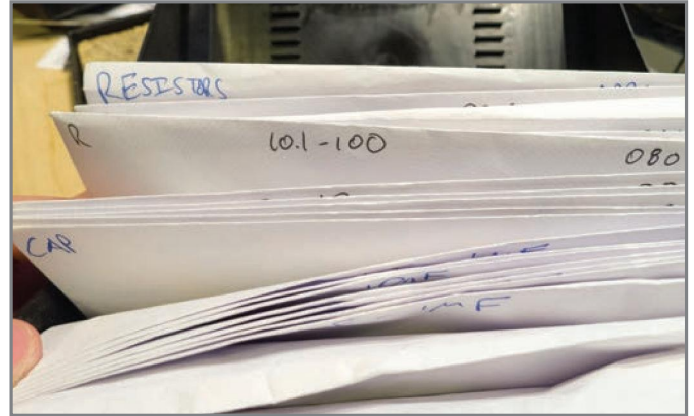
COLIN: Wireless and the "Internet of Things" will eventually be a big thing, which means design engineers will need to become more familiar with things

like protocols and realistic transmission characteristics. I use the word "realistic," as part of this world is separating hype from reality. There's certainly a huge disconnect between the marketing hype around all these various wireless protocols and how well they work in practice. When designing a product that will use a wireless technology, it's likely some commercial off-the-shelf (COTS) module will be used, so the engineer may think they can remain blissfully unaware of RF or networking things. But the engineer still needs to have a rough idea about how many devices might fit in an area on a single network or the advantage of selecting certain protocols.

Another thing of interest to me is programmable logic, such as FPGAs. It's been interesting to see the tools that try to turn anybody into an FPGA designer becoming more mainstream, or at least letting you program FPGAs in more common languages (e.g., C/C++). They are still fairly specialized and more likely to be used by a hardware engineer looking to improve productivity, compared to a software engineer who needs to offload an algorithm into a FPGA. But I think they could fairly quickly get to the point that engineers with some FPGA experience could implement considerably more complex designs than they would



Here is Colin at his custom-built workbench.



Colin stores his SMD parts in envelopes for easy reference.

have otherwise been able to had they been required to design everything from scratch.

In a somewhat similar vein, we are starting to see the availability of multicore devices coming down to embedded levels. Learning to program them in a way to take advantage of these new cores is a useful skill to pick up. I recently started using both the OpenMP API and Cilk++ development software on some of my programs. My work wasn't targeting an embedded project, but instead regular full-size multicore computers, but it's still a useful (and fairly simple) skill to pick up.

NAN: Tell us a little about your workbench. What are some of your favorite design tools?

COLIN: My initial workbench was the kitchen table, although other family members were frequently concerned about eating in the same space as these various items with warning labels about lead. My next workbench was a long, custom-built bench in Hamilton, Ontario. My current bench in Halifax was again custom-built, and I'll take you few of its features. I'd like to point out by "custom-built" I mean built by myself with a jigsaw and some plywood, not an artesian finely crafted piece of furniture.

Due to a back injury, I work standing up, which you can't see in the photo. It's actually quite refreshing, and combined with a good quality antifatigue mat and stool to lean up against means I can work long hours without tiring. A cover comes down to hide everything in my desk, which was a feature partially required by my significant other, who didn't want guests to see the typical mess of wires it contains. When closed, it also gives it some protection against any rogue water leaks. For my computer, I use a trackball instead of a mouse, and the keyboard and trackball are mounted on a plate tilted underneath the desk in a "negative" tilt angle, adjusted to most natural angle. And, because there is no way to see the keyboard while typing, it tends to keep anyone else from borrowing my computer to look something up!

I've wired a ground fault interrupter (GFI) into the desk, so all my power outlets are protected. If I ever did something dumb like dropping a scope ground on a live wire, the GFI socket would at least give me a hope of protecting the scope and myself. There are many outlets above and below the desk, and also a ground jack for the antistatic strap beside the thermal

wire strippers. The outlets under the desk let me plug in things in a hidden manner—printers, USB hubs, and other permanent devices get wired in there. I've wired a number of USB hubs to the top of my desk, so I typically have around 12 free USB slots. You always seem to run out otherwise!

Most of my tools are off the desk and stored in the drawers to either side. I made the "drawers" just pieces of wood with minimal sides—the idea being most of the time you are placing PCBs or tools down, so the lack of high sides prevents you from piling too much into them! All the cables get stored on hooks to the left of my desk, and I've got a whiteboard that sticks up when I'm working on a problem.

I store all my SMD parts in small envelopes stored in index card holders in the bottom left of my desk. While I'm not a static-phobic, I also didn't want to use plastic film strips or plastic bags. So the paper envelopes at least I hope don't generate much static, even if they don't dissipate it. It's very easy to label all your parts and also this system holds up to a high dynamic range of stock numbers. For example, capacitors get split into 10.1–99.9 nF, 100 nF, 100.1–999.9 nF, and so forth. Because you seem to end up with loads of 100-nF capacitors, they get their own envelope. It's trivial to change this division around as you get more parts, or to group part sizes together.

In terms of interesting tools: my soldering station is probably my favorite tool, a Metcal MX500 I got used from eBay. The response time on these is unbelievable. I put a video up to show people just because I've been so impressed with it. There are other manufactures that now make stations with the same RF-heating technology I believe, and I always encourage everyone to try one. I've been using the DG8SAQ Vector Network Analyzer (VNWA) for a while too. It's a very affordable way to get familiar with VNA and RF measurements. It's especially fun to follow along with some of the "Darker Side" columns in *Circuit Cellar*. Rather than just hearing about the mysterious world of RF, you can do experiments like viewing the response of several different decoupling capacitors mounted in parallel. I've got an old TiePie TiePieSCOPE HS801 parallel-port oscilloscope mounted underneath my desk, and still use it today. A lot of my work is digital, so have an Intronix LogicPort digital analyzer, a Beagle USB 480 protocol analyzer, and oodles of microcontroller programming/debug tools from different manufacturers. 📺



Fault-Tree Analysis

Fault-tree analysis tracks a system event—most often a failure—down to its root cause. It also supports the system's safety and reliability assessment by determining the event's probability of occurrence.

In my last *Circuit Cellar* article, I discussed failure mode and criticality analysis (FMECA). This month, I'll examine a similar tool, fault-tree analysis (FTA), which is also known as event-tree analysis (ETA).

FMECA and FTA are similar. Their fundamental difference is that FMECA is bottom-up analysis, while FTA (or ETA) is top-down analysis. FMECA is used to trace the propagation of a failure of a component or a function up to the effect on the system's behavior. The tree analysis starts with an event, which is then traced down. If that event is a failure, the analysis is called FTA. FTA then tracks down the failure to identify its root cause. Like FMECA, you can perform the analysis on the functional block level or you can analyze the individual component levels and their specific failure modes. This can become a gigantic task, especially when many events need to be analyzed.

RELIABILITY

Similar to FMECA, you need components' estimated failure rates obtained during reliability prediction. The commercial reliability-prediction programs I mentioned in my previous articles include modules to assist in FTA generation. They

draw Boolean symbols reflecting the system architecture, insert previously generated reliability numbers, and calculate results. But the programs can't understand the system's structure or faults propagation. That's up to you to provide. So, while the module is not a part of the free mean time between failures (MTBF) calculator, you can use any graphics program to draw FTA, or it can be hand drawn.

FTA and FMECA use the same reliability data, but they present the data from different perspectives. To analyze a system, both analyses are needed. FTA starts with an interesting or problematic system event or a behavior. Take, for example, the issue of some automobiles' undesired acceleration. It would be difficult to use FMECA to determine the root cause of acceleration. However, FTA will quickly drill down the system to identify all possible causes and their individual probabilities.

PROBABILITY

Figure 1 shown an example of FTA. As you can see, logic symbols are mainly used for OR and AND functions. Here, event A can only occur if events B and C take place. Event B can be caused by either event D or E, while event C can be the result of events F or G. Then, event E can be the result of any

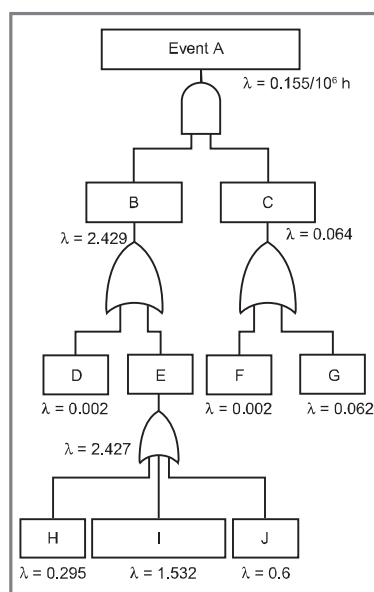


Figure 1—Logic symbols for OR and AND functions are mostly used in this example of FTA.

one of three events, H, I, and J. This can continue down until a blown transistor, a broken wire, or an operator's erroneous input is identified as the culprit. Each impetus has a probability attached to it, which is derived from the failure rate lambda (λ) when preparing the reliability prediction.

Notice that when either impetus can cause a resulting event (described by Boolean function OR) the lambdas are added. When two or more events must concurrently occur for a certain result, the model is Boolean AND (the failure rates multiply). The upshot of all this is that to achieve an event's low probability to satisfy specification requirements, you often have to include some AND gates in the architecture. This is usually accomplished through redundancy and diagnostics.

Consider a simple example. Your embedded controller activates a solenoid valve that delivers high-pressure hydraulic fluid to an actuator. Safety standards require that the solenoid valve must never be accidentally energized. "Never" is typically considered a probability of an event less than 10^{-9} per million hours (i.e., one in 114 billion years), which is several times the age of the universe, but you will find systems with even tougher requirements. The probability of deployment of aircraft engine thrust reversers in flight, for instance, is typically less than 10^{-12} per million hours.

In the example, the solenoid valve is energized through a power MOSFET. For starters, assume the controller is perfect

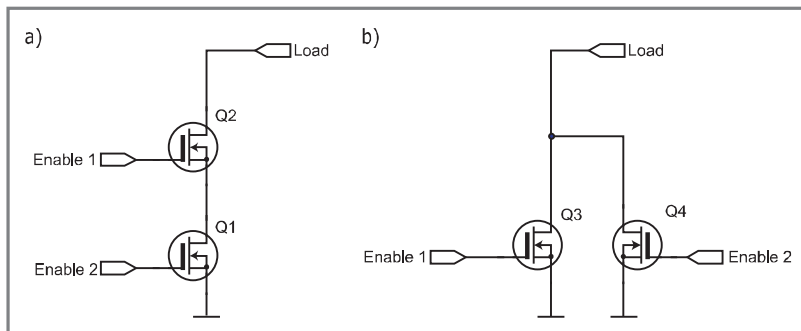


Figure 2a—With two MOSFETs in series, both would have to short out to energize the valve. **b**—You would need to connect two, three, or more MOSFETs in parallel to ensure high probability of the valve to be energized.

with a $\lambda = 0$ failure rate, so there is no possibility it will issue a wrong command. The valve, by design, cannot open without being energized. Therefore, the only cause of the hydraulic pressure being present would be solenoid valve energization through a shorted MOSFET.

The probability of a typical MOSFET failing with respect to its operating conditions is about $\lambda = 1.895 \times 10^{-4}$ per million hours. Distribution of its failures is roughly 50/50 for an open or short circuit, so the probability of the unwanted valve energization is $\lambda = 0.9475 \times 10^{-4}$. This doesn't satisfy the specification.

By putting two MOSFETs in series, both would have to short out to energize the valve (see Figure 2a). The probability of that is 8.97×10^{-9} . It's close, but not close enough! It is still too high by almost an order of magnitude, the probability of a

Gigabit Technology

- ARM-based
- System on a Chip
- Gigabit Ethernet
- Small, Cheap, Fast
- Both QFP and BGA Packages
- Standard Development Tools
- 10 Year Life Guarantee
- Royalty Free RTOS, TCP/IP V4/6



\$10.00 each
(Qty 100)


The gridARM™ System on a Chip (SOC) is a high performance, low cost, low power, highly integrated single chip with 10 / 100 / 1000 Mbps Ethernet, USB, CAN, Serial, SRAM Memory, SPI, I2C, RTC and internal peripherals designed to provide a complete solution for embedded applications.

THE NETWORKING EXPERTS




800.975.4743 USA • 1 630.245.1445
gridconnect.com/gridarm.html

Leaders in the embedded and networking marketplace providing network hardware, high quality software and services



I-40 Layer PCB
Components Sourcing
PCB Assembly
&
Everything Electronics



www.ezpcb.com

wrong command notwithstanding. You may be able to derate the MOSFET or find another MOSFET to obtain a lower failure rate. Failing that, you must use three MOSFETs in series. This would result in the probability of unwanted energization of 8.5×10^{-13} . Combined with the probability of the controller issuing a wrong command, that would likely take you to an acceptable result less than 10^{-9} . There may be an opposite requirement. You would need to connect two, three, or more MOSFETs in parallel to ensure high probability of the valve to be energized (see Figure 2b). Needless to say, the “enable” commands must be independent to avoid a single-point failure.

Because this issue affects the embedded controller’s architecture, it is important to perform the reliability and safety analyses as early as possible.

20/20 HINDSIGHT

Ignoring the importance of these analyses may lead to disastrous results. I witnessed a situation where engineers performed those analyses at the end of the program, just to satisfy the customer’s deliverables list. Their controller missed the safety target by a mile. The architecture was wrong. The repair

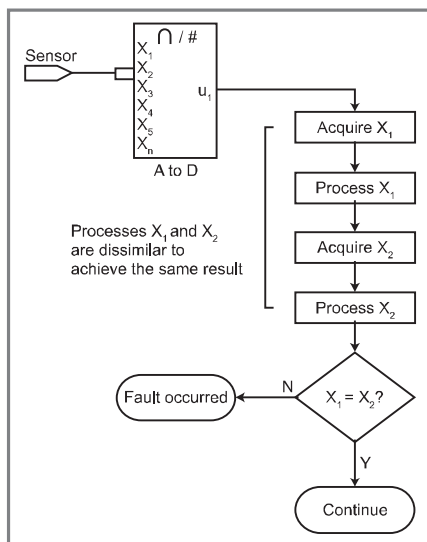


Figure 3—This is one approach to processing critical data in software.

required two years of redesign. In the meantime, a “dead man’s switch” (i.e., a push button the operator must hold down for the system to operate) had to be added until the redesigned controllers became available. All those delivered in the meantime had to be replaced. It was expensive and embarrassing. Had the engineers done their homework before embarking on the design, the embarrassment, the failure to deliver what the customer wanted on time, the budget overruns, and the penalties would have been avoided.

With software, it is always a good idea to run critical functions through two or more independent paths to reduce the probability of a forbidden execution. Figure 3 shows one approach.

Here, a sensor data is digitized by two ADC channels and processed by two different function calls. If the results are identical, the process can continue. Depending on the required safety level, the processing can be done by one or several independent microcontrollers, separate ADCs, and so forth. For high reliability and to avoid common-mode failures, you need to use dissimilar hardware and dissimilar software, including different algorithms, if possible. The combinations are endless.

I have demonstrated how important it is to perform safety analyses in a timely manner. Without them, the success of even a simple project may be unpredictable. Don’t underestimate the importance of preliminary analyses. You may not like the results. 📉

George Novacek (gnovacek@nexicom.net) is a professional engineer with a degree in Cybernetics and Closed-Loop Control. Now retired, he was most recently president of a multinational manufacturer for embedded control systems for aerospace applications. George wrote 26 feature articles for Circuit Cellar between 1999 and 2004.

RESOURCES

Defense Technical Information Center, “Failure Mode, Effects, and Criticality Analysis (FMECA),” AD-A278-508, 1993, www.dtic.mil/dtic/tr/fulltext/u2/a278508.pdf.

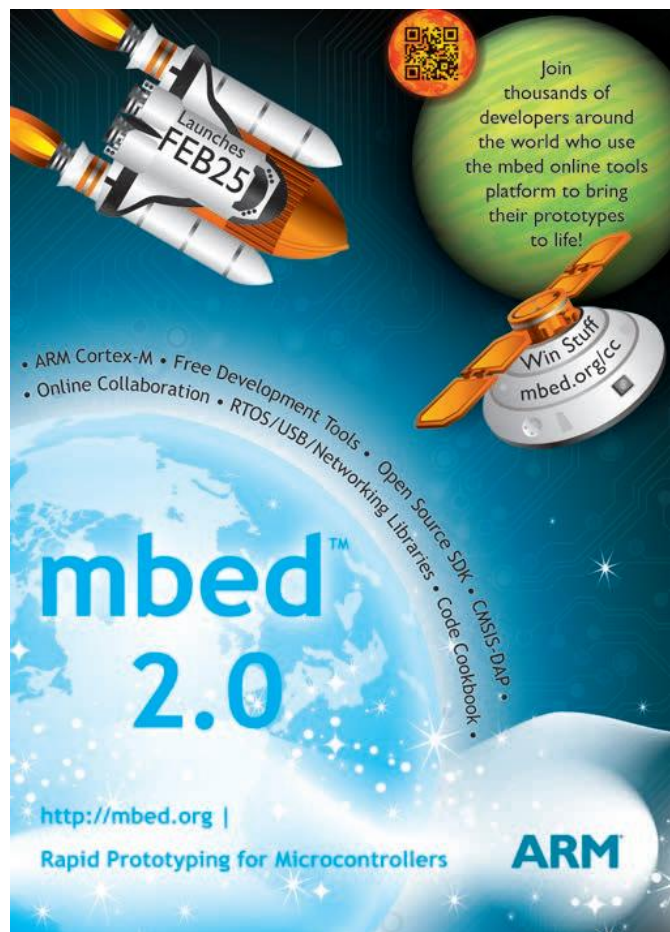
———, “Reliability Techniques for Combined Hardware and Software Systems,” RL-TR-92-15, 1992, www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA256347.

Department of Defense, “Procedures for Performing a Failure Mode, Effects and Criticality Analysis,” MIL-STD-1629A, 1980, <http://sre.org/pubs/Mil-Std-1629A.pdf>.

———, “Standard Practice System Safety,” MIL-STD-882E, 2012, www.system-safety.org/Documents/MIL-STD-882E.pdf.

G. Novacek, “Failure Mode and Criticality Analysis,” *Circuit Cellar* 270, 2013.

SoHaR, Inc., www.sohar.com.





Arduino Survival Guide

Digital I/O

The digital I/O bits on an Arduino connect it to the real world, but those microcontroller pins have analog properties that many designers neglect. Ed explores how the components inside the microcontroller affect the circuit's operation and shows how to avoid some common problems.

Assuming that you're providing proper power to your Arduino board, perhaps using the techniques I described in my October column, you must now connect the microcontroller's digital input and output bits to your project. Unlike the ideal worlds created with pure programming, the real world imposes requirements on both your circuitry and the Arduino, that, if ignored, can damage both sides of the interface.

In this column I'll examine the circuitry behind the Arduino's digital I/O pins and explain how it interacts with your circuitry. Because you can find Arduino programming information elsewhere, we can concentrate on the analog electrical properties of the digital pins: yes, digital pins have analog properties.

BEHIND THE PINS

Photo 1 shows the section of an Arduino UNO board with its 14 digital I/O pins, one ground pin, and several other pins that aren't relevant here. The digital I/O pins, named 0 through 13, line up along the edge in two headers. The small tilde (~) characters denote the six digital pins that can produce PWM outputs, a topic that I'll discuss in the next column. Incidentally, the tiny gap between pins 7 and 8 dates back to a layout error on the original Arduino PCB that has baffled and annoyed every single user since then.

Three digital I/O pins have predefined functions

that will probably interfere with any attempt to use them in your circuitry. Pin 13 connects to the standard LED on the Arduino board: the `Blink` sample sketch flashes that LED and your sketches should do the same. Pins 0 and 1 provide serial I/O through the USB adapter chip, so you should regard them as dedicated to that function.

The Arduino's six analog input pins, named A0 through A5, can also serve as digital I/O pins. Some versions of the Arduino Pro Mini board break out the A6 and A7 pins, for a total of eight additional digital I/O bits. Everything in this column applies to the A0 through A7 pins in digital mode; I'll discuss their analog functions in the next column.

Many of the problems occurring in Arduino projects arise from misunderstanding the interaction between an external circuit and the microcontroller's hardware. For example, although we generally think of digital pins as always having voltages close to 0 V and V_{CC} , the circuitry shown in Figure 1 suggests some additional complexity.



Photo 1—Arduino boards bring out 14 digital I/O pins, of which six can also act as PWM analog outputs. The TX/RX pins connect to the USB interface and generally aren't useful for other purposes. (Photo adapted from <http://arduino.cc/en/Main/ArduinoBoardUno>)

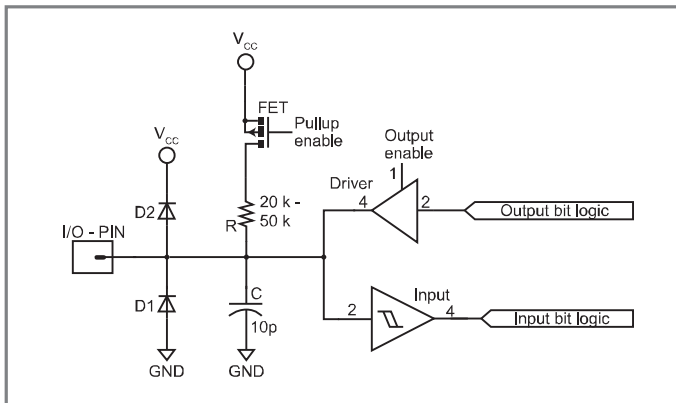


Figure 1—Behind each I/O pin lurks a collection of ESD protection diodes, a switched pullup resistor, some stray capacitance, and the internal I/O hardware that makes the pin operate according to the documentation.

The first line of the Absolute Maximum Ratings in [Figure 2](#) require that any voltage applied by the external circuit must be between 0.5 V below circuit ground (GND) and 0.5 V above the supply voltage (V_{CC}). A standard Arduino board has $V_{CC} = 5$ V, so the pin voltage must not fall below -0.5 V or rise above $+5.5$ V. D1 and D2, the protection diodes in [Figure 1](#), begin conducting when the pin voltage exceeds those limits, clamping the pin one diode drop beyond either GND or V_{CC} . If the external circuit does not limit the current, the ensuing short circuit will destroy at least the diode and, more likely, the entire chip.

The third line of [Figure 2](#) specifies a maximum current of 40 mA that the external circuitry may ram into or draw from any pin under any circumstances. It does not, contrary to what some folks seem to believe, mean that the microcontroller actively limits the pin current to 40 mA: your circuit must enforce that limit to prevent damaging the microcontroller.

For example, if you connect a digital output pin directly to V_{CC} , nothing will happen as long as the pin driver remains set to HIGH. However, if the Arduino sketch sets the output LOW, the pin driver will attempt to pull the power supply to 0 V. That attempt won't succeed and the driver will sink far more than 40 mA while destroying itself.

A 125 Ω resistor inserted in series between V_{CC} and the pin will limit the fault current to about 40 mA under the worst-case condition:

$$R = \frac{V}{I} = \frac{5 \text{ V} - 0 \text{ V}}{40 \text{ mA}} = 125 \Omega$$

As you might expect, the driver cannot maintain 0 V at the

Symbol	Parameter	Condition	Min	Max
V_{OL}	Output Low Voltage	$I_{OL} = 20 \text{ mA}$, $V_{CC} = 5 \text{ V}$		0.9 V
V_{OH}	Output High Voltage	$I_{OH} = -20 \text{ mA}$, $V_{CC} = 5 \text{ V}$	4.2 V	
V_{IL}	Input Low Voltage	$V_{CC} = 2.4 \text{ V} - 5.5 \text{ V}$	-0.5 V	$0.3 V_{CC}$
V_{IH}	Input High Voltage	$V_{CC} = 2.4 \text{ V} - 5.5 \text{ V}$	$0.6 V_{CC}$	$V_{CC} + 0.5 \text{ V}$
I_{IL}/I_{IH}	Input Leakage Current	$V_{CC} = 5.5 \text{ V}$, pin low/high (absolute value)		1 μA

Figure 3—These DC Characteristics from the ATmega328 datasheet define the microcontroller's normal operating conditions. Although your circuit may exceed the DC limits (but not the Absolute Maximum Ratings!), the results aren't specified. The datasheet includes many footnotes and additional conditions that don't appear here.

Voltage on any Pin except RESET with respect to GND	-0.5 V to $V_{CC} + 0.5 \text{ V}$
Maximum Operating Voltage	6.0 V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA

Figure 2—These Absolute Maximum Ratings from the ATmega328 datasheet set the limits for operation, but your design should also respect the limits set by the rest of the specifications.

pin while sinking the Absolute Maximum current, so the actual current will be somewhat lower. The first line of [Figure 3](#) states that V_{OL} , the output voltage for a logic LOW condition, will be less than 0.9 V while sinking 20 mA. You can assume the actual output voltage will rise as the pin sinks more current, but the datasheet does not provide a specification.

Pop Quiz: Calculate the resistor value that will maintain a valid V_{OL} at the rated test current.

Now that you have an idea of what *not* to do with the microcontroller's pins, we can examine their behavior during normal operation.

DIGITAL INPUTS

After a hardware reset, whether caused by turning the power on or releasing the RESET button, the Arduino firmware leaves all the digital pins configured as inputs. The DRIVER shown in [Figure 1](#) is disabled, the FET above resistor R is turned off, and the Schmitt trigger INPUT converts the voltage on the pin to the LOW or HIGH value that the Arduino firmware will return when your sketch executes a `digitalRead()` function.

Pop Quiz: Assume that the Arduino has just emerged from a hardware reset and switch S1 is open, as shown, in [Figure 4](#). Will a `digitalRead()` return LOW or HIGH?

Bonus 1: What value will `digitalRead()` return with your finger on the switch, but not pressing hard enough to close the contacts?

Bonus 2: Immediately after you release the switch?

Bonus 3: One minute later?

As nearly as I can tell, every single Arduino user has reasoned (at least once!) thus:

A. Because a closed switch always returns LOW

B. An open switch must return HIGH

C. Therefore, the correct answer to all four questions must be HIGH.

After some experience, most users will realize all the answers are "There's no way to know." If it's any consolation, I confess to falling into that trap myself.

Because open switch contacts present an extremely high resistance, the complete circuit for [Figure 4](#) actually appears in [Figure 1](#): all of the interesting components lie *inside* the microcontroller, not in the external circuit,

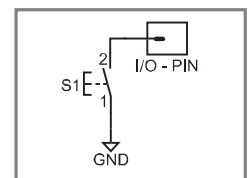


Figure 4—A simple push button switch on an input pin may not operate as you expect without proper configuration.

and, being hidden from view, nobody considers their effect on the circuit's operation. With only one external component that's not really there, what could possibly go wrong?

The answer involves capacitor C , which represents the stray capacitance from all of those internal components to ground. Any current, whether from the pin or the IC, will add to or remove charge from the capacitor and, as with all capacitors, more charge implies a higher voltage:

$$V = \frac{Q}{C}$$

The total charge moved by a constant current equals the product of the current and the time it flows:

$$Q = I \times T$$

Even with nothing connected to the pin, current may flow: Figure 3 gives the specification for I_{IL} and I_{IH} , the input leakage currents with an external circuit holding the pin LOW or HIGH. Under those conditions, less than $1 \mu A$ will flow through the pin to the circuit. The spec gives the absolute value of the current, not its direction, so you must not assume current will always flow out of a pin held below V_{IL} or into a pin held above V_{IH} . Of course, leakage current won't flow when the voltage on the capacitor reaches the limits of either 0 V or V_{CC} .

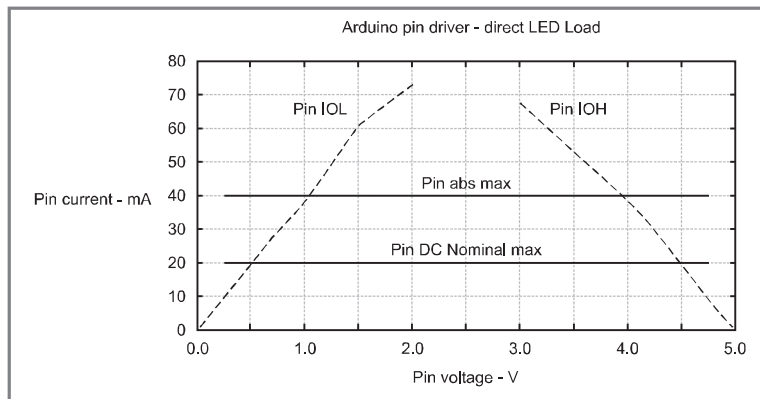


Figure 5—The MOSFET pin driver has an on-state resistance of about $45 \text{ m}\Omega$ for both high and low outputs. The external circuit must ensure that the resulting current remains below the maximum limits.

For the circuit in Figure 4, the capacitor will accumulate charge from the leakage current, with the pin voltage changing accordingly. The specs don't define the leakage current in that situation, but $1 \mu A$ should be a reasonable estimate. Combining those two equations, solving for V/T and plugging in the numbers shows how rapidly the pin voltage will change:

$$\frac{V}{T} = \frac{I}{C} = \frac{1 \mu A}{10 \text{ pF}} = 100 \text{ V/s}$$

That means the leakage current can change the pin voltage by 5 V (the maximum possible difference) in 50 ms. Because



@editor_cc

#microcontroller#circuit#embedded#FPGA#electricity#EEPROM
#tech#volts#ADC#analog#DSP#WiFi#robotics#programming
#RFID#code#schematic#logic#PWM#electronics#debug#bit#MCU
#RTOS#ohm#byte#sensor#engineering#PCB#signal#processor
#RAM#servo#CPLD#encoder

 Follow us on Twitter

Keep in touch and interact with the *Circuit Cellar* editorial department

Pitch ideas for articles

Stay informed with valuable product announcements

Learn about upcoming industry events, conferences, and more



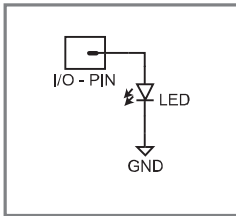


Figure 6—An output pin can't directly drive an LED: don't try this on your bench!

the current can flow in either direction; however, you cannot predict whether the pin will be LOW or HIGH even a fraction of a second after the switch in Figure 4 opens. In fact, it may be neither: the net leakage current may be zero at some capacitor voltage between the limits.

The capacitor charge can change as the result of an external electrostatic field or a strong RF signal. Even though a short length of wire or PCB trace forms a poor low-frequency antenna, the electric field intensity due to nearby power line currents can impose a 60 Hz (or 50 Hz, depending on the power frequency) waveform on the input pin voltage. An Arduino sketch expecting a pushbutton signal will produce bizarre results when the input toggles at 60 Hz!

The Arduino doc suggests using a bare digital input to sense photodiode current, build capacitive switches, and measure other interesting physical phenomena. Unless your circuit includes such esoterica, however, you almost certainly want to turn on the FET that connects the digital input pin to V_{CC} through R , the pullup resistor. The current through the resistor will be at least 100 μA with the pin at 0 V, far more than the leakage current, and will ensure the pin remains HIGH with no external circuitry.

Prior to Version 1.0.1, the Arduino mechanism for enabling the pullup resistor mimicked the microcontroller's hardware design:

```
pinMode(pin_number, INPUT);
digitalWrite(pin_number, HIGH);
```

I think the notion of *writing* to an *input* bit seemed so counter-intuitive that many users simply could not remember it, leading to erratic input behavior and considerable head-scratching. The current Arduino firmware performs the write automatically when it encounters this statement:

```
pinMode(pin_number, INPUT_PULLUP);
```

As shown in Figure 1, the pullup resistor value may lie anywhere between 20 k Ω and 50 k Ω . If your circuit requires a specific resistance, you must provide a suitable external resistor and leave the internal pullup disabled:

```
pinMode(pin_number, INPUT);
```

Remember that a digital input pin requires a pullup resistor only when the external circuit doesn't provide a low-resistance path for the leakage current. A logic gate or driver will hold the input in a known state without the need for a pullup resistor.

A microcontroller with all inputs and no outputs can't do much more than dissipate power, so let's look at how digital outputs work.

DIGITAL OUTPUTS

Because all digital I/O pins default to being inputs after a reset, the Arduino sketch must specifically configure each output bit as part of the `setup()` function:

```
pinMode(pin_number, OUTPUT);
```

Prior to that statement, the pin behaves as an input with its pullup resistor disabled. Afterward, the DRIVER block in Figure 1 will hold the pin LOW. If the external circuit doesn't have a pulldown resistor, the pin voltage may change from HIGH to LOW when it becomes an output.

The DRIVER block in Figure 1 contains a pair of MOSFETs that connect the pin to either GND or V_{CC} , corresponding to the output from the Arduino sketch:

```
digitalWrite(pin_number, LOW);
digitalWrite(pin_number, HIGH);
```

The Absolute Maximum DC Current Per I/O Pin specification, shown in Figure 2, requires that the external circuit must prevent more than 40 mA from flowing into or out of the pin. As discussed earlier, a 125 Ω resistor will suffice, but the V_{OH} and V_{OL} specifications in Figure 3 suggest somewhat more gentle handling, with a test current of only 20 mA.

Figure 5 plots typical I_{OL} and I_{OH} currents against pin voltage (using data points eyeballed from graphs in the datasheets), showing that the DRIVER FETs have an on-state resistance of about 45 m Ω . A datasheet footnote gives the strict requirements permitting pin current far in excess of the Absolute Maximum limits, but very few practical circuits can comply with those rules.

Figure 6 shows one regrettably common circuit that operates outside the rules: a 5 mm LED driven directly from an Arduino digital output, without a ballast resistor to limit the current. This seems to be a classic case of the irresistible force meeting an immovable object, but, at least for some LED colors, it comes surprisingly close to working.

Although you could build a Spice model to find the current through the LED, there's an easier way that dates back to the

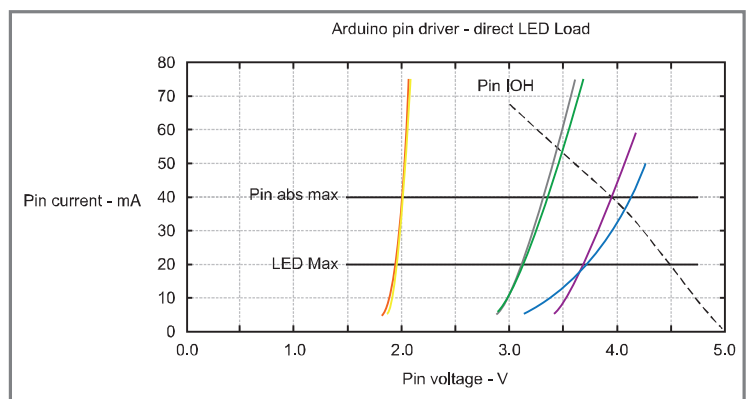


Figure 7—The intersection of an LED load curve with the Arduino pin driver source curve gives the actual current and voltage in the circuit. The blue and violet LEDs operate just barely within the microcontroller's current limit, but far beyond the maximum DC current allowed for a 5 mm LED. The red and amber LEDs would operate, perhaps briefly, at 90 mA.

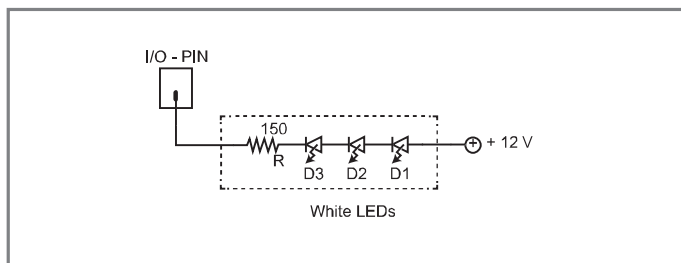


Figure 8—An Arduino cannot directly drive a white LED strip light, because the applied voltage exceeds the pin limits. The protection diodes in Figure 1 will conduct current when the pin driver goes HIGH.

early days of vacuum tube circuits: graphical load-line analysis. The general idea involves plotting both the driver and load characteristics on a single graph, so that the axes show the current and voltage common to both devices. If the curves intersect, the coordinates of that point give the current and voltage that satisfy both characteristics.

Figure 7 shows the load-line analysis of several LEDs from my collection, along with the pin driver V_{OH} curve from Figure 5. The violet LED intersects the driver curve at (4.0 V, 40 mA) and the blue LED at (4.1 V, 35 mA). Both LEDs barely meet the Absolute Maximum limit for an Arduino pin, so you might think such a circuit would work.

However, most 5 mm LEDs have a 20 mA Absolute Maximum DC current limit due to their power dissipation capacity. Those two LEDs dissipate 150 mW, far over their rating, and will probably fail in fairly short order.

The green and white LEDs would operate around (3.5 V, 55 mA) and burn nearly 200 mW. The red and amber LED curves would intersect the I_{OH} curve outside the graph at about (2.2 V, 90 mA), also dissipating 200 mW. I have seen such a project (*not* one of mine) using unballasted red LEDs: they were very, very bright!

The curves in Figure 7 can help you select the ballast resistor quickly and easily. First, pick the operating current, then read the pin output voltage and the LED forward drop at that current. The ballast resistor must drop the remaining voltage while carrying the LED current. Driving a green LED at 10 mA, for example, requires a 170 Ω resistor:

$$R = \frac{V}{I} = \frac{4.7 - 3.0 \text{ V}}{10 \text{ mA}} = 170 \Omega$$

Remember that those curves apply to the LEDs in my collection, so you should measure the LEDs you intend to use at the appropriate current.

Homework: Analyze the pin circuit when *sinking* current from an LED connected directly to V_{CC} . You will find the LED and pin data in this column's downloadable files.

Bonus: Calculate the proper ballast resistor for a green LED in that circuit.

The circuit in Figure 8 shows another tempting circuit that might work: a white LED strip light connected directly to an Arduino pin. The voltage at the pin seems acceptable, as the forward drop of three white LEDs subtracts 9.6 V from the supply at their normal 20 mA operating current, leaving 2.4 V at the pin. The built-in 150 Ω resistor limits the current from the 12 V supply to well within the pin driver limit:

$$I = \frac{12 - (3 \times 3.2 \text{ V})}{150 \Omega} = 16 \text{ mA}$$

Figure 1, however, shows that D2, the upper protection diode, will be forward-biased to V_{CC} when the driver goes HIGH. Remember that the current through a diode depends exponentially on its forward voltage, which means that the simple "diodes don't conduct below the knee" rule-of-thumb does not predict their actual behavior.

Rather than risk incinerating an Arduino, I breadboarded a section of LED strip with an ordinary diode and a bench supply. Applying 7 V produced about 2 μ A, enough to barely light the LEDs in a tribute to modern LED efficiency. Admittedly, they were dim, but most applications require complete extinction rather than an "almost off" glow.


Homework: Duplicate that experiment with a few red or amber LEDs.

The bottom line is that you must use a buffer between the pin and the LEDs to isolate the port from both excessive current and high voltage. Small MOSFETs with logic-level gates make nearly perfect buffers for most projects, because their relatively high channel resistances don't dissipate much power for low currents.

Along the same lines, small relays don't work well when directly connected to Arduino outputs without a buffer. An output pin obviously can't drive a 12 V relay coil, but even 5 V DIP relays generally require more current than the pin can provide. In either case, you must also manage the winding's inductive current during turn-off with a flyback diode and, perhaps, a resonant snubber circuit on the MOSFET drain.

Finally, the last line of Figure 2 imposes an overall 200 mA limit on the total current through the microprocessor's power and ground pins. While driving 20 mA through a few output pins will be no problem, driving 20 mA through all 19 available digital outputs at the same time will certainly not work the way you expect!

CONTACT RELEASE

Now that you understand the limits of the hardware behind the digital I/O pins, your Arduino program should see and control the real world without confusion. Well, at least confusion due to analog effects; any other confusion is just a simple matter of software... 

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

RESOURCES

Arduino information and photos, <http://arduino.cc>.

Atmel Corp., ATmega168 and ATmega328 microcontroller datasheets, www.atmel.com/devices/ATMEGA328P.aspx.

SOURCE

Arduino UNO

Arduino| <http://arduino.cc/en/Main/ArduinoBoardUno>



Introduction to Standing Waves

Standing waves are oscillations that remain in a constant position. This article describes a signal generator-based experimental design you can use to see and measure standing waves. It also shows how signal reflectors are used to generate standing waves on a transmission line.

Welcome back to the Darker Side. Some concepts, such as voltage between two nodes or current circulating into a wire, are easy to understand. On the contrary, some are a little more difficult to grasp. Standing waves—often encountered when working on high-frequency signals and in particular antennas—are probably in this second category. The good news is, experimenting with standing waves is an excellent way to understand them!

In this article, I will demonstrate how you can see and play with standing waves. Moreover, I will use a homemade low-cost experimental setup that can be easily reproduced with a signal generator.

WHAT ARE STANDING WAVES?

Let's start with the basics. What are standing waves? Imagine you have a sine wave propagating at a constant speed on a given transmission medium. It could be a mechanical vibration on a rope, a sound vibration in the air, or an electrical signal over any kind of transmission line. Imagine now that, for some reason, this forward wave is reflected back at a given point. The reflected wave will have the same wavelength and will propagate backward at the same speed. Depending on the reflector's characteristics, this reflected wave will have a given amplitude and phase, but it will inevitably add up to the forward wave (see [Figure 1](#)). Depending on the relative phase between the forward and the reflected waves, the resulting amplitude at a given point can be either higher (i.e., constructive interference) or lower (i.e., destructive interference) than each individual wave.

Note: I used the LibreOffice Calc spreadsheet program to generate the figures in this article. The corresponding spreadsheets are available on *Circuit Cellar's* FTP site.

Now, the fun part. What happens if you examine the experiment over time? As the forward wave propagates, the reflected wave is successively in and out of phase with the former, and the resulting combination's amplitude oscillates.

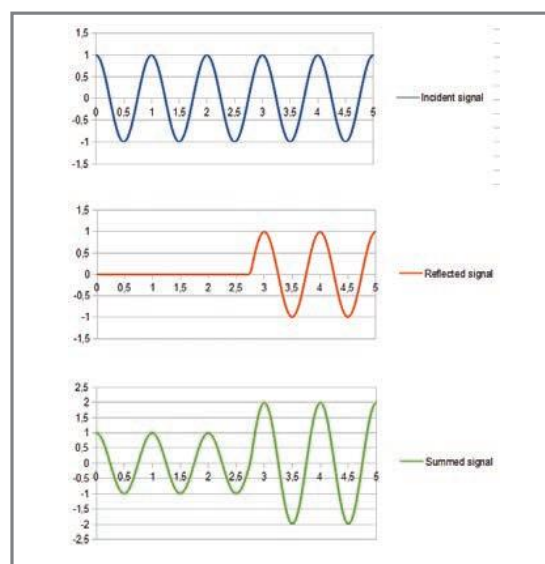


Figure 1—The blue curve is a simulated forward wave. It is reflected back on the right side of the graph with the same amplitude and no phase shift (red curve). The forward and reflected waves add up and provide the bottom green shape.

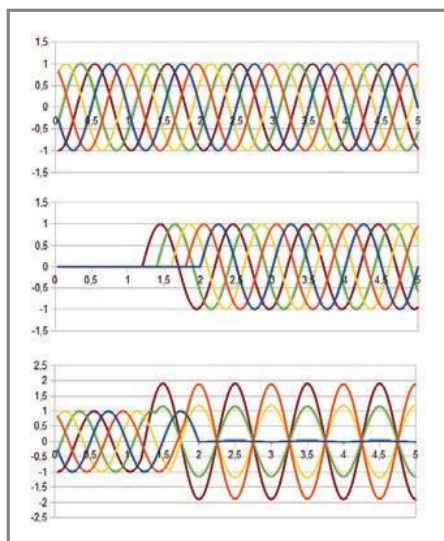


Figure 2—This is the same simulation as in Figure 1, but with snapshots at different time steps (in different colors). It's interesting that standing waves appears on the bottom plot. Here, there is a maximum node for each half wavelength, with the first one on the right (position of the reflector, as the reflector induces no phase shift on the wave). The null nodes are a quarter of a wave length away.

What's interesting—and can be easily demonstrated if you love trigonometric formulas—is that this resulting wave has fixed maximum and minimum at given points in space. These points, called nodes, don't move over time. This is why this phenomenon is called a “standing wave.” More specifically, each pair of maximum (or minimum) nodes is separated by half the wavelength. Their exact positions and amplitudes are

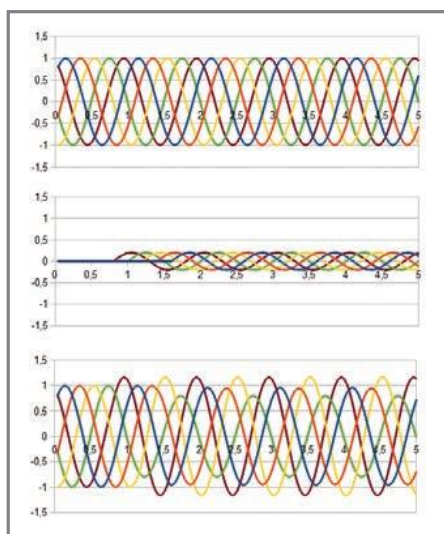


Figure 4—If the reflected wave is smaller in amplitude, there are still standing waves, but their amplitudes are lower. For example, look at the maximum points of the sine curves on the lowest graph. The curve connecting these maxima is still a sine wave.

dependent on the reflector's characteristics. **Figure 2** shows a simulation of a perfect reflector (i.e., the reflected wave has the same amplitude and phase as the forward wave). In that case, there is a maximum node at the reflector position, a null node a quarter of a wavelength away, another maximum a quarter of a wavelength away, and so forth. Don't be confused. If you put a probe somewhere and measure the wave amplitude over time you will always get a sine signal. However, its amplitude will be the highest if you are at a maximum node and could be null if you put your probe on a minimum node. I admit this is a little strange. You have to accept that there could be no signal at all at a given point even if two waves are propagating in both directions, but this is a fact. If you are interested, you will find animated plots and videos of rope-based experiments online that may help you to understand. (See the Resources section at the end of this article.)

Now, what happens if the reflector is shifting the signal's phase by 180°? **Figure 3** shows you the answer. You would get the same behavior, but the position of the maximums and nulls are shifted by a quarter of the wavelength. In that case, there is a null at the mirror position, not a maximum, as in the previous case. It's easy to remember. If the reflector shifts the phase by 180°, then the sum of the forward and reflected waves is always null at the reflector position. Lastly, **Figure 4** shows what happens if the reflected signal's amplitude is significantly smaller than the forward wave. It's the same behavior, but the minimums are not null and the maximums are lower in amplitude.

EXPERIMENTAL PLATFORM

That's enough theory and simulations. I wanted to use electrical signals to build an easy-to-replicate standing-wave demonstration setup. *Circuit Cellar* readers are more used to electrons than mechanical waves. With the help of my colleague, Yannick Avelino, I first built a 40-cm long, 50-Ω microstrip transmission line. As I have already devoted a full article on that topic (“Microstrip Techniques,” *Circuit Cellar* 223, 2009), I will just remind you this type of microstrip line is simple to build. You just need to etch a copper track on the top of a PCB with a full ground plane on the other face. The track's width should be calculated for the required impedance depending on the PCB characteristics. Here I used a 0.8-mm thick FR4 laminate,

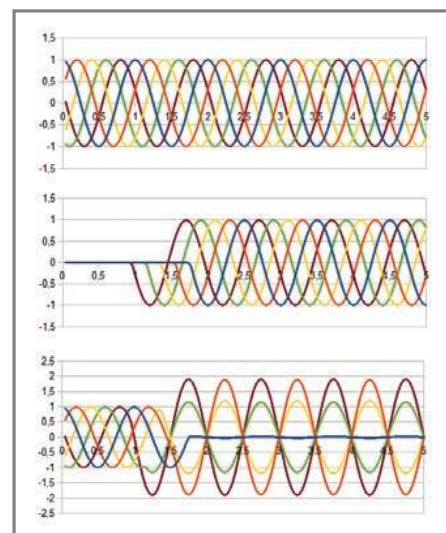


Figure 3—The reflector adds a 180° phase shift on the signal. The shapes are similar to Figure 2, but the maximums and null nodes are shifted by a quarter of a wavelength.

which required a 1.5-mm wide track for 50-Ω impedance. Instead of using chemistry, I simply used a milling machine to etch this track and soldered SMA connectors on both ends. **Photo 1** shows the result, which was verified using the Hewlett-Packard HP8510 vector network analyzer, which is also shown in the photo.

Next, I needed an input signal. To have measurable standing waves, I had to use a signal with a frequency high enough compared to the transmission line's length. After some experimentation, I settled on a 500-MHz frequency, which corresponds to a wavelength of $L = c/f = 3 \times 10^8 / 500 \times 10^6 = 60$ cm in the air. On a microstrip line, the wavelength is smaller than in the air, with a ratio of approximately 1.77 (the square root of the effective dielectric constant of



Photo 1—I use this microstrip transmission line for my experiments. The line is simply a copper track on top of a plain copper ground plane. On the top side, there are still two ground planes on both sides, but they are significantly far from the transmission line. On the back you can see the HP8510 vector network analyzer used to check the setup.

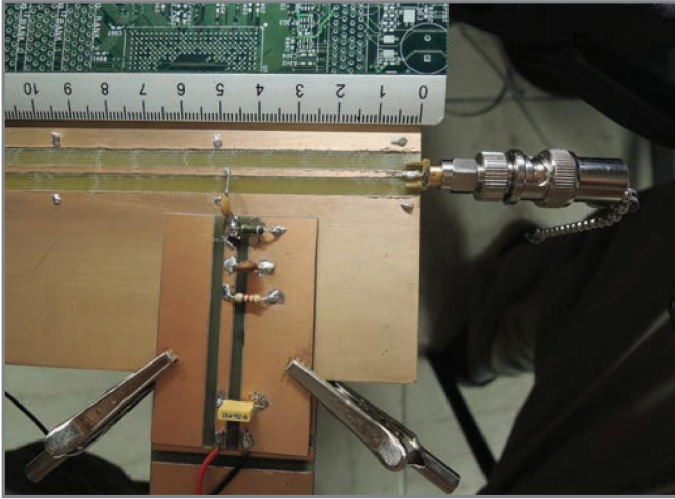


Photo 2—The assembled test probe slides on the test transmission line and is kept in place with two grabbers, which interconnect the ground planes at the same time. On the right, you can see the 50-Ω test load connected to the SMA plug.

the PCB substrate), providing an effective 33.8-cm wavelength, which is smaller than the PCB's size. I used one of our lab RF generators to produce this 500-MHz signal. Since the signal frequency's accuracy is not critical for this experiment, you can use any RF generator able to provide a 500-MHz signal with a reasonable output power (e.g., 10 mW). You can build one too, but it will require hours of extra work.

I also needed a way to measure the signal amplitude. The idea was to build a small slider that could be easily positioned anywhere on the microstrip line. I wanted to use a simple diode detector circuit to convert the RF signal amplitude into a DC voltage that could then be evaluated with a standard multimeter. [Photo 2](#) shows the prototype, which was built on another small PCB. Admittedly, I spent more time than anticipated on this circuit, simply because you can only achieve good results if the measurement probe itself doesn't disturb the signal you want to measure, which is more easy to write than to design. I ended up with the schematic shown in [Figure 5](#). The signal is grabbed on the line through a 100-Ω SMT resistor, which must

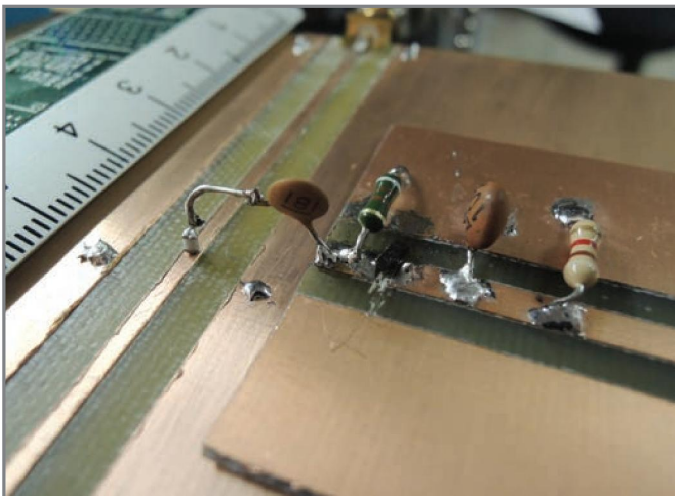


Photo 3—Here is a close up of the test probe assembly. The detecting diode is the small black part between the two ceramic disk capacitors. You can also see the small white SMT 100-Ω resistor that is just touching the transmission line with the ceramic capacitor used as a spring. Soldering this SMT resistor is a good exercise!

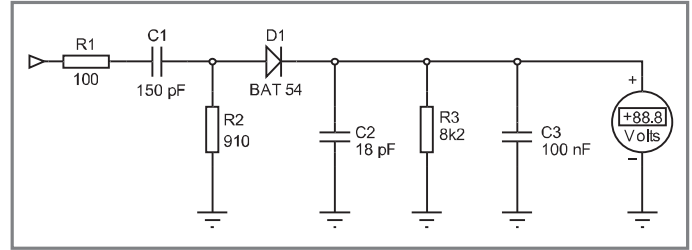


Figure 5—The test probe schematic is quite simple. The RF signal is grabbed by a small 100-Ω resistor (on the left) to avoid any extra load on the line. A standard multimeter is used to rectify, filter, and measure it.

be in direct contact with the line to avoid any extra load on the microstrip. I soldered this resistor on the end of a 150-pF ceramic capacitor, which serves as contact spring (see [Photo 3](#)). The signal is then centered to the ground level with a 910-Ω resistor, detected with a BAT54 Schottky diode (this model is not supposed to work at such a high frequency), and filtered out using a 18-pF ceramic capacitor paired with a larger 100-nF standard capacitor. As usual, the smallest capacitor must be soldered as closely as possible to the diode, and all the components working at the RF frequency must be soldered as closely as possible to each other. This probe PCB is maintained on the transmission line PCB thanks to a pair of grabbers, which connect both ground planes together. Be careful, the diode detector is not calibrated at all, so the measured amplitude is just a qualitative measurement. But, it is enough to know if the wave is strong or not at a given point. The components values are not critical, so feel free to experiment with what you have on your shelf.

REFLECTORS

Lastly, I needed sample signal reflectors to generate standing waves on the line. This is an easy task. RF circuit theory states that there is a reflected signal each time there isn't a perfect impedance matching somewhere, and in particular, between the line and the load on which it is connected. The higher the mismatch, the higher the reflection. There are two ways to create a perfect impedance mismatch: leaving an RF line open or short circuiting it to ground. In both cases, maximum standing waves would be generated, but with a different phase condition. A short circuit implies there is a null voltage at the extremity of the line (and, by the way, a maximum current, but that's another story). Therefore, a null node on the standing-wave pattern at the short-circuit position is expected. Conversely, an open line implies a null current and a maximum voltage at the end of the line. In that case, a maximum node where the line is opened is expected.

For the experiment, an open line is easy to do, just leave the output SMA connector unconnected. For the short circuit you can either use a ready-made "short" reference plug or use a male SMA and a very short short-circuiting wire to build one yourself. A test could be made with a 50-Ω reference load. In that case, the line impedance is matched with the load, so no reflection and therefore no standing waves are expected.

MEASUREMENTS!

Now it's time to check if the theory matches the experiment! I successively connected a short, a 50-Ω load and an open circuit to the output SMA connector, switched on the RF generator, gently slid the probe on the line, and measured the signal amplitude every 1 cm. Then I plotted the results on a graph.

Feel the Love Take \$25 off

Purchase **CCGold**
between now and
the end of March
and save \$25



For details and to purchase visit:
www.cc-webshop.com

CIRCUIT CELLAR®

Figure 6 shows the uncorrected figures. Let's analyze it.

First there are, as expected, standing waves. I measured regularly spaced nulls and maximums on the signal voltage both with open and shorted terminations. These nulls are spaced by approximately 16 cm, which must correspond to half a wavelength of the 500-MHz signal. This is a great first result, as it enabled me to actually measure the light's velocity on the microstrip line. Let's do the math. The measured wavelength is 32 cm (i.e., 2×16). Therefore, the signal's velocity is 160,000,000 m/s (i.e., $V = 500 \text{ MHz} \times 0.32 \text{ m}$). That's 53% the speed of light in free space (i.e., 3.10^8 m/s), which is close to the expectation. (This factor was expected to be the inverse of the square root of the effective dielectric constant of the line, so $1/1.77 = 56\%$. We are close.)

Then look at the signals' phase. The open-line test (plotted in orange in Figure 6) shows a maximum close to 0 cm from the end, as expected, and the short-circuited test (yellow) shows a null nearly at the same position, as expected. More exactly, the standing-wave node is a little to the right of the "0-cm" mark, which is normal as the effective open or short circuit is at the end of the SMA adapter, around 1 cm away from my rule's origin. Once again, it's very close to the theory.

The most intriguing case is the 50- Ω load test (shown in blue on the graph). I expected a straight line (i.e., no standing waves). However, if you look closely, you will see that the measurement shows two things. First, the overall curve is moving down from left to right (the minimums are, for example, lower on the right than on the left). This means there is some power leakage on the line, which isn't a surprise. The more surprising fact is that there is a significant amount of standing wave, which isn't supposed to be present with a matched load. This enabled

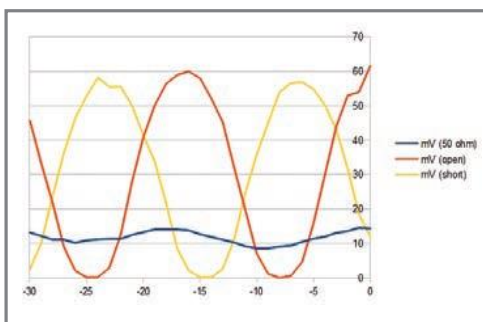


Figure 6—This is the actual result of the standing waves measurement on my experimental setup. The yellow plot was made with a short circuit as a reflector. As expected, there is a null close to the reflector position (on the right of the plot, 0 cm). Similarly, the orange curve was made with an open circuit. The blue curve was made with a 50- Ω load.

me to introduce the notion of voltage standing wave ratio (VSWR).

Measure the voltage of the standing wave pattern, divide it by the voltage of the minimum nodes, and you've got the VSWR figure. This is, therefore, a figure representative of the load matching quality, the closer to 1 the better. So, you shouldn't be surprised that ham radio guys are speaking of their antennas' VSWR all the time. If the VSWR isn't close enough to 1, the antenna is not well adapted to the 50- Ω line and some power is lost. You can find the formulas that link VSWR with the other characteristics of the impedance matching (e.g., impedance and reflection coefficient, return loss, etc.) online.

In my experiment, the measured VSWR with the 50- Ω connected on the line is about 1.42 (i.e., 14 mV/9.8 mV). This is far higher than what you should get with a proper 50- Ω load. I was wondering what was going on and decided to double check the 50 Ω I was using. To be honest, this load was in my lab for years. It was a low-quality BNC load probably from an old 10baseT Ethernet network. As we are lucky enough to have some nice test equipment in the lab, I switched on our HP8510 vector network analyzer, connected the 50- Ω load on its test port, and, I got the results shown in Photo 4. The load is effectively 50 Ω at 0 Hz, but its measured impedance at 500 MHz is $54.7 + 9.4j \Omega$. That's 10% off the expected 50- Ω impedance, which is a significant deviation. The analyzer was also able to calculate the corresponding VSWR. So, after finding the correct button on the analyzer, I got the results shown in Photo 5.

My reference load has a measured VSWR of 1.22. This is not exactly the 1.42 I got on my VSWR experiment, but at least that explains a significant part of the error. My 50- Ω load was not exactly a 50- Ω load. The remaining discrepancy is probably linked to both the low-quality SMA-to-BNC adapters I've used and to measurement errors.



Photo 4—The 50- Ω test load was measured on a HP8510C vector network analyzer. On such a plot, a 50- Ω perfect impedance is at the center. Here the cursor position, at 500 MHz, shows that the measured impedance is not exactly 50- Ω , more precisely it is measured at $54.7 + 9.4j \Omega$. Such a complex impedance includes both a magnitude and a phase measurement.

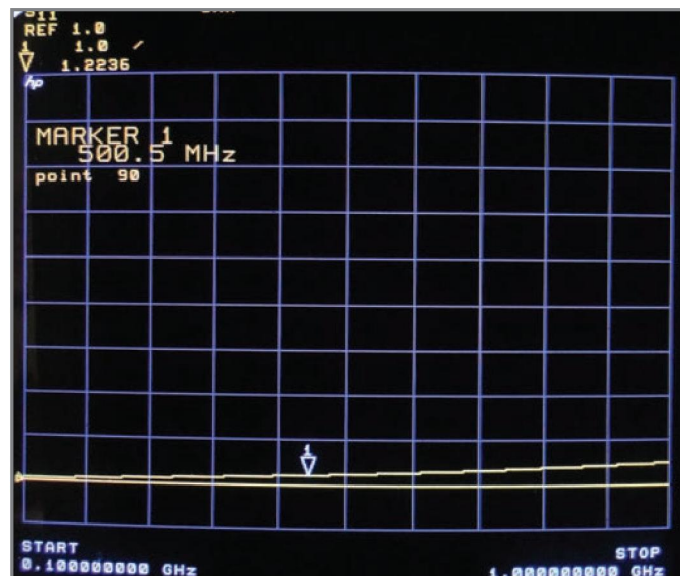


Photo 5—The test load's corresponding VSWR is plotted against frequency. At 500 MHz, the measured VSWR is 1.22.

WRAPPING UP

Here we are. I will be honest: This was the first time I actually measured standing waves, even if I have to play with VSWR figures quite often. Moreover, I have not invented anything in this article. The methods I showed you were the only methods an RF engineer could use to evaluate impedance matching prior to the development of the first vector network analyzers. If you search eBay, you can still find some so-called "slotted lines," which are the equivalent of my microstrip line but with far greater precision measurement capabilities. This equipment, like the well-known Hewlett-Packard HP 805A slotted line and the accompanying HP 415A standing-wave indicator, was developed in the early 1950s. You can learn a lot by looking at their characteristics (see the Resources section).

Anyway, I was pleased to see my low-cost experiment was enough to get results nicely fitting with the theory. Standing waves are fun and they are easy to play with. I hope you will try it yourself. There is no better way to bring a subject out of your own darker side than by practicing! ☺

Robert Lacoste lives near Paris, France. He has 24 years of experience working on embedded systems, analog designs, and wireless telecommunications. He has won prizes in more than 15 international design contests. In 2003, Robert started a consulting company, ALCIOM, to share his passion for innovative mixed-signal designs. You can reach him at rlacoste@alciom.com. Don't forget to write "Darker Side" in the subject line to bypass his spam filters.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2013/271.

RESOURCES

R. Lacoste, "Microstrip Techniques," *Circuit Cellar* 223, 2009.

LibreOffice, www.libreoffice.org.

The Memory Project, "Microwave Measurement Accessories: Impedance Measurement," www.hpmemory.org/wa_pages/wall_a_page_06.htm.

Microwaves101.com, "Slotted Line Measurements," 2009,

www.microwaves101.com/encyclopedia/slottedline.cfm.

MindBites, Inc., "Physics in Action: Standing Waves on a Rope," 2009, www.mindbites.com/lesson/4603-physics-in-action-standing-waves-on-a-rope.

D. Russell, "Acoustics and Vibration Animations: Reflection from an Impedance Discontinuity and the Standing Wave Ratio," Graduate Program in Acoustics, The Pennsylvania State University, 2011, www.acs.psu.edu/drussell/Demos/SWR/SWR.html.

Wikipedia, "Standing Wave," http://en.wikipedia.org/wiki/Standing_wave.

SOURCE

HP8510 Vector network analyzer

Hewlett-Packard, available through distributors such as eBay (www.ebay.com)

\$51^{For 3} PCBs

FREE Layout Software!

FREE Schematic Software!



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

expresspcb.com

FROM THE ARCHIVES

In celebration of *Circuit Cellar's* 25th year, we're running an article from the archives each month that exemplifies something special about this magazine, its contributors, and its readers. We hope you'll enjoy reading (or perhaps rereading) these innovative projects as much as we did preparing them. This month, we feature a 2003 article about a microcontroller-based device that uses a PCMCIA card to interact with wireless networks.

by Roy Franz
Circuit Cellar 157, 2003

The WiFi SniFi

Sniffing In and Out of Wireless Networks

Are you having trouble locating 802.11b wireless networks? Don't worry. Roy has the perfect solution. The WiFi SniFi is a compact, easy-to-build device that can "sniff" out wireless networks and display the appropriate packet information.

The WiFi SniFi, which I pronounce "wiffy sniffy" for the fun of it, sniffs out 802.11b wireless networks and displays captured packet information (see [Photo 1](#)). This little device can remain quiet on the network or associate with an access point and act as a network node. In Monitor mode, the WiFi SniFi can listen to a specific channel or scan all of the channels.

The big news is that the WiFi SniFi performs these functions without a high-powered microprocessor. In fact, one of its main advantages is that it generates a large amount of functionality from its 8-bit, 5-MHz microcontroller. Wireless local area networks (WLANs) operate at much higher speeds than the microcontroller, but the WiFi SniFi manages to nab the all-important management frames. It even grabs some of the data frames.

Even if you don't want to nab 'n' grab WLAN frames, you might find some useful hardware and software nuggets in the WiFi SniFi that apply to other types of microcontroller designs. I created the WiFi SniFi as part of the 2002 Esprit de



Photo 1—The WiFi SniFi uses a PCMCIA card to interact with wireless networks, a small LCD to display packet contents, and an old mouse scroll wheel to get user inputs. An 8-bit, 5-MHz microcontroller runs the entire thing.

KORE contest, which was sponsored by NEC Electronics America, so the device is a demo that shows how to leverage microcontroller resources to achieve surprising results.

If you savor the kind of minimalist design techniques that squeeze the most out of system resources, you might appreciate this design's interfaces with its input switches, PCMCIA card, and serial EEPROM. You can easily reuse the software for the PCMCIA interface in applications that don't have a suitable PCMCIA interface in hardware, even if you're working with something other than a wireless LAN card.

The design's user interface may require some adaptation for other applications, but it could be useful in a wide range of designs. The user interface consists of a scroll wheel taken from a mouse and a single push button. Roll the wheel to scroll through menu items and press it to select items. Use a separate push button to clear entries. The interface is simple, compact, and easy to use.

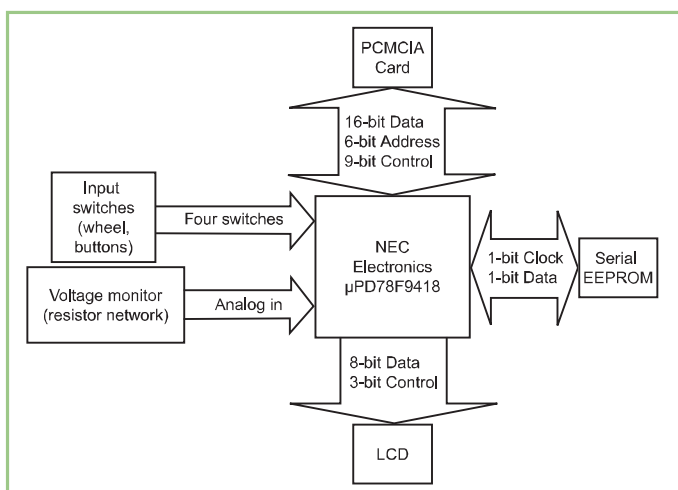


Figure 1—The WiFi SniFi consists of few components other than the 8-bit microcontroller, which implements the PCMCIA interface in software. The microcontroller includes a hardware LCD controller/driver. Some of the LCD I/Os are shared with other devices.

WHAT'S INSIDE?

The WiFi SniFi couldn't be simpler (see [Figure 1](#)). Most of the necessary resources are inside the NEC Electronics μ PD78F9418 microcontroller. The only major external components include a

4 × 20 LCD, a 128-Kb EEPROM that's used for saving captured WLAN frames and device configuration data, the mouse wheel, and an Intersil Prism 2-based PCMCIA WLAN card. These are on a board I built that attaches to an NEC KORE9418 development board (see [Figure 2](#)).

You can deliver power to the WiFi SniFi with batteries, an AC adapter, or a car cigarette lighter adapter. Either of the latter two power sources will charge the batteries. The voltage monitor indicated in [Figure 1](#) is used for monitoring the battery's charge state.

The WLAN card acts as a basic 802.11b node. I designed the WiFi SniFi without formal documentation for this card, but you can get information about it from the various open-source device drivers available for it. Refer to the Resources section of this article for a few useful links.

I had a difficult time finding detailed PCMCIA information on the 'Net, but I obtained a lot of useful information from F. Imdad-Haque's book, *Inside PC Card: CardBus and PCMCIA Design*. In addition, you can download the 802.11b specification from IEEE (www.ieee.org).

Important aspects of the microcontroller include its on-board 32-KB flash memory and 512-byte RAM. The application code resides in the on-board flash memory. Most significantly for this design, the microcontroller has 43 I/O pins, including multiple

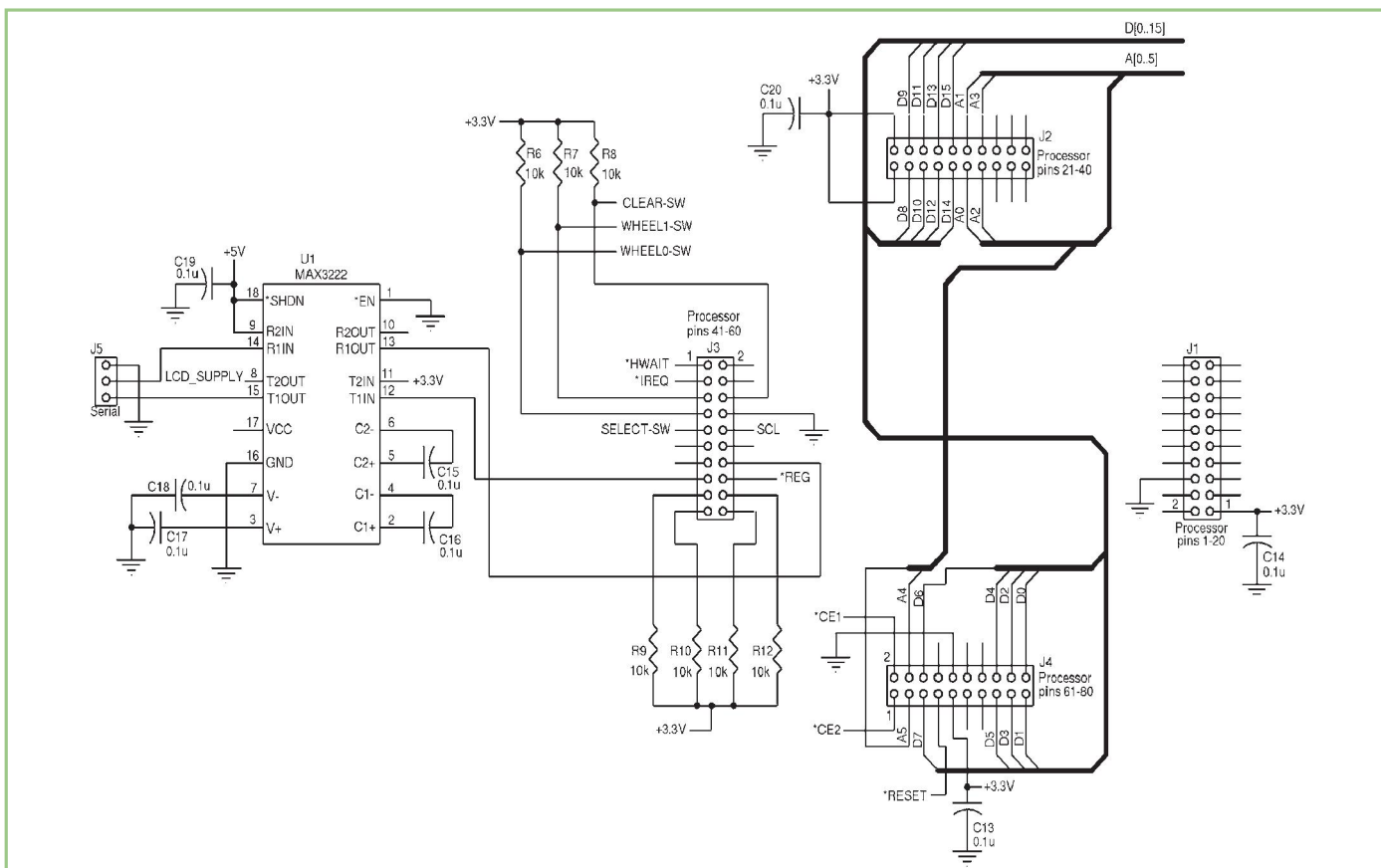


Figure 2—The WiFi SniFi board is designed to mate with the four 20-pin headers that the KORE development board provides. The MAX3222 is used for debugging serial output and providing a negative voltage source to drive the LCD.

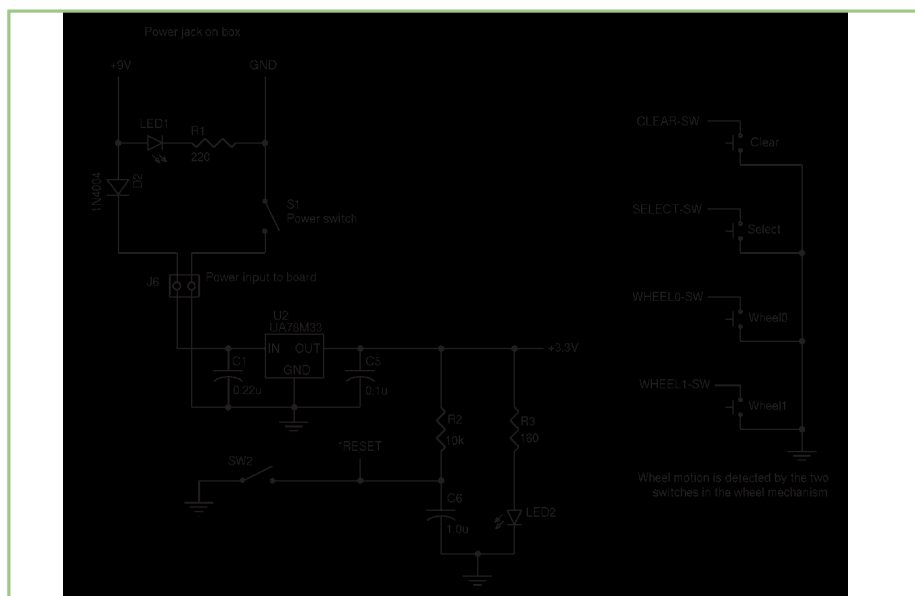


Figure 3—The linear regulator keeps the power design simple. The Clear switch is a separate switch, while the other three are part of the mouse scroll wheel.

A/D input channels. I used one of the A/D channels to monitor battery voltage. If you are used to working with microprocessors, the microcontroller presents a somewhat different interface task because it has no external data or address buses. Most of the I/Os on this particular microcontroller are general-purpose ports, which means that they adapt to different purposes under software control.

WIFI SNIFI CAPABILITIES

To understand what the WiFi Snifi does, you need to know a few facts about 802.11b-based WLANs. These wireless networks operate in the 2.4-GHz frequency range and offer a maximum physical-layer signaling rate of 11 Mbps. Although real-world networks achieve only 6 or 7 Mbps of throughput, the data rates are rather high for a 5-MHz system to handle. In fact, it is interesting to see how well the device keeps up, which probably has a lot to do with the large amount of buffering the wireless card performs. With the ability to buffer several dozen packets in its internal memory, the card greatly eases the timing constraints of interfacing to the microcontroller.

The 802.11b networks in the U.S. can use 11 channels. All but three channels overlap and interfere with one another,

so only three WLANs can operate unhindered in one area.

A WLAN can operate in either Infrastructure mode (BSS mode, where the network is controlled by an access point, or AP) or Ad Hoc mode (IBSS mode, where there is no access point). The service set identifier (SSID) is a string that identifies a WLAN. Stations only associate with an AP (or another station in Ad Hoc mode) that has the same SSID. When you turn on a station such as a notebook computer with a wireless card, the station scans all the channels checking for a beacon frame that has the station's SSID in it. This beacon frame identifies the channel and the address of the AP that the station is seeking. If the station finds no such

beacon frame, it cannot join any networks, so no wireless communication is possible.

The WiFi Snifi has two basic modes. It can find and monitor a wireless network by capturing management and data frames. Or it can associate with a network, display frames, and respond to pings and address resolution protocols, or ARPs, which are named for mapping an IP address to a physical machine address.

The WiFi Snifi won't give you Internet access, but it does enable you to poke about the landscape to find the WLANs. You can do so by placing the WiFi Snifi in Monitor mode and listening for frames. When frames come in, the WiFi Snifi displays them on the LCD and stores them in nonvolatile memory. You can configure the WiFi Snifi to listen on a specified channel or scan all of the channels by selecting channel zero via the user interface. Because of the overlapping channels, frames are often received while listening on a channel other than the one they were transmitted on. Most frames do not indicate which channel they were transmitted on, but management frames such as beacons include this information so the stations know which channel to use.

You can also control the types of frames that the system displays and saves. Because data frames tend to be fairly large, the WiFi Snifi can display only a portion of their content on the 4 × 20 LCD. Similarly, the EEPROM stores only part of each data frame. The type of

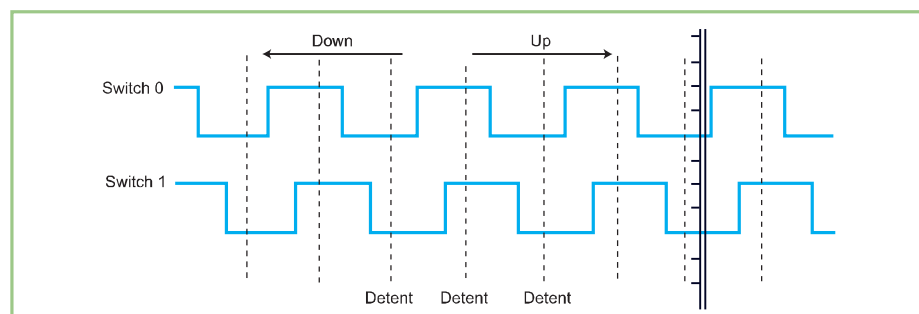


Figure 4—As you rotate the scroll wheel, two switches open and close to generate the waveforms. Detents are stopping points on the wheel that you can feel as you rotate it, and both switches are in the high or low state at each detent. If you analyze which waveform goes high or low first, you can tell which way the wheel is rotating. By detecting wheel-up and wheel-down events, software can iterate through menu items and packet listings on the LCD.

FROM THE ARCHIVES

information displayed varies based on the type of frame, because not all of the fields are present in every frame type.

The WiFi Snifi is best adapted for capturing management frames, which are generally small. The system can save and display most of the information in these frames. Management frames include beacons, probe requests/responses, association requests/responses, reassociation requests/responses, disassociation, authentication, and deauthentication. Management frames are often the most interesting from a network-troubleshooting point of view because they control the process of joining and leaving networks.

When it's monitoring, the WiFi Snifi displays the current saved frame number in the upper right corner of the LCD. The frame number increments until the EEPROM is full (containing 255 saved frames). Then, the WiFi Snifi continues to display frames without saving them. Use the Clear Captured Frames configuration menu option to clear the EEPROM. You can set the WiFi Snifi to capture or

ignore a specific type of frame, including contention-free control frames (CF-Ack, CF-Poll, CF-Ack+CF-Poll, CF-End, CF-End+CF-Ack).

To join a network with the WiFi Snifi, you must configure two fields in the configuration menu: the SSID, which is a string that identifies stations that are logically on the same network, and the IP address. Then, you can select the network node mode in the configuration menu.

If the WiFi Snifi successfully associates with the AP, the "link status change: connected" message will appear on the LCD. At that point, the WiFi Snifi is ready to respond to ARPs and pings. The system displays but does not capture the received frames. Also, note that the WiFi Snifi does not support the WEP encryption that is increasingly used to secure 802.11b WLANs.

The WiFi Snifi does not need a netmask to associate with a WLAN because the device does not initiate network traffic. The WiFi Snifi has no concept of a

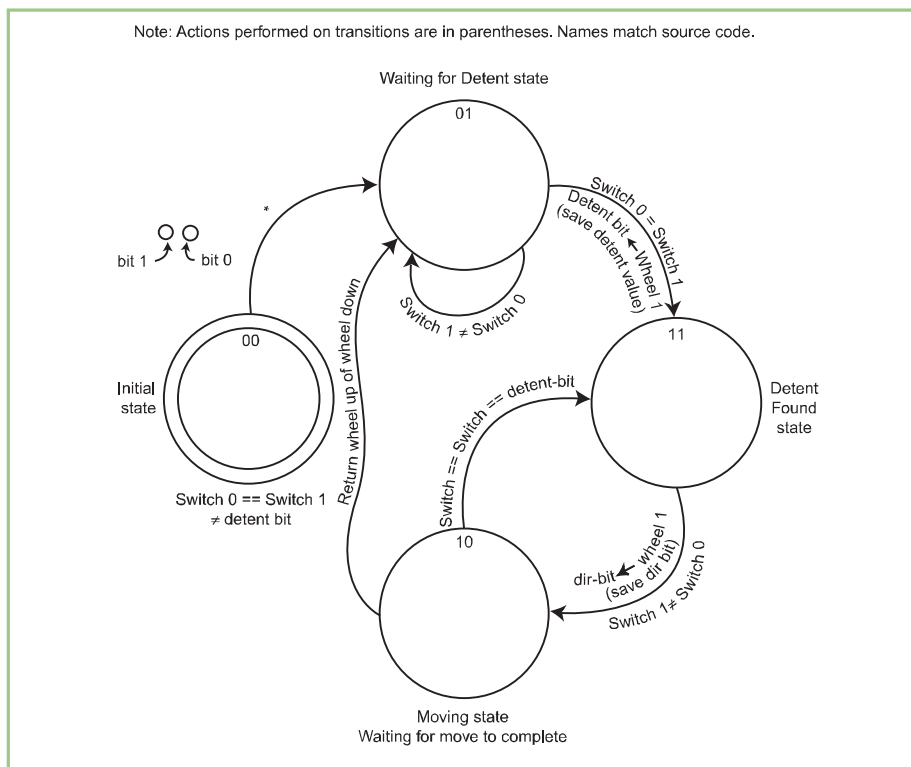


Figure 5—The state diagram describes the way software handles inputs from the scroll wheel. As soon as the software state machine detects a valid wheel-up or wheel-down event, the software looks for the next event.

HUMANDATA

FPGA /CPLD Boards from JAPAN

SAVING COST-TIME with readily available FPGA boards

- Basic and simple features, single power supply operation
- Same board size and connector layout - ACM/XCM
- All stocked items are ready to be shipped immediately
- Over 100 varieties of FPGA/CPLD boards are available
- Free download technical documents before purchasing

PLCC68 series

- FPGA Module IC socket mountable
- 3.3V single power supply
- Very small size (25.3 x 25.3 [mm])

XP68-03 Spartan-6 PLCC68 FPGA Module

Spartan-6 PLCC 68
XC6SLX45-2CSG324C
 16Mbit Configuration Device
 Two User LEDs
 One User Switch(Slide)
 RoHS compliant

AP68-04 Cyclone III PLCC68 FPGA Module

Cyclone III PLCC 68
EP3C25U256C8N
 16Mbit Configuration Device
 Two User LEDs
 One User Switch(Slide)
 RoHS compliant

ALTERA FPGA Board

Cyclone IV GX F484 FPGA board ACM-024 series

Cyclone IV GX **DDR2** **SIF40**

EP4CGX50CF23C8N
EP4CGX75CF23C8N
EP4CGX110CF23C8N
EP4CGX150CF23C7N
 Credit card size (86 x 54 mm)
 RoHS compliant

Cyclone IV GX F484 FPGA board ACM-108 series

Cyclone IV GX **DDR2**

EP4CGX50CF23C8N
EP4CGX110CF23C8N
EP4CGX150CF23C7N
 Compact size (43 x 54 mm)
 RoHS compliant

XILINX FPGA Board

Virtex-5 LXT FFG665 FPGA board XCM-017 series

Virtex-5 **SDRAM** **RocketIO** **SIF40**

XC5VLX30T-1FFG665C
XC5VLX50T-1FFG665C
 Credit card size (86 x 54 mm)
 RoHS compliant

Spartan-6 LXT FGG484 FPGA board XCM-111 series

Spartan-6 **DDR2** **RocketIO**

XC6SLX45T-2FFG484C
XC6SLX75T-2FFG484C
XC6SLX100T-2FFG484C
XC6SLX150T-2FFG484C
 Compact size (43 x 54 mm)
 RoHS compliant

Universal Board (Type2)

ZKB-106

- One for general power (3.3V 3A max) and the Two variable outputs for Vccio (0.8V to 3.3, 3A max)
- For ACM/XCM-2 series FPGA boards
- Power Switch and LED
- Power input: DCSV/2.1 [mm] Jack/ Terminal Block (option)
- Board size : 156x184 [mm]
- 4 Layers PCB, Thru-hole

www.hdl.co.jp/CC/
HuMANDATA LTD.
 E-mail: s2@hdl.co.jp
 Fax: 81-72-620-2003

FROM THE ARCHIVES

local versus nonlocal IP address. When a frame is received, the WiFi Sniffi responds to the source MAC and source IP addresses within that frame.

USING THE WIFI SNIFFI

Before getting into how some of the WiFi Sniffi's features work, let's look at how you can use them. The WiFi Sniffi has a simple interface consisting of a Clear button and the ex-mouse scroll wheel. Pressing the scroll wheel activates a switch that is used as a Select button.

If you press the Select button in normal operation, you'll enter the WiFi Sniffi configuration menu. Then, you can roll the wheel up and down to cycle through the configuration menu selections. The top line of the LCD displays the current configuration item, and the bottom three lines display a short help message. Pressing the scroll wheel selects the currently displayed menu option. When you select an item, you can edit its value.

The way you edit menu items depends on the context. Sometimes you select a value from several fixed choices. Other times you need to enter a value, such as the SSID or IP address, in which case, you must change the value of a character by rolling the wheel up or down.

Pressing the Select button moves the cursor one character to the right, and the Clear button moves left. If you press the Clear button when the cursor is all the way to the left, you'll cancel the editing of the field. Pressing the Select button while in the right-most position accepts the displayed value. To stop SSID editing, scroll to a space character and press the Select button.

During normal network node or monitor operation, the Clear button clears the LCD, leaving it blank until the WiFi Sniffi has new information to display. When you browse captured frames, the wheel selects the frame to be displayed. Pressing the Select button toggles between two screens of frame information, although some frames have only one screen.

Because the LCD is small, the screen displays frame information in a compact

Listing 1—The code that implements the state machine shown in Figure 5 uses polling to obtain the condition of the scroll wheel switches. The code uses an interrupt for the WiFi Sniffi's Select button. It's the only interrupt in this design.

```
//Get button (and switch) state, debounce inputs
bs = pollButtons(); //bs.WHEEL0 bs.WHEEL1 are the
                    //wheel switch inputs
//State machine for scroll wheel (four possible states)
if (GS_WHEEL_STATE_1_BIT)
{
    if (GS_WHEEL_STATE_0_BIT)
    {
        //State 11: Detent Found, waiting for move, which is indicated by
        //the two wheel switches having different values
        if (bs.WHEEL1 != bs.WHEEL0)
        {
            GS_WHEEL_STATE_0_BIT = 0; //Go to state 10
            GS_WHEEL_DIR_BIT = bs.WHEEL1; //Save direction
        }
    }
    else
    {
        //State 10: Moving and waiting for move to complete. Move is complete
        //when both wheel switches are the same value (i.e., you have reached
        //another detent), and this value is different from the previous detent
        //value. If it's equal to the previous detent, you've either missed some
        //events or the user only slightly moved the wheel, and let it return to
        //the original detent.
        if (bs.WHEEL1 == bs.WHEEL0)
        {
            //Either go to Waiting for Detent (if you detect a move) or waiting
            //for a move (if you're back at the original detent)
            GS_WHEEL_STATE_0_BIT = 1;
            if (!(bs.WHEEL1 == GS_WHEEL_DETENT_BIT))
            {
                //You've moved, so take action, and set the next state to Waiting
                //for Detent. Even though you're at a detent, you will likely have work
                //to do, which will delay your next poll of the switch states. If the
                //wheel is being scrolled quickly, you may miss many transitions.
                GS_WHEEL_STATE_1_BIT = 0; //Go to state 01
                //Move complete. Action!
                if (GS_WHEEL_DETENT_BIT == GS_WHEEL_DIR_BIT)
                    return(BA_wheelUp); //Wheel up
                else
                    return(BA_wheelDown); //Wheel down, or else
                    go to state 11
            }
        }
    }
}
else
{
    if (GS_WHEEL_STATE_0_BIT)
    {
        //State 01: Waiting for Detent. When wheel is in a detent
        //position, both wheel switches have the same value.
        if (bs.WHEEL1 == bs.WHEEL0)
        {
            GS_WHEEL_STATE_1_BIT = 1; //Go to state 11 and
            //save detent value
            GS_WHEEL_DETENT_BIT = bs.WHEEL1;
        }
    }
    else
    {
        //State 00: initial state of state machine. This state is not nec-
        //essary, but it's implemented so that all possible state values are
        //handled properly. It's used at startup because the global state vari-
        //able is initialized to zero, and this way you don't have to initialize
        //it to a special value. Go to state 01 Waiting for Detent.
        GS_WHEEL_STATE_0_BIT = 1;
    }
}
return(BA_noAction); //If you make it here, nothing has happened.
```


FROM THE ARCHIVES

manner, using abbreviations for many fields. You may download a list of these abbreviations and other terms from the *Circuit Cellar* FTP site. They may prove useful for understanding 802.11b-type networks.

In Monitor mode, the top line of the LCD displays the same information for all frame types: the type of frame, the channel that the card was configured for when the frame was received, the signal strength, and the frame number in the captured frame buffer. As you can see in Photo 1, I labeled the fields above the LCD. The information displayed in the remaining three lines of the LCD depends on the type of frame that's involved. If you configure the WiFi Sniffi to scan channels, it displays the channel it is listening to each time the channel changes.

In network node mode, the LCD displays ARP and ping frames as they are received. The WiFi Sniffi does not save these frames to the captured frame

buffer. When the status of the wireless connection changes (i.e., when the WLAN card associates or disassociates with an AP), the WiFi Sniffi displays a message indicating the new status.

THE MOUSE WHEEL

The mouse scroll wheel handles almost all of the WiFi Sniffi's input needs. Only one other switch is needed to act as the clear input. The wheel is also extremely easy to design in.

Rotating the wheel opens and closes two switches (see Figure 3). The wheel has small detents at regular intervals that you can feel as you rotate the wheel. Both switches are in the same state at each detent (both open or both closed). In this implementation, I tied one input of each switch low. I connected the other switch input to the microcontroller input and pulled it high with a resistor. As a result, the I/O line is high when the switch is open and low when the switch is closed.

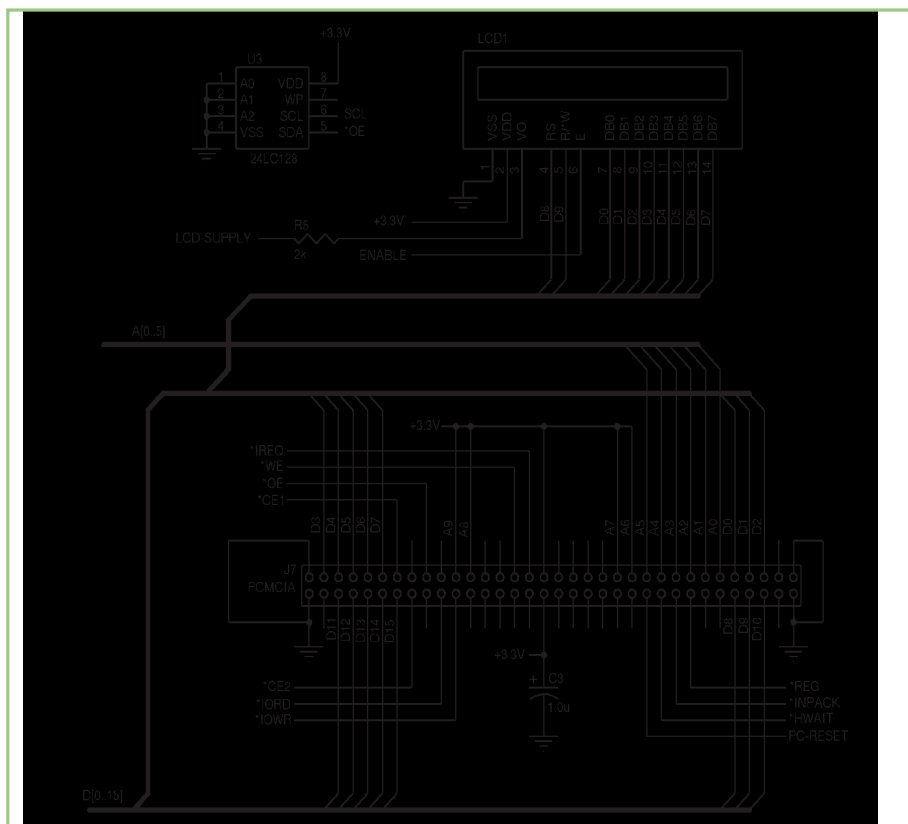
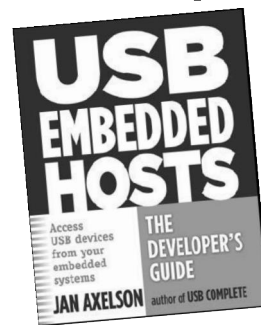


Figure 6—The PCMCIA interface connects directly to the μ PD78F9418's I/O pins. The data bus lines are used for both the PCMCIA interface and the LCD, which works because they both have separate enable signals.

Access USB Devices from your Embedded Systems



USB Embedded Hosts The Developer's Guide Jan Axelson

\$29.95

LVR.COM

From the author of *USB Complete*

PIC-SERVO MOTION CONTROL

MOTION CONTROLLERS FOR
BRUSH, BRUSHLESS AND
STEPPER MOTORS.

- controller chips
- controller boards

www.picservo.com

JEFFREY KERR, LLC

Connect With Design Engineers From Around The Globe.

Reserve advertising
space in *Circuit
Cellar* and *CC
News Notes* today!

Strategic Media Marketing
978.281.7708
peter@smmarketing.us
www.smmarketing.us

FROM THE ARCHIVES

The switches open and close at different wheel positions; therefore, if you rotate the wheel at a constant speed, the switches produce the waveforms shown in [Figure 4](#). These out-of-phase waveforms allow you to tell which way the wheel is rotating. From the detent marked “Up,” for instance, you’ll know the wheel is rotating upwards if switch 0 goes high before switch 1.

[Figure 5](#) describes the way I dealt with the wheel states and transitions. Note that the 2-bit values for each state keep track of whether the state machine has found a detent or is waiting for one; these values do not reflect the values of the wheel switches.

Aside from the initial state, the state machine includes three operating states. The first state, Detent Found, involves waiting for a move (11). When you enter this state, you save the value of the switches, so you can tell whether or not a complete move has occurred. The value also helps detect missed transitions. After exiting the state, save one of the switch values so you know which way the wheel is moving.

The second state, Moving, entails waiting for a move to end (10). A move is complete when you reach another detent. If you return to the same detent, you might have only rocked the wheel slightly, or you could have missed several transitions. If you reach another detent and a move is complete, return the correct action code and transition to the third state, Waiting for Detent. Even though you reach a detent, the calling code may do something as a result of returning an action code. So, to be on the safe side, look for a new detent when you are called again.

The Waiting for Detent state involves waiting for both switches to have the same value. When they do, transition to the Detent Found state. Even if you find a detent when you leave a state (10), you need to wait for another detent because a significant amount of time may have passed since you left the previous state. Finally, note that the initial state (00) allows the state machine to operate properly with the state variable initialized to zero.

The state machine tracks changes in a way that includes error checking. The machine waits for a detent after detecting a move to make sure the move ended. After a move, if the wheel ends up back in its previous state, the state machine returns no action. If the move generates a different value, the machine returns a wheel-up or wheel-down event so software can appropriately iterate through the menu items.

Software polls the wheel switches to detect changes, and the polling must occur often to keep up with rapid wheel movements. The wheel’s Select switch activates an interrupt. [Listing 1](#) shows the code associated with the scroll wheel.

Listing 2—A direct interface between an 8-bit microcontroller and a PCMCIA card is a rare thing, especially when the interface does not use any PCMCIA byte operations. The code shown here implements the entire interface using the microcontroller’s general-purpose I/Os.

```
//Perform a 16-bit read of I/O space @param addr address to
read from @return value read from card
__callt norec unsigned int ioRead(unsigned char addr)
{
    unsigned int temp;
    //Configure K0 data port to Input mode
    DATA_H_PORT_MODE = INPUT_PORT;
    DATA_L_PORT_MODE = INPUT_PORT;
    //Set up address
    ADDR_0_3_PORT = addr;
    ADDR_4_5_PORT = (ADDR_4_5_PORT & ~ADDR_4_5_PORT_MASK) |
    ((addr>>4) & ADDR_4_5_PORT_MASK);
    //Move these to one register. REG- could be an OR of CE1-
    and CE2-, as REG is low during both I/O cycles and attribute
    memory cycles. Can REG- be tied low? Reg, ce* can be set low
    at the same time the address is presented.
    hreg, ce1, ce2 low    #if 0
    //CE1- and CE2- signals are always low, so you only perform
    16-bit accesses
    CE_1_PORT_BIT = 0;
    CE_2_PORT_BIT = 0;
    #endif
    REG_PORT_BIT = 0;           //REG- may also be able to
                                be tied low, not checked
    //Set IO read low.
    IORD_PORT_BIT = 0;

    #ifndef PCMCIA_SIM
    while (!HWAIT_PORT_BIT);    //Wait for hwait- high
    #else
    NOP();
    NOP();
    NOP();
    #endif
    //Read data
    temp = DATA_H_PORT;
    temp = temp << 8;
    temp |= DATA_L_PORT;
    IORD_PORT_BIT = 1;
    //reg/ce1/ce2 high
    #if 0
    CE_1_PORT_BIT = 1;
    CE_2_PORT_BIT = 1;
    #endif
    REG_PORT_BIT = 1;
    return(temp);
}
```

PCMCIA INTERFACE

The microcontroller in the WiFi Snifi interfaces to the WLAN PCMCIA card via the microcontroller’s general-purpose I/Os (see [Figure 6](#)). The same is true for the I2C interface to the serial EEPROM. Therefore, I implemented these interfaces in software. [Listing 2](#) shows the code for the PCMCIA interface. The code is suitable for reuse in any application that involves a PCMCIA card.

You may download a list of the signals in the PCMCIA interface from the *Circuit Cellar* FTP site. The PCMCIA interface is asynchronous, so the microcontroller works well as the timing master for the I/O bus. After looking at the PCMCIA timing diagrams, I initially thought that I wouldn’t need delays in the read

and write routines, but I found it necessary to poll the *HWAIT line. The polling loop probably exits after the first time the *HWAIT signal is polled, because putting a few NOPs in place of the polling loop also worked reliably.

To put the card into I/O mode, you need to set the COR register appropriately, because all PCMCIA cards power up by default in Memory Only mode. Because I had hard-coded the COR register address, I did not need to read the information from the attribute memory to determine its location. To put the card in I/O mode, however, I found that I had to read the COR register before writing it. As a result, the OE signal had to be connected and used for the attribute memory read operation.

I also discovered during the course of development that I did not need to change the signals (*CE1 and *CE2) that control whether a bus access is for the high byte, low byte, or both. I can tie the signals low because I am always performing 16-bit accesses. Even though attribute memory is only 8 bits wide, 16-bit accesses ignore the upper 8 bits.

Implementing an 8-bit PCMCIA interface would have reduced signal count only minimally and would have complicated the development. The WLAN card is documented to support 8-bit interfacing, but I could not find information on implementing the 8-bit option. Many of the card's internal buffers use autoincrementing of 16-bit addresses, so 8-bit operations on the registers would raise special cases. Implementing an 8-bit interface also requires the use of *CE1, *CE2, and A0, which is always zero because all of the accesses are 16 bits. Thus, the total interface width would only decrease by 5 bits.

Although the WLAN card decodes 10 address lines, six suffice for accessing all of the addresses required to operate the card, with the exception of the COR register in attribute memory. Because the register access is the only operation that needs the top four address bits, I tied the lines to the values required to access the COR register. Even though I connected the A0 line to the microcontroller, the connection is unnecessary because the 16-bit-only accesses make all of the addresses even.

If you want to support generic PCMCIA cards, you'll need more address lines to parse the card information structure (CIS) in attribute memory so you can determine the card capabilities and the COR register location. For the WiFi SniFi, I determined the COR register address in the Linux development environment and hard-coded the address in the WiFi SniFi implementation.

The PCMCIA interface's high pin count did not pose a big problem, because I reused most of the signals for interfacing to other peripherals. For instance, 10 of the 11 bits in the LCD interface are shared with the PCMCIA data bus. Polling the PCMCIA card eases the resource sharing. You could still share interface pins in an interrupt-driven design, but you would have to handle the sharing with greater care.

The WiFi SniFi uses only one interrupt line, and it's for the Select button. I wanted the button to be responsive to user inputs without continual high-frequency polling. That leaves the main loop free to continuously poll the WLAN card, and it does

so at a high enough frequency to keep up fairly well with the packets.

The other switch inputs—the wheel and Clear—are polled fast enough to be responsive after the WiFi SniFi is in Menu mode. Therefore, they do not need to be interrupt-based.

EXPANSION POSSIBILITIES

Every design leaves you wishing you could do more. For the WiFi SniFi, it would be nice to include a simple stateless TCP Internet server for downloading captured frames to a PC. Additionally, you could use the microcontroller's serial port for interfacing to a GPS receiver, so the WiFi SniFi could automatically record the location of wireless networks.

The WiFi SniFi's power management could be improved with a switching regulator and microcontroller-managed power supply for the PCMCIA card. (The design's linear regulator is simple but inefficient.) The PCMCIA card is by far the system's largest power drain, so turning off the card while browsing captured frames would greatly improve battery life.

In a similar vein, the WiFi SniFi could use a more sophisticated charging circuit. The microcontroller's analog input could monitor battery temperature with a thermistor (along with the voltage monitoring currently implemented) to enable faster charging.

In addition to specific WiFi SniFi improvements, the device's core functionality could be developed into a platform that allows for the addition of wireless networking to a variety of microcontroller-based designs. As the WiFi SniFi project proves, you can go wireless with a surprisingly small amount of hardware. 📡

Roy Franz earned a B.S. in Computer Science from the University of California, Davis. He's been developing software for embedded systems for more than 10 years. His technical interests include low-level code that interfaces with hardware as well as systems that interact with their environment. When time permits, Roy enjoys hiking and rollerblading. You may contact him at rfranz@beartreedesign.com.

PROJECT FILES

To download the code and additional files, go to ftp://circuitcellar.com/pub/Circuit_Cellar/2003/157.

RESOURCES

F. Imdad-Haque, *Inside PC Card: CardBus and PCMCIA Design*, Butterworth-Heinemann, Newton, MA, 1996.

Host AP driver information hostap.epitest.fi.

SOURCES

PRISM 2 WLAN Card

Intersil Corp. | www.intersil.com

μPD78F9418 Microcontroller

NEC USA, Inc. | www.necus.com



QR Coding for Engineers

Quick response (QR) codes are 2-D matrix barcodes with fast readability and large storage capacities. This article examines QR formats and describes how to implement a Windows PC program that formats your data into QR code symbols. Options such as error correction, masking, and other modifications are also discussed.

During the 1970s, the grocery industry's trade association was looking for a way for machines to read their newly defined numeric product identification data. IBM engineer George J. Laurer invented the 11-digit coding system (with an additional check digit) used for the first Universal Product Code (UPC) codes, UPC-A (see [Figure 1](#)). Individual patterns indicate a start, end, or decimal digit. Each pattern consists of seven equal-width spaces or lines, starting with at least one space followed by two transitions from spaces to lines. Consecutive spaces or lines are permitted. The check digit is calculated using the formula $10 - [\text{modulo } 10 (\text{total of odd digit values} \times 3) + (\text{total of even digit values})]$. The UPC is divided into three parts, a Manufacturer's code, a Product code, and a check digit. Naturally, other UPC specification versions followed to increase the data capacity and UPCs are still used today. Note: Patterns following the Manufacturer's code use an inverted (or one's complement) digit code.

You've undoubtedly had to find the UPC code if you've ever used a self checkout line. Laser scanners read the UPC patterns and translate them into a manufacturer/product code. That code is looked up in the store's database to determine the item's price. Eventually, a clamor arose for a machine-printable/readable

symbol with more information than a UPC symbol (or other linear bar codes) could handle. To expand this linear technology, the Codablock bar code symbol stacked multiple bar codes into a single symbol. While this was acceptable for the short term, the density wasn't there, and it was not used to any extent. However, it did get people thinking in two dimensions instead of one dimension, as presented in the linear UPC symbols.

ENCORE

One of the first 2-D codes I ran into was developed by United Parcel Service (UPS). The MaxiCode symbol held a Structured Carrier message containing key information about a package (see [Figure 2](#)). Right away you notice the bull's-eye pattern in the center used as a registration point. Using a structured carrier means the data doesn't have to be of any fixed format other than to fit the block. While

multiple symbols can be chained together to contain larger data (e.g., greater than 93 characters), the symbol itself is not expandable based on the message size.

A more significant 2-D format is the QR code. This format was invented by Toyota subsidiary Denso Wave for use in the automotive industry in the early 1990s. Its recent popularity is due to its fast readability and

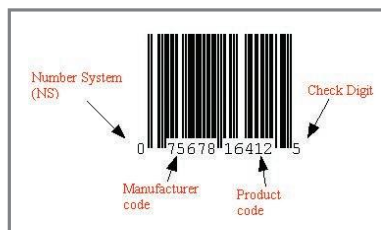


Figure 1—Thanks to the Universal Product Code (UPC) found on every product, a cashier no longer must manually enter prices at their register. This speeds up the sale and eliminates entry errors. (Image courtesy of Morovia Corp.)

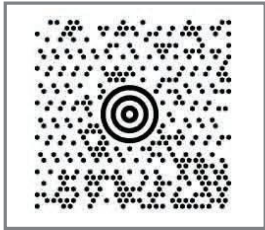


Figure 2—Who hasn't received a package from UPS with this coding symbol on it? Digital scales and specialized software enabled every shipping department to automate their shipping departments. (Image courtesy of Wikipedia)

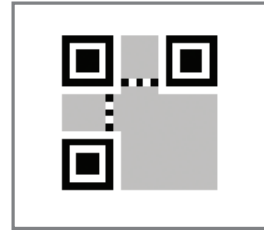


Figure 3—Essential orientation information is consistent in every QR symbol. This enables decoding applications to orient the symbol properly for data extrusion.

large storage capacity. From the get go, the design specifications required the format to be variable based on data size. At the maximum designed size, a QR code will carry more than 7,000 digits or almost 3,000 ASCII characters. Why the difference? A restricted character set enables data to be packed together more efficiently.

In this article, I'll examine all the pieces of a QR format and implement a program (using Liberty BASIC, which you can run on your Windows PC) that formats your own data into a QR code symbol. You can play around with some variables (e.g., error correction strength and masking) to see how these can be modified to make your symbol more robust. Let's begin with the skeleton that supports your data.

STRUCTURE

Providing an expandable structure to house data is an important task. With the wrong size house, the data either won't fit inside or you will have an abundance of wasted space. When you expand a house, you don't just make everything larger, you need to provide additional supporting elements to make the increase in volume both safe and habitable. The basic QR frame consists of a square matrix like a checkerboard where each square is either black or white. You can think of this as an array of elements that are either 1s or 0s. QR size is designated by version number. It begins with Version 1 and extends up to Version 40. The version number defines the matrix's size using the formula: $N = 4 \times \text{version} + 17$. This means a Version 1 array will have 21×21 elements and a Version 40 will have 177×177 elements. You can begin simulating this by setting up an array of binary information sized to your chosen version.

There are a few areas with this matrix that will always have a specific pattern. The first is the orientation pattern, a matrix of 7×7 elements forming a square bull's eye. Three identical bulls' eyes are placed in the upper left and right and the lower left corners. An additional band of white is used to separate the orientation patterns from the rest of the elements for a total orientation pattern of 8×8 elements. Next, there is a row and a column of timing marks that connect the inside corners of each bull's eye. **Figure 3** shows these items in black and white with all other (unaffected elements in gray) for a Version 1 symbol. Notice the large white area around the symbol. While not part of the matrix, the specifications suggest a clear border of four

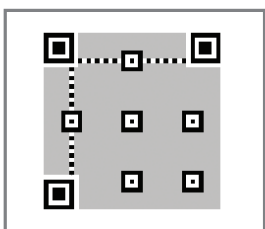


Figure 4—As a QR symbol grows in size, additional alignment patterns help decoding applications correctly interpret individual elements.

elements around the symbol to lessen the confusion of any surrounding marks being part of the symbol.

As the version size increases, the specifications call out for additional alignment marks to be placed within the symbol (see **Figure 4**). These are smaller 5×5 bulls' eyes and do not have the extra band of white around them as do the orientation patterns. These help estimate the location of individual elements by the decoding application. **Table 1** indicates how many of these alignment patterns are used and where they are positioned for each version symbol.

With the mechanism used to help figure out orientation and pixilation, two other areas of interest can be located. The first area is the format information consisting of the EDC level of error correction and the mask used to present the data. The

Version	Number of Alignment Patterns	Layout
1	0	O O O
2-6	1	O O O x
7-13	6	O x O x x x O x x
14-20	13	O x x O x x x x x x x x O x x x
21-27	22	O x x x O x x x x x x x x x x x x x x x O x x x x
28-34	33	O x x x x O x O x x x x x
35-40	46	O x x x x x O x O x x x x x x

Table 1—Additional alignment patterns are strategically placed in the symbol as the version size grows. This table shows the first in the fourth quadrant aligned with the inside edges of the orientation marks and additional alignment patterns centered and in rows and columns between the orientation marks. These patterns aid in the estimation of the element centroids by the decoding application.

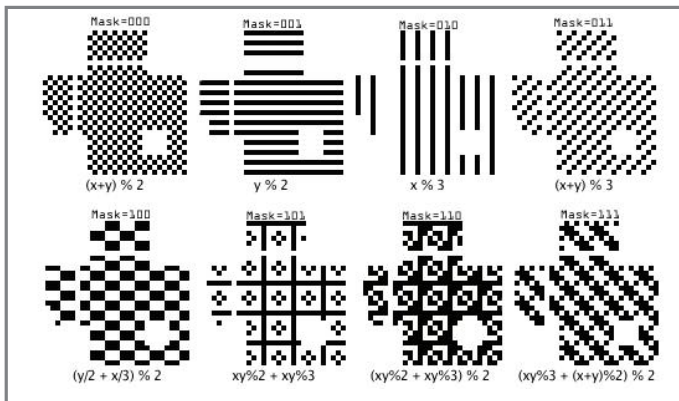


Figure 5—Eight masking patterns can be chosen on a basis of presenting data in a way that the message can be most easily and successfully revealed.

EDC level is a 2-bit value where b"00"=M (15%), b"01"=L (7%), b"10"=H (30%), and b"11"=Q (25%). Your data and the EDC data share a fixed amount of space in any particular version size. Therefore, choosing an EDC level with better correction capabilities results in less space for the actual data and may require using a larger version. One of eight potential masking patterns will alter the way data is presented. Displaying the same data using different masks can change the displayed pattern's overall look. Imagine if the data looked like alignment marks! This could be confusing to the decoding application. Masking the data would change the way it looks. However, as long as you include what mask was used you can reveal the true data (by unmasking it). **Figure 5** shows these different masking patterns.

The 2-bit EDC-level digits along with the 3-bit mask digits make up the first 5 bits of the 15-bit format. The format's remaining 10 bits are error-correcting data for the upper 5 bits. With only 2^5 (32) possibilities, a look-up table can be used to determine the appropriate code (precalculated using BCH ECC and masked with its own special mask). I'll discuss masking in more detail later in this article. The 15-bit format is wrapped around the upper left orientation mark running along the bottom and up the right side of the mark. To improve the potential of recovery, this is repeated again running up the right side of the lower orientation mark and continuing along the bottom of the upper right orientation mark.

The last area of interest is the version code, which is only presented in QR symbols that are larger than Version 6. The 6-bit version data and 12 bits of error-correcting data are arranged in a 3×6 block along the upper left-hand side of the upper right orientation mark and again (mirror image and perpendicular) along the top of the lower orientation mark. **Figure 6** shows the placement of format data (red) and the version data (blue) within the QR symbol.

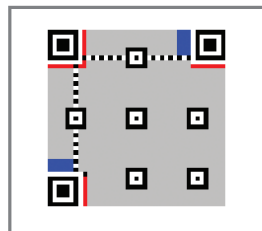


Figure 6—Version and format data include their own error-correction bits and are placed twice within the QR code symbol to improve recovery rates.

Figure 7 shows all the information data or I-space (black) I just discussed, including orientation, timing, alignment, format, and version. This information mask must remain untouched by the data (or the mask used to improve the readability of the data), which will be placed into the data area or D-space. Note: There is one pixel that has been added to the I-space, it can be seen just above the first 6 bits of the format running up the right-hand side of the lower orientation mark in **Figure 6**. This is an odd pixel and should remain as a "1."

ARE WE THERE YET?

Just when you thought it was time to add the data to the D-space, I need to stop and add another level of complication to all of this. The size of the D-space, once the I-space is removed, is a fixed number of elements or bits depending on the version size. **Table 2** shows the number of bits for each. The data inserted into the D-space comes from a long stream of self-describing data (SDD). The SDD is of the type/length/value (TLV) format. The type describes what kind of data follows. The length indicates the number of data pieces. The data includes not only the actual data, but filler and also the error-correcting data. All of this is one long string of bits that fills in the D-space. I should mention that the SDD likes to keep things in relatively small blocks. This means the long bitstream is made of multiple blocks. It may make more sense to look at how I used ShopTalk Systems's Liberty BASIC programming software to implement this. I'll begin with the I-space.

Implementing this application to create a QR symbol begins with a bit of housekeeping. However, not much can be done until the user enters some data for the QR symbol to represent. Note, a number of menu bar selections are presented across the top of the application window (just above the information boxes) shown in **Photo 1**. Clicking on the Data tab brings up a pop-up data entry form for your data. Data can be packed when it conforms to some special rules. The data might be a telephone

Version	Numeric Data 0–9	Alphanumeric 0–9 A–Z plus \$ % * + - / : <space>	Binary 8 bits
	3 characters	2 characters	1 character
1–9	10 bits	9 bits	8 bits
10–26	12 bits	11 bits	16 bits
27–40	14 bits	13 bits	16 bits

Table 2—Some data types are packed using multiple characters. This enables more characters to be fit into the available space. Therefore, the number of characters that can fit into a symbol is based on the data type. A fixed number of bits (describing the length) is expected (based on the maximum possible number).

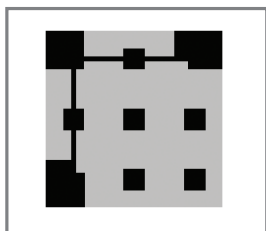


Figure 7—All the information thus far is in support of data recovery. I call this the "I-space." It is not affected by any masking done on the remaining D-space.

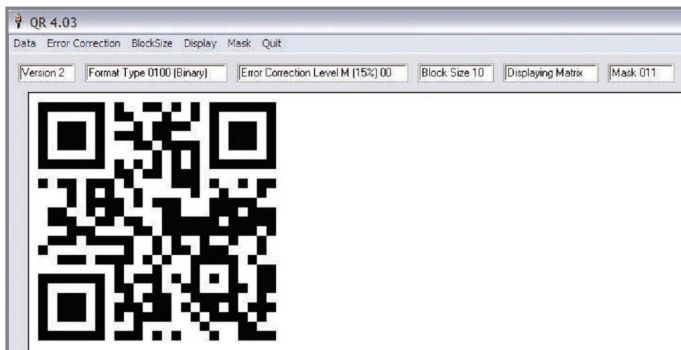


Photo 1—This application enables you to change parameters (e.g., ECC level and masking) to see how these affect the symbol's "look."

number (i.e., numeric), or a website address (i.e., alphanumeric), or even binary (e.g., ASCII) data. Your typed data gets entered into the string `EnteredString$`. For this discussion, I'll be using my website (WWW.IMAGINETHATNOW.COM) as `EnteredString$`, the length of which is 22, stored into the variable `DataLength`. I search this string looking for any non-numeric or alphanumeric characters to determine what format to use. I assign the variable `fVar$`="0001" for numeric, "0010" for alphanumeric, or "0100" for binary. The new format is updated to the screen.

Note: Throughout this application, I'll be storing data in strings in one of two ways, in binary string format (using only the characters 0x30 "0" and 0x31 "1"), and in character format (as a byte 0x00-0xFF). The binary format enables you to see the data (as a 1s and 0s) when printed.

The application defaults to an error-correction level of "L" (15%). You can use the Error Correction tab to choose a higher level, if you wish. This sets `EDCLevelBlock$`="01" for level "L," "00" for level "M," "11" for level "Q," or "10" for level "H." The new EDC level is updated to the screen. This is the minimum information necessary to begin building a QR symbol. To find out what version (i.e., size) QR symbol is required, begin by packing the data if necessary.

The SDD's first 4 bits describe the type of data that is to follow. While these 4 bits can define up to 16 data types, for this discussion, I will touch on three of the four native kinds, numeric "0001," alphanumeric "0010," and binary "0100." I'm going to use the string `DataString$` to hold the SDD and initialize it with the type of data. Since I entered my website's URL in uppercase characters, this is considered alphanumeric data and `DataString$`="0010".

Next I add the data length, in characters using `DataLength`, to the SDD. This must be done using the appropriate number of bits according to the data type. This can become confusing because the available space is fixed, but packed numeric or alphanumeric data takes up less room than binary. So the number of bits it takes to define the data's potential maximum length is different for each data type. Table 2 shows how this breaks down. Note: 9 bits (512 alphanumeric characters) is the smallest number of bits for alphanumeric data. If it is less than 512 (and it is, 22) then 9 bits are used to represent the length. $22 = b"10110"$, but I need to pad this to 9-bits, "000010110". Append this onto `DataString$` and it is now "0010000010110".

And now comes the actual data! Alphanumeric data is packed into 11-bit chunks. The alphanumeric character set has been redefined because it has only 45 characters. Multiply the first character's value by 45 and add it to the second character's value. `EnteredString$` now appends what is shown in Table 3. If this is not divisible by 8, then pad the end with "0s." In this case, I need to append "00" for a total of 136 bits or 17 bytes. The 17 bytes is the magic number I will use to determine version size.

Since Liberty BASIC has good string manipulation, I use strings and arrays of strings to simplify the programming. There are a lot of tables of information in this application. Most are initialized at the beginning by reading data statements into them. These tables are easily viewed in the listing thanks to their data statement format. `Blocks(164,6)` is one the of largest tables used to hold information in this application. Table entries are indexed by version number (1–40) × ECL (0–3) and each entry holds six pieces of information, `Block1Count`, `Block1DataLength`, `Block1ECCWLength`, `Block2Count`, `Block2DataLength`, and `Block2ECCWLength`. I need to locate the appropriate table entry. This begins with selecting a level of error correction (ECL). The higher the level of error correction, the greater the proportion of total space set aside for the error correction code. For this example, I'll choose level "M," which can correct a QR symbol with about 15% of its data mangled. The (`Blocks`) table is now reduced to only 40 entries (the 40 version entries pertaining to the chosen ECC level.) To identify the version required, I need to compare my data requirement (17 bytes) to the potential capacity of each version, starting with Version 1. The version's data capacity is its $(\text{Block1Count} \times \text{Block1DataLength}) + (\text{Block2Count} \times \text{Block2DataLength})$. The first table entry (Version 1, level "M") is 1, 16, 10, 0, 0, 0. The capacity for this version is therefore $(1 \times 16) + (0 \times 0) = 16 + 0 = 16$. Since I require more data room than the 16-byte capacity of Version 1 (M), I move on to the next entry. The second table entry (Version 2, level "M") is 1, 28, 16, 0, 0, 0. The capacity for this version is therefore $(1 \times 28) + (0 \times 0) = 28 + 0 = 28$. This is capable of holding my 17 bytes, so I will use Version 2.

Note: This table also indicates the number of error-correcting bytes `Block1ECCWLength` and `Block2ECCWLength` for each version

WW	= 32 × 45 + 32	= 1440 + 32	= 1472	= b"01111000000"	= "01111000000"
W.	= 32 × 45 + 42	= 1440 + 42	= 1482	= b"01111001010"	= "01111001010"
IM	= 18 × 45 + 22	= 810 + 22	= 832	= b"101000000"	= "01101000000"
AG	= 10 × 45 + 15	= 450 + 15	= 465	= b"110100001"	= "00111010001"
IN	= 18 × 45 + 23	= 810 + 23	= 833	= b"101000001"	= "01101000001"
ET	= 14 × 45 + 29	= 630 + 29	= 659	= b"1010010011"	= "01010010011"
HA	= 17 × 45 + 10	= 765 + 10	= 775	= b"100000111"	= "01100000111"
TN	= 29 × 45 + 23	= 1305 + 23	= 1328	= b"10100110000"	= "10100110000"
OW	= 24 × 45 + 32	= 1080 + 32	= 1112	= b"10001011000"	= "10001011000"
.C	= 42 × 45 + 12	= 1890 + 12	= 1902	= b"110110110110"	= "11101101110"
OM	= 24 × 45 + 22	= 1080 + 22	= 1102	= b"10001001110"	= "10001001110"
DataString\$="001000001011011100000010111001010011010000000 01110100010110100000101010010011011000001110100110000100010 1100011011011011010001001110". The packed length of DataString\$ =4+9+11+11+11+11+11+11+11+11+11+11=134.					

Table 3—The alphanumeric data is packed into 11-bit chunks.

Elektor.STORE

The world of electronics
at your fingertips!

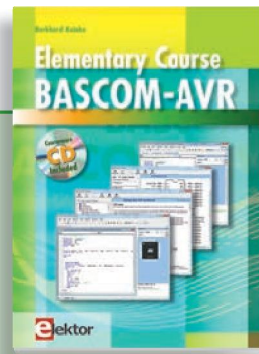


This book and more
are available at
www.elektor.com/books

Books Programming step-by-step **Android Apps**

This book is an introduction to programming apps for Android devices. The operation of the Android system is explained in a step by step way, aiming to show how personal applications can be programmed. A wide variety of applications is presented based on a solid number of hands-on examples, covering anything from simple math programs, reading sensors and GPS data, right up to programming for advanced Internet applications. Besides writing applications in the Java programming language, this book also explains how apps can be programmed using Javascript or PHP scripts. When it comes to personalizing your smartphone you should not feel limited to off the shelf applications because creating your own apps and programming Android devices is easier than you think!

244 pages • ISBN 978-1-907920-15-8 • \$56.40

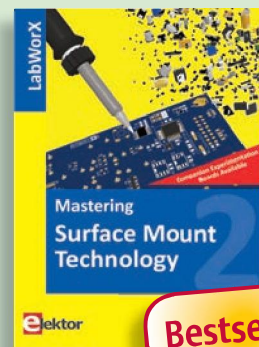


Free Software CD-ROM included

Elementary Course **BASCOM-AVR**

The Atmel AVR family of microcontrollers are extremely versatile and widely used. Elektor magazine already produced a wealth of special applications and circuit boards based on ATmega and ATtiny controllers. The majority of these projects perform a particular function. In this book however the programming of these controllers is the foremost concern. Using practical examples we show how, using BASCOM, you can quickly get your own design ideas up and running in silicon.

224 pages • ISBN 978-1-907920-11-0 • \$56.40



Bestseller!

LabWorX 2: Straight from the Lab to your Brain

Mastering **Surface Mount Technology**

This book takes you on a crash course in techniques, tips and knowhow to successfully introduce Surface Mount Technology in your workflow. Besides explaining methodology and equipment, attention is given to parts technology and soldering technique. Several projects introduce you step by step to handling surface mounted parts and the required technique to successfully build SMT assemblies. Many practical tips and tricks are disclosed that bring surface mounted technology into everyone's reach without breaking the bank.

282 pages • ISBN 978-1-907920-12-7 • \$47.60



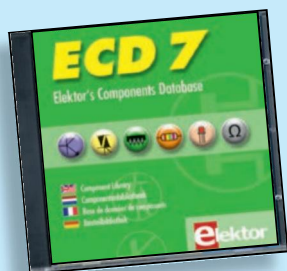
New!

140 Min. Video Presentation

DVD Masterclass Feedback in Audio Amplifiers

In this Masterclass we address several aspects of feedback in audio amplifiers. The focus, although not entirely math-free, is on providing insight and understanding of the issues involved. Presenter Jan Didden provides a clear overview of the benefits that can be obtained by feedback and its sibling, error correction; but also of its limitations and disadvantages. Recommended to all audio designers and serious audio hobbyists!

ISBN 978-1-907920-16-5 • \$40.20



More than 75,000 components

CD Elektor's Components Database 7

This CD-ROM gives you easy access to design data for over 11,100 ICs, 37,000 transistors, FETs, thyristors and triacs, 25,100 diodes and 2,000 optocouplers. The program package consists of eight databanks covering ICs, transistors, diodes and optocouplers. A further eleven applications cover the calculation of, for example, zener diode series resistors, voltage regulators, voltage dividers and AMV's. A colour band decoder is included for determining resistor and inductor values. All databank applications are fully interactive, allowing the user to add, edit and complete component data. This CD-ROM is a must-have for all electronics enthusiasts!

ISBN 978-90-5381-298-3 • \$40.20



Bestseller!

Embedded Linux Made Easy

Today Linux can be found running on all sorts of devices, even coffee machines. Many electronics enthusiasts will be keen to use Linux as the basis of a new microcontroller project, but the apparent complexity of the operating system and the high price of development boards has been a hurdle. Here Elektor solves both these problems, with a beginners' course accompanied by a compact and inexpensive populated and tested circuit board. This board includes everything necessary for a modern embedded project: a USB interface, an SD card connection and various other expansion options. It is also easy to hook the board up to an Ethernet network.

Populated and tested Elektor Linux Board

Art.# 120026-91 • \$93.30

Elektor is more than just your favorite electronics magazine. It's your one-stop shop for Elektor Books, CDs, DVDs, Kits & Modules and much more!

www.elektor.com/store



Elektor US
111 Founders Plaza, Suite 300
East Hartford, CT 06108
USA
Phone: 860-875-2199
Fax: 860-871-0411
E-mail: order@elektor.com

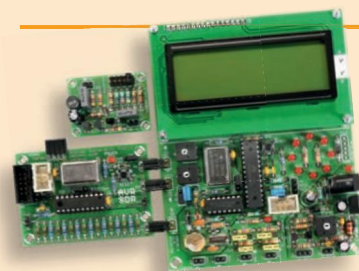


USB Isolator

If your USB device ever suffers from noise caused by an earth loop or if you want to protect your PC against external voltages then you need a USB isolator. The circuit described in Elektor's October 2012 edition offers an optimal electrical isolation of both the data lines as well as the supply lines between the PC and the USB device.

Populated and tested Board

Art.# 120291-91 • \$101.40



AVR Software Defined Radio

This package consists of the three boards associated with the AVR Software Defined Radio articles series in Elektor, which is built around practical experiments. The first board, which includes an ATtiny2313, a 20 MHz oscillator and an R-2R DAC, will be used to make a signal generator. The second board will fish signals out of the ether. It contains all the hardware needed to make a digital software-defined radio (SDR), with an RS-232 interface, an LCD panel, and a 20 MHz VCXO (voltage-controlled crystal oscillator), which can be locked to a reference signal. The third board provides an active ferrite antenna. This bundle also includes the assembled and tested FT232R USB/Serial Bridge/BOB PCB!

Signal Generator + Universal Receiver +
Active Antenna: PCBs and all components +
USB-FT232R breakout-board

Art.# 100182-72 • \$133.00

(I'll get to this shortly). While it's not evident by the first two entries, large data is divided up into multiple chunks (of two potential sizes, `Block1` and `Block2`) for easier processing.

Since my 17 bytes of data will not fill up the capacity of the Version 2 data space, I need to append my 17 bytes with special data bytes. "11101100" (0xEC) and "00010001" (0x11) are alternately appended until the length equals the capacity of the version used (here I add 11 special bytes). The `DataStream$` is now complete. But before I can create the QR code, I need to produce the error-correcting code that will go with my data.

ECC

Since the early days of computing, error correction has been used to restore or at least point out possible data errors. When retransmission of errant data isn't possible, it is necessary to include enough additional information to be able to reconstruct the original data. The Reed-Soloman algorithm used to detect and correct multiple random errors is based on a finite (i.e., Galois) field with a length of 256 (0–255). Error-correction data is derived from polynomial division between the data (message polynomial) and a generator polynomial. To eliminate the need for intense calculations, the application uses predetermined generator polynomials, selected from a table based on the `BlockECCWLength` I found earlier in the `Block()` table. In my case, `Block1ECCWLength = 16`, so I use entry 16 from the `GeneratorPolynomial$()` table. It is 16 bytes long (0x78, 0x68, 0x6B, 0x6D, 0x66, 0xA1, 0x4C, 0x03, 0x5B, 0xBF, 0x93, 0xA9, 0xB6, 0xC2, 0xE1, 0x78). For every 8-bits of data in `DataStream$,` I will perform a calculation that will result in a new generator polynomial. I will use 256-byte tables to perform log and antilog conversions, which enable me to use simple addition instead of division. Let's look at how this is implemented in BASIC.

It is important to remember that this routine must be performed on every data block to calculate each block's error-correction code. I only have one block to worry about in this example (see [Listing 1](#)).

`Block1Data$` and `Block1Code$` now contain the character string data which together will fill the D-space. Let's begin to build the QR symbol.

QR ARRAY

Version 2 has a matrix equal to $(\text{version} \times 4) + 17$, or $(2 \times 4) + 17 = 25 \times 25$ elements. To keep things as simple as possible, I will use two separate arrays `QR(26,26)` holding all the data for both I-space and D-space and `QRMask(26,26)` indicating which Space each element belongs to. Elements of `QRMask` array that are in the D-space will get data (from `Block1Data$` and `Block1Code$`) and will be affected by the masking to be performed after the QR symbol has been filled with data (more on this soon).

Binary data "1" will be displayed as a black element while "0" data will be displayed as a white element. Let's begin by adding the three target patterns to the arrays. `FinderPattern(8,8)` has been filled by data statements and represents the target patterns copied into the upper left, right, and lower left corners of the arrays. The `QR()` array receives the actual data from `FinderPattern(8,8)` while `QRMask()` array has only "1s" written into the same element locations. A "1" in

any element of the `QRMask()` array signifies an I-space. A "1" is written into both arrays to indicate that the odd element should be black and is in the I-space (always in the same position relative to the lower left target pattern.) Next the timing patterns are written to the arrays (between the top target patterns and the left target patterns.) Then the alignment patterns are written into the arrays (quantity and placement based on the version number).

The QR symbol's format data is made up of the ECC level chosen earlier ("M" = "00") and the final mask used, so it may change once I get to that last function. For now, I'll use the (bit string) format data for Mask 0. For ECC level "M" using Mask 0, that would be "101010000010010". On Versions 7–40, the pre-calculated (bit string) version information is copied from the `VersionBlock$()` array into the matrix arrays. In my case, Version 2 does not have any version information held within the QR symbol, so I'm done setting up the I-space for the arrays `QR()` and `QRMask()`.

While you might be tempted to assume the data is just sprayed into the D-space like the words on a page, forget it. The D-space is filled beginning with the lower right corner element and proceeding in a column of two upward. The first columns in my case are $x=25$ and $x=24$. These columns include rows $y=25$ through $y=1$. Note: I said D-space. Any I-space you come to should be skipped, continuing on with the next element. When I've finished row 1, I move 2 columns left ($x=23$ and $x=22$) and proceed downward (to $y=25$). Note: column 7 ($y=7$) will be skipped in its entirety, there is no D-space in column 7. This example has a single Block consisting of `Block1Data$` and `Block1Code$`. When my data is contained in multiple blocks, I just alternate between Block 1 and Block 2 data until all blocks have been placed. Should there be an uneven number of Blocks, I just skip over the unused Blocks. If any D-space is left, I just leave it empty ("0s").

MASKING

The last thing I must do is to apply a data mask to the D-space. If the data in the D-space is to filled with data that looks like alignment patterns, for example, the decoding application will find it difficult to determine the "real" alignment patterns. To eliminate this, a special mask is applied to all the data in the D-space. There are eight masks from which to choose. A test can determine which mask will provide the highest level of success by a decoding application.

Each mask uses a different algorithm of row and column position to determine whether or not each element in D-space should be inverted. I've already stored the format information for Mask 0, so I'll continue and use the Mask 0 algorithm ($(\text{row} \times \text{column}) \bmod 2 = 0$) on each element in D-space. For each row and column position that is in D-space, if the algorithm is true for that position, I invert the data.

After applying the mask, the matrix can be displayed. I haven't done any evaluation on the matrix at this point, but after all this I need to "see" something. For small QR symbols, I want to magnify (if you will) the matrix so it isn't too small. I do this by letting the user choose an element's size (i.e., how many pixels wide and high). With this information, I can go through the `QR()` array and display white or black squares

Penalty Rule	Penalty Points
"00000" or "11111"	+ 3 (+1 for each additional consecutive element)
"00" "11" "00" or "11"	3
"00001011101" or "10111010000"	40
Ratio of "0s" to "1s"	+ ABS (INT (50 - (qty '1s' *100 /qty '0s'))) * 2

Table 4—For the first three rules, you can search the matrix (vertically and horizontally) for matches to the particular pattern and total the penalty points. The last rule is based on the "1" to "0" ratio. The more lopsided the ratio, the higher the penalty.

depending on the data in the array. I also enable the user to switch between displaying the QR symbol versus the I-Space (i.e., mask of non D-space), just so you have an idea of how much space is devoted to information versus "real" data.

Note: This application version does not do an analysis of the QR symbol using each of the eight masks. It simply displays the result using your mask choice. This enables you to visually see the difference between using various masks, which I think is more interesting than just letting the application determine the best Mask. This way you can actually use "Google Goggles" to try and decode each masked version to see how it reacts. If you would like to add automatic mask testing to the application, read on to see how the evaluation is performed.

EVALUATION

To give decoding applications the best shot at interpreting what they see, the QR symbol should have certain characteristics that can be easily discerned without ambiguity. Having an extra bull's eye or massive single-colored areas just creates potential confusion. To reduce this to a minimum, the D-space must be masked with selected patterns. While I can take a look at a finished symbol and "see" if there are areas of confusion, getting an application to do this requires a way of rating each masking result. Table 4 lists rules and the penalties incurred for each infraction.

You might think you only need to check for penalties on the D-space because the I-space remains constant for each mask. However, the format code (part of I-space) does change. Also, some rules are based on all elements and not just those in D-space. Liberty BASIC has an INSTR(a\$,b\$,n) command that can be used to easily look for matches. This is a good reason for using binary strings as opposed to character strings.

Finally, you may wish to save the QR symbol. It can be saved as a bit-mapped image (.BMP) or printed out. There is nothing magical about using black and white or square element shapes. Decoding applications are looking for enough contrast between a "0" (light) and a "1" (dark) to determine the element's state. You can find some QR symbols using multiple colors. In addition, you can also find logos and other graphics superimposed atop a symbol. And, because of the high error-correction level used, the decoding application can still figure out what's hidden. There's a fine balance between artistic license and readability.

I have one last comment. As you know, a QR symbol can hold just about any kind of information. This includes executable code. Decoding applications should inform you of this and let you decide if you want this code to (attempt) to execute. Unfortunately, similar to what we see in e-mail attachments, there may be malicious intent and it is best to cancel

any process unless you feel confident in the source. With that said, I hope you have fun creating your own QR symbols. 📱

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

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2013/271.

RESOURCES

J. Brown, *Matcha Design*, "QR Code Demystified: Part 1," 2011, www.matchadesign.com/blog/qr-code-demystified-part-1.

C. Eby, *Thonky*, "QR Code Tutorial," 2012, www.thonky.com/qr-code-tutorial.

RedTitan Technology, Ltd., "QR CODE Layout," 2011, www.pclviewer.com/rs2/qrtopology.htm.

SOURCE

Liberty BASIC programming software for Windows
Shoptalk Systems | www.libertybasic.com

NEED-TO-KNOW INFO

Knowledge is power. In the computer applications industry, informed engineers and programmers don't just survive, they *thrive* and *excel*. For more need-to-know information about some of the topics covered in this article, the *Circuit Cellar* editorial staff recommends the following content:

CE Marking

A Process to Ensure Product Conformity

by Robert Lacoste

Circuit Cellar 257, 2011

What is CE marking and why is it important? CE stands for *Conformité Européenne*, which is French for "European Conformity." CE marking states that the manufacturer of the product ensures that the product is compliant with the essential requirements of the relevant European directives. Learning about CE marking is useful for engineers who are both using and designing products. Topics: Coding, Product Marking

Error Checking

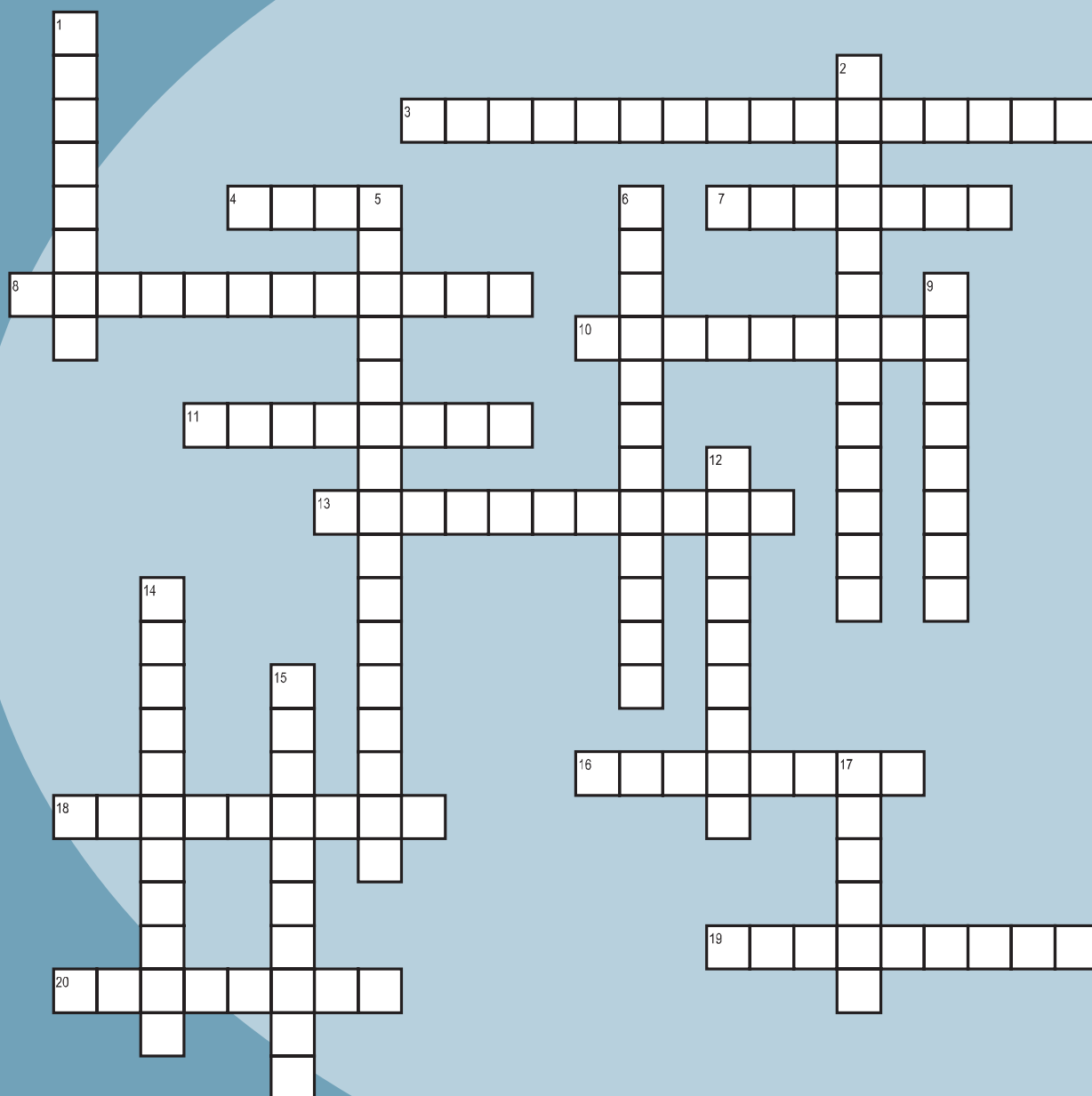
by Jeff Bachiochi

Circuit Cellar 250, 2011

Data security isn't a perk. It's a requirement. This article covers the topics of error checking, checksums, and the cyclic redundancy check. Error checking matters. Topics: Data Security, Error Checking

Go to *Circuit Cellar's* webshop to find these articles and more: www.cc-webshop.com

CROSSWORD



Across

3. Used on PCBs intended for extreme environments [two words]
4. An often repetitious code sequence
7. Measures program volume [two words]
8. An electric current identifier
10. "n!"
11. Evaluates radio frequency circuits [two words]
13. Non-binary code [two words]
16. One of a group of three co-inventors who, in 1956, were awarded the Nobel Prize in Physics for creating the transistor
18. An insulating board's surface
19. Concept proposed by Belgian engineer Charles Bourseul in 1856
20. Electric motors and loudspeakers, for example

Down

1. A type of inductor or transformer whose windings form a closed circular tube
2. May be used to control volume on audio equipment
5. Often used to produce and detect high voltages, sound, and electronic frequency generation
6. An electronic circuit board manufacturing method [two words]
9. An energy-storing device
12. A single-channel power amp with high current power
14. Quality over time
15. Used to measure small objects' thickness
17. Austrian engineer (1907–1992) credited with inventing the printed circuit

The answers are posted at circuitcellar.com/crossword and will be available in the next issue.

IDEA BOX

THE DIRECTORY OF PRODUCTS AND SERVICES

AD FORMAT: Advertisers must furnish digital files that meet our specifications (circuitcellar.com/mediakit).
ALL TEXT AND OTHER ELEMENTS MUST FIT WITHIN A 2" x 3" FORMAT. E-mail adcopy@circuitcellar.com with your file.
For current rates, deadlines, and more information contact Peter Wostrel at 978.281.7708 or peter@smmarketing.us.

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

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

CAN REPEATER \$164

3-Port CAN Repeater helps distribute CAN bus to multiple trunk lines

(623)-399-4688
www.apoxcontrols.com
sales@apoxcontrols.com

USB-CAN \$125
USB-ISOCAN \$157

Free Windows Example source code written in C++ available.

All CAN Features are programmable. 11&29 bit identifiers, Filters, Masks, Baud rate up to 1Mbps.

Optically isolated version available for harsh conditions.

CAN-Repeater has DIN rail mount for your industrial machine applications. (Requires +9 to +24V supply) Great device to distribute power and CAN to many Nodes on your Bus! This is a great device to help clean up the noise on CAN buses which have too many nodes. Each branch has selectable termination to optimize bus reflections.

Ultrasonic Distance Sensing Made EZ

HRUSB-MaxSonar®-EZ™

- Multi-Sensor operation
- USB interface
- Easy integration
- 1 mm resolution
- Calibrated beam pattern
- Starting at \$49.95

HRXL-MaxSonar®-WR™

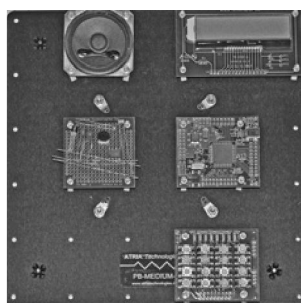
- Incredible noise tolerance
- IP67 rated
- 1 mm resolution
- Multi-Sensor operation
- Calibrated beam pattern
- Starting at \$109.95

I2CXL-MaxSonar®-EZ™

- Great for UAV's and robotics
- Incredible noise immunity
- I2C interface
- 1cm resolution
- Automatic calibration
- Starting at \$39.95

www.maxbotix.com

MICROCONTROLLER KITS



LEARN
CREATE
EXPLORE

BASIC ON BOARD

ATRIA Technologies Inc.

www.AtriaTechnologies.com

I²C™/SMBus

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

MCC
Micro Computer Control

www.mcc-us.com

BPS BusBoard Prototype Systems

Prototyping PC Boards
Many Patterns

See our site: www.BusBoard.us

Available From

JAMECO ELECTRONICS
amazon.com

MOUSER ELECTRONICS
a tti company
amazon.co.uk

NEW CCS www.ccsinfo.com
sales@ccsinfo.com
262-522-6500

VERSION 5 IS THE CODE CONQUEROR

- Flow control and Interrupt buffered to serial routines libraries-specify size of transmit buffer, size of receive buffer, interrupt usage or no interrupt usage, pin for CTS and pin for RTS
- C Profiler-continuous logging and analyzing run-time events to give a profile of the program
- IDE Enhancements-faster debugging, upgraded wizards and Windows 8 compatible!
- Input Capture and PWM Libraries-make better use of capture/compare PWM, Input capture and output capture peripherals on the PIC MCU

WWW.CCSINFO.COM/CCFVER5

Low Cost CAD Software for Windows XP, NT and Vista

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

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

Add a Touch Screen to Your Embedded Product


• No special OS or Library Required.
• Programming GUI is Simple.
• Development Kit = Up and Running in Days.

Learn more at www.reachtech.com, or contact us at 510-770-1417 or sales@reachtech.com.

REACH TECHNOLOGY INC.
Development Kits include serial LCD controller board, display, touch screen, cables, sample images/code, power supply, technical support, 100% Satisfaction Guarantee.

microEngineering Labs, Inc.
www.melabs.com 888-316-1753

Programmers for Microchip PIC® Microcontrollers



PC-Tethered USB Model (shown):

- Standalone software
- Command-line operation
- Hide GUI for automated use
- Override configuration with drop-downs

Stand-Alone Field Programmer:

- Power from target device or adapter
- Program file stored on SD-CARD
- Programming options stored in file
- Single-button operation

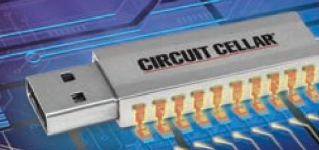
Starting at \$79.95

Program in-circuit or use adapters for unmounted chips.
Zero-Insertion-Force Adapters available for DIP, SOIC, SSOP, TQFP, and more.

PIC is a registered trademark of Microchip Technology Inc. in the USA and other countries.

ONE POWERFUL TOOL FOR YOU

The Entire *Circuit Cellar* Magazine Archive on a Limited-Edition 25th Anniversary USB drive!



Order today at cc-webshop.com

CIRCUIT CELLAR
25
YEARS OF EMBEDDED INSIGHT

GHz BGA/QFN Sockets 0.3mm to 1.27mm

Industry's Smallest Footprint

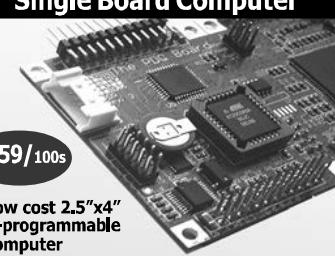
- Up to 500,000 Insertions
- Bandwidth up to 40 GHz
- 2.5mm per side larger than IC
- Ball Count over 3500
- Body Size 2 - 100mm
- Five different contactor options
- Optional heatsinking to 100W
- Six different Lid Options
- <25 mΩ Contact Resistance throughout life



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

PDQ Board™ - A Fast I/O-Rich Single Board Computer



\$159/100s

- Low cost 2.5"x4" C-programmable computer
- 16-bit HCS12 processor clocked at 40 MHz
- 8 PWM, 8 counter/timer, and 8 digital I/O
- 16 10-bit A/D inputs
- Dual RS232/485 ports, SPI and I²C ports
- 512K on-chip Flash, 512K RAM with Flash backup
- Plug-in I/O expansion, including Ethernet, Wi-Fi, GPS, 24-bit data acquisition, UART, USB, Compact Flash card, relays, and more ...

Mosaic Industries Inc.
tel: 510-790-1255 fax: 510-790-0925
www.mosaic-industries.com

LISTEN TO YOUR MACHINES

Ethernet PLCs for OEMs



FMD88-10 and FMD1616-10

Integrated Features :

- ETHERNET / Modbus TCP/IP
- 16 or 32 digital I/Os
- 10 analog I/Os
- RS232 and RS485
- LCD Display Port
- I/O Expansion Port
- Ladder + BASIC Programming

\$229 and \$295
before OEM Qty Discount

tel : 1 877 TRI-PLCS
web : www.tri-plc.com/cci.htm

TRI TRIANGLE
RESEARCH
INTERNATIONAL

HUSB™

OEM\$79

- Add a high speed USB port to a TERN controller.
- High performance USB stack chip (FT232H, FTDI)
- Ready to use, royalty free USB drivers
- USB 1.1 and USB2.0 compatible
- Data transfer rate to 8 MB/sec with D2xx driver
- 2.1"x1.3"



100+ Low Cost Controllers with ADC, DAC, UARTs, 300 I/Os, solenoid, relays, CompactFlash, LCD, Ethernet, USB, motion control. Custom board design. Save time and money.



1950 5th Street, Davis, CA 95616 USA

Tel: 530-758-0180 • Fax: 530-758-0181

www.tern.com • sales@tern.com



LOGICAL DEVICES, INC.



Universal USB Programmer

Starting at **\$195.00!!**

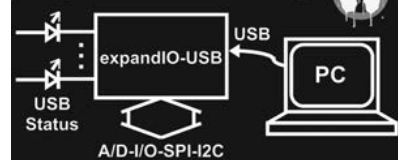
- UV Eraser \$69.00
- PLD design Software \$49.00
- Stand Alone Copier \$175.00

Tel: 303-923-8080

support@logicaldevices.com

www.logicaldevices.com

Revolutionary new expandIO-USB chip



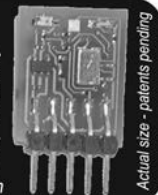
- Ideal for adding USB to sensors & peripherals
- No drivers needed for Windows, Mac, Linux
- No microcontroller programming required
- Also check out our USB-232 USB to UART

www.hexwax.com - Buy from Mouser & Farnell

Amazing PIC programmer

Most devices supported
ICSP, SQTP, & copy limits
at Digikey
& Mouser

\$32
www.flexipanel.com



Actual size - patents pending

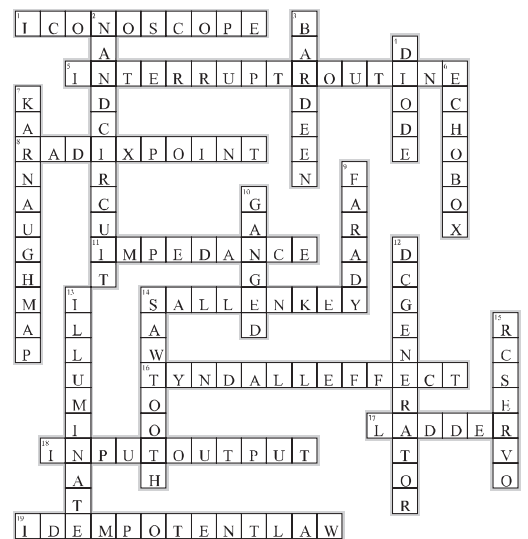
CROSSWORD ANSWERS from Issue 270

Across

- The first widely used television camera tube
- Responds to disturbances [two words]
- On the right of an integer, on the left of a fraction [two words]
- Bridge circuit used to measure resistance
- A versatile, easy-to-design filter [two words]
- Light scattering [two words]
- A kind of passive filter
- MCU pin [two words]
- The result never changes [two words]

Down

- Combines two types of functions in a binary circuit with two or more inputs and one output [two words]
- Won the Nobel Prize in Physics twice
- Developed in 1904 by English engineer John Ambrose Fleming
- A device that receives part of a transmitted pulse and transmits it back to the receiver [two words]
- Used to simplify algebra expressions [two words]
- English scientist (1791-1867) who published the law of induction
- Tuning that uses a single control to tune two or more circuits
- French instrument maker Hippolyte Pixii developed a prototype for this in 1832 [two words]
- What an LED does
- A waveform with a slow linear rise time and a fast fall time
- An absolute-positioning actuator that is typically limited to a 180° rotation [two words]



Can anyone deny that we're on the verge of some major breakthroughs in the fields of microcomputing, wireless communication, and robot design? Tech the Future is a recurring section devoted to the ideas and stories of innovators who are developing the groundbreaking technologies of tomorrow.

Open-Source Hardware for the Efficient Economy

By Catarina Mota and Marcin Jakubowski

In the open-source hardware development and distribution model, designs are created collaboratively and published openly. This enables anyone to study, modify, improve, and produce the design—for one's own use or for sale. Open-source hardware gives users full control over the products they use while unleashing innovation—compared to the limits of proprietary research and development.

This practice is transforming passive consumers of “black box” technologies into a new breed of user-producers. For consumers, open-source hardware translates into better products at a lower cost, while providing more relevant, directly applicable solutions compared to a one-size-fits-all approach. For producers, it means lower barriers to entry and a consequent democratization of production. The bottom line is a more efficient economy—one that bypasses the artificial scarcity created by exclusive rights—and instead focuses on better and faster development of appropriate technologies.

Open-source hardware is less than a decade old. It started as an informal practice in the early 2000s with fragmented cells of developers sharing instructions for producing physical objects in the spirit of open-source software. It has now become a movement with a recognized definition, specific licenses, an annual conference, and several organizations to support open practices. The expansion of open-source hardware is also visible in a proliferation of open-source plans for making just about anything, from 3-D printers, microcontrollers, and scientific equipment, to industrial machines, cars, tractors, and solar-power generators.

As the movement takes shape, the next major milestone is the development of standards for efficient development and quality documentation. The aim here is to deliver on the potential of open-source products to meet or exceed industry standards—at a much lower cost—while scaling the impact of collaborative development practices.

The Internet brought about the information revolution, but an accompanying revolution in open-source product development has yet to happen. The major blocks are the absence of uniform standards for design, documentation, and development process; accessible collaborative design platforms (CAD); and a unifying set of interface standards for module-based design—such that electronics, mechanical devices, controllers, power units, and many other types of modules could easily interface with one another.

Can unleashed collaboration catapult open-source hardware from its current multimillion dollar scale to the next trillion dollar economy?

One of the most promising scenarios for the future of open source hardware is a *global* supply chain made up of thousands of interlinked organizations in which collaboration and complementarity are the norm. In this scenario, producers at all levels—from hobbyists to commercial manufacturers—have access to transparent fabrication tools, and digital plans circulate freely, enabling them to build on each other quickly and efficiently.

The true game changers are the fabrication machines that transform designs into objects. While equipment such as laser cutters, CNC machine tools, and 3-D printers has been around for decades, the breakthrough comes from the drastically reduced cost and increased access to these tools. For example, online factories enable anyone to upload a design and receive the material object in the mail a few days later. A proliferation of open-source digital fabrication tools, hackerspaces, membership-based shops, fab labs, micro factories, and other collaborative production facilities are drastically increasing access and reducing the cost of production. It has become commonplace for a novice to gain ready access to state-of-art productive power.

On the design side, it's now possible for 70 engineers to work in parallel with a collaborative CAD package to design the airplane wing for a Boeing 767 in 1 hour. This is a real-world proof of concept of taking development to warp speed—though achieved with proprietary tools and highly paid engineers. With a widely available, open-source collaborative CAD package and digital libraries of design for customization, it would be possible for even a novice to create advanced machines—and for a large group of novices to create advanced machines at warp speed. Complex devices, such as cars, can be modeled with an inviting set of Lego-like building blocks in a module-based CAD package. Thereafter, CNC equipment can be used to produce these designs from off-the-shelf parts and locally available materials. Efficient industrial production could soon be at anyone's fingertips.

Sharing instructions for making things is not a novel idea. However, the formal establishment of an open-source approach to the development and production of critical technologies is a disruptive force. The potential lies in the emergence of many significant and scalable enterprises built on top of this model. If such entities collaborate openly, it becomes possible to unleash the efficiency of global development based on free information flows. This implies a shift from “business as usual” to an efficient economy in which environmental and social justice are part of the equation.



Catarina Mota is a New York City-based Portuguese maker and open-source advocate who cofounded the openMaterials (openMaterials.org) research project, which is focused on open-source and DIY experimentation with smart materials. She is both a PhD candidate at FCSH-UNL and a visiting scholar at NYU, and she has taught workshops on topics ranging from hi-tech materials and simple circuitry. Catarina is a fellow of the National Science and Technology Foundation of Portugal, co-chair of the Open Hardware Summit, a TEDGlobal 2012 fellow, and member of NYC Resistor.



Marcin Jakubowski graduated from Princeton University and earned a PhD Fusion Physics from the University of Wisconsin. In 2003, Marcin founded Open Source Ecology (OpenSourceEcology.org), which is a network of engineers, farmers, and supporters. The group is working on the Global Village Construction Set (GVCS), which is an open-source, DIY toolset of 50 different industrial machines intended for the construction of a modern civilization (<http://vimeo.com/16106427>).

www.ftdichip.com



ENHANCED USB PERFORMANCE

Streamlined USB
Bridge Solutions

X-CHIP

EXtensive Interfaces
UART, FIFO, SPI, I²C, FT1248

EXtended Features
Battery charger detection
Low active power (8 mA, typical)
Internal MTP memory
Expandable clocking; clock generation
and system clock out

EXceptional Drivers
Windows, MacOS, Android, and Linux



Power of the **Quoting Revolution** in the palm of your hand!

At a click of a button we will find the best
price in the market compiled in one easy to
read spreadsheet
...just like your BOM.



Join the Revolution
Happening Now @
www.PCBnet.com

847-806-0003 sales@PCBnet.com
ITAR, I SO 9001:2008, UL Approved