#177 April 2005

www.circuitcellar.com

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

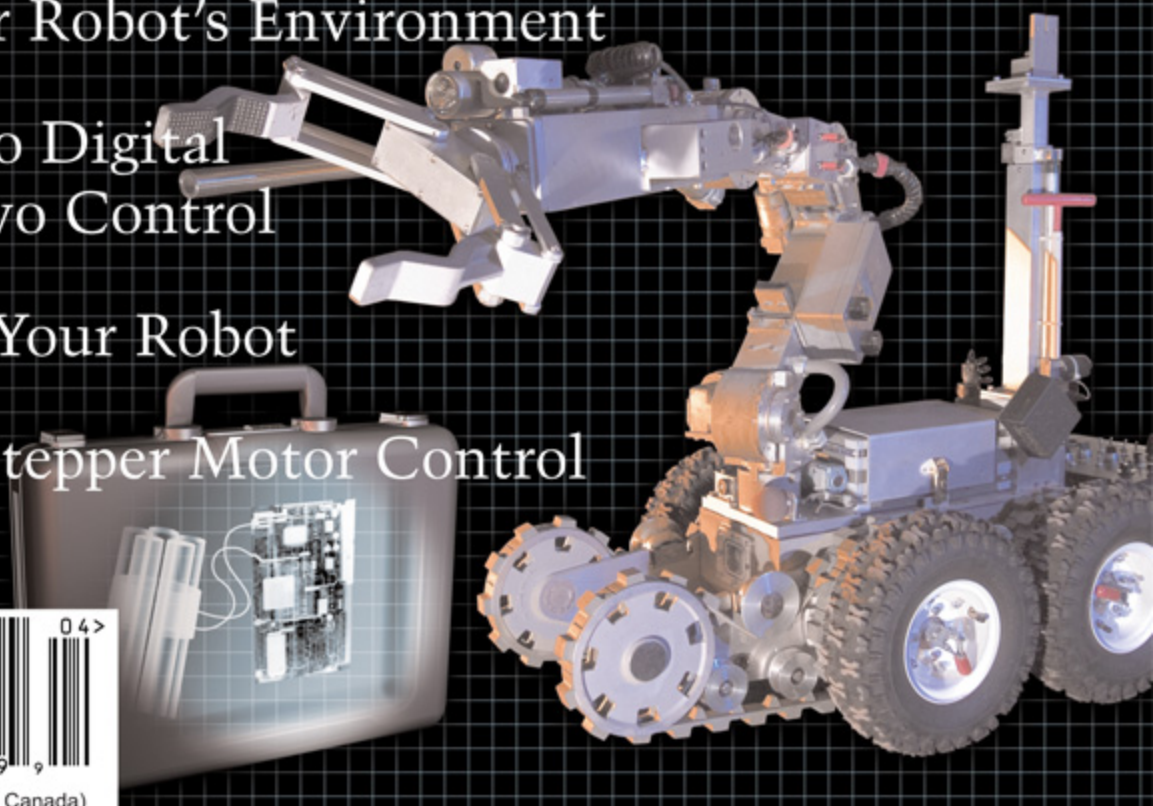
ROBOTICS

View Your Robot's Environment

Upgrade to Digital Serial Servo Control

Sensitize Your Robot

Improve Stepper Motor Control



04>
7 25274 75349 9
$4.95 U.S. ($5.95 Canada)

NetBurner Mod 5270

Do the Math!

95 MHz

+95 MIPs

+32 bits

$59 *Priced per 1000*

NetBurner

Networking in 1 day!

1-800-695-6828 • www.netburner.com

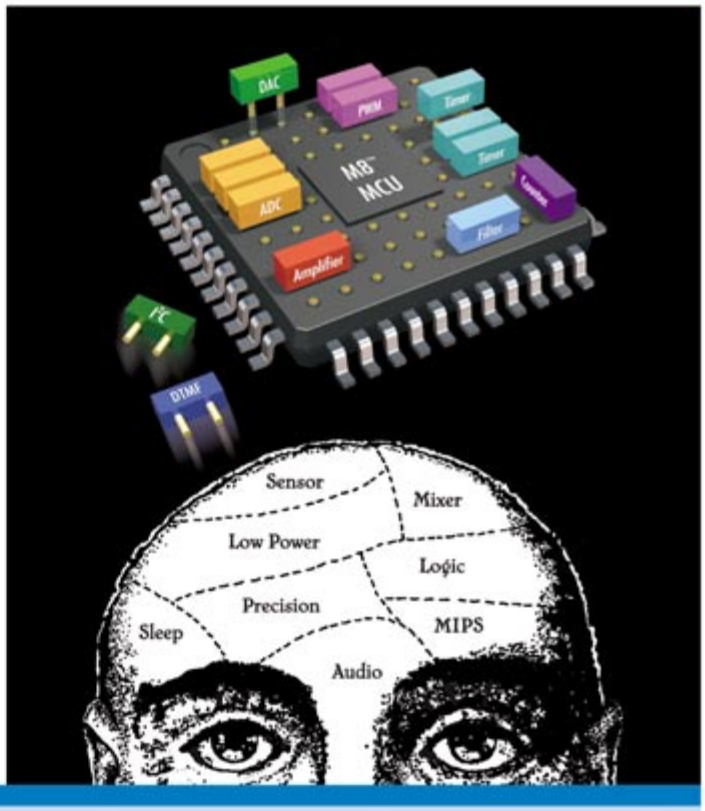
30% OFF
All PSoC Items in Cypress Online Store
www.cypress.com/ad/psoc-cea1

PSoC™ Mixed-Signal Array.
It ll change the way you think about embedded design.

Now with instrumentation-quality programmable analog.

Powerful new programmable analog and digital blocks with memory and MCU for less than $2. Winner of the EDN Innovation of the Year award, our PSoC™ Programmable System-on-Chip™ device is changing the face of embedded design.

- Replace 1,000s of fixed-function devices with a couple of keystrokes
- Dynamically reconfigure a PSoC device, changing functionality on the fly in any application
- Select from hundreds of predefined blocks in our mixed-signal library
- Already designed into 1,000s of applications; check out our online app note library



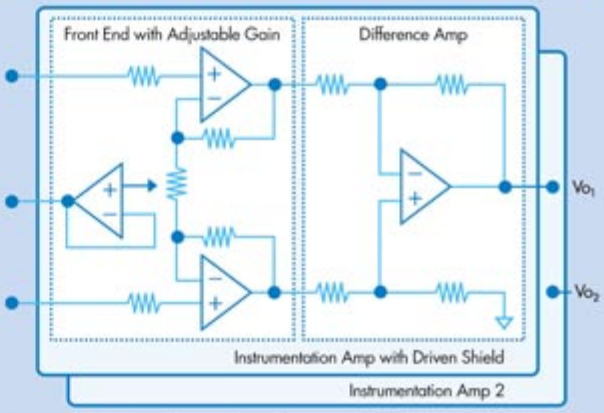
Reduce board size and BOM up to 80%

PSoC Mixed-Signal Arrays with M8 Microcontroller

	CY8C27X	CY8C24X	CY8C22X
CEA ANALOG BLOCKS	12	6	3
DIGITAL BLOCKS	8	4	4
HI REL SONOS FLASH	16K	4K	2K
SRAM	256	256	256
COST	low as $1.99	low as 99¢	low as 69¢

CYPRESS ENHANCED ANALOG™ (CEA™)

• Rail-to-rail analog • Instrumentation amps • Lower voltage offset
• Lower input leakage currents • Programmable gains • Better stability

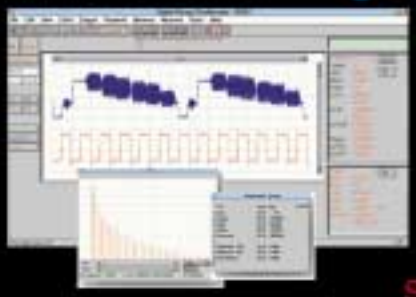


One of 1,000s of examples of programmable analog blocks.

Check out our free online training and our 4-hour applications support:
www.cypress.com/ad/psoc-cea1

CYPRESS MICROSYSTEMS
A Subsidiary of Cypress Semiconductor

Logic Analyzers



NEW!

New LA-5000 Series

More memory and USB 2.0 support

- 40 to 160 channels
- up to 500 MSa/s
- Variable Threshold
- 8 External Clocks
- 16 Level Triggering
- up to 512K samples/ch
- USB 2.0 and Parallel Interface
- Pattern Generator option

LA5240 (200MHz, 40CH)	**$1700** USB 2.0/Parallel
LA5280 (200MHz, 80CH)	**$2350** USB 2.0/Parallel
LA5540 (500MHz, 40CH)	**$2500** USB 2.0/Parallel
LA5580 (500MHz, 80CH)	**$3500** USB 2.0/Parallel
LA55160 (500MHz, 160CH)	**$7500** USB 2.0/Parallel

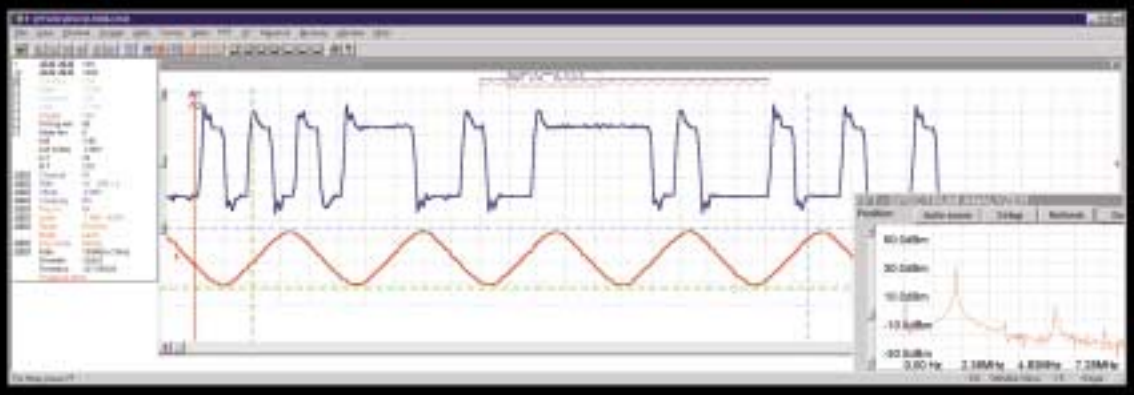
All prices include Pods and Software

Small and portable LA-2124

- 24 Channel Logic Analyzer
- 100MSa/S max sample rate
- Variable Threshold Voltage
- Large 128k Buffer
- Small, Lightweight and Portable
- Only 4 oz and 4.75" x 2.75" x 1"
- Parallel Port Interface to PC
- Trigger Out
- Windows Software

LA2124-128K (100MSa/s, 24CH) Clips, Wires, Interface Cable, AC Adapter and Software **$800**

Digital Oscilloscopes



- 2 Channel Digital Oscilloscope
- **100 MSa/s** max single shot rate
- 32K samples per channel
- Advanced Triggering
- Only 9 oz and 6.3" x 3.75" x 1.25"
- Small, Lightweight, and Portable
- **USB or Parallel Port** interface
- Advanced Math
- FFT Spectrum Analyzer optional

DSO-2102S	$525
DSO-2102M	$650
DSO-2102S(USB)	$600
DSO-2102M(USB)	$725

All prices include **Oscilloscope**, Probes, Interface Cable, Power Adapter and Software

Link Instruments (973) 808-8990
17A Daniel Road East • Fairfield, NJ 07004 • Fax (973) 808-8786
www.Linkins4.com

Seetron Serial LCDs

...almost too easy!

LCDs have a well-deserved reputation for being a pain in the neck to interface. Their non-standard parallel busses and weird timing are enough to turn your hair gray—if you don't pull it out first. Our LCDs have an easy-to-program serial interface (up to 9600 bps) that makes them a snap to use, particularly with friendly micros like BASIC Stamps®.

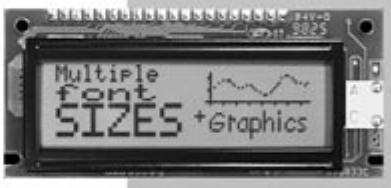
• *Easy hookup and mounting.* We don't stop at simplifying display interfacing and programming. For easy, quick connections, we offer a variety of ready-made wiring harnesses. Need mounting hardware? We've got that, too.

• *In stock right now.* Hate to wait? Our products are in stock and ready to ship the day you place your order.

• *Online satisfaction.* Our customers love our web site—it's fast, uncluttered, and jammed with tech data on our products. The online store runs weekly sales with savings up to 40%. Checkout is a breeze with a simple one-page form.

BPI-216N
2x16 serial LCD
ultra-low power
45.00

ILM-216L
2x16 serial LCD
w/ switch inputs,
config EEPROM
49.00



SGX-120L
120x32-pixel
serial graphics LCD
99.00

TRM-425L
4x20 serial LCD
w/ keypad input,
95-screen memory
119.00

www.seetron.com

Many more models available at seetron.com

Scott Edwards Electronics, Inc.
1939 S. Frontage Rd. #F, Sierra Vista, AZ 85635
phone 520-459-4802 • fax 520-459-0623
web www.seetron.com • email sales@seetron.com

TASK MANAGER

My Favorite Subject

I'd have to say Robotics is my favorite issue theme. We get to feature some of the most fascinating projects. That's not to say the 11 other themes throughout the rest of the year don't attract great applications. It's just that this particular topic brings out an incredibly high level of creativity among designers. Their excitement for these inventive and fun projects is contagious. Each year I look forward to reading about the newest advances in robotics and the interesting ways they're being applied. We selected a few for this issue that are sure to give you some great ideas for your own projects.

A couple Atmel AVR 2004 Design Contest winners used the contest as an opportunity to experiment with AVR microcontrollers in robotics applications, and the results were impressive. Turn to page 12 to learn about the AVRcam. Designed around ATmega8 and ATtiny12 microcontrollers, this camera provides real-time tracking of multiple different-colored objects in addition to still photography capability. John Orlando discusses his range of goals—which included designing an inexpensive system that would be easily expandable—and how he and codesigner Brent Taylor accomplished them. Their successful design won them Second Prize in the contest. The well-designed AVRcam is also appropriate for motion detection and object recognition applications.

For another terrific AVR-based robotics application, turn to page 44. Eric Gagnon won Honorable Mention for his 32-channel RC digital servo controller. This ATmega8515L microcontroller-based project is well suited for projects that require versatile servos, including walking robots and animatronics applications. By upgrading from typical RC servos to digital RC servos, he achieved a hardware-based solution that features 16-bit accuracy and 12-bit resolution. In the first part of this two-part series, Eric covers the architecture. Be sure to come back next month, when he'll discuss the circuits and FPGAs.

Columnist Jeff Bachiochi also delves into robotics as he analyzes Paratech's quantum tunneling composite (QTC) technology (page 48). He wanted to explore ways to improve the sensitivity, or rather a lack thereof, of robots. Sensitivity and input feedback will become increasingly important as robots become more integrated into fields such as manufacturing and military applications. By adding QTC sensors to his Heathkit Hero, Jeff was able to make the robot pick up an egg without crushing it.

Finally, we have an interesting article entitled "Three-Axis Stepper Motor Driver," written by the design team of Viraj Bhanage, Prajakta Deshpande, and Praveen Deshpande (page 68). Their RC system, which was built around Philips P89C51RD2 and Atmel AT89C2051 microcontrollers, improves component control. It was designed to aid laser technicians who have to precisely control optical components.

I hope you enjoy reading these intriguing articles as much as I did. A special thank you goes out to the Connecticut State Police for allowing us to photograph their bomb disposal robot for the this month's cover. One last note to the robotics enthusiasts headed to Hartford this month: Good luck in the Trinity College Fire-Fighting Home Robot Contest!

jennifer.huber@circuitcellar.com

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

EDITORIAL DIRECTOR/FOUNDER
Steve Ciarcia

MANAGING EDITOR
Jennifer Huber

TECHNICAL EDITOR
C.J. Abate

WEST COAST EDITOR
Tom Cantrell

CONTRIBUTING EDITORS
Ingo Cyliax
Fred Eady
George Martin
George Novacek
Jeff Bachiochi

NEW PRODUCTS EDITOR
John Gorsky

PROJECT EDITORS
Steve Bedford
Ken Davidson
David Tweed

CHIEF FINANCIAL OFFICER
Jeannette Ciarcia

CUSTOMER SERVICE
Elaine Johnston

CONTROLLER
Jeff Yanco

ART DIRECTOR
KC Prescott

GRAPHIC DESIGNER
Mary Turek

STAFF ENGINEER
John Gorsky

QUIZ COORDINATOR
David Tweed

ADVERTISING

PUBLISHER
Dan Rodrigues E-mail: dan@circuitcellar.com

ASSOCIATE PUBLISHER/DIRECTOR OF SALES
Sean Donnelly Fax: (860) 871-0411
(860) 872-3064 E-mail: sean@circuitcellar.com
Cell phone: (860) 930-4326

ADVERTISING COORDINATOR
Valerie Luster Fax: (860) 871-0411
(860) 875-2199 E-mail: val.luster@circuitcellar.com

ADVERTISING ASSISTANT
Deborah Lavoie Fax: (860) 871-0411
(860) 875-2199 E-mail: debbie.lavoie@circuitcellar.com

Cover photograph Chris Rakoczy—Rakoczy Photography
www.rakoczyphoto.com

PRINTED IN THE UNITED STATES

CONTACTING CIRCUIT CELLAR

SUBSCRIPTIONS:
INFORMATION: www.circuitcellar.com or subscribe@circuitcellar.com
To Subscribe: (800) 269-6301, www.circuitcellar.com/subscribe.htm, or subscribe@circuitcellar.com
PROBLEMS: subscribe@circuitcellar.com

GENERAL INFORMATION:
TELEPHONE: (860) 875-2199 Fax: (860) 871-0411
INTERNET: info@circuitcellar.com, editor@circuitcellar.com, or www.circuitcellar.com
EDITORIAL OFFICES: Editor, Circuit Cellar, 4 Park St., Vernon, CT 06066
NEW PRODUCTS: New Products, Circuit Cellar, 4 Park St., Vernon, CT 06066
newproducts@circuitcellar.com

AUTHOR CONTACT:
E-MAIL: Author addresses (when available) are included at the end of each article

For information on authorized reprints of articles, contact Jeannette Ciarcia (860) 875-2199 or e-mail jciarcia@circuitcellar.com.

CIRCUIT CELLAR®, THE MAGAZINE FOR COMPUTER APPLICATIONS (ISSN 1528-0608) and Circuit Cellar Online are published monthly by Circuit Cellar Incorporated, 4 Park Street, Suite 20, Vernon, CT 06066 (860) 875-2751. Periodical rates paid at Vernon, CT and additional offices. **One-year (12 issues) subscription rate USA and possessions $21.95, Canada/Mexico $31.95, all other countries $49.95. Two-year (24 issues) subscription rate USA and possessions $39.95, Canada/Mexico $55, all other countries $85.** All subscription orders payable in U.S. funds only via VISA, MasterCard, international postal money order, or check drawn on U.S. bank.

Direct subscription orders and subscription-related questions to Circuit Cellar Subscriptions, P.O. Box 5650, Hanover, NH 03755-5650 or call (800) 269-6301.

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

Ready or Not, Here Comes the Lead-Free Compliance Deadline.

To meet the July 1, 2006 deadline for the Restrictions on Hazardous Substances (RoHS) Directive, many PCB fabricators are scrambling for solutions. They may be making promises, but are your PCB suppliers simply standing still?

Sierra Proto Express
has already crossed the finish line.
We build lead-free PCBs to withstand the demanding, high-temperature lead-free assembly process. We are the first fabricator to address the issue of reliability through the high-temperature lead-free solder process—both in the lab and in the field.

Can your supplier guarantee reliable Lead-Free circuit boards for your next order?

We offer quickturn and volume production of standard technology and high-technology lead-free printed circuit boards proven to withstand up to 2000 cycles of -45°C to 145°C.

For lead-free boards guaranteed to be reliable during the punishing lead-free assembly process, **call 1-800-763-7503 x500, email files@protoexpress.com or visit www.protoexpress.com/leadfree.**
Sierra Proto Express is located at 1108, West Evelyn Ave., Sunnyvale, CA 94086, USA.

QUALITY LEADER
SIERRA
PROTO EXPRESS
SINCE 1986

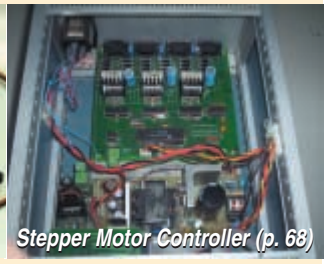
April 2005: Robotics

FEATURES


AVR-Based Camera (p. 12)


LPC2138-Based DAQ Project (p. 20)

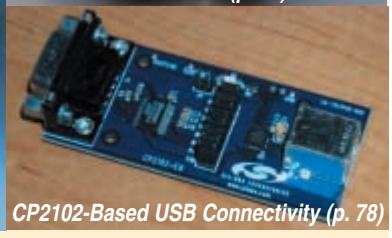

Alternative to Typical RC Servo Controllers (p. 44)


Stepper Motor Controller (p. 68)

COLUMNS

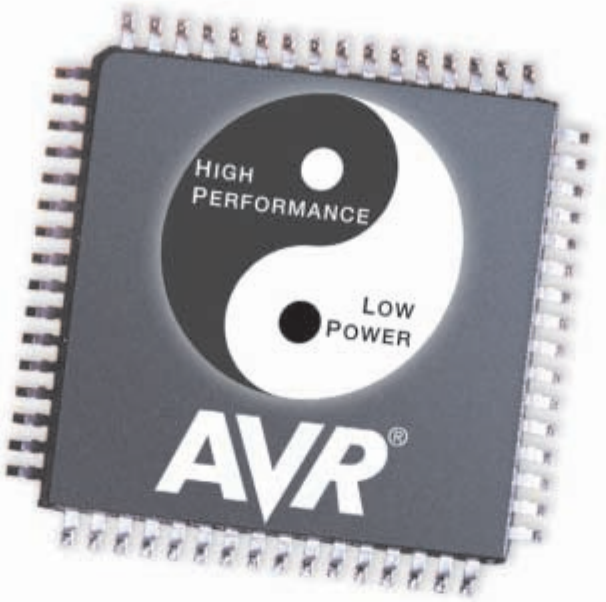
Heathkit Hero Makeover with QTC Sensors (p. 48)


LED Solution for Flashlight Performance (p. 40)


CP2102-Based USB Connectivity (p. 78)

DEPARTMENTS

Come see us at ESC SF
March 8–10, 2005
Booth# 1314



AVR microcontrollers
The perfect combination of performance and power consumption

Specifically designed for the latest battery operated portable applications, Atmel's new Low Power AVR microcontrollers provide six low-power operation modes. These devices work down to 1.8 volts providing maximum battery life. In low power modes such as standby, AVR turns off all internal peripherals and the core. Standby mode leaves only the external crystal running to deliver start-up times as low as 6 clock cycles.

AVR is a RISC CPU that executes instructions in a single clock cycle, with a rich CISC-like instruction set and 32 working registers. AVR microcontrollers have a very high code density and extremely fast execution speeds up to 24 MIPS. The Flash program memory and on-chip EEPROM will help you slash months off your development efforts – save thousands of dollars in project costs, and save YOUR energy as well.

AVR comes in a full selection of package and performance options, and is supported by a full compliment of integrated tools to make your development process fast, simple and successful. Atmel dedicated staff as well as certified technical consultants are available to help and support your projects.

See for yourself: Atmel's Butterfly evaluation kit demonstrates AVR's low power capabilities, and also serves as a development kit for the mega169 microcontroller.

So get your project started off right. Learn more about AVR at www.atmel.com/ad/lowpoweravr, and **register to win a free butterfly evaluation kit.**

See why AVR offers the perfect combination.

Check out AVR today at: www.atmel.com/ad/lowpoweravr

Low Power AVR Typical Power Consumption				
Operating Voltage	Power Down	Real Time Counter	Active 32 kHz	Active 1 MHz
1.8 Volts	Less than 100 nA	4.5 µA	15 µA	240 µA
3 Volts	Less than 250 nA	7 µA	26 µA	420 µA

ATMEL®
Everywhere You Are℠

NEW PRODUCT NEWS

Edited by John Gorsky

OPTICAL SENSOR OPERATES AT DISTANCES UP TO 40 mm

A new reflective optical sensor capable of detecting objects at operating distances of up to 40 mm is now available. The surface-mount TCND5000 reflective sensor, which includes an IR emitter and PIN photodiode in a single package, is well suited for use in object presence sensors, touch sensors, and proximity sensors in a broad range of consumer, industrial, and automotive applications.

With high sensitivity and an operating distance from 2 mm to as great as 40 mm, the highly integrated TCND5000 sensor offers designers a compact and reliable solution for enhancing the performance of electronic systems such as cell phones, in which the device compensates for the proximity of your ear and adjusts the volume of the speakerphone function accordingly.

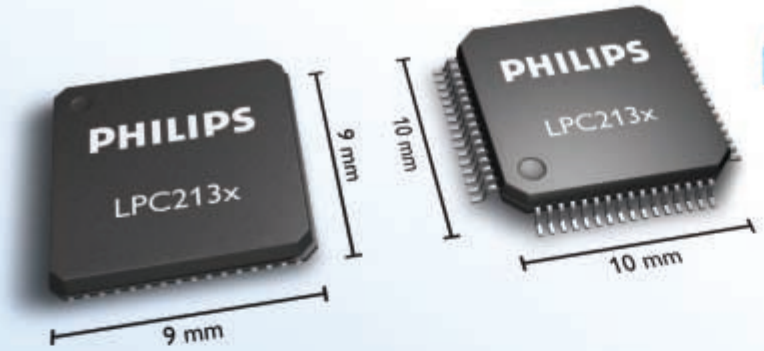
The sensor's high-intensity IR emitter features an operating wavelength of 950 nm, while a 950-nm IR band-pass filter eliminates interference from daylight. An optical barrier between the emitter and detector reduces crosstalk to very low levels.

The TCND5000 sensor is 6 mm × 3.76 mm × 3.9 mm (h). The WEEE-compliant sensor is a restriction of hazardous substances (RoHS) device. It meets JEDEC level 4 standards and are available in a lead-free, surface-mount-style package. The sensor is designed for IR reflow soldering with a peak temperature of 260°C; therefore, it's suitable for lead-free solder processes.

Pricing in 50,000-piece quantities starts at $**75** per 100 pieces.

Vishay Intertechnology, Inc.
www.vishay.com

Get your design skills ready. Coming soon...
ARM Design contest 2005



LPC213x
32-bit ARM7TDMI-S
microcontrollers

We've squeezed more in so you can get more out

- 60 MHz operation (54 MIPS) from both on-chip Flash and SRAM
- Two UARTs, two I²C, one SPI and one SPI / SSP interfaces
- 8-channel 10-bit ADC and 10-bit DAC
- User-code security
- Real-time Debugging & Trace
- ISP, IAP, Parallel Programmer Support
- Tiny packages – HVQFN64 (9 x 9 x 0.85 mm), QFP64 (10 x 10 x 1.4 mm)

Ease your migration to 32-bit performance with Philips LPC213x microcontrollers. The industry's fastest ARM7 family provides up to 512 Kbytes of 60 MHz Flash memory with a host of on-chip peripherals and interfaces. All of this comes with low power consumption, tiny packages and full support from recognized third party tool providers. It's the performance you would expect at a price you can afford.
For more information go to www.philips.com/microcontrollers

Part Number	Memory		Timers			I/O	Serial Interfaces			Analog		Interrupts	F. max	Package
	Flash	RAM	# of Timer	PWM	RTC/ WD	Pins	UART	I²C	SPI	A/D ch/ 10b	DAC ch/ 10b	Ext. Levels	(Mhz)	
LPC2138	512K	32K	4	6	Y	47	2	2	2	2x8	1	22(4)16	60	LQFP64, HVQFN64
LPC2136	256K	32K	4	6	Y	47	2	2	2	2x8	1	22(4)16	60	LQFP64
LPC2134	128K	16K	4	6	Y	47	2	2	2	2x8	1	22(4)16	60	LQFP64
LPC2132	64K	16K	4	6	Y	47	2	2	2	1	1	22(4)16	60	LQFP64, HVQFN64
LPC2131	32K	8K	4	6	Y	47	2	2	2	1	-	22(4)16	60	LQFP64

PHILIPS
sense and simplicity

NEW PRODUCT NEWS

Edited by John Gorsky

40-MHz PICs WITH SELF-REPROGRAMMABLE MEMORY

Four new high pin-count, high-density members of the **PIC18Fxxxx** family are now available. These devices offer a cost-effective 96 and 128 KB of self-reprogrammable, high-endurance flash memory with up to 10 MIPS performance over a wide operating voltage range (2 to 5.5 V). These features, combined with nanoWatt Technology power management and a rich set of analog and digital peripherals, allow the microcontroller series to compete with 16-bit devices in high-end embedded applications while retaining ease of use and helping you to preserve your 8-bit development tool and software investments.

This family meets the needs of engineers with 8-bit code and development tools. There's a growing need for microcontrollers with increased computational power and larger program memory sizes because of the transition of code development methodologies from assembly to C language. The PIC18F8722 8-bit microcontroller series addresses these performance and memory needs by providing linear access (no pages) to a memory space as large as 2 MB, while offering complete code and tool compatibility with smaller Microchip microcontrollers. In addition, the new microcontrollers include two synchronous serial ports (capable of SPI or I²C) and two asynchronous serial ports (LIN-capable USARTs) for expanded connectivity.

Pricing starts at **$6.81** in 10,000-piece quantities.

Microchip Technology Inc.
www.microchip.com

CIRCUIT CELLAR

Test Your EQ

Edited by David Tweed

Problem 1—Is it possible to create a digital all-pass filter that has a group delay that's a fraction of the sample period?

Problem 2—In digital signal-processing applications, there is sometimes an effect known as the Gibbs phenomenon, which is a characteristic ringing associated with sharp edges and transients. Is this a function of sampling, quantization, or filtering in the system? Or is it a combination of all three? Is this a problem?

Problem 3—If you could operate your automobile on Mars, what would the braking distances be like relative to Earth?

Problem 4—What are the five built-in and 10 "transient" commands that come in a standard CP/M 2.2 distribution? What's the difference between the two categories?

Contributed David Tweed

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

AVR® 2004 DESIGN CONTEST

Second Prize

AVRcam

A Low-Cost Embedded Vision System

The AVRcam is a low-cost image-processing engine. The system shows great promise for robotics applications, as well as motion detection and object recognition.

I've always been amazed by how well humans perceive and react to the world around them in real time. I can drive a car at 65 mph on the expressway, following the white lane-separation stripes and the yellow border stripes, without getting into an accident. Throw me a ball, and I'll catch it—or at least get out of the way before it hits me. Trying to figure out how to mimic some of these capabilities in a man-made system has driven my interest in robotics ever since I was 6 years old. This is in part the reason that I've found a lifelong hobby in robotics.

The eyes are arguably the most complicated sensors attached to the human body. Thus, artificial vision systems tend to be extremely complex. These systems typically require a considerable amount of computing power to acquire and process their environments in real time. The end goal of most vision systems is to determine specific information about the environment: How many different objects exist? Are objects moving? What are the colors of the objects? How far away are the objects? The answers to these questions enable the appropriate post-processing of the environment, as would be required by a particular application.

Several existing systems can perform vision processing at different levels of capability. For hobbyists, the CMUcam is a small, low-cost system capable of tracking one colored object at 16.7 frames per second (www-2.cs.cmu.edu/~cmucam/). It uses a Ubicom SX28 microcontroller running at 75 MHz to acquire and process images through software. However, the CMUcam isn't very extensible because almost all of the SX28's resources are required for the aforementioned image-processing task. A step up from the CMUcam is the Cognachrome system (www.newtonlabs.com), which is capable of tracking multiple objects of various colors at 60 frames per second. This system, which costs several thousand dollars, performs much of its image processing in hardware to achieve its impressive processing tasks.

I had considered trying to develop a small, real-time vision engine for several years. This effort mostly involved back-of-the-envelope calculations about the kind of processor/logic needed to process a pixel stream and the amount of RAM needed to make it work. It seemed like an interesting problem. The bottom line was that I wanted to give my robots the ability to see the world around them. This article describes how I did it.

In terms of image processing, I was interested in being able to track multiple objects of different colors in real time, defined as 30 frames per second, which is about as fast as the human eye can perceive a change. This would allow my robots to map their world according to specific color codings. I also wanted to be able to take full-color snapshots with the system in order to evaluate its surroundings.

In addition to the requirements of a generic image-processing engine, I wanted the system to be inexpensive, easy to build, and easily expandable so I could add new features as needed. I also wanted to be able to do the majority of the software development in a high-level programming language such as C. This would allow me to test algorithms on different platforms before moving to the target hardware. Finally, I wanted to be able to program the system in-circuit so that I could easily test new ideas. And thus, the AVRcam was born.

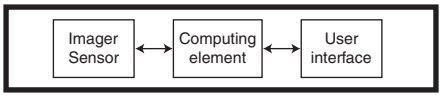


Figure 1—*A generic image-processing system must be capable of sampling an image, processing the image, and providing the processed results to the outside world.*

VISION SYSTEM SYNOPSIS

A vision system is typically a subsystem of a larger entity, thus taking the burden of the computationally intensive image-processing task off the main controller. The purposes of a vision system are to sample and process images, and then provide high-level post-processed image information to the main controller. A generic vision subsystem is shown in Figure 1.

The image-processing chain begins by capturing the images of interest with an image sensor. Typically, a CMOS or CCD camera is used to capture an image and convert it into a series of electrical signals. Color cameras have tiny filters over each sensor element. This allows only a specific wavelength of light to pass through to the sensor. Thus, an array of sensor elements with red, green, and blue filters results in a color image being captured by the camera because all colors can be created by combining these three. These sampled pixel signals are then output from the camera in either digital or analog format.

Next, a computing element is

required to receive the sampled pixel signals and to perform the necessary processing. The hardware required for this task ranges from a small 8-bit microcontroller to a full-blown PC. The exact processing required varies by application, and usually performs tasks such as looking for abrupt changes in sampled pixels or calculating the average pixel value across an entire image.

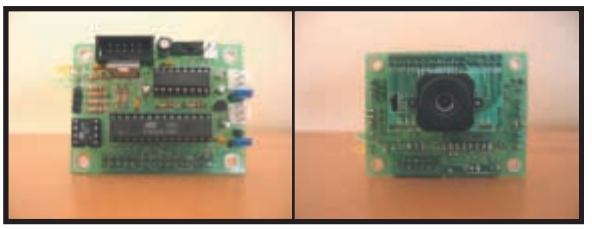


Photo 1—*The AVRcam measures 2.4″ × 1.9″. The OV6620 CMOS color image sensor provides a digital stream of pixel data.*

The computing element also has some throughput requirements because it must be fast enough both to sample the pixel stream and to perform the needed processing. In some cases, it's useful for the computing element to sample an entire frame of pixels from the camera and store it in memory to allow the entire image to be processed at once after it's captured. However, the amount of memory required for such a task can be large, adding additional hardware to the system.

Finally, after an image is processed, the resultant data needs to be output so that some other computing entity can make use of it. The post-processed data typically contains the high-level visual information extracted from the scene. This user interface isn't normally going to be an actual user, but this abstraction is a good way to think about the interface to the vision system.

AVRcam

The AVRcam uses the OmniVision OV6620 CMOS color image sensor, which provides a digital stream of pixel data (see Photo 1). The sensor is available already mounted to a circuit board with supporting components and a lens in the form of the C3088 evaluation board. The pixel data is provided in Bayer format, which means that red, green, and blue pixels are interleaved to generate a colorful image.

The OV6620 has a native resolution of 352 × 288 pixels. It can be configured to a lower resolution of 176 × 144 pixels. This sensor provides the digitized pixel samples of each frame at a rate of 30 frames per second through two 8-bit data buses. The sensor also provides various timing signals needed for synchronization, such as a pixel clock (PCLK), a horizontal reference signal to indicate a new line is about to start (HREF), and a vertical sync signal to indicate a new frame is about to start (VSYNC).

An Atmel ATmega8 microcontroller processes the OV6620's stream of pixels. It has a Harvard architecture and a RISC instruction set, providing many single clock-cycle instructions. The ATmega8 also provides a rich set of on-chip peripherals, such as 8- and 16-bit timers, external interrupts, hardware UART, and plenty of general-purpose I/O for sampling the data buses.

The AVRcam's user interface is provided through the on-chip hardware UART, which offers a simple way for external devices to communicate with the AVRcam. A simple protocol allows you to command and control the system. All post-processed object tracking information is provided through this UART as well.

SIMPLE HARDWARE

The AVRcam hardware's simplicity contributes to the system's low cost and low-power consumption (see Figure 2). The upper nibble of each pixel data bus on the OV6620 is connected directly to the lower nibble of the ATmega8's port B and port C, which are both configured as inputs. Note that the lower nibble of each pixel data bus isn't connected to the ATmega8. Early experiments showed that the system would be capable of differentiating and tracking color blobs effectively with only 4 bits of color data per channel. This reduces the range of each of the three color channels (red, green, and blue) from 256 possible values (2^8) down to 16 possible values (2^4). Thus, the system can represent 1,024 different colors ($2^4 \times 3$).

The vertical sync (VSYNC) signal from the OV6620 is connected to the ATmega8's external interrupt 1, which allows the microcontroller to be interrupted every time a new frame is about to begin. The horizontal reference (HREF) signal, which is connected to external interrupt 2, triggers each time a new line of pixel data is about to begin. Finally, the pixel clock (PCLK) signal from the OV6620 is

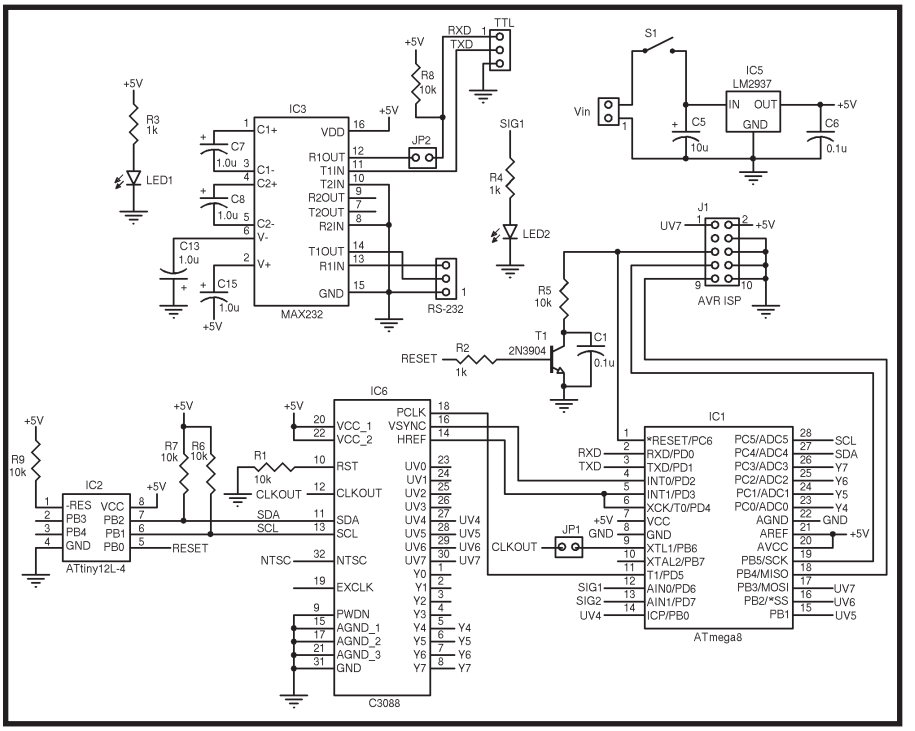


Figure 2—*Building the low-cost AVRcam is fairly simple. An Atmel ATmega8 microcontroller drives the system.*

directly connected to the ATmega8's 16-bit Timer1. The timer is configured to count external rising-edge transitions with an interrupt on overflow. It's preloaded to overflow after an entire line of pixel data has been sampled.

Although the OV6620 provides a multitude of signals, there's still a considerable amount of data flowing from the OV6620 to the ATmega8 that needs to be processed. How much data? Approximately 3 million bits per second!

$$\frac{4\text{ bits}}{\text{pixel}} \times \frac{176\text{ pixels}}{\text{line}} \times \frac{144\text{ lines}}{\text{frame}} \times \frac{30\text{ frames}}{\text{second}}$$

That's a lot of data! It's actually too much data, so a couple of simplifications are made.

For starters, there are twice as many green pixels as there are red and blue, and only one green is needed. So, every other green pixel is ignored. In addition, when the system is tracking colors, every other two-pixel block is skipped, reducing the horizontal resolution to 88 pixels per line. This leaves the actual amount of data to be processed a little over 1 million bits per second. How can this much data be processed? Figures 3 and 4 demonstrate the main reasons.

Another hardware trick is to use the 17.7-MHz crystal source from the OV6620 to drive the clock input of the ATmega8. The microcontroller is only rated to run up to 16 MHz, so this is clearly exceeding its limitations, but only slightly. The benefit here is huge: there's inherent synchronization between the pixel data flowing from the image sensor and the sampling of pixel data on the ATmega8. Thus, after synchronization at the beginning of each image line, it isn't necessary to sample PCLK to determine when valid data exists on the data buses. Ensuring the sampling of the pixel buses at the appropriate time (or, in this case, after the appropriate number of executed instructions) is the only requirement.

TWO'S A CROWD

There is only one problem with the aforementioned scenario. The 17.7-MHz crystal signal isn't output by the OV6620 by default; however, setting one of the registers in the OV6620 through its I²C interface can enable it. But the ATmega8 can't set this register in the camera because it has no clock source to execute instructions. Ah, yes, the old chicken before the egg problem. The best solution I could come up with was to add the smallest possible microcontroller to the board (to take care of setting up the appropriate register on the OV6620 to output its clock source).

Enter the AVR ATtiny12. This small 8-pin microcontroller had everything I needed: an internal 1-MHz oscillator and up to six I/O lines. Two of the I/O lines were configured to bit-bang an I²C interface to configure the appropriate register on the OV6620 to output its clock signal. I used another I/O line to hold the ATmega8 in Reset mode until the clock signal was set up and stable. This solution worked well, and added only a slight delay at start-up.

There isn't much left to the hardware. The system includes a standard MAX232 level-conversion chip, which supports RS-232-level serial communications through the ATmega8's UART, in addition to TTL-level signaling. Finally, the standard AVR STK200/300 10-pin ISP header is integrated in with the system to allow for the complete reprogramming of the ATmega8's firmware.

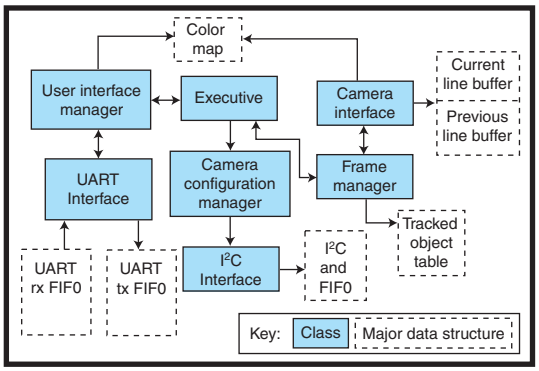


Figure 3—*Familiarize yourself with the AVRcam software architecture. Here are the major classes with their data structures.*

AVRcam SOFTWARE

The software behind the AVRcam ties everything together and makes the system work. Figure 3, a class diagram of the AVRcam firmware, highlights the main modules in the system. As you can see, a simple executive sits at the center of the system. It's used to dispatch events to the classes as needed.

There are actually two types of events in the system: regular events and fast events (denoted in the code by EV_xxx or FEV_xxx). Regular events are low-priority events placed in an event FIFO. Fast events are placed in an event bit mask because the time to insert and remove them from a FIFO is too long. Fast events are used for the time-critical events, like a line of pixels ending. Regular events are used for less time-sensitive events, such as when a serial byte of data is received.

The user-interface manager is responsible for parsing and processing incoming serial commands from the UART interface. It also generates the appropriate responses. This includes publishing events after a command is received. The camera configuration

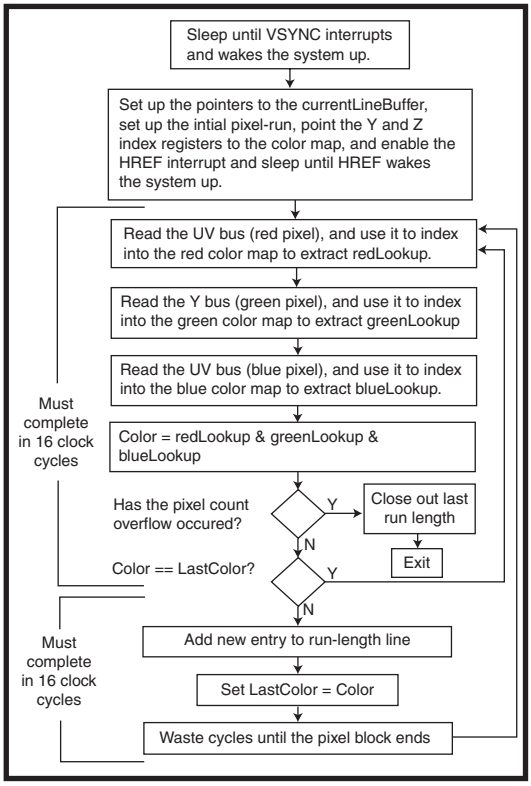


Figure 4—*The heart of the low-level camera interface samples pixels, maps them to actual colors, and run-length ecodes the image line, all in 16 clock cycles.*

manager is responsible for configuring the internal registers in the OV6620.

The frame manager provides most of the line-level processing tasks, such as finding connected regions between two contiguous lines of pixels. It also provides frame-level processing such as parsing through the tracked-object table to remove objects that are too small to be considered tracked objects. Finally, the camera interface provides the low-level interface to the OV6620 through the pixel data buses and the various synchronization signals.

The vast majority of the AVRcam firmware was written in the C programming language, using the open-source AVR-GCC C compiler that's part of the WinAVR distribution. The ANSI-compliant C compiler does an excellent job at generating code. And, best of all, it's free, which is one of the reasons why it's so popular. In keeping with the spirit of open-source code, all of the source code developed for the AVRcam project was released as open source under the GNU General Public License (GPL) in the hope that other developers would improve and enhance the system.

The AVRcam definitely has space left on the ATmega8 for new features. Approximately 4 KB of the 8-KB on-chip flash program memory, 700 bytes of the 1-KB on-chip RAM, and 48 bytes of the 512 bytes of EEPROM are used by the firmware. You may download the code from the *Circuit Cellar* ftp site. Although most of the code was written in C language, the two functions at the heart of the system were coded in assembly language to ensure that they met some extremely strict timing requirements. One of these functions is used for color blob tracking. The other is for image dumping.

The color blob tracking function is required to do a lot in a short amount of time (see Figure 4). Essentially, it needs to sample the red, green, and blue pixels, map that RGB combination into an actual color of interest in the color map (or no color if it doesn't match one), and run-length encode the line to reduce the amount of data required to process the image. All of this must take place in 16 clock cycles! Needless to say, there isn't a cycle to spare in the routine. Getting the routine exactly right took me several weeks of plotting, which included a change in the hardware to remap which pixel data bus was connected to which I/O port on the ATmega8. The complete source code for this function is shown in Listing 1 (page 16).

This routine uses a novel algorithm for mapping individual color channels to a particular color in the color map.[1] Typical implementations use a large look-up table in which the individual channels form an index into a color table to extract the actual color represented. If this solution were used, a look-up table containing 2^{12} entries, or 4 KB, would be required. Considering the ATmega8 has only 1 KB of RAM, this solution isn't feasible. The algorithm I used required only 48 bytes of precious RAM to provide a map that determined if a combination of red, green, and blue map into a trackable color.

SEEING IS BELIEVING

In addition to the AVRcam

firmware, my friend Brent Taylor developed a PC application called AVRcamVIEW to provide a cross-platform calibration and test environment for the AVRcam. The application, which makes the system accessible to anyone with a PC, was written in Java. It can be executed on any platform with the appropriate Java runtime engine. (Windows, Linux, and Mac OSX are currently supported.) Note that it interfaces to the AVRcam through the RS-232 port on the PC.

AVRcamVIEW allows you to take full-resolution color snapshots with the AVRcam. It displays the captured image in both a Bayer format and a bilinear-interpolated version in which it calculates a 12-bit color value for each pixel in the image. Each image dump takes about 4 s to transmit the full image back to the PC at a data rate of 115.2 kbps. This image then can be used to better understand how the AVRcam views the surrounding environment.

A dumped image also can be used to configure the color map on the AVRcam. You can use the color map to set the bounds of the eight colors that the AVRcam can track. In addi-

Listing 1—*The core function is for sampling pixels, mapping to colors, and run-length encoding the line. The routine has strict timing requirements regarding the number of cycles it takes to execute.*

```
*****************************************************************
;Track frame handler
*****************************************************************
_trackFrame:
sleep  ; ...And we wait for HREF to wake us up
;Returning from the interrupt/sleep wakeup will consume 14 clock
;cycles (seven to wake up from idle sleep, three to vector, and four
;to return)
;Disable the HREF interrupt
in      tmp1, _SFR_IO_ADDR(GICR)
andi    tmp1, HREF_INTERRUPT_DISABLE_MASK
out     _SFR_IO_ADDR(GICR), tmp1
;A couple of NOPs are needed here to sync up the pixel data determined
;empirically by trial and error.
nop
nop
*****************************************************************
;Acquire pixel block (R-G-B triplet)
*****************************************************************
_acquirePixelBlock:                          ;Clock Cycle Count
in    ZL,RB_PORT      ;sample the red value (PINB)       (1)
in    YL,G_PORT       ;sample the green value (PINC)     (1)
andi YL,0x0F          ;clear the high nibble             (1)
ldd   color,Z+RED_MEM_OFFSET ;lookup red membership      (2)
in    ZL,RB_PORT      ;sample the blue value (PINB)      (1)
ldd   greenData,Y+GREEN_MEM_OFFSET ;green membership(2)
ldd   blueData,Z+BLUE_MEM_OFFSET    ;blue membership     (2)
and   color,greenData ;mask memberships together (1)
and   color,blueData  ;to produce the final color        (1)
;If some interrupt routine came in and set our T flag in SREG, then we
;need to hop out and blow away this frame's data (common cleanup).
 brts _cleanUpTrackingLine                    (not set…1)
 cp     color,lastColor ; check if the run continues    (1)
 breq  _acquirePixelBlock       ;               (it does…2)
;                                             ___________
;                                            16 clock cycles
;                     (16 clock cycles = 1 pixelBlock time)
;Toggle the debug line to indicate a color change
 sbi   _SFR_IO_ADDR(PORTD),PD6
 nop
 cbi   _SFR_IO_ADDR(PORTD),PD6
 mov   tmp2,pixelRunStart  ;Get the count value of the current pixel run
;Get the current TCNT1 value, reload the pixel RunStart for the next
;run, and calculate pixel run
 in    pixelCount,_SFR_IO_ADDR(TCNT1L)
 mov   pixelRunStart,pixelCount
 sub   pixelCount,tmp2
 st    X+,lastColor        ;Record the color run in the current
 st    X+,pixelCount       ;Line buffer with its length
 mov   lastColor,color     ;Set lastColor so you can figure out when
                           ;it changes.
 Nop                       ;Waste one more cycle for a total of 16
 rjmp   _acquirePixelBlock
```

HOLTEK
www.holtek.com

I/O MTP MCU
HT48EXX

Universal Multi-Programmable MCUs for Intelligent Control

Single pack development kit to get beginners up and running with Holtek devices asap!

- Superior Price/Function Ratio
- Industrial Specifications: -40°C ~+85°C
- High noise immunity and ESD protection
- Internal Data EEPROM
- Wide Operating Voltage: 2.2 ~ 5.5V
- Up to 56 I/O lines
- Compatible with OTP/Mask type devices (HT48E06 excepted)

Part No.	Program	Data Memory	Data EEPROM	I/O	Timer	Interrupt	Pins
HT48E06	1Kx14	64x8	128x8	13	8bitx1	2	18/20
HT48E10	1Kx14	64x8	128x8	19	8bitx1	2	24
HT48E30*	2Kx14	96x8	128x8	23	8bitx1	2	24/28
HT48E50	4Kx15	160x8	256x8	33	8bitx1,16bitx1	3	28/48
HT48E70*	8Kx16	224x8	256x8	56	16bitx2	3	48/64

*Under development, available in May, 2005.

Order your Holtek In-Circuit-Emulator today and get 10% discount

(valid for USA and Canada customer only; offer expires on 12/31/2005; discount code: FHSIUCC)

HOLTEK SEMICONDUCTOR INC.

USA Sales Office : Holmate Semiconductor, Inc.

46712 Fremont Blvd., Fremont, CA 94538 Tel: (510) 252-9880 Fax: (510) 252-9885

www.holmate.com

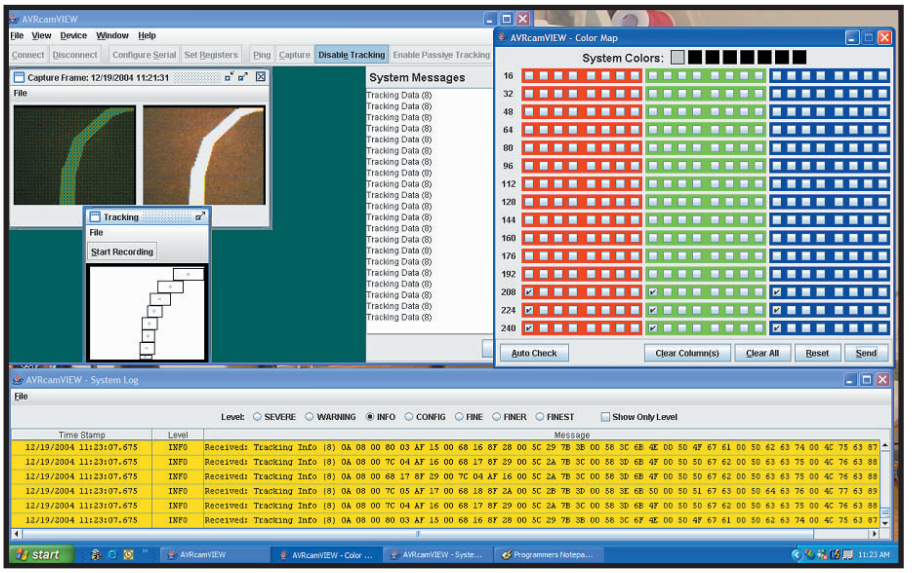


Photo 2—*Take a look at the real-time tracking results from the AVRcam. It's tracking a white line on a black background, which has been segmented into eight different tracked objects.*

tion, you can tweak the map to allow for more variation in a particular color channel. After it's downloaded to the AVRcam, it will use the new color map the next time color tracking is enabled.

When tracking is enabled, a series of tracking packets containing the number of currently tracked objects, the colors of each tracked object, and the bounding box coordinates of each tracked object are sent via the user interface. Up to eight objects can be tracked simultaneously. The AVRcamVIEW application allows you to see the tracking results in real time as objects are being tracked.

The application also provides a time-stamped log of all the packets from the AVRcam. You can save the log as a text file or XML file specific to the AVRcam protocol. This will enable you to analyze the tracking packets at a later time. Photo 2 is a screen shot of the AVRcamVIEW application tracking multiple objects simultaneously.

ChiBots TEST

After I had the AVRcam working in various test environments, it was time to give it a real-world test. The Chicago Area Robotics Group (www.chibots.org) holds a robot contest twice a year. Each ChiBots contest consists of several different events, including basic and advanced line following, mini sumo wrestling, solar roller, and maze solving. I had entered my EyeBo robot (now in version 3) in the line-following contests in the past, but now it was time to strap the AVRcam to my robot and see how it would fare against the competition in the ChiBots 2004 competition (see Photo 3).

The line-following contests require each robot to follow a white line on a black background. Each robot must complete three laps around the course. The robot that completes the laps the fastest wins the contest. Typical entries use infrared sensors facing straight down to determine if the robot is straying off the line and to perform the needed adjustment. I attached my AVRcam to the EyeBo-3 so that it could see approximately 2′ ahead and track the line visually. The color map was set to track the color white, thus providing bounding box information about the line.

In practice, however, the white line appeared as one tracked object without enough contour information to describe the orientation of the line ahead. I made a simple, one-line firmware modification to the AVRcam to force the system to start tracking a new object after the currently tracked object reached a vertical height of 17 pixels. This resulted in the AVRcam tracking eight white bounding boxes that followed the contour of the tracked line (see Photo 2).

I arrived early at the ChiBots contest site so I could take snapshots of the course with the AVRcam. This helped me determine how white the line appeared to the system. The course seemed much more difficult than previous ones, so I was excited to set the EyeBo-3 free to see how it would hold up (see Photo 4).

When it was my turn to run, I nervously powered up my robot. It immediately locked on the line and started following it. When it approached the zigzaging section (normally, the most difficult portion of the course), it zipped right through it without skipping a beat. The junctions proved to be a challenge because they weren't all 90° angles. The EyeBo-3 took a wrong turn once or twice, as did all of the other robots.

My EyeBo-3 ended up winning second place in the advanced line-following contest. The AVRcam worked exactly as I had expected. It allowed the robot to see the course and make decisions based on the visual information.

Now all that's left is to figure out how to use the tracking information to better follow the line ahead. I guess that'll have to wait for EyeBo-4 and the next contest.

A LONG, STRANGE JOURNEY

The AVRcam is a great project. It allowed me to have a crack at building a low-cost, embedded vision engine that enables my robots to see the world around them.

The system has a lot of potential. I'm thinking about including motion detection and object recognition. Because of the open-source nature of the project, anyone can experiment with the code and add features as needed.

I'm thinking about adding a generic analog video input to allow for an NTSC or PAL video signal to be used as the image source. This would allow

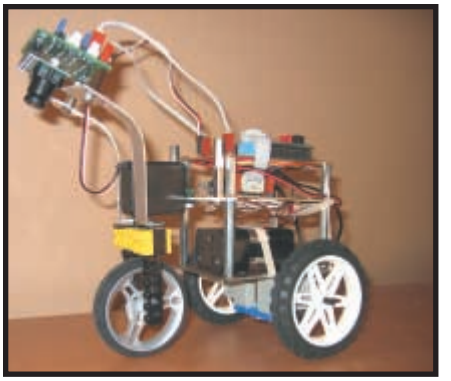


Photo 3—*The EyeBo-3, with the AVRcam, was my entry in the ChiBots 2004 line-following contest.*

Photo 4—*Take a look at the advanced line-following course for the ChiBots November 2004 competition. Looks like fun, doesn't it?*

the system to work with any of the small video cameras that are abundant on the market today. But such a project may require a step up from the good ol' AVR. I guess I'll have to wait and see.

Author's note: I'd like to thank my wife Lindsay for putting up with my robotics addiction. She knew how important this project was to me and gave me the time I needed to make it work. I'd also like to thank my mom, Anne Orlando, for teaching me the meaning of hard work and perseverance. Her example has provided me with a foundation like none other. Finally, I'd like to thank my friend Brent Taylor for developing the AVRcamVIEW application. It turned out great!

John Orlando received his B.S.E.E. from the Rose-Hulman Institute of Technology in 1998. He is currently finishing his M.S. in computer science at the Illinois Institute of Technology. John is a software engineer for the Applied Technology research group at Motorola. He also owns JROBOT (www.jrobot.net), which specializes in various robotic endeavors. He is an active member of the Chicago Area Robotics Group. You can reach John at john@jrobot.net.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/177.

REFERENCE

[1] J. Bruce et al., "Fast and Cheap Color Segmentation for Interactive Robots," Carnegie Mellon University, Pittsburgh, PA, www-2.cs.cmu.edu/~trb/papers/wirevision00.pdf.

SOURCES

ATmega8 and ATtiny12 MCUs
Atmel Corp.
www.atmel.com

OV6620 CMOS sensor
OmniVision
www.ovt.com

WinAVR
SourceForge
http://sourceforge.net/projects/winavr

SX28 Microcontroller
Ubicom
www.ubicom.com

Simple USB Data Acquisition

Simple data acquisition is only a project away. Bruce shows you how to build a simple data acquisition device around an LPC2138. The system features a simple GUI that allows you to view graphed data instead of the streaming serial data in a terminal emulator session.

Just ask any of my friends, and they'll tell you I'm definitely an embedded system nut. I love trying out the latest microcontrollers and chips that can breathe new life into my designs. A couple of my current favorites are Philips ARM-based microcontrollers and USB-to-UART bridges. I incorporated both of these types of devices into my last few designs and I've been extremely impressed with the results.

Another recent addition to my bag of tricks has been on the front end of my designs. Adding simple PC graphical user interfaces (GUI) that can communicate with my embedded designs has put the finishing touches on them. By adding a nice PC GUI that can communicate with the embedded system over a serial port, you can perform things like system setup, real-time diagnostics, and tests. Besides these benefits, your end user or customer will have a more professional, user-friendly interface to work with.

After thinking about ways to combine all of this in a single project, I decided to build a simple USB data acquisition project. The system collects temperature data from an analog temperature sensor and graphs it via a PC GUI. Everyone wants to collect data of some sort (temperature in my case). And what better way than over USB via an ARM-based microcontroller? Of course, taking the data and doing something with it is also an important part of the process, so I'll show you a PC GUI. By the end of the article, you'll be able to create your own simple USB data acquisition device. Most importantly, though, you'll know how to develop with an ARM-based microcontroller, how to use a USB-to-UART bridge, and how to make a PC GUI to tie it all together.

SYSTEM OVERVIEW

I usually design my own boards, but for this project I used a couple of evaluation boards to implement my minimal USB data acquisition system. The boards are readily available, so a hardware design isn't required to get up and running.

The system is comprised of two boards, an analog temperature sensor, and a PC running the GUI (see Figure 1). The Keil MCB2130 evaluation board contains the new ARM-based LPC2138 microcontroller (see Photo 1). The MCU reads the temperature sensor's analog output voltage via its ADC and sends the reading via its UART. For this particular application, I used the board's serial port circuitry (RS-232 transceiver and connector), expansion connector (for hooking in the temperature sensor), and power input connection. One of the board's neat features is that it's powered from an on-board USB connector. This means you don't need a clunky wall wart to power the system. You can just run another USB line to it for power. This is a definite advantage to using USB, as long as your board doesn't draw more power than the USB connection can handle.

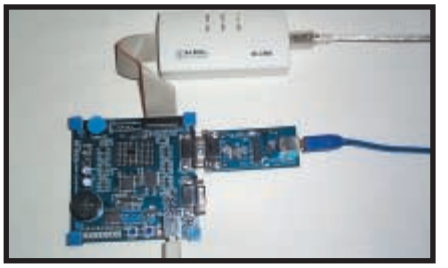


Photo 1—*I used a Keil ULINK JTAG debugger to in-circuit debug and program the LPC2138 microcontroller. I soldered an LM60 temperature sensor to the prototyping area of the MCB2130 board.*

The Silicon Labs CP2101 evaluation board contains the CP2101 USB-to-UART bridge chip and an RS-232 transceiver. This allows you to plug in an RS-232 communicating device on one side and a USB communicating device on the other. The board and its virtual COM port software drivers form the link between the MCB2130 board's RS-232 port and the PC's USB port.

The National Semiconductor LM60 is a simple three-pin analog Celsius temperature sensor. It's wired into the expansion connector on the MCB2130 board, which connects to the LPC2138's ADC to read the analog voltage from the sensor. Its output is

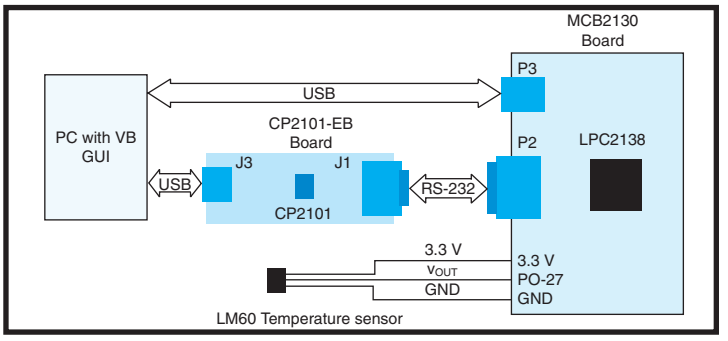


Figure 1—*Where does the power come from? The USB is used for more than just communicating with the PC; it's also used to power both boards, which enables you to remove those ugly black wall warts. The schematics are posted on the* Circuit Cellar *ftp site.*

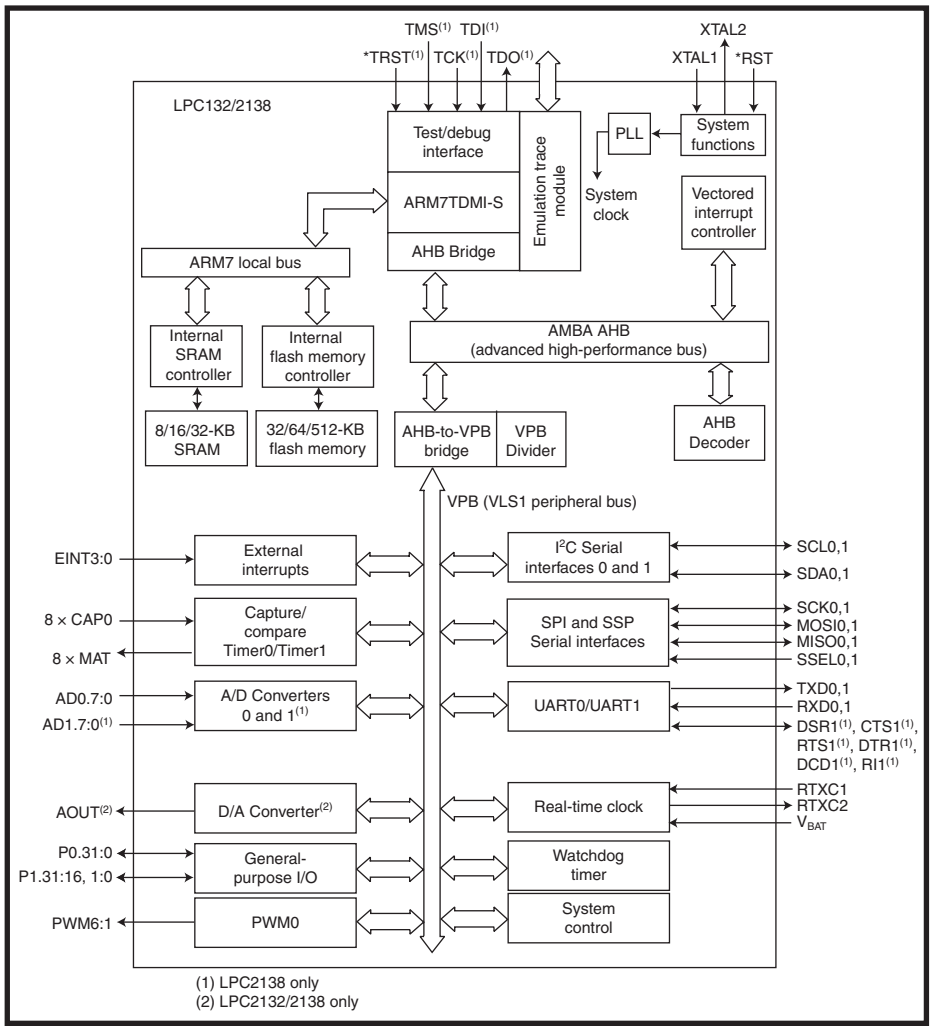


Figure 2—*A 32-bit ARM7 core lies at the heart of the LPC2138. Given the chip's high-performance core and numerous peripherals, it can cover a vast number of applications.*

linearly proportional to temperature (6.25 mV/°C), and it has a DC offset of 424 mV to accommodate negative temperatures. This makes it a fairly easy sensor to deal with in software after it's read via the LPC2138's ADC.

The PC contains the Visual Basic GUI. It reads the raw temperature data sampled by the ADC over the virtual USB COM port, converts it to temperature, and displays and charts the results over time. The GUI puts the finishing touches on the design, making it a more user-friendly and professional-looking system. Imagine how much easier it will be to look at graphed data instead of the streaming serial data in a terminal emulator session.

LPC2138 MCU

The LPC2138 is one of Philips's newest ARM-based microcontrollers. Having previously designed with the LPC2106, the LPC2138 piqued my interest given its vast assortment of added peripherals. The addition of ADCs, DACs converters, an external memory controller, and edge-sensitive interrupts made it the perfect migration part for my LPC2106 designs (see Figure 2).

The small LPC2138 contains everything but the kitchen sink. In addition to a ton of peripherals and general-purpose I/O, it's loaded with 512 KB of flash memory (128 bits wide for high speed) and 32 KB of RAM—definitely not the typical memory sizes I'm used to seeing in plain-vanilla 8-bit microcontrollers. Another remarkable feature is the chip's size. The 64-pin QFP part measures in at 10 mm × 10 mm, making it perfect for tightly spaced applications.

And then, of course, there's the one thing that makes this microcontroller shine: an ARM 32-bit ARM7TDMI-S core. This 32-bit ARM core yields 54 MIPS when running at 60 MHz, which is easily achieved by utilizing the LPC2138's on-board PLL. So, not only do you get a vast number of peripherals and tons of memory, you get all the benefits of using an ARM core! What are the benefits, you ask? An obvious one is its high performance and low-power consumption combination. Others are its vast software tool support, real-time debugging, and code density options (Thumb) for high-volume applications with memory restrictions.

The ARM core definitely has found its way into numerous applications via microprocessors, ASICs, SoCs, and FPGAs. And now, with its growing use in cost-effective microcontrollers, I may think twice before choosing a performance-limited 8-bit microcontroller for my next application. Either way, if you're into embedded design, there's no doubt that having ARM experience under your belt would be beneficial to your career. ARM is an interesting and detailed topic in and of itself. Refer to the Resources section of this article for more information.

I hope I've piqued your interest in the LPC2138. Now let's examine how the LPC2138 fits into the USB ARM data acquisition (DAQ) design.

DAQ VIA ARM

In this simple USB ARM DAQ application, the LPC2138 must read analog temperature sensor voltage at a timed interval. The data must be formatted and then sent to the MCB2130 board's serial port. Therefore, you must use the LPC2138's timer peripheral for the interval timer, its A/D peripheral for reading the analog temperature sensor voltage, and its UART peripheral for serial communication.

The LPC2138's timer is a 32-bit timer/counter with a programmable 32-bit prescaler. It's an extremely flexible timer given its capture channels, match registers, external outputs, and interrupt capabilities. I preloaded a timer match register for this application. The LPC2138 generates a timer interrupt when the timer counter matches this value. Its A/D converter is a 10-bit successive approximation

A/D converter. A 10-bit reading of the analog temperature sensor provides more than enough resolution for the USB ARM DAQ example.

The LPC2138's UART is your typical UART with data rate generation, but it also includes 16-byte receive and transmit FIFOs for added flexibility. Given that temperature data is sent out every few minutes, the data rate is set to 9,600 bps. Now that you're familiar with the LPC2138's peripherals, let's move on to the embedded software.

EMBEDDED SOFTWARE

Before writing actual LPC2138 application code, the device needs to be set up after it's powered on. Fortunately, most IDEs will either set this up for you or provide some kind of boot assembly code to handle the task. For this particular project, I used the evaluation version (16-KB code size limitation) of the Keil µVision3 environment. I was pleasantly surprised with its boot-up implementation. The graphical configuration wizard allows you to modify the proper setup registers for your application. This made the boot and start-up process transparent and allowed me to focus on the application itself.

Let's look at the LPC2138's PLL setup as an example. To change the PLL multiplier value on the LPC2138, you must perform a few extra steps after writing the new multiplier and control values to the PLLCFG and PLLCON registers. These steps entail writing 0xAA and then 0x55 to the PLL feed register (PLLFEED). This action loads the PLL control and configuration information from the PLLCON and PLLCFG registers into the shadow registers that actually affect PLL operation. It's basically a good way to prevent accidental changing of the PLL value. This code implementation is taken care of with the provided boot code in µVision3. Punching in the desired multiplier in the GUI automatically updates the boot code. I learned this the hard way in a different IDE when designing with the LPC2106. The point is that using the graphical configuration tool is an easy and fast way to set up the microcontroller so you can start working on your application.

Now that the boot up code is taken care of, let's concentrate on the main application. I chose C language over the native ARM assembly language to write the driver and application code. So, the next step involved writing a C code driver for a timer interrupt, an A/D scan, and the UART. Fortunately, the example C code that came with the µVision3 IDE had examples for all the peripherals. I modified and used them.

The code for each peripheral was extremely straightforward and easy to understand and integrate. Creating the application code, including the C code for each peripheral, resulted in the code shown in Listing 1. In this code a timer match interrupt occurs from the interval timer, and then the AIN-1 A/D channel is read to sample the analog output voltage from the LM60 temperature sensor. The analog temperature data is then masked because only 10 bits are valid because of the 10-bit A/D resolution. Now the reading is ready to be sent out the serial port via the UART through the `printf` statement. Listing 1 is all the code you need to read the LM60 temperature sensor every few minutes and send the raw ASCII-converted A/D result out of the serial port.

You must download code to the board and begin debugging at this point. I used the ULINK JTAG debugger, which integrates nicely with the Keil µVision3 IDE. The debugger connects to the MCB2130 debug connector and communicates directly with the ARM7 core inside the LPC2138 via its EmbeddedICE logic (see Photo 1).

The typical debug options are available in µVision3. Single stepping, watch windows, break points, and memory snooping are all possible with the LPC2138. An interesting item in the µVision3 IDE is the ability to interact with LPC2138 peripherals while the program is idle. A separate GUI can be opened for the various LPC2138 peripherals that allows for interaction and control of them. Things like manually scanning the LPC2138's A/D converter and flipping of one of its GPIO bits are possible. A couple of the windows are shown in

Listing 1—*The C function from the LPC2138 takes care of reading the LM60 temperature sensor via its on-board A/D converter. The data is then sent out the serial port to the PC running the Visual Basic application.*

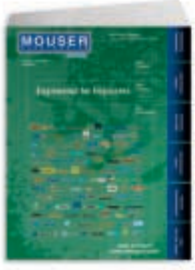
```
void tc0 (void) __irq          //Timer counter 0 interrupt executes
                               //every second at 60-MHz CPU clock
                               //for testing purposes
{
static char LedFlag = 0;
unsigned int AtoDValue;

if (LedFlag == 0)              //Simply toggle the on-board
                               //LED for debug
     {
     IOSET1 =   0x00010000;
     LedFlag = 1;
     }
else
     {
     IOCLR1 =   0x00010000;
     LedFlag = 0;
     }
AD0CR |= 0x01200000;           //Start an A/D conversion
do
     {
     AtoDValue = AD0DR;        //Read A/D data register
     }
while ((AtoDValue & 0x80000000) == 0); //Wait for end of A/D
                                       //conversion
AD0CR &= ~0x01000000;          //Stop A/D conversion
AtoDValue = (AtoDValue >> 6) & 0x03FF; //Extract AIN1 10-bit A/D value
printf ("A%d\n", AtoDValue); //Output A/D conversion result to
                               //serial port

T0IR         = 1;              //Clear interrupt flag
VICVectAddr = 0;               //Acknowledge ARM interrupt
```

NEW Products
NEW Suppliers
NEW Technologies

Delivering the latest with the greatest.

Engineers turn to Mouser to find the latest technologies from industry leading manufacturers. Mouser is the catalog distributor focused on the needs of engineers. We deliver the latest technologies with the great service and support you need.

SEMICONDUCTORS | PASSIVES | INTERCONNECTS | POWER | ELECTROMECHANICAL | TEST, TOOLS & SUPPLIES



Supplier Authorized Distributor

New Products
New Suppliers
New Technologies
New Catalog Every 90 Days!

MOUSER ELECTRONICS
a tti company

(800) 346-6873
www.mouser.com

Photo 2. This is a good way to get to know some of the peripherals and their associated registers on the LPC2138. Given all these features, I quickly downloaded code to the board, ran it, and debugged it.

CROSSING THE BRIDGE

After the code is running on the LPC2138, the raw temperature data exits the MCB2130 board's serial port and meets the USB-to-UART bridge board, which contains the CP2101 USB-to-UART bridge controller and an RS-232 transceiver. The board's power also comes from the USB port, once again eliminating the need for an ugly wall wart power supply. I used the board to convert RS-232 serial data from the MCB2130 board to compatible USB data for the PC.

The CP2101 is highly integrated and requires no components other than a USB connector. It includes a USB 2.0 full-speed function controller, USB transceiver, oscillator, EEPROM, and asynchronous serial data bus (UART) with full modem control signals. The device's packaging is an unbelievably compact 5 mm × 5 mm MLP-28. I've soldered many surface-mount components under the microscope on prototype boards, but this device was by far the trickiest, especially because it doesn't have external leads! (If you plan on soldering by hand, make your PCB footprint pads a bit longer to allow for better solder flow with a fine-tipped iron.)

Looking toward the software end of things, the nice part about using this device is that special software isn't needed for the RS-232-to-USB conversion. This allows USB communication to become totally transparent for the LPC2138 and its UART. Just connect the LPC2138 UART pins to the CP2101, and it will take care of the rest.

You've probably guessed that there must be some software intervention for CP2101 data to get to the PC over USB. Yes, there is. It's via a virtual COM port driver installed on the PC side. These drivers make your USB port seem like another COM port on your PC's operating system (thus the virtual COM port name). Silicon Labs provides the virtual COM port drivers with its development board for Windows, MAC, and Linux.

The interesting thing about the drivers is that existing PC applications, like terminal emulators, will work with them. You can have a terminal session over USB or use existing applications that use COM ports to talk over USB. The Visual Basic PC application does this.

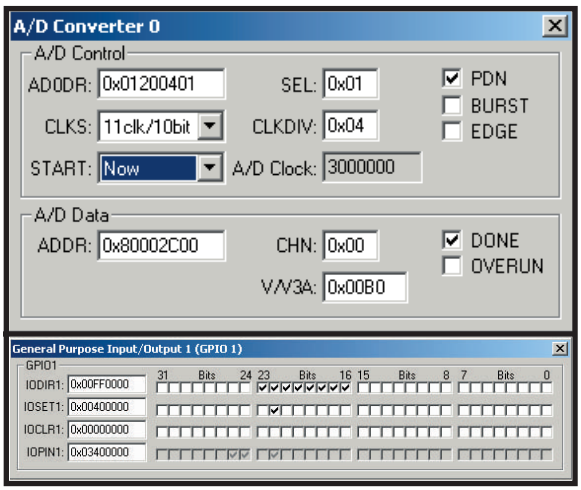


Photo 2—*The debug peripheral windows in the Keil debugger were useful when I was experimenting with the ADC and GPIO port bits. They enabled me to scan the ADC for the analog temperature value prior to connecting up the GUI and to alter the state of the GPIO bits at will. This eliminated the need for writing special test code.*

DAQ GUI

The user interface is the final piece of the USB ARM DAQ system. It's the finishing touch that gives the system a professional-looking way to view the temperature data. The serial data feeding the GUI comes from the USB port and originates from the CP2101 bridge board. The PC application thinks the data is coming over a standard COM port. In reality, however, it gets data from the USB port via the Silicon Labs virtual COM port driver. This gives you the benefit of using the USB port for communication without all the complexity because it looks like just another serial COM port.

The GUI was developed in Visual Basic. If you're familiar with BASIC programming languages like Qbasic, you'll probably find the migration to Visual Basic to be fairly straightforward. Writing code for various actions or events, like receiving serial characters or a button click, is extremely simple. For this application, the goal was to take serial data from the virtual COM port, convert it from a raw temperature A/D value to a real temperature (Celsius), and then graph and display it. Visual Basic provided the control components for the graphing and the serial data communications, and I wrote the temperature conversion code.

The serial data is received over the COM port. It's provided by a control component called MSComm Control in Visual Basic. By simply adding this component to the project, you can set up and open and close serial ports (data rate, etc.). It also allows you to respond to a variety of events, like receiving characters.

The COM port speed is preset to 9,600 bps to match the speed from the MCB2130 board. The receive Comm event will provide a receive character event and allow viewing and reacting to incoming serial characters from the virtual COM port. After the raw temperature A/D data has been received, it can be converted to real temperature data for graphing. This conversion involves multiplying the raw temperature data, which is the LPC2138's A/D sampled voltage, by the A/D reference (3.3 V) divided by 10 bits (2^{10} = 1,024). Following this, the LM60 sensor's 424-mV offset must be subtracted. The value left must be divided by 6.25 (6.25 mV/°C) to get the actual temperature in degrees Celsius, and then converted to an integer.

After the temperature conversion is finished, MSChart Control provides the graphing. By adding this component to the project, a variety of charts and display options are provided for graphing data. Little set-up code is needed because the chart is set up beforehand via the chart properties windows. This is definitely an advantage. You can modify the component controls via a categorized list of options without having to write code to do it.

Combing all the control component interface and application code results in the code in Listing 2 (page 26), which is all you need to accept a serial string from the virtual COM port via USB, convert it to degrees Celsius, and graph and display it. Graphing the data takes only one real line of code!

Innovation and creativity could win you a share of $20,000*

The Renesas M16C Design Contest 2005 is the perfect opportunity to work with the Renesas M16C family. Your innovation and creativity could win you a share of $20,000 in cash prizes!*

The Renesas M16C™ MCU platform is the industry's broadest microcontroller product line, offering seamless code, peripheral and pin compatibility from small 20-pin to large 144-pin devices, all within a single common software development environment. The complete line of highly optimized MCUs delivers practical solutions for your applications.

For the official rules and entry form, go to www.circuitcellar.com/renesas2005m16c.

Deadline: June 30, 2005

Everywhere you imagine RENESAS

M16C PLATFORM™

DESIGN CONTEST 2005

*See the web site for complete details.

RENESAS CIRCUIT CELLAR®

This is a powerful design tool for GUIs. The end result is shown in Photo 3. The temperature data is graphed nicely (the red line), and the current temperature updates every time a serial string is received over the virtual COM port. This puts the finishing touch on the USB ARM DAQ system and makes for a professional-looking demo.

SWITCHING TO ARM?

I hope you now have a better understanding of ARM-based microcontrollers, USB-to-UART bridges, and the process of implementing simple GUIs. You can combine all three to make a simple data acquisition device.

The new ARM microcontrollers like the LPC2138 are opening doors for designers, many of whom are now questioning the use of the venerable 8-bit microcontroller for some applications. When you account for the ARM7 core's processing power, low-power consumption, vast number of peripherals, memory size, tool/debug support, and incredibly small physical footprint, switching to a 32-bit ARM microcontroller may be a reasonable choice.

The USB-to-UART bridges like the CP2101 make it simple to update UART peripherals on microcontrollers (or legacy RS-232 devices) and enable USB connectivity. Embedded code isn't required to make this transition, so the update process is fairly seamless. The virtual COM port drivers provided by companies like Silicon Labs also allow PC applications, such as the Visual Basic GUI I created, to send and receive USB data without additional code overhead.

The Visual Basic-based GUI allows for a professional-looking GUI in an easy-to-use design environment. The built-in component controls put the complex pieces in a simple format that you can easily integrate into your application. The language should be familiar to anyone with BASIC language experience. Explore these topics in greater detail before you begin your project. Good luck!

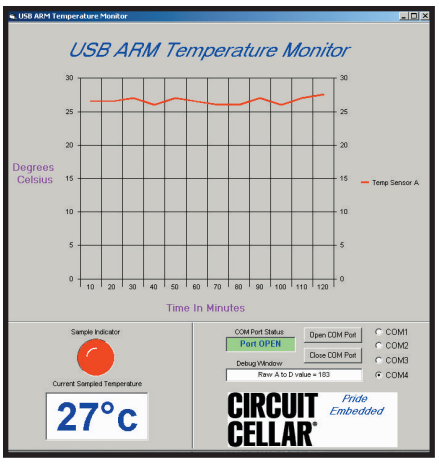


Photo 3—*Watching live graphics updates is a lot more interesting than watching raw datastreams in a terminal emulator.*

Listing 2—*The Visual Basic code takes the serial string from the LPC2138, converts it, and then graphs it to the chart.*

```
Case A_COMMAND                  'Temperature sensor A reading
 CommandInProcess = False       'End of current command
 If (CurrentStringSize >= 1) Then 'Get any characters?
  ArrayCount = 1                'Start of data
For i = ArrayCount To CurrentStringSize
   TemperatureString = TemperatureString & TemperatureSerialData(i)
                                 'Make the string
  Next i
  Led1.LED_Colour = vbRed       'Blink the virtual LED
  Timer1.Enabled = True
  TemperatureDecNum = Val(TemperatureString) 'Convert string to decimal
  DebugText.Text = "Raw A to D value = " & TemperatureDecNum  'Display
                                 'raw A/D value as read from LM60
                                 'sensor via the LPC2138's ADC
  TemperatureDecNum = TemperatureDecNum * (3.3 / 1024) 'Convert
                                 'A/D reading, 3.3-V ref/10 bit A/D
  TemperatureDecNum = TemperatureDecNum - 0.424 'Remove 424-mV offset
  TemperatureDecNum = TemperatureDecNum / 0.00625 '6.25 mV per
                                 'degree C
  Temperature = TemperatureDecNum  'Convert the converted value to
                                 'an integer
  TemperatureText.Text = Temperature & "°c" 'Output the string
  With MSChart1                  'Now lets graph it
    .Data = TemperatureDecNum 'Plot data to current location
                                 '(autoincrement is on)
  End With
End If
End Select                       'Command finished
```

PROJECT FILES

To download the code and schematics, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/177.

RESOURCES

S. Furber, *ARM System-on-Chip Architecture*, Addison-Wesley, Boston, 2000.

R. Martin, "ARMs to ARMs," *Circuit Cellar*, 148–150, Nov. 2002–Jan. 2003.

Philips, "LPC2131/2132/2138," Prelim. Datasheet, rev. 0.1, Nov. 2004.

Silicon Laboratories, "CP2101: Single-Chip USB to UART Bridge," rev. 1.6. 2005.

SOURCES

MCB2130 board and ULINK JTAG
Keil Software, Inc.
www.keil.com

Visual Basic 6.0
Microsoft
www.microsoft.com

LM60 Temperature sensor
National Semiconductor
www.national.com

LPC2138 Microcontroller
Philips Semiconductors
www.semiconductors.philips.com

CP2101 USB-to-RS-232 bridge
Silicon Labs
www.silabs.com

Bruce M. Pride holds an A.S. and B.S. in electronics and currently works as a senior electrical engineer. He enjoys embedded system hardware and software design, particularly designs with 32-bit microprocessors, FPGAs, and various microcontrollers. Bruce spends his spare time with his family, playing acoustic guitar, competing in Circuit Cellar *design contests, and consulting via his side business, Pride Embedded, LLC. You may contact him at Bruce Pride@PridEEmbedded.com.*

Need Device Connectivity To The Pervasive Internet?*

Save Money, Save Time , Use The PowerCore

"Based on experience, I can tell you that Rabbit stuff works before you can get it all out of the box."

–Fred Eady, *Circuit Cellar*

The Core of your Embedded System

The PowerCore FLEX is a complete Rabbit microprocessor system with optional features such as a power supply, Ethernet, and a low-cost rugged A/D system. It plugs into a motherboard you design. You pay only for the options you need, and we quickly manufacture it. Development Kits include a powerful software development platform and extensive libraries.

From $29 qty. 5000 **Pay Only For What You Need**

Network and Internet Support

Wi-Fi

Your embedded device can network and serve web pages. Hardware and software supports Ethernet, Wi-Fi, and cellular telephone data networks. TCP/IP and most associated protocols are included in the Development Kit. SSL secure server software and other premium software modules are available at a nominal cost. With the implementation of optional software modules such as RabbitWeb, SSL, PPP, and AES, you can securely and conveniently communicate with your embedded device from anywhere in the world.

Configurable Features

- Microprocessor Speed
- On-Board Power Supply
- SRAM / Flash
- Serial Data Flash
- Ethernet Option
- A/D Converter System
- Battery-Backed SRAM

Build Your Own

Development Kit Includes

- A loaded PowerCore FLEX with all options or non-Ethernet low-end version
- Prototyping board with development area.
- Documentation on CD.
- Dynamic C integrated development environment.
- Serial cable, power supply, and more.

Standard Features

- 39 General Purpose I/O
- 5 Serial Ports
- Real-Time Clock
- Watchdog / Supervisor

* *Pervasive Internet: Embedded devices providing internet connectivity for commercial and industrial applications. Learn more at* ***www.powercoreflex.com***

Buy A Development Kit – From $129!

Includes Dynamic C (IDE) and hardware development tools. For a limited time, get a free copy of "Embedded System Design Using the Rabbit 3000 Microprocessor" with purchase.

www.PowerCoreFLEX.com

$49 Value

RABBIT Semiconductor

2932 Spafford Street, Davis, CA 95616
Tel **530.757.8400**

Automatic Gate Control

Peter's automatic gate control simulation project involves everything from wireless communication to motor control. Read on to learn how to build the controllers and place the sensors.

Keeping students interested in a subject area can be a major headache. Some teachers argue that applied disciplines have the luxury of lab work that stirs interest in students. This may be true, but it's an ongoing challenge, particularly in the dynamic field of electronics.

Years ago, teachers wooed their classes with demonstrations of technologies that were beyond the means of their students. But times have changed. Today, kids can buy off-the-shelf gizmos with the sort of processing power that would have been the envy of the NASA engineers who won the race to the moon.

Here at the University of the West Indies in Barbados, a major requirement in my microcontroller applications course (ELET3150) is a lab. Course lectures are traditionally reinforced by laboratory experiments. Almost every major topic mentioned on a syllabus is allocated a separate experiment to illustrate the principles taught in the classroom. Students taking the course are usually computer science majors (mainly programmers, who have a tendency to think that CPUs come only in desktop and laptop machines). Many of these students would rather take a basic digital course, but they probably haven't had too much hands-on experience at a component level.

Rapidly growing technologies (e.g., embedded control and photonics) demand a more dynamic structure. As the number of lecture topics increases, so does the number of experiments. To avoid a plethora of separate experiments, consolidation is required and a process to combine topics into a single experiment is mandatory.

Prior to instituting this process, a few instructors decided to make an effort to capture the students' attention and hopefully light a fire in their imagination. We agreed that the experiments had to combine topics from the course syllabus and maintain the objective of stimulating student interest. One of the first successful attempts at this was based on a design that included a toy truck and a motorized gate. A small-scale simulation of an automatic remote gate entry system evolved after we purchased a basic toy truck with a battery-operated motor. The result incorporated aspects of wireless communication and networking, optical sensors, motor control, and, of course, microcontroller applications—all of which are covered in our courses.

SYSTEM MODULES

My students work in pairs because many of them don't have experience with microcontrollers. They're first given a basic scenario of how the simulation should unfold. A command center located a safe distance from a gate controls the entry of vehicles into a compound. The students' task is to set up a wireless control system that communicates with the gate and the vehicle.

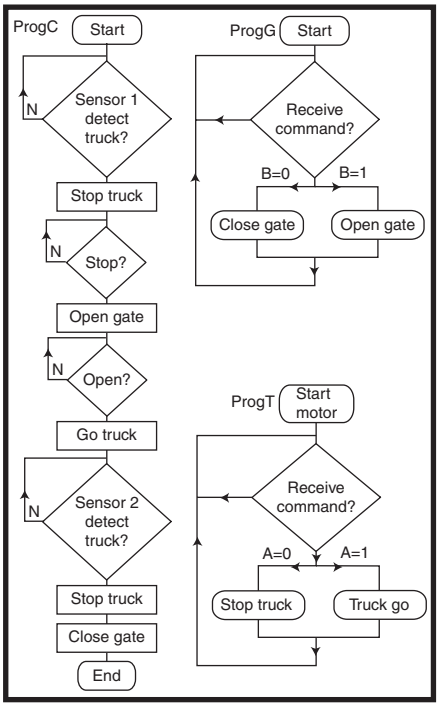


Figure 1—*ProgC is the main program in the command center. ProgG and ProgT handle gate and truck control.*

The project introduces students to the concept of a master/slave multiprocessor environment. During preliminary discussions with tutors, the students soon realize that this is the way to go. The concept of distributed processing is also discussed at this stage.

Next, the three modules of the system are outlined and a basic flowchart is developed (see Figure 1). Each system is equipped with a transceiver, even if both sections (transmitter and receiver) aren't used, in anticipation of future experiments based on these systems, which may require both sections in a module.

Motion and position detectors are added to increase the command center's intelligence. Optical sensors are used, because they are inexpensive, to monitor the truck. The sensors, which operate on a simple proximity reflection principle, should be infrared (not visible) to minimize room light interference. The microcontroller, however, doesn't have interrupt capability, so the virtue of interrupts (hardware) versus polling (software) arises for discussion. Although hardware interrupts win the debate, students are compelled to move ahead knowing they aren't available!

Both truck and gate modules contain motors, one analog and one digital (stepper motor). The students are instructed to examine analog motor speed control and direction principles, which necessitates a discussion of H-bridge operation and PWM control. The trucks, however,

only move forward in this simulation, so a single MOSFET is fine and will employ PWM, which is easily implemented in the microcontroller software by a single command.

The gate module uses a stepper motor. I discuss with students the methods of driving steppers so they learn that a dedicated stepper motor IC controller can ease the main controller's work. This illustrates a simple example of hardware distributed processing, a concept that's developed later in the course.

I cover other concepts too. For instance, I explain the principle of handshaking and its importance in the process of sending and acknowledging commands between the modules. Students quickly realize the need for an agreed upon

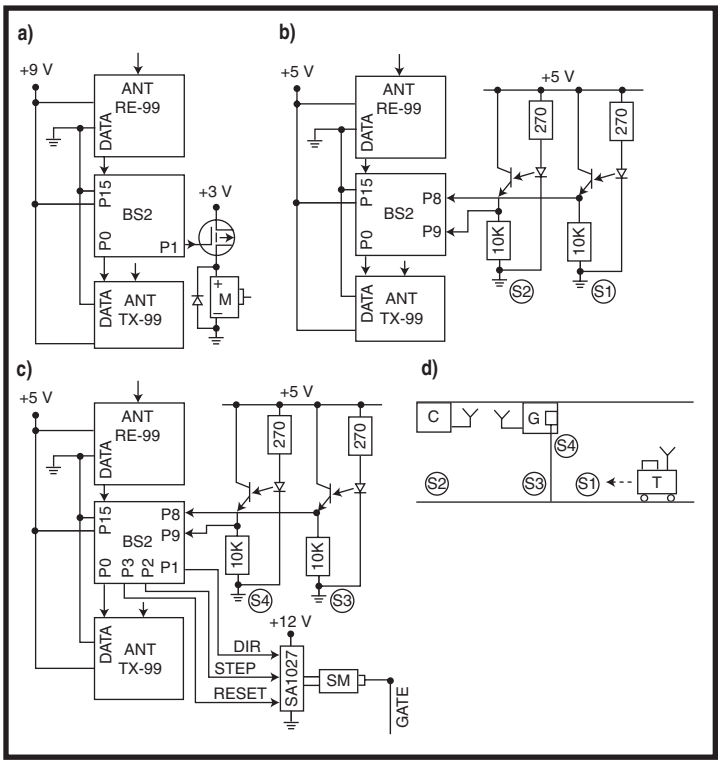


Figure 2—*All three modules include a BASIC Stamp 2, a small transmitter, and a small receiver. The truck controller has an additional MOSFET for turning the motor on and off* ***(a)****. The command center has ground optical sensors* ***(b)****. The gate controller has a stepper motor interface and optical sensors for detecting openings and closings* ***(c)****. The truck approaching from the right crosses sensor S1 and initiates the simulation* ***(d)****.*

protocol (for message transmission), unique addresses for each module in a system, the acknowledgement of a command, and the disasters that can occur when they aren't present and a problem arises. The need for features such as error checking, checksums, and CRC becomes apparent even though our basic simulation doesn't include all of these niceties.

SIMULATION

Let's take a look at how the entry simulation works. As a truck approaches a closed gate, a ground sensor tells the command center to send a wireless signal to the truck to stop it. The truck stops before the gate and signals its status to the command center, which then signals the gate control to open the

gate. It then waits for the gate control to indicate that the gate is indeed open.

After receiving a gate-open signal, the command center tells the truck to proceed forward. After the truck passes the gate, it triggers another ground sensor, which is monitored by the command center. This indicates that the vehicle is now safely inside the gate. The command center then tells the truck to stop and it informs the gate control to close the gate. The experiment is over after the control closes the gate and informs the command center.

Figure 2 shows the apparatus and connections. Figure 3a is a schematic of the truck controller with a single additional MOSFET for turning the motor on and off. Figure 3b is a diagram of the command center with ground optical sensors. The gate controller has a stepper motor interface and optical sensors for opening and closing the gate (see Figure 3c).

COMPONENTS

BASIC Stamp 2 microcontrollers control the three electronic modules in the design: the command center, the truck, and the gate controller. There's a degree of overkill here. Less powerful controllers can be substituted in, but for quick and easy development, I used an interpreter rather than a compiler.

The modules are equipped with a Ming Microsystem wireless transmitter (TX-99) and a receiver (RE-99) pair, which are distributed by Reynolds Electronics. Operating at 300 MHz (AM), the transceiver setup is adequate for the short distances of the experiment. The transmitter requires a short (9.36") wire antenna, whereas the receiver has its own built-in loop antenna. Both modules operate from 5 VDC, drawing a meager 1.6 mA each.

The master microcontroller in the command center handles communication and monitors two ground sensors, S1 and S2, which are reflective optical sensors for detecting the truck position. The microcontroller in the gate control monitors two slotted optical sensors to detect gate open/gate close positions. It also controls the stepper motor that operates the gate.

The microcontroller in the truck receives command center signals and sends back status data. It also controls the small siren circuit that comes with the truck. The virtues of the BASIC Stamp 2 are well known.

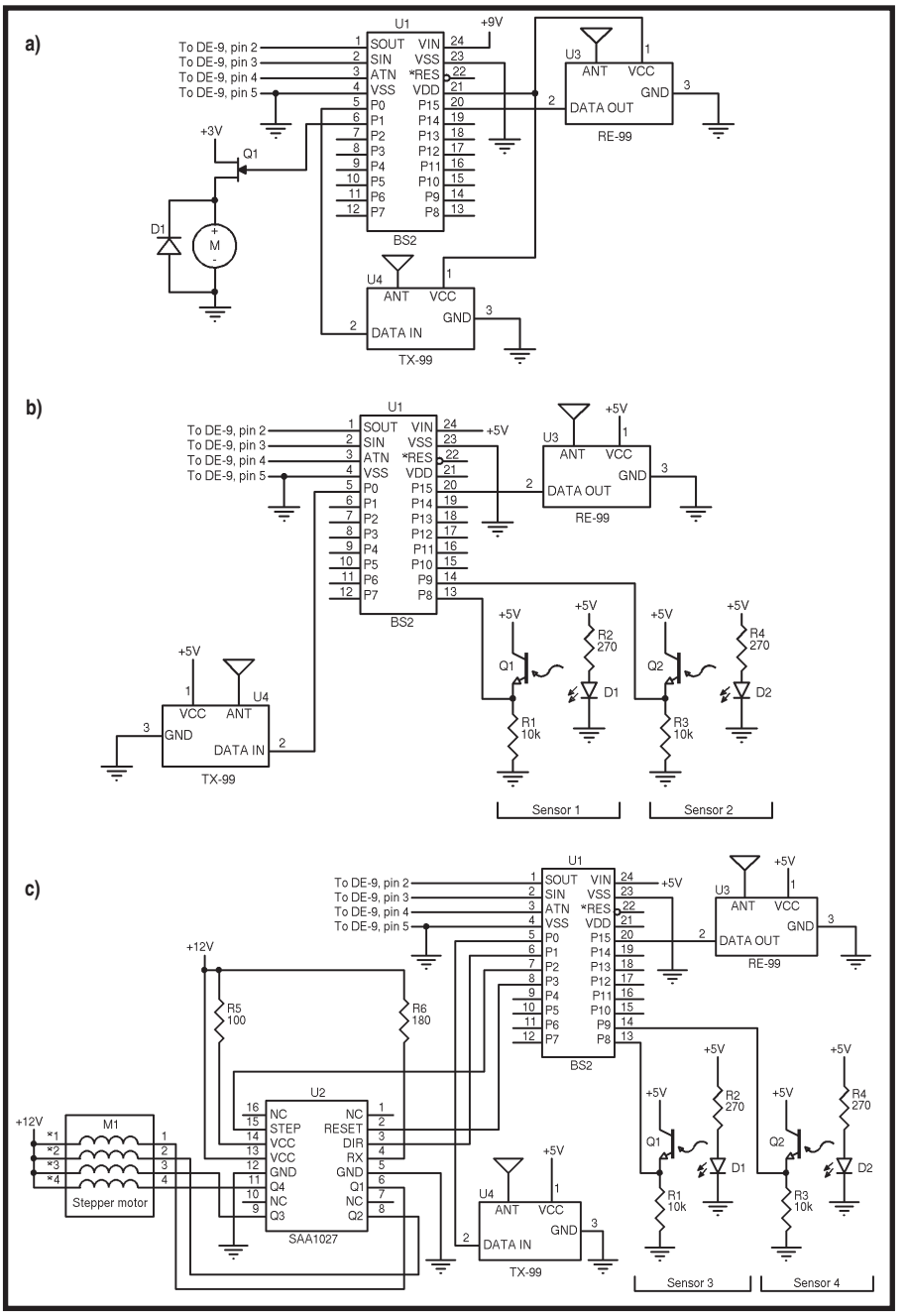


Figure 3—*All the modules have RS-232C connections to a PC for programming. The truck controller uses a power MOSFET to control a small motor* ***(a)****. The command center interfaces to two reflective optical sensors* ***(b)****. The gate controller uses a stepper motor IC to simplify the process of driving the motor* ***(c)****.*

The stepper motor operating the gate, as well as the optical sensors for position direction, is from an old floppy drive. Although an SAA1027 stepper motor driver IC generates the control pulses to operate the motor, you can experiment with other methods. The SAA1027 requires 3 bits to operate its functions: step, direction, and reset. Photo 1 shows the layout and main components.

CIRCUITRY

The setup is mounted on a 6′ × 1.5′ × 0.75″ sheet of wood. The circuit boards are screwed into the base with 0.5″ plastic spacers.

The command center consists of a BASIC Stamp 2 board, a Ming RX-99 receiver and TX-99 transmitter, and optical sensors S1 and S2. Bit 0 connects to the TX-99's data input from the BASIC Stamp 2 board. Bit 15 receives data from the RE-99, while bits 8 and 9 receive input status from the optical sen-

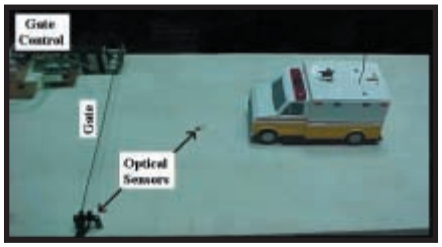


Photo 1—*The simulation starts with the truck approaching the plastic gate attached to the stepper motor's hub. A wire antenna sticks out the top of the truck. A safety switch (on/off) is on top of the truck.*

sors—S1 and S2, respectively—regarding the truck's position. The circuits are powered by 5 VDC (see Figure 2b).

The gate controller consists of a BASIC Stamp 2 board, a Ming RX-99 receiver and TX-99 transmitter, optical sensors S3 and S3, a stepper motor driver board, and a motor. Bit 0 connects to the TX-99's data input from the BASIC Stamp 2 board. Bit 15 receives data from the RE-99. Bits 8 and 9 receive input status from the optical sensors—S3 and S4, respectively—regarding the gate's position. Bits 1, 2, and 3 control the direction, step, and reset signals to the stepper motor IC (SAA1027). The motor and IC are powered by 12 VDC. The remaining boards are powered by 5 VDC (see Figure 2c).

The truck controller consists of a BASIC Stamp 2 board, a Ming RX-99 receiver and TX-99 transmitter, an N-channel FET that controls the small analog truck motor, and the siren circuit board. Bit 15 receives data from the RE-99. Bit 1 controls the siren. Bit 0 controls the motor's speed via the Stamp's PWM command. A 9-VDC PP9 battery powers the BASIC Stamp 2. The BS2 regulated 5-V output (pin 21) powers the RE-99. The truck's motor and siren are powered by a 3-VDC battery system (two type AA) mounted under the truck (see Figure 2a). Photo 2 shows the truck's components.

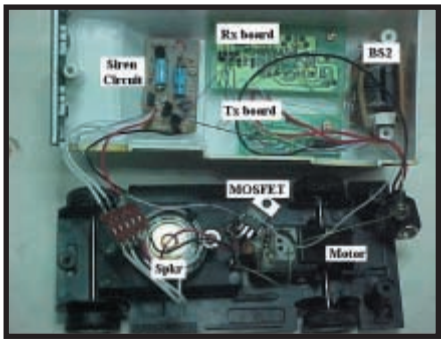


Photo 2—*The truck's siren, speaker, and motor are original components. The modules are new. The 9-V battery is hidden under the BASIC Stamp 2. The 3-V motor battery system is in a small compartment under the speaker.*

SOFTWARE

Three separate STAMP programs are required, one for each of the three modules. Try developing a minimal working system before adding advanced features. Sample programs are posted on the *Circuit Cellar* ftp site: ProgC, ProgG, and ProgT. ProgC is the master program. ProgG and ProgT are slave programs.

Following power-up and port initialization, all of the programs enter polling loops. ProgG and ProgT wait for commands from ProgC, which checks if the gate is closed and waits for the truck to approach. The simulation is now ready to begin! The event initiates when the truck crosses optical sensor S1. This causes ProgC to fall out of its polling loop and immediately transmit a stop command to ProgT.

You should experiment at each

stage with various methods to determine if a specific task/command was successful. Did the truck stop? ProgC may assume the command was successful. To be sure, repeat the command twice, and wait 2 s. (An errant truck would have passed S1 by now.) Next, check S1. If S1 detects a truck, assume that the truck has stopped. Otherwise, assume it hasn't stopped. Go to start, modify, and try again! An errant truck will crash into the gate at this stage unless you call out the Marines. Pick up the truck and check the system for errors. Try another approach.

Assuming the truck stops, ProgC delays and then transmits an open gate command to ProgG. A check is performed to determine if the command was successful. Repeat the command twice and wait 1 s. (The gate should have moved up by now.) Next, check S3 to determine if gate still closed. If S3 detects an open gate, assume the gate is open, or else assume it didn't respond. Go to start, modify, and try again. Once again, you must troubleshoot the system.

After the gate opens, ProgC waits for ProgG to send a gate open confirmation. After receiving the signal, ProgC transmits a go command to ProgT for the truck to proceed into the compound. Again, the command repeats, and then ProgC proceeds to poll optical sensor S2. This indicates that the truck has moved past the gate and is inside the compound.

After ProgC detects that the truck is inside, a stop command to ProgT stops the truck. Then a close gate command transmits to ProgG. ProgC waits for confirmation from ProgG and its work is done.

After initialization, ProgT waits endlessly for commands from ProgC. Only two commands are understood: stop and go. After executing a command, a confirmation of receipt status is sent back to ProgC. Now its back to the polling loop.

Like ProgT, ProgG also waits endlessly for commands from ProgC. Only two commands are understood: open gate and close gate. After executing one, a confirmation of receipt status is sent back to ProgC and polling begins again.

These three basic controlling programs are enough to test that the simulation works. Now you can add bells and whistles. For instance, you can turn on the siren whenever the truck is in motion. Attach a spare receiver to a PC and the PC will monitor all transmissions and then display the progress of the simulation as it unfolds, textually and graphically. I implemented the former and it has proven to be extremely useful for debugging. With lots of unused bits on the controllers, start adding some LEDs to indicate status, maybe headlights for the truck. Use your imagination!

IMPROVEMENTS

Improvisation is the keyword for low-budget projects. You can always scrounge parts from old equipment. It isn't hard to find unused disk drives, hard drives, CD-ROMs, printers, and other peripherals.

Most of the problems associated with this project are related to unreliable data reception. The transceivers are extremely inexpensive. When you apply power, the transmitter section will transmit the carrier. You'll now have three transmitters of identical frequency flooding the airwaves and

without T/R switches. The receivers will get blasted close up, but the basic system will work (to a certain degree).

You could use a few of the microcontroller port bits in each module to control the power to the transmitter and receiver. For instance, when the command center is transmitting, its receiver section is powered down, as are the transmitter sections of the truck and gate. These modifications will immediately improve data communication.

Using an MCU with interrupt capabilities will eliminate wasteful software polling loops. Improving the communications protocol and adding error-correcting techniques and checksums will create a more reliable system. Replacing optical detection with ultrasonic sensors will eliminate the process of planting sensors in the truck's path. Using FM transceivers rather than AM ones will surely improve reliability. And the list goes on.

I've asked my students to suggest improvements like these in their reports. Some have gone a bit further. A student recently suggested planting a solenoid under the truck's path in front of the gate. The idea is to destroy hostile vehicles that attempt to enter the compound. Deadly force!

PERFECT PROJECT

This project has excited my students. Most of all, it has generated meaningful answers to the "what if" questions they've asked along the way. Many students now realize that they can use a computer for more than surfing the Internet and playing games. They now know that the majority of processors go into control applications, not PCs. When they begin, they view the experiment as a simple modeling and simulation exercise. But it quickly dawns on them that it could easily become complicated.

My students have willingly committed extra time to this project, despite its simplicity. I think it's the sort of spark that can ignite a lifelong interest in design. How do I know? I still remember the excitement of turning on and off my first LED under TRS-80 software control.

Peter Gibbs is a senior lecturer in physics and electronics at the University of the West Indies (Cave Hill) in Barbados. He holds bachelor's degrees in physics and math as well as a diploma of education from the University of the West Indies. He also earned a master's degree in physics from the University of Guelph (Ontario). When he isn't working on embedded systems control apps, he enjoys long-distance swimming. He recently swam across Lake Ontario in 18 h, 40 min. You may contact him at pgibbs@uwichill.edu.bb.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/177.

SOURCES

BASIC Stamp 2
Parallax Inc.
www.parallax.com

SAA1027 Stepper motor driver
Philips Semiconductors
www.semiconductors.philips.com

RX-99 and TX-99 RF modules
Reynolds Electronics (distributor)
www.rentron.com

Embedded Security Design (Part 2)

Circuit Board

In the first part of this series, Joe described superficial design solutions that will help protect your embedded products. This month he sheds light on security issues related to the circuit board. He's got your entire design covered.

Welcome back! Last month I described security-related design solutions for your product's enclosure. My goal was to help you understand and mitigate some of the risks associated with attacks against your product. Designing the right enclosure can prevent tampering and reduce the possibility of passive attacks. This month I'll peel back a layer of the embedded system and describe security issues on the circuit board level. Many of the weaknesses, security vulnerabilities, and design flaws of a product are identified when you analyze the circuit board.

There are a number of things you can do to make attacks against a product more difficult. To help you understand what you're up against, I'll describe a few successful hardware attacks. By studying such attacks, you'll learn how to improve your products.

CIRCUIT-LEVEL ATTACKS

Basic circuit board-level attacks range from the modification of microprocessor content to the replacement of components. More advanced attacks involve microprobing (i.e., a chip package is opened, its internals are accessed with semiconductor test equipment, and the internal data paths are observed or manipulated) or fault generation attacks in which the device operates under environmental stress conditions outside its designed operational range (e.g., extreme temperature, supply voltage variations and spikes, protocol violations, and partial system resets).

Nothing is ever 100% secure. No matter how many layers of security you use, a determined attacker can undermine your product. As gloomy as this sounds, your challenge is to figure out what you need to protect in your product and how much it's worth to protect it.

REVERSE ENGINEERING

Reverse engineering a product usually requires knowledge of the part numbers and functionalities of the major components on the board. Understanding what the components do may provide details about particular signal lines that may be useful for active probing during operation. The part numbers and the manufacturer's marking on the package easily identify components. An attacker can also follow their traces to see how they interconnect with other components. Nearly all IC manufacturers post component datasheets on the 'Net. On-line services like IC Master (www.icmaster.com) and Data Sheet Locator (www.datasheetlocator.com) provide part number searches and pinout and package data for hundreds of thousands of components.

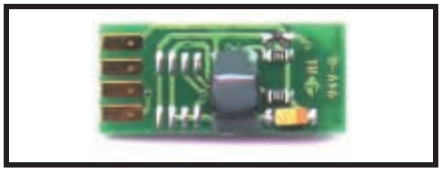


Photo 1—*Early USB authentication tokens improperly used epoxy encapsulation to cover a serial EEPROM. The adjacent footprint can be used to read the contents of the protected device.*

To increase the difficulty of reverse engineering and device identification, I recommend scratching off the marks on the top of your chips. This is known as demarking, or black topping. Depending on the quantities of ICs you are ordering, you may be able to ask the manufacturer to leave the markings off of the devices you order. The next best option is to remove the markings in-house during the manufacturing process. You can use a laser etcher or demarking machine (typically a controlled microbead blasting process that removes all identifiable markings from the device). You can also use a stainless steel brush or small sander (manual or computer-controlled) to remove the markings.

Using off-the-shelf components make it extremely easy for an attacker to obtain information about your product. Designing with proprietary, custom devices (or at least off-the-shelf cores with custom functionality) will prevent most attackers from easily reverse engineering them. But this solution isn't practical for all designs.

ACCESSING COMPONENTS

Make sure sensitive components that are most likely to be targeted for an attack (e.g., a microprocessor containing product firmware, ROM, RAM, and programmable logic) are difficult to access. Covering specific components with a conformal coating or an epoxy resin can help to prevent tampering.

Conformal coatings and encapsu-

lates are typically used to protect devices from moisture, fungus, dust, corrosion, and tampering. Urethane provides a hard, durable coating that offers excellent abrasion and solvent resistance. It shrinks significantly during curing, which may stress components. Epoxies also offer excellent resistance to moisture and solvents. Usually consisting of a two-part resin, the coating also shrinks during curing, leaving a hard, difficult to remove film that makes it difficult for attackers to probe the device and remove it from the board.

Such products are provided by a large number of manufacturers. Unfortunately, I don't know of any coatings that are specifically designed for security purposes (meaning that an attacker can't remove them without harming the underlying components). Note that chemicals like methylene chloride, sulfuric acid, and fuming nitric acid can remove protective coatings, so be sure your compound is suitable for the protection level you desire.

When conformal coating is incorporated in a design to protect components, make sure that it serves its intended purpose. Photo 1 shows an example of an early USB authentication device that stored critical data on the encapsulated serial EEPROM. Aside from being able to scrape off the epoxy with a hobby knife to gain access to the device's surface-mount pins, an attacker could simply solder wires to the exposed footprint adjacent to the device, which is intended for another serial EEPROM, and read the memory contents using an industry-standard device programmer. The misuse of epoxy coupled with the device's accessibility could result in a successful attack on the product.[1]

Using ball grid array (BGA) packages increases the difficulty of casual probing, manipulation, and attack because all die connections are located underneath the device's packaging. However, debugging with these packages is difficult. Manufacturing is typically more expensive because X-rays verify that the solder has properly bonded to each of the ball leads. A dedicated attacker could remove the target device and add a socket for easier access. As a result, it's recommended to place critical devices in areas of the circuit board that may not have enough area or vertical height around the component for a socket to be mounted properly.

Another solution is to employ chip-on-board (COB) packaging, in which the silicon die of the integrated circuit is mounted directly to the PCB and protected by epoxy encapsulation. Even though methods exist to gain access to COB devices, and even though an attacker can probe vias and traces extending from the encapsulate, direct manipulation with the device and its connections are less of a threat. Using COB devices also increases manufacturing costs and isn't necessarily supported by all manufacturers because specialized equipment is required to manipulate the wire bonds between the silicon die and the circuit board.

A relatively new technology known as chip-in-board (CIB) embeds the silicon die within the layers of a PCB. The concept is similar to COB, although a cavity is created in the circuit board to hold the die. An encapsulate is filled in over the die and cavity, creating a flat PCB surface. I don't know what kind of financial burden this technology creates.

DIE ANALYSIS ATTACKS

The analysis of IC dies, although commonly done for failure analysis and chip design, long has been the most difficult vector for attack purposes. With access to the die, an attacker may be able to bypass many of the available on-chip security mechanisms to determine the device's content. Such attacks are more likely to occur if the adversary can't easily remove the device from the board but still has space to access the component freely.

Decapsulation products, such as those made by Nippon Scientific (www.nsc-net.co.jp/e) and ULTRA TEC Manufacturing (www.ultratecusa.com), will delid or decap the top of the housing from ICs (using hazardous chemical or mechanical means, or a combination of both) while leaving the die intact and fully functional. Furthermore, after a successful attack, the IC package can be refilled with an epoxy encapsulate and the device will still operate normally.

Equipment has become available on the surplus market and the prices have been reduced to an affordable level (approximately $10,000 to $15,000). Many academic institutions also provide access to such equipment.

After the die is accessible, a technique known as voltage contrast microscopy can be performed with a scanning electron microscope to visually (and passively) extract voltage information from a flash ROM storage cell.[2] It also would be possible for an attacker to physically modify the die (e.g., changing the security bits' settings). Using focused ion beams (FIB), specialist companies like Fibics (www.fibics.com) and FIB International (www.fibinternational.com) can cut bond pads to remove a trace or add ion deposits to add a jumper or set a bit on the die.

Oliver Kömmerling and Markus Kuhn's 1999 article, "Design Principles for Tamper-Resistant Smartcard Processors," details techniques to extract software and data from smart card processors, including manual microprobing, laser cutting, FIB manipulation, glitch attacks, and power analysis. Much of this attack research is based on Friedrich Beck's *Integrated Circuit Failure Analysis*, which details failure analysis techniques for opening the package/chip insulation, etching procedures for removing layers of chip structure, and health and safety procedures. Even though all of these die analysis attacks are easier said than done, the available technology is fascinating and the threat is real.

BUS PROTECTION

Device operation and information can be gleaned by analyzing an embedded system's internal address, data, and control bus lines using a logic analyzer, digital oscilloscope, or custom circuitry. Targeted bus lines could be probed by simply removing the solder mask on the outer layers of a circuit board. Critical traces should be hidden on inner board layers. Trace paths should be obfuscated to prevent the easy reverse engineering of the circuitry. Use buried vias—which connect two or more inner layers (but no outer layer) and cannot be seen from either side of the board—to reduce

potential probing points for the attacker. If a multilayer board isn't used, protective encapsulant should be applied at least to the target traces.

In *Hacking the Xbox: An Introduction to Reverse Engineering*, Andrew Huang describes a tap board used to intercept data transfer over the Xbox's HyperTransport bus. Huang was able to retrieve the symmetric encryption key used for the protection of a secret boot loader, which ultimately allowed him to execute untrusted code on the system. This attack might have been prevented if those critical bus lines were on internal board layers.

MEMORY DEVICES

Most memory devices are notoriously insecure. Some have security features that prevent access to data, whether through fuses in ROMs and boot block protection in flash memory or via a password-protected memory area. Even though they may be able to be bypassed with die manipulation attacks, such features should be used because they will sufficiently raise the bar against attackers who may be trying to clone or reverse engineer the device. In this case, they add an insignificant level of protection.

Atmel's CryptoMemory family of devices includes EEPROMs and synchronous and asynchronous flash memory with authentication, password, and encryption features (www.atmel.com/products/securemem). Most standard memory devices don't have this type of functionality and are readable, often in-circuit, with standard tools.

Reading RAM or other volatile storage areas may yield temporarily stored system data or other useful information (e.g., cryptographic keys or plaintext left behind from an encryption routine). In *Data Remanence in Semiconductor Devices*, Peter Gutmann shows that it's extremely difficult to securely and totally erase data from RAM and nonvolatile memory. This means that remnants of such data still may exist; they may be retrievable from devices long after power has been removed or the memory contents have been rewritten. (Temperature also plays a role in the retention of data, but the retention time is unpredictable even between two of the same memory device.)

Unfortunately, there aren't many practical solutions to avoid the known problems. The current best practice is to limit the amount of time that critical data is stored in the same regions of memory. Storing data in a fixed RAM location can lead to burn-in that will enable data recoverability even if power is removed from the volatile device. Either move the secret around to different RAM locations (while overwriting the previous area) or periodically flip the stored bits of the secret.

PLDs & FPGAs

Depending on the product, protecting your intellectual property inside programmable logic devices (PLDs) and field programmable gate arrays (FPGAs) can be just as important as protecting firmware and data in memory. Essentially, SRAM-based devices are the most vulnerable to attack because of their requirement to have configuration memory external to the device (stored in separate nonvolatile memory or program firmware), which is then loaded into the FPGA at power-up. The bitstream between the configuration memory and FPGA simply needs to be monitored to retrieve the FPGA configuration.

Currently available flash memory-based FPGA devices are arguably more secure than their SRAM-based siblings. Some product types worth investigating further are Actel's Antifuse FPGAs (www.actel.com) and QuickLogic FPGAs (www.quicklogic.com), both of which eliminate the need for external configuration memories required by SRAM-based devices.

With your programmable logic, make sure you implement protection against simple I/O scan attacks, in which an adversary attempts to reverse engineer a programmable logic design by cycling through all possible combinations of inputs and then monitoring the outputs to determine the internal logic functions. This type of attack is easiest against low-density PLDs with dedicated inputs and outputs and for designs containing only asynchronous circuits and latches. A solution is to use unused pins on the device to detect probing or tampering. Pins can be set to inputs, and if they detect a level change, the device can assume it's being probed and can perform a countermeasure or response.

When designing state machines in FPGAs, ensure that all conditions are covered and there are defaults in place for unused conditions. Otherwise, an attacker may be able to put the FPGA into an indeterminate state through fault-generation attacks.

Also, consider adding digital watermarks to your design in the form of unique features or attributes that can be used later, if necessary, to prove that a design claimed to be original by a competitor is actually a copy. Legal means are usually the last resort when protecting your intellectual property, but having some hidden identifier might make your case much stronger.

POWER SUPPLY

Precautions should be taken to prevent the intentional variation of the power and clock. Minimum and maximum operating limits should be defined and protected using comparators or supervisory circuitry (available from manufacturers like Maxim and Linear Technology). Don't rely on the end user to supply a voltage within the recommended operating conditions. Using a low-dropout linear regulator or DC-DC converter (instead of a direct voltage input into your system) will help ensure that your circuit receives power within its expected range, regardless of an improper voltage supplied at the input. Such circuitry can be bypassed if the attacker has access to the board.

An attacker can use power supply variation to gain access to your system. For instance, he can clear the security bit in a PIC16C84 microcontroller without erasing the remaining memory, thus gaining complete access to the once-protected area. This is achieved by raising V_{CC} to 0.5 V_{PP} (approximately 13.5 V) during repeated write access to the security bit.[3]

An attacker also can passively monitor the power supply fluctuations to determine information stored on a device. In "Introduction to Differential Power Analysis and

Related Attacks," the authors describe the process of monitoring the electrical activity of a smart card and using mathematical methods to determine cryptographic keys. Simple power analysis (SPA) is a predecessor to DPA in which an attacker directly observes a system's power consumption, which varies according to the operation the microprocessor's performing. Intensive operations, such as cryptographic functions, can be easily identified.

Although SPA attacks primarily use visual inspection to identify relevant power fluctuations, DPA attacks use statistical analysis and error correction techniques to extract information correlated to secret keys. In "Power Analysis Attack Countermeasures and Their Weaknesses," Thomas Messerges looks at five countermeasures to prevent such attacks, including a noise generator using power randomization, power signal filtering using active and passive filters, detachable power supplies, and time randomization by desynchronizing the clock. He discusses the pros and cons of each.

I/O PORT PROPERTIES

All unused I/O pins should be disabled or set to a fixed state. For example, the Motorola MC68328 DragonBall processor enables by default the clock output (CLKO) pin at reset. CLKO, which is used for testing the internal PLL, outputs a 16.67-MHz sine wave on the pin. If it isn't disabled during normal device operation, the extraneous signal may cause unwanted noise.

Unused I/O pins can be configured to detect probing or tampering by setting them to inputs and waiting for a level change. If one is detected, the device can assume it's being probed and initiate a response. This type of detection mechanism would work only while the device is active, which is the most likely time for probing by an attacker to occur.

In order to prevent against ESD attacks, implement ESD protection devices on any connectors or I/O pins (e.g., keypads, buttons, switches, or displays) exposted to the outside world. ESD protection can simply be in the form of clamping diodes or transient voltage suppressor (TVS) devices. Manufacturers include Vishay (www.transzorb.com/diodes/protection-tvs-esd) and Semtech (www.semtech.com).

CRYPTOGRAPHY

The strength of a cryptography implementation relies on the secrecy of a key, not the employed algorithm. However, it's insufficient to assume that a large key size will guarantee security. Even if a trusted encryption algorithm like DES or AES is used, improper implementation could make the product easy to break. You must have a complete understanding of the requirements and functionality of an encryption solution before it's implemented in a system. Many times, companies will claim to encryption in their product, when in reality, it's nothing more than a simple encoding scheme (usually some type of logical operation like an XOR against a constant block). Rolling your own custom encryption solutions is typically a bad idea because they end up being extremely insecure.

An example of improperly used

encryption is detailed in "DS1991 MultiKey iButton Dictionary Attack Vulnerability," which I wrote in 2001. The DS1991 iButton is a hardware authentication token that has three internal 48-byte data areas, each of which is protected by a distinct password. Only the correct password will grant access to the data stored within the area. Although the marketing literature claimed that incorrect passwords written to the DS1991 would automatically invoke a random number generator that replies with false responses (which eliminates attempts to break security by pattern association), it was determined that the data returned on an incorrect password attempt wasn't random at all. It was calculated based on the input password and a constant block of data stored within the DS1991 device.

The secure coprocessor originated from the notion of a protected subsystem that can execute sensitive functions in a trusted manner. Programmable, secure coprocessors typically contain a number of essential ingredients including: hardware tamper response, randomness, a layered design, self-initialization, authentication, a general-purpose processor and coprocessor, persistent storage, and a third-party programming interface. If possible, any cryptographic functions in your design should be moved out of the firmware and into a dedicated cryptographic device. Physical security is a central assumption upon which secure distributed systems are built. Without a secure foundation, even the best cryptosystem and the most secure kernel will fail.

The IBM 4758 is probably the most recognized, commercially available secure coprocessor. Its design has been presented in a number of academic papers and articles, including Joan Dyer et al.'s "Building the IBM 4758 Secure Coprocessor." Mike Bond and Richard Clayton provided the first known attack against the IBM 4758, which has since been fixed, by taking advantage of a flaw in the common cryptographic architecture (CCA) support software to export any and all of the program's DES and 3DES keys (www.cl.cam.ac.uk/~rnc1/descrack/). Other vendors that provide cryptographic devices include Hifn (www.hifn.com) and SafeNet (www.safenet-inc.com).

THINK LIKE THE ENEMY

It has been said that the only way to stop an attacker is to think like one. Stay aware of the latest attack methodologies and trends, which will enable you to choose the proper means of protection for your particular product and help you keep tabs on what attackers might attempt against your product.

Try to break the security of your product. Fix it, and try to break it again. Allow time for this iterative process during the design cycle. Don't release a version of your product and plan to implement security into a later revision. Properly retrofitting security mechanisms into an existing product is extremely difficult, and political and financial pressures usually prevent it from actually happening.

Although I've only scratched the surface of the topic of embedded security, you can use this series of articles as an embedded security cookbook. You have a lot of security options to choose from for your particular design.

Hopefully, I've left you with the confidence (and a twinge of fear) to design secure products. Good luck with your future projects.

Joe Grand is the president of Grand Idea Studio, a product development and intellectual property licensing firm. He specializes in embedded system design, computer security research, and inventing new concepts and technologies. Joe holds a B.S.C.E. from Boston University. His interests include competitive running, collecting classic video games, and banging on drums. You may reach him at joe@grandideastudio.com.

REFERENCES

[1] J. Grand (Kingpin), "Attacks on and Countermeasures for USB Hardware Token Devices," Proceedings of the Fifth Nordic Workshop on Secure IT Systems, 2000, www.grandideastudio.com/files/security/tokens/usb_hardware_token.pdf.

[2] C. O'Dale, "Integrated Circuit Troubleshooting Using Voltage Contrast Techniques," http://testequipmentcanada.com/VoltageContrastPaper.html.

[3] "PIC16C84 Security," posted April 26, 1995,www.brouhaha.com/~eric/pic/84security.html.

RESOURCES

F. Beck, *Integrated Circuit Failure Analysis: A Guide to Preparation Techniques*, John Wiley & Sons, Hoboken, NJ, 1998.

J. Dyer et al., "Building the IBM 4758 Secure Coprocessor," 2001, www.cs.dartmouth.edu/~sws/papers/comp01.pdf.

J. Grand, "DS1991 MultiKey iButton Dictionary Attack Vulnerability," 2001, www.grandideastudio.com/files/security/tokens/ds1991_ibutton_advisory.txt.

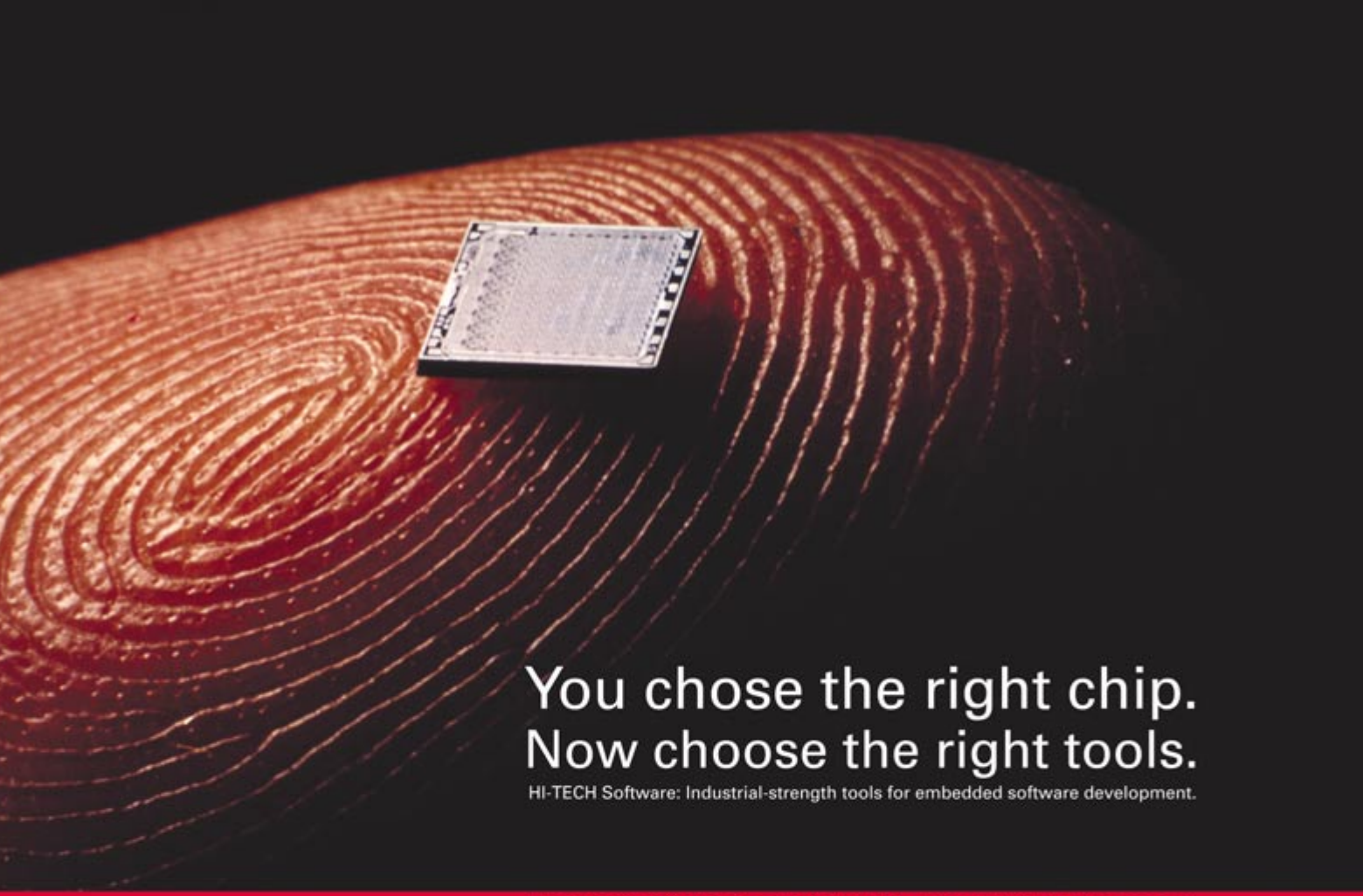
P. Gutmann, "Data Remanence in Semiconductor Devices," Proceedings of the Tenth USENIX Security Symposium, 2001, P. Gutmann, "Data Remanence in Semiconductor Devices," Tenth USENIX Security Symposium, 2001, www.usenix.org/publications/library/proceedings/sec01/gutmann.html.

A. Huang, *Hacking the Xbox: An Introduction to Reverse Engineering*, No Starch Press, San Francisco, 2003.

P. Kocher et al., "Introduction to Differential Power Analysis and Related Attacks," Cryptography Research, Inc., 1998, www.cryptography.com/resources/whitepapers/DPATechInfo.pdf.

O. Kommerling and M. Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors," 1999, www.cl.cam.ac.uk/~mgk25/sc99tamper.pdf.

T.S. Messerges, "Power Analysis Attack Countermeasures and Their Weaknesses," Communications, Electromagnetics, Propagation, and Signal Processing Workshop, 2000, www.iccip.csl.uiuc.edu/conf/ceps/2000/messerges.pdf.



You chose the right chip. Now choose the right tools.

HI-TECH Software: Industrial-strength tools for embedded software development.

Compact, efficient code for ARM, PIC and 8051 processors

Whatever processor family you are targeting, whether it is the ARM, PIC, 8051 or one of the other chip families we support, HI-TECH tools can help you write better code and bring it to market faster. Our reputation is proven, our tools are first rate, and our support staff is the best in the industry.

HI-TECH ARM-C:
The HI-TECH ARM-C is a high-performance C compiler for the powerful RISC microcontrollers based on the ARM architecture, delivering excellent code density and reliability.

HI-TECH PICC:
HI-TECH PICC is the overwhelming choice for developers on the Microchip PICMicro family - supporting every one of the extensive range of PICMicro devices.

HI-TECH 8051C:
The 8051 family is the most successful 8-bit microcontroller family in the industry, available from over 40 manufacturers in a huge range of chips. HI-TECH 8051 C is a field-proven ANSI C compiler supporting over 800 different microcontrollers.

HI-TECH C is also available for several other popular embedded microcontrollers.

You thought long and hard about which chip to use for your next project. It's a crucial decision - choose the right architecture and you just might create a product that changes the world. Choose the wrong one and your project may be off to a slow start. The next critical decision is what tools to use. You know you want compact, efficient code. You know you need reliable and dependable tools from a company with a stellar reputation and first-rate technical support. And you know that it must be a company that has proven that customers can rely on their tools to succeed.

For over two decades HI-TECH Software has delivered the industry's most reliable tools for developing efficient and compact code to run on the most popular embedded processors. Used by tens of thousands of customers including General Motors, Whirlpool, Qualcomm, John Deere and many others, HI-TECH's development tools and world-class support staff have helped serious programmers to create hundreds of successful embedded products.

Find out more or download a demo at http://cc.htsoft.com/

HI-TECH Software LLC 6600 Silacci Way Gilroy, CA 95020 USA
Ph: 800 735 5715 **Web:** http://cc.htsoft.com/

HI-TECH SOFTWARE

Foolish LED Tricks

Ed recently bought a cheap LED flashlight that barely worked. Rather than ask for a refund, he rebuilt the flashlight and learned a lot about LED technology in the process.

A while ago, I ordered a cheap white-LED flashlight along with a batch of other electronic parts. When I dropped in three AA cells and poked the switch, one of the four LEDs began flickering like a strobe light. For what it had cost, it wasn't worth returning. Yes, I felt like a fool even though the calendar didn't read April 1.

The "circuit," if you can call it that, inside the flashlight consisted of the battery, a switch, and four white LEDs in parallel. No current-limiting resistor, no voltage regulator, no nothing. Well, maybe that's the reason why my *good* LED flashlight cost me five times more than this thing!

Early in your first electronics class, you learn the exponential relation between a diode's current (I) and its terminal voltage (V):

$$I = I_0\left(e^{\frac{qV}{k_B T}} - 1\right)$$

I_0 is the diode's reverse leakage current, ranging from a few nanoamps to a few milliamps. The value of $k_B T/q$ is approximately 25 mV at room temperature. For example, with I_0 = 1 nA, the current is 54 nA at 100 mV, 3 µA at 200 mV, 485 mA at 500 mV, and 235 MA at 1 V. Yes, 235 mega-amps. Work it out!

Although the simple formula obviously doesn't reflect reality, it's true that small changes in the voltage applied to a diode cause huge changes in its current. Let's take a detailed look at some LEDs, shed some light on the innards of that flashlight, and see how to drive LEDs correctly.

CURRENT DRIVE

Diodes must be driven from a current source, not a voltage source, to maintain control over their current. You've certainly used a simple series resistor to limit the current available from a voltage source, which, although it isn't a great current source, is often good enough for LEDs. When you're characterizing diodes, though, you need a current source that permits simple, stable adjustments, while preventing overcurrent accidents (a.k.a. releasing the magic smoke).

Figure 1 shows a quick and easy adjustable current source. The op-amp regulates the transistor's base current so the voltage across R1 equals the voltage set by the trimpot. With R1 = 10 W, the voltage across it is numerically equal to 10 times its current: 100 mV for 10 mA.

The 39-kΩ resistor in series with the 1-kΩ trimpot limits the maximum set point to 300 mV and the maximum current to 30 mA, 50% over the usual 20-mA operating current for small LEDs. Because you'll directly measure the current while adjusting the trimpot, there's no need for high-precision resistors. You can use any op-amp and small NPN transistor.

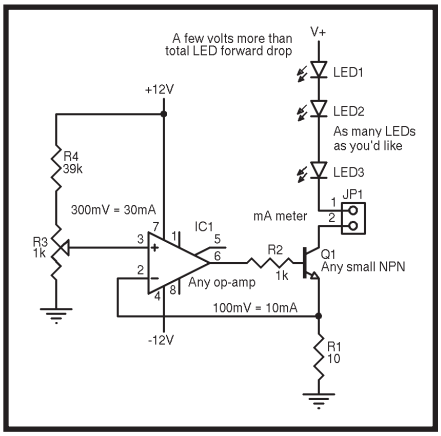


Figure 1—*This simple current sink drives LEDs with a constant current. The power supply must be a few volts higher than the total LED forward drop.*

Figure 2a shows the forward voltage across 10 supposedly identical white LEDs from my collection, measured at currents ranging from 1 to 30 mA. I drove all 10 LEDs in series from a 40-V power supply, recorded the forward voltage across each LED, and massaged the results in a spreadsheet. You can download the results, along with measurements and graphs for several more colorful LEDs, from the *Circuit Cellar* ftp site.

The LEDs' datasheet states that the forward voltage is typically 3.6 V and less than 4 V at the 20-mA maximum current. The maximum DC current allowed is 30 mA, although the peak current can be 100 mA for 10 ms at most, with a 10% duty cycle. The overall power dissipation must be less than 120 mW. All of the LEDs in my sample lie well within their specifications.

Even over this limited range of currents, you can easily see the exponential shape of the curves. Figure 2b plots each diode's forward voltage against the logarithm of the current, a task that formerly required semi-log graph paper, to show that all but one of the curves have the expected straight line shape.

LED D evidently has a problem because its forward voltage tops out around 3.65 V and actually drops slightly between 25 and 30 mA. This is particularly obvious in Figure 2b, with a distinct bend in the upper end of its curve. I measured this LED several times and came up with similar results, so it's not my technique!

TERMINAL CONDITIONS

The voltage-versus-current graph for a

particular LED tells you two things. As with my circuit, if you know the current, the graph tells you the resulting voltage. It also shows the current the LED draws at a given voltage.

Pop quiz: suppose a designer connected LEDs D and C in parallel with a 3.100-V voltage source. What would be the forward current in each LED? Extra credit: What about their brightness?

Peering at Figure 2a, you can see that the curve for LED D crosses the 3.100-V line at 1 mA and LED C's curve crosses at 5 mA. That makes a five-to-one difference in current, although the other LEDs in this (tiny) sample would be more closely matched. Because LED brightness is roughly proportional to current, LED C would be five times brighter than LED D. I found it impossible to take a credible photograph, but the difference is "eyeballometrically" obvious.

Not only does brightness vary with current, it also depends on the exact composition of the LED, which varies from batch to batch. Despite what you think you know about the precision of semiconductor manufacturing, there's a lot of art and magic ritual involved, so LEDs from different batches can vary dramatically.

If you're using multiple LEDs for backlighting or displays, you'll want consistent brightness. Tweaking the current of each LED isn't feasible, so you must buy brightness-matched LEDs.

Manufacturers test each LED in every production lot for brightness (at a specific current, typically 20 mA) and sort them into bins. LEDs within each bin differ by about a factor of two, and the range across all bins spans several orders of magnitude. Every functional LED goes into one of the bins, which makes every LED salable: the brightest LEDs command the highest prices and the dimmest ones go to surplus dealers.

For critical applications, such as backlights, a second sorting within each brightness bin groups the LEDs by color. You can specify extremely accurately matched LEDs if you pay a premium price for the testing and sorting.

FAILURE ANALYSIS

Shortly after I got the blinking LED flashlight, I took it apart and replaced the offending LED. It wasn't designed for easy repair, so I used Gorilla Glue to secure the battered LED assembly in place. When I reassembled the flashlight, a second LED began blinking merrily away. Now, one failure could be a bad part. Two failures represent a trend!

I took the flashlight apart again, albeit with considerably more difficulty, removed the LED assembly, and unsoldered the LEDs. No matter what I discovered, I would replace all of the LEDs this time!

One of the non-blinking LEDs failed at 30 mA before I began recording its voltage. Although that's above the usual 20-mA continuous rating, a device intended for use directly across three AA cells shouldn't fail at such a low current. That makes *three* failures of the four original LEDs.

The blinking LED worked fine for currents under approximately 18 mA. Above that level, it exhibited brief, intermittent open-circuit failures. Although I couldn't see anything obviously wrong with the internal wire-bond attachments, I don't have failure analysis facilities.

Figure 2c shows the voltage-versus-current curves for the two remaining flashlight LEDs, the first white LED I installed, and a red LED as a baseline. Notice that the forward voltages for the two flashlight LEDs are much, much higher than that of the white LED from my stash.

The flashlight uses three AA cells that provide approximately 1.6 V each when they're new for a maximum of 4.8 V. The non-blinking LED passes 15 mA at that voltage; the blinking LED passes less than 10 mA. Why it blinks at that current, I cannot say, other than that AA cells certainly differ from my test rig.

Notice that the graph also shows that current through my white LED is off-scale high. Ooops! Well, if you've never

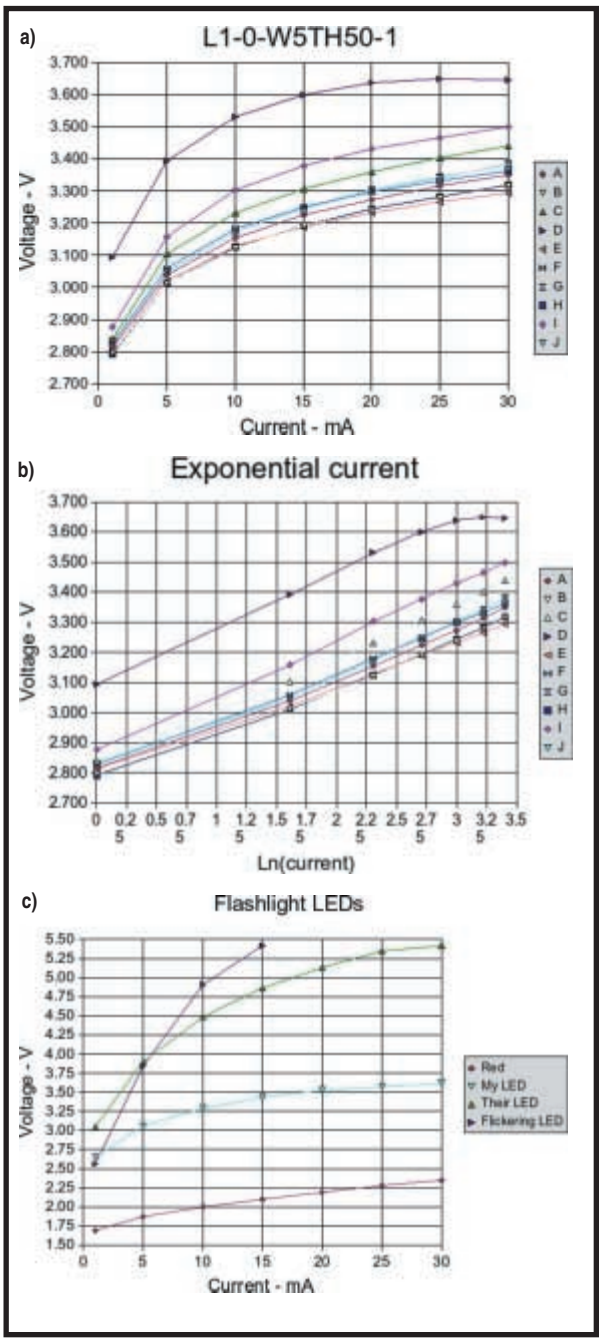


Figure 2a—*Ten nominally identical white LEDs show a wide variation in forward voltage for a given current. The voltage for LED D, the top trace, actually drops slightly between 25 and 30 mA. What would happen if you were to apply a 3.75-V voltage source to its terminals?* **b**—*Plotting the forward voltage against the log of the current shows that the relation isn't exactly exponential, but it's extremely close, except for LED D, which obviously has some problems.* **c**—*The two upper traces are LEDs from a cheap flashlight. The middle trace is a good-quality white LED. The lower trace is an old red LED for comparison. The top trace stops at 15 mA, just before the LED began flickering.*

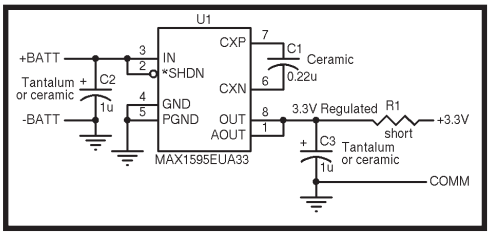


Figure 3—*A MAX1595 regulated charge pump produces a nominal 3.3 V from two AA cells to drive about 70 mA into four white LEDs. In theory, anyway!*

made a mistake like that, you're a better engineer than me. The diode's internal bulk resistance limits its actual maximum current, and the bond wires didn't vaporize in this case.

I suspect the LEDs in that flashlight came from a production-line sort for high forward voltage. This was a cheap flashlight and I doubt the manufacturer bought a custom run of LEDs. The fact that three of the LEDs failed under what seem like normal conditions makes it reasonable to assume that these are, in fact, production line culls.

DOING IT RIGHT

Because LED brightness is proportional to current and current is exponentially related to voltage, LED flashlights become dim extremely quickly as their batteries discharge. The nominal fully discharged voltage for alkaline cells is approximately 0.8 V, so the voltage in a three-cell flashlight varies from approximately 4.8 V down to 2.4 V. Figure 2c shows that the corresponding currents range from 15 mA down to just about zero, so you'd probably replace the cells at 4 V and 6 mA, thus wasting much of their capacity.

More expensive flashlights use a power supply to boost and regulate the battery voltage. I decided to use a Maxim MAX1595EUA33, which charge-pumps a nominal 3-V battery to a regulated 3.3-V output. The schematic in Figure 3 shows that the entire power supply consists of only four parts. As you'll see, R1 isn't required.

I replaced one of the three AA cells in my flashlight with a machined aluminum gizmo holding the charge-pump circuit and four white LEDs, as shown in Photo 1a. Photo 1b shows a rear view of the

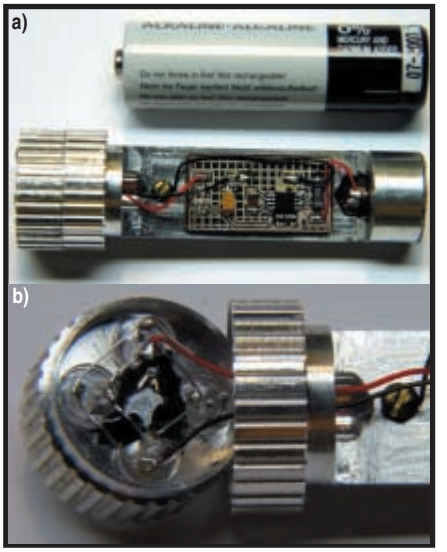


Photo 1a—*The charge pump circuit replaces one of the three AA cells in the flashlight. I machined the aluminum cylinder from a slightly larger toothed-belt drive pulley.* **b**—*The four matched LEDs align on their flanges in closely fitted holes. The black epoxy blob simply holds them in place.*

LEDs aligned in snug-fitting holes, which work much better than the original, rather flimsy plastic.

The four LEDs contain the same chip as those profiled in Figure 2a, but in a dif-

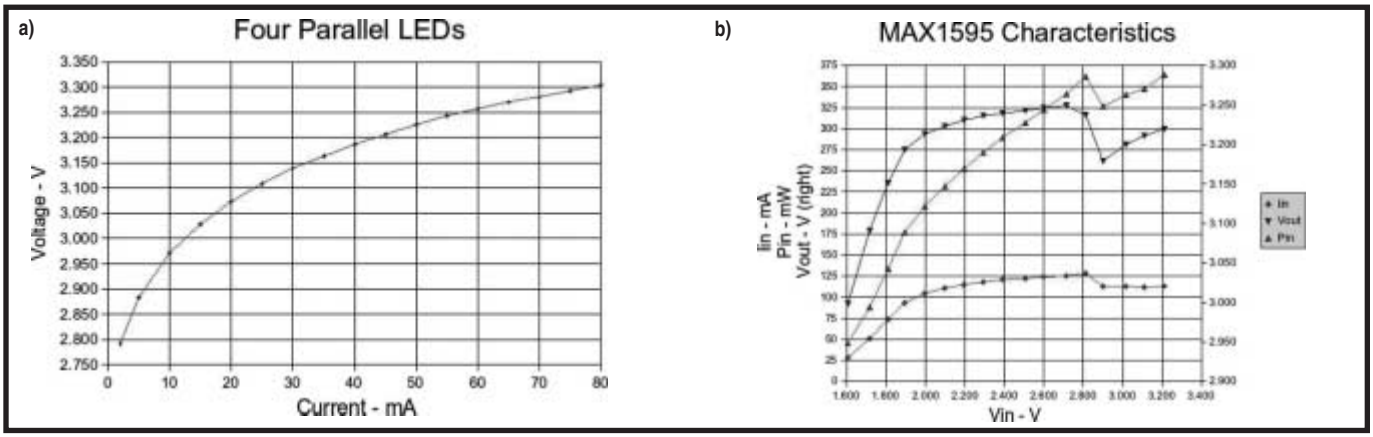


Figure 4a—*The voltage-versus-current characteristic of four matched white LEDs will work nicely with a 3.3-V source. Isn't that a pretty curve?* **b**—*The MAX1595EUA33 output remains between 3.2 and 3.25 V through most of the battery's usable life. The LED current will vary between 40 and 60 mA, a visually indistinguishable difference.*

ferent case that produces a 20° beam width. I selected four LEDs that differed by only 4 mA at 3.250 V. (The complete data is in the spreadsheet posted on the *Circuit Cellar* ftp site.) Figure 4a shows their combined current as a function of voltage. They draw almost exactly 80 mA, an average of 20 mA each, at 3.300 V, which is a perfect match for the MAX1595's nominal output. Incidentally, that tidy exponential curve is real data. I'm not making it up!

I measured the supply's input current and output voltage as a function of its input voltage to get an idea of how it would perform as the battery discharged. Figure 4b shows that the output voltage remains between 3.25 and 3.20 V as the battery voltage drops from 3.2 to 1.9 V. The output hits 3 V just before the MAX1595 automatically shuts down.

Comparing the output voltage to the LED characteristics, you can see that the current varies between 60 and 40 mA, dropping to 10 mA just before MAX1595 shuts off at 1.6 V. The LEDs will run with reasonably consistent brightness, decreasing by a third, over much of the battery's life. That's fine because a single LED brightness bin covers a factor of two difference at the factory.

I don't know what causes the kink in the MAX1595's output for input voltages near 2.9 V. The current is fairly close to the chip's rating, so the LEDs may pose a difficult load: a low-voltage drop in the regulator with a relatively high and rapidly increasing current.

The MAX1595 datasheet calls for three ceramic capacitors, which, because of their lower ESR, have better efficiency than electrolytic capacitors. My parts stash doesn't include 1-µF ceramics. I used a tantalum electrolytic capacitor on the input and an aluminum capacitor on the output with a 220-nF ceramic pump capacitor. That's the most critical capacitor, and, because it sees both forward and reverse voltages, it mustn't be polarized.

At 2.6 V in and 3.25 V out, the supply draws 323 mW from the battery. At that voltage, Figure 4a shows that the four LEDs dissipate approximately 180 mW for an overall efficiency of 55%. That's the trade-off for a nearly constant light output over the life of the batteries.

For comparison, a 3-V flashlight bulb draws 530 mA and dissipates 1.6 W. My favorite little incandescent penlight may have a tighter, brighter beam, but it uses five times more power than this LED flashlight.

Because the regulator's voltage output is closely matched to the LED characteristics, I omitted a ballast resistor and its power loss. The current variation over the supply's voltage range is well within the LED's specifications. Pragmatically speaking, the MAX1595EUA33 can't produce much more than 80 mA with a 3-V input. The LEDs are safe from destruction, even by a voltage-source power supply.

If your LEDs require more than 3.3 V, the MAX1595EUA50 produces 5 V. The circuit board layout includes space for R1, which is the ballast resistor to convert that voltage source into an LED driver.

CONTACT RELEASE

After all that effort, I'm pleased to report that the rebuilt LED flashlight works much better, with a better beam and constant bright light. If you're illuminating a gizmo with white LEDs, check out the latest generation of LED driver ICs (the MAX1595 is just one example) for better ways to get the job done. There's no need to be foolish anymore!

I just got a surplus batch of four-LED clusters intended for two-cell flashlights. As expected, one has a dim LED, and they all have poor construction. That supports my theory that rejected LEDs wind up in cheap flashlights.

Ed Nisley is an E.E, P.E., and author in Poughkeepsie, N.Y. You may contact him at ed.nisley@ieee.org. Write "Circuit Cellar" in the subject line to clear the spam filters.

PROJECT FILES

To download the code and graphs, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/177.

RESOURCES

Eagle CAD circuit design, www.cadsoftusa.com.

General LED information, http://ledmuseum.home.att.net/leduv.htm.

Maxim Integrated Products, "MAX1595: Regulated 3.3V/5.0V Step-Up/Step-Down Charge Pump," 19-2107, rev. 1, 2002, www.maxim-ic.com.

SOURCES

LEDs

All Electronics
www.allelectronics.com

Electronic Goldmine
www.goldmine-elec.com/default.htm

LED Supply
www.ledsupply.com

AVR 2004 DESIGN CONTEST
Contest Winner

Digital RC Servo Controller (Part 1)

32-Channel Design

The days of linking serial servo controllers to support a high number of channels are over. In the first part of this series, Eric explores RC serial servo controller theory. Plan on using a 32-channel controller in your next robotics project.

Several months ago, I saw an advertisement for a 25-channel radio-controlled (RC) serial servo controller. A serial servo controller's function is to control multiple RC servos while providing a simple serial command interface. Armed with a renewed interest in the subject, I decided to research the market further. It turns out that a number of commercial products allow for serial-based control of RC servos, most of which only support up to eight channels at 8-bit resolution.

RC servos have enjoyed a long period of popularity among RC enthusiasts. The recent explosion in homebrew robotics projects has created a new home for the versatile servos. Robotic insects in particular require a large number of servos to generate walking motion. As well, some animatronics creations demand numerous servos to bring them to life. In the past you were limited to purchasing several serial servo controllers and chaining them together to support the high number of channels. Things are different now.

In this article I'll describe a true hardware-based, 32-channel digital serial servo controller that's well suited for these applications. Unlike all the other commercial offerings, this design uses true dedicated parallel hardware resources for PWM pulse train generation at 16-bit accuracy and 12-bit resolution with all 32 channels fully synchronized!

SERVO BASICS

Photo 1 shows a typical RC servo. Different horn attachments for the servo shaft allow it to generate rotary and push-pull motions. Whereas a DC motor rotates continuously with the application of a voltage, an RC servo uses an internal electronic angular position control loop, which moves the servo to a commanded position and holds it in that position until the power is removed or the command changes.

RC servos also contain internal gearing, which can provide large output torques. Different servo sizes and shapes are available, as well as plastic and metal gearing options. Three wires are used to interface to an RC servo. The first two are the servo power and the ground wires. The third is the angular position control signal input.

Powering RC servos is often done with rechargeable battery packs. Servos can usually accept an input voltage between 4.8 and 6 VDC. Depending on the mechanical load applied to the output shaft and the size of the servo, its current draw can climb into the 1 A range. The no-load operating current for a standard servo is in the 100 to 200 mA range. When powering several servos from the same supply, you must ensure that the current rating on the power supply is adequate.

Photo 1—*A typical RC hobby servo has removable horn attachments to generate rotary or push-pull motions. Three wires are used to interface to a servo driver. Different servo manufacturers may use different wire color codes and pinouts.*

The command signal input to an RC servo is a PWM digital pulse train (see Figure 1). The amplitude of the PWM pulse is nominally 3 to 5 V. The pulse train has a period between 20 ms (50 Hz) and 16 ms (60 Hz). The period's actual value isn't critical.

The width of the positive pulse, which can vary between 1 and 2 ms, determines the absolute angular position of the servo. A pulse width of 1.5 ms corresponds to the center position of the servo travel. A typical servo will swing through a 90° range corresponding to a 1- to 2-ms pulse width variation. This range, however, can also vary from servo to servo, so check the specifications.

Most servo vendors provide a safety margin that allows their servos to rotate beyond the 90° when sending commands outside the 1- to 2-ms boundary. As a result, a lot of commercial servo control boards support an extended mode, where the pulse varies between 0.5 and 2.5 ms. Be careful, though, because crashing the servo in its hard stops can damage its gears. This is why some board vendors also allow you to set soft limits on each servo channel so that minimum and maximum positions can be set and stored in nonvolatile memory.

The angular resolution of a typical servo is of roughly 0.7° out of a 90° range, or 7 bits, because of the built-in dead-band of approximately 8 µs that's

needed to reduce mechanical oscillations (servo chatter). This means that 8 µs is the smallest practical positional increment in a pulse width command. On the other hand, the absolute servo positioning accuracy is affected by the mechanical backlash and material properties. The newer digital servos claim a resolution of up to 10 bits, or 0.09° (1 to 3 µs deadband), at the expense of higher power consumption.

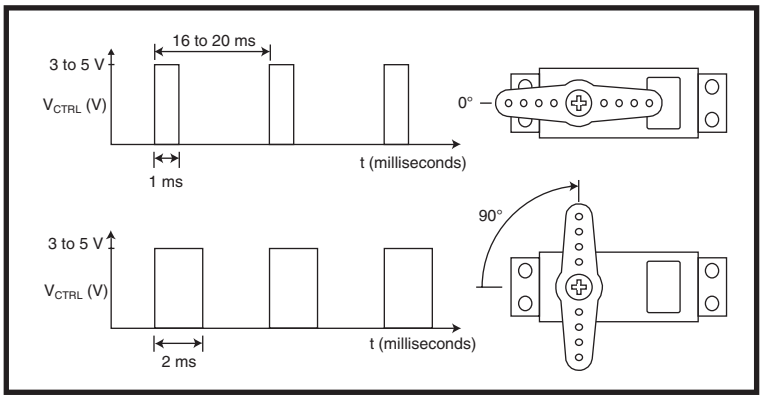


Figure 1—*An RC servo position control signal consists of a pulse-width modulated digital pulse train. The 1.5-ms pulse width moves the servo to its middle range, or center position. Varying the pulse width from 1 to 2 ms will rotate the servo from one extreme to the next.*

The trick for a lot of beginners is to figure out how to actually generate the PWM signal to control one or more servos simultaneously. A microcontroller is usually part of the solution. Most new MCUs have a dedicated hardware PWM capability, but they're commonly limited to one to four channels. Also, because the useful pulse control range of the PWM waveform in the specific RC servo application is only 1 ms out of 20 ms (i.e., 5% of the full scale PWM range), you get limited effective output resolution. For instance, a dedicated PWM module in an MCU using an 8-bit counter would only provide an effective PWM output resolution of 12 steps (256 × 5%), or 4 bits, instead of 8 bits. The generic PWM module is designed to cover a full range of 20 ms, with 8 bits as opposed to covering the servo's 1 ms useful range with 8 bits. Likewise, a generic 16-bit PWM module would generate an effective 12 bits of servo resolution.

A solution to this problem is to perform the PWM waveform entirely in firmware through bit banging. Unfortunately, this can use up a lot of the MCU's resources. It requires critical firmware timing. A simpler approach is to purchase a purpose-built servo controller board and let it handle the critical multichannel timing while sending low overhead position commands on a standard serial port (UART).

COMMERCIAL CONTROLLERS

Scott Edwards Electronics's Mini SSC II is the best-established serial servo controller on the market. It allows you to control up to eight RC servo channels simultaneously. The Mini SSC II uses an efficient unidirectional binary serial protocol for controlling the servo channels (see Figure 2). This protocol has quickly become the de facto standard.[1]

A large number of serial servo controller boards on the market now mimic this protocol; nevertheless, others use their own proprietary versions. Current board vendors include Pontech, New Micros, Pololu, Lynxmotion, Picobytes, and Parallax. It seems as though a new one appears every month!

On the software side, a few vendors have developed generic servo control software that supports the Mini SSC II protocol (as well as others). These include Brookshire Software with its Visual Servo Automation, Reynolds Electronics's Robo-Ware, Mister Computer's Mini SSC Panel, and Roscoe Robotics with various offerings.

PRODUCT LIMITATIONS

With few notable exceptions, most of the current low-cost RC servo controller boards on the market exhibit some type of functional trade-off or limitation. Of course, depending on the end application, these trade-offs may or may not result in appreciable performance differences.

Serial RC servo controllers have several desirable properties like a low-jitter PWM waveform generated preferably in hardware and greater than 10-bit resolution for the new high-accuracy digital servos. They also have a large number of parallel channels to accommodate demanding applications. The controllers have fully synchronized servo channels to ensure accurate servo trajectory tracking, and they include independent programmable soft limits to limit the range of travel of servos. The speed control on each channel is to reduce servo twitching when commanding large motions. There are programmable servo start-up positions too.

Also note that you have the ability (optional) to turn off servos (i.e., go limp) and allow them to be moved manually. The serial servo controller's UART data rates can be changed to suit your application. Finally, you can transparently daisy chain several controllers for system expansion.

Currently, market offerings appear to be partitioned into three main categories. The first represents the boards, which control only a limited number of channels (1 through 4). These boards typically use an MCU with built-in hardware PWM generation modules. Doing the PWM in hardware reduces pulse jitter. The waveforms usually can be synchronized. However, the effective position resolution may not be the full 8 bits, so carefully examine the specifications. The small number of PWM channels supported makes this approach the low-cost low end of the spectrum.

A second category of servo controllers offers up to eight channels (sometimes 16). These boards generally use an MCU with firmware-based bit-banging generation of the PWM on all channels. The bit-banged algorithm is based either on timed interrupts or simply on the open-loop firmware delays (or a combination of both). Probably the simplest interrupt-based implementation of a multichannel

Byte 1	Byte 2	Byte 3
0xFF (Sync value)	0x00–0xFE (Servo #)	0x00–FE (Position)

Figure 2—*The Mini SSC II serial servo controller protocol has become the de facto standard supported by many vendors. Note that 3 binary bytes of data are sent to command a servo position. The 0xFF value is reserved as the sync marker. Servo addresses and position commands are limited to the 0 to 254 range as a result.*

PWM would be to generate an interrupt at the finest granularity (resolution) of the output PWM signal, say, 8 µs for a standard servo. Therefore, at each clock tick, a system counter is incremented. A table of the pulse widths for each channel is then scanned and the corresponding I/O port pins are toggled low when the main counter has exceeded the respective counts.

After the 20-ms window is reached, all the PWM lines are set high again and the cycle repeats. This doesn't always work too well because the resulting interrupt rate is higher than 125 kHz. Because of the high overhead, most 8-bit MCUs don't have enough horsepower to do the processing required between the interrupt clock ticks!

A slightly better interrupt-based approach is to use the MCU's built-in timer. At the beginning of a PWM pulse, the timer is reloaded with the length of the pulse. It's automatically decremented by the internal hardware clock. When it reaches zero, an interrupt is generated. At that point the MCU lowers the port pin, reloads the balance of the 20-ms delay to raise the pin high again, and starts the cycle over. This works fine for one channel, but when a large number of multiple channels are involved, the MCU must manage and share the timer with all 8/16 parallel channels. It means establishing a global schedule of all time deltas between the falling edge of one channel and that of the next channel to be toggled off.

The problem here is that if two channels must be toggled off at the same time (or one time step after each other), the interrupt will be triggered too quickly for the MCU to have time to respond and reload the counter. This results in timing errors. In fact, you can verify this by commanding two or more channels to have the same pulse width (or even a few counts apart). By looking at the PWM signals on an oscilloscope, you may see timing errors.

To attempt to solve this problem, a serial servo controller will stagger the pulses of the 8/16 channels in time so that the pulse for the first channel is generated. This is followed by the pulse for the next channel and so on (see Figure 3). At any point in time, the MCU resources are only focused on generating the timing for a single pulse, which greatly simplifies the firmware. The limitation here is that in 16 to 20 ms of overall period, you can only squeeze in between 16 and 20 concurrent channels before filling up the possible time slots. Now the servos aren't synchronized. In other words, there's almost a 16- to 20-ms delay between updating the pulse width of the first channel and that of the last channel. This can result in a 5° to 6° degree dynamic tracking mismatch for a standard 0.2 s/60° servo.

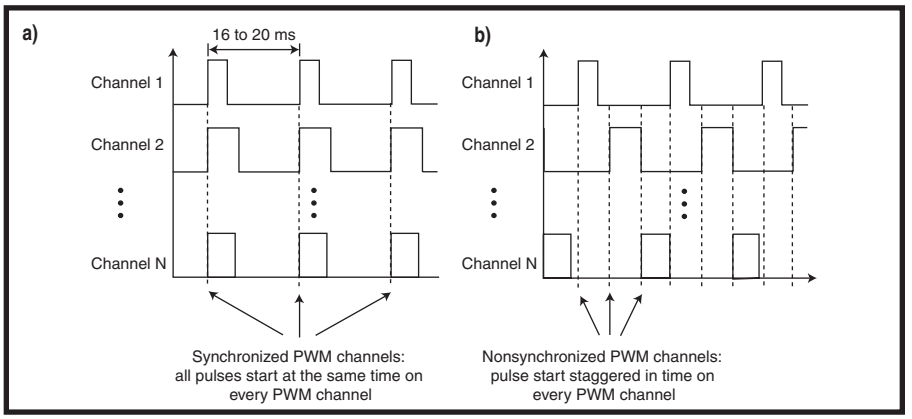


Figure 3—*Take a look at the differences between synchronized multichannel PWM generation* ***(a)*** *and nonsynchronized (staggered) PWM generation* ***(b)****. The illustrations aren't drawn to scale. All of the PWM channel pulses in* ***a*** *start at the same point. This ensures that all servos move synchronously. The scenario in* ***b*** *results in relative servo trajectory tracking delays.*

In addition to all of this, the MCU's internal UART, which must process incoming serial commands, can also generate additional interrupts. This adds random jitter to the PWM waveforms.

There are numerous other approaches to generating PWM pulses, such as fixed code-timing loops and precomputed lookup tables. Most exhibit additional limitations. For example, by turning off system interrupts, the UART, which must also process serial commands, has to be polled during the dead time between the periodic pulses. This usually restricts the maximum data rate.

Finally, the third category of RC controllers on the market positioned at the higher end of the spectrum includes those with more than 16 channels using hardware-based PWM generation. One notable example is the New Micros 25-channel ServoPod, which uses a DSP with internal dedicated hardware PWM generation. I'll introduce you to a novel FPGA-based approach belonging to the third category. It's a true 32-channel hardware-based PWM generation unit under the tight control of an MCUthat's responsible for all higher-level functions.

SYSTEM ARCHITECTURE

One of the best ways to generate jitter-free, accurate, synchronized, multichannel PWM signals is through a hardware implementation. Figure 4 depicts the overall circuit.

A Xilinx FPGA contains an internal array of 32 independent PWM generation units each tailored to generate a 0.5- to 2.5-ms pulse at 12-bit resolution (0.5 µs) and 16-bit timing accuracy. Each of these channels is dedicated to the control of one servo. A

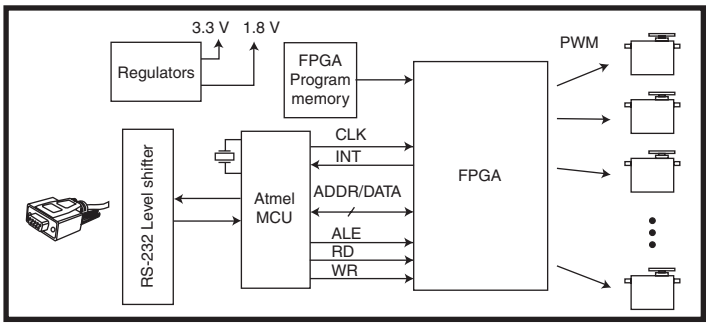


Figure 4—*A Xilinx FPGA contains the 32 parallel hardware PWM generator modules. The modules are controlled through memory-mapped registers. The heart of the circuit is the ATmega8515L MCU, which makes full use of its external memory bus. A level shifter connects the board to a standard RS-232 PC serial port.*

small Xilinx flash program memory is used to load the FPGA code every time the board is powered.

An ATmega8515L MCU is the heart of the system. Making full use of its external memory bus interface, it controls the set up and operation of the 32 memory-mapped PWM generation units and handles other system activities.

The ATmega8515L is responsible for running the serial protocol and updating the PWM registers. An external interrupt pin times the loading of the registers to occur in the dead time of the PWM cycle so that runt pulses aren't generated. The ATmega8515L also runs the servo speed control algorithms and performs real-time limit checking on each servo axis. The speed and soft travel limits on each channel are user-defined and stored in the nonvolatile EEPROM memory inside the ATmega8515L. An RS-232 level shifter allows for access to the ATmega8515L via a standard serial port.

STAY TUNED

So far I've explored the theory behind RC serial servo controllers, reviewed some commercial offerings, and introduced the architecture of a true hardware-based 32-channel controller. Next month, I will present the circuit schematics and delve deeper into FPGA territory to expose the inner workings of the circuit. Until then, start sharpening your FPGA skills.

Eric Gagnon, M.A.Sc., P.E., has been hooked on electronics since the age of 12. He earned both of his degrees in electrical engineering from the University of Ottawa. Eric has more than 10 years of embedded design experience. He has worked on projects related to the International Space Station, industrial robotics, 3-D machine vision, and embedded video. Eric currently runs Digital Creation Labs, Inc. (www.digitalcreationlabs.com). You can contact him at egagnon@digitalcreationlabs.com.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2004/177.

REFERENCE

[1] Scott Edwards Electronics, "Mini SSC II Serial Servo Controller," SSC-ASD2, www.seetron.com/pdf/ssc2_mnl.pdf.

SOURCES

ATmega8515L Microcontroller
Atmel Corp.
www.atmel.com

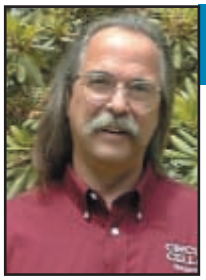
25-Channel ServoPod
New Micros
www.newmicros.com

SV203 Servo controller
Pontech
www.pontech.com

Servo control software
Roscoe Robotics
http://members.aol.com/iamflb/

Mini SSC II servo controller
Scott Edward Electronics, Inc.
www.seetron.com

FPGAs and development tools
Xilinx, Inc.
www.xilinx.com



by Jeff Bachiochi

Stay in Touch

Sensor Material for Robotics Applications

Peratech's quantum tunneling composite (QTC) technology is going to change the way you approach your robotics projects. Read on to learn how Jeff softened the grip of a Heathkit Hero robot with a QTC sensor.

Touch your thumb and index finger together. I'll wait. Feel that? Now try the same thing using different pressures. Feel any difference? No? Me neither. Except for the muscle strain I felt, I couldn't gauge the amount of pressure. Nevertheless, I can pick up an egg with out crushing it. What allows me to do this?

Understanding of the world would be difficult without your sense of touch. Although your entire body is sensitive to touch, your glabrous areas (e.g., fingers and toe pads) are more sensitive because of the types of skin receptors found there (see Figure 1). Your epidermis, the top layer of skin, contains Merkel cells that respond to pressure on the skin and Meissner corpuscles that respond to vibrations (20 to 40 Hz). Below the epidermis is the dermis, which houses Ruffini endings that respond to pressure and Pacinian corpuscles that respond to vibrations (150 to 300 Hz). Nerve fibers behave in two distinct ways. The Ruffini endings are slow-adapting receptors that fire based on an on-going stimulus. The Pacinian corpuscles are rapidly adapting receptors that fire with changing activity. Think of it like the difference between speed and acceleration (see Figure 2).

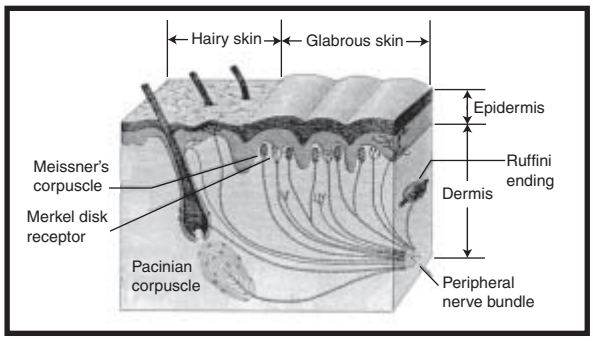


Figure 1—*The sense of touch comes from two types of nerve cells. Merkel cells and Ruffini endings respond to pressure. Meissner and Pacinian corpuscles respond to vibration. Both types are found beneath the glabrous skin of the body.*

Crime scene investigators use fingerprints for identification purposes. But those little valleys and ridges are more than identifying marks; they play an important role in your sense of touch. Take a sharp object like a pin (please play carefully here) and lightly draw it across your finger pad. Can you feel each time the pin rubs over one of those tiny ridges? It's just like scratching a phonograph needle over a record's grooves. (No this won't work on CDs, although they will skip if the surface is scratched, but that's another story.) Not only do fingerprints help detect movement, they also aid in judging speed.

Wouldn't it be nice if your favorite robot had a similar sensitivity? In this column, I'll introduce you to Peratech's quantum tunneling composite (QTC) technology. I used it in a Heathkit Hero robot project.

ROBOTICS

Intelligence alone does not a robot make. Robotics projects depend on the synergy of I/O functions and intelligence.

You can watch robots slug it out in an RC arena with flippers and blades. But the robots aren't autonomous; they require an operator. They have no input feedback between the processor and an output function. The only input feedback is what the operator sees. Locomotion accuracy is based solely on the operator's ability to adjust the robot's movement in real time. Weapons are also operated in this fashion. Many times, however, winning a battle depends more on equipment functionality than on a particular battlefield strategy.

Robots are also used on assembly lines. Do the sparks generated by automatic line welders fill you with a feeling of respect for industrial power, or do they worry you about the future of the human labor force? The point here is that even those robots have little or no feedback. Many will continue performing their programmed tasks

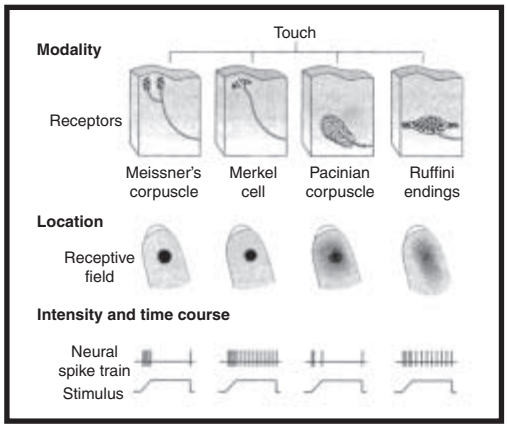


Figure 2—*Take a look at how each type of nerve cell responds to stimuli. The pressure cells respond only to changes in pressure. The vibration cells have a continuous response.*

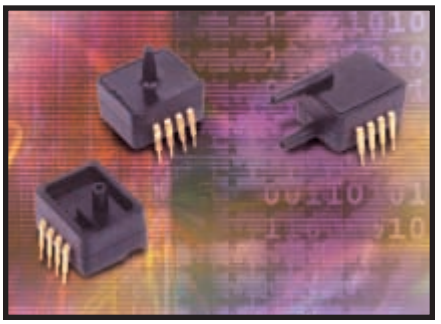


Photo 1—*Solid-state pressure sensors contain sensing elements that consist of four piezoresistors buried in the face of a thin, chemically etched silicon diaphragm. A pressure change causes the diaphragm to flex, thus inducing a stress or strain in the diaphragm and the buried resistors. The resistor values change in proportion to the applied stress.*

even when the product is removed or a person gets in the way. Dumb, but efficient. This may be fine if humans are removed from the picture, but the future of robotics is changing. As industrial technology evolves, robots will be tasked to work more closely with laborers. This will create a new set of safety issues that will be tackled only by way of feedback. Although feedback can be used to prevent a robot from doing harm, it's also a requirement for accomplishing tasks effectively.

Many robots are currently used as autonomous delivery vehicles. When the technology was first introduced, the robots simply followed lines, but it quickly became clear that external issues were preventing tasks from reaching completion. The first line-following robots couldn't avoid the obstacles in their paths. As a result, input feedback has become the eyes and ears robots use to make autonomous decisions.

ROBOTIC TOUCH

You touch things to explore your environment. Touching things also helps you form emotional bonds with other people.

Photo 3—*Tekscan uses a thin semiconductive coating (ink) to produce Flexiforce, which is a flexible tactile force sensor.*

The bit is the ultimate input. It's decisively simple. Yes/no. On/off. Open/close. It either is or it isn't. Most microcontrollers give the bit its own instructions. Robots use switches to acquire binary information. Although other types of input can give the robot a sense of where it stands in relation to other objects, the robot's bumpers indicate when it comes into contact with something. The problem, of course, is that objects can hit the robot without striking a bumper. Someday robots will be covered with a sensory material like human skin.

After a robot has the ability to move around freely, it can provide delivery assistance. Having the ability to load and unload the cargo would make the robot truly autonomous. This might be the mail, medicine, or your favorite beverage. The best appendage would be some kind of arm that could grasp objects. Sounds easy, right?

Assuming you've perfected the robot's ability to move around and position its arm, you've got to address the issue of touch. How much force is needed to pick up an object? You can design an end effector that's matched to the necessary function. For instance, an arm with a scoop wouldn't necessarily require information about the items it would scoop. However, an arm with a grasping effector would require feedback to complete the task. Again, simple and specific tasks can be performed using simple feedback switches. The switch contacts should be positioned within the grasping effector. OK, so now your robot can grab a rock, but what about an egg?

WITHSTANDING PRESSURE

ICs come in either antistatic plastic tubes or black foam. The latter contains carbon particles in the foam. This allows IC pins to be shorted together so a floating pin won't pick up a potentially damaging external charge. This material can be used as pressure sensitive material because it has a measurable resistance. Unfortunately, there are some serious drawbacks to using this material. It's a high-resistance material that lowers its resistance with pressure. Because it doesn't return to its original uncompressed state after the compression is released, it's difficult to get repeated accuracy. Another problem is that the foam can crumble with use.

Photo 2—*The load cell uses strain gages usually in a full bridge format to measure the load applied directly to a point on the sensor.*

The idea of using some material to directly measure pressure is a useful one. Popular solid-state pressure sensors can measure air and liquid pressures (see Photo 1). Load cells measure weight and force through tension and compression (see Photo 2). Resistive ink force sensors come the closest to the use of the black conductive foam (see Photo 3). One of the newest entries in this market is Peratech's QTC material, which has the relaxed qualities of a switch (off) with the compressed qualities of a conductive material (see Photo 4).

QTC MATERIAL

QTC material is made from metal particles (as opposed to carbon) combined with an elastomeric binder. Unstressed, the material looks like an

Photo 4—*QTC components from Peratech come in four basic styles: Pills, Switch Substrates, and Cable and Force Sensors. Pills can be added to current switch designs to change any push-to-make switch into a fully proportional voltage or current controller. Switch substrates can be incorporated into membrane or keypad switches to provide far greater reliability and optionally proportional control. Cable can be used in machine protection or security applications, or it can detect the presence or removal of loads. Force sensors, which offer considerably more range than standard force sensing resistors, are much more resilient to handling and environmental factors.*

Composite material used	Unstressed	Fully stressed	Change in force for a 10× change in resistance	Current carrying capacity
Carbon	$10^5\ \Omega$	$10^2\ \Omega$	Medium	Low
Metal	$10^{12}\ \Omega$	$< 1\ \Omega$	Low	High

Table 1—*Conductive foam has been manufactured using carbon for years now. Peratech's metallic approach has significant benefits. This chart shows the resistivity and current carrying differences between the two methods.*

insulator. Applying a mechanical deformation initiates conduction. An almost metallic conduction is achieved with sufficient pressure. Table 1 shows how using carbon or metallic particles in the composite changes the material's properties.

The compression of carbon-based composites enables carbon particles to come in contact with one another resulting in conductive pathways. The metal particles in QTCs don't touch; they're just extremely close. In fact, they're so close that quantum tunneling is possible between particles. Quantum tunneling is the effect by which a free electron within a conductor passes through an insulator even though it doesn't support free electrons. Based on a constant electron energy level, compressing the QTC material essentially reduces the insulation barrier. This allows some electrons to tunnel through. Conversely, electron tunneling can occur (for any given unstressed QTC) if you raise the applied voltage (electron energy).

Stress on the QTC material produces a resistance change that's an inverse exponential function of the applied force. Actually, the way in which the QTC material is compressed affects its behavioral curve because external compression in one direction distorts the material in multiple directions. Peratech can adjust the behavior of QTC during the manufacturing process by varying the composite mix and physical dimensions. It developed four basic parts that cover a wide number of uses.

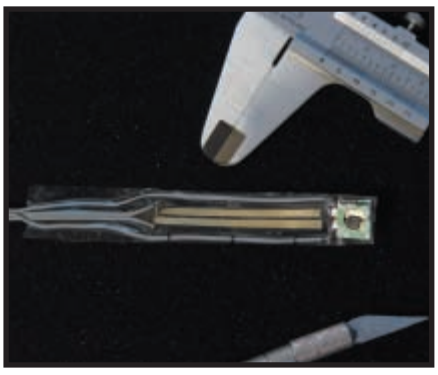


Photo 5—*You can see the conductors inside the clear packing tape sandwich. Holes punched in the upper tape allow a QTC Pill to make contact with the conductors. Double-sided foam tape on the fingertip shows how a QTC Pill is held over the conductors.*

QTC SWITCH SUBSTRATE

For touch pads and keypads, the QTC Switch Substrate is a thin application of QTC on the aluminum side of a polyester sheet. When placed over inter-digit electrodes (PCB traces), the material provides a conductive path between digits when compressed on the traces.

The QTC Switch Substrate material comes in three sensitivities. Unlike membrane switches, an insulation spacer isn't necessary. This simplifies the manufacturing process. You also have the benefit of each set of contacts becoming variable pressure-sensitive inputs.

QTC PILL

You can use the QTC Pill for low-power applications that require the direct control of devices like motors or lights. The QTC Pill is a 1 mm thick piece of QTC material (3.6 mm square). It can be used as a switch across PCB traces (like the Switch Substrate sheet), or it can go right in series with any low-power device, thus replacing the power switch contacts and adding a variable speed feature to the device. For high-powered devices, the QTC Pill can operate as the gate control to the power circuitry.

QTC CABLE

If your robotics project requires machinery safety guards, bumper detection, or impact counting, the QTC Cable is the solution you're looking for. QTC Cable is manufactured like RG-59 coax cable. The core insulator located between the inner conductor and the outer shield replaced with QTC material.

Note that the QTC Cable can be terminated with a standard BNC plug. Its minimum bend radius (10 cm) prevents erroneous switching.

QTC FORCE SENSORS

A QTC Force Sensor has a large laminated surface area capable of sensing large impacts. It can be manufactured in various sizes and shapes to over 1 m.

Similar to the Switch Substrate, the Force Sensor features a connector tail for electrical attachment via conductive epoxy, solder, or riveting. The choice of lamination provides some level of force/impact distribution. When mounted to a hard flat surface, you can alter the sensor's sensitivity by adjusting the size of a pressure spreader attached to the top surface.

MY HERO

What comes to your mind when you see the word "robot"? Many people think about R2-D2 and C-3P0. (Kudos to George Lucas for making these two robots popular trivia subjects.) Robotics enthusiasts usually think of the Heathkit Hero, which was one of the most successful robots ever produced. You won't find robots on Heathkit's web site (www.heathkit.com); however, you can follow the support link to sites that are continuing the proud tradition.

Although the original Hero's arm

Photo 6—*My Hero robot can successfully pick up an egg without cracking the shell. To avoid the safety issues of coexisting with humans, the robots of the future will need an extensive array of sensors.*

could reach out, it couldn't touch things because its gripper lacked feedback. Peratech's PTC material can solve this problem, and today's flexible circuitry is ideal for adding this type of sensor to the Hero's gripper.

I built a prototype sensor. First, I scouted out some flex circuitry I could hack. There are some nice flex items used in disk drives. I experimented with a short section of multi-conductor 0.5-mm flex cable by placing a QTC Pill on the uninsulated contacts of one end. I then measured the opposite end's contact with an ohmmeter. This worked well because the QTC Pill bridged many contacts and added redundancy against trace failure. It also comes with a means of terminating the connections using a standard flex cable connector. This proved the theory to be a sound one; however, it didn't fit the Hero's gripper in its stock state. I quickly realized that this had to be a custom job.

My Hero's gripper has a cross section similar to that shown in Figure 3. The inside of the gripper has a raised center section, which is the first area to touch an object. The sensor's design made use of this surface. I picked up some thin sheet metal stock from a local hobby shop and cut a few 0.1" conductors. I wrapped the inside of the gripper with two parallel contacts with several QTC Pills located along them. I originally wanted to use sensors on the inside of both gripper fingers in order to detect the pressure placed on an object between them. After looking at the gripper, I realized I needed to sense only one side of its finger. I decided that the second sensor would make more sense on the gripper's fingertip (sensing the complete closure of the gripper). Knowing when the gripper is completely closed indicates when an object is too small to grasp (unless the gripper tips are used).

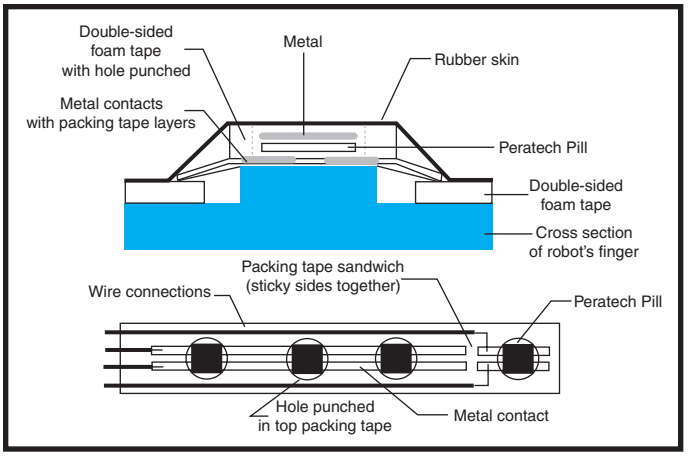


Figure 3—*The lower drawing shows the prototype sensor based on the Hero's gripper dimensions. The top drawing shows you how the sensor is mounted on the Hero's gripper finger.*

Photo 5 shows the sensor I made using clear packing tape to position and seal the contacts. Using a paper punch (in the top piece of packing tape), I positioned holes over the contacts where the QTC Pills were to be located. The double-sided foam tape atop the assembled sensor's contacts had corresponding holes punched. The wells, which hold a QTC Pill, have a top conducting disk punched from the thin sheet metal stock placed on top of them. The double-sided tape holds the sensor on the gripper; it allows a surface of rubber (from a Playtex glove) to seal the entire sensor. The rubber also prevents the gripper's surface from becoming slippery.

You can surface-mount the circuit in Figure 4 to flex circuitry to make a complete sensor package. The SPI interface allows an internal ADC to measure each sensor and report back with sensor pressure data. Alternatively, you can program a TTL output as a go/no-go output using preprogrammed EEPROM values as trigger points. The SPI interface can set or redefine these values. Your biggest concern should be choosing the divider resistors. These are based on the type of QTC material you're using and the gripper's sensitivity requirements.

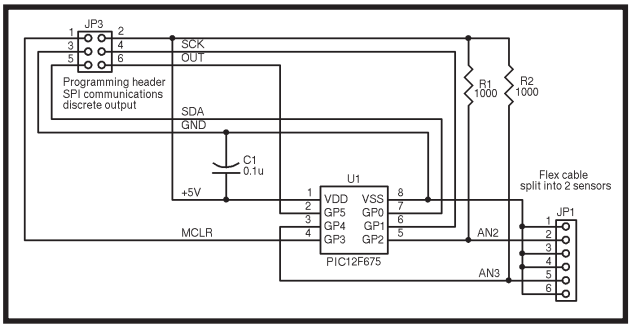


Figure 4—*The use of a surface-mount processor enables you to construct the complete sensor on a flex circuit. This design uses SPI for communication with the robot. This could be I²C, USB, or any other type communication channel you may be using on your robot.*

ELECTRIC DREAMS

Although my Hero still needs sensors to guide its hand toward an object, it can now grip things without mashing them to bits (see Photo 6). Robots of the future may be covered with skin made of QTC material. We've come a long way, baby. And although we have a long way to go, it's a seemingly small technological advance like this that will make all of our dreams possible.

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

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/177.

SOURCES

Load button cell
Futek Advanced Sensor Technology
www.futek.com

Solid-state pressure sensors
Honeywell
www.honeywell.com

PIC12F675 Microcontroller
Microchip Technology
www.microchip.com

QTC Pills and Switch Substrates
Peratech
www.peratech.co.uk

FlexiForce A201 sensor
Tekscan
www.tekscan.com

by Fred Eady

Test-Driving the Micro64

Planning an AVR-based industrial control or data logging application? Check out the Micro64, which is a modular AVR-based controller housed in a 42-pin, 1.5-cubic-inch potted brick.

As most of you know, designing a professional embedded project that requires a microcontroller complex, a PCB, and a protective enclosure from scratch isn't a trivial task. Many hours go into the mental design process before you spend days transforming your ideas into hardware. All aspects of the new design must be thoroughly thought through. Sometimes the enclosure is more difficult to create than the circuitry it envelopes. Making mistakes, especially when it comes to the enclosure, during the design and prototype process will cost you both time and money.

It's always a good idea to know how your final product will be used. But it isn't always possible. I've designed stuff for a particular purpose (at least it was in my mind) that my customers found handy for a different application. For example, suppose your embedded design must be able to be networked with similar devices or with devices capable of speaking your design's network language. To add a little insult to injury, what if your design needs to speak multiple protocols and you don't know which one you'll encounter in the field. What do you design in? What do you leave out? Oh, yeah, cost is a significant factor too. What do you do?

A number of reputable manufacturers offer numerous types of microcontrollers. You also can procure modularized microcontroller platforms that house some of the common off-the-shelf microcontrollers from third-party companies. The Atmel ATmega64 is the microcontroller of choice this month. I won't be telling you anything you don't know when I say that the AVR is an extremely popular microcontroller. But have you had the chance to sit down and read through an AVR datasheet lately?

Depending on the microcontroller, there are lots of knobs to twist and hardware configurations to choose from. If you go with an AVR, you'll be breadboarding or creating PCBs to implement your initial and final designs. At minimum, you'll need a regulated power source, a clock source, and a microcontroller reset circuit. If you need to communicate (most embedded designs do), you'll probably have to design in network interface circuitry. Chances are that somewhere down the road you'll need to revise your firmware or squash a latent bug. So, you'll have to design in a user-friendly programming port as well. There's never enough extra SRAM floating around. Depending on what your little AVR will have to do, you may find yourself clipping on an external SRAM IC.

If an AVR project is in your future, and if some of the aforementioned requirements sound familiar, don't reinvent the wheel at your bench. Take some time to get familiar with Micromint's new Micro64, which is a modular AVR-based controller housed in a 42-pin, 1.5-cubic-inch potted brick.

Micro64 BASICS

The Micro64's core is an ATmega64 microcontroller running at 11.0592 MHz. As you may already know, the ATmega64 natively supports 64 KB of in-system reprogrammable flash memory, 2 KB of EEPROM, and 4 KB of SRAM. Micromint's Phil Champagne enhanced the raw AVR core embedded within the Micro64. He added a full 32 KB of SRAM and a factory-loaded bootloader application that shares the AVR flash memory space with a set of factory-loaded, flash memory-resident utilities that provide you easy access to the Micro64's unique hardware features.

The Micro64A incorporates an optional two-channel 12-bit A/D converter. Both the Micro64A and standard Micro64 include the AVR standard on-chip, eight-channel, 10-bit A/D converter. The ante is upped on the communications side because both models sport upgraded multifunctional USART subsystems. A Micromint C library module provides code to enable both USART0 and USART1 in TTL, RS-232C, RS-485, or RS-422 modes.

Forget about selecting and populating a clock source and building up a regulated power supply. The Micro64 internalizes both functions. You can run the Micro64 with a 12-V wall wart. The 11.0592-MHz clock circuit is, shall I say, potted in. If you choose the Micro64 or the Micro64A, you'll be able to use the bootloader utilities to exercise the Micro64's real-time clock/calendar. And, yes, there's a battery input pin (VBAT) to keep the time real while the other Micro64 subsystems are snoozing.

Phil did a great job with the Micro64's enhancements. He also managed to squeeze in and reserve 29 general-purpose I/O lines, four timer/counters

with Compare and PWM modes, a two-wire (I^2C) interface, and an SPI port. The AVR's standard power management modes and watchdog timer are also available. The Micro64 communications subsystem makes it easy to network a number of Micro64 modules.

I've written enough about the Micro64 hardware. Download the datasheet and application notes from Micromint's web site for more information. Now I'll describe how I applied some power to the Micro64 and test-drove it.

CODING THE Micro64

My Micro64 came with a development board that provides all of the necessary power and communication hookups. Supporting power circuitry, communications subsystems, and a large breadboard area surround the Micro64 (see Photo 1). The various jumpers allow the communications port to be configured for RS-485, RS-232, or RS-422 operation. Screw terminals make easy work of attaching the Micro64 to a twisted-pair, half-duplex RS-485 or RS-422 network. The Micro64 development board is configured for standard RS-232C, which allows you to connect immediately to your PC's serial port via the Micro64 development board's DS14C232C RS-232 converter IC.

The Micro64 is based on the ATmega64, which means you can program it using AVR assembler, C language, or compiled BASIC. Micro64 example code for CodeVisionAVR and BASCOM-AVR is posted on Micromint's web site.

Micromint's bootloader program that runs on your PC communicates with the Micro64 bootloader firmware via a fast serial connection. There's a copy of BASCOM-AVR in the Florida room's compiler library. I came across the latest version of CodeVisionAVR as well. I have CodeVisionAVR up and running right now. So, as Steven Stills would say, "love the one you're with."

CodeVisionAVR & Micro64

CodeVisionAVR is an AVR C compiler from the house that Pavel built: HP InfoTech. CodeVisionAVR's

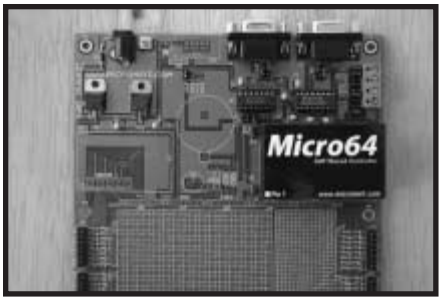


Photo 1—*The Micro64 was designed to participate in rugged industrial environments. The light-footed ATmega64 coupled with the 32 KB of SRAM allows the Micro64 to be used in a variety of applications.*

strength lies in its simplicity. The CodeVisionAVR C compiler is easy to set up and just as easy to learn to use. The support is outstanding as well.

The CodeVisionAVR compiler follows as closely to the ANSI standards as the AVR architecture will allow it to before giving in to the quirks of the AVR. The package includes a bunch of utilities that do everything from drive LCDs to configure and communicate with Dallas 1-Wire protocol devices.

If you can't find what you want in the example code, you can rub the lamp and summon the CodeWizardAVR automatic program generator. There's a big chunk of external SRAM begging to be used in the Micro64 brick. I dusted off the old lamp and called on the Wizard to see if he would help me write a simple Micro64 external SRAM read/write application.

After opening the CodeVisionAVR IDE, I opted to create a new project. I was asked if I wanted to call upon the Wizard. I did, and Photo 2a appeared on my screen. As you can see, I chose to create an application project.

The next step involved choosing the appropriate amount of external SRAM and defining how it should be accessed. My choices are shown in Photo 2b. I found that the Micro64 worked well with or without adding wait states. The external SRAM was addressed immediately following the AVR's 4-KB chunk of on-chip SRAM.

For the ATmega64, the first external SRAM address was 0x1100, with an ending address of 0x7FFF. The lower 4 KB of external SRAM is overlaid by the ATmega64's internal block of SRAM. Therefore, there is effectively only 28 KB of external SRAM available. If you need to access all 32 KB of external SRAM, the ATmega64 datasheet describes a way to mask the high address bits and swap in and out of External Memory mode to get at the hidden SRAM. Phil has also prepared an application note detailing how you can get at the 32 KB of SRAM within the Micro64.

USART1 was used by the Micro64 bootloader. However, I wanted to spill my SRAM read results out of a serial port, and that left USART0 as my only choice. Note that 9,600 bps was fast enough. I really didn't need a receiver for this application. Entering my USART0 configuration choices in Photo 2c was as far into Micro64 hardware configuration as I needed to go. I wasn't planning to use any of the other Micro64 peripherals in this spin.

I took some time to fill in some of the project information just to see where the Wizard would put it (see Photo 2d). After that, I generated my configuration and saved the files (micro64_sram.c, micro64_sram.prj,

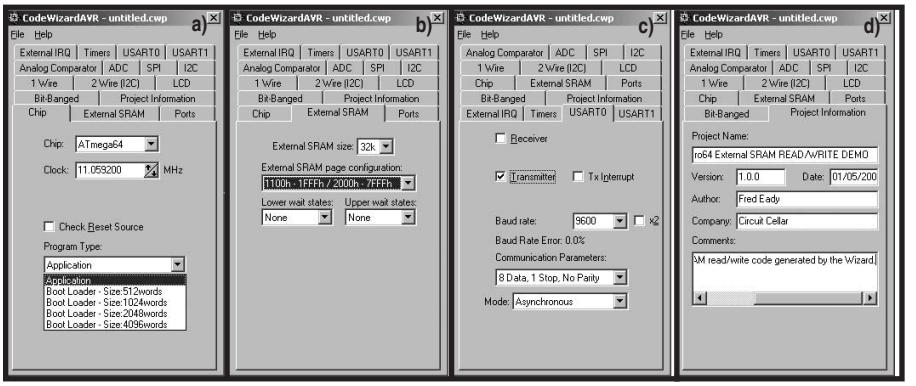


Photo 2—*I like to use Wizards to check my work. I'm old fashioned. I like to beat the datasheet pages ragged. When I'm done, I run the Wizards to see if I did things correctly.*

Listing 1—*If you take the time to enter all your hardware information into the CodeWizardAVR tabs, you'll end up with a skeleton source code page with all the peripheral initialization code in place. It sure beats tracking down the bits in the datasheet.*

```
#include <mega64.h>
//Standard I/O functions
#include <stdio.h>
//Declare your global variables here
void main(void)
{
// Declare your local variables here
//External SRAM page configuration:
//1100h-1FFFh/2000h-7FFFh
//Lower page wait state(s): None
// Upper page wait state(s): None
MCUCR=0x80;
XMCRA=0x10;
//USART0 initialization
//Communication parameters: 8 Data, 1 Stop, No Parity
//USART0 Receiver: Off
//USART0 Transmitter: On
//USART0 Mode: Asynchronous
//USART0 Baud rate: 9600
UCSR0A=0x00;
UCSR0B=0x08;
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x47;
while (1)
      {
// Place your code here
      };
}
```

Listing 2—*I didn't have to tap too many keys to get this little code snippet compiled and running. I can envision all sorts of data structures in external SRAM that can be exploited with this simple pointer technique.*

```
#include <mega64.h>
#include <MMRS485.h>    //Micromint's library for using both USARTs
#include <stdio.h>
// Declare your global variables here
int COM;                //If COM = 0, then use USART0 if it = 1 and
//then use USART1.
void main(void)
{
//Declare your local variables here
unsigned int *sram_pointer;
unsigned char sram_data;
*********************************************************************
//External SRAM page configuration:
//1100h-1FFFh/2000h-7FFFh
//Lower page wait state(s): None
//Upper page wait state(s): None
MCUCR=0x80;
XMCRA=0x10;
*********************************************************************
COM = 0;
//USART0 initialization
//Communication parameters: 8 Data, 1 Stop, No Parity
//USART0 Receiver: Off
//USART0 Transmitter: On
//USART0 Mode: Asynchronous
//USART0 Baud rate: 9600
UCSR0A=0x00;
UCSR0B=0x08;
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x47;
*********************************************************************
sram_pointer = 0x1100;          //Aim at SRAM location
sram_data = *sram_pointer;      //Read targeted SRAM location
printf("\r\nSRAM Data at 0x%04X BEFORE = 0x%02X",sram_pointer,sram_data);
sram_data = 0xAA;
*sram_pointer = sram_data;      //Write targeted SRAM location
sram_data = *sram_pointer;      //Read targeted SRAM location
printf("\r\nSRAM Data at 0x%04X AFTER  = 0x%02X",sram_pointer,sram_data);
while(1);
}
```

and micro64_sram.cwp) in my project folder. I cleaned up the original Wizard-generated code in Listing 1 by eliminating all of the unwanted initialization code for the ports, interrupts, and timers.

The next step involved adding source content that I knew I would need to select and use the Micro64's USART0. For starters, I added an `include` statement for the Micromint communications library (MMRS485.h). I declared a COM variable. The value of the COM variable determined which USART would be used in the application.

I accessed the Micro64 external SRAM with simple pointers. In addition to bringing in support for USART0, I added the local variables `*sram_pointer` and `sram_data` (see Listing 2). I aimed the pointer (`*sram_pointer`) at the first external SRAM location following the on-chip 4 KB of SRAM (0x1100) and read the SRAM location to put a stick in the ground for what was there already. I then pushed `0xAA` into the same external SRAM cubby I was pointing to and read the location again.

Now that you know how it's done with the CodeVisionAVR C compiler, let's take a look at a similar external SRAM read/write task. I used the BASCOM compiler.

BASCOM-AVR

The BASCOM-AVR BASIC compiler is similar to the CodeVisionAVR C compiler in that there are plenty of good examples and AVR-biased utilities included in the package. Some of the built-in BASCOM-AVR goodies are shown in Photo 3a, which is where I began setting up the BASCOM-AVR compiler for the external SRAM application. I selected wait states in this spin just for grins. The only other tabs I had to access were both of the Communication tabs, which allowed me to set the microcontroller clock speed and designate a default data rate and PC COM port.

I entered the Micro64 clock frequency and microcontroller USART0 data rate in the window shown in Photo 3b. The entries in Photo 3c are for the terminal emulator built into the BASCOM-AVR IDE.

The rubber hit the road in Listing 3. BASCOM-AVR allows you to carve out a chunk of external SRAM using the `$xramstart` and `$xramsize` operators. The `$xramstart` operator allows you to begin at a desired point within the external SRAM address range while the `$xramsize` operator overrides the size of the external SRAM I designated in the set-up window in Photo 3a.

After configuring and initializing the Micro64's USART0, I defined XRAM byte `Sram_data` and XRAM pointer `Sram_pointer`. The actual external SRAM data location I was exercising was at 0x0100 (`Sram_data`). I created a pointer to the external SRAM memory location (`Sram_pointer`) using the BASCOM-AVR's Overlay parameter. This allowed me to access the Micro64's external SRAM via pointers, just as I had done with CodeVisionAVR.

Both BASCOM-AVR and CodeVisionAVR include terminal emulators in their IDEs. Photo 4 (page 59) is the combined result of running the CodeVisionAVR C and the BASCOM-AVR BASIC external SRAM access code I've just described.

Listing 3—*Use of the Overlay operator is a clever way to create a pointer in BASIC language. The idea here is the same as with the CodeVisionAVR C code. The only real difference is where the compilers choose to call the beginning of the external SRAM space.*

```
$regfile = "m64def.dat"
$baud1 = 9600
$xramstart = &H100
$xramsize = &H1000
'Configure the serial port.
Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0
'Open the serial port
Open "com1:" For Random As #1
Dim Sram_data As Xram Byte At &H0100
Dim Sram_pointer As Xram Byte At &H0100 Overlay
Sram_pointer = &H00
Print #1 , "SRAM LOCATION 0x0100 BEFORE = 0x" ; Hex(sram_pointer)
Sram_pointer = &HAA
Print #1 , "SRAM LOCATION 0x0100 AFTER  = 0x" ; Hex(sram_pointer)
Close #1
End
```

HOW'D HE DO THAT?

So as not to put the cart before the horse, I need to explain how I set up my Micro64 development board in relation to the Micro64 Bootloader PC application. Photo 5 (page 59) pretty much tells the story. I attached the Micro64's USART1 to my PC's COM1 via the Micro64's development board nine-pin shell connector J2. I used COM1 as the bootloader programming port for both the CodeVisionAVR and BASCOM-AVR environments. I permanently attached COM2 to the USART0 port on the Micro64 (J3). I used it to pass data between the Micro64 and the respective IDE terminal emulators in both compiler environments.

Programming the Micro64 with the bootloader arrangement was fast and easy. After the C or BASIC code was compiled into a hex file, I simply selected the resultant hex file in the Application Code window and clicked the Download button. I then pressed the Micro64 Reset button and the bootloader components on the PC and within the Micro64 communicated and performed the download/program operation. If I had any EEPROM data to load into the Micro64, the program/download operation was identical to the application data download/program sequence (see Photo 5).

Micro64 UTILITIES

The Micro64's utilities, which are a group of functions preloaded with the bootloader into the upper 4 KB of program space, are one of the neatest things about it. Data relating to the functions is passed via a block of reserved SRAM between addresses 0x0FFB and 0x0FFF. The functions cover processes of using the 12-bit A/D converter, the real-time clock, and the I²C communications subsystem.

Suppose you want your application to read the RTC and determine the day of the week. For the Micro64, the function that reads the day of the week is located at offset 0x7D05. To access this particular function via the CodeVisionAVR, you must declare and define the function with the following statement:

```
void(*Read_Day_Of_Week)(void)=
0x7D05;
```

Either the result of the operation or an error indication will transmit via SRAM addresses 0x0FFE and 0x0FFF.

You must define the SRAM area to obtain the resultant data. A simple C statement does this:

```
unsigned int Result @ 0x0FFE
```

Assuming you do the housekeeping

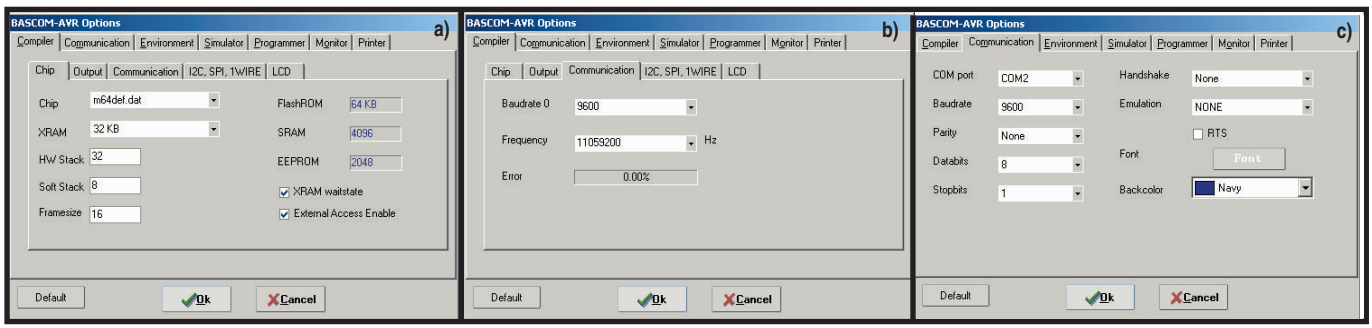
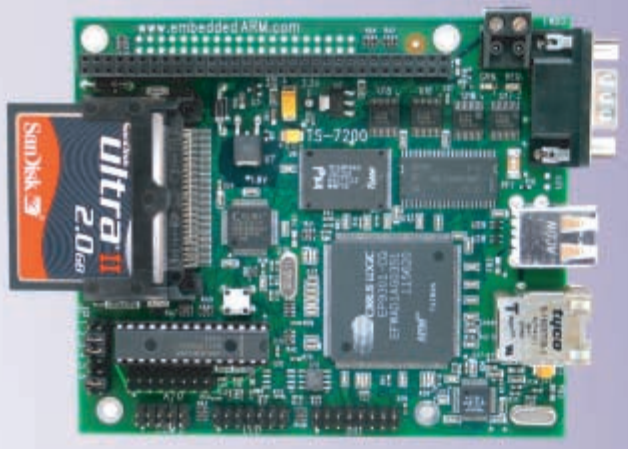


Photo 3—*You can't miss if you have the right set of numbers. As you can see in the tab structures, you can invoke some hefty automation within your BASCOM-AVR project.*

200 MHz ARM - PC/104



Shown with optional A/D converter, Compact Flash and RS-485

TS-7200 $149 $119 qty 100

- 32 MB SDRAM (64 MB optional)
- 8 MB Flash (16 MB optional)
- Compact Flash socket
- 10/100 Ethernet
- 2 USB ports
- 20 Digital I/O, A/D options
- 2 Serial Ports
- Optional USB WiFi adapter
- Support for Windows® CE

- EP9302 ARM9 processor with MMU
- Boots TS-Linux from on-board flash in 5 seconds
- TS-Linux for ARM is open source GNU/Linux
- Low power +5v @ 400mA ▪ 3.8" x 4.5"

Linux 2.4 NetBSD

Windows is a registered trademark of Microsoft Corporation

200 MHz ARM - PC/104

Shown with optional A/D converter, USB Flash drive and RS-485

TS-7250 $149 $119 qty 100

- 32 MB SDRAM (64, 128 MB optional)
- 32 MB Flash (128 MB optional)
- 10/100 Ethernet
- 2 USB ports
- 20 Digital I/O
- 2 Serial Ports
- 5 ch. 12 bit A/D (optional 13 channels)
- Optional USB WiFi adapter

- EP9302 ARM9 processor with MMU
- Boots TS-Linux from on-board flash
- TS-Linux for ARM is open source GNU/Linux
- Low power +5v @ 400mA ▪ 3.8" x 4.5"

The 256 MB USB drive shown holds our Debian 3.0 self-hosting development environment for ARM. Booting to the development file system allows code development to take place directly on the target hardware eliminating any issues with transfering newly compiled files from the desktop to the target device.

Cross-compiler tools are also available for Windows and Linux.

www.embeddedx86.com

(480)-837-5200
16610 E. Laser Drive #10
Fountain Hills, AZ, 85268 USA

Technologic SYSTEMS

133 MHz 5x86 - PC/104

TS-5600
$279

- PC/104
- PCMCIA slot
- 10/100 Ethernet
- SPI bus
- 2 Serial Ports
- Compact Flash
- 24 Digital I/O
- 12-bit 8 chan. A/D option

- AMD Elan 133 MHz 586 with 32 MB RAM (64 MB optional)
- PC compatible: BIOS, DOS, RTC, I/O & memory map
- TS-Linux 3.0 is open source GNU/Linux - Glibc 2.2.5
- Power +5v @ 830mA
- 4.3" x 5.6"

Shown with optional flash modules A/D, 3S-485 and Merlin cellular modem

33 MHz 386 - PC/104

TS-3400
$159

- PC/104
- Low EMI
- DiskOnChip
- DiskOnChip
- 37 Digital I/O
- 2 Serial Ports

- 33 MHz Intel 386EX with 8 MB RAM (16 MB optional)
- DOS-ROM or TS-Linux pre-installed
- PC compatible: BIOS, DOS, RTC, I/O & memory map
- 1 MB flash drive
- Power: +5v @ 500 mA

Shown with optional flash module and RS-485

PC/104 Ethernet Adapter

TS-ETH10 **$49**

- PC/104 daughter board
- 10baseT Ethernet
- DOS, Linux and NetBSD drivers
- Link and activity LED
- Header for external LED hookup
- 3.5" X 1.6"
- Jumper selection allows up to 4 boards per system

Shown with optional 16 bit connector

Technologic SYSTEMS

- PC/104 Computer Boards
- Enclosures
- PC/104 Peripherals
- Custom Designs
- Exceptional Support

PC/104 Peripherals

Enclosures $85
Metal enclosure for Single Board Computers provides regulated +5 VDC from 12-38 VDC input power. Connections available at back panel. Suitable for office or factory.

TS-SER4 $89
Four PC compatible COM ports using the 16550 UART with 16-byte FIFOs, full handshaking on two ports, RS-485 option on two ports, jumper configurable.

TS-DIO24 $39
24 0-5V DIO lines, opto-22 compatible 50 pin connector, 4 outputs can drive up to 30V @1 Amp, 4 inputs are IRQ capable, I/O address jumper selectable.

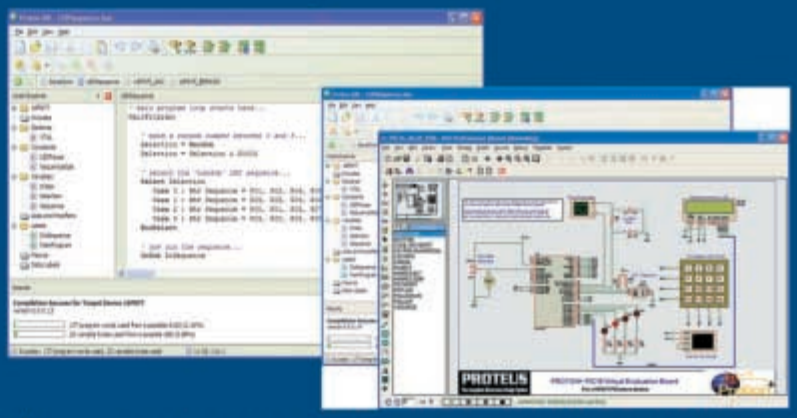
TS-Modem $129
33.6K baud, jumper configurable as COM1-6, supports IRQs 3-15, V.32 V.42 compliant, standard AT commands, caller ID, 56K baud optional.

TS-Linux v3.0

- Introducing TS-Linux version 3.0 Our latest distribution is built from open-source featuring a 2.4.18 kernel, Apache web server, FTP client/server, bash, etc. With a footprint of 15 MB or less this is a great place to start your application.
- The TS development environment compiles application code using the libraries and headers installed in TS-Linux. Applications developed on your desktop computer copy directly.
- Our User Mode Linux patched kernel runs TS-Linux 3.0 kernel and file system as an application on your desktop computer during development.

PIC Programming Made Easy!

Proton+ PICBASIC Development Suite



Next Generation IDE

Proton IDE is a professional and powerful visual Integrated Development Environment (IDE) which has been designed specifically for the Proton Plus compiler. Proton IDE accelerates product development in a comfortable user environment without compromising performance, flexibility or control.

- *Code Explorer*
- *Compiler Results*
- *Programmer Integration*
- *Integrated Bootloader*
- *Real Time Simulation Support*
- *Serial Communicator*
- *Online Updating*
- *Plugin Architecture*

INTRODUCTORY OFFER! Save $10 LIMITED TIME

SAVE TIME WITH END TO END INTEGRATIONS

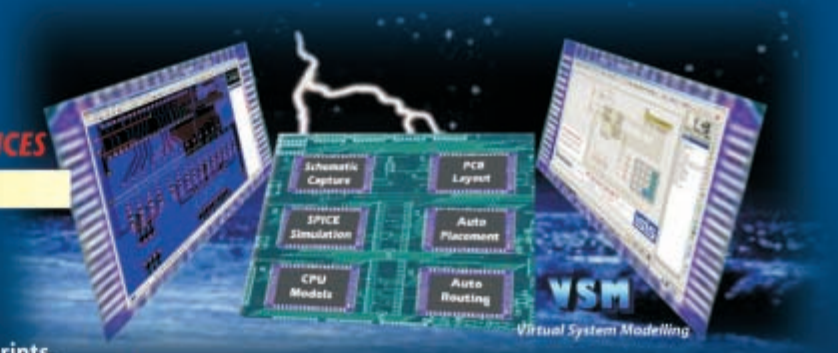
NEW IDE - New IDE makes development using Proton+ even faster and more intuitive!
COMPILER - The popular Proton+ compiler has enhanced support for I²C, SPI, Dallas 1-wire bus, RS232, X10, Compact Flash Memory Cards and USB.
VIRTUAL SIMULATION - Simulate your project in RealTime using the integrate Proteus Virtual PIC Boards.

Visit www.r4systems.com to see our latest projects using **Proton+**

PROTEUS

The Complete Electronics Design System

EASY TO USE CAD TOOLS AT FANTASTIC PRICES



Schematic and PCB Layout

- Powerful and flexible schematic capture.
- Auto-component placement.
- Rip-up and Retry PCB routing.
- Polygonal gridless ground planes.
- Library of over 8000 schematic and 1000 PCB foot prints.
- Bill of materials, DRC reports and more.

Mixed Mode SPICE Circuit Simulation

- Berkeley SPICE3F5 simulator with custom extensions for true mixed mode and interactive simulation.
- Six virtual instruments and 14 graph based analysis types.
- 6,000 models including TTL, CMOS and PLD digital parts.
- Fully compatible with manufacturers' SPICE models.

Proteus VSM - Co-simulation and debugging for popular Micro-Controllers

- Supports PIC16 & PIC12, AVR, 8051, HC11 and ARM micro-controllers. Latest version includes 40 new PIC18's.
- Co-simulate target firmware with your hardware design.
- Includes interactive peripheral models for LED and LCD displays, switches, keypads, virtual terminal and much more.
- Provides source level debugging for popular compilers and assemblers from HiTech PICC, Crownhill, IAR, Keil and others.

PCB AutoRouting

- Proteus PCB design includes an interface to the Electra Gridless autorouter.

"For the cost of the software compared to the productivity gains, I consider Proteus to be pivotal in the commercial viability of my company and by far represents the best value for money of anything Tempus possesses."

ROB YOUNGS, Tempus Consulting

FREE DOWNLOADABLE DEMO

Save Time. Save Money.

Proteus Starter Kit – $199

Complete Systems from $449

"This is clearly superior in every respect."

R4 SYSTEMS INC.

www.r4systems.com
Tel: 905•898•0665 info@r4systems.com

correctly, all you have to do to get the day of the week is call the function at 0x7D05:

```
(*Read_Day_Of_Week) ();
```

The `Result` variable will be held at SRAM location 0x0FFE in the form of an integer. As a result, all you have to do is simply read the `Result` memory location to obtain the day of the week. Now I'll show you how to do this with the language of BASCOM-AVR.

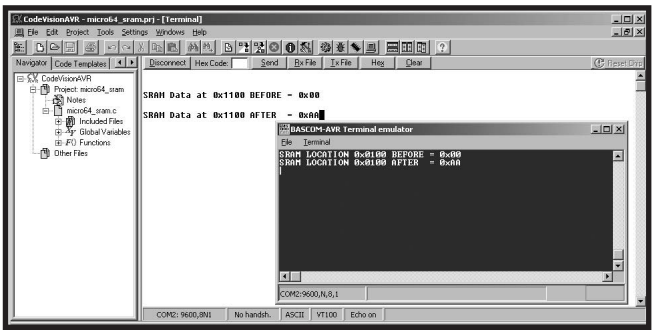


Photo 4—*Accessing the Micro64's external SRAM is extremely simple after you've got the hang of using the pointers. Both the CodeVisionAVR and BASCOM-AVR compilers are easy to use and loaded with tools to make your project design flow smoothly.*

Things are different with BASCOM-AVR. You don't need to declare and define the function because inline assembler statements perform the function call. The BASCOM-AVR function call for the day of the week looks like this:

```
$asm
!Call $7D05
$end Asm
```

The function of your SRAM word at 0x0FFE is the same in BASCOM-AVR and CodeVisionAVR. It's declared in BASCOM-AVR in this manner:

```
Dim Result As Word At &HFFE
```

After you've made the inline assembler call, the result will be passed on to the BASCOM-AVR application via the word at SRAM location 0x0FFE. Again, just like you did with CodeVisionAVR, you must simply read the result location that you defined using a word variable that's been defined and declared in the BASCOM-AVR application. For instance, if you declare "Day_Of_Week" as a word (`Dim Day_Of_Week As Word`), then getting the day of the week data would be as simple as `Day_Of_Week = Result`.

If the day of the week isn't correct, you must be able to correct it by writing to the RTC. As far as calling the functions goes, the song remains the same, with the exception of stuffing the reserved SRAM location 0x0FFD with the day of the week data and calling the write day of week function at 0x7D29.

After you've got the hang of making the calls, stuff in your parameters and retrieve your resultant data. You can easily navigate and use any of the Micro64 built-in functions. For instance, the I²C send-byte and receive-byte functions (0x7CB8 and 0x7CDD, respectively) utilize the function-reserved SRAM address of 0x0FFB as the holder of the I²C slave address with reserved SRAM location 0x0FFC carrying the I²C data. The call to the I²C function is performed in an identical manner to the function calls I've described for reading and writing the RTC. The idea is to provide a simple interface to the Micro64's enhancements.

LAYING BRICKS

You probably noticed there's a Micro128 brick in Photo 5 that's based on the ATmega128. The Micro128 operates in the same way as the Micro64; however, the Micro128's program memory space is doubled along with the doubling of the Micro128's EEPROM memory space. Of course, calling the functions is no different from what I've described. But the function call addresses will be different for the Micro128. All the differences between the Micro64 and the Micro128 bricks are described in the Micro64/128 datasheet.

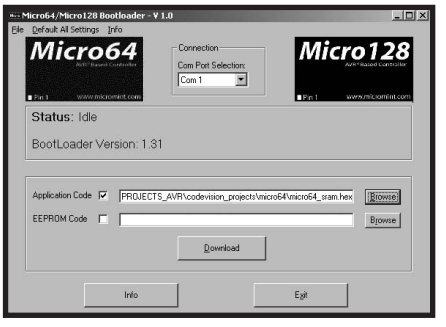


Photo 5—*It doesn't get much easier than this. Just compile your code into a hex file, point to it, click the Download button, and reset the Micro64.*

Whether you're an advanced microcontroller system designer or a what-pin-is-that? beginner, it's easy to harness the power of the Micromint AVR bricks. I've heard that making and transporting the mortar is the hardest work associated with bricklaying. If I were to equate programming the Micro64 with laying bricks, I would say that the mortar (CodeVisionAVR and BASCOM-AVR) has been perfectly premixed and the bricks (Micro64 and Micro128) are the highest quality. The Micro64 is perfectly suited for networked industrial control and data-logging applications. Embedding a Micro64 in your next project won't be complicated.

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

RESOURCE

Micromint, "Micro64: Microcomputer /Controller Featuring the ATmega64 or the ATmega128," rev. 1.1, 2005, www.micromint.com/datasheets/Micro64.pdf.

SOURCES

ATmega64/128 Microcontrollers
Atmel
www.atmel.com

CodeVisionAVR
HP InfoTech
www.hpinfotech.com

BASCOM-AVR
MCS Electronics
www.mcselec.com

Micro64/128 Embedded controllers module
Micromint
www.micromint.com

Practical Application for TDD (Part 2)

Automated Test-Driven Environment

Last month you learned about the numerous advantages to test-driven development (TDD), which is a primary component of an agile development technique. Now you'll find out how to build a prototype TDD environment for an embedded system.

In the first part of this series, we introduced you to the idea of test-driven development (TDD). Briefly described, TDD involves replacing volumes of documents for customer requirements and developer ideas with a series of specified and agreed upon tests before coding starts. The TDD environment offers advantages to those of you who want to fully embrace the TDD concepts. It's also advantageous to those of you who realize that an automated test environment quickens the familiar test-last approach.

Last month we also showed you a simple high school recruitment project that featured the Analog Devices ADSP-BF533 Blackfin microcontroller (see Photo 1). We explained that developing up-front customer tests is a new, hard-to-acquire skill. However, using TDD for developer tests involves a change in point of view rather than a total change of mind-set.

Lastly, we showed you how to develop tests for the software to calculate the temperature from the duty cycle of an Analog Devices TMP03 thermal sensor. Note that this temperature calculation was a straight software testing issue. This could have been handled with the original CppUnitLite TDD environment developed by Michael Feathers and distributed under the terms and conditions of the GNU license (http://c2.com/cgi/wiki?CppUnitLite).

In this article, we'll describe the process of modifying TDD to produce a prototype TDD environment for embedded systems.

CppUnitLite

The key to adapting Michael's CppUnitLite for an embedded system involved minimizing memory usage in order to ensure that it would fit within the embedded system environment. In particular, we replaced print statements that involved formatting with simpler ones like `puts(astring)`. The sheer generality of the formatting associated with statements like `cout << value` and `printf("%d", value)` pulls sufficient code from the <stdio> library to occupy most of the available program memory space, possibly leaving little room for additional code.

The changes don't solve all the problems associated with the reports generated during TDD in a live rather than simulated environment. Messages are sent back from the target to the development environment over a serial, USB, or JTAG connection. Transmitting strings is a complex task. The target must be stopped and switched into Emulator Interrupt mode. This permits the message to be wangled out of the target one character at a time, but it completely disrupts the real-time operation of the embedded system.

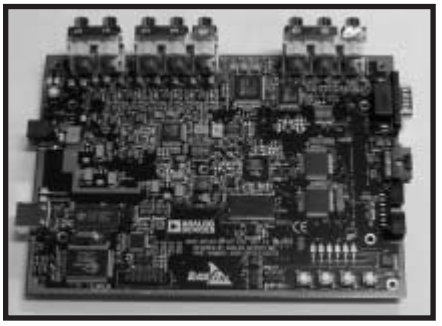


Photo 1—*The 500-Mhz ADSP-BF533 Blackfin EZ-Lite evaluation board has audio and video capability and digital inputs. There are six LEDs that can flash.*

There is a background telemetry channel (BTC) in the ADSP-BF533 Blackfin development environment specifically designed to permit message interchange with minimal disruption of real-time operation. We've demonstrated the advantages of BTC embedded TDD, but we have more work to do before gaining the full advantage of this high-speed communication channel.

EMBEDDED TDD

The modified TDD environment's source code was compiled, linked, and downloaded to the embedded system without changes to the standard environment running on the VisualDSP++ IDE that came with the evaluation board. Given that the code was written in C++ language, we didn't expect any compatibility issues. We were concerned, however, about code size issues.

The original TDD environment requires a number of key extensions to be added for use in the embedded system environment. The first requirement is the ability to reconfigure the hardware environment to a known state prior to issuing a series of tests. To capture this functionality, we developed three new hardware-oriented TDD procedures. The `__CaptureKnownState()` function is called as the first line of a `main()` function run on a powered board. It automatically saves the C++ initial

environmental setup to a file. We wrote this function to be easily upgradeable when new features of the embedded system are added (i.e., when new peripherals are developed). `__SaveUserRegAndReset()` and `__RecoverUserReg()` are a pair of functions used at the start and end of a test. The former saves the current user processor state and resets the system to a known state. The latter restores the initial user processor state.

The jury is out regarding the utility of the functions. Having the system in a known state before testing prevents many errors; however, we found that testing with a system that's unintentionally in an unknown state also uncovers unexpected system configuration issues.

A second useful hardware test group, the `WatchData` class and `WATCH_MEMORY_RANGE()` macro, provides the ability to watch planned (or unintentional) action in a memory range on the processor. These work through hardware breakpoints on a running system rather than via static profiling on an architectural simulator or statistical profiling (snapping a quick look) on a running system. This test class was initially developed as an instructor test. An instructor (with close knowledge of the system architecture) would write such a test to examine whether or not students learning about a processor have properly configured the registers of a peripheral.

In practice, this test class proved to be much more utilitarian. It's a test that might be useful if you know exactly what needs to be done, but you're distracted by an interruption in the middle of the creative process. Used in this way, it's probably better called the `AVOID_LSD_ERROR` test class, where *LSD* stands for the *little stupid details* that waste time.

Finally, `Timer` class provides a series of timing utilities and a test. The first permits a measurement of the execution time of a `MEASURE_EXECUTION_TIME(Func(pars))` macro. The second is an actual test called `MAXTIME_ASSERT(Func(pars))`, which determines whether or not a function satisfies some strict real-time test. This is the first of a series of functional and nonfunctional tests for embedded systems planned for development. An example of a nonfunctional test is `MAXPOWER_ASSERT()`, which is used to determine, as a design requirement, whether or not a given algorithm meets the specific power restriction designated by a customer.

The proposed `timer` and `WatchData` classes are conceptually general, but they must be implemented specifically in order to meet the available resources on a given processor. At the same time, they mustn't restrict the normal development of code. The hardware watchpoint unit on the Blackfin processor can determine the number of reads and writes in either data or program RAM. These were used for the `WatchData` class. The Blackfin `timer` class was developed with the system clock.

HARDWARE SETUP

Figure 1 is a diagram of our thermal arm wrestling video game. The major hardware component involves the need to read the asynchronous signals from the two temperature sensors and to determine the values of the two switches that reset and start the game. The four inputs—programmable flags (PF)—are easily accessible on the Blackfin EZ-Lite evaluation board.

Our original idea was to use an approach where changes on the PF-driven interrupts cause changes in various flags, allowing the main task to transition between three states: waiting for the game to start; playing the game; and resetting the game while it's running (or after a winner has been determined). The interrupts on the temperature sensor-driven lines would be used to access the value of a free-running core timer. This timing information can be processed to generate the temperature values.

We decided instead to use a single core timer-driven interrupt rather than four PF-driven interrupts. During the timer interrupt routine, we can determine the sense of the input signals and set the necessary flags to transfer between the main task states. We also can count the number of interrupts between the various high-to-low and low-to-high transitions of the temperature sensors. The counts would be sufficient as only a ratio of times, rather than absolute times, is needed to calculate the temperature. We can guarantee sufficient precision in the time counts by having the timer interrupts occur frequently.

TIMER TESTS

You'd think setting up the Blackfin core timer would be a simple task. You must place the required period (reload value) in the TPERIOD register and put the initial count in the TCOUNT register. Next, you have to set the TSCALE register. This determines the relationship between the core timer clock and the 500-MHz system clock. All that remains is switching the bits in the TCNTL control register so the timer comes out of Low-Power mode, the automatic reload feature is enabled, and the timer is on.

Everything seems straightforward, so why bother to write a test? Listing 1 (page 62) shows the `SetCoreTimer()` test. Listing 2 (page 62) shows the Blackfin assembly code required to satisfy the test. During the test, a `WatchData` class object called `coretimer_reg` is established to watch the action occurring among the four memory-mapped core timer registers. The expected register values after the

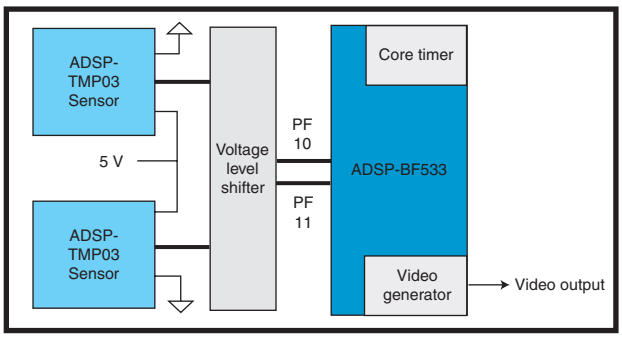


Figure 1—*Two ADSP-TMP03 thermal sensors provide square waves whose duty cycles vary with temperature. The pulse signals (5 V) are limited to 3 V by a resistor/Zener diode network before being fed into the PF signal lines of the Blackfin evaluation board. The on-chip core timers are used to determine the duty cycles. The calculated temperatures are used to position the game cursor on video frames stored in the SDRAM on the board. A background DMA task transfers the video frames from the SDRAM to the video decoder and then to the video screen.*

SetCoreTimerASM() function are set up and used for validation.

The WATCH_MEMORY_RANGE macro is called to use hardware breakpoints to record the internal events when activating the SetCoreTimerASM() function. Finally, two checks are made. Using the getReadsWrites() method of the WatchData class, the first determines whether the expected number of writes occurs to four core-timer registers. The second, using the getFinalValue() method, compares the actual final and expected values of the core timer registers.

The Blackfin assembly code in Listing 2 is easy to follow because of the processor's C language-like assembly code syntax in the processor. First, a pointer register is initialized with the address of the required memory-mapped core timer register. Following this, the function's incoming parameters COUNT (in R0), PERIOD (in R1), and SCALE (in R2) are stored in the timer registers. The old timer control value is stored and then returned when the function exits (in R0) before the timer is enabled in the required mode.

As it turns out, this code sequence, as written, requires the expected four writes to the timer registers, together with an additional reading of the timer control register (to capture its initial value) that wasn't taken into account during the initial test design. Because the initial test was developed when only four reads and writes were planned, the test macro must be rewritten to CHECK(coretimer_reg.getReadsWrites()==5); to meet the true SetCoreTimer() function requirements. The test and code match perfectly after this change. Or do they? Listing 3 shows the test some students wrote as part of an assignment using TDD to validate their SetupTimer() routines. Much to our embarrassment, our code didn't pass that test. Can you spot the difference between the tests? More importantly, why did our code pass our test and fail the students'?

The students' test was a better test. We used the same values for both the PERIOD and COUNT registers. A developer could have switched the values when writing to the core timer registers. That code would've passed a test where the same period and count values were used, but it would've contained a serious hidden defect. The students' test avoids this potential defect. The assembly language code in Listing 2 doesn't suffer from this

Listing 1—*The test for the SetupCoreTimerASM() function is straightforward. Three known parameters pass into the function for direct placement into the core timer register. A fourth constant value is placed in another register. However, the students revealed a fatal flaw in our test.*

```
#define SCALE 0
#define PERIOD 0x2000
#define COUNT 0x2000
typedef ulong unsigned long int;
TEST(Test_SetCoreTimer, INSTRUCTOR) {
   __SaveUserRegAndReset ( );
WatchData<unsigned long> coretimer_reg(4, pTCNTL, pTPERIOD,pTSCALE,pTCOUNT);
//Set up expected values
   unsigned long expected_value[] = {0x0, PERIOD, SCALE, COUNT};
   WATCH_MEMORY_RANGE(coretimer_reg,
       (SetCoreTimerASM(COUNT, PERIOD, SCALE)),
       READ_CHECK | WRITE_CHECK);
   __RestoreUserReg( );
   CHECK(coretimer_reg.getReadsWrites() = = 4);
   ARRAYS_EQUAL(expected_value, coretimer_reg.getFinalValue(), 4);
}
```

Listing 2—*Setting up the core timer registers requires three writes with passed parameters and a fourth with a known fixed value.*

```
        .global _SetCoreTimerASM;
        .section L1_code;
//ulong SetCoreTimerASM(ulong count, ulong period, ulong scale);
_SetCoreTimerASM:
        P0.H = hi(TCOUNT);      P0.L = lo(TCOUNT);
        [P0] = R0;      //First parameter passed in R0
        ssync;
        P0.H = hi(TPERIOD);     P0.L = lo(TPERIOD);
        [P0] = R1;      //Second parameter passed in R1
        ssync;
        P0.H = hi(TSCALE);      P0.L = lo(TSCALE);
        [P0] = R2;      //Third parameter passed in R2
        ssync;
        P1.H = hi(TCNTL);       P0.L = lo(TCNTL);
        R0 = [P1];      //Return the old control register value
        ssync;
        R1 = 7;         //RELOAD | ENABLE | NORMALPOWER
        [P1] = R1;      //Activate the timer
        ssync;
_SetCoreTimerASM.END:
        RTS;
```

Listing 3—*This student-developed test might be a better test than the one in Listing 1. Different values are assigned to the period and count registers. This can catch a code defect where writes to these registers have been swapped when implementing the SetCoreTimer() function. Although our SetCoreTimerASM() assembly function routine doesn't have that particular code defect, it still fails the test.*

```
#define SCALE 0
#define PERIOD 0x2000
#define COUNT 0x4000
typedef ulong unsigned long int;
TEST(Test_SetCoreTimer, STUDENT) {
     __SaveUserRegAndReset ( );
WatchDataClass<unsigned long> coretimer_reg(4, pTCNTL, pTPERIOD,
  pTSCALE, pTCOUNT);
//Setup expected values
     unsigned long expected_value[] = {0x0, PERIOD, SCALE, COUNT};
     WATCH_MEMORY_RANGE(coretimer_reg,
         (SetCoreTimerASM(COUNT, PERIOD, SCALE)),
         READ_CHECK | WRITE_CHECK);
     __RestoreUserReg( );
     CHECK(coretimer_reg.getReadsWrites() = = 4);
     ARRAYS_EQUAL(expected_value,
         coretimer_reg.getFinalValue(), 4);
}
```

Listing 4—*This test demonstrates the test for the ISR.*

```
#include <sys/exception.h>
EX_INTERRUPT_HANDLER(InterruptHandler);
extern volatile long int number_coretimer_interrupts;
#include <signal.h>
TEST(INTERRUPT_TEST,  Coretimer) {
        __SaveUserRegAndReset ( );
        register_handler(ik_timer, InterruptHandler);
        int number_of_times_round_loop = 0;
        number_coretimer_interrupts = 0;
        while (number_coretimer_interrupts < 10) {
                raise(ik_timer); //Force a software interrupt
                number_of_times_round_loop++;
        }
        register_handler(ik_timer, EX_INT_IGNORE);
        CHECK(number_of_times_round_loop = = 10);
        __RestoreUserReg( );
}
```

defect, but it fails the students' test. Our code failed because of a feature in the Blackfin silicon that wasn't in the hardware manual.

The Blackfin processor has a number of timers: the core timer (used in this project), a system clock, a watchdog timer, and three pulse-width-modulated timers for various microcontroller applications. The PWM timers are designed to continuously change duty cycles. However, the core timer typically isn't programmed in this mode. Its standard mode of operation is to produce interrupts at regular intervals, which means the period value is continuously used to reload the count value. If the core timer is powered down, loading the PERIOD register automatically loads the COUNT register with the same value.

We wrote the assembly code version of the `SetupCoreTimerASM()` function from the bottom up (i.e., the first parameter into a timer register, the second parameter into a register, and so on). However, because of the way the timer hardware had been optimized, this meant that the value put into the COUNT register was overwritten with a PERIOD value that was written to the timer registers at a later time. If we had used a top-down approach (i.e., third parameter, rather than the first, into a Blackfin register, etc.), or if we had had a different order of the parameters in the function prototype, then the COUNT register value wouldn't have been destroyed by the period value. This result indicated an unanticipated advantage in the `WatchData` class: the ability to spot undocumented features and changes between different releases of the board and other hardware.

FLAG TESTING

Because of space limitations and processor-specific characteristics of the code, we won't explain how the core timer interrupt service routine (ISR) measures the temperature sensors and switch positions. Nor will we explain how to set up the code and use the digital level programmable flags. However, the tests developed for these routines reveal the capabilities, limitations, and potential of the TDD environment.

Listing 4 tests for the core timer ISR's basic functionality. A software interrupt is issued using the `raise()` procedure found in the standard C language development. A software interrupt occurs each time around a loop. The ISR is supposed to increment a known semaphore `number_coretimer_interrupts`. The `register_handler()` function provides calls the VisualDSP C++ run-time environment to place the ISR's address in the event handler and unmask the timer interrupt bit in the global IMASK interrupt mask.

It was unclear whether or not a software interrupt produced by calling the `raise()` function requires a corresponding call to a `lower()` function to clear the interrupt and stop reentrance to the ISR. Therefore, we added an additional `CHECK()` operation to ensure that the number of

times the ISR was entered was equal to the number of calls to `raise()`, which should be equal to the number of times around the loop. In hindsight, this was a pointless test. If a call to the `lower()` function is required to stop the ISR being reentered, then the `CHECK()` statement wouldn't be reached. Regardless, the test shown in Listing 5 demonstrates the functionality of the ISR.

Developing the test for the functionality of the programmable flag input routines showed us the problems. We needed to push the test-driven environment to the next level. Now the tests must involve peripherals with externally generated signals rather than the internal signals provided by the core timer. One possible solution is to switch the tests away from the real board and place them on the architecturally accurate simulation environment available with the development IDDE. Although the VisualDSP++ IDDE for the ADSP-BF533 supports numerous peripheral simulations, PF simula-

Listing 5—*Known input signals can be manually established prior to the test. But this approach defeats the purpose of an automated testing approach.*

```
TEST(CheckPFInput, MANUALTEST) {
        __SaveUserRegAndReset ( );
        WatchData<unsigned short> pf_reg
        (1, (unsigned short*) pFIO_FLAG_D);
//Set up a return value
        unsigned short ret_value = 0;
        WaitForHardwareReady("Set switch pattern to 0x03");
        ResetPFForInputASM( );
        WATCH_MEMORY_RANGE(
         pf_reg, (ret_value = ReadPFASM( )),
        READ_CHECK | WRITE_CHECK);
        __RestoreUserReg();
        unsigned short expect_val = 0x03;
        USHORTS_EQUAL(expect_val, ret_value);
        CHECK(pf_reg.getReadsWrites() = = 1);
        __RestoreUserReg( );
}
```

Listing 6—*You can avoid the process of manually setting the input signals by using other internal or external resources to simulate the timing of expected signals from controlled devices. But keep in mind that the reliable and accurate generation of the simulated signals, as well as the recovery of the simulation from (probable) error states, can be complex and time-consuming.*

```
volatile long int __LEDstate = 0;
volatile long int __number_of_interrupts = 0;
//Interrupt task using a variable duty-cycle
//TIMER2 signal to mimic the changes in the pulse-width modulated
//ADSP-TMP03 temperature sensor signal
EX_INTERRUPT_HANDLER(SimulateTMP03_ISR) {
        __number_of_interrupts++;
        *pTIMER_STATUS |= TIMIL2;
//Clear the interrupt
        if ((__number_of_interrupts & 1) == 1) {
```

(Continued)

Listing 6—*Continued.*

```
            __LEDstate  |= _TMP03_HIGH_STATE;
//Output current simulated TMP03 signal
        WriteOutputASM(__LEDstate);
//Prepare for next simulation state
        *pTIMER2_PERIOD = TIMER2_LOW_STATE;
        }
        else {
            __LEDstate &= TMP03_LOW_MASK;
//Output current simulated TMP03 signal
           WriteOutputASM(__LEDstate);
//Prepare for next simulation state
            *pTIMER2_PERIOD = TIMER2_HIGH_STATE;
        }
}
TEST(MeasureTemperature, AUTOMATEDTEST) {
        __SaveUserRegAndReset ( );
        temperatureValue4 value;
//Estabilsh the simulation environment using the signal from the
//pulse-width modulated Timer2 signal to mimic the TMP03 sensor signal
        register_handler(ik_ivg11, SimulateTMP03_ISR);
        ActivateOutputASM( );
        ResetPFForInputASM( );
//Test may hang if correct functionality not present
//Test environment will hang if output is not connected to input
//prior to test
//Put out status message
        puts("Testing MeasureTemperature( )\n");
        for (int count = 0; count < 4; count++) {
             __SetUpTimer2(TIMER2_LOW_STATE);
             __ActivateTimer2Interrupt( );
             value = MeasureTemperature(FAHRENHEIT);
             __DeactivateTimer2Interrupts( );
//Expected result is within 2 of 272
             SHORTS_CHECK(abs(value- 272), <=, 2);
        }
        __RestoreUserReg( );
}
```

tion is slated for a later release.

The second approach we attempted was to fake the input signals by writing known values to the FIO_FLAG_DPF input register and pretend that the signals came from an outside device. However, the FIO_FLAG_D I/O register appears to be like many I/O registers on other processors. If you write a value to a pin that's configured as an input, the written value is ignored. We still think we might be able to get around this limitation, but it won't happen in the immediate future. A different approach is required for simulating the expected external signals.

Listings 5 and 6 show two different approaches to this problem. The test in Listing 5 involves manually setting up a known input hardware configuration and then testing for the input. It's straightforward, but this approach completely defeats the automated testing procedure that's a major feature of the TDD environment. Manual set up is just too inconvenient to be practical.

An automated test is programmed

BitScope
Pocket Analyzer

USB Oscilloscope & Logic Analyzer

8 Channel 40MS/s Logic Analyzer
Capture digital signals down to 25nS with arbitrary trigger patterns.

3 Input 100MHz Analog DSO
Classic Analog Scope using a standard x1/x10 BNC probe. Additional inputs on the POD for dual channel operation.

8 + 1 Mixed Signal Scope
True MSO to capture an analog waveform time-synchronized with an 8 channel logic pattern triggered from any source.

Real-Time Spectrum Analyzer
See the spectrum and waveform of analog signals simultaneously and in real-time

Waveform Generator
Load up to 32K arbitrary waveform and replay via the onboard DAC (10MS/s) or a digital pattern from the POD (40MS/s)

The new generation Scope for the age of microelectronics.

Turn your PC or NoteBook into a powerful Scope and Logic Analyzer!

See inside your circuit in the analog and digital domains at the same time to make tracking down those elusive real-time bugs much easier.

Pocket Analyzer combines a high speed sample-synchronized storage scope and logic analyzer with a programmable waveform and logic pattern generator. Also included is an integrated real-time spectrum analyzer and powered "Smart POD" expansion interface so you've got all bases covered!

About the same size and weight as a Pocket PC, this USB powered BitScope needs no bulky accessories. It's the perfect low cost "go anywhere" test and debug solution.

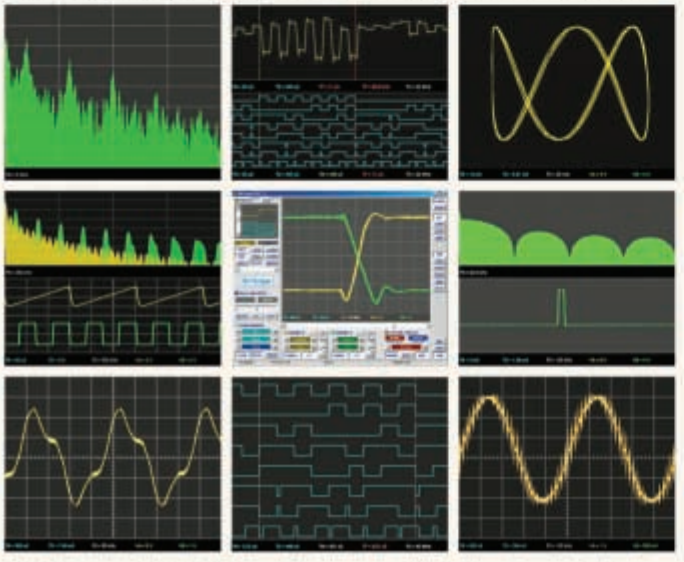
Standard 1M/20pF BNC Input
200uV-20V/div with x10 probe
S/W select AC/DC coupling
S/W select 50ohm termination
Arbitrary Waveform Generator

BitScope "Smart POD" Connector
8 logic channels, 2 analog channels
Dual channel capture from POD A/B
Async serial I/O for external control
Logic Pattern generator 32K 40MS/s

BUS Powered USB 2.0 Device
Single USB cable to your PC
Compressed data transmission
Simple ASCII control protocol
BitScope Scripting Language

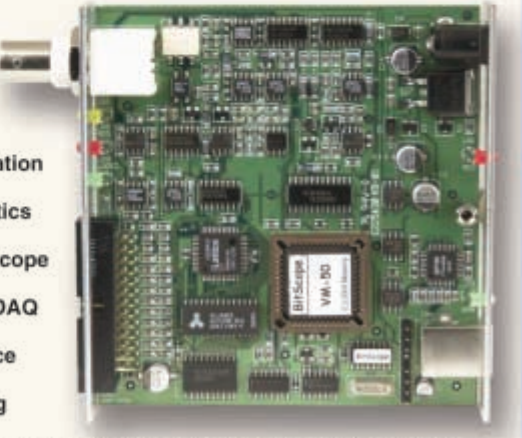
External/Passthru Power Supply
Auto senses an external supply - removes power load from USB for use with unpowered hubs. Supplies up to 500mA via POD

BitScope and your PC provide an array of Virtual Instruments



BitScope DSO 1.2 software for Windows and Linux

- R&D
- Education
- Robotics
- Lab Scope
- Fast DAQ
- Service
- Debug



BitScope Pocket Analyzer uses highly integrated Surface Mount technology to provide functionality you would expect from scopes many times the size and price. Its programmable Virtual Machine architecture means new functionality can be added via software. For custom Data Acquisition, export directly to your spreadsheet.

www.bitscope.com

in Listing 6. The output lines used to control the LED display on the Blackfin evaluation board are looped back to the PF input lines. A background interrupt task involving one of the processor's timers is used to mimic the TMP03 temperature sensor signal. A self-test was performed in this situation using the LED and PF lines on one board. However, for more complex tests, it would be practical to store the test code in the flash memory of a second Blackfin board. The second board could be used to generate a variety of video, audio, and logic-level tests in response to requests from the TDD package on the first board transmitted over the high-speed serial port (SPORT), over low-speed UART, over the high-speed serial parallel interface (SPI), or through the JTAG emulation environment's available multiprocessor capability.

Having an easy approach to generating these hardware tests is the next step in our research project. We're examining the possibility of adapting the automated software component environment (a typical software engineering design process) to generate or make easily available tests that can be used across a variety of platforms and design requirements.

TDD SUGGESTIONS

We described how we added functionality to a TDD environment to make it suitable for hardware and software design in small embedded systems. We demonstrated the system's capabilities through the design of our thermal arm wrestling game.

This system provides an automated test environment for embedded systems capable of handling standard code validation, core timer operations (including interrupts), and basic I/O. The ADSP-BF533 Blackfin microcontroller's numerous internal resources support the hardware side of the TDD. We found that the current state of this TDD tool has made it easier to develop and explain hardware-based projects at the university level.

We are left with some questions though. Is this approach practical for embedded systems with less internal resources than the Blackfin? Can we develop a simple automated signal generator than runs on a second embedded system in response to signals from the system being tested? Is our system unnecessarily complicated? Are the issues we covered simply an indication that our TDD environment has moved out of its diapers and into the real world? We'd love to hear from you. Any suggestions?

Mike Smith has been writing computer-related articles since the early 1970s. He is a professor of electrical and computer engineering at the University of Calgary, Canada. You may contact him at smithmr@ucalgary.ca.

Moreno Bariffi is an international internship student visiting from the University of Applied Science, Fribourg, Switzerland. You may contact him at mbariffi@hotmail.com.

Warren Flaman is an electronics technician at the University of Calgary. You can reach him at wflaman@ucalgary.ca.

Adam Geras is a graduate student at the University of Calgary. You can reach him at ageras@ucalgary.ca.

Lily Huang is a graduate student at the University of Calgary. You may contact her at fhuang@ucalgary.ca.

Andrew Kwan is a computer engineering student at the University of Calgary. You may contact him at akckwan@ucalgary.ca.

Alan Martin is an electrical engineering student at the University of Calgary. You may reach him at ajcmarti@ucalgary.ca.

James Miller is a professor of electrical and computer engineering at the University of Alberta, Canada. You may contact him at jm@ee.ualberta.ca.

RESOURCE

TDD tutorials, The University of Calgary, www.enel.ucalgary.ca/People/Smith/embeddedTDD/.

SOURCE

ADSP-BF533 Blackfin microcontroller, TigerSHARC family, and TMP03
Analog Devices
www.analog.com

by Viraj Bhanage, Prajakta Deshpande, & Praveen Deshpande

Three-Axis Stepper Motor Driver (Part 1)

Design Basics

Technicians in high-tech laser laboratories use optical components such as lenses, mirrors, and filters to direct laser output. A multiple-axis stepper motor controller gives technicians greater control over the components. Special firmware enables remote control from a PC.

We recently visited a high-power laser laboratory. We learned that laser output is generated through a sequential chain of amplifiers. Numerous optical components (e.g. lenses, filters, and mirrors) are placed in the laser's path. The critical components that require frequent adjustment are mounted on a motorized precision x-y platform.

Our assignment was to find a new way to control the 2-D position of the optical components (typical resolution is 10 µm). In a few cases, however, a third motor (z-axis) was also used to control the height and angle of rotation of optics and targets. Because there were so many optical components to manage, we wanted to create a remote control industrial workstation in the facility's laser control room. In addition, we wanted the lab technicians to be able to store each component's x-y-z absolute position in a database.

After studying all of the lab's needs, we developed a general-purpose, three-axis stepper motor driver (see Photo 1). We also wrote firmware that allows you to configure and remotely control one or more drives from a PC. The firmware gained in power as the drive was used for more applications. In the first part of this series, we'll focus on the hardware.

MOTOR DRIVING

Stepper motors are widely used as digital actuators for position control applications. Their shafts rotate in an angular fashion whenever a current pulse is sequentially applied to their windings. Available in a variety of shapes and sizes, they're classified according to their number of stator phases. Two-phase motors are the most common, although three- and five-phase motors provide higher stepping resolution.

In terms of drive electronics, the number of electronic switches required to achieve motion may classify stepper motors. You can have a unipolar drive requiring two switches per phase or a bipolar drive with four switches per phase. The latter is more popular because it yields more torque than the unipolar scheme. Most motors provide all the winding leads externally. This gives you the flexibility to connect them in either a unipolar or bipolar configuration.

A stepper motor's dynamics are controlled by its time constant, L/R, where *L* is the motor winding inductance and *R* is the total resistance in the winding current path. The stepping speed obtained is mainly governed by the slew rate of the winding current in a stepper motor.

Photo 1—*Take a look at the driver in the 42T, 3U Euro case. If the motors rotate continuously, you must have proper ventilation to cool the L298N IC already mounted on the heatsink.*

At lower step rates, the current easily slews to the peak value (I_{RATED}) before a direction change. As motor speed increases, the current won't reach the full value because the time between the current direction change is shorter. Therefore, to achieve the same torque at a higher speed, you should increase the rate of rise of the winding current. To do so, increase the supply voltage for the motor (V_{MOTOR}) well above the product of the motor current (I_{RATED}) and winding resistance (R). The winding current will now peak to V_{MOTOR}/R. Decreasing the L/R time constant of the winding and increasing the voltage applied across the winding increases the winding current slew rate.

To achieve a desired velocity profile, a motor and drive combination must generate enough torque to accelerate the load inertia at the desired rates and drive the load torque at desired speeds. The size of the bipolar motor generally dictates its low-speed torque. The ability of the driver to force current through the windings of the motor dictates the high-speed torque.

You can drive a bipolar stepper motor at its rated voltage using an H-Bridge such as an L298 or one implemented discretely using power transistors. Because the motor winding resistance is the only current-limiting element, you can't allow the V_{CC} of the H-Bridge to exceed the motor's rated voltage. This puts an upper limit on the motor current slew rate as the

time constant is fixed. To overcome this limitation, add a series power resistor to reduce the L/R time constant of the winding and increase the motor supply voltage beyond the rated value.

This scheme generates an L/R drive. One disadvantage to this approach is significant power dissipation in the series resistance that affects the drive's efficiency. Another approach involves using a chopper drive by applying V_{CC} (V_{RATED}) with a feedback driver switching the H-Bridge to hold the winding current at the rated value. A low-value resistance in the H-Bridge's ground lead converts the winding current into proportional feedback voltage. This feedback voltage is compared to the reference voltage in a control loop.

When the feedback voltage is less than the reference voltage, full supply voltage is applied across the winding. When the feedback voltage is equal to the reference voltage, the winding is short circuited to reduce the current. This chopping action limits the value of peak current to a level determined by the reference voltage, and thereby improves the drive efficiency to better than 75%. A single IC (e.g., L297) contains all the electronics needed to realize a chopper drive.

MCU SUPERVISOR

In principle, you can drive a stepper motor via push buttons that provide manually controlled clock and direction signals. Adding an MCU gives you preprogrammed precision motion control at the click of a button. A low-cost, 8-bit microcontroller with enough integrated memory and peripheral devices will keep the project within our budget. Choosing the right development tools for the code is equally as important.

We chose Philips's P89C51RD2 for a number of reasons. Most importantly, it has 64-KB downloadable flash memory and 1-KB data on-chip memory. This enables you to build a single-chip application without adding external memory devices other than the nonvolatile storage. The microcontroller also has on-chip boot ROM, which enables you to download the firmware code via an RS-232 serial interface to the 64-KB flash memory. This eliminates the need for a resident monitor, and it simplifies the overall development process. No extra debugging hardware! The microcontroller has all the necessary blocks available for generating the clock and direction signals that can drive multiple stepper motors without much CPU intervention. This frees up the CPU to perform other functions such as data acquisition and control.

For driving the motors, you need to consider two parameters: motor speed and direction. The former is proportional to the rate at which clock pulses are fed to the L279 (i.e., the frequency at which the square wave must be controlled to maintain a constant motor speed). The motor speed is typically specified in terms of revolutions per minute (RPM). If you know the motor's steps per revolution specification, you can compute the number of steps to fire per minute, which in turn gives you the required pulse repetition rate or frequency.

You can generate a square wave by toggling a port line in the software. But this requires the CPU's full attention. The requirement of driving two or more motors simultaneously and independently complicates the situation. The P89C51RD2 has five programmable counter array (PCA) channels. This on-chip peripheral is similar in function to the popular Intel 8254 programmable timer IC.

The PCA provides more timing capabilities with less CPU intervention than standard timer/counters. Its advantages include reduced software overhead and improved accuracy. The PCA consists of a dedicated timer/counter that serves as the time base for an array of five compare/capture modules. PCA timer modules are all 16 bits wide. If an external event is associated with a module, the function is shared with the corresponding port 1 pin. If the module is not using the port pin, the pin still can be used for standard I/O. Each of the five modules can be programmed in any one of several modes: Rising and Falling Edge Capture, Software Timer, High-Speed Output (HSO), Watchdog Timer, and Pulse-Width Modulator (PWM).

The timer/counter for the PCA is a free-running 16-bit timer consisting of registers CH and CL, which are the high and low bytes of the count values. It's the only timer that can service the PCA. The clock input can be selected from any one of four sources. Special-function register CMOD contains the count pulse select bits (CPS1 and CPS0) that specify the PCA timer input. This register also contains the ECF bit, which enables an interrupt when the counter overflows. In addition, you have the option of turning off the PCA timer in Idle mode by setting the counter idle bit (CIDL) to reduce power consumption. Each of the five compare/capture modules has a mode register called CCAPMn (n, e, 0,1,2,3, or 4) to select

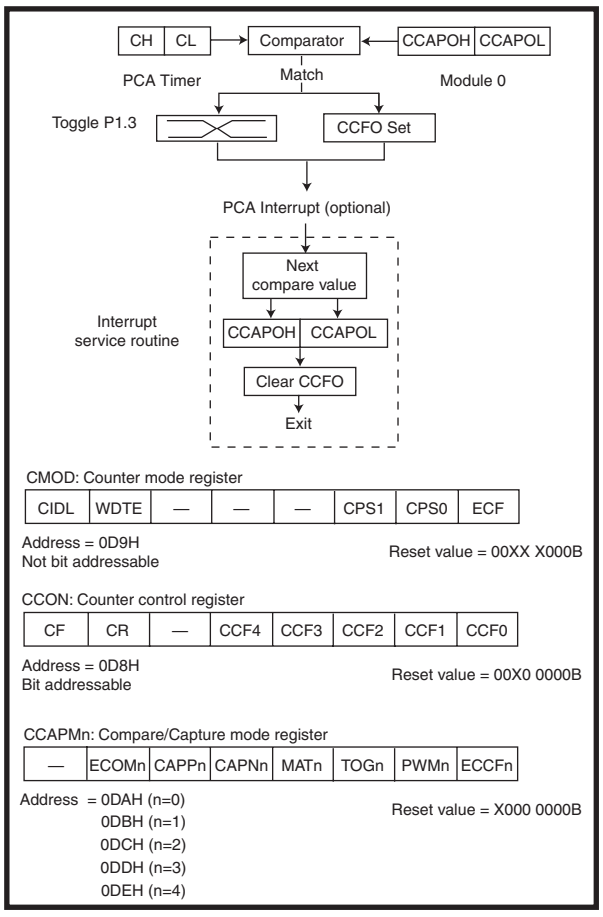


Figure 1—*The PCA hardware is extremely useful for timing generation and monitoring independent of the CPU. You can use the PCA to run the motor in the background using its HSO mode.*

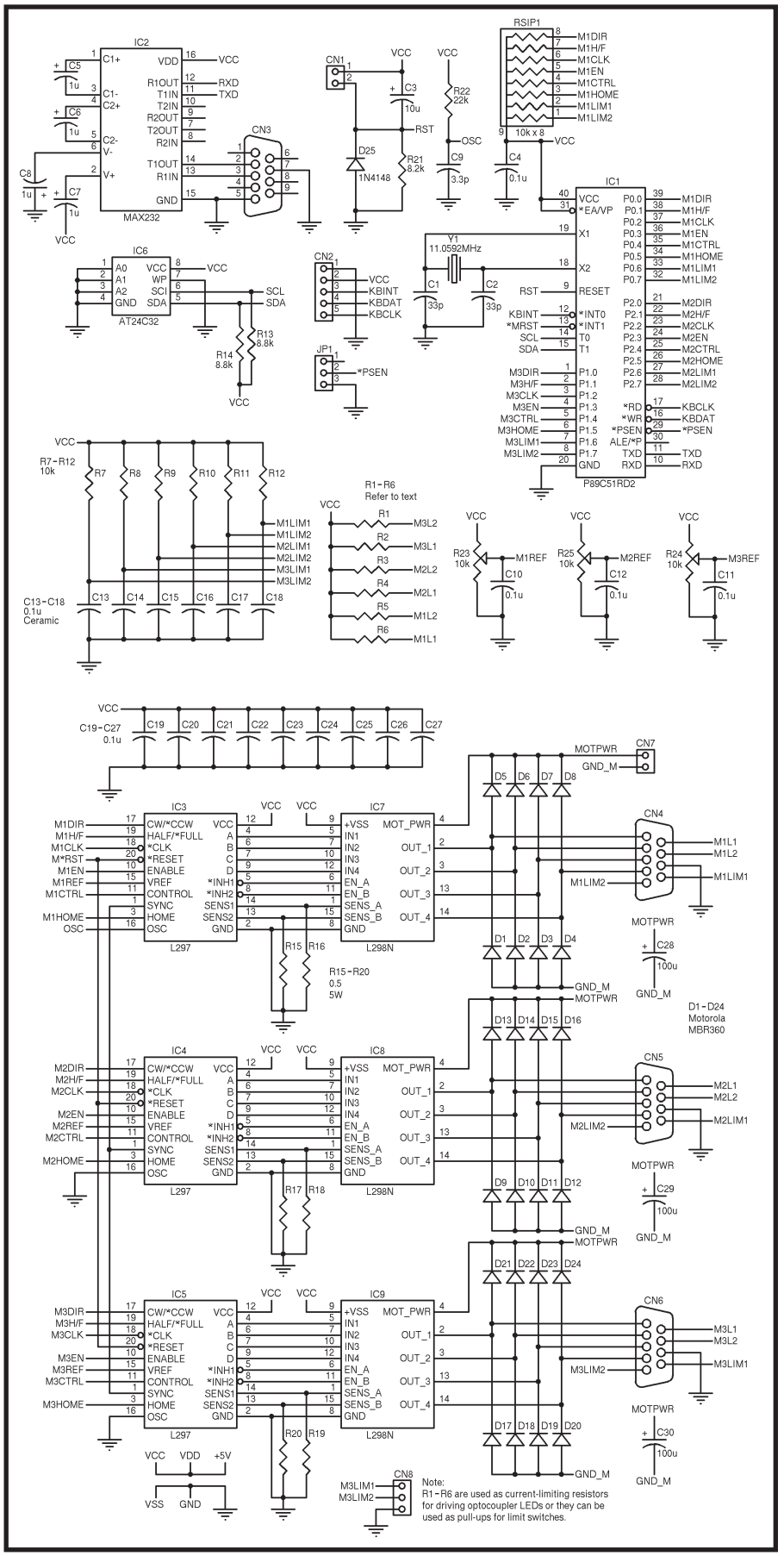


Figure 2—*The three-axis stepper motor driver is implemented as a compact single-board driver with on-board terminations emerging from the back of a 42T, 3U Euro instrument case.*

the function it will perform. The ECCFn bit enables an interrupt to occur when a module's event flag is set.

HSO mode toggles a port pin when a match occurs between the PCA timer and the preloaded value in the compare registers (see Figure 1, page 69). This provides more accuracy than the process of toggling pins in software because the toggle occurs before it branches to an interrupt (i.e., interrupt latency will not affect the accuracy of the output). In fact, the interrupt is optional. Only if you want to change the time for the next toggle is it necessary to update the compare registers. Otherwise, the next toggle will occur when the PCA timer rolls over and matches the last compare value.

You can use HSO mode for programmable square wave generation. You need to load an appropriate 16-bit value into the 16-bit PCA module register and configure it for HSO mode. It sets the CCF0 flag that generates a PCA interrupt. In the interrupt service routine, you can reload the module register CCAPMn (next value to compare) to make additional comparisons. This way you'll generate a programmable square wave signal that you can use to drive a stepper motor along with a sequencer. Because the chosen PCA module handles most of the driving, you can drive the motor in the background. This also enables you to assign a PCA module to each stepper motor sequencer. You can drive all of them in the background simultaneously and independently.

You can also configure the PCA hardware module to count pulses. Thus, you can count an encoder's output pulses to know how far the motor has reached in a closed-loop system.

DRIVER ELECTRONICS

Figure 2 shows a three-axis stepper motor driver. The P89C51RD2 (IC1), which has a built-in oscillator, requires only an external crystal (Y1) and two capacitors (C1 and C2). Although we could have used any crystal frequency up to 20 MHz, we chose 11.0592 MHz because it provides many standard data rates for communication with the PC.

Capacitor C3, R21, and diode D25 form a power-on reset circuit. You may connect an external push button reset switch with CN1. A three-wire RS-

Everywhere you imagine. RENESAS

MCU Solutions Matched To Your Design Imagination

RENESAS IS THE #1 MCU SUPPLIER IN THE WORLD. **www.renesas.com**

M16C Advantages

- Advanced 16-bit and 32-bit CISC cores that perform at up to 32 Dhrystone 1.1 MIPS
- Banked-register architecture optimized for fast context switching
- Wide range of high-performance flash memory: 8 KB to 512 KB
- Best-in-class EMI/EMS noise immunity
- Designed to be pin- and function-compatible between families and packages
- Extensive peripheral integration: everything from POR and LVD to CRC counters

Low-cost Renesas Toolchain

- Fully integrated suite of hardware and software tools
- Highly-optimized C compiler and assembler — FREE (64K limited version): Download latest version from web (www.renesas.com/skp)
- Low-cost Starter Kit (target board, USB debugger, complete toolchain), starting at $49
- Easy online access to application notes, example code, software development tools

Get your product to market faster using Renesas Interactive!

Renesas INTERACTIVE

Evaluate, test, and learn about Renesas products in an interactive online environment.

www.renesasinteractive.com

SKP	Device	C Compiler, Assembler, IDE	Cost
SKP8CMINI13	R5F21134FP	FREE (64K limited version)	$49
SKP8CMINI17	R5F21174FP		$49
SKP16C26A	M30260F8AGP		$49
SKP16C28	M30280FAHP		$49
SKP16C62P	M30626FHPFP		$49
SKP32C83	M30835FJGP		$119
SKP32C84	M30845FJGP		$49

Special pricing is in US dollars from Renesas authorized North American distributors only, and is subject to change.

Popular Devices

Series/ Group	Part Number	CPU (Bits)	Package	Freq. (MHz)	Dhrystone 1.1 MIPS (Max)	Program Flash (KB)	Data Flash (KB) (10K Rewrites)	SRAM (Bytes)	Serial I/O Channels	I²C	I/O	A/D 10-bit	Ring Osc.	LVD POR	Timers	Price *
R8C/17	R5F21172SP	16	20 Pin SSOP	20	9	8	2	512	1	1	15	4	X	X	3+WDT	$1.90*
R8C/13	R5F21134FP	16	32 Pin LQFP	20	9	16	4	1K	2	–	24	12	X	X	4+WDT	$2.38*
M16C/26A	M30260F8AGP	16	48 Pin LQFP	20	13	64	4	2K	3	1	39	12	X	X	8+WDT	$4.76*
M16C/28	M30281FAHP	16	64 Pin LQFP	20	13	96	4	8K	4	1	55	13	X	X	9+WDT	$5.90*
M16C/62P	M30624FGPGP	16	100 Pin QFP	24	16	256	4	20K	5	3	87	26	X	X	11+WDT	$8.25*
M16C/62P	M30626FHPGP	16	100 Pin QFP	24	16	384	4	31K	5	3	87	26	X	X	11+WDT	$10.05*
M32C/84	M30845FJGP	16	144 Pin LQFP	32	32	512	–	24K	5	5	124	34	X	X	11+WDT	$16.88*

* Suggested 10,000 piece price. Please contact your local Renesas distributor for volume pricing.

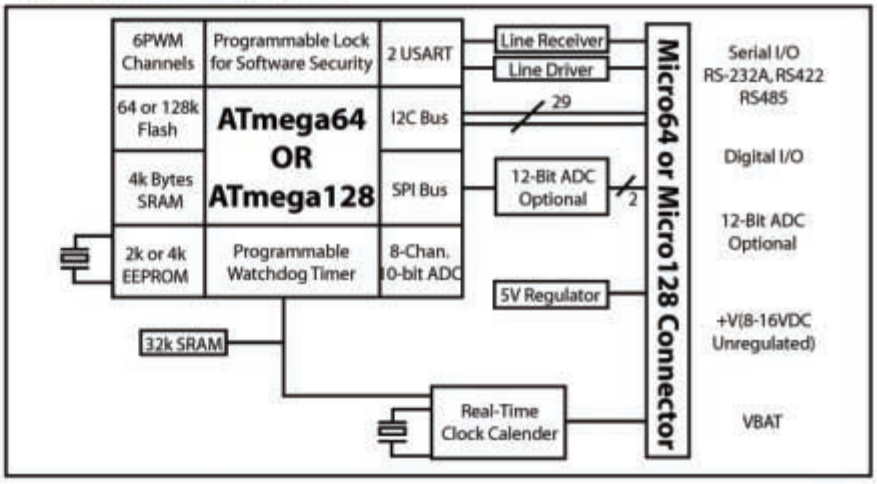
FREE Micro64! to the first 50 people to respond to this ad. For more information visit www.micromint.com/promo/ or call 800-635-3355

ATMEL AVR®
Based Controller

- Small size - 1.5" x 2.1" x 0.52
- Dual powered - operates on +5V or 6.5-20V at 55 mA (typical)
- Program and Data Memories
  - 64k or 128k Bytes of In-System Reprogrammable Flash with 10,000 Write/Erase Cycles
  - 2 or 4 K Bytes EEPROM with 100,000 Write/Erase Cycles
  - 36K Bytes SRAM
  - Programming lock for Software Security
- Peripheral Features

Starting at $119 - Single Qty

- Optional 2-channel 12-bit ADC
- 8-channel 10-bit ADC
- 8 Single-ended Channels
- 7 Differential Channels
- 2 Differential Channels with Programmable Gain (1x, 10x, 200x)
- Byte-oriented Two-wire Serial Interface (I2C)
- Dual Programmable Serial USARTs
- 1 TTL
- 1 RS-232A, RS-422, or RS485
- Master/Slave SPI Serial Interface
- Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
- Two Expanded 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Two 8-bit PWM Channels
  6 PWM Channels with Programmable Resolution from 1 to 16 Bits
- Programmable Watchdog Timer with On-chip Oscillator
- 29 Digital I/O that can sink or source 20 mA



Data sheets, Online Ordering and a Complete Catalog of Products at...

www.micromint.com

Delivering Embedded Controllers for 24 Years

Reduce Your Time to Market.

Minimize your time from conception to production by utilizing one of Micromint's market-proven controllers. Whether your concerns are digital or analog, inputs or outputs, Micromint has a product to fit your needs. Order quantities of one to thousands. Custom design and configurations are available.

With over 500,000 controllers in the marketplace, Micromint has been providing innovative, turn-key solutions to the OEM market for 24 years—from design through production, as well as packaging and shipping the final product. Our broad line of embedded controllers and turn-key solutions can turn your imagination into reality.

Visit our website @ www.micromint.com to see our complete line of OEM Solutions.

Micromint

115 Timberlachen Circle | Lake Mary, FL 32746 | 800-635-3355 | 407-262-0066 | Fax 407-262-0069

Looking for cutting edge products to solve your toughest design and measurement problems?

Temperature/Humidity Logging

RH-02/PT104 - Temperature and Humidity Logger for PC. High resolution and accuracy. No power supply required.
from $256/600!

Micro Controller

WMD Wireless Micro Board uses fastest Atmel Mega - 16MIPs for wireless controllers. Great for mobile robots and data logging applications.
from $239!

USB to 24 x I/O

USBI/O24 - 24 latched I/O lines self-powered from your PC's USB port. Ubicom SX52 allows individual pin I/O programming. I/O control via PCs USB port.
from $69!

CANbus Cards

CANbus - Starter Packs and cards for almost any board format & OS. CAN/Ethernet bridges, industrial automation solutions from Janz AG as used by top companies all over the world.

Alfat IC

Alfat - Easily add MB storage to your new design - full FAT file system stack on a powerful 32-bit processor. Send serial/SPI data to ALFAT and it takes care of the rest. **only $29.95!**

Mini Web Server

PicoServ™ Miniature Web Server allows remote I/O and servicing. Collect analog & digital data over a web connection. Remote sensing/control, SCADA, building control, etc. **from $189**

easyRADIO

easyRADIO™ ER900TRS Transceiver Modules make wireless data transmission simple for USA and Europe! Add wireless capability to your project today! **from $45 (100pr)**

RS232<>422/485

K2 9p-9p self-pwrd RS-422/485
K3 9p-9p isolated RS-422/485
K3-232 9p-9p isolated RS232
K232-ISOL 25p -25p RS232
KD485-STD DINrail - isolated
KD485-PROG programmable!

I2C for PCs

PCI93LV: industry-standard I2C card for PCs. **WINI2C/PCI** software gives windows-interface to develop and debug I2C bus systems. **UCA93LV** is new USB version. ***NOW!*** - transparently monitor at 400kHz! **only $499!**

USB in one IC!

FT232BM Single chip USB-232 solution with all Windows /Mac/Linux drivers. No programming/USB knowledge required! Easiest answer to update RS232 devices! **only $2.55 (10k)**

Logging Module

TDS2020F makes a custom CF card A/D data logger or controller - *quickly!* High-level software completes projects in *hours*. Store **GPS/CANbus/A or D** data to card. **only $199 (50 qty)**

PC Scope Adapters

ADC-212/100 turns your PC into a great high-speed scope. Sampling to 100MS/s at 12-bit res'n. FREE s/w turns PC into powerful 2-ch scope, DVM & spectrum analyzer. **$95 - $1145**

BASIC modules

Tigers - tiny multitasking BASIC modules for quick projects. >100,000 instr's/s! **iCOM200** ready-made controller with LCD and keypad. **Touch240** controller - with touchpad and LCD display.

Crystals / Oscillators

Euroquartz - Europe's largest mfr of quartz crystals, osc's, filters and frequency products. Custom filters, high rel. rad tol. osc's. Novel **EQ-HM** spread-spectrum osc's reduce EMI problems

SM PCB Adapters

OM-Adapt SM miniboards have two footprints on either side. Now use your ultra fine pitch SMD components with more useful 0.1" inline spaced holes. One-to-one pinouts. **$6.99**

Morph-IC

Morph-IC combines FT2232C and Altera ACEX FPGA - ideal for USB-upgradable products. Uses MPSSE to program / reprogram the FPGA over USB in mSs! **only $125!**

4/8 x RS232

4COM - add 4 or 8 COMports to your PC with this unit that plugs into your PC's USB port. 4COMi version gives 4 X RS 422/485 ports with h/s.
USB-4COM **from $105!**

USB Logic Analyzer

ANT8 Matchbox-sized USB-powered logic analyzer. Sample 8 channels at up to 500 million samples-per-sec. View traces on PC. Print or save for later review! (16-ch. ver. too!) **$222!**

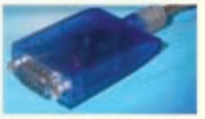
USB <> RS485

USB-COMi (non-isolated) or **USB-COMiSI** (isolated) offer self-powered USB to RS485 conversion with baud rates 184bps - 3Mbps. PC thinks it's talking to a COMport! **only $48/$85!**

FT2232C

FT2232C - latest version of FTDIs easy-use USB-serial ic - combines two serial or parallel devices combined in one ic. = (FT232BM + FT245BM)
only $4.29 (10k)

USB-Serial Adapter



Serial port for laptops without one! **US232B** instantly updates older RS232 products to USB! PC thinks it's using COMport but uses the USB connection.
only $39 (1) or $29 (100)

NEW Swordfish

Hand Held PROBE

PS40M10 - unique hand-held PC Scope: 5MHz scope, 20kS/s data logger, FFT spectrum analyzer, voltmeter and frequency meter. 10-bit ADC with a true sampling rate of up to 40MS/s.
Only $275

DrDAQ

DrDAQ - is a PC adapter with sensors for light, PH, volts and temp. Great for science fairs! Supplied with ready-to-run software and lots of physics/chem experiments. **from $101.48!**

Power Supply

- 1.5 to 30V d.c. output
- 0 to 1A output current
- 100 to 240 V a.c. input
- Over-volt/current & short circuit protection

PSU-130 **$79.95!**

CAN-USB

CAN-USB - intelligent CAN connection from PC's USB port. Provides plug'n'play opto-isolated SJA1000 CAN controller with 128kB of on-board Flash and SRAM memory. **from $253.65!**

USB Bus Analyzers

Tracker 110/Explorer 200 high-performance, economical USB 1.1/2.0 protocol analyzers. Perfect for troubleshooting USB, optimizing data flow and USB training! **only $999/$2999!**

NEW STINGRAY

PC SCOPE ADAPTOR

DS1M12- 1 MS/s sampling 12-bit 2-ch USB scope adapter for PC. Looks like a "Digital Scope" on your PC screen! Great for laptops, and servicing.
Only $220

"If we can't answer your questions immediately, we research it and get a reply within 24 hours...Guaranteed!"
- Alan Lowne, President

USB Temp Logger

EL-USB-1 - Standalone USB temperature data logger (25 to +80°C). Stores 16K Li-backed readings. Log data for >1 year!
EL-USB-1 **from $101.48!**

Data Modem

Data Modem - USB-powered radio transceiver boards, add wireless RS232 to your device! Get remote control for your product from PC USB port!
only $125!

Tinytag Instrument

Tinytag Instrument - A range of high resolution data loggers designed for the monitoring of temperature across a wide range. High accuracy (12 bit).
TGI-3080 **only $175!**

USB to I2C

USB I2C/IO - provides a simple "drop-in" solution for connecting your PC to 90kHz I2C + 20 I/O lines. Free software. Use multiple boards for more I2C/IO.
USB I2C/IO **only $79!**

FREE COFFEE

Call 1-888-7SAELIG
Mention Offer# SBWNV

to get a free STARBUCKS Card with your order!

(Not available with any other offers!)

SAELIG MARKETING: Attention Midnight Engineers! We specialize in creating a market for unique products. Have you designed a great products but don't want to be bothered with product fulfillment (advertising, answering calls, packing shipping, etc.)? Let us sell your products for you! Contact np@saelig.com

Saelig Co. Inc.

p: 585-385-1750 f: 585-385-1768
www.saelig.com/ad/cc405.htm • info@saelig.com

Saelig Co. *brings to USA unique, easy-to-use control and instrumentation products from overseas. Customers include: Intel, Philips, NEC, Kodak, Nokia, US Military, Microsoft, Dell, Xerox, Universities, T.I., Harris, Sony, J&J, Thomson, Sandisk, General Dynamics, H-P/Compaq, etc.*

CIRCUIT BOARD solutions for your robotic needs

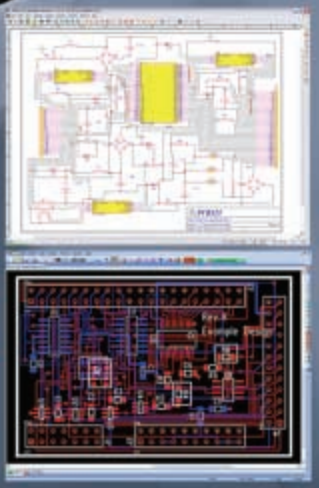
Need FREE design software?

FREE Download the new version at www.pcb123.com/cc

PCB123 VERSION 2.0

PCB123 Provides the Complete Circuit Board Solution, Design Through Order

- Order up to 100-pcs with Quick Turn Around
- New libraries contain over a 1000 common devices (PIC MCU, AVR, Flash, OTP, EPROM and much more)
- DigiKey, Jameco & Mouser part numbers
- Templates for PC104, PC104+, Rabbit, PCI & more



PCB123 13626 S Freeman Rd. Mulino, OR 97042 USA
Phone: (800) 228-8198 x236 / Fax: (503) 829-6657

DOWNLOAD THE NEW VERSION
www.pcb123.com/cc

Have design files?

YOUR TARGET FOR PROTOTYPE CIRCUIT BOARDS

LEADING INTERNET SUPPLIER

PCBexpress

PCBexpress.com

PROTOTYPE CIRCUIT BOARDS

Successfully selling online since 1997.
Experience you can trust! Try us out today at:
www.PCBexpress.com/cc

Enter the link for a special Circuit Cellar discount!

232 serial interface on a nine-pin D-type male connector (CN3) is implemented by way of a MAX232 (IC2) transceiver.

The stepper motor sequencer IC L297 is the heart of the driver. In the past, we've used three similar devices (IC3, IC4, and IC5) for the x, y, and z-axis motors, respectively. An external resistor (R22), an external capacitor (C9), and an on-chip oscillator circuit (IC3) generate a common clock signal for all the L297 devices. Each L297 device generates four logic output signals and two enable signals for driving the L298 H-Bridge device in Chopper mode.

We assigned MCU (IC1) ports (P0, P2, and P3) for each motor axis. The ports are bit addressable, so the individual port pins are used to drive the different motor control signals (e.g., CLK, ENABLE, HALF/FULL STEP CONTROL, etc.) of the corresponding L297 IC.

The L298 driver IC (IC7, IC8, and IC9) has a dual H-Bridge for driving a two-phase stepper motor in bipolar configuration. The D1 through D28 diodes are 2-A, freewheeling diodes that provide an alternative path for the inductive decaying current when the power to the motor winding is suddenly removed.

R15/16, R17/18, and R19/20 are high-power sense resistor pairs used for sensing the motor winding current in each phase for each motor. Each of the three potentiometers R23 through R25 provides necessary reference voltage for IC L297 to generate the required chopper output in a closed-loop control. You should adjust these potentiometers for the appropriate reference voltage for each motor, and take into account the values of the sense resistor and required motor winding current.

Connectors CN4, CN5, and CN6 have L298 outputs where you can connect the x, y, and z motors along with optional optical/mechanical end-limit detection logic. We used a Sanyo Denki 200-step-per-revolution, 2-A stepper motor for each motor axis. You may need to detect the end limits while driving a linear transnational stage coupled to a stepper motor.

Resistors R1 through R6 are current-limiting resistors for driving the LED inside an optical interrupter used as an end-limit detector. Alternatively, they act as pull-up resistors if you have mechanical end-limit switches. An RC circuit (R7 and C13) debounces the mechanical limit switches.

The AT24C01 (IC6) is a 128-byte serial EEPROM into which the firmware stores the configuration information. Because IC1 doesn't have a hardware I^2C serial port, the interface is implemented in software. You can make motor power supply connections with the CN7 connector. You can use any unregulated power supply between 12 and 35 V for the motor. We prefer an SMPS as an efficient, compact alternative to the linear power supply.

We used the Flash Magic freeware utility to program IC1 through the RS-232 serial interface. You must place JP1 in the 2-3 position and push the reset switch. This invokes an on-chip boot ROM on IC1. Flash Magic handles the protocol for downloading the Intel hex format executable file to the on-chip flash memory. It also enables you to program the security bits that protect your code! The utility supports in-circuit programming (ICP) and in-system programming (ISP) for most Philips MCUs.

MOTOR CONTROL

You can drive the motors with the aforementioned hardware. You need a user interface to control the motors from a front panel. Most of the P89C51RD2's hardware resources are occupied in motor control circuit. You're left with only an RS-232 serial port and a few I/O port lines. We wanted to be able to control the motors remotely (using an RS-232 or RS-485 serial interface), so we implemented a complete command set for PC configuration and control.

Listing 1 (page 76) shows the rudimentary code we used for driving each stepper motor by toggling the motor's corresponding clock signal. You may use this code to test the driver. You can also test the half/full-step operation by setting the corresponding pin to 0 or 1. You can change the motor's speed by adjusting the delay between adjacent clock pulses. If the motor is driving a linear transitional stage, you can detect the limiting position by monitoring two switches often installed at the endpoints.

You might be unwilling to devote a PC for the alignment job. So, we decided to use the free I/O lines to interface a tiny 20-pin (AT89C2051) slave microcontroller that acts as a 4 × 20 LCD and keypad controller interface. To simplify the system, we assigned three keys (move up, move down, and go home in a column) to each motor. You can invoke the system configuration menu and adjust the various motor parameters using the plus and minus keys.

Figure 3 (page 76) is a schematic of the keyboard and display interface. The LCD has a 4- to 8-bit data bus and three control lines. The keyboard is arranged in a 3 × 4 matrix. To conserve the I/O lines, the 4-bit LCD data bus is shared with return lines of the keypad. The on-chip UART on the slave microcontroller provides the necessary serial communication link for full-duplex communication with the host P89C51RD2.

The P89C51RD2 has only one UART channel that's reserved for the RS-232 remote control interface. The other UART channel required for communication with the keypad and display controller (89C2051) is implemented in software. Note that the software implementation has a drawback. You can't receive the serial data in the background. For this reason, the serial communication protocol with P89C2051 is implemented in a command/respond fashion.

The P89C2051 doesn't speak up by itself; it responds only when it's polled by the host CPU. One I/O line is reserved for notifying the host CPU when a valid key is detected. This frees the host CPU from frequent polling of the P89C2051 for a key. (Polling the slave at 9,600 bps requires an exchange of at least 1 byte requiring a minimum of approximately 2-ms CPU time!)

Refer to Listing 2 (page 77) for a taste of the firmware inside the P89C2051. `Timer0` scans the matrix keypad every 10 ms. A state machine provides key detection, debouncing, repeating, and other keypad features. A serial interrupt is generated when a command is received. The code invokes the appropriate command handling function using a function pointer array. You may download the complete code for the P89C2051 slave from the *Circuit Cellar* ftp site.

ROOM FOR IMPROVEMENT

You'll have to make a few changes if you want the driver to be a completely

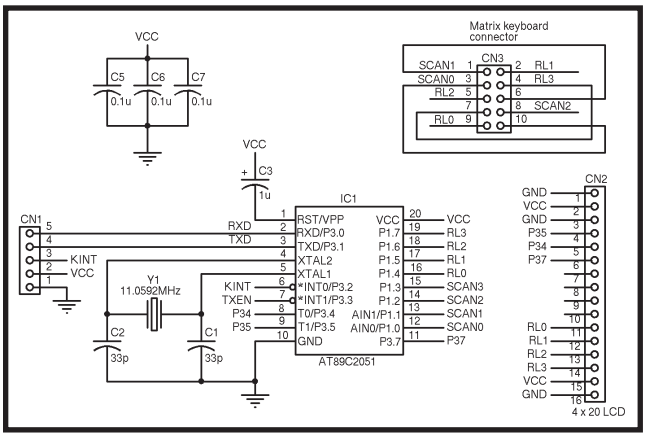


Figure 3—*The 4 x 20 LCD and 4 x 4 matrix keypad controller are implemented with a 20-pin AT89C2051 MCS-51 derivative. The tiny board is mounted on the LCD. It can be hooked to any microcontroller with a three-wire serial interface.*

general-purpose stepping motor controller. The motor current is presently programmable via a potentiometer. You can adjust the voltage on the reference pin of L297 and divide it by the sense resistor value to set the motor current. This requires you to expose the potentiometer from the front panel and insert a screwdriver to set the required voltage.

The board doesn't have an on-board ADC, so you must measure the reference voltage with a voltmeter as you set up. You must open the top panel to do so. If you want to do it remotely with a PC or without opening the box, you can use a DAC to generate the reference voltage. This requires you to add one DAC per motor. There's one more option. You can implement a DAC using a PWM. If you can spare a few PCA channels or a timer, this can be done in software. We didn't do it because we were short on time.

Another significant improvement would be to incorporate microstepping. To do so, you must apply sine and cosine currents to the two windings of the motor. This would require a dual DAC or two PWM channels for generating the sine and cosine voltages. But the L297 device doesn't support separate $V_{REFERENCE}$ input for each motor winding phase, so you can't microstep with it. You must replace the sequencer IC with an L6502 for microstepping and redesign the drive.

Stepper motors are often used in an open loop without feedback. You may have an application in which you need to generate precise motion control. A stepper motor can miss a step if it isn't able to provide enough torque at a certain point because of excessive friction. This results in a mismatch between the actual and computed position. The solution is to incorporate a rotary encoder for position feedback. The encoder would emit a number of pulses corresponding to the angular incremented movement when coupled directly to the shaft. By counting the encoder pulses, you can obtain information about the actual angular movement. You can derive a signal for the direction of the rotation from the encoder output. You can count the encoder pulses using one of the PCA channels programmed to detect rising and falling edges.

A disadvantage of using a PCA counter is that it doesn't support up and down counting; therefore, if the motor changes direction, the PCA counter would still count up and the actual position would be lost. Special devices are available for encoder-to-microprocessor interfacing (e.g., LS7166). Such devices incorporate a filter circuit at their inputs to eliminate the effect of noise.

FIRMWARE TO COME

Are you ready to build a generic multiple-axis stepper motor controller? Next month, we'll describe the firmware. Until then, happy (step) motoring!

Viraj Bhanage received a bachelor's degree in electronics and telecommunication from Shivaji University,

Listing 1—*Use this code to test stepper motor hardware after connecting the motors and its power supply. You can set the motor's speed as well as select the direction of rotation and full- and half-step operation.*

```
//Code to test stepper motor hardware
#include<reg51f.h>        //Header file for P89C51RD2 CPU
sbit M1Dir = P0^0;        //0 - counter clock wise, 1 - clock wise
sbit M1HalfFull = P0^1;   //0 - Full step , 1 - Half step operation
sbit M1Clock = P0^2; //Clock input
sbit M1En = P0^3;         //Enable/Disable motor1 control
sbit M1Lim1 = P0^6;       //Limit switch1
sbit M1Lim2 = P0^7;       //Limit switch2

#define    DIRCCW     0    //Motor direction is counter clock wise
#define    DIRCW      1    //Motor direction is clock wise
#define    HALFSTEP   1    //Half step
#define    FULLSTEP   0    //Full step

#define    XMOTOR     1    //Define constant to indicate XMOTOR
#define    YMOTOR     2    //Define constant to indicate YMOTOR
#define    ZMOTOR     3    //Define constant to indicate ZMOTOR

#define    MEDIUM_SPEED_DELAY     180  //Delay between clock pulses
//Run 'motor' one step in current direction
char RunMotor(char motor)
{
unsigned char d= MEDIUM_SPEED_DELAY;    //Medium speed is selected
char limit=0;                           //Indicates limit reached
  switch(motor) {
          case XMOTOR:                  //Code for XMOTOR
          {
           if (GetMotorDirection(motor) == DIRCW) {
                   limit = ReadLim2(motor);
           }
           else { //DIRCCW
           limit = ReadLim1(motor);
           }
           if (limit==0) return 0;      //Indicate motor limit
           M1Clock = 1;                 //CLK = 1
           M1Clock = 0;                 //CLK = 0
           while(d--);                  //Wait for specified delay
           M1Clock = 1;                 //CLK = 1
           d = MEDIUM_SPEED_DELAY;      //Wait for a delay
           while(d--);
           return 1;                    //Operation success
          }
```

(Continued)

Listing 1—*Continued.*

```
            break;
    }
}
//Rotate motor for N steps
unsigned long RotateMotor(char motor, unsigned long int NumOfSteps)
{
unsigned long int c;
unsigned int k;

            for (c=0; c < NumOfSteps; c++) {
                        if(RunMotor (motor) == 0) return (c);
            }
}
```

Listing 2—*This code fragment is for the LCD and matrix keypad controller implementation using the P89C2051 microcontroller. With this code, you can connect the keypad and display to any microcontroller with a three-wire serial interface.*

```
#include <reg51.h>
typedef void (*Fn_Pointer)(void);//Makes it easy to invoke req'ed func
Fn_Pointer fn_ptr1[] = {MyStatus, SendKBStatus, SendKey, //Keypad func
    ClrLCDCmd, DspChr, GotoXYCmd, DspStr}; //LCD related functions

bit EvCmd;              //Reading and processing command

void SerComInPtr (void) interrupt 4     //Receive command in serial ISR
{
  if (TI) {             //Interrupt is due to TI flag
       fTxEmpty = 1;    //Store TI status
       TI =0            //Clear TI to avoid further interrupt
       return;          //Transmission section done
  }
//Command byte is received. Go Process the command
       TI = fTxEmpty;   //Restore TI flag to use SendChrw() function
       ES = 0;          //Disable the serial ISR for the moment
       Cmd = RecvChrw();//Read command byte
       EvCmd=1;         //Notify that command is received
}

void  Timer0ISR (void) interrupt 1 //Scan a matrix keypad
{
       TH0 = COUNTHIGH;         //Reload the timer0 16-bit register
       TL0 = COUNTLOW;          //For 10-ms keypad scanning
       TR0=1; //Run the timer0
       Kbd();                   //Execute the keypad state machine
}

code char Sign_OnMsg[] = "Keypad & LCD server";

main()
{
        InitSer51();            //Init serial port 9600,8,N,1
        InitLCD();              //Init the 4 × 20 LCD
        Str2LCD(Sign_OnMsg);    //Notify this controller ready
        GotoXY(0,3);            //Position cursor at (0,3)

        TMOD = 0x21;            //Timer0 (mode1), timer1-serial
        TH0  = COUNTHIGH;
        TL0  = COUNTLOW;        //Init timer for 10 ms
        TR0  = 1;               //Run timer0
        ET0  = 1;               //Enable timer0 Interrupt
        ES   = 1;               //Enable serial Interrupt
        EA   = 1;               //Enable global interrupt

      while(1) {
            if (EvCmd) {        //Is valid command ready?
               if (Cmd < 7) {
                       (*fn_ptr1[Cmd])();//Receive & process command
                       EvCmdServed =1;   //Notify command executed
               }
               ES =1;           //Enable serial interrupt
               EvCmd=0;         //Clear previous command
            }
      }
}
```

Kolhapur, India. He currently designs laser-based instruments. Viraj is interested in laser-based instrumentation for industrial and meteorological applications and embedded systems. You may contact him at viraj@cat.ernet.in.

Prajakta Deshpande holds a B.S. from Sagar University, India and an M.C.A. from Barkatullah Vishwa Vidyalaya, Bhopal, India. She is a lecturer in computer science. Prajakta may reach her at ppd@nettaxi.com.

Praveen Deshpande holds a bachelor's degree in electronics engineering from the Regional Engineering College (REC), Nagpur, India. He is currently working as a senior scientific officer in the field of distributed data acquisition and control. His interests include real-time operating systems and distributed control. You may contact him at ppd@cat.ernet.in.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/177.

RESOURCES

Intel Corp., "8XC51 FA/FB/FC PCA Cookbook," AP-440, February 1990.

———"MCS 51 Microcontroller Family User's Manual," http://developer.intel.com/design/mcs51/manuals/272383.htm.

T. Kenjo, *Stepping Motors and Their Microprocessor Control*, Oxford Clarendon Press, 1985.

Philips Semiconductors, "Comprehensive Product Catalog," 9397 750 11146, vol. 5, April 2003.

STMicroelectronics, "The L297 Stepper Motor Controller," AN470, November 2003.

SOURCES

AT89C2051 Microcontroller
Atmel Corp.
www.atmel.com

P89C51RD2 Microcontroller
Philips Semiconductors
www.semiconductors.philips.com

L297 Stepper motor controller
STMicroelectronics
www.st.com

USB Easy Riders

Tom takes a closer look at USB technology and the process of upgrading applications with USB connectivity. Chips like the CP2102 make the procedure a whole lot easier.

It was way back in September 1996 when I first wrote about the then nascent Universal Serial Bus ("Oh Say Can USB?" *Circuit Cellar*, 74). Now, after 10 years (and more than 1 billion USB devices, according to www.usb.org), it's clear that USB is a boon for PC connectivity and is here to stay.

Reports of the demise of the classic serial and parallel ports are slightly, but only slightly, exaggerated. Printers, keyboards, mice, and all manner of PC-oriented gadgets are on board with USB. Now it's time for the embedded crowd to follow suit.

Porting embedded applications to USB isn't a new concept. There have been more than a few articles on the subject right here in *Circuit Cellar*. (Refer to the resources section of this article for a list of them.) Nevertheless, time and silicon march on, so let's take a look this month at the latest and greatest.

GOING, GOING, GONE?

But let's start with some not so great. What's up with USB On-the-Go? Not much, as far as I can tell.

To refresh your memory, the USB On-the-Go (OTG) initiative was announced to great fanfare way back in 2001. The idea behind OTG has been to evolve USB beyond its PC-centric host/device architecture to a more democratic peer-to-peer device/device nirvana. OTG proponents look forward to the day when your cell phone, PDA, MP3 player, etc. can talk directly to each other without needing a PC in the loop.

Sounds good, but don't hold your breath. If you poke around the 'Net, you'll find all manner of press releases in the years since the announcement. The annual PR ritual invariably includes a prediction that USB OTG-enabled devices will be "on the shelf" by "the end of this year" (i.e., whatever year it was when the hopeful hype was penned). There may be some shelves somewhere with OTG on them, but they aren't the ones at my local electronics emporium.

Sure, OTG makes sense from 50,000'. From that lofty vantage point, oxygen deprivation-induced euphoria tends to make a lot of ideas look good, including the 99% that go nowhere. Only from a much lower altitude do the messy details, and the devils that come with them, become apparent.

Yes, the first OTG-capable chips are peeking out from behind the curtain. Usually, the availability of silicon says a standard's time has arrived. But in OTG's case, I think it just means there's no more avoiding the troubling issues that remain. Chief among these is the fundamental assumption that universal device-to-device connectivity is a good thing. As the old saying goes, "where you stand is where you sit." For instance, consider one example of a problem OTG is supposed to fix. Many PDA users buy full-size keyboards for entering large amounts of data. And each brand of PDA typically has its own proprietary keyboard offering. Wouldn't it be nice if every PDA user could instead choose from a variety of standard keyboards without being tied to one brand? The answer is yes if you're Joe-PDA keyboard user. But what if you're the PDA manufacturer? Do you really want your valuable add-on keyboard monopoly destroyed by cheapo clone keyboards?

In fact, the OTG crew recognized the issue with provision for the so-called target peripheral list (TPL). In essence, this concedes the fact that the aforementioned PDA manufacturer may prefer to support only their own add-on keyboards.

Another oft-cited example of the need for OTG is the idea of a digital camera being able to connect directly to a printer instead of having to two-step a picture file from the camera to the PC, and then from the PC to the printer. But instead, how about just giving the printer itself limited USB host capability? Check out the photo printers down at the emporium, and you'll see that's exactly what's happening (look for the PictBridge logo).

Remember that the original success of USB was only by virtue of a huge up-front investment by Intel. They put USB connectors on the motherboard long before there was much of anything to plug in. That made sense for Intel, because it made PCs easier to use (more sales, more add-ons, less support). Oh, yeah, because they don't make the gadgets that plug into USB, the commoditization of add-ons was no skin off their nose.

So, who's going to volunteer to drive USB OTG? Given its goal to eliminate the PC as a middleman, I doubt Intel will be leading the charge. What other valiant supplier will take the risk of being first, and maybe last, on the block? And of that short list, who has the muscle to move the market?

Think I'm being overly pessimistic about OTG? Well, check out the market predictions in Eric Huang's "USB On-The-Go Overview," which was presented at an August 2004 USB OTG training seminar (see Figure 1).[1]

The OTG projections look rather yawn-inspiring if you ask me.

KEEP IT SIMPLE

Rather than spending time on tomorrow's problems of tomorrow, I'm more interested in ways to get a simple, not to mention home-brew, gadget hooked up to USB today, preferably with a minimum of muss and fuss. One option is to use one of the many USB-enabled MCUs as the basis for your design. The good news is that there is a large and ever-growing list of parts to choose from. Nevertheless, your choice of platform is limited (i.e., the number of USB-enabled MCUs is a tiny fraction of the entire MCU universe).

Another concern is that you may have to learn more than you want to about the innards of USB, even presuming a lot of support from the chip supplier (sample drivers, application notes, etc.). At the least you'll have to delve into the subject enough to assure yourself that handling the USB stuff doesn't compromise your application's performance and robustness.

After you decide to sell a USB gadget to the unwashed masses, the PC-side driver becomes a huge stumbling block. Writing the driver is one thing, but it's a problem that pales in comparison to the fact that you're now in the PC software business whether or not you like it. Calls in the middle of the night from brand X PC users. Upgrades and bug fixes. Microsoft sneezes and you catch pneumonia. Yechh!

One good idea is to piggyback on the drivers Microsoft already provides. The simplest and most generic is to impersonate a keyboard or mouse as a member of the human interface device (HID) class. It might be tempting to go for a higher performance class (such as mass storage and, most recently, video), but they come with a lot more baggage, overhead, and complexity.

I've concluded that, for now, one of the best ways (certainly the easiest) to get on board is a simple USB-to-serial adapter solution. In conjunction with a virtual COM port (VCP) driver on the PC, a converter can allow serial port-based applications to hook up to USB, yet run as they are with no hardware or software changes. If you're connecting an existing design that includes an RS-232 transceiver and the familiar DE-9 or DB-25 connector, there are plenty of off-the-shelf converter dongles to choose from.

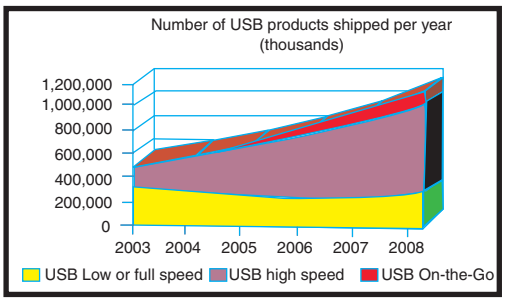


Figure 1—*Ready, set, don't go. USB On-the-Go faces a long and winding road before it achieves anything near the popularity of the original. (Source: In-Stat/MDR February 2004)*

Besides compatibility and simplicity, using a converter offloads all of the USB processing from your host MCU. And now you have the luxury of choosing any MCU with a UART (hardware or bit-banged) for your application processing. If you have the luxury of making a PCB, you can streamline things a lot by connecting a USB-to-serial adapter directly to your UART, eliminating the need for two RS-232 transceivers and connectors, and presenting a clean USB-only facade to the outside world.

Kudos to Future Technology Devices International (FTDI) for moving the USB-to-serial adapter forward. Besides highly integrated chips, which you'll find under the hood of many stand-alone converters, they deliver a royalty-free VCP driver. They also support a direct driver option that comes with the requisite dynamic link library (DLL) and application program interface (API). In short, FTDI bites the PC software bullet (i.e., maintenance, support, bug fixes, etc.) so you don't have to.

FTDI's latest chip, the FT2232C, incorporates a lot of features (see Figure 2). Like earlier versions (e.g., FT232BM), it includes practically everything you need for a USB serial solution. On the USB side of the chip, you just need to provide a 6-MHz clock (boosted to the 48-MHz clock USB required by an integrated PLL) and a handful of discretes and power logic depending on whether your USB gadget is bus-powered or self-powered. Bus-powered designs are served with an on-chip voltage regulator that converts the nominal 5-V USB bus voltage to 3.3 V.

You can choose to add a small serial EEPROM chip if you want to override the default USB settings (vendor ID, product description, transfer mode, etc.). The EEPROM connects directly to the '2232C and FTDI provides a utility that allows programming it via USB. The EEPROM can be shared with a local MCU by virtue of the fact that the '2232C EEPROM interface pins (i.e., chip select, clock, and date) are tristated when the chip is held in reset.

It's on the MCU inter-

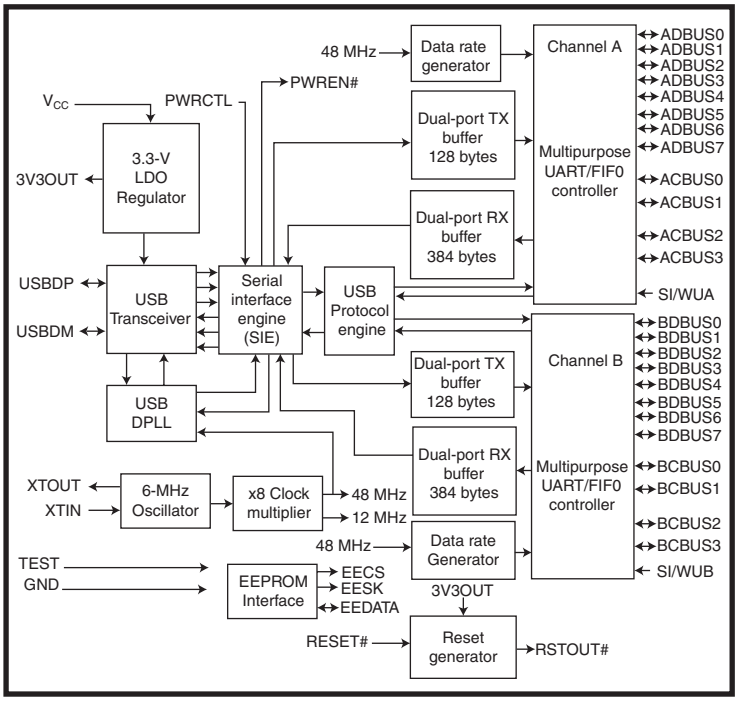


Figure 2—*FTDI started the USB-to-serial adapter chip craze. Now they're upping the ante with the two-channel FT2232C.*

face side of the part that the ′2232C breaks new ground, starting with two completely independent ports. The power-up default configuration has both ports configured as PC COM port-style UARTs including a complete set of modem control lines (RTS, CTS, etc.). The big news is that the ports also can be configured in a variety of other modes, including FIFO, MCU bus emulation, and clocked serial. These additional modes enable higher performance across a broad range of applications. In some cases they may even eliminate the need for an MCU altogether.

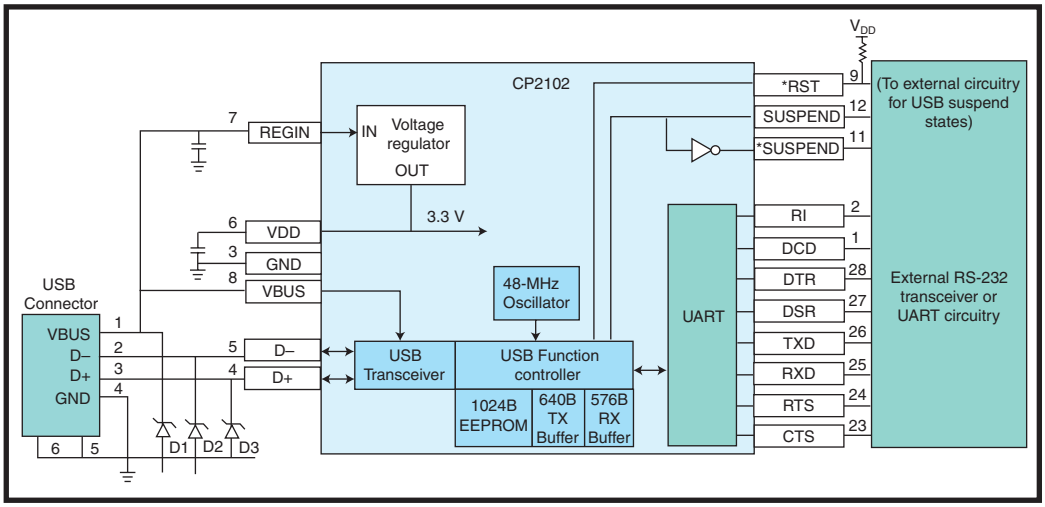


Figure 3—*Silicon Laboratories's CP2102 represents a new low in USB-to-serial adapter chips: lower chip count, lower power, and less board space.*

For instance, to support the migration of PC-based emulators, downloaders, and flash memory programmers to USB, the ′2232C Clock Serial mode provides the basis for USB-JTAG emulators. Another option described in the datasheet is an optoisolated connection to serve the growing stable of industrial and scientific add-ons (e.g., data logging and digital oscilloscope).

KEEP IT SIMPLER

Witnessing the success of FTDI, it's no surprise to see competitors emerge. As an aside in last month's column, "′51 Favorites," I noted that Silicon Labs's (formerly Cygnal) ′51 evaluation board exploited their serial-to-USB adapter chip, the CP2102 (see Figure 3). Let's take a closer look.

In terms of basic functionality, the CP2102 is similar to FTDI's FT232BM.

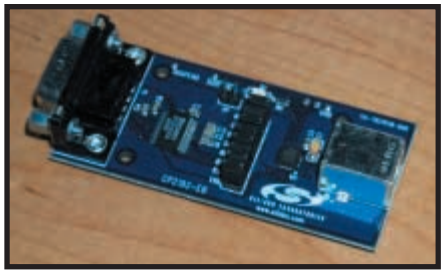


Photo 1—*The Silicon Laboratories CP2102 evaluation board bridges the gap between the old (RS-232) and the new (USB). Notice the relative component count and floor space required for each, which only reinforces the conclusion that it's time to make a change.*

Both are minimalist converters featuring the aforementioned USB direct connection on one side and a COM port-oriented UART with a full complement of modem controls on the other. Like the FTDI parts, royalty-free drivers (both Virtual COM Port and Direct DLL) are available. However, although similar in terms of basic functionality, the CP2102 delivers some key cost-saving advantages when it comes to design-in.

In what I believe may be a first, the CP2102 integrates the 48-MHz USB clock on-chip, completely eliminating the need for an external clock source (crystal, resonator, etc.). This is quite an accomplishment, because the on-chip clock must be accurate enough to meet USB specs, a nontrivial challenge considering aging and varying environmental factors (e.g., temperature and voltage).

The CP2102 also integrates the EEPROM required for nondefault USB settings that would otherwise require an external chip (as is the case for the FTDI parts). Like FTDI, Silicon Labs provides a PC-based utility for programming the EEPROM via USB.

Although both the Silicon Labs and FTDI parts have a similar pin count (28 pins for the CP2102 and 32 pins for the FT232BM), the CP2102 at only 5 mm × 5 mm has a much smaller footprint than the 9 mm × 9 mm FT232BM. Another key spec difference is the -40° to 85°C temperature range for the CP2102 versus 0° to 70°C range for the FT232BM.

Price projections always come with caveats related to volume, timing, channel (direct versus distribution), and your negotiating skill, not to mention the challenge posed by editorial lead times. That being said, as I write this column, it appears pricing for the two parts is similar at about $5 in low quantity dropping to $2.50 in high volume. Obviously, given its integrated clock and EEPROM, this gives a system cost advantage to the CP2102.

GETTING VIRTUAL

I decided to give the CP2102 evaluation board a look-see. The $49 board implements the usual USB-to-RS-232 adapter function in two chips (the CP2102 and a Sipex RS-232 transceiver), the requisite connectors, and little else (see Photo 1).

The CD that comes with the kit includes Virtual COM port drivers for Windows (98 through XP), Mac (OS-9 and OS-X), and Linux 2.40. The driver installation seemed a bit quirky. After plugging in the board to the PC USB port, I went through two sessions with the Windows New Hardware Wizard. First it installed the CP210X USB composite device and then the CP210*x* USB-to-UART bridge controller. All went well, and I used the Windows device manager to verify that the CP2102 board was assigned to an open COM port slot.

Thanks to the COM port impersonation, initial checkout is easy using a simple terminal emulator such as HyperTerminal, which comes with Windows. However, I've always found that program, admittedly designed with

Remember when...

...electronics stores were stocked to the rafters

with every component imaginable and there was ALWAYS

a knowledgeable person there to help you?

Think those days are gone?

Well they're not!

You just need Jameco!

Call for a FREE 256-page catalog today.

JAMECO® ELECTRONICS

Great Products. Awesome Prices.

1.800.831.4242

We have the electronic components and knowledgeable Technical Support people who can help **bring back the fun in your electronics hobbies, inventions, and DIY projects.**

Be sure to visit our new website at

www.Jameco.com/ccj

Jameco Electronics

—where electronics is still fun!

Sometimes A Cable Just Isn't Long Enough

Embedded Wireless Solutions For Control and Communications

Wi-Fi supports data transmission rates ranging from 11 Mbps to 1 Mbps and is ideal for networking and web-enabling your embedded applications.

Wi-Fi

Wi-Fi Add-On Kit
Kit Price
$199

Wi-Fi Add-On Kits

To add flexibility to Rabbit-based networks, Rabbit Semiconductor offers Wi-Fi Add-On Kits for use with many RabbitCores. These kits contain the tools needed, including a LinkSys Wi-Fi card, to add Wi-Fi communications to existing embedded systems. Connection to existing applications is via interposer boards.

Wi-Fi

Wi-Fi Application Kit
Kit Price
$599

Wi-Fi Application Kit

This kit provides all of the tools needed to create a Wi-Fi enabled embedded application including a RabbitCore Module, prototyping board, LinkSys Wi-Fi card, Integrated Development Environment, and royalty-free TCP/IP stack, with sample programs and libraries to simplify Wi-Fi development.

GPRS is a standard that allows data to be sent and received across the GSM mobile telephone network.

GPRS

Kit Price
$649

M2M

Kit Price
$699

GPRS / M2M Application Kits

GPRS / M2M Application Kits

These kits provide all of the tools needed to create a GPRS enabled embedded application including a RabbitCore Module, prototyping board, GSM modem, Integrated Development Environment, and royalty-free TCP/IP stack, with sample programs and libraries to simplify development. The M2M kit provides additional tools to facilitate machine-to-machine communication.

Free Reference Book With Kit!

"Embedded Systems Design using the Rabbit 3000 Microprocessor" free with wireless kit purchase.

Limited Time Only

www.rabbitsemi4wireless.com

RABBIT Semiconductor

2932 Spafford Street, Davis, CA 95616
Tel **530.753.8400**

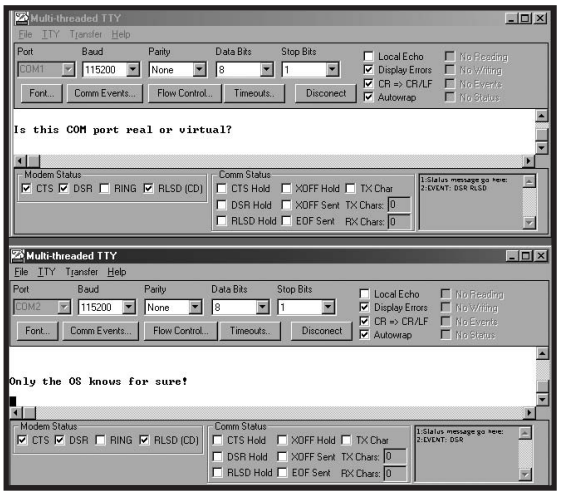


Photo 2—*After installing the virtual COM port (VCP) drivers, you can perform initial CP2102 loop-back tests on a single PC by running one copy of a terminal emulator talking to the virtual COM port and one copy talking to the real COM port.*

modems in mind, a little clunky when it comes to direct RS-232 communications. Instead, I usually use an emulator called MTTTY, which is a sample program that comes with a Microsoft serial port application note, itself useful reading.[2]

To start, I set up a simple loop-back configuration, connecting the PC's real COM port to the CP2102 board's virtual COM port. I had to admit some concern that between Windows, MTTTY, and the Silicon Labs VCP driver, the PC might get all choked up trying to talk to itself. But, as shown in Photo 2, I was able to connect each copy of MTTTY to its COM port (real and virtual, respectively) and establish communication.

Next, I moved up to true computer-to-computer communication. The configuration consisted of a new and fast XP PC and an old slow Windows 98 laptop. I experimented in both directions with each computer taking a turn connecting by real (RS-232) and virtual (USB) COM ports. This confirmed that both versions of the VCP driver (i.e., XP and Windows 98) were functional and could talk with each other.

Next, I exercised the various handshaking features, both hardware (RTS/CTS and DTR/DSR) and software (XON/XOFF). As expected, with handshaking disabled, I was able to induce overrun errors when transferring a big file from the fast computer to the slow laptop (see Photo 3). With handshaking enabled the overrun problem went away, and the file transferred successfully. This serves as a reminder that even though the CP2102 does an admirable job, it can't work miracles such as making a slow computer fast.

Data rates are another possible gotcha. The CP2102 datasheet touts the chip's ability to deliver data at up to 1 Mbps. Besides the fact such a high rate would no doubt push the envelope of what a particular PC can handle, most existing applications don't know about anything above and beyond the standard data rates (e.g., 300 bps to 115.2 kbps).

To add a twist, the workaround is different depending on your specific PC operating system. For Windows 98, it's possible to access the UART control registers directly to program the desired data rate. However, XP doesn't allow application software to have direct access to chips. (Stop me before I crash again.) The recommended hack is to change the clock divisor factors stored in the USB .INF file to allow a non-standard data rate to impersonate a standard one. For example, you could set the entry corresponding to 115.2 kbps with the clock divisor factors corresponding to 1 Mbps. Then, when an application specifies 115.2 kbps, the rate is actually set to 1 Mbps behind the scenes.

Although it's tempting to succumb to the illusion of real COM ports, don't forget there's a lot that goes on between the two ends of the virtual connection. Consider an example application that takes direct control of the modem control lines, perhaps to monitor a switch closure or drive an LED. With a real COM port, it's possible to get a reasonably accurate handle on the timing. But a virtual COM port introduces a degree of uncertainty due to the fact the actual I/O timing depends on USB scheduling beyond the control of your application.

I wrote a simple test program in BBC BASIC to demonstrate the difference (see Photo 4). Refer to the "London Calling" sidebar (page 84) for more information about BBC BASIC. All the program does is sit in a tight loop toggling the RTS line as fast as possible. Sure enough, probing with an oscilloscope revealed the real RTS line was switching about 100 times faster than the virtual one (e.g., 20 kHz versus 200 Hz).

The oscilloscope session also served as a cautionary tale that has nothing to do with the CP2102 directly, but rather the RS-232 interface. I noticed the voltage swing was twice as high on the RTS output from the PC (e.g., ± 10 V) compared to the output from the USB

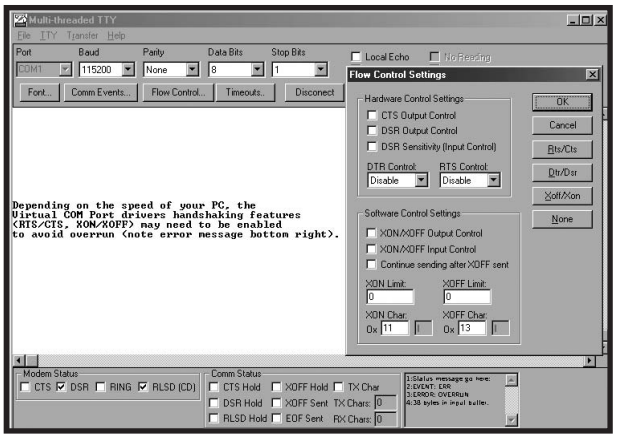


Photo 3—*The CP2102 includes a healthy amount of buffer RAM to support high data rates, but take care not to overdo it. In this case, transferring a large file from a fast to a slow PC caused overruns on the latter. Use hardware and software handshaking (or lower the data rate) to make the problem go away.*

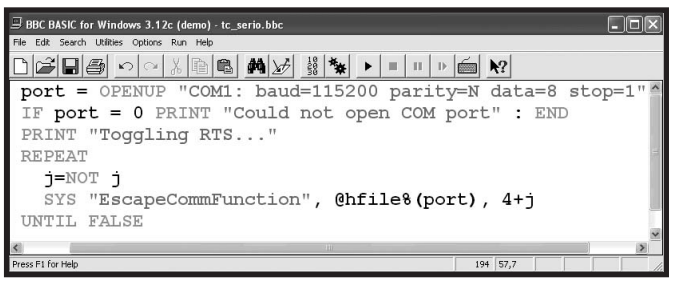


Photo 4—*It may not matter in most applications, but this simple program to toggle the RTS line reveals there are low-level timing differences between a real and virtual COM port.*

bus-powered RS-232 transceiver on the CP2102 board (e.g., ±5 V). I didn't have any problems myself, but it isn't hard to imagine scenarios (cabling, noise, using an RS-232 line as a power source) in which the difference could lead to glitches and head scratching.

JUMPSTART

Chips like the CP2102 minimize the pain of upgrading your application with USB connectivity. Yes, you could just stick with RS-232 and rely on users to buy their own USB-to-RS-232 converter dongle. That may seem like an easy way out, but does it really make sense?

Consider the fact that by the time you add an RS-232 transceiver and DE-9 connector, there will be little (if any) cost, size, or power consumption advantage compared to a USB-to-serial adapter chip and connector (the latter is optional as with a USB keyboard or mouse).

Presenting a USB-only face to the world also gives your design the credibility associated with products from major players who have the wherewithal to roll their own USB solution. It also delivers cross-platform plug ability (e.g., PC, Mac, Linux, etc.) that a serial or parallel solution can't.

Finally, the hope that leaving the USB issue to the user will relieve you of support headaches is likely an illusion. The fact is that if the user has a problem using your RS-232 gadget with an external USB converter dongle, you have a problem too. Chances are you'll still get a call if a problem arises, and you may end up having even more USB-related support headaches (e.g., multiple vendors dongles, drivers, etc.) than if you were to bite the bullet and take charge of the USB side of the equation yourself.

There's no doubt that the time has come to ditch the past and get on the USB bandwagon. The only question I have is, What the heck should I do with my closet full of soon-to-be-obsolete nine-pin (and even older 25-pin) RS-232 stuff? Oh well, that's progress.

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

London Calling

I wanted to write some simple programs to check out the CP2102. Given the USB connection, that meant programming the PC instead of a simple SBC that I would typically use. However, I hadn't been keeping up with the times, especially when it came to today's complicated PC programming mega-suites. Studying up on the subject a bit, I quickly convinced myself that the fancy tools out there were not only overkill, but also way too bloaty to get my hands around in short order. Oh, for the good old days when PCs came with a simple BASIC built in.

Almost as a matter of whimsy, I punched "BASIC for Windows" in Google, which faithfully reported 25 million hits. Sigh. But wait, right there on the first page, I saw three links for a "BBC BASIC for Windows." What the heck? No harm in clicking.

BBC BASIC goes back to the roots of personal computing in the United Kingdom, even before the IBM PC days, a story filled with names from the past like Acorn (progenitor of ARM), Sinclair, and even CP/M. I enjoyed the trip down memory lane, but even better the fact I came away with a free download evaluation version of BBC BASIC for Windows. Head on over to www.cix.co.uk/~rrussell/products/bbcwin/bbcwin.html and check it out.

Compared to the original BASICs of yore, BBC for BASIC is extremely improved. You can have long variable names, and line numbers are only required for statements that are the target of branches. Fortunately, you don't need many of those because it has a good complement of structured programming features (e.g., procedures with local variables), making for readable code instead of GOTO spaghetti.

Otherwise, BBC BASIC is an interesting mix of powerful features (32-bit integer and 32- or 64-bit floating-point math) and eclectic ones (obscure graphics commands from the days of Teletext and EGA). The GUI is simple and clean, just the thing for dashing out some one-liners, and it has excellent built-in HELP. Best of all, it came with a healthy complement of serial I/O features that were easy to use and worked properly on both my XP and Windows 98 PCs.

The bad news is the free evaluation version is limited to 8 KB of program space. The good news is that because it's a tokenizing interpreter, 8 KB goes further than you might imagine. For example, the program I wrote to toggle the RTS line only consumed a couple of hundred bytes. Or go ahead and splurge on the full version, which is only £29.99 and includes a compiler (generate an .EXE file). Jolly good show, I say!

REFERENCES

[1] E. Huang, "USB On-The-Go Overview," www.usb.org/developers/onthego/san_jose_otg_presentations.zip.

[2] A. Denver, "Serial Communications in Win32," 1995. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnfiles/html/msdn_serial.asp.

RESOURCES

J. Axelson, "USB Chip Choices: Finding a Peripheral Controller," *Circuit Cellar*, 120, July 2000.

J. Bachiochi, "USB DMX," *Circuit Cellar*, 172, November 2004.

———"USB in Embedded Design," *Circuit Cellar*, 165, April 2004.

F. Eady, "Mission Possible: Achieve Cheap USB Connectivity," *Circuit Cellar*, 157, August 2003.

J. Pollard, "USB, FTDI Style," *Circuit Cellar*, 132, July 2001

USB Implementers Forum, www.usb.org.

SOURCES

FT232BM and FT2232C Chip
Future Technology Devices International
www.ftdichip.com

CP2102 Chip
Silicon Laboratories
www.silabs.com

IDEA BOX

THE DIRECTORY OF PRODUCTS AND SERVICES

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

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

INDEX OF ADVERTISERS

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

Preview of May Issue 178
Theme: Communications

'Net Radio: Build an Internet Radio Receiver

Connect with USBLab

USB 2.0 Interface

Add USB to Any System

Network GPIB Controller

Digital RC Servo Controller (Part 2): Circuitry Details

Three-Axis Stepper Motor Driver (Part 2): Software Implementation

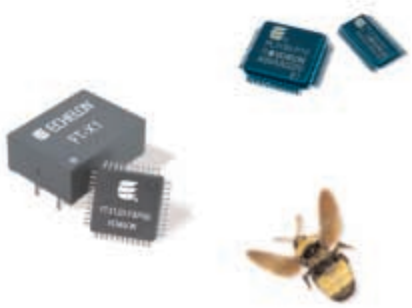
USB Interface Development

APPLIED PCs Build a Wi-Fi Web Server

FROM THE BENCH Speech Synthesis with SpeakJet

SILICON UPDATE More Flash, Less Cash

High performance. Low cost. No sting!



Do you want to add networking to your embedded devices but keep getting stung by high product and development costs? Help is at hand. Echelon's embedded LONWORKS® platform is high on performance and low on product cost and development time. Our new Mini EVK gets you from design to delivery in no time, and at very little cost. With more than 40 million devices fielded, isn't it time you found out why so many embedded system developers choose Echelon. Call +1 408 938 5200, or visit us online at www.echelon.com/mini to get started today.

ECHELON®

PRIORITY INTERRUPT

by Steve Ciarcia, Founder and Editorial Director

Dead as a Doornail

The average person these days has dozens of portable devices. We've got radios, laptops, cell phones, flashlights, cameras, navigation systems, iPods, PDAs, LCD TVs, DVD players, etc. Let's face it. We live in a gadget-happy universe. As much as we might want to deny it, we love the whole idea that science fiction is fast becoming science reality. Take cell phones as just one example. "Personal communicators" were barely thought of 20 years ago, but today you can have a cell phone with a 0.5-megapixel color view screen and features scheduling, word processing, GPS navigation and mapping, FM radio, a web browser, a TV receiver, an MP3 player, a digital camera, a video player, and a voice recorder. Cell phones are fast becoming the Swiss Army knives of our electronic universe.

Of course, these feature-laden devices are a great idea until you turn on a few of the functions and the batteries die after about 10 min. How many times has your laptop's battery icon claimed it has, say, 3 hours and 41 min. left, but an hour later little flashing things start appearing on the screen, warning you of a low battery and imminent shutdown? The manufacturer of the laptop neglected to tell you that the 4-hour battery life drops 30% if the screen is on (so you can actually read it), and another 50% if the Wi-Fi is running (the only reason you bought the laptop).

Battery technology is the one critical part of our electronic universe that never heard about Moore's law. What kind of processing improvement have we seen since 1965? A million times? I just read that disk drive capacity alone has increased over 400,000% in the last 15 years. Rechargeable battery capacity (energy density) has increased just 300% in the same period of time. All the great features we want in portable devices aren't missing because of a lack of technology. It's just that the available battery power can't handle their present level of electronic integration.

Battery design is fundamentally an issue of chemistry. Battery technology hasn't changed much because the periodic table hasn't changed either. Every time you start your car, keep in mind that it's using basically the same lead acid battery designed by Gaston Plante in 1859. All those NiCd batteries used in your power tools trace back to discoveries by Waldmar Jungner in 1899. Talk about a slow evolution.

Energy density improvements in recent years are primarily the result of fiddling with the zinc-copper-lead-carbon-nickel-cadmium-mercury-lithium soup to tailor specialized power curves. The good news is that Li-ion batteries have made many of the devices we use today a reality simply because we finally have enough power to run them. The bad news is that we may not be able to tweak the chemistry much more. Conventional battery technology might still improve, but that will take a long time, and it will never result in the magic bullet that gadget manufacturers really want as a portable energy source.

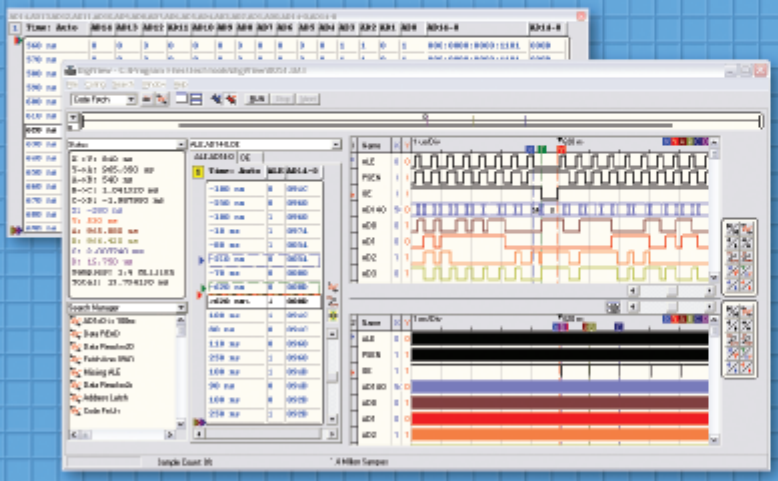
In the meantime, how do we keep adding new features to all these gadgets? First, we improve efficiency. If companies want to sell portable computers with everything including the kitchen sink and more than a half-hour run-time, then they have to design special energy-efficient processors and circuitry that shut off unused buses, logic, and memory blocks. They also need to find an alternative to energy-wasting LCD backlights for portable devices with video displays. One technology with potential is organic LED (OLED) display. Of course, there are little problems. It's expensive, fragile, and the colors fade. But hey, all that used to be true about the stuff we commonly use today.

The ultimate solution is a fuel cell battery. On the drawing board for the last 50 years, a fuel cell is essentially a battery that combines hydrogen and oxygen to produce electricity. Conceivably, when you need to recharge, you would simply pour in a little methanol and away you would go. Like gasoline in a car engine, the appealing advantage is that, for a given volume, methanol has far greater energy density than something like Li-ion. Of course, before fuel cells become a reality, there are little issues to deal with: some fuel cells like to run at 350°C; they prefer being constant-current generators; and they're made with expensive platinum. Plus, methanol is hard to find, and you can't get on a plane with it. Of course, every technological advance started with lots of technical obstacles and naysayers predicting their failure. If you look at how far we've come, you can easily envision that fuel cell batteries will be in common use eventually. I for one never want a battery that is dead as a doornail again. I prefer the option where I can simply say, "Fill 'er up."

steve.ciarcia@circuitcellar.com

DigiView

100 MHz, 18 Channel, Portable Logic Analyzer!

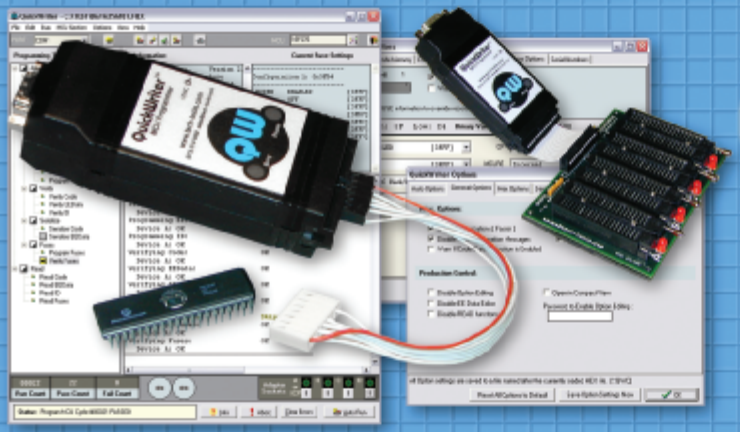


Auto Hardware Compression Captures from 128K to 30 Billion Samples @10ns resolution!

only $499.00

- Save up to 5 minutes of Data @ 10ns.
- Edge and Pattern Triggers.
- Real-time Hardware Compression.
- USB Powered for Portability.
- Includes Capture & Analysis Software.
- Pattern Searches with Match, Duration & Animation.
- Drag & Place and Drag & Snap Markers.
- Waveform Next/Previous/Nearest Edge snapping.
- Multi-Signal Data Tables with Link Groups.
- Split Waveform Views with Link Groups.

Download and try the Software!



QuickWriter

Multi-Function PICmicro® MCU Programmer In-Circuit & Gang Operation

only $199.00

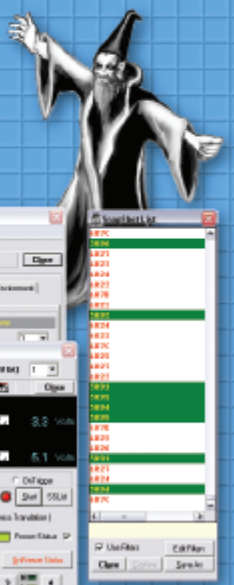
- Supports 12, 14, 16 & 18 Series PICmicro MCUs.
- Edit Code, EE Data, IDs and Configuration Fuses.
- Auto or Manual Serialization in Code or EE Data.
- Firmware auto synchronizes to Software version.
- Control Files protect Option Settings.
- Accepts Command-Line Parameters.
- Tests and Monitors Voltages.
- Control Signal for Self-Powered Circuits.
- Single and 4 Gang Adapters available.
- Includes Cables, Software & U.S. Power Supply.

See Our Other Embedded Development Tools

EconoROM III & FlexROM III

EPROM & FLASH Emulation at its best!

from $179.00



TechTools

www.tech-tools.com

(972) 272-9392 • sales@tech-tools.com

Get Started with Robots...

Ready to jump into robotics? *Begin your adventure with our BASIC Stamp® controlled Boe-Bot® Robot Kit. We've packed it full of everything you'll need to build an intelligent robot (just add a PC). The included "Robotics with the Boe-Bot" text will walk you through robot assembly, programming in PBASIC, and on to many illuminating Boe-Bot experiments. The solderless breadboard on the Board of Education® makes it easy to create a circuit and very versatile for when you're ready to move beyond the book. Free technical support completes the package.The Boe-Bot robot is suitable for ages 14 and up.*

Boe-Bot Robot Kit (Serial); #28132; $199.00
Boe-Bot Robot Kit (USB); #28832; $199.00

...or keep going...

We stock an endless variety of robots, accessories, components and sensors. Pictured above (from left to right) are the HexCrawler robot, the Crawler accessory kit, the SumoBot®, the Toddler® walking robot, the Tank Tread accessory kit.

With free technical support and an active Robotics Discussion Forum, Parallax robots simply cannot be beat. *For more information visit the ROBOTICS section of www.parallax.com.*

Order robots online at www.parallax.com *or call the Parallax Sales Department toll-free at 888-512-1024 (Mon-Fri, 7 a.m. to 5 p.m., PT).*

PARALLAX INC

www.parallax.com