CIRCUIT CELLAR

THE WORLD'S SOURCE FOR EMBEDDED ELECTRONICS ENGINEERING INFORMATION

APRIL 2013
ISSUE 273

EMBEDDED PROGRAMMING

Build a G-Code Controller for a CNC Router

Linux and Concurrency

Weatherize Your Embedded App

Calculate Software Failure Risk

PLUS

Enterprising Engineer

// Cloud Electrofusion Machine
// Wireless Temp Control
// Data-Gathering Applications
// And More

$9.00US $10.00CAN
04>
0 74470 75349 0

circuitcellar.com

Easy-to-use Serial to Ethernet Solutions

- **SBL2e 100**
  2-Port 3.3V TTL serial device support
  with analog to digital converters & digital I/O
  Part No. SBL2E-100IR: $21.95 Qty. 1K

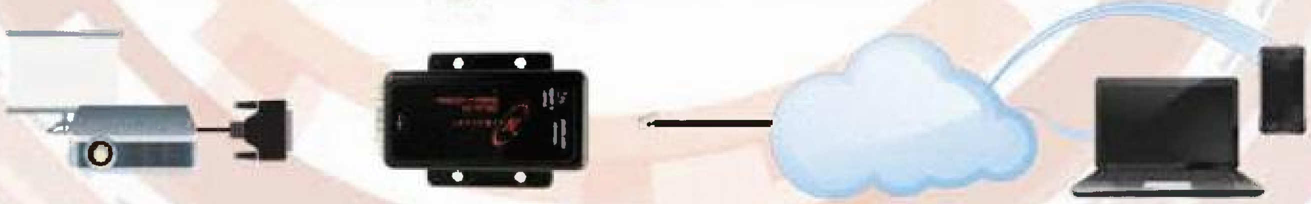
- **SBL2e CHIP (80-pin LQFP)**
  2-Port 3.3V TTL serial device support
  with analog to digital converters & digital I/O
  Part No. SBL2ECHIP-236IR: $12.50 Qty. 1K

For as low as **$12.50**

- **SBL2e X**
  1-Port RS-232 serial device support
  Part No. SBL2EX-100IR: $79.00 Qty. 1K

- **SBL2e XA**
  1-Port RS-232 and 3.3V TTL serial device support
  with analog to digital converters & digital I/O
  Part No. SBL2EXA-100IR: $79.00 Qty. 1K

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



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

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

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

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

Need a custom solution?

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

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

NetBurner
Networking In One Day!

Information and Sales | sales@netburner.com
Web | www.netburner.com
Telephone | 1-800-695-6828

freescale
Alliance Member

mouser.com

The Newest Products for Your Newest Designs℠

The NEWLY redesigned
Product Knowledge Center.

More Product Knowledge
More New Products
More Technologies

Authorized distributor of semiconductors and electronic components for design engineers.

MOUSER ELECTRONICS

EDITOR'S LETTER

Necessity and Invention

Tom Cantrell wanted to stop fiddling with his sprinklers as he tried to balance conserving water in California and keeping his lawn green. So he asked himself if he could craft a weather-savvy sprinkler controller.

In this issue, he describes how to weatherize an embedded app. He uses a Texas Instruments MSP430 microcontroller and a WIZnet W5200 smart Ethernet chip to access National Weather Service forecasts and data (p. 36).

Engineer and entrepreneur Michael Hamilton also has found that necessity breeds invention—which in turn can start a new business. "While working for Ashland Chemical in clean room environments, I realized there was a need for an accurate humidity controller," he says. "This led me to design my own temperature and humidity controller and form my first company, A&D Technologies."

In our interview, he talks about what he has done since, including founding another company and becoming an award-winning designer in the RL78 Green Energy Challenge (p. 44).

Jeff Bachiochi isn't a musician. But he didn't need to be one to work with the musical instrument digital interface (MIDI), which relays instructions on how to play a piece directly to an instrument (bypassing the musician). In this issue, he describes the circuitry needed to connect to MIDI communication and display messages between devices (p. 60).

Also, Brian Millier describes how he built a microcontroller-based G-code controller for a CNC router. Even if you are not interested in building such a controller, you can learn from the techniques he used to provide the multi-axis stepper-motor motion (p. 30).

You also might find Scott Weber's experience instructive. After placing microcontroller-based devices throughout his home, he found he needed a control panel to enable him to update the devices and check on their operation. He shares his panel's basic structure and its software design. Its display shows him all the information he needs (p. 22).

A shift in the timing signal—or jitter—of a digital transmission can adversely affect your high-speed designs. Robert Lacoste explains how to diagnose a case of the jitters (p. 54).

While wear and tear affect the reliability of hardware, software reliability is different. Whatever causes software to fail is built-in, through errors ranging from poor coding to typos to omissions. On page 51, George Novacek shares some methods of calculating the probability of faults in your firmware.

Also this month, Bob Japenga continues looking at concurrency in embedded systems. In the sixth article of his series, he discusses two Linux mechanisms for creating embedded systems—POSIX FIFOs and message queues (p. 48).

Finally, "From the Archives" features a 2003 article by Mark Balch about Verilog HDL. He discusses how to use it in your custom logic designs for digital systems (p. 68).

Mary Wilson
editor@circuitcellar.com

CIRCUIT CELLAR®

THE WORLD'S SOURCE FOR EMBEDDED ELECTRONICS ENGINEERING INFORMATION

EDITORIAL CALENDAR

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

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

Communications: Projects that deal with computer networking, human-to-human interaction, human-to-computer interaction, and electronic information sharing (e.g., speech recognition, data transmission, Ethernet, USB, I^2C, and SPI)

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

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

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

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

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

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

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

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

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

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

UPCOMING IN CIRCUIT CELLAR

FEATURES

Embedded DNA Sequencer, by Fergus Dixon

Serial Port-to-SPI Programming Dongle, by Jesús Calviño-Fraga

G-Code CNC Router Controller (Part 2): Axis and Host Control, by Brian Millier

Wi-Fi-Connected Home Power Monitor, by Donald Kunzig and Robert Kunzig

COLUMNS

Build a MIDI Communication Device (Part 2): MIDI Output, by Jeff Bachiochi

Hall Effect Sensor Calibration, by Ed Nisley

Testing and Testability (Part 1): Design a Simple Dedicated Tester for an Electronic Device, by George Novacek

Programming memory?

Emulating a microcontroller?

Programming devices?

Don't let our good looks fool you...

Total Phase tools can do it all.

USB SPI I2C CAN

Clockwise from top left: Aardvark™ I2C/SPI Host Adapter, Cheetah™ SPI Host Adapter, Beagle™ USB 480 Protocol Analyzer, USB Komodo™ CAN Duo Interface

Total Phase host adapters, interfaces, and protocol analyzers deliver best in industry results anywhere you put them to work. Total Phase I2C, CAN, USB and SPI tools give you:

Versatility. Non-intrusive monitoring or active communication.

Flexibility. Native support to program a wide variety of devices.

Speed. Fast programming and analysis times.

Ease. Easy operation for all tasks.

Plus Flash Center, Control Center, and Data Center always included.

Design West 2013

Visit us at Booth #1824 and receive your choice of a free SOIC-8/8W socket board or a 10-pin split cable.

(Offer limited to quantities on hand)

TOTAL PHASE

Industry-Leading Embedded Systems Tools for USB, I2C, SPI and CAN.

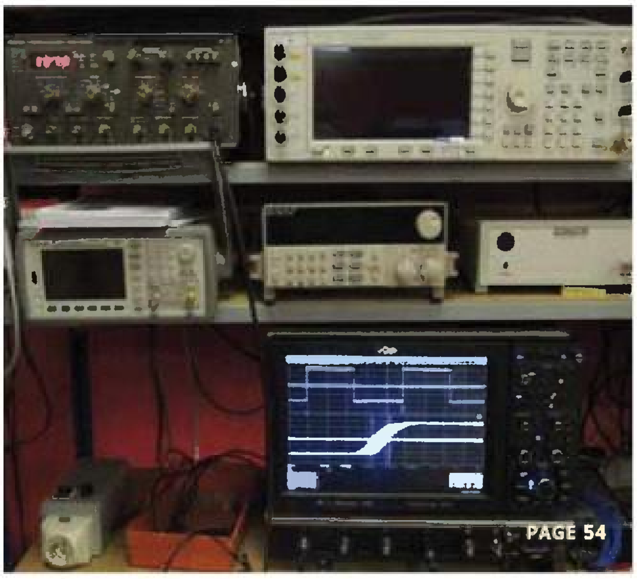
www.totalphase.com

CONTENTS

April 2013

Embedded Programming

ISSUE 273



NEWHAVEN DISPLAY INTERNATIONAL

New Capacitive Touch TFTs

- 3.5", 4.3" and 5.0" Capacitive Touch TFTs
- Built-in touch panel controller with I^{2}C interface
- 6H hardness, >1 million touches
- 5 simultaneous touches (including lift, zoom & touch gestures)
- 6mA at 3.3V supply, 0.03mA sleep mode; 60HZ refresh rate; >100dpi resolution

NEWHAVEN DISPLAY INTERNATIONAL

Newhaven Display International Inc. now offers projective mutual capacitive touch TFTs with a built-in touch panel I^{2}C interface controller. This allows for quick time to market while keeping your development costs low. The design of the capacitive touch TFTs will be available in 3.5", 4.3" and 5.0" size displays. The capacitive touch panel is the ideal tool for enhancing the user interface experience with any TFT application. These touch TFTs have multi-touch ability and are extremely durable in most environments including withstanding water contact, dust and grease. Capacitive touch TFTs activate with either a bare finger or a capacitive stylus and have an excellent dragging performance.

For more information visit: circuitcellar.newhavendisplay.com
Contact us at: 847-844-8795

SGS

RoHS Compliant

THE TEAM

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

THE NETWORK

audioXPRESS

CIRCUIT CELLAR
THE WORLD'S SOURCE FOR EMBEDDED ELECTRONICS ENGINEERING INFORMATION

elektor

techthe future
Tech the Future explores the solutions for a sustainable future provided by technology, creativity and science.

VOICE COIL

OUR INTERNATIONAL TEAMS

United Kingdom
Wisse Hettinga
+31 (0)46 4389428
w.hettinga@elektor.com

USA
Hugo Van haecke
+1 860 875 2199
h.vanhaecke@elektor.com

Germany
Ferdinand te Walvaart
+49 (0)241 88 909-0
f.tewalvaart@elektor.de

France
Denis Meyer
+31 (0)46 4389435
d.meyer@elektor.fr

Netherlands
Harry Baggen
+31 (0)46 4389429
h.baggen@elektor.nl

Spain
Eduardo Corral
+34 91 101 93 85
e.corral@elektor.es

Italy
Maurizio del Corso
+39 2.66504755
m.delcorso@inware.it

Sweden
Wisse Hettinga
+31 (0)46 4389428
w.hettinga@elektor.com

Brazil
João Martins
+351214131600
joao.martins@editorialbolina.com

Portugal
João Martins
+351214131600
joao.martins@editorialbolina.com

India
Sunil D. Malekar
+91 9833168815
ts@elektor.in

Russia
Nataliya Melnikova
8 10 7 (965) 395 33 36
nataliya-m-larionova@yandex.ru

Turkey
Zeynep Köksal
+90 532 277 48 26
zkoksal@beti.com.tr

South Africa
Johan Dijk
+27 78 2330 694 / +31 6 109 31 926
J.Dijk@elektor.com

China
Cees Baay
+86 (0)21 6445 2811
CeesBaay@gmail.com

Issue 273 April 2013 ISSN 1528-0608

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

Cover photography by Chris Rakoczy—www.rakoczyphoto.com

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

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

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

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

Membership Counter

We now have 279453 members in 83 countries.

Not a member yet?

Sign up at circuitcellar.com

Supporting Companies

All Electronics Corp.	77	Grid Connect, Inc.	21	Mouser Electronics, Inc.	1
AP Circuits	49	Hannoware	78	Myro Electronic Control	79
ARM.	71	Humandata, Ltd.	11	NetBurner	C2
Atria Technologies, Inc.	77	Imagecraft	78	Newark element14	17
Beta Layout, Ltd.	57	Imagineering, Inc.	C4	Newhaven Display Int'l	5
BusBoard Prototype Systems	76	Ironwood Electronics	76	Pololu Corp.	15
CC Gold Archive USB	41	Jeffrey Kerr, LLC.	52	Reach Technology, Inc.	76
Cleverscope.	21	Kozio, Inc.	43	Rigol Technologies	19
Custom Computer Services	79	Labcenter Electronics	13	Saelig Co., Inc.	53
DESIGN West 2013	63	Lakeview Research	52	Scidyne.	76
Elektor	29	Lauterbach GmbH	33	Sensors Expo & Conf. 2013	27
Elektor	34, 35	Linksprite	77	SID Display Week 2013	46
Elprotronic, Inc.	49	Loadstar Sensors	78	Technologic Systems	8, 9
EMAC, Inc.	57	Logical Devices	79	Techsol Engineering, Inc.	76
ExpressPCB	67	Maxbotix, Inc.	77	Tern, Inc.	77
EzPCB.	39	Mental Automation	78	Total Phase, Inc.	3
Flexipanel, Ltd.	79	Micro Computer Control	76	Triangle Research International, Inc.	79
Front Panel Express	39	Microengineering Labs, Inc.	77		
FTDI Chip	C3	Mosaic Industries, Inc.	77		

Not a supporting company yet?

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

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

Embedded Systems

High-End Performance with Embedded Ruggedness

3x faster and backward compatible with TS-72xx

TS-7800
500MHz ARM9

$229 qty 100

$269 qty 1

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

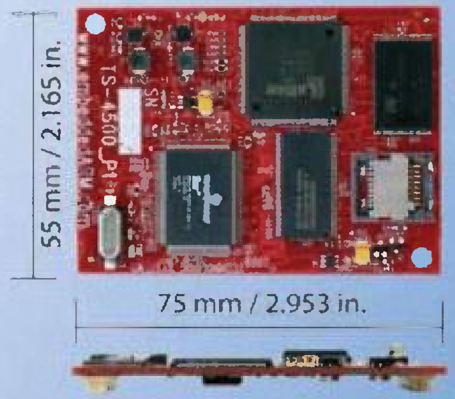
TS-SOCKET Macrocontrollers
Jump Start Your Embedded System Design

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

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

series starts at $92 qty 100

$139 qty 1



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

- Over 25 years in business
- Open Source Vision
- Never discontinued a product
- Engineers on Tech Support
- Custom configurations and designs w/ excellent pricing and turn-around time
- Most products stocked and available for next day shipping

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

New Products

Touch Panel Computers
800MHz with Video Acceleration

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

Fully enclosed TPC available Q3

NEW!

series starts at

$415 qty 100

$479 qty 1

picture of TS-8820-BOX

NEW!

series starts at

$199 qty 100

$229 qty 1

Industrial Controllers
Powerful, Rugged, Affordable

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

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

Technologic Systems

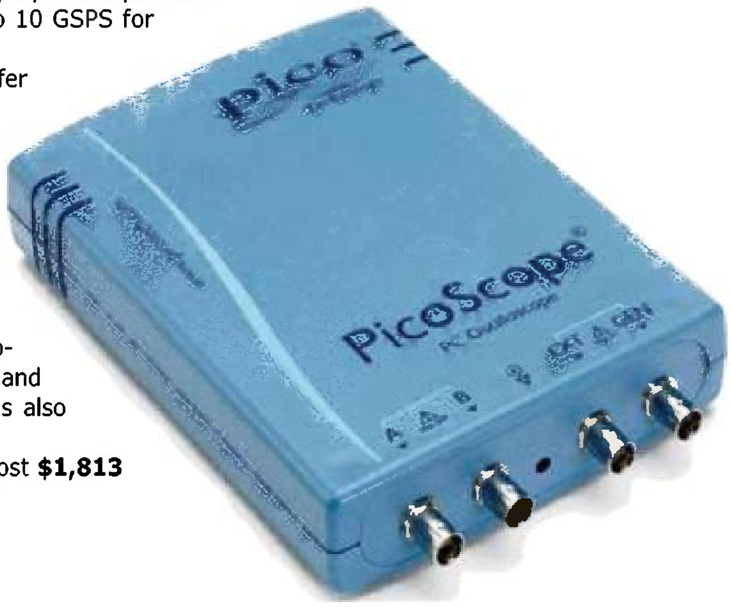
We use our stuff.

Visit our TS-7800 powered website at

www.embeddedARM.com

USB 3.0 OSCILLOSCOPES

The **PicoScope 3207A** is a two-channel USB oscilloscope with a 250-MHz bandwidth, a 1-GSPS sampling rate, a 256-ms buffer memory, and a built-in function generator. Its basic time base accuracy is ±2 ppm. The 3207A's additional features include digital triggering for accurate, stable waveform display and equivalent-time sampling, which boosts the effective sampling rate to 10 GSPS for repetitive signals.



The **PicoScope 3207B** oscilloscope has a 512-ms buffer memory and includes an additional 32,000-sample arbitrary waveform generator with a 100-msps update rate. Since the oscilloscope is powered from the USB port, an external power adaptor is unnecessary.

The oscilloscopes come with the PicoScope software for Windows, which includes automatic measurements, serial decoding of RS-232/UART, SPI, I²C, CAN, local interconnect network (LIN) and FlexRay data, and mask limit testing. Software updates are free of charge. A free software development kit (SDK) that includes sample code in many languages and enables you to write your own data-acquisition programs, is also available.

The PicoScope 3207A and 3207B USB 3.0 oscilloscopes cost **$1,813** and **$1,978**, respectively, including a set of two probes.

Pico Technology
www.picotech.com

ULTRA-COMPACT DC-TO-DC CONVERTER FAMILY

The **JCA10** series of single- and dual-output 10-W metal cased DC-to-DC converters is well suited for a variety of application environments. The converters operate from –40°C to 100°C and do not derate until 70°C. No additional heatsinking or forced air cooling is required.

The small-footprint converters are packaged in an ultra-compact 0.4″ × 0.8″ × 1″ (10 mm × 20.3 × 25.4 mm) format. Since the converters occupy less PCB space than conventional designs, you can easily reduce the size of new developments or utilize the board space for additional features. The industry-standard DIP-24 pinout makes the converter an ideal drop-in replacement for existing designs.

Accommodating a 2:1 input range, the JCA10 series offers a choice of four nominal input voltages (5, 12, 24, and 48 VDC). Its inputs are 4.5–9, 9–18, 18–36, or 36–75 VDC with standard under-voltage lockout. For each input voltage, the single-output models offer 3.3-, 5-, 12-, or 15-VDC outputs. The dual-output versions provide ±5, ±12, or ±15 VDC. Outputs are fully regulated, varying no more than ±0.3% over all input conditions, and less than ±1% across all load conditions. The JCA10 converter series offers basic 1,500-VDC input-to-output isolation and 500-VDC case-to-input or output.

The JCA10 single-output models cost **$26.52** in 500-unit quantities. The dual-output models cost **$28.89**.

XP Power, Ltd.
www.xppower.com

NEW PRODUCTS

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

HUMANDATA® **FPGA / CPLD Boards from JAPAN**

SAVING COST=TIME with readily available FPGA boards

- Basic and simple features, single power supply operation
- Quality and reliability is provided by years of sales
- Same board size and connector layout – ACM/XCM series
- All stocked items are ready to be shipped immediately
- Over 100 varieties of FPGA/CPLD boards are available
- Customizing speed grade and/or any features are possible
- Free download technical documents before purchasing
- High quality and highly reliable FPGA /CPLD boards from Japan
- Almost all products are RoHS compliance

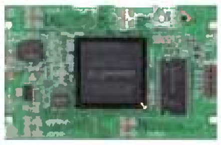
ALTERA FPGA Board

Cyclone IV E F780 FPGA board

ACM-204 series NEW

Cyclone IV E | SDRAM

EP4CE30F29C8N
EP4CE40F29C8N
EP4CE115F29C8N
Credit card size (86 x 54 mm)
RoHS compliant

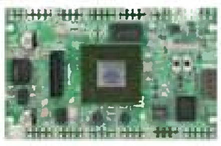


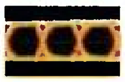
Arria II GX F572 FPGA board

ACM-025 series

Arria II GX | DDR2 | SIF40

EP2AGX45DF25C6N
EP2AGX65DF25C6N
EP2AGX95DF25C6N
EP2AGX125DF25C6N
Credit card size (86 x 54 mm)
RoHS compliant



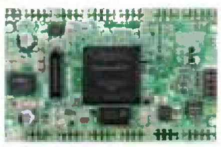


CycloneIV GX F484 FPGA board

ACM-024 series

Cyclone IV GX | DDR2 | SIF40

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



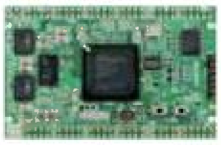
XILINX FPGA Board

Spartan-6 FGG484 FPGA board

XCM-018/018Z series

Spartan-6 | MRAM | DDR2

XC6SLX45-2FGG484C
XC6SLX75-2FGG484C
XC6SLX100-2FGG484C
XC6SLX150-2FGG484C
Credit card size (86 x 54 mm)
RoHS compliant



Virtex-5 FFG676 FPGA board

XCM-109 series

Virtex-5 | SDRAM

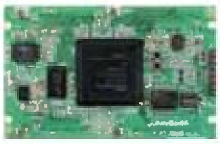
XC5VLX30-1FFG676C
XC5VLX50-1FFG676C
XC5VLX85-1FFG676C
XC5VLX110-1FFG676C
Compact size (43 x 54 mm)
RoHS compliant

Spartan-6 FGG676 FPGA board

XCM-206 series

Spartan-6 | MRAM | DDR2

XC6SLX100-2FGG676C
XC6SLX150-2FGG676C
Credit card size (86 x 54 mm)
RoHS compliant

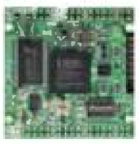


Spartan-3A FTG256 FPGA board

XCM-305 series

Spartan-3A | MRAM

XC3S700A-4FTG256C
XC3S1400A-4FTG256C
Compact size (54 x 53 mm)
RoHS compliant



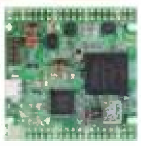
USB- FPGA Board

Cyclone IV USB-FPGA Board

EDA-301 NEW

Cyclone IV E | USB Config. | USB Comm. | HI-SPEED

EP4CE15F17C8N
Compact size (54 x 53 mm)
RoHS compliant

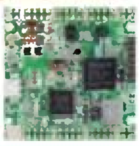


Spartan-6 USB-FPGA board

EDX-301 NEW

Spartan-6 | USB Config. | USB Comm. | HI-SPEED

XC6SLX16-2CSG225C
Compact size (54 x 53 mm)
RoHS compliant



FPGA/CPLD Stamp Module PLCC68 Series

Easy and Quickly Mountable Module

FPGA Module IC socket mountable

- 50 I/Os (External clock inputs are available)
- 3.3V single power supply operation (Voltage converters for auxiliary power supply are built-in)
- Separated supply-inputs: Core, I/O drivers
- JTAG signal
- All PLCC68 series have common pin assignment
- Very small size (25.3 x 25.3 [mm])
- RoHS compliance
- MADE IN JAPAN

XILINX PLCC68 Series

Spartan-6 PLCC68 FPGA Module

XP68-03

Spartan-6 | PLCC 68

XC6SLX45-2CSG324C
3.3V single power supply operation
On-board oscillator, 50MHz
RoHS compliant

Spartan-3AN PLCC68 FPGA Module

XP68-02

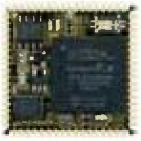
Spartan-3AN | PLCC 68

XC3S200AN-4FTG256C
FPGA internal configuration ROM
Two User LEDs
On-board oscillator, 50MHz
RoHS compliant

ALTERA PLCC68 Series

Cyclone III PLCC68 FPGA Module

AP68-04 NEW

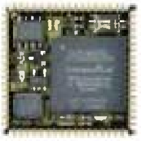


Cyclone III | PLCC 68

EP3C25U256C8N
3.3V single power supply operation
On-board oscillator, 50MHz
RoHS compliant

Cyclone III PLCC68 FPGA Module

AP68-03



Cyclone III | PLCC 68

EP3C10U256C8N
4Mbit Configuration Device
Two User LEDs
One User Switch(Slide)
On-board oscillator, 50MHz
RoHS compliant

Universal Board (Type2)

ZKB-106

- One for general power(3.3V 3A max) and the Two variable outputs for Vccio(0.8V to 3.3, 3A max)
- For ACM/XCM-2 series FPGA boards
- Power Switch and LED
- Power input:DC5V/2.1[mm] Jack/ Terminal Block (option)
- Board size :156x184 [mm]
- 4 Layers PCB, Thru-hole

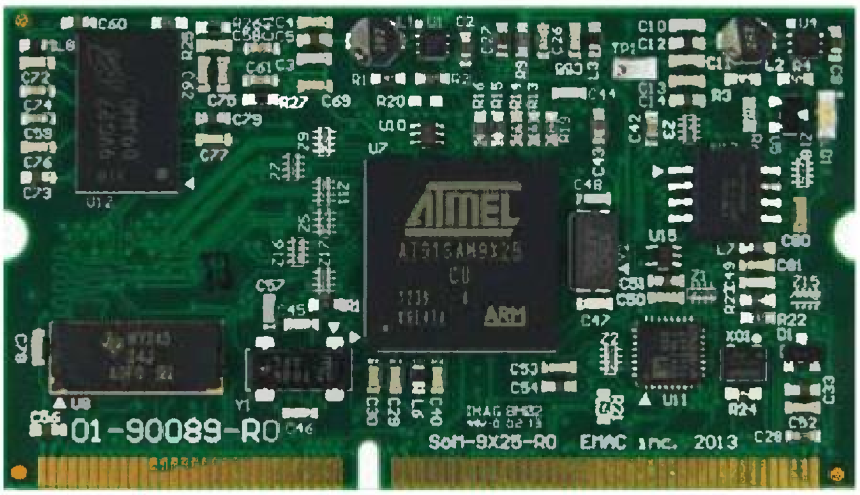
See all our products, A/D D/A conversion board, boards with USB chip from FTDI and accessories at : www.hdl.co.jp/CC/

WIDE-TEMPERATURE SYSTEM ON MODULE

The **SoM-9X25** is a system on module (SoM) based on Atmel's AT91SAM9X25 processor. A SoM is a small embedded module containing a microprocessor system's core.

This wide temperature, fanless ARM9 400-MHz SoM includes an Ethernet PHY along with six serial ports with auto RS-485. It utilizes up to 1 GB of NAND flash, up to 8 MB of serial data flash, and up to 128 MB of DDR2 RAM.

The SoM-9X25 uses the same small 144-pin SODIMM form-factor (1.5" × 2.66") as other EMAC SoMs. The board includes the entire ARM processor core (i.e., flash, memory, serial ports, Ethernet, SPI, I^{2}C, I^{2}S audio, CAN 2.0B SDIO, PWMs, timer/counters, digital I/O lines, video, clock/calendar, etc.)



The SoM-9X25 plugs into a carrier board containing all the connectors and any custom I/O an application requires. This approach enables you or EMAC to design a custom carrier board that meets your I/O, dimensional, and connector requirements without worrying about the processor, memory, and standard I/O functionality.

The SoM-150ES, which is the SoM-9X25's recommended off-the-shelf carrier board, enables you to use Linux or the WinCE operating system and tools to immediately start coding your application.

A free Eclipse IDE for Linux development is available from EMAC. All the compiling, linking, downloading, and debugging inherent to software development can be accomplished from one easy-to-use, high-level interface. Microsoft Visual Studio 2005/2008 can be used when developing for Microsoft Windows CE 6.0 applications. Visual Studio and Eclipse supply everything needed to develop SoM-9X25 applications. EMAC also provides an SDK for the SoM-9X25, which contains source examples and drivers.

Contact EMAC for pricing.

EMAC, Inc.
www.emacinc.com

RFID-BASED PCB TRACEABILITY SOLUTION

Murata's **MAGICSTRAP** for PCB radio frequency identification (RFID) tracking device has been integrated with Cogiscan's **Track Trace Control.** The joint Murata-Cogiscan solution expands on standard PCB tracking by utilizing ultra-high frequency (UHF) RFID technology, which provides a read-writable data repository on the PCB and enables fast, reliable scanning. The seamless integration of Murata's RFID technology with Cogiscan's track and trace platform makes RFID tracking of circuit boards possible as a plug-and-play technology.

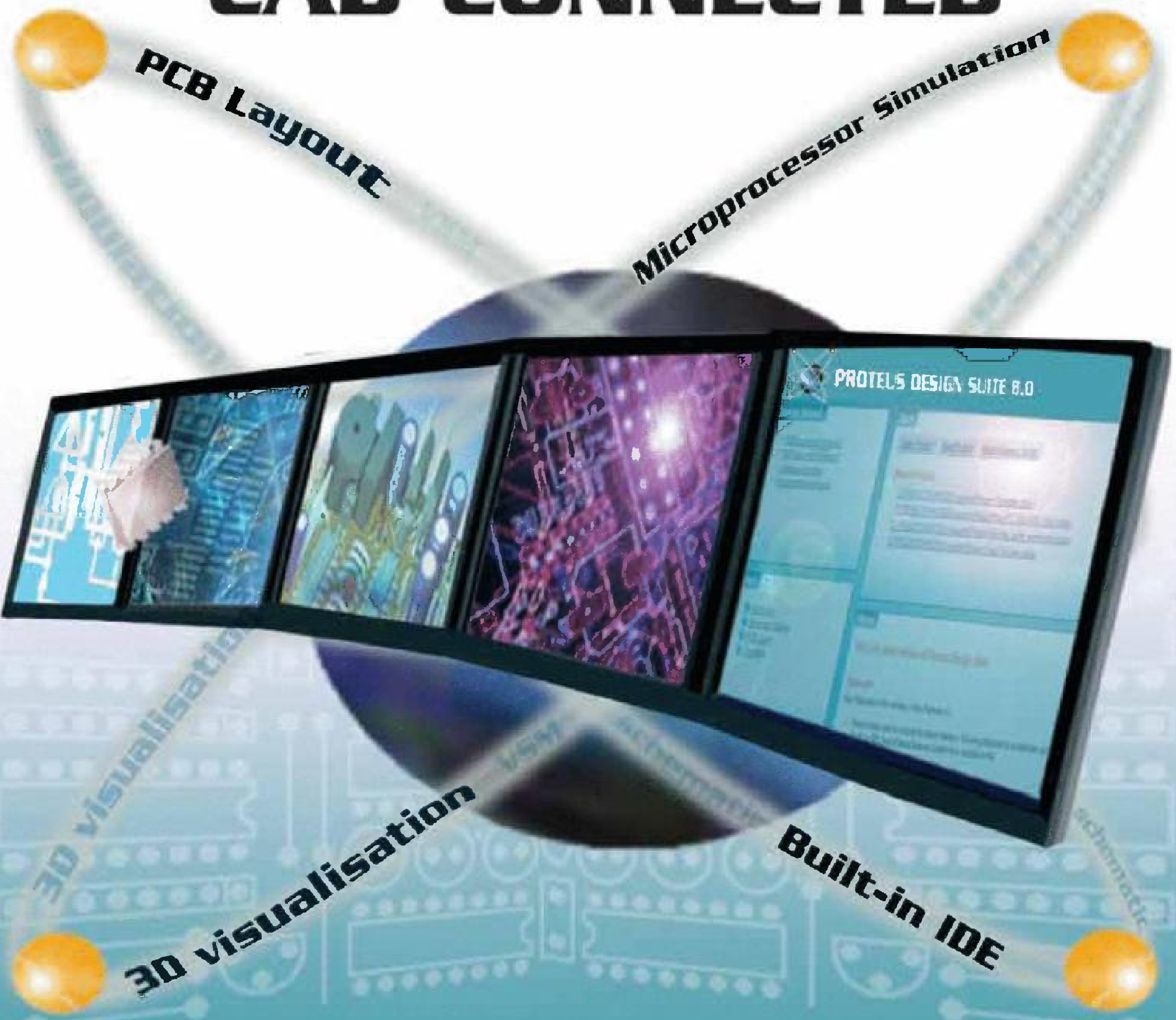
The integrated, RoHS-compliant product includes a 515-bit memory capacity, a 4-5-m read range, and a compact, 0.55-mm × 1.6-mm × 3.2-mm footprint.

Contact Cogiscan or Murata for pricing.

Cogiscan, Inc.
www.cogiscan.com

Murata Power Solutions, Inc.
www.murata-ps.com

CAD CONNECTED



PROTEUS DESIGN SUITE VERSION 8

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

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

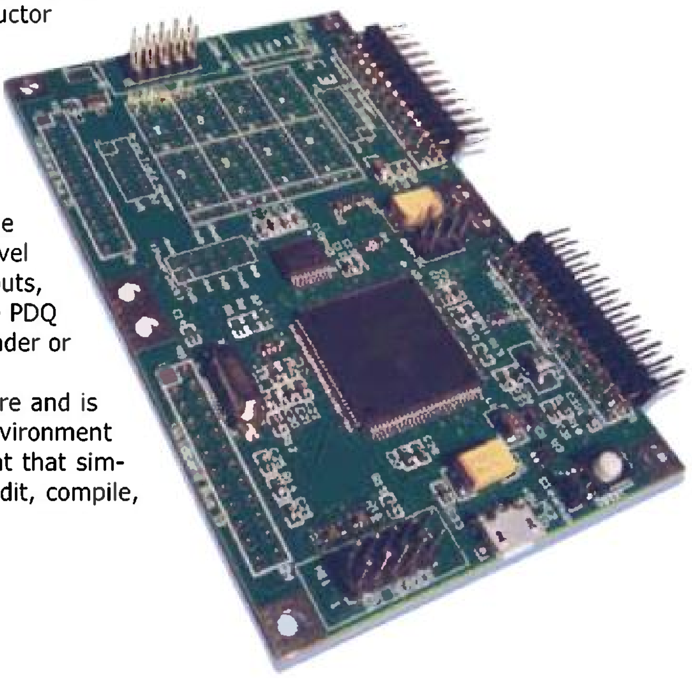
labcenter Electronics

www.labcenter.com

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

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

SINGLE-BOARD COMPUTER SIMPLIFIES INSTRUMENT CONTROL



The **PDQ Board Lite** is a low-cost, single-board computer (SBC) and development board that hosts the Freescale Semiconductor HCS12/9S12 microcontroller and an embedded real-time operating system (RTOS). This GNU C-programmable instrument controller is well suited for data acquisition and control, PWM drive, I²C sensor interfacing, laboratory automation, scientific instruments, supervisory control and data acquisition (SCADA), and instrumentation.

The compact 2.5" × 4" SBC provides the same I/O as the Freescale MC9S12A512 processor chip, including dual logic-level and standard RS-232 serial ports, 10-bit resolution analog inputs, I²C, dual SPI links, PWM, and timer-controlled digital I/O. The PDQ Board Lite is powered by 5 V, which is delivered by an I/O header or a standard microcontroller-USB connector.

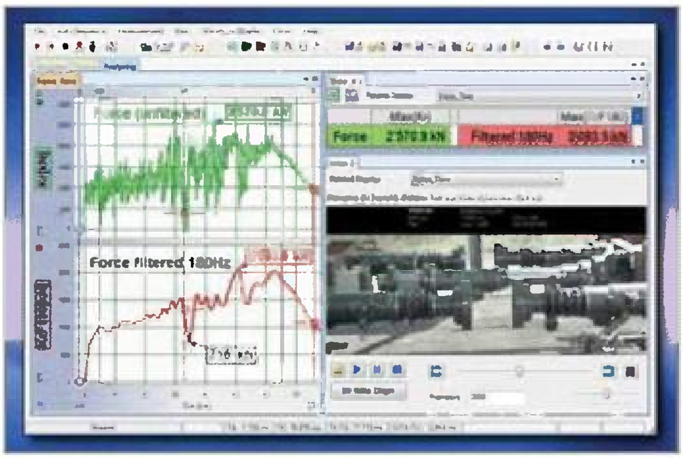
The PDQ Board Lite contains an embedded RTOS in firmware and is programmed using a C-based integrated development environment (IDE). The Mosaic IDE+ is a comprehensive GNU environment that simplifies multitasking application coding and enables users to edit, compile, download, interactively debug, and run application programs.

Contact Mosaic for pricing.

Mosaic Industries, Inc.
www.mosaic-industries.com

DATA-ACQUISITION SOFTWARE ADDS APPLICATION-BASED FEATURES

Elsys has added new features to its **TranAX 3.4** (formerly TransAS) client-server-based transient recorder application and signal analysis software. The software remotely controls data acquisition equipment over 1-Gb LANs from any location worldwide. Applications that benefit from the enhancements include ballistics measurement, crash testing, structural health, seismic research, field testing, stray voltage detection, variable-frequency drive diagnosis, connector conductivity testing, high-voltage switching, and rail and automotive control and monitoring.



Video recordings in common formats (e.g., AVI, MP4, MPG, etc.) can be imported and synchronously displayed with actual measurements signals. This enables you to use a high-speed camera simultaneously on screen to analyze measurement data and recordings. To analyze acoustic signals, frequency spectra can be displayed as standard, octave, or one-third octave. Audio signals can be weighted in dB-A or dB-C.

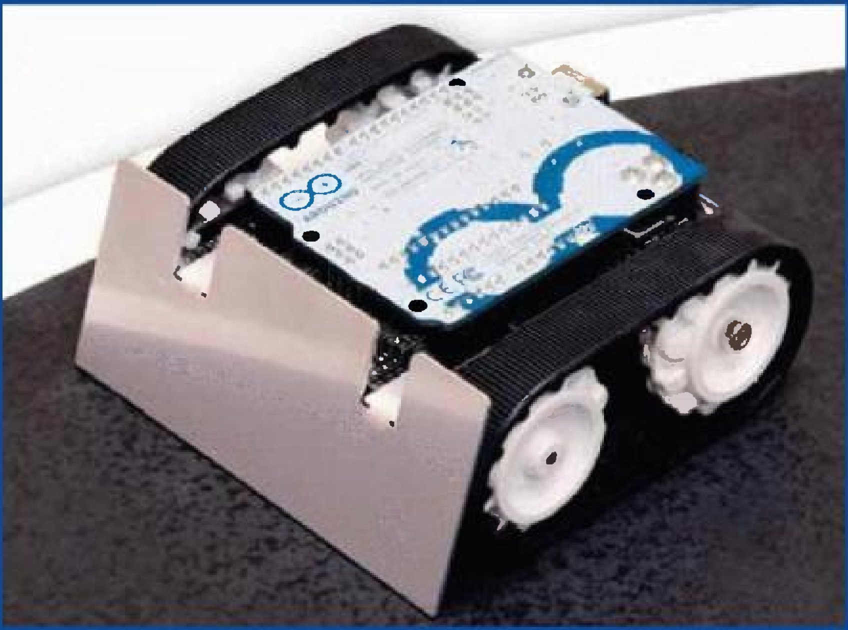
TranAX 3.4's redesigned formula editor features more than 60 mathematical functions and includes an unlimited number of math traces for extensive signal analysis. Program functions (e.g., if/then, loops, end, true/false, etc.) make computing post-measurement results from multiple, simultaneous and/or sequential recordings easy and efficient. Events benefiting from these functions include multiblock recordings, a series of stored measurement files recorded by auto-sequence, or single-shot acquisitions with multiple events in one record.

The software enables application-specific computing algorithms to be stored and recalled for later use. Programming can be done separately in a higher-level language under Microsoft .NET. The resulting software code can then be integrated as DLL in TranAX.

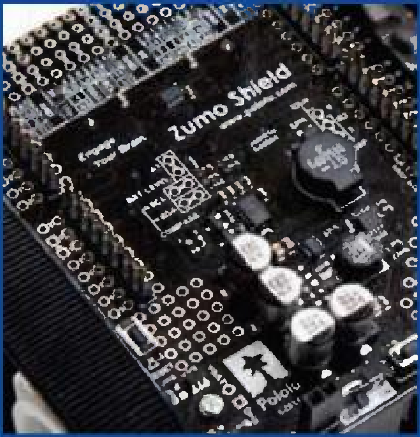
Contact Elsys for pricing.

Elsys, LLC
www.elsys-instruments.com

NEW PRODUCTS



Build and customize your Zumo!



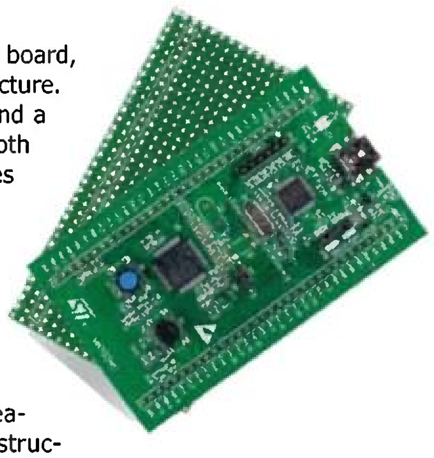
Small enough for Mini-Sumo; flexible enough to make it your own.

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

Pololu
Robotics & Electronics

Find out more at www.pololu.com/zumo

CORTEX-M4 CORE-BASED MCU



The **STM32** expansion boards help increase the functionality of the STM32 F4 Discovery board, which is built around the STM32F4 processor and features 32-bit ARM Cortex-M4 architecture. The newly available accessories include an LCD module (a 3.5″ LCD and driver board) and a camera module (with an OV9655, which is a 1.3-megapixel CMOS SXGA image sensor). Both modules connect to the third accessory, a hardware extension base board that provides Ethernet connectivity by plugging directly into the Discovery board. The baseboard helps to extend out and conveniently offer all the interfaces on the STM32F4 Discovery board.

The hardware accessories are custom designed for the Discovery board STM32 F4 and come with the necessary software drivers. This series includes devices with pin-to-pin and software compatibility with the STM32 F2 series, but with more performance, DSP capability, a floating-point unit, more SRAM, and peripheral improvements (e.g., full duplex I^{2}S, less than 1-µA RTC, and 2.44-MSPS ADCs). The ARM Cortex-M4 core features built-in single-cycle multiply-accumulate (MAC) instructions, optimized single instruction, multiple data (SIMD) arithmetic, and saturating arithmetic instructions.

The STM32 F4 series includes devices with 512 KB to 1 MB of on-chip flash memory, 192 KB of SRAM, and 15 communication interfaces. The expansion board's additional features include a 1.2-V voltage regulator with power scaling capability, high-speed data transfer, and fast peripherals.

Contact STMicroelectronics for pricing.

STMicroelectronics
www.st.com

SSL LED DRIVER OPTIMIZED FOR COMMERCIAL & WIRELESS LIGHTING

The **iW3630** is a two-stage, digital AC/DC LED driver with 45-W output power that supports a PWM digital dimming interface for wireless solid-state lighting (SSL) applications. The iW3630 utilizes iWatt's Flickerless technology to eliminate flicker across the entire 1% to 100% dimming range, while providing tight, ±5% LED current regulation. Flickerless technology incorporates a power factor correction (PFC) circuit comprised of a chopping circuit that ensures a high power factor and virtually eliminates the input line-voltage frequency component.

The iW3630's built-in isolation transformer driver works directly with 0-to-10-V dimming systems, eliminating the need for additional driver circuitry components and microcontrollers. Its PWM digital interface simplifies integration into wireless lighting systems.

The driver offers a low total harmonic distortion (THD) of less than 15% to meet stringent global energy regulations, along with a built-in over-temperature protection (OTP) and derating function to improve a system's predictability and reliability. In commercial and wireless applications, the built-in over-temperature protection and derating function eliminates the need for additional components for heat control. In addition, iWatt's PrimAccurate primary-side control technology eliminates the need for a secondary-side regulator and optical feedback isolator, while iWatt's EZ-EMI technology simplifies electromagnetic interference (EMI) filtering to further minimize the external component count.

The iW3630's digital control architecture simplifies ballast driver designs by enabling it to adapt to wide input and output conditions. Therefore, one configuration can support a range of LED string lengths to cover the entire output power range.

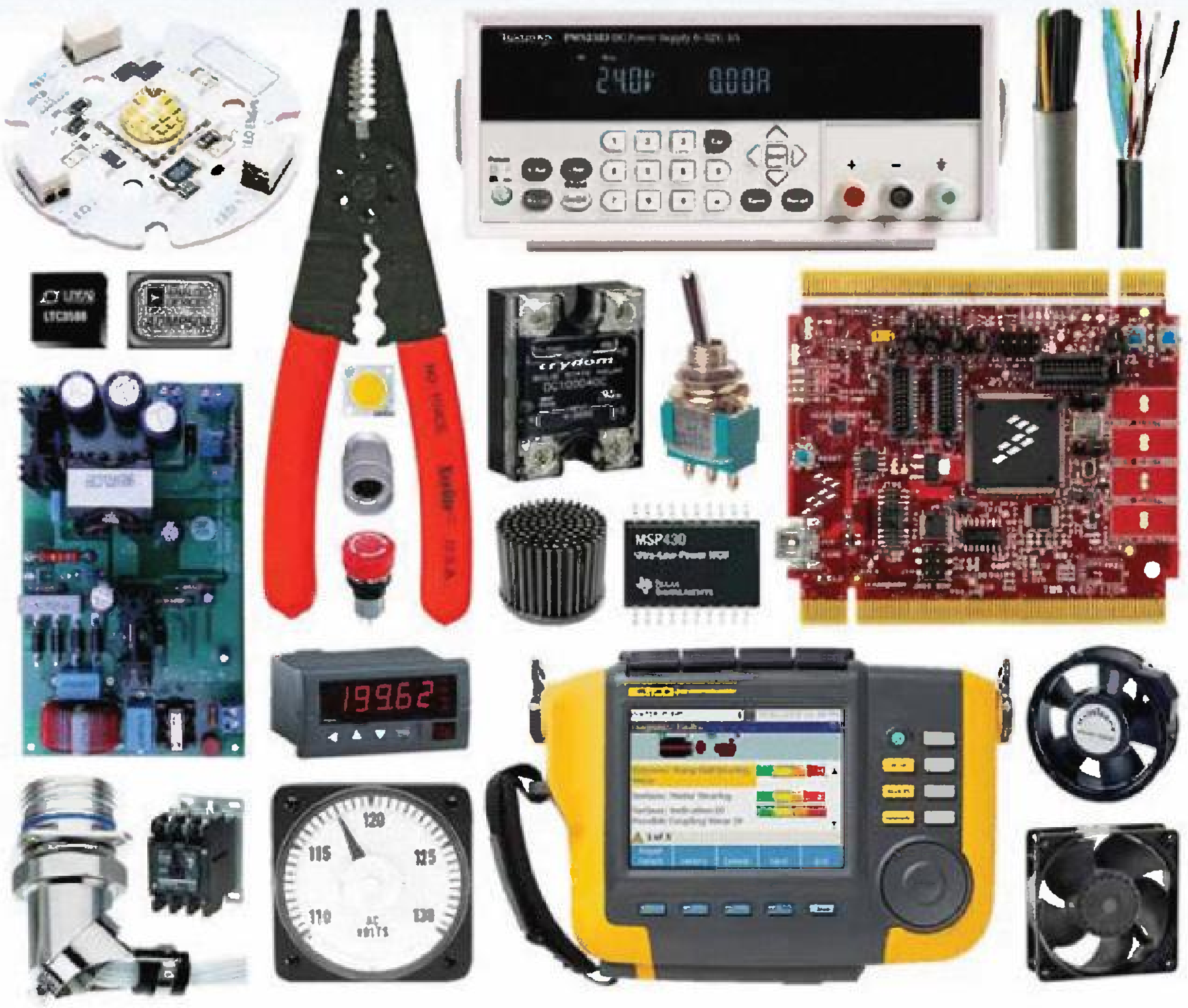
The driver's on-chip over-temperature protection and derating feature monitors the temperature inside the ballast. When thermal conditions reach a point set by the system designer, the iW3630 LED driver automatically reduces the current drive to the LEDs, lowering the power dissipation and resulting in a cooler overall operation. This reduces the risk of thermal runaway, ensures the electrolytic capacitors in the lighting system's temperature rating is not exceeded, and enables a predictable operating life.

The iW3630's built-in protection features include LED open/short, input overvoltage, overcurrent, and current-sense resistor short protections. Additional features include 3-to-45-W output power and greater than 0.95 power factor.

The iW3630 is available in a standard, 14-lead SOIC package. It costs **$1.16** in 1,000-piece quantities.

iWatt, Inc.
www.iwatt.com

Stock. Options.

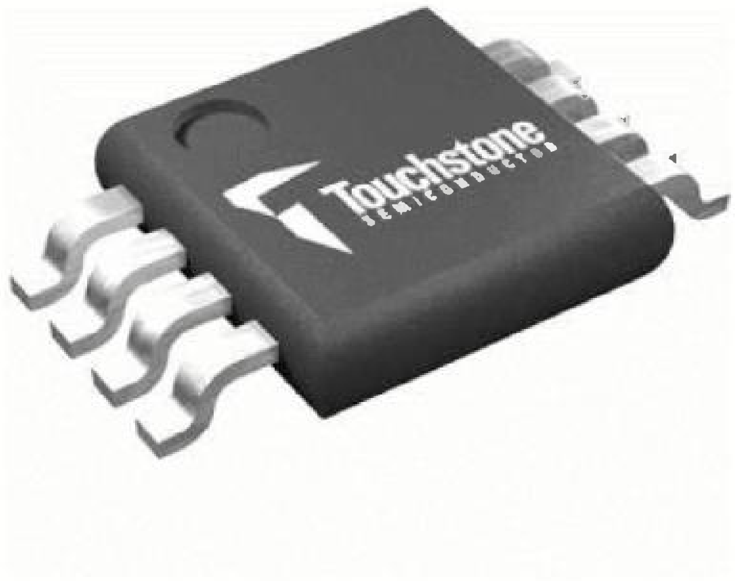


Newark element14 makes it easy to find all your electronics products and solutions – fast. newark.com

Newark element14

TWO-CHANNEL, 12-BIT SAR ADC

The **TS7001** is an easy-to-use, stand-alone, 12-bit, 187.5-ksps ADC well suited for low-power, industrial, process control, and data-acquisition applications. Some portable and fixed-form-factor applications for the TS7001 include: optical sensors, touch panels, personal digital assistants, programmable logic controllers, and medical instrumentation.

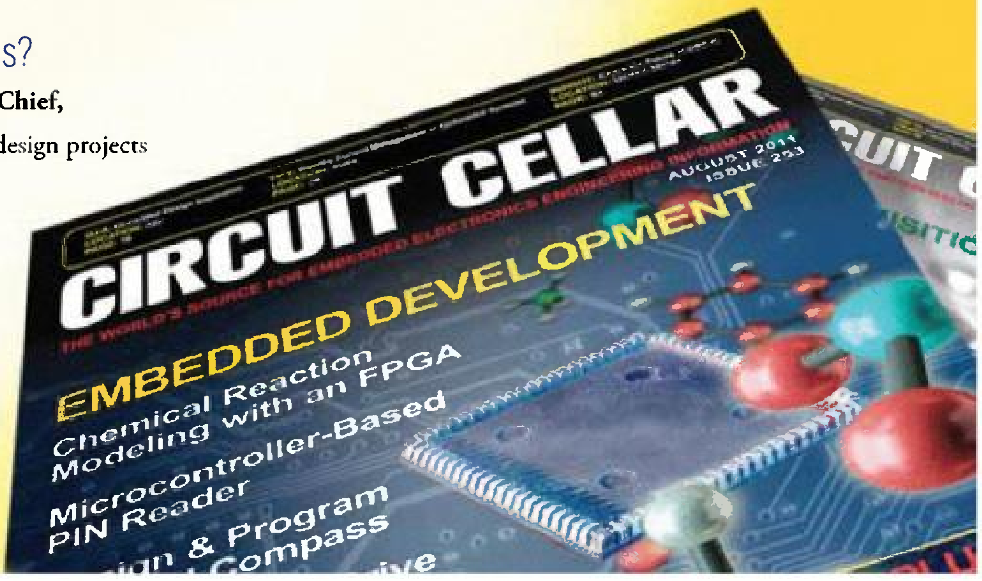


The ADC combines a 10-MHz track-and-hold, a high-speed three-wire serial digital interface, and an internal ±0.5% initial accuracy 2.5-V reference. Operating from 2.7-to-3.6-V supplies, the ADC consumes approximately 3 mW when converting at 187.5 ksps.

The TS7001's features include: One or two analog inputs each with a 0-V-to-V_{REF} or a 0-V-to-V_{DD} input range; four user-programmable, low-power operating modes including auto standby and auto power down; a 1-µA (max) shutdown-mode supply current; a –40°C-to-85°C operating temperature range; and an integrated 2.5-V, ±0.5%, 30 ppm/°C reference.

The TS7001 costs **$1.15** in 1,000-piece quantities.

Touchstone Semiconductor, Inc.
http://touchstonesemi.com


CIRCUIT CELLAR
THE WORLD'S SOURCE FOR EMBEDDED ELECTRONICS ENGINEERING INFORMATION
AUGUST 2011
ISSUE 253
EMBEDDED DEVELOPMENT
Chemical Reaction Modeling with an FPGA
Microcontroller-Based PIN Reader
Design & Program a Compass

Gordon Dick

Member Name and Location: Gordon David Dick, Stony Plain, AB, Canada (Stony Plain is suburb of Edmonton, home of the Oilers)

Education: MS, the University of Saskatchewan in Saskatoon, SK, Canada

Occupation: Gordon is semi-retired. He used to be an Electronics Technology and Computer Engineering Technology instructor at the Northern Alberta Institute of Technology in Edmonton.

Member Status: Gordon says he used to have *Circuit Cellar* issues dating back to 1995. "We were getting issues under the 'college program' then," he explained. Later, his department subscribed to the magazine and the issues came directly to Gordon. Then he obtained a personal subscription. "I still have my paper copies containing my own articles. And we bought the CDs to get all the back issues."

Technical Interests: Gordon has always been interested in electronics, both as a hobby and as a profession. He focused first on audio, then turned to microcontrollers. He has built a few microcontroller-based instruments, some of which have been the topics of his *Circuit Cellar* articles. "For a time, I was involved in building microcontroller-based dog training equipment. I built a microcontroller-based weather station, which I also wrote an article about. I have several microcontroller-based projects in my home that are specific to my needs. My cold-room temperature controller is microcontroller-based, for example."

Most Recent Embedded Tech-Related Acquisition: Gordon bought a SparkFun Electronics FG085 frequency generator kit.

Current Projects: He is working with a Freescale Semiconductor MPL3115 pressure/temperature I²C transducer for his weather station. "It has amazing barometric pressure resolution," Gordon explained.

NEW! DSA815

Spectrum Analyzer

Our new analyzer with all-digital IF technology redefines the product category!

- 9 kHz to 1.5 GHz Frequency Range
- Typical -135 dBm Displayed Average Noise Level (DANL)
- -80 dBc/Hz @10 kHz offset Phase Noise
- Total Amplitude Uncertainty <1.5 dB
- 100 Hz Minimum Resolution Bandwidth (RBW)

Get a Spectrum Analyzer at oscilloscope prices!

Starting at $1,295

RIGOL Beyond Measure

Before your next compliance test, check out the DSA815... save one trip to the compliance lab and it pays for itself!

RigolAnalyzer.com

ARM, Ltd.

Client: ARM, Ltd.
www.arm.com and
www.arm.com/tools

Location: 110 Fulbourn Road, Cambridge, GB-CB1 9NJ, Great Britain

Contact: sales.us@keil.com

Embedded Products/Services: The ARM tools range offers two software development families that provide you with all the necessary tools for every stage of your software development workflow.

ARM Development Studio 5 (DS-5) provides best-in-class tools for a broad range of ARM processor-based platforms, including application processors and multicore SoCs. Find out more by visiting www.arm.com/products/tools/software-tools/ds-5/index.php.

Keil MDK-ARM is a complete software development toolkit for ARM processor-based microcontrollers. It is the right choice for embedded applications based on the ARM Cortex-M series, ARM7, ARM9, and Cortex-R4 processors. To find out more, visit www.arm.com/products/tools/software-tools/mdk-arm/index.php.

Product Information: The MDK-ARM is a complete software development environment for Cortex-M, Cortex-R4, ARM7, and ARM9 processor-based devices. MDK-ARM is specifically designed for microcontroller applications. It is easy to learn and use, yet powerful enough for the most demanding embedded applications.

The MDK-ARM is available in four editions: MDK-Lite, MDK-Basic, MDK-Standard, and MDK-Professional. All editions provide a complete C/C++ development environment and MDK-Professional includes extensive middleware libraries.

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

CIRCUIT CELLAR EDITOR @editor_cc

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

Pitch ideas for articles

Stay informed with valuable product announcements

Learn about upcoming industry events, conferences, and more

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

twitter Follow us on Twitter

CIRCUIT CELLAR
EMBEDDED DEVELOPMENT
Chemical Reaction Modeling with an FPGA
Microcontroller-Based PIN Reader
Design & Program a Digital Compass
AUGUST 2011 ISSUE 253
DATA ACQUISITION
MCU-Based Analyzer Unit

TEST YOUR EQ

April 2013

CONTRIBUTED BY DAVID TWEED

Answers for Issue 272

Answer 1—In switching applications, a single transistor can saturate, resulting in a low V_{CE} of 0.3 to 0.4 V. However, in a Darlington pair, the output transistor is prevented from saturating by the negative feedback provided by the driver transistor. If the collector voltage drops below the sum of the V_{BE} of the output transistor (about 0.7 V) and the $V_{CE(SAT)}$ of the driver transistor (about 0.3 V), the drive current to the output transistor is reduced, preventing it from going into saturation itself. Therefore, the effective $V_{CE(SAT)}$ of the Darlington pair is 1 V or more, resulting in a much higher dissipation at a given current level.

Answer 2—One clever trick is to repeat the bits you have as many times as needed to fill the output field width. For example, if the 3-bit input value is ABC, the output value would be ABCABCAB. It yields mapping that interpolates nicely between full black and full white (see Table 1 on *Circuit Cellar*'s FTP site). This mapping preserves the original bits. If you want to go back to the 3-bit representation, take the most-significant bits and you have the original data.

Answer 3—Believe it or not, yes it can.

An induction motor has no electrical connections to the rotor; instead, a magnetic field is induced into the rotor by the stator. The motor runs slightly slower than "synchronous" speed—typically 1,725 or 3,450 rpm when on 60-Hz power.

If the motor is provided with a capacitive load, is driven at slightly higher than synchronous speed (1,875 or 3,750 rpm), and has enough residual magnetism in the rotor to get itself going, it will generate power up to approximately its rating as a motor. The reactive current of the load capacitor keeps the rotor energized in much the same way as when it is operating as a motor. (See "An Easy Way to Build and Operate Induction Generator," QSL.net, 1998, www.qsl.net/ns8o/Induction_Generator.html for additional details.)

Answer 4—The actual sampled data, represented by the square dots in the diagram, contains equal levels of Fsignal (the sine wave) and Fsample-Fsignal (one of the aliases of the sinewave). Any reconstruction filter is going to have difficulty passing the one and eliminating the other, so you inevitably get some of the alias signal, which, when added to the desired signal, produces the "modulation" shown.

In the case of a software display of a waveform on a computer screen (e.g., what you might see in software used to edit audio recordings), they're probably using an FIR low-pass filter (sin(x)/x coefficients) windowed to some finite length. A shorter window provides faster drawing times, so they're making a tradeoff between visual fidelity and interactive performance. The windowing makes the filter somewhat less than brick-wall, so you get the leakage of the alias and the modulation.

In the case of a real audio DAC, even with oversampling you can't get perfect stopband attenuation (and you must always do at least some of the filtering in the analog domain), so once again you see the leakage and modulation.

In this example, Fsignal = 0.9×Fnyquist, so Falias = 1.1×Fnyquist and Falias/Fsignal = 1.22. To eliminate the visible artifacts, the reconstruction filter would need to have a slope of about 60 dB over this frequency span, or about 200 dB/octave.

Check whether the "return from interrupt" affects any other processor state (e.g., popping a status word from the stack) and prepare the stack accordingly. Also, be aware that another interrupt could occur immediately thereafter, and ensure the master ISR is reentrant beyond that point.

What's your EQ? You can contact the quizmasters at eq@circuitcellar.com.

Gigabit Technology

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

$10.00 each (Qty 100)

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

THE NETWORKING EXPERTS

gridconnect

800.975.4743 USA • 1 630.245.1445

gridconnect.com/gridarm.html

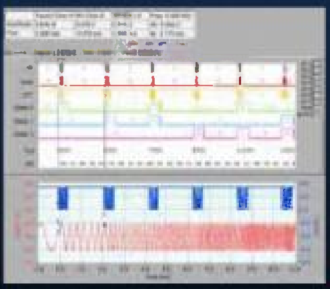
Leaders in the embedded and networking marketplace providing network hardware, high quality software and services

CS328A-XS

100MHz Mixed Signal Oscilloscope + Signal Generator

Embedded Development

EASY AS!



- + Two mixed signal triggers
- + Protocol decoding
- + Spectrum analysis
- + Symbolic maths
- + Custom units
- + Copy & paste
- + Signal generator
- + USB or Ethernet
- + 8M samples storage
- + 100 MHz sampling
- + Dual 14 bit ADC
- + Ext Trigger, 8 Digital Inputs
- + 1.5 MSa/sec HD streaming

14 Bits 100 MSPS MSO

Saelig Unique electronics

www.cleverscope.com

by Scott Weber (USA)

Control Center Software Design

Using a Model-View-Controller Paradigm

A collection of nearly autonomous microcontroller-based, RS-485 interconnected devices was a useful addition to this designer's home. But it also came with inconveniences. What if the devices needed updates or quality checks? A simple control panel solved the dilemma. This article describes the control panel's basic structure and its Model-View-Controller (MVC) paradigm, which is used to organize the software functionality.

If you've read any of my *Circuit Cellar* articles, you may recall that I placed a collection of microcontroller-based devices throughout my home, all interconnected with an RS-485 bus. Once initially set, each device is basically autonomous. However, what if I need to update a schedule or examine a devices status? The solution involved plugging my PC into an adapter I fabricated and using a Windows program to send instructions to the devices. I could easily use the PC program to change a few things; however, if I wanted to send commands the PC program wasn't programmed to do, I would have to modify it. Also, I didn't want to have to go into my workshop, boot up a PC, plug it into the network, remember the commands, and so forth. I wanted a better, easier way.

To solve this dilemma, I built a simple control panel that displays to a four-line LCD module, showing me the information I want to see. The control panel also enables me or anyone else to control the devices. In this article, I will describe the control panel's basic structure and go into depth about the software paradigm referred to as the "Model-View-Controller (MVC)." This paradigm is used to organize software functionality into a structure that is easier to develop, maintain, and enhance. It takes more ROM to handle it, but in some circumstances, the trade-off is worth it. I figured this is one of those cases.

HARDWARE

The unit I built began as a glorified clock. It ran a 32,768-Hz watch crystal oscillator to keep the time, occasionally synchronizing it with the time messages that came from the time broadcaster described in my article "Time Broadcasting: A GPS-Based Time Server for the RS-485 Network" (*Circuit Cellar* 268, 2012). The crystal connects to the microcontroller's Timer 1 inputs. Adding user push buttons and other interfaces was easy after that. Photo 1 shows the completed and installed unit.

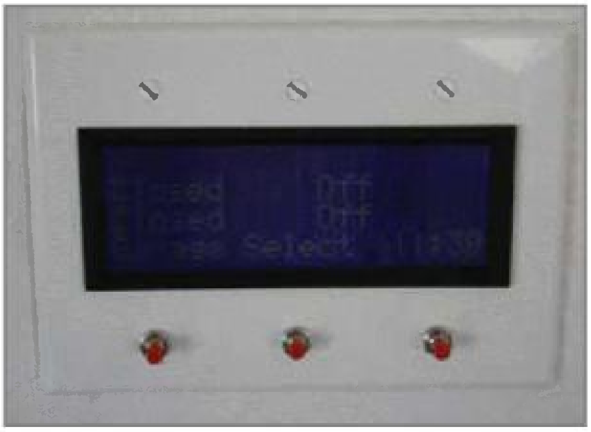


Photo 1—The completed control panel includes a three-gang wall box. I used a nibbling tool to cut out the display's bezel.

Figure 1 shows the entire unit schematic, which connects to the RS-485 network cable through J1 and obtains both 5-V power and the differential signal. Diodes D1 and D2 protect the signal lines from overvoltage. The signal is then delivered to the DS 1487 differential driver where it is converted to transistor-transistor logic (TTL) levels and flows into the microcontroller UART transmit and receive lines.

I used a large-memory microcontroller because of the larger code and RAM the user interface software needed. User inputs were taken from push buttons connected to J2, J3, and J4, which have pull-up resistors to hold the input high when the button is not pressed. The LCD unit was attached to the 16-pin connector at J5. The pinout is the

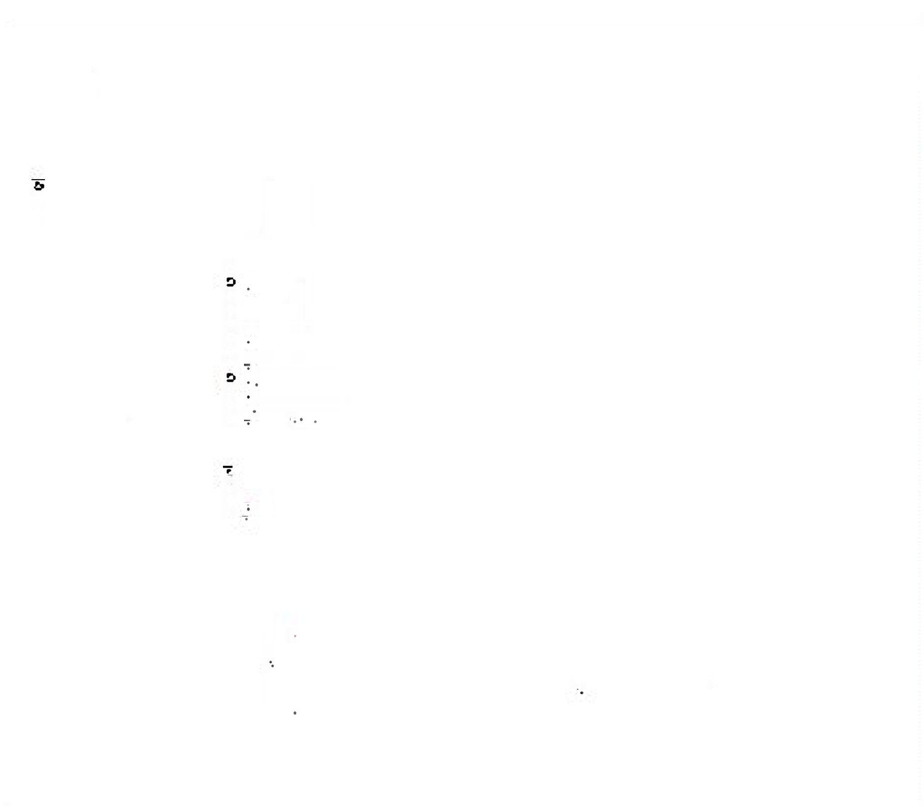


Figure 1—This schematic shows the entire control panel. J1 supplies the power and attaches to the RS-485 network. The J5 header connects to the LCD, communicating in 4-bit mode. The user switches are momentary push buttons connected to J2 through J4.

standard Hitachi 14-pin layout, with pins 15 and 16 used to drive the backlight LED. Although the LCD can be driven with 8 bits in parallel, I used the LCD in 4-bit mode, which sends each bytes upper bits then lower bits. This enables the use of three of the 8-bit register's lines to control the unit's clocking. The advantage is that all communication is accomplished through a single 8-bit register. The LCD's backlight is driven by transistor Q2. This is a simple current switch driving the LED. The current source is under the control of the microcontroller, which turns the LED off after a non-use period.

An external speaker was connected to RA1 through Q1, which enables the control panel to provide an audible alert. This is accomplished by toggling output Port A, bit 1, based on a separate timer. Transistor Q1 is a simple emitter follower used to increase the speaker current. When Q1 is not switching, the transistor current is limited by the collector resistor R1. This prevents high current in Q1 and the speaker if the microcontroller port output is left high when any sound output stops.

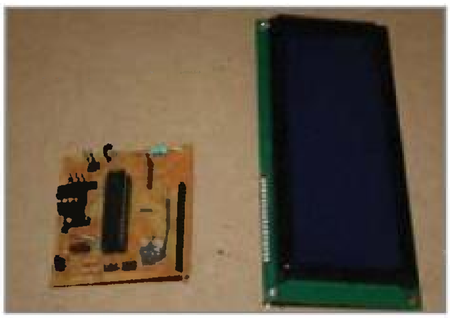


Photo 2—The home-fabricated board uses the 16-pin header to ride on the LCD. The remaining headers are for RS-485, push buttons, and a speaker.

ASSEMBLY

The LCD module is a MikroElektronika WH2004L-TMI-ET# LCD module large-character backlit display board. Photo 2 shows the LCD and the microcontroller board.

The display was mounted in a three-gang wall box, which had the same interior width the same as the LCD module. I only needed to provide some small angle brackets to secure the board to the box and cut out a window in a three-wide blank wall box cover.

For the cover, I used a nibbling tool to carefully trim a hole in a three-gang blank wall plate. It generated a pile of little white plastic dropping and a few blisters on my hands, but I managed to get the LCD bezel to fit perfectly. The lower screw holes were drilled out to accommodate the three push-button switches. Photo 3 shows the two boards attached to each other and it shows the wires attached to the switches mounted in the faceplate. Photo 4 shows the unit assembled and ready to be placed in the wall.

SOFTWARE

If you've read about any of the other devices I built, you will notice that with each device, I've used larger and larger microcontrollers. This came about mostly because I wanted to take advantage of higher-level languages when writing the code. I started with the Microchip Technology assembler for the PIC16 then moved on to the CCS compiler. Next I moved up to the PIC18F2221 microcontroller and wanted to use it here as well. However, this project's code is substantially bigger and more complex. On my first attempt to compile the software, the microcontroller ran out of flash memory. As I examined the compiler-generated map file, which shows the function addresses and size, I realized I would need more program memory if I wanted to continue adding functionality. So I jumped up to the Microchip PIC18F2523, which has 32 KB of ROM. This would accommodate any new features I might add for some time to come. Also, although I had already designed the PCB, plugging in an upgraded microcontroller was not a problem because Microchip keeps the same pinout across its similar devices. So, the PIC18F2221's 28-pin DIP now also works for the PIC18F2523.

This project used seven source code files, all written in C, and was built with Microchip's MPLAB C18 C compiler. Each

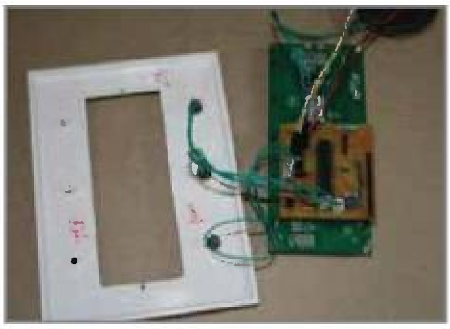


Photo 3—The microcontroller board is attached to the LCD module with its push buttons attached to a wall plate. I used a nibbling tool to cut out the LCD viewport. After the bezel was installed, the appearance dramatically improved.

Photo 4—The unit is assembled and inserted into a three-gang wall box, ready for mounting. There are three screws holding the faceplate on the top and the push buttons are screwed to the faceplate and slid into the wall box's screw holes.

Listing 1—This is the mainline loop for processing the MVC design. Each module performs only its designated tasks and returns in the minimal amount of time. In large-scale systems, these are broken up into threads or performed on different processors.

```
while (1) {
    ClrWdt();              /* Make sure we stay on top of things */

    ProcessController();  /*  Anyone tells us what to do? */
    ProcessModel();       /* Any new data from the network? */
    ProcessView();        /* Any changes to the display? */
    Delay10TCYx(1);       /*  Whew! That was a lot of work, need a break */
}
```

source module provided specific functions that are grouped together. For example, the LCD.C module handles the LCD setup and a collection of functions that will clear the display, jump to specific cursor locations, and write text strings. Setting up the LCD module to use 4 bits enables me to reserve another 3 bits from the same 8-bit port to handle signaling. This way, only one port was used to talk to the LCD module. This source code can be reused on other devices with fewer ports available to them. In fact, this source was originally ported up from use in other PIC16F projects.

Because of this project's complexity, I decided to use a different software design paradigm, the MVC, which is common in object-oriented programming languages and projects. But don't get discouraged by that fact. This philosophy can be applied to regular C or even Assembler. That's because MVC is about keeping software pieces' behaviors "encapsulated" and easier to maintain and enhance.

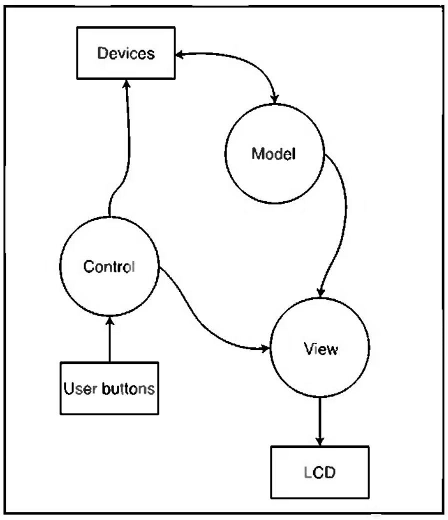


Figure 2—In this MVC interaction, the user input is read by the controller and used to send commands into the network and change the view. The model information is passed to the view so it can display selected information to the user.

MODEL-VIEW-CONTROLLER

A MVC design is a software architecture that segments the source modules into three primary components, each of which is responsible for only one specific operation. As the name implies, the model is the component that contains all the system's information. The view is responsible for drawing the pieces the user is presented, anything that they can observe. The controller is responsible for obtaining input from the user. In a PC-based system, these modules typically run in individual threads. However, in a microcontroller environment, processing can be accomplished with an RTOS, or simply by having each of the modules execute an entry function in a round-robin fashion and spending a minimal amount of time executing that function. For simplicity, I chose the latter (see Listing 1).

The model is responsible for processing the messages from other network devices. The controller sends out commands but does not obtain any responses. If needed, the controller can also tell the view to provide a different screen. However, no feature like that has yet been implemented. Figure 2 sums up which software module talks to which.

THE MODEL

The model is the core piece of the software design. It is where the data is aggregated from all the network devices. The model can be a state machine or, in this design, it is simply a collection of the data that represents the states or persistence information about the network nodes. In higher-level systems, the model is usually a server database. But, in this case, it is just the devices' statuses (e.g., the garage, the lights, and the GPS), which are in my house.

The data is collected by sending out polls at fixed time intervals and processing responses. The polls are fired off and forgotten about. The routine sends any outgoing messages, looping in a `wait` for the UART to complete transmission to switch the differential driver back to Receive mode. This is done through Port A, bit 2. Waiting for this to complete is not a big concern because at a 19,200-bps rate, a transmission's duration is 52 µs per bit and the average data block is 5 bytes (i.e., 50 bits). So, the entire message can be sent in about 2.6 ms. The next time the model code is executed, a response is processed if it is present in the input buffer.

As responses are received via the interrupt routine, the data is stored in a receive buffer, and a semaphore flag is set to indicate that the receive buffer needs to be processed by the model at each pass. This is usually status information and is recorded into the model in an array of structures. Currently, there are five blocks in the array, one for each network device, excluding the control panel itself. Table 1 shows the structure layout. The 3-byte buffer area is simply a block of bytes used to store other pieces of information. The information depends on the device polled. Other messages that are received during operation may arrive without being requested. So far, this is just a time message, but it could easily be enhanced to include other information.

Listing 2 is a section of code showing

Name	Size
Status	1 byte
Reset counts	4 bytes
Buffer	3 bytes

Table 1—This is the model's data layout. There is one block for each device monitored on the network.

ASSEMBLY LANGUAGE ESSENTIALS

Circuit Cellar's first book, ***Assembly Language Essentials***, is a matter-of-fact guide to Assembly that will introduce you to the most fundamental programming language of a processor.

A Guide to Powerful Programming for Embedded Systems
Assembly Language Essentials
Larry Cicchinelli
CIRCUIT CELLAR

Author Larry Cicchinelli provides readers with:

- An introduction to Assembly language & its functionality
- Essential terminology pertaining to higher-level programming languages & computer architecture
- Important algorithms that may be built into high-level languages — multiplication, division, and polynomial evaluation
- Overview of Interrupt Service Routines
- Free, downloadable Assembler program ...

and more!

On Sale NOW!

$47.50

WWW.CIRCUITCELLAR.COM • WWW.CC-WEBSHOP.COM

CIRCUIT CELLAR
THE WORLD'S SOURCE FOR EMBEDDED ELECTRONICS ENGINEERING INFORMATION

the processing within the model. The functions are clearly named. The amount of time it executes within the function is intended to be kept to a minimum.

THE VIEW

The view is the most straightforward of the software modules. It is responsible for updating the LCD with whatever the user wants to see. To prevent constant updates to the LCD when it is not needed, a copy of the model data is kept in the code's view section. When the view is executed, it compares the view data with the model data and updates the LCD any place where the two differ. The view is updated to match the model. This takes up RAM storage but prevents annoying updates to the display. The communication source code that handles the LCD is in a single file and handles the initializing, serializing of the 8 bits into 4, testing the busy status, and triggering the enable line. The backlight LED is also handled in this module. The controller tells the view to turn it on by asserting register C, bit 5. The view will simply add 10 s to the current second counter and remember it. When the counter matches the current second from the clock, the backlight is turned off.

Another view feature is that it displays two different screens. The first screen is a primary screen that shows me the lights and garage doors' statuses. The view displays this information by default, unless told to do otherwise. When instructed by the controller, the view shows the reset reasons' current values, which are obtained from one of the devices connected to the network. The view actually draws a template of the fields then fills in the values later.

THE CONTROLLER

The controller is simply a state machine that reacts to user input and executes state changes. The user input utilizes three push buttons. When a button press is detected, it notifies the view to turn the backlight on. If the buttons are being pressed, the view

Listing 2—This is a part of the code that executes in the model. The comments and function names explain it fairly well. The step to process time simply checks the garage door's status and activates the alarm if needed. The actual alarm sound is driven by an interrupt timer.

```
void ProcessModel() {
        /* If the time has changed, see if we need to do a poll */
        if (sec != nowsec) {
                nowsec = sec;
                /* Build any outgoing requests */
                ProcessRequests();
                /* If there is a message to send, fire it off */
                if (messageout[3] != 0) SendMessage();
                ProcessTime();
        }
                /* Check for any inbound message */
        if (cmd_state == CMD_COMPLETE) {
                ProcessIncomingMsg();
                cmd_state = CMD_WAITING;
        }
}

void ProcessTime(void) {
        if ((hour == 21) &&
                (min == 0)  &&
                (sec == 15)) {
                if ((client[GARAGE1].status & 0x01) ||
                        (client[GARAGE2].status & 0x01)    ) {
                        ActivateAlarm(1);
                }
        }
}
```

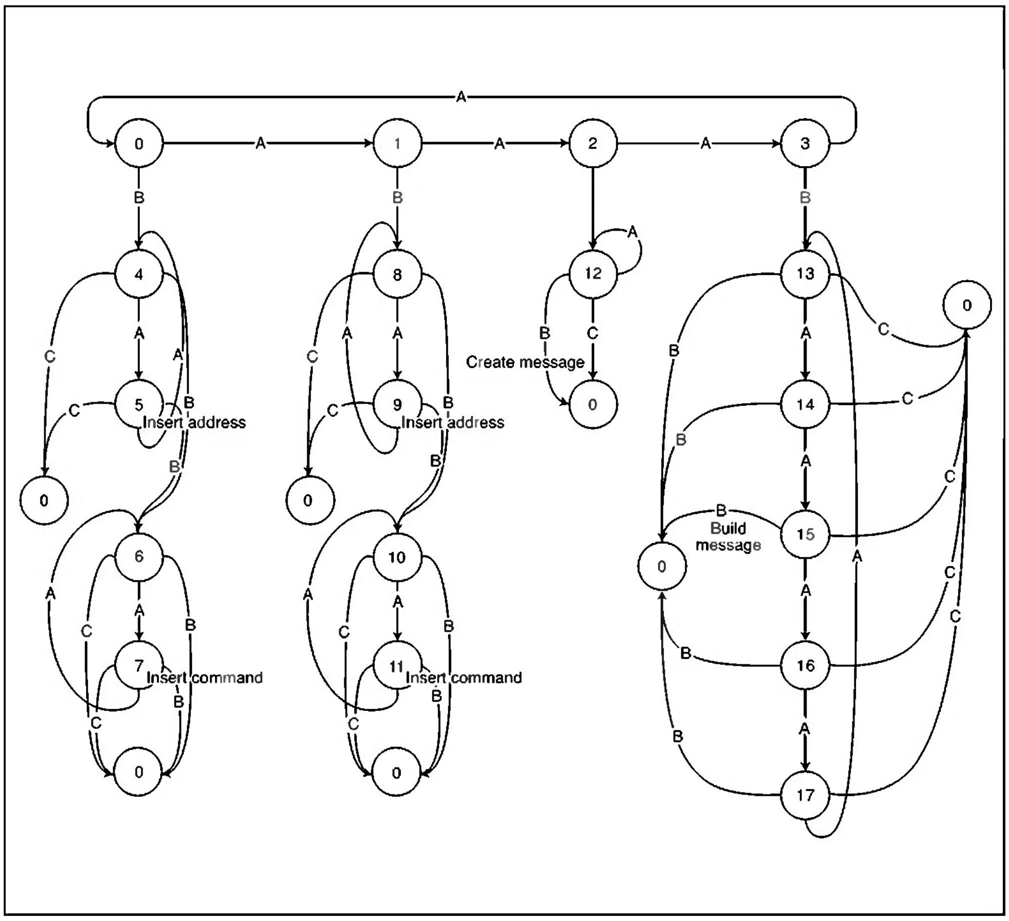


Figure 3—These are the user button state machines. These states do not reflect changes that are made to the view because the view only displays information from the model. Therefore, the controller is responsible for updating the display portion that decides what each button will do.

sensors
expo & conference
www.sensorsexpo.com

Sensing Technologies Driving Tomorrow's Solutions

Registration is Open for Sensors 2013
Sign up today for the best rates! www.sensorsexpo.com

Pre-Conference Symposia: June 4, 2013
Conference & Expo: June 5-6, 2013
Donald E. Stephens Convention Center • Rosemont, IL

CIRCUIT CELLAR elektor
Special Subscriber Discount!
Register with code A306L for $50 off Gold and Main Conference Passes!*

Innovative Applications. Expert Instructors. Authoritative Content. Tomorrow's Solutions.

Register today to attend one of the world's largest and most important gatherings of engineers and scientists involved in the development and deployment of sensor systems.

What's Happening in 2013:

"I come to Sensors Expo to see new things, learn about new technologies, and get my sensing questions answered."
—Nick Plessas, N.D.P.

- › Tracks:
  - Big Data & Analytics
  - Cutting-Edge Sensing Applications & Innovations NEW
  - Energy Harvesting
  - Gas Sensing NEW
  - MEMS
  - Novel Approaches to Measurement & Detection NEW
  - Remote Monitoring
  - Sensors @ Work NEW
  - Wireless Sensing Solutions
- › Full day Pre-Conference Symposia
- › Technology Pavilions on the Expo Floor
  - Big Data & Cloud
  - Energy Harvesting
  - Intelligent Systems NEW
  - MEMS
  - Wireless
- › Co-location with Intelligent Systems Source NEW
- › Best of Sensors Expo 2013 Awards Ceremony
- › Networking Breakfasts
- › Welcome Reception
- › Sensors Magazine Live Theater
- › And more!

Visionary Keynotes:

Connecting with the Emerging Nervous System of Ubiquitous Sensing
Dr. Joseph A. Paradiso
Associate Professor of Media Arts and Sciences, Responsive Environments Group
MIT Media Laboratory

Sensors in Space: The Robotic Exploration of Mars and its Environment
Dr. Raymond Arvidson
James, S. McDonnell Distinguished University Professor, Washington University
Participating Scientist, NASA's Mars Curiosity Rover Mission

Register at www.sensorsexpo.com or call 800-496-9877

OFFICIAL PUBLICATION: sensors

INDUSTRY SPONSORS: CIRCUIT CELLAR elektor

NEW CO-LOCATION: ISS Intelligent Systems Source

*Discount is off currently published rates. Cannot be combined with other offers or applied to previous registrations.

Table 2—This is the relationship between the controller states and the menu text for button A on the panel.

is informed and continues to extend the backlight time limit by 10 s.

Figure 3 shows the controller's state machine design. The state numbers are written in each circle and used in a switch statement to track the change direction from one state to the next. The transition lines are labeled A, B, and C to correspond to the three buttons used as inputs. The text above button A changes depending on the state. The text above button B is always "Select." The text above button C can be the time, the "Back" option, or "STOP," which is used to stop the alarm function (described later). If a circle does not have a state change line for a button, it means the button is ignored (or turns back on itself, as shown in the A line for State 12).

The state numbers in the circles are mapped to the text that appears on the display above button A. Table 2 shows the relation between the states and the button. As the state changes, the controller is responsible for updating the LCD's bottom row of the LCD.

States 4–12 are menu options that are used send commands to devices. When the Select button is pressed in one of these states, the controller creates and sends a message. The message is built in memory and sent in a "fire and forget" concept. No reply is expected. The next time the model polls one of the devices, it obtains the new state and updates the status screen.

States 13–17 are used to request that the view display the current count of reset reasons from the device. Currently, the devices are simply listed by their hexadecimal addresses. When one of those states is executed by pressing the Select button, the controller builds a request message for that device that asks it to report the number of resets due to power up, watchdog resets, brown out, and memory clear. This message is also sent in a "fire and forget" way. However, the reply will be picked up by the model when the response it sent. The controller will also call a function in the view telling it to change the display to the reset status, and which device to show. Pressing the C button when the unit is in 0 state causes it to tell the view to return to the normal status screen.

The controller sets the text above each button, which acts as a menu. In theory, the view could be responsible for this part of the display. However, that also requires the view to maintain a copy of the state information that is used by the controller. Then, the controller must interrogate the view to determine the current button action. While all this is possible, and technically correct, the tradeoff between keeping the components encapsulated with the increase in complexity compared to considering the microcontroller resources must also be taken into consideration.

KEEP THINGS SIMPLE

MVC is a useful programming paradigm, and I hope this has been a useful introduction. There are many books and web resources on the subject, and even more disagreements about which units can talk to each other and what direction information is permitted to flow. In my experience, if you are spending any time fighting a battle over how the things are allowed to interact, you are not doing the right thing. And I've seen that a lot. The right thing is to make it work for the user.

The advantage in the MVC paradigm is to keep things simple and organized when enhancing the device's function. For example, I added a new feature after accidentally leaving the garage doors open late at night. I included a simple test that examined the current time, and at 9:00 P.M. and 15 s, if either of the garage doors were still open, an alarm would sound. In this case, the alarm was just a sweeping audio signal created by a microcontroller's interrupt driver timer. While the alarm is on, the controller displays the word STOP above button C, and enables the user to halt the alarm condition. My wife asked if the control panel could just chose to close the garage doors for us. I said, "Sure. But what if we happen to be outside at the time and get locked out?" Too much automation isn't always a good thing.

After working in software development for 20 years, Scott Weber (scotty42@csweber.com) is tired of the direction the PC software world is taking. He is now striving to complete his Electrical Engineering degree at the University of Texas Arlington.

RESOURCES

S. Weber, "Time Broadcasting: A GPS-Based Time Server for the RS-485 Network," *Circuit Cellar* 268, 2012.

———, "Controlling Access with a Proximity Card: Opening the Door to Manchester Encoding," *Circuit Cellar* 269, 2012.

Wikipedia, "Model-View-Controller," http://en.wikipedia.org/wiki/Model_view_controller.

SOURCES

PIC18F2221 and PIC18F2523 Microcontrollers and MPLAB C18 C compiler
Microchip Technology, Inc. | www.microchip.com

WH2004L-TMI-ET# LCD ModuLE
MikroElektronika | www.mikroe.com

by Brian Millier (Canada)

G-Code CNC Router Controller (Part 1)

G-Codes and Motion Control

Are you interested in building a microcontroller-based G-code controller for a CNC router? This article explains the basics of G-code language and how to design hardware and firmware to control the stepper motors.

My article "CNC Router Design: A Look at Computer-Controlled Machinery" (*Circuit Cellar* 248, 2011) chronicled my DIY CNC router "build." This was my first foray into computerized machining. I have been satisfied with the machine's operation over the past two years. Since I also retired two years ago and lost ready access to the machine shop where I worked, having the capacity to do my own milling has become quite important.

As I mentioned in my original article, I used a PC with ArtSoft's Mach3 CNC software to run my CNC router. This combination is probably the most commonly used among CNC "hobbyists," because even PCs that are now considered obsolete are still powerful enough to run the Mach3 software.

ArtSoft's Mach3 software is reasonably priced (less than $200) and well-documented. And, and since so many people use it, there are a lot of support resources available. Given these advantages, I had no reason to change my setup. However, as I was using the machine, I became increasingly curious about the way the G-code (used to program the machine) was converted into three-axis motion using the machine's three stepper motors. It seemed like the PC/Mach3 combination could be replaced by a microcontroller-based unit. I thought designing such a controller would be a challenging project and decided to give it a try. Photo 1 shows the end result.

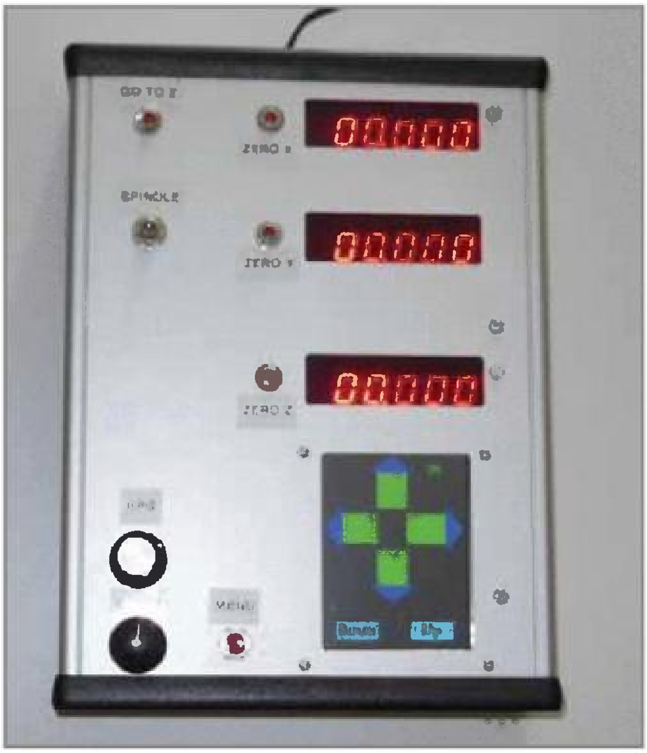


Photo 1—Atmel's ATmega88 and ATmega1284 microcontrollers are at the heart of the CNC controller.

I want to be upfront and say this is probably not the most practical project I have ever done. You can usually pick up a used PC for free, and the Mach3 software is professional-grade and handles much more complex G-code programs than my DIY controller will. However, it did provide me with a challenging programming task, and I learned a lot about designing a program with many concurrent tasks, all of which are time critical. Even if you are not interested in building such a controller, you may find some of the techniques and tricks I used to provide the multi-axis stepper-motor motion useful.

This project involved two main tasks. The first was to understand the G-code language used to program CNC machines well enough to write the firmware that would parse the G-code commands into something a microcontroller could use to control the stepper motors for each of the three axes. The second task was to design the hardware/firmware that would actually control the three stepper motors, all of which had to move synchronously at accurate, ramped speeds. First, let's take a brief look at G-code.

G-CODE

Many books have been written about G-code programming, and I doubt any one particular book would be comprehensive enough for all types of milling machines. The G-code language is arguably as complicated as any computer language and probably harder for a computer to interpret because its syntax is not as strictly defined as most computer languages. I will only touch on the basics in this article. (For more information, search online for a comprehensive list of tutorials and so forth about G-code.)

G-code is a group of single-letter mnemonic codes that are used to define all the necessary operations required by any CNC machine tool. Because many operations are required by different milling tools, the single-letter codes are expanded in function by a numeric value that is appended to that code. Parameters (e.g., the position to which you want the cutting head to move) are expressed as ASCII numeric strings, containing as few or as many decimal places of accuracy as are needed for the operation. G-code command files are basically text files, with a CR/LF termination between each line of code. More than one G-code command can be on a single line, and there can also be comment lines. Command lines are not normally numbered, but it is acceptable to do so for your own purposes. The position coordinates' order does not need to be specific (i.e., x, y, z). Not all target position coordinates need to be assigned with every command; only those that differ from the current position (in that axis) must be specified.

For the purposes of my controller, only linear motion commands (called linear interpolation in CNC) are implemented. The G-code language also contains commands for circular interpolation. These commands are more complex for the controller, so I did not include them.

That is not to say that this controller won't handle curves, arcs, and circles. Many programs that provide G-code output do so using only the linear motion commands and use many small linear motions to generate curves and arcs. All the programs I use fall into this category, so my controller can handle this.

Among the linear motion commands, I manage two main commands. The G00 code, which is called the rapid code, is meant to move the tool rapidly to a target position at a rate independent from (and normally higher than) the programmed feed rate. In rapid moves, the tool is not cutting and the intent is to quickly get it to the target position. If the target position is different in more than one axis, the tool does not necessarily move at an angle. However, it may do one axis after another, until the desired target position is achieved. The G01 code is used for normal linear moves in which the tool is cutting. Therefore, up to three axes may be simultaneously moving if the required cut is either a 2-D or 3-D (compound) angle.

In addition to the above commands, I also manage the M3 and M2 commands, which turn the spindle motor (i.e., the cutter) on and off, respectively. The last command I handle is the F command, which sets the feed rate for cutting operations.

The host controller firmware is given a G-code file to run after the operator picks one out of the directory list for the inserted SD card. It then parses that file one line at a time, loading up the proper x, y, and z coordinates, feed rate, and so forth. After it finds either a rapid or a linear interpolation motion command, it initiates motion from one to three axes, as dictated by the parameters it has just encountered. I'll describe how this is accomplished in the next section.

MOTION-CONTROL ELECTRONICS

While I was committed to using a microcontroller for the project, I still spent a lot of time trying to choose the best one for the job (with the criterion that I wanted to use a microcontroller in a DIP package because it would be easier to handle with the custom PCB I would design). Since I knew it was going to be a significant programming effort, I decided to use Atmel AVR microcontrollers so I could write the program in MCS Electronics's BASCOM-AVR, the language with which I have the most experience.

Admittedly, this severely limited my choices. It eliminated many sophisticated microcontrollers that could have handled all the tasks, including the timing-critical motion-control algorithms for three axes. However, it didn't seem reasonable to use an unfamiliar microcontroller to tackle a complex project such as this. After a lot of analysis, I decided the only way to achieve the critical timing needed for three-axis motion control was to dedicate an individual (i.e., slave) microcontroller for each axis. In addition to the three axis controllers, I would use a host microcontroller to read the G-code file, interpret it into motion commands for the three axes, and dispatch the tasks to the individual axis controllers. The host microcontroller would also be responsible for handling the various manual motion commands the user would enter while setting up the machine for the actual CNC milling job. It was cost effective to use the dedicated microcontroller approach. The slave axis controllers each use an Atmel ATmega88 and the host controller is an Atmel ATmega1284. The total cost of the four microcontrollers is about $15, which compares favorably to using a single, more complex microcontroller.

Another design choice I made early on involved the actual user interface. I needed some type of LCD screen to handle the various required functions. I could have chosen an inexpensive, alphanumeric LCD panel, but I decided to include a 4D Systems μLCD-32PT 3.2" thin film transistor (TFT) color touchscreen display with graphics capability. Using a touchscreen display meant I could eliminate many of the switches, as well as the numeric keypad that would normally be needed in this type of unit. TFT touchscreen displays are becoming inexpensive, while good-quality switches and numeric keypads are becoming more expensive.

Although it was tempting to display the three axis positions on the TFT display, this would have been less than satisfactory, as they would have been too small to comfortably read while using the machine itself. Therefore, I included a six-digit red

LED display for each axis, which is visible from many feet away.

The last major design choice involved the type of media I would use for the G-code files. There were two basic choices: USB flash drives or SD memory cards. I chose the latter as it is easy to interface SD cards to AVR microcontrollers using MCS Electronics's AVR-DOS and the microcontroller's SPI port.

While I have used Future Technology Devices International's VDrive2 Vinculum modules to interface to USB flash drives, they use a UART port and my host microcontroller only had two UART ports, both of which were allocated to other devices. (Note: The VDrive2 modules also contain a SPI port, but it uses a nonstandard protocol that I haven't yet used successfully.) On the PC side, there are small, inexpensive memory card adapters available that enable you to read and write many different memory cards via the PC's USB port. Figure 1 is a block diagram of the complete CNC controller showing the host and one of the three-axis controllers.

BASIC CNC MOTION CONTROL

CNC machines use two drive methods: Servomotor/position encoder or stepper motor. Professional CNC machines, particularly larger ones, generally use the servomotor/position encoder method, as it enables much faster axis motion. However, servomotor/position encoder drives are expensive. Stepper motors have markedly dropped in price and they are now much more powerful, for a given frame size, than ever before. I used Automation Technologies's KL23H276-30-8B stepper motors, which are rated for 282 oz-in of torque and cost less than $40.

I used Geckodrive's G540 four-axis digital step drive, which contains the high-power drivers for four motors and the specialized digital circuitry needed for microstepping (i.e., the ability to move the stepper motor in steps smaller than the 200 steps per revolution, for which the Keling motors are rated). The Geckodrive drivers are an industry standard for small CNC machines and work well.

For any CNC machine, you must determine how many step pulses are required per inch of linear travel along the axis involved. This is determined by the formula:

$$\text{Pulses per inch of linear travel} = \frac{\text{Motor Steps}}{\text{Revolution}} \times \text{Microstep Ratio} \times \text{Mechanical Ratio}$$

In my case, the motor steps are 200 per revolution. The G540 uses a microstep ratio of 10. You feed it 10 pulses for every discrete step of the stepper and the motor makes 10 microsteps for every 1.8° physical step (i.e., 360° for every 200 motor steps).

In my CNC router, the stepper motors' rotary motion is converted into linear motion using an Acme lead screw and an anti-backlash lead nut. Acme lead screws are commonly sold with 10 threads per inch (TPI). This would mean that the screw provided a linear motion of 0.1" per revolution.

However, there is another parameter called "start," which can be one, two, or five. A one-start Acme lead screw would most resemble the standard machine screw you are familiar with (i.e., there is one continuous thread running the length of the screw). A two-start Acme lead screw, on the other hand, would have two discrete threads, each one of which would start at 180° from the other (if you were to look at a screw's cross-section), and the thread's pitch would be 0.5 that of a one-start screw. A five-start screw would have five discrete threads, so a 10-TPI screw would, in this particular case, move 0.2" per turn of the screw. You would use a five-start Acme lead screw for higher speeds, or a one-start screw for more resolution. I chose the latter, so my mechanical ratio was 10 turns per inch of linear travel.

Therefore, using the preceding formula, I needed 20,000 pulses per inch of linear motion. Because the stepper motor is only capable of turning a couple of revolutions per second, under load, my machine is capable of less than 1" per second of linear motion. But that is sufficient for my tasks (e.g., cutting wood, plastic, and relatively thin aluminum cabinets and panels). In my microcontroller-based CNC-controller design, I aimed for a 25,000 per second maximum pulse rate, although this is somewhat higher than needed for my motor and lead screw setup.

To achieve good cutting results, the cutting tool's feed rate must be tailored to several variables

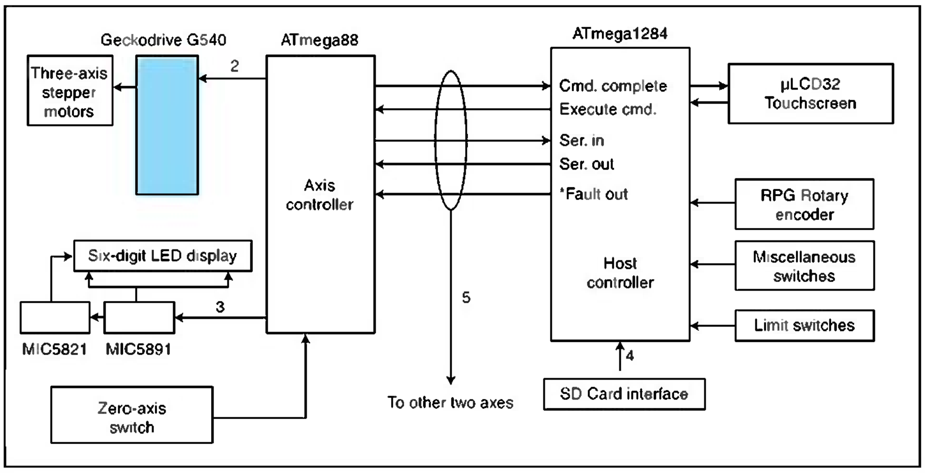


Figure 1—This CNC controller block diagram includes an Atmel ATmega1284 host controller, a 4D Systems µLCD-32PT touchscreen display, and Micrel MIC5821 and MIC5891 latched drivers, which are used to drive the LED displays. The Geckodrive G540 shown in blue is a separate commercial unit.

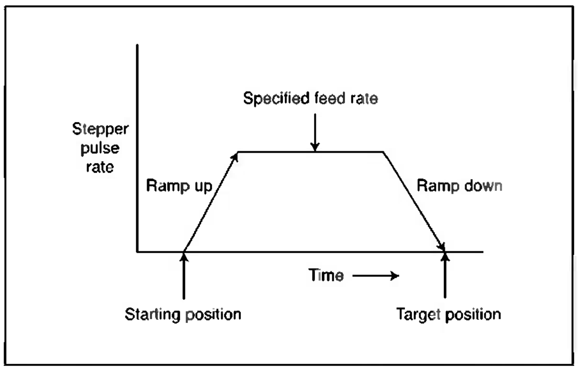


Figure 2—Stepper motors can run at modest speeds, but they cannot accelerate or decelerate instantaneously. Therefore, it is customary to use a trapezoidal stepping profile to accommodate this limitation.

(e.g., the type and thickness of the material being machined, and the cutting tool's rotational speed). There is a big difference in the feed rate you can use when cutting a groove in a thin piece of pine compared to a thick piece of steel. Because of this, it is important that you can vary the cutting tool's feed rate in each of the three axes. However, for the above reason alone, it is not important that an exact feed rate be achieved. You must achieve this feed rate approximately.

In practice, you must be able to cut at any angle and cut a curved profile. To do so, you must specify a feed rate for each axis that has a precisely defined ratio with respect to the other two axes. If not, the angle cut will be wrong or the curve will be incorrect. For this reason, it is essential that the pulse rate produced by the controller is accurate and of high resolution. While a PC containing a CPU with a clock rate in the hundreds (or thousands) of megahertz can accomplish this for three axes simultaneously, I found it was also possible to achieve this using one low-cost microcontroller per axis.

While it has not been previously mentioned, it is not possible to run a stepper motor at anything other than really slow speeds, unless you gradually ramp up the speed at the start of a programmed move, and then gradually ramp down as you approach the end of that move. This is most commonly done using a trapezoidal stepping profile, as shown in Figure 2. Therefore, in addition to requiring an accurate, high-resolution source of stepper-motor pulses, those pulses must be delivered with up and down ramps, as shown in Figure 2.

In Part 2 of this article series, I'll describe the design and provide details about the design's two functional blocks: The axis controller block (powered by three small microcontrollers) and the host controller block (powered by a more powerful microcontroller). ■

Brian Millier (bmillier1@gmail.com) runs Computer Interface Consultants. He was an instrumentation engineer in the Department of Chemistry at Dalhousie University (Halifax, Canada) for 29 years.

RESOURCES

J. Mikeln, *BASCOM-AVR Programming*, AX elektronika, 2012.

B. Millier, "CNC Router Design: A Look at Computer-Controlled Machinery," *Circuit Cellar* 248, 2011.

SOURCES

μLCD-32PT Touchscreen display
4D Systems, Ltd. | www.4dsystems.com.au

Mach3 CNC Control software
ArtSoft USA | www.machsupport.com

ATmega88 and ATmega1284 Microcontrollers
Atmel Corp. | www.atmel.com

KL23H276-30-8B Stepper motor
Automation Technologies, Inc. | www.automationtechnologies.com

VDrive2 Vinculum modules
Future Technology Devices International, Ltd. | www.ftdichip.com

G540 Four-axis digital step drive
Geckodrive, Inc. | www.geckodrive.com

BASCOM-AVR and AVR-DOS
MCS Electronics | www.mcselec.com

MIC5821 and MIC5891 Latched drivers
Micrel, Inc. | www.micrel.com

CD/DVD-ROMs

140 Min. Video Presentation

DVD Masterclass Feedback in Audio Amplifiers

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

ISBN 978-1-907920-16-5 • $40.20

More than 75,000 components

CD Elektor's Components Database 7

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

ISBN 978-90-5381-298-3 • $40.20

Bestseller!

A whole year of Elektor magazine onto a single disk

DVD Elektor 2012

The year volume DVD/CD-ROMs are among the most popular items in Elektor's poduct range. This DVD-ROM contains all editorial articles published in Volume 2012 of the English, American, Spanish, Dutch, French and German editions of Elektor. Using the supplied Adobe Reader program, articles are presented in the same layout as originally found in the magazine. An extensive search machine is available to locate keywords in any article. With this DVD you can also produce hard copy of PCB layouts at printer resolution, adapt PCB layouts using your favourite graphics program, zoom in / out on selected PCB areas and export circuit diagrams and illustrations to other programs.

ISBN 978-90-5381-273-0 • $37.90

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

www.elektor.com/store

elektor

Elektor US
111 Founders Plaza, Suite 300
East Hartford, CT 06108
USA
Phone: 860.289.0800
Fax: 860.461.0450
E-mail: order@elektor.com

Improved Radiation Meter

This device can be used with different sensors to measure gamma and alpha radiation. It is particularly suitable for long-term measurements and for examining weakly radioactive samples. The photodiode has a smaller sensitive area than a Geiger-Müller tube and so has a lower background count rate, which in turn means that the radiation from a small sample is easier to detect against the background. The optional PC-based software displays the energy spectrum, permitting a very detailed analysis to be carried out.

Kit of parts incl. display and programmed controller

Art.# 110538-71 • $57.30

Bestseller!

Embedded Linux Made Easy

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

Populated and tested Elektor Linux Board

Art.# 120026-91 • $93.30

Kits & Modules

by Tom Cantrell (USA)

Weatherize Your Embedded App

Why water your lawn when a rainstorm is imminent? Web-enabled devices can take advantage of weather conditions and online forecasts to fine-tune their operation. All it takes are a few chips and a few kilobytes of code to see what the meteorological future holds. This article details how to use a Texas Instruments MSP430 microcontroller and a WIZnet W5200 smart Ethernet chip to access National Weather Service forecast data.

"How's the weather?" Since the dawn of time, that's been a good conversation starter. Despite our best efforts to seek shelter, weather still matters, and it can still have life and death consequences. No wonder the weather has its own TV channel.

These days, we have a lot more options than watching TV (or looking outside the cave). The science of weather measurement and forecasting has taken off (literally, in the case of weather satellites). Now we have a lot of answers to the "How's the weather?" question, but what about our machines?

My interest in the matter is a bit geocentric. Where I live in California, we have a lot of green lawns, but for much of the year, there is not enough rainfall to keep them that way. Even during what passes for the winter "rainy season," there can be long stretches of warm and dry weather. Sprinklers are a must.

The problem is the typical residential sprinkler, operating on a fixed-time schedule, is rather simple-minded. Despite ever-present calls to conserve water, when rainstorms do arrive, you can drive around any neighborhood and, sure enough, you'll find sprinklers running.

I try to keep ahead of the game. If I know rain is in the forecast, I'll try to remember to turn off the sprinkler timer and hope I don't forget to turn it back on (emphasis on the words "try," "remember," and "forget"). Admittedly, my manual override results are marginal at best. The irony is, the sprinkler controller has a "computer" inside, yet I end up serving the machine rather than vice versa.

The hassles of manually fiddling with my sprinkler controller, seeing an ever-escalating water bill, and being frustrated with the wasted runoff got me thinking. How hard would it be to craft a weather-savvy sprinkler controller?

DATA IS KING

This project's success depends on the availability of accurate, reliable, and timely weather data at a low cost, or better yet, free of charge. Fortunately, here in the US, we have the National Weather Service from the National Oceanic and Atmospheric Administration (NOAA), which fits all those bills (see Figure 1). Hundreds of facilities, thousands of employees, and billions of dollars are devoted to dissecting national weather data 24 h a day, seven days a week, 365 days a year. And it's all "free," at least in the sense that someone else (i.e., taxpayers at large) is paying for it.

Updated hourly, the current conditions and forecast are available at www.weather.gov. It's just a matter of getting at the data. I could use a PC or Linux/real-time operating system (RTOS) single-board computer (SBC), but that seems like overkill. Instead, I decided to see how far I could get using just a low-cost microcontroller,

Figure 1—This map shows the location of the National Oceanic and Atmospheric Administration (NOAA) facilities in the continental US. (Image courtesy of the NOAA, www.st.nmfs.noaa.gov/documents/NOAA_Facilities.ppt)

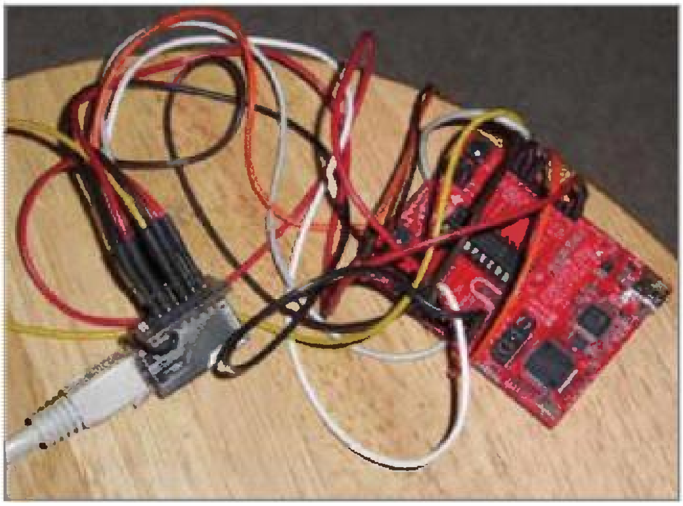


Photo 1—The TI LaunchPad connects to a WIZnet Wiz820io.

which was a perfect excuse to play with the Texas Instruments (TI) MSP-EXP430G2 LaunchPad.

It may look like a typical demo board, but the LaunchPad has a lot going for it, starting with the TI MSP430 Value Line microcontrollers it supports (see Photo 1). The lineup encompasses a range of MSP430s that offer low-cost, mainstream features. Consider the MSP430G2533 that comes with the LaunchPad. It delivers good performance (16 MHz) and a healthy complement of 16-KB flash and 512B RAM. But the I/O features are where the "value" really begins. The MSP430G2533 includes an eight-channel, 10-bit, 200-ksps ADC (with internal reference and autoscan), and an eight-channel analog comparator. It also has dual 16-bit timers with three capture/compare registers and two hardware serial ports covering all the options (e.g., UART with a baud-rate generator/LIN/IrDA, SPI, I^{2}C). There's a versatile phase-locked loop (PLL) clock generator and a factory-calibrated on-chip oscillator, so a crystal may not be necessary. The LaunchPad board adds an emulator front-end (another MSP430) and USB interface that hooks your PC into the microcontroller's on-chip debug hardware and in-system programmable flash. All this is available in a chip that costs less than $1 in volume.

But it gets even better. TI offers a free code-size-limited version of Code Composer Studio (CCS), its flagship C compiler and Eclipse integrated development environment (IDE). It has a 16-KB code size limit, which matches the MSP430G2533's flash size. Oh, did I mention the LaunchPad board itself costs just $4.30? "Value Line," indeed!

EASY ETHERNET

Since grabbing the weather service webpage once an hour doesn't push the networking envelope, a software TCP/IP stack might be an option. I researched online and found TI's "MSP430 Internet Connectivity" application report. I also found an open-source stack, "Adam Dunkel's uIP on the Olimex EasyWeb2 and LPC-E2124," which is derived from the application report. I quickly realized the software stacks needed higher-end MSP430s with more RAM than the Value Line chips offer. Also, the application report and stack Ethernet drivers target a transceiver with a byte-wide interface that needs a lot of pins.

Given so few pins and little memory to work with, I took the easy way out and used a WIZnet Wiz820io Ethernet module, which combines its W5200 smart Ethernet chip and a HanRun Electronics HR961160C surface-mount RJ45 jack on a breadboard-friendly (0.1" centers) PCB. An image of the pinout is available on *Circuit Cellar*'s FTP site.

The Wiz820io is an easy add on since it has a simple SPI and runs on the same 3.3-V supply as the MSP430G2533 (see Photo 1). Besides power and ground, it only takes six microcontroller pins to make the Wiz820io connection, the SPI (MISO, MOSI, SCLK, and nSS), a reset line (nRESET), and a power-down control (PWDN). The Wiz820io also has an interrupt output (nINT) but I'm not using it yet.

The Wiz820io module's W5200 chip includes the hardware TCP/IP protocol processing and large RAM buffers (transmit and receive, 16 KB each) configurable to support up to eight simultaneous connections (i.e., "sockets"). I'm counting on the extra intelligence to offload the network burden and make the microcontroller's job easier.

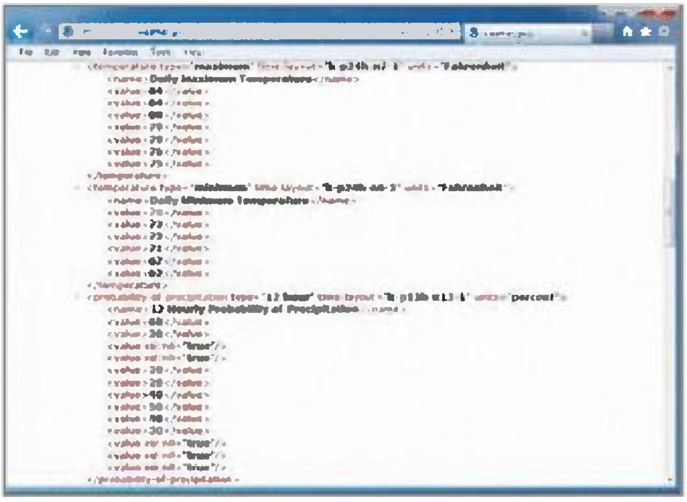
X(ML) MARKS THE SPOT

If you visit the following URL: http://forecast.weather.gov/MapClick.php?lat=40.71435&lon=-74.0059731&unit=0&lg=english&FcstType=dwml, you'll be taken to an XML data page showing New York City's forecast and current conditions. I'm using NYC for testing, since the weather there is more interesting than it is in California. Feel free to change the latitude and longitude values to your own (US) location.

As you can see, the webpage has a lot of information, including current conditions and multi-day forecasts for maximum and minimum temperature and probability of precipitation (see Photo 2). It seems pretty simple after all. Simply connect to the weather server, issue the `GET` request, retrieve the webpage, and sift through the XML data to extract information. Now I will explain how to do that.

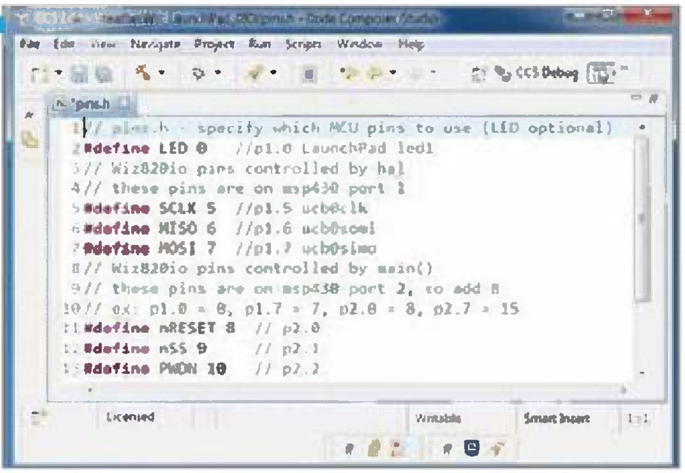
CODE REVIEW

The best way to explain what's going on is by going through some of the code. But first, there is a caveat. Those of you who



```
<temperature type="maximum" time-layout="k-p24h-n7-1" units="Fahrenheit">
  <name>Daily Maximum Temperature</name>
  <value>84</value>
  <value>84</value>
  <value>88</value>
  <value>79</value>
  <value>79</value>
  <value>76</value>
  <value>75</value>
</temperature>
<temperature type="minimum" time-layout="k-p24h-n6-2" units="Fahrenheit">
  <name>Daily Minimum Temperature</name>
  <value>70</value>
  <value>73</value>
  <value>73</value>
  <value>71</value>
  <value>67</value>
  <value>62</value>
</temperature>
<probability-of-precipitation type="12 hour" time-layout="k-p12h-n13-1" units="percent">
  <name>12 Hourly Probability of Precipitation</name>
  <value>60</value>
  <value>30</value>
  <value xsi:nil="true"/>
  <value xsi:nil="true"/>
  <value>30</value>
  <value>20</value>
  <value>40</value>
  <value>30</value>
  <value>90</value>
  <value>30</value>
  <value xsi:nil="true"/>
  <value xsi:nil="true"/>
  <value xsi:nil="true"/>
</probability-of-precipitation>
```

Photo 2—This is a screen shot of the National Weather Service XML data page.



```
// pins.h - specify which MCU pins to use (LED optional)
#define LED 0    //p1.0 LaunchPad led1
// Wiz820io pins controlled by hal
// these pins are on msp430 port 1
#define SCLK 5  //p1.5 ucb0clk
#define MISO 6  //p1.6 ucb0somi
#define MOSI 7  //p1.7 ucb0simo
// Wiz820io pins controlled by main()
// these pins are on msp430 port 2, so add 8
// ex: p1.0 = 0, p1.7 = 7, p2.0 = 8, p2.7 = 15
#define nRESET 8  // p2.0
#define nSS 9     // p2.1
#define PWDN 10   // p2.2
```

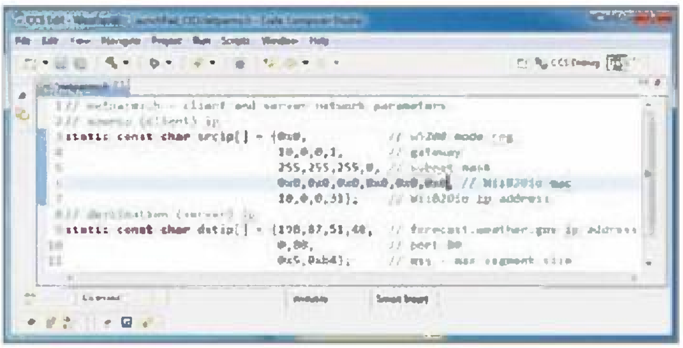
Listing 1—The `pins.h` code defines the microcontroller pins used to connect the WIZ820io Ethernet module.

are C programming experts and/or networking experts and/or MSP430 experts will soon realize I'm none of the above. I guess that's the point. You don't have to be an expert to do something useful.

Let's start at 50,000'. The overall project consists of three header files (`pins.h`, `netparms.h`, and `get.h`) and two code files (`hal.c` and `main.c`). I divided it up this way so the project can be ported to different hardware (e.g., MSP430 variants or another microcontroller brand) and a particular network without having to change `main.c`.

`pins.h` defines the six microcontroller pins required for the hardware connection to the Wiz820io and an (optional) seventh pin for an LED to monitor activity (see Listing 1). I used an LED included on the LaunchPad.

Listing 2 shows `netparms.h`, which specifies the IP addresses and such for my (i.e., client) side of the connection (`srcip`) and



```
// netparms.h - client and server network parameters
// source (client) ip
static const char srcip[] = {0x0,                      // w5200 mode reg
                             10,0,0,1,                 // gateway
                             255,255,255,0,            // subnet mask
                             0x0,0x0,0x0,0x0,0x0,0x0,  // Wiz820io mac
                             10,0,0,31};               // Wiz820io ip address
// destination (server) ip
static const char dstip[] = {198,87,51,48,             // forecast.weather.gov ip address
                             0,80,                     // port 80
                             0x5,0xb4};                // mss - max segment size
```

Listing 2—The `netparms.h` code designates the IP addresses for the connection's client (`srcip`) and server side (`dstip`).

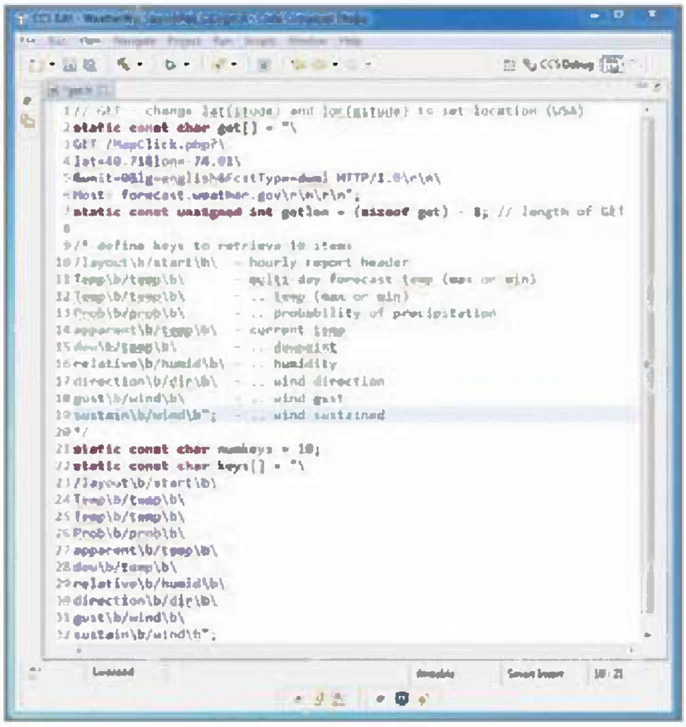
the weather service (i.e., server) side (`dstip`). A real app might want to add DHCP and DNS to dynamically set the addresses, but for experimentation, hardwiring them is fine. To get a current IP address for the weather report, simply ping forecast.weather.gov and plug it into `dstip`. You'll also need to come up with your own MAC address. I used one from an old LAN card.

`get.h` packages the weather service URL you use with a web browser into an HTTP `GET` request (see Listing 3). I used the line-continuation character "\" to split out the latitude and longitude. Just change them to your location of interest. Following the `GET` request are the "keys" that identify which data items to retrieve, as I'll explain shortly.

`hal.c` is a hardware abstraction layer that, along with `pins.h`, encapsulates all the hardware-specific aspects (see Listing 4). Like a BIOS, it includes low-level routines that `main.c` calls to initialize the hardware, drive the pins, output to display, and so forth.

As previously described, it is simply a matter of issuing the `GET` statement then searching the webpage to capture and display the particular data of interest (see Listing 5, `main.c` code).

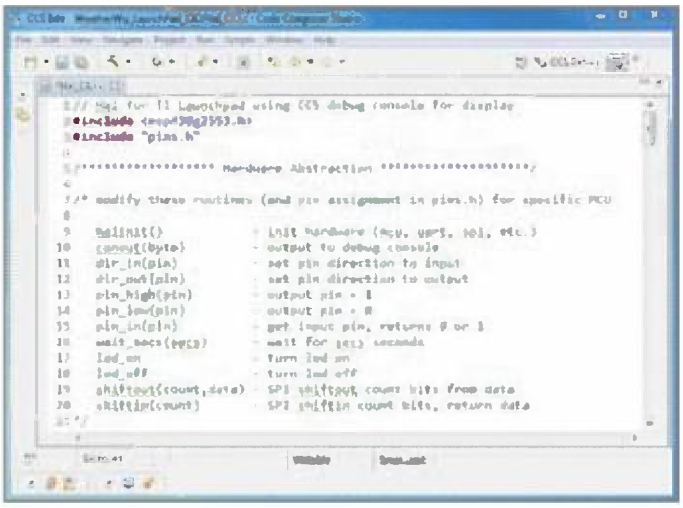
Now refer back to the keys definition in `get.h` to see how the search and capture works. The slashes make it seem more complicated than it really is. I'm retrieving 10 data items from the webpage, so there are 10 entries each consisting of a start and finish key separated by a backspace ('\b', 0x08). The `Searchweatherpage` routine finds a start key then captures data that appears between subsequent ">" and "<" pairs until



```
// GET - change lat(itude) and lon(gitude) to set location (USA)
static const char get[] = "\
GET /MapClick.php?\
lat=40.71&lon=-74.01\
&unit=0&lg=english&FcstType=dwml HTTP/1.0\r\n\
Host: forecast.weather.gov\r\n\r\n";
static const unsigned int getlen = (sizeof get) - 1; // length of GET

/* define keys to retrieve 10 items
/layout\b/start\b\    - hourly report header
temp\b/temp\b\        - multi-day forecast temp (max or min)
temp\b/temp\b\        - .. temp (max or min)
Prob\b/prob\b\        - .. probability of precipitation
apparent\b/temp\b\    - current temp
dew\b/temp\b\         - .. dewpoint
relative\b/humid\b\   - .. humidity
direction\b/dir\b\    - .. wind direction
gust\b/wind\b\        - .. wind gust
sustain\b/wind\b";    - .. wind sustained
*/
static const char numkeys = 10;
static const char keys[] = "\
/layout\b/start\b\
Temp\b/temp\b\
temp\b/temp\b\
Prob\b/prob\b\
apparent\b/temp\b\
dew\b/temp\b\
relative\b/humid\b\
direction\b/dir\b\
gust\b/wind\b\
sustain\b/wind\b";
```

Listing 3—The `get.h` code defines the `GET` request issued to the weather service webserver and the keys specifying which XML data to retrieve.



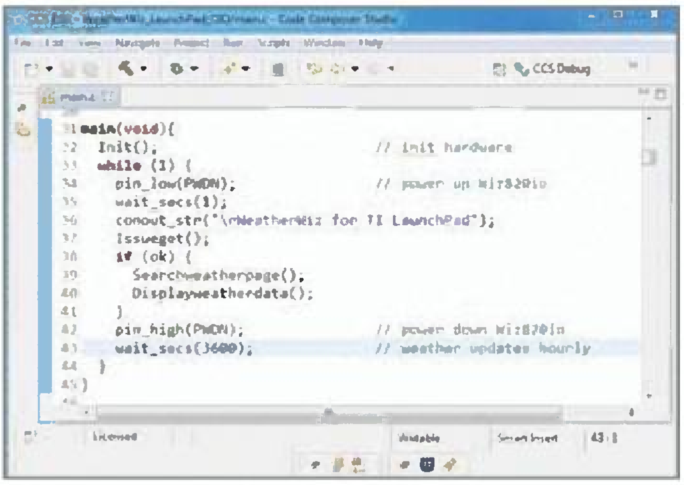
```
// hal for TI Launchpad using CCS debug console for display
#include <msp430g2553.h>
#include "pins.h"

/******************* Hardware Abstraction *******************/

/* modify these routines (and pin assignment in pins.h) for specific MCU

   halinit()             - init hardware (mcu, uart, spi, etc.)
   conout(byte)          - output to debug console
   dir_in(pin)           - set pin direction to input
   dir_out(pin)          - set pin direction to output
   pin_high(pin)         - output pin = 1
   pin_low(pin)          - output pin = 0
   pin_in(pin)           - get input pin, returns 0 or 1
   wait_secs(secs)       - wait for (secs) seconds
   led_on                - turn led on
   led_off               - turn led off
   shiftout(count,data)  - SPI shiftout count bits from data
   shiftin(count)        - SPI shiftin count bits, return data
*/
```

Listing 4—The `hal.c` and the `pins.h` code isolate all hardware-specific parameters.



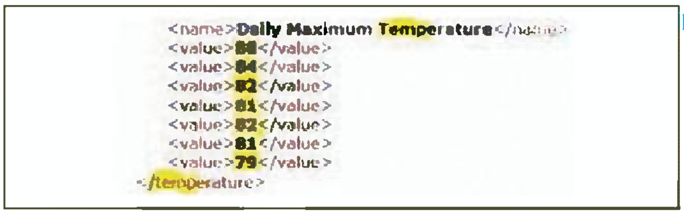
```
main(void){
  Init();                                   // init hardware
  while (1) {
    pin_low(PWDN);                          // power up Wiz820io
    wait_secs(1);
    conout_str("\rWeatherWiz for TI LaunchPad");
    Issueget();
    if (ok) {
      Searchweatherpage();
      Displayweatherdata();
    }
    pin_high(PWDN);                         // power down Wiz820io
    wait_secs(3600);                        // weather updates hourly
  }
}
```

Listing 5—The `main.c` code is simple: Power up the WIZ820io, issue the `GET` request, search the weather page, display the found data, and power down until the next reading.

the finish key. Note that a single key entry may result in the capture of one (e.g., current temperature) or more values (e.g., seven-day temperature forecast (see Photo 3).

THANKS FOR THE MEMORIES

At power-up, the program starts with Init (see Listing 6). This program, in turn, calls halinit to initialize the hardware, doing things like setting the microcontroller clock rate (e.g., factory-calibrated 1-, 4-, 8-, or 16-MHz) and configuring I/O modules



```
  <name>Daily Maximum Temperature</name>
  <value>88</value>
  <value>84</value>
  <value>82</value>
  <value>81</value>
  <value>82</value>
  <value>81</value>
  <value>79</value>
</temperature>
```

Photo 3—Using the key "Temp\b/temp\b" the `Searchweatherpage` routine captures the multi-day temperature forecast, as shown in this screen shot.

(e.g., SPI and UART). Then the Wiz820io pins controlled by main.c (e.g., RESET, nSS, and PWDN) are initialized (Note: MISO, MOSI, and SCLK are controlled by hal.c) and the Wiz820io is reset and loaded with the network addresses (defined in `netparms.h`).

The final step is to initialize the W5200 transmit and receive buffer configuration, taking advantage of the extra RAM to simplify the program and minimize microcontroller RAM usage. The W5200 16-KB receive buffer is large enough to capture and retain a copy of the entire webpage (typically 10 to 12 KB) so there's no bother juggling things piecemeal. As for the search results, the 10 data items I captured (plus a timestamp) typically add up to about 140–150 bytes. That's a significant portion of the 512-byte microcontroller RAM. A more realistically full-featured search function might capture even more data. Fortunately, the W5200 transmit buffer works as general-purpose RAM and just 2 KB is used to send the `GET` requests, so the remaining 14 KB can be used to store search results and other application data.

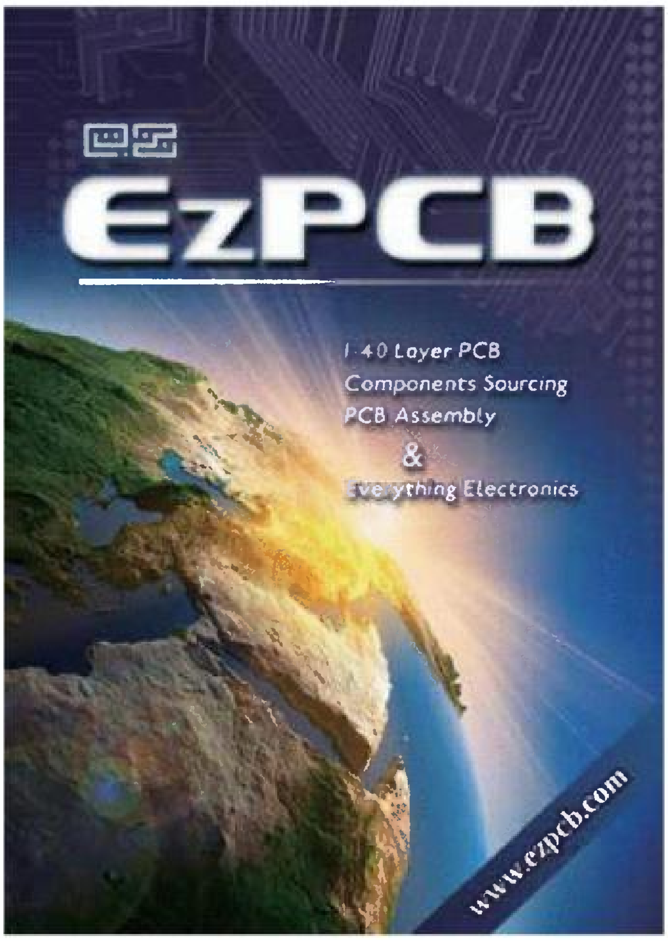
The Convenient All-in-One Solution
for Custom-Designed Front Panels & Enclosures

You design it
to your specifications using our FREE CAD software, Front Panel Designer

We machine it
and ship to you a professionally finished product, no minimum quantity required

- Cost effective prototypes and production runs with no setup charges
- Powder-coated and anodized finishes in various colors
- Select from aluminum, acrylic or provide your own material
- Standard lead time in 5 days or express manufacturing in 3 or 1 days

FRONT PANEL EXPRESS
FrontPanelExpress.com
1(800)FPE-9060



DATA ACQUISITION

The `Issueget` routine is where the action is (see Listing 7). After setting the source port address, the socket is configured with the W5200 `OPEN` command then the `CONN` command is used to establish connection with the server. If either of these steps fail (as indicated by the socket status = 0x00, i.e., socket closed) I set a flag to '`Not OK`' (i.e., ok = 0) and return.

Once connected, issuing the `GET` request requires some code to handle eventual wraparound of the 2-KB transmit buffer (see Listing 8). If the `GET` request's length would put it past the end of the buffer, it gets split into an upper and a lower portion. Once the `GET` request is in the transmit buffer, the W5200 `SEND` command transmits it across the Internet to the server.

After receiving the `GET` request, the server will start sending the webpage. The W5200 automatically captures the data into the receive buffer, so the program just waits for the server to initiate a disconnect (signifying the transfer is complete) then issues a W5200 `CLOSE` command to terminate the connection.

During development, I encountered glitches (e.g., a server timeout in response to a faulty `GET` request) so I checked whether less data was returned than expected and, if so, displayed the data (e.g., HTTP response header) to see what had happened.

TIPS AND TRICKS

Photo 4 shows the program in action. I used the CCS built-in debug console for display (i.e., the `hal 'conout'` routine). The CCS console is convenient, but also very slow since it uses software breakpoints to transfer data. I swear I heard the clacking of an old Teletype as the characters crawled onto the screen. It's not a showstopper for this demo (and you can speed things up by increasing the buffer size in the microcontroller's RAM), just a heads-up so you're not surprised. Poking around under the hood of the W5200, you can see the receive buffer contains the HTTP response header followed by the webpage (see Photo 5).

In the interest of brevity I won't go through the `Search weatherpage` and `Displayweatherdata` routines, as these are just demo stand-ins for a real application. The entire project is available on *Circuit Cellar*'s FTP site. In the meantime, Photo 6 shows these routines' function.

As you're dissecting the weather service data, keep in mind it's actually generated by humans, and thus, subject to some variability. For instance, you may be wondering why I used the same key for the maximum and minimum multi-day temperature forecast:

```
Temp\b/temp\b
Temp\b/temp\b
```

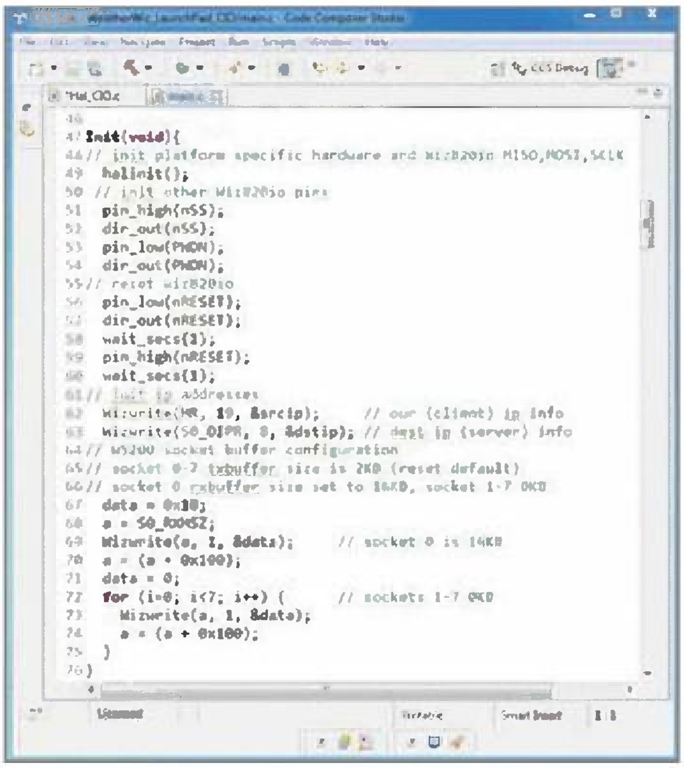
instead of:

```
Max\b/temp\b
Min\b/temp\b
```

I originally used the latter and it seemed to work fine. But then one day, `Searchweatherpage` started acting up. After a bit of head scratching, I noticed the webpage's "max" and "min" entries had switched position, so apparently the order is not set in stone (see Photo 7).

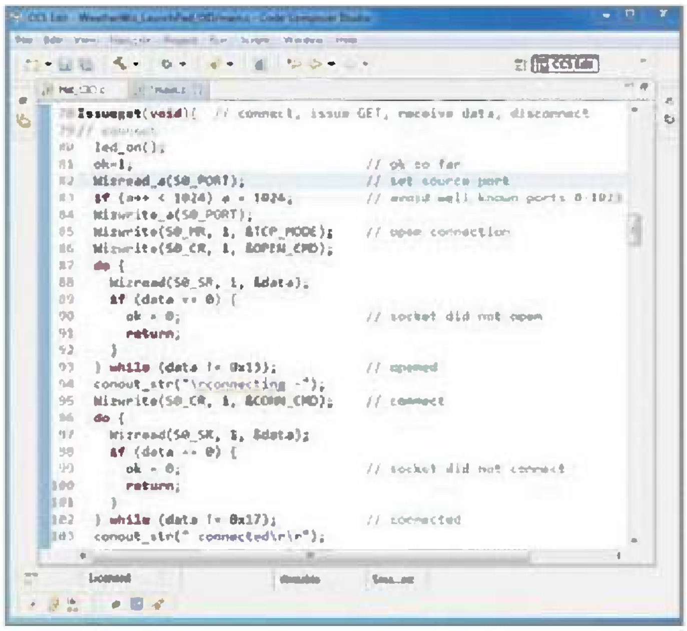
In the implementation of `Searchweatherpage` I used, the keys needed to be in the order they appeared on the webpage, so the switch messed things up. Since the webpage is stored in the receive buffer, there are certainly ways to handle this (e.g., multi-pass search), but it's also easy enough to determine which temperatures are maximums and which are minimums by inspection.

Speaking of numbers, when the time comes to convert the ASCII on the webpage to numbers for calculation, note that an



```
Init(void){
// init platform specific hardware and Wiz820io MISO,MOSI,SCLK
  halinit();
// init other Wiz820io pins
  pin_high(nSS);
  dir_out(nSS);
  pin_low(PWDN);
  dir_out(PWDN);
// reset wiz820io
  pin_low(nRESET);
  dir_out(nRESET);
  wait_secs(1);
  pin_high(nRESET);
  wait_secs(1);
// init ip addresses
  Wizwrite(MR, 19, &srcip);      // our (client) ip info
  Wizwrite(S0_DIPR, 8, &dstip); // dest ip (server) info
// W5200 socket buffer configuration
// socket 0-7 txbuffer size is 2KB (reset default)
// socket 0 rxbuffer size set to 16KB, socket 1-7 0KB
  data = 0x10;
  a = S0_RXMSZ;
  Wizwrite(a, 1, &data);      // socket 0 is 16KB
  a = (a + 0x100);
  data = 0;
  for (i=0; i<7; i++) {       // sockets 1-7 0KB
    Wizwrite(a, 1, &data);
    a = (a + 0x100);
  }
}
```

Listing 6—The programs starts with a platform-specific setup (`halinit()`) then initializes the WIZ820io pin connections and socket buffer configuration.



```
Issueget(void){  // connect, issue GET, receive data, disconnect
// [illegible]
  led_on();
  ok=1;                           // ok so far
  Wizread_s(S0_PORT);             // set source port
  if (s++ < 1024) s = 1024;       // avoid well known ports 0-1023
  Wizwrite_s(S0_PORT);
  Wizwrite(S0_MR, 1, &TCP_MODE);  // open connection
  Wizwrite(S0_CR, 1, &OPEN_CMD);
  do {
    Wizread(S0_SR, 1, &data);
    if (data == 0) {
      ok = 0;                     // socket did not open
      return;
    }
  } while (data != 0x13);         // opened
  conout_str("\rconnecting -");
  Wizwrite(S0_CR, 1, &CONN_CMD);  // connect
  do {
    Wizread(S0_SR, 1, &data);
    if (data == 0) {
      ok = 0;                     // socket did not connect
      return;
    }
  } while (data != 0x17);         // connected
  conout_str(" connected\r\r");
```

Listing 7—The `Issueget` routine opens a TCP socket and connects to the weather service webserver.

THIS IS

CC Gold

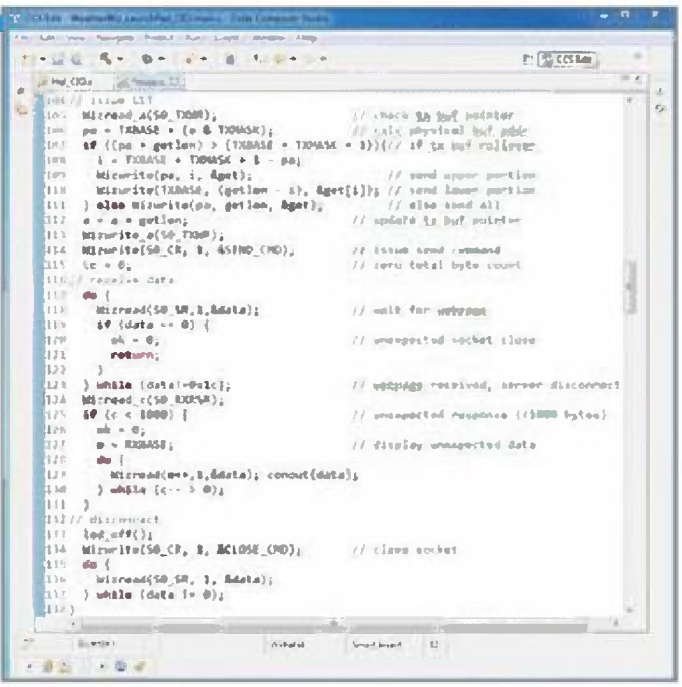
OVER TWO DECADES OF ENGINEERING EXCELLENCE

At Your Fingertips

Now you can have the complete Circuit Cellar issue archive and article code stored on a stylish, durable and portable USB flash drive. You can easily keep your CC Gold archive up to date by purchasing subsequent issues from our webshop or by downloading issues with a Circuit Cellar Digital 2.0 Subscription. Issues appear in searchable PDF format.

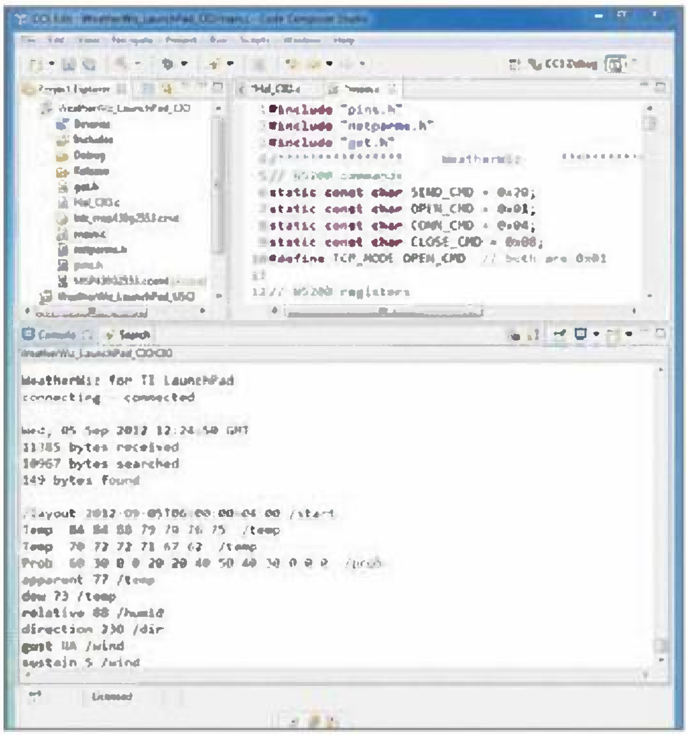
Complete archive includes PDFs of all issues in print through date of purchase.
(For display purposes) USB drive shown is larger than the actual size.

Visit www.cc-webshop.com to purchase



```
// issue GET
wizread_s(S0_TXWR);                        // check tx buf pointer
pa = TXBASE + (s & TXMASK);                // calc physical buf addr
if ((pa + getlen) > (TXBASE + TXMASK + 1)){// if tx buf rollover
  i = TXBASE + TXMASK + 1 - pa;
  wizwrite(pa, i, &get);                   // send upper portion
  wizwrite(TXBASE, (getlen - i), &get[i]); // send lower portion
} else wizwrite(pa, getlen, &get);         // else send all
s = s + getlen;                            // update tx buf pointer
wizwrite_s(S0_TXWR);
wizwrite(S0_CR, 1, &SEND_CMD);             // issue send command
tc = 0;                                    // zero total byte count
// receive data
do {
  wizread(S0_SR,1,&data);                  // wait for webpage
  if (data == 0) {
    ok = 0;                                // unexpected socket close
    return;
  }
} while (data!=0x1c);                      // webpage received, server disconnect
wizread_s(S0_RXRSR);
if (s < 1000) {                            // unexpected response (<1000 bytes)
  ok = 0;
  p = RXBASE;                              // display unexpected data
  do {
    wizread(p++,1,&data); conout(data);
  } while (s-- > 0);
}
// disconnect
led_off();
wizwrite(S0_CR, 1, &CLOSE_CMD);            // close socket
do {
  wizread(S0_SR, 1, &data);
} while (data != 0);
}
```

Listing 8—The code that sends the `GET` request needs to handle eventual wraparound of the WIZ820io 2-KB transmit buffer.



```
#include "pins.h"
#include "netparms.h"
#include "get.h"
//*****************          WeatherWiz          **********
// W5200 commands
static const char SEND_CMD = 0x20;
static const char OPEN_CMD = 0x01;
static const char CONN_CMD = 0x04;
static const char CLOSE_CMD = 0x08;
#define TCP_MODE OPEN_CMD   // both are 0x01

// W5200 registers
```

```
WeatherWiz for TI LaunchPad
connecting - connected

Wed, 05 Sep 2012 12:24:50 GMT
11385 bytes received
10967 bytes searched
149 bytes found

/layout 2012-09-05T06:00:00-04:00 /start
Temp  84 84 88 79 79 76 75  /temp
Temp  70 72 72 71 67 62  /temp
Prob  60 30 0 0 20 20 40 50 40 30 0 0 0  /prob
apparent 77 /temp
dew 73 /temp
relative 88 /humid
direction 230 /dir
gust NA /wind
sustain 5 /wind
```

Photo 4—This Code Composer Studio screen shot shows the WeatherWiz program in action.

ostensibly numeric field may contain something like "NA" or "MM." You'll also find shorthand such as "999" for "variable" wind direction. Similarly, looking at the "Probability of Precipitation" values on the webpage you'll notice the use of the following as XML-speak for zero:

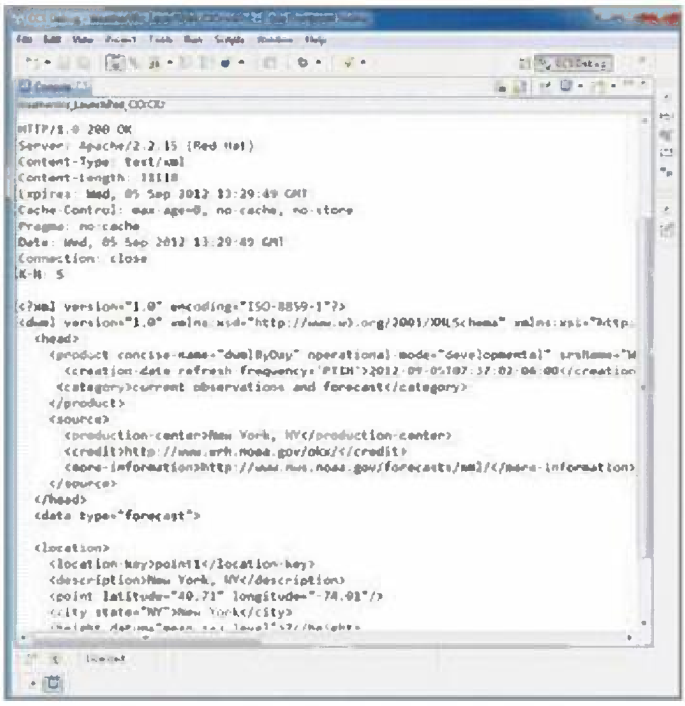
```
<value xsi:nil="true"/>
```

I previously mentioned the MSP430 comes preprogrammed with factory calibrated 1-, 4-, 8- and 16-MHz clock settings for the on-chip oscillator. TI thoughtfully includes a separate crystal in the kit that you can solder onto the circuit board (with good eyes and a steady hand, it's in a tiny surface-mount package) but so far, I haven't missed it. An irrigation controller doesn't need super accurate timing, so a software real-time clock could start with the factory-calibrated settings and use the timestamp in the HTTP response header to compensate for drift.

IF THE "C" FITS

The whole point of this experiment was to see what it takes in the way of hardware and software to network-enable a small microcontroller. As Photo 8 shows, the answer is "not much."

I was pretty sure the 16-KB microcontroller flash would be adequate and that turned out to be an understatement. The program is just 3 KB. I was more concerned about the 512 bytes of

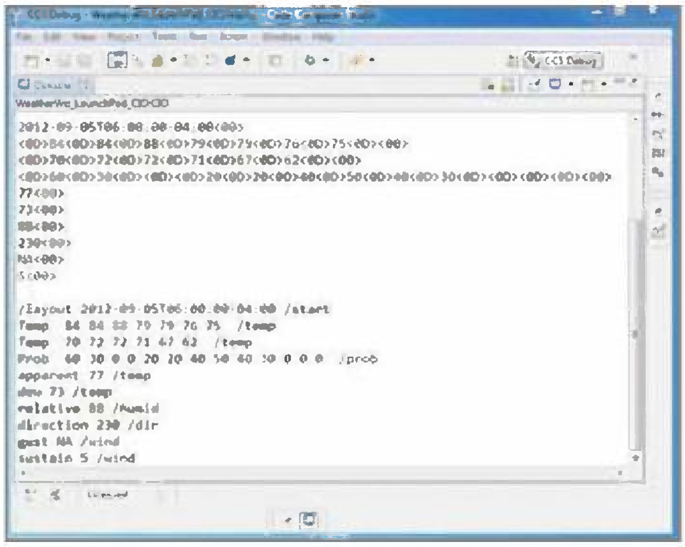


```
HTTP/1.0 200 OK
Server: Apache/2.2.15 (Red Hat)
Content-Type: text/xml
Content-Length: 11118
Expires: Wed, 05 Sep 2012 13:29:49 GMT
Cache-Control: max-age=0, no-cache, no-store
Pragma: no-cache
Date: Wed, 05 Sep 2012 13:29:49 GMT
Connection: close
X-N: S

<?xml version="1.0" encoding="ISO-8859-1"?>
<dwml version="1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http:
  <head>
    <product concise-name="dwmlByDay" operational-mode="developmental" srsName="W
      <creation-date refresh-frequency="PT1H">2012-09-05T07:37:02-04:00</creation
     <category>current observations and forecast</category>
    </product>
    <source>
      <production-center>New York, NY</production-center>
      <credit>http://www.erh.noaa.gov/okx/</credit>
      <more-information>http://www.nws.noaa.gov/forecasts/xml/</more-information>
    </source>
  </head>
  <data type="forecast">

  <location>
    <location-key>point1</location-key>
    <description>New York, NY</description>
    <point latitude="40.71" longitude="-74.01"/>
    <city state="NY">New York</city>
```

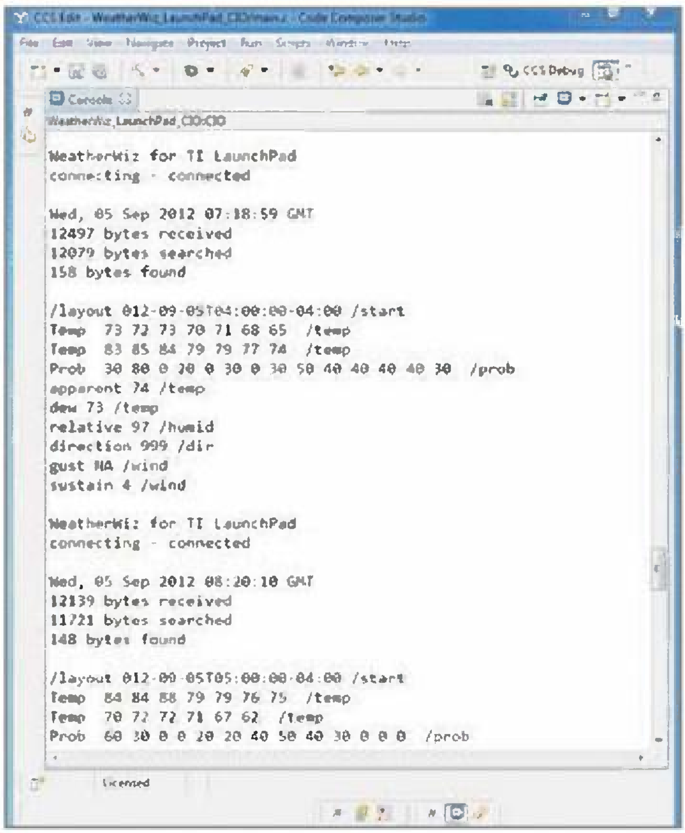
Photo 5—Here is a Code Composer Studio screen shot of the receive buffer.



```
2012-09-05T06:00:00-04:00<00>
<00>84<00>84<00>88<00>79<00>79<00>76<00>75<00><00>
<00>70<00>72<00>72<00>71<00>67<00>62<00><00>
<00>60<00>30<00><00><00>20<00>20<00>40<00>50<00>40<00>30<00><00><00><00><00>
77<00>
73<00>
88<00>
230<00>
NA<00>
5<00>

/layout 2012-09-05T06:00:00-04:00 /start
Temp  84 84 88 79 79 76 75  /temp
Temp  70 72 72 71 67 62  /temp
Prob  60 30 0 0 20 20 40 50 40 30 0 0 0  /prob
apparent 77 /temp
dew 73 /temp
relative 88 /humid
direction 230 /dir
gust NA /wind
sustain 5 /wind
```

Photo 6—Data is captured by `Searchweatherpage` (top) and displayed by `Displayweatherdata` along with the search keys (bottom).



CCS Edit - WeatherWiz_LaunchPad_CIO/main.c - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
CCS Debug
Console
WeatherWiz_LaunchPad_CIO:CIO
Licensed

```
WeatherWiz for TI LaunchPad
connecting - connected

Wed, 05 Sep 2012 07:18:59 GMT
12497 bytes received
12079 bytes searched
158 bytes found

/layout 012-09-05T04:00:00-04:00 /start
Temp  73 72 73 70 71 68 65  /temp
Temp  83 85 84 79 79 77 74  /temp
Prob  30 80 0 20 0 30 0 30 50 40 40 40 40 30  /prob
apparent 74 /temp
dew 73 /temp
relative 97 /humid
direction 999 /dir
gust NA /wind
sustain 4 /wind

WeatherWiz for TI LaunchPad
connecting - connected

Wed, 05 Sep 2012 08:20:10 GMT
12139 bytes received
11721 bytes searched
148 bytes found

/layout 012-09-05T05:00:00-04:00 /start
Temp  84 84 88 79 79 76 75  /temp
Temp  70 72 72 71 67 62  /temp
Prob  60 30 0 0 20 20 40 50 40 30 0 0 0  /prob
```

Photo 7—Notice the order of appearance of the maximum and minimum temperature forecasts may change.

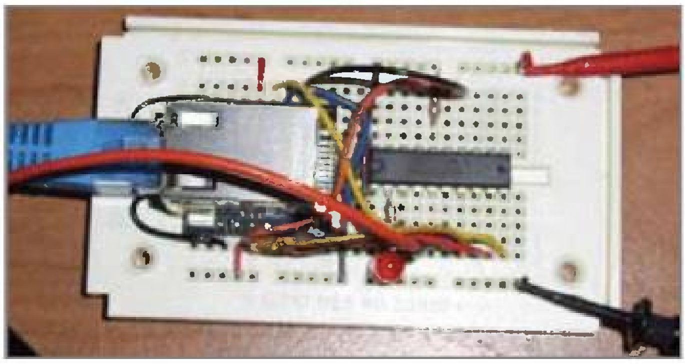


Photo 8—This is the MSP430 and Wiz820io breadboard.

microcontroller RAM, but using the W5200 buffers to store the data meant the program only needed about 100 bytes of microcontroller RAM for the default stack (80 bytes) and the global variables.

The MSP430 Value Line chips run up to 16 MHz. But, just for kicks, I set the microcontroller clock rate to 1 MHz. The slow-down is naturally quite noticeable (the LED shows the MSP430 chugging through `Searchweatherpage` for almost 1 min.), but the program worked correctly.

WEATHER OR NOT

No doubt, weather-related apps are, pardon the pun, a "hot" topic. But the idea of giving low-cost embedded designs access to the global web of real-world data goes further than that. Indeed, the possibilities are as limitless as the web itself. The "Big Data" is out there. Why not let our gadgets make the most of it? ■

Tom Cantrell (microfuture@att.net) has been working on chip, board, and systems design and marketing for several years.

PROJECT FILES

To download the project files and an image of the pinout go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2013/273.

RESOURCES

P. Curtis, "Adam Dunkel's μIP on the Olimex EasyWeb2 and LPC-E2124," Rowley Associates, Ltd., www.rowley.co.uk/msp430/uip.htm.

A. Dannenberg, "MSP430 Internet Connectivity," Application Report SLAA137A, Texas Instruments, Inc., 2004, www.ti.com/lit/an/slaa137a/slaa137a.pdf.

SOURCES

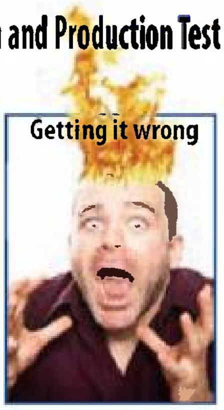
HR961160C Surface-mount RJ45 jack
HanRun Electronics Company, Ltd. | www.hanrun.com

MSP-EXP430G2 LaunchPad, MSP430 Value Line microcontroller, and Code Composer Studio IDE
Texas Instruments, Inc. | www.ti.com

WIZ820io Ethernet module and W5200 smart Ethernet chip
WIZnet Co., Ltd | www.wiznet.co.kr

Get it Right the First Time!

Embedded Design Verification and Production Test

Getting it right	Getting it wrong
	
✓High quality design	✓Errors found by software
✓Fewer board spins	✓Errors found by production
✓On-time delivery	✓Unexplained crashes
✓Lower development costs	✓Missed schedules
✓Lower production costs	✓Unnecessary board spins
✓Lower maintenance costs	✓Poor production yields
	✓Field returns

Verification and Test OS (VTOS™) gets it right the first time!

KOZIO®

http://www.kozio.com/techpapers or call us at 303-776-1356

Engineering and Entrepreneurship

An Interview with Michael Hamilton

Michael Hamilton has been designing microcontroller-based systems for 25 years. Over the past 10 years, he has spearheaded two companies: A&D Technologies, which supplies wireless temperature and humidity controllers, and Point & Track, which provides data-gathering apps and other business intelligence tools. In January, I interviewed Michael about his longtime interest in electronics, his first microcontroller design, his award-winning Renesas Electronics RL78 project, and his praise for 3-D printers.—Nan Price, Associate Editor

NAN: Where are you located?

MICHAEL: Lewisville, TX.

NAN: Give us some background information. Did anything specific spark your interest in engineering?

MICHAEL: My dad was an instrument repairman for Ashland Refinery in Canton, OH. As part of his training, he brought home his electronics trainer along with the instruction manuals.

When I was 13, I was interested in explosives and I needed an electronic timer for safety reasons. I studied his coursework and figured out how to build a timer using the famous 555 timer chip.

My family had a Christmas tree farm, where I spent many hours working. This led me to the decision that I needed to go to college instead of doing hard labor. I ended up going to Ohio University in Athens, OH, to study chemical engineering.

The reason I chose chemical engineering was, during high school I entered the science fair with a project called the "Distillation of Crude Oil." This project was very successful as I made it to the state science fair and won lots of prizes.

After I graduated from college, my interest turned back to electrical engineering and I started reading every book I could find on electronics. I remember my first big electronics purchase was a Tektronix 485 oscilloscope.

While working for Ashland Chemical in clean room environments, I realized there was a need for an accurate humidity controller. This led me to design my own temperature and humidity controller and form my first company, A&D Technologies (www.a-dtechnologies.com) in 2003.

The Tektronix 485 oscilloscope was Michael's first electronics acquisition.

Michael used a 555 timer chip to build a timer when he was just 13 years old.

NAN: What types of products and services does your company provide?

MICHAEL: A&D Technologies supplies wireless temperature and humidity controllers (e.g., HTC100) along with custom control panels. Using the latest technology, the control panels communicate to the outside world using SMS texting via cellular modems and e-mail via Ethernet.

Along with another partner, we created a new company, Point & Track (www.pointandtrack.com), which provides custom data-gathering apps for mobile devices such as iPhone and Android, secure database management, and business intelligence tools used to analyze collected data. The company also provides the ability to export geographic information system (GIS) data directly to customer-owned databases.

NAN: What type of work did you do prior to A&D Technologies and Point & Track?

MICHAEL: I was a project engineer who designed and installed automated equipment such as a fully automatic coiling systems using an ABB robot.

NAN: How long have you been designing microcontroller-based systems?

MICHAEL: Twenty-five years.

Michael's company, A&D Technologies, makes wireless temperature and humidity controllers.

NAN: What was the first microcontroller you worked with?

MICHAEL: It was Microchip Technology's PIC18F84. I designed a laser viscometer that was used to determine the viscosity of plastic resins while working for Ashland Chemical in Los Angeles, CA. I learned how easy it was to provide precise timing and work with digital I/O. It was so much easier than trying to work with individual integrated circuit (IC) chips.

NAN: What is the worst problem you have encountered with embedded microcontrollers?

MICHAEL: By far, the worst issue has to do with electromagnetic interference (EMI) from nearby devices, such as switching of solenoids or transformers. An extensive amount of time is spent designing PCBs so they will be immune to the external environment. Things like ground planes, metal oxide varistor (MOVs), transient voltage suppressors (TVSes), and capacitor/resistor networks are used to minimize the susceptibility of the microcontrollers, but it seems like you can never predict the environment for these kinds of issues. Maybe someone will write an article discussing these issues and how to prevent them.

NAN: Any recent tech purchases?

MICHAEL: I recently purchased a Rigol Technologies DSA-815-TG spectrum analyzer. This device is a must have, right behind the oscilloscope. It enables you to see all the noise/interference present in a PCB design and also test it for EMI issues.

NAN: Do you have any other unique tools on your workbench?

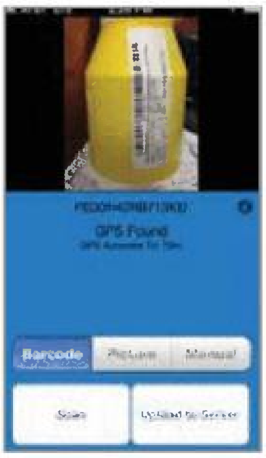
MICHAEL: I have a three-axis CNC machine and a MakerBot 3-D printer. I use the CNC machine to cut out enclosures and the 3-D printer to create bezels for LCDs and also to create 3-D prototypes. These machines are extremely useful if you need to make any precise cuts or if you want to create 3-D models of future products.

NAN: What is the fastest way to learn about programming and electronics?

MICHAEL: In the last six months, I have learned the following languages: Arduino, PHP, HTML 5, CSS, MySQL, Android, JavaScript, and jQuery. This was done by watching YouTube videos while exercising at the gym.

NAN: Your project, the Cloud Electrofusion Machine, recently won second prize in the 2012 RL78 Green Energy Challenge. Tell us about your project and your contest-entry process.

MICHAEL: The project created an inexpensive and energy-efficient way to weld polyethylene pipe together. Commercial



Along with a partner, A&D Technologies created a new company, Point & Track, which provides data-gathering apps such as the one shown here.

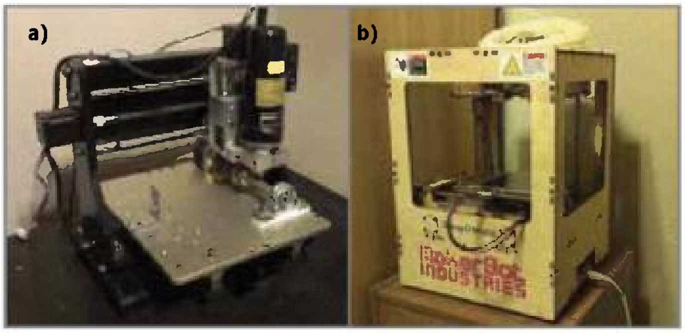
machines cost around $4,000. This machine can be built for less than $200. It utilizes a light dimmer to drop the voltage from 110 to 40 VAC and controls the amount of time that the power is applied to a coil inside an electrofusion fitting. By incorporating a barcode scanner, all the specific properties of the fitting can be easily entered into the microcontroller. Then, after the fusion is complete, all the data is sent to a cloud server via a wireless cellular modem.

The RL78 was very easy to use and program. I really didn't have many problems with the design.

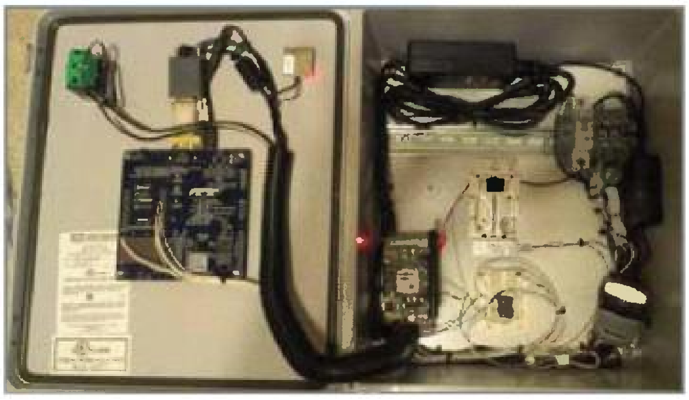
NAN: Your article "Infrared Radiation Measurement: FFT Double-Beam Infrared Spectrophotometer" (*Circuit Cellar* 229, 2009) describes a spectrophotometer built around a Microchip Technology dsPIC30F4012 digital signal controller. How does the microcontroller work in the design?

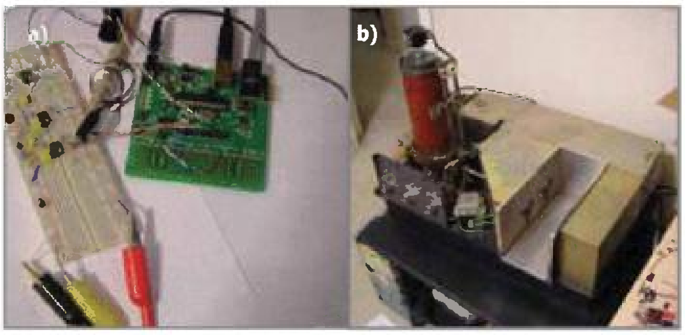
MICHAEL: The project created an instrument to identify chemical substances by using infrared spectroscopy. The dsPIC30F4012 uses a 10-bit ADC to measure the infrared radiation signal that

Michael recently bought a Rigol Technologies DSA-815-TG spectrum analyzer, which he says is a "must have" in PCB design.



Michael's workbench includes a three-axis CNC machine **(a)** and a MakerBot 3-D printer **(b)**.


Michael's Cloud Electrofusion Machine won second prize in the 2012 Renesas RL78 Green Energy Challenge.



A Microchip Technology dsPIC30F4012 digital signal controller **(a)** is at the heart of Michael's spectrophotometer **(b)**.

passes through a chemical substance. Then the microcontroller computes the fast Fourier transform (FFT) of the signal. This creates a table of amplitude versus frequency. The amplitude is then scaled to provide a relative transmittance. This information is serially transmitted over USB to a computer for plotting. This USB interface is accomplished using a PIC18F2450.

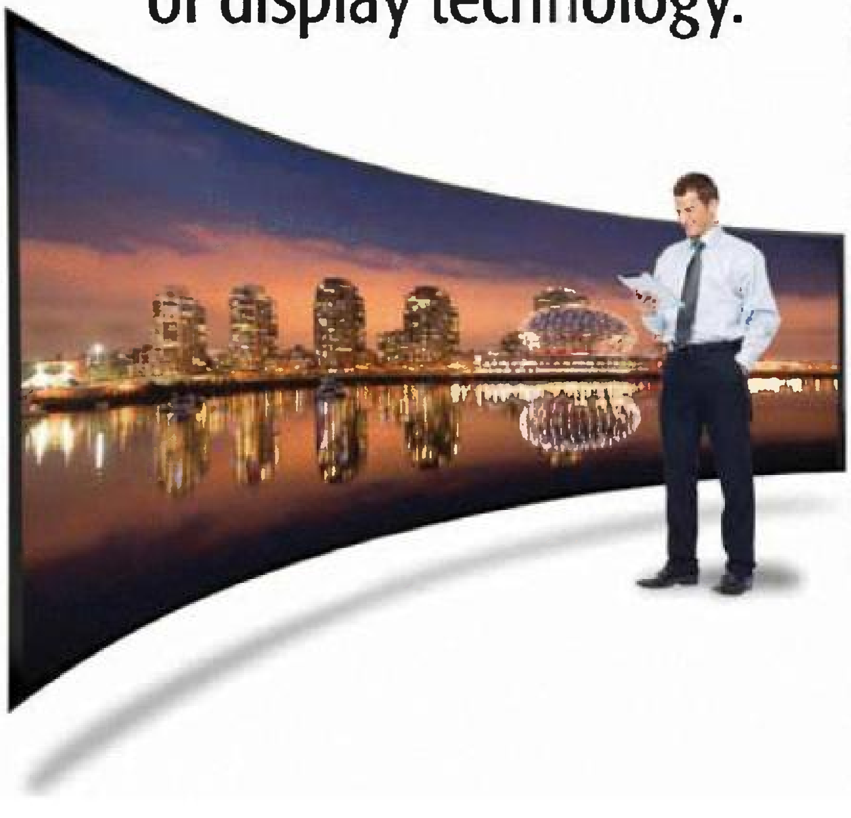
NAN: Are you currently working on or planning any microprocessor-based projects?

MICHAEL: We are currently working on a cloud fusion logger. This device reads all the data from the welding process in the field and transmits it to a cloud server. Later, the data can be analyzed and reports can be generated. A Raspberry Pi is used as the embedded controller. It is very fast and easy to use since it is based on Linux. We are working on getting the Android operating system loaded so existing code can be used and it will interface well with an Android smartphone, which will be used as the operator interface.

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

MICHAEL: One of the issues with embedded controllers is how to maintain the firmware and fix bugs after the devices are installed in the field. Using various wireless technologies, the devices will be automatically updated. Smartphones already use this technology. ◼

Come see the future of display technology.



What trends and innovations will be taking consumers by storm in just a few years? Meet the business and technology master- minds behind the major progress in OLEDs, Oxide TFT, 3D, ultra HD televisions, flexible displays and more at Display Week, SID's International Symposium, Seminar and Exhibition. Taking place in **Vancouver, Canada from May 19-24 at the Vancouver Convention Center**, be the first to witness the future of displays. Seeing is believing and you won't want to miss the premier display exhibition in North America. *www.displayweek.org*

SID DISPLAY WEEK
Vancouver 2013

CIRCUIT CELLAR

Microprocessor Design Using Verilog HDL

With the right tools, such as this **new book, designing a microprocessor can be easy.** Okay, maybe not easy, but certainly less complicated. Monte Dalrymple has taken his years of experience designing embedded architecture and microprocessors and compiled his knowledge into one **comprehensive guide to processor design in the real world.**

Yours for just $45.00

Monte demonstrates how Verilog hardware description language (HDL) enables you to **depict, simulate,** and **synthesize an electronic design** so you can **reduce your workload and increase productivity.**

Microprocessor Design Using Verilog HDL will provide you with information about:

- Verilog HDL Review
- Verilog Coding Style
- Design Work
- Microarchitecture
- Writing in Verilog
- Debugging, Verification, and Testing
- Post Simulation and more!

www.cc-webshop.com

by Bob Japenga (USA)

Concurrency in Embedded Systems (Part 6)

POSIX FIFOs and Message Queues

POSIX message queues work well under Linux, but Android does not support them. So what can you do when you're working on a hardware design that must run both OSes? POSIX FIFOs are an option. This article covers POSIX FIFOs, message queues, and how to decide when to use one or the other.

The last few articles in this series have discussed some mechanisms embedded systems designers can utilize with embedded Linux. Specifically, we have examined the mechanisms that aid us in creating systems with concurrency. Part 4 of this article series discussed the various multitasking options (i.e., threads and processes) that can be used to create concurrency. Part 5 provided details on how shared memory, semaphores, and mutexes help us communicate between concurrent processes. If you have been following this series, you'll know that we prefer using the POSIX standard mechanisms to implement our systems so we can easily port our design to another operating system (OS) should the need arise. Thus, all the mechanisms we have examined in this series have been the POSIX-compliant versions.

Recently at work, this cherished value of using standards-based mechanisms was severely challenged. We were creating some Java native interfaces (JNIs) for a new hardware design that would run both Linux and Android. As prime candidates for the *Android For Dummies* books, we proceeded with our POSIX-compliant design "unencumbered by the thought process." In particular, our design used POSIX message queues. Everything came up fine under Linux and things were proceeding on schedule. Then we started testing under Android. Oh, did I forget to mention Android does not support POSIX message queues? But isn't Android built on Linux? Aren't message queues implemented in the kernel? Yes, but...

After much pain and grief, we switched our interface from POSIX message queues to POSIX first in, first outs (FIFOs), which are supported under Android. So, it seems appropriate for me to address these two mechanisms available for interprocess communication (IPC) under Linux in this month's article.

POSIX FIFOs

A FIFO provides a unidirectional IPC that has a read end and a write end. Both hardware and software designers should be familiar with FIFOs. Basically, if you put the letters A, B, and C into a FIFO, they will come out in the order they were put in (A, B, and C). UARTs are the most common hardware devices that can contain FIFOs. If 16 characters come into the FIFO over the serial port, they come out in the order they went in.

Under Linux, you can create a FIFO and use it just like a hardware FIFO. If you put 16 bytes into the FIFO, you pull the bytes out in the order they were put in. Once you have pulled the data out of the FIFO, it no longer exists in the FIFO.

After you create the FIFO (using the `mkfifo` function), you can open it exactly like a file. All the standard attributes of files now apply to your FIFO (e.g., permissions, read only, etc). Under conventional operation, one process will open the FIFO for writing and one will open it for reading. The default operation forces both the reader and the writer to block until the other task opens its end of the FIFO. This can be overridden by using the O_NONBLOCK flag with slightly different results for the reader and the writer. If the reader uses the O_NONBLOCK flag to open the FIFO and the FIFO has not been opened for writing, the open will not block. If the writer uses O_NONBLOCK to open the FIFO and the FIFO has not been opened for reading, the open will fail. Table 1 provides a summary of the differences between POSIX FIFOs and regular files.

FIFOs	Regular Files
Once the data is read out of the FIFO, the data is no longer available.	Data can be read anywhere in the file using lseek.
Under default operation, both the writer and the reader of the FIFO will block until the other end is opened. (As described, the O_NONBLOCK flag can alter that behavior.)	Open and never block
The FIFO data is volatile (deleted at power-up).	File data volatility depends on the underlying file system
Even though the FIFO has a path name in the file system, the data put into a FIFO does not affect the underlying file system. This means you don't need to place the FIFO on your RAM drive rather than your flash file system to minimize flash usage. Even if you place the FIFO in the path on the flash file system, the name and the data are stored in volatile RAM. Another way to say this is that all reads and writes to the FIFO are passed internally in the kernel and do not affect the file system.	Performance of reads/writes is dependent on the underlying file system. Flash file systems have limited write cycles.
All data disappears if both reader and writer close the file.	Data does not go away when the file is closed.
The amount of data a FIFO can contain is limited by a kernel parameter (`FIFO_SIZE`) and is set to 65536 for Linux 2.6.11 and greater.	Data size is limited only by the file system's size.
All writes of less than or equal to `PIPE_BUF` (4096 for most Linux systems but the actual size varies on different kernels) are atomic. POSIX requires `PIPE_BUF` to be at least 512, so that is what we use in our designs.	Data writes are not atomic across processes.

Table 1—There are many differences between FIFOs and regular files.

DEBUGGING FIFOs

Since FIFOs look like files on the file system, the data being put in and taken out can be viewed from the command line (e.g., using `hexdump` or `cat`). This could be useful during debugging or even remotely should a released piece of code contain problems (which of course, your code never does!).

POSIX MESSAGE QUEUES

Linux provides both System V message queues and POSIX message queues. As we have previously discussed, our thin slice will limit my discussion to POSIX message queues.

When we discovered that Android did not support POSIX message queues, we chose to port this API with a POSIX FIFO. We determined that FIFOs were the closest IPC available under Android to the message queue for our purposes. One major thing we lost was that message queues enable variable-length packetized data. In other words, the writer function would write a variable-length message into the queue and the reader would read that variable-length message out as a complete packet. FIFOs are byte streams and do not make any

AP CIRCUITS
PCB Fabrication Since 1984

As low as...
$9.95
each!

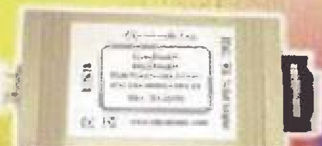
Two Boards
Two Layers
Two Masks
One Legend

Unmasked boards ship next day!

www.apcircuits.com

VISA MasterCard PayPal IPC MEMBER BBB

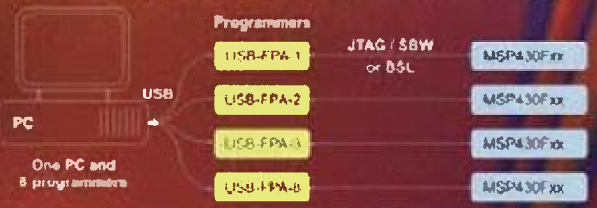
FlashPro430
FlashPro-CC
FlashPro2000
GangPro430
GangPro-CC



USB Flash Programmers for Texas Instruments' MCUs
MSP430, Chipcon CCxx, C2000 DSPs

Reliable and the fastest programmer on the market.
Perfect for production usage.

- can assign unique serial number
- up to eight programmers can be connected to one PC and program target devices simultaneously



Elprotronic
Incorporated
www.elprotronic.com

distinction concerning message boundaries. Thus, on the reader side, we did not know how many bytes to read out of the FIFO. We needed to use a terminator to determine the message size. And that is a big reason to use message queues instead of FIFOs in your designs.

POSIX message queues were not available under Linux prior to 2.6.6. Unlike FIFOs, which use standard `open`, `write`, `read`, and `unlink`, POSIX message queues are opened with `mq_open`, written to with `mq_send`, read with `mq_receive`, and unlinked with `mq_unlink`.

MESSAGE QUEUE ATTRIBUTES

POSIX message queues have the following settable attributes: the maximum number of messages you would allow and the maximum message size (in bytes). These values are set when the queue is created and cannot be changed after that. In addition, the programmer can obtain the number of messages currently in the queue and know whether or not the queue was created for blocking or nonblocking I/O (which can be changed after the queue is created).

MESSAGE NOTIFICATION

POSIX message queues enable a reader process to be asynchronously notified when a previously empty queue is no longer empty. Thus, your process does not need to block waiting on a `mq_receive`, but can perform other functions while waiting for the message. Your process can be notified by the standard signal mechanism in Linux (a topic for another day) when a message comes in. It does this through what is called registration. Message notification is another major advantage of queues vs. FIFOs.

There are some things to keep in mind about message notification. A process is only notified if a message comes in after it registers to be notified. This means that if there are messages in the queue when a process registers, it will not be notified of that existing message.

Only one process can be notified upon the arrival of a message in the message queue. You should use blocking if your design requires two listeners. To enable multiple listeners, each process must re-register after each notification.

MESSAGE QUEUE LIMITATIONS

Just as all OSes provide some limit to the number of open file handles, Linux limits the number of message queues. This number is 256 by default but can be dynamically adjusted up to a maximum of `INT_MAX`. This number is architecture dependent but never less than 4 billion. If you are creating a system with more than 4 billion message queues, you are designing systems far more complex than I can handle.

The maximum message size is also defined for Linux as 8,192 by default, but this can be dynamically adjusted up to a maximum of 1,048,576. Linux also provides the programmer a way to limit the number of bytes all queues for a particular user can use. This is a helpful check to protect your system against some runaway process that puts data into the queues but never removes it.

DEBUGGING MESSAGE QUEUES

Although the message queues are internal to the kernel, many message queue parameters are accessible from the command line. We have had systems that have exhibited aberrant behavior (must have been hardware problems!) and we were able to use this feature to remotely determine what was going on in the message queues. To access the queue, you need to first create a mount point for the queue:

```
mkdir /dev/my_queue
```

then mount the queue with the command:

```
mount -t mqueue none /dev/my_queue
```

At that point, you can determine the queue size, the signals used to notify processes, and you can even peek into the data. In addition, you can tell the message queue to send a notification to the process waiting for a signal from a particular message.

WHEN TO USE FIFOs OR MESSAGE QUEUES

I would say (unless you are developing an Android-native interface) that message queues are the preferred method since they are more flexible and are not significantly more complex to use than FIFOs. If you need the ability to handle variable-length data and require notification rather than blocking, message queues win, hands down. We have never noticed any appreciable difference in the resource utilization (real time and memory) between the two. In other words, we have not noticed that FIFOs are significantly faster than message queues, nor have we seen significant differences in memory usage (although we suspect that FIFOs are slightly faster and use less memory). So the only time I would recommend using FIFOs instead of message queues for IPC would be if the slight overhead advantage makes a difference.

THE COMPLEXITY OF EMBEDDED SYSTEMS

Creating embedded systems is a complex task and it is getting more complex every day. The embedded design community is now creating some pretty amazing products because of tools like Linux. POSIX FIFOs and message queues are two tools you can use with Linux. In my next article, I'll show how Linux can help you create your own amazing products. ■

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

RESOURCES

B. Japenga, "Concurrency in Embedded Systems (Part 4): Introducing Linux and Concurrency," *Circuit Cellar* 269, 2013.

———, "Concurrency in Embedded Systems (Part 5): Designing Robust Systems with Linux, *Circuit Cellar* 271, 2013.

by George Novacek (Canada)

Calculating Software Reliability

Determine Firmware Fault Probability

It is hard to imagine a product that does not require software. While the physical reliability and life of a product's hardware can be estimated, what about the failure probability due to a software bug? This article covers how to calculate software reliability and the practical use of the results.

In this article about reliability, I'll discuss software. As an engineer with a lifetime of experience in embedded-controller design, I'll limit my views to the software, or rather firmware, used therein. There may be different aspects in the development of commercial and other applications. Since this is an embedded-electronics magazine, I shall stay within its scope.

When estimating hardware reliability, the components' baseline failure rates λ (Greek letter lambda) and their modifiers π (Greek letter pi) are generally available from published documents. They have been empirically determined from many samples and, as we know, the larger the test sample, the more accurate the results.

Software is different. It doesn't age, nor is it affected by stress or environment. Its faults are built in, like typos, caused by omissions, defects in specifications, misunderstanding, incompetent design, poor coding practices, insufficient testing, and so forth. It is generally accepted that software characteristics are determined by the quality of the development process. Therefore, standards have been written to guide it. The US Department of Defense's DOD-STD-2167 dates back to the early 1980s, followed by MIL-STD-498, aerospace DO-178 and others. The standards are to enforce discipline and streamline the development process to minimize the probability of human error.

Software reliability λ_{SW} is then a probability of existence of faults (i.e., bugs) based on statistical categorization of the work of programmers and software-development organizations. We evaluate ourselves to fit a quality category that determines the probable number of errors we made originally and their reduction as we test, find, and correct them. The calculated λ_{SW}, as the theory goes, should then enable us to estimate the remaining number of faults in the software and thus determine if and when the testing can be terminated.

CALCULATING SOFTWARE RELIABILITY

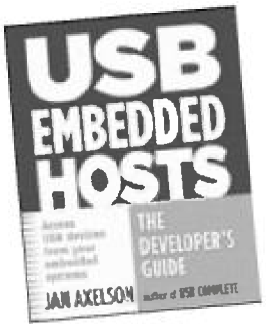
I will demonstrate two methods to calculate software reliability and show how different the results can be. Consider a typical embedded controller using a Freescale Semiconductor MPC555 microcontroller running at a 40-MHz clock rate with about 50,000 lines of code (LOC) written in C/C++. The hardware's mean time between failure (MTBF) is typically 30,000 h (i.e., λ = 33.33 faults per million hours). I would want the firmware λ_{SW} to be insignificant compared with the hardware λ (i.e., at least two orders of magnitude smaller, e.g., $\lambda_{SW} = 0.33$).

The basic equation for λ_{SW} per RIAC-HDBK-217Plus is:

$$\lambda_{SW} = \left(\frac{F_{TI-1} - F_{TI}}{730}\right)(DC \times FL \times FA \times AS) \times 10^{6} \text{ [1]}$$

This reliability model does not require specific controller data (e.g., MIPS). It is based on the process-quality category, which determines the original fault density. The LOC and the testing time for stabilization are taken into account. Reliability growth due to correction of failures discovered in the field must, for most embedded controllers, essentially be completed before the software can be deployed. While fixing faults as they are discovered in the field may be acceptable for PC applications, most embedded controllers cannot be deployed

Access USB Devices from your Embedded Systems



USB Embedded Hosts
The Developer's Guide
Jan Axelson
$29.95 **LVR.COM**
From the author of *USB Complete*

PIC-SERVO
MOTION CONTROL

MOTION CONTROLLERS FOR BRUSH, BRUSHLESS AND STEPPER MOTORS.

- controller chips
- controller boards

www.picservo.com
JEFFREY KERR, LLC

Connect With Design Engineers From Around The Globe.

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

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

without a reasonable assurance they are virtually fault free.

Variables in Equation 1 are mostly statistical data derived from failure reporting and corrective action systems (FRACAS). These are difficult to develop in smaller organizations because of the available sample size. For instance, I was responsible for the development of 18 aircraft embedded controllers over 20 years. Given the technology advances, people, and requirement changes over the two decades, hardly any usable statistics can be drawn from this. Therefore, I have to fall back on the RIAC-HDBK-217Plus defaults, which are based on the Software Engineering Institute's (SEI) Capability Maturity Model (CMM). I consider only the software certified to the highest quality (Level 1 and later Level A per DO-178) and, therefore, use the most favorable default multipliers suggested.

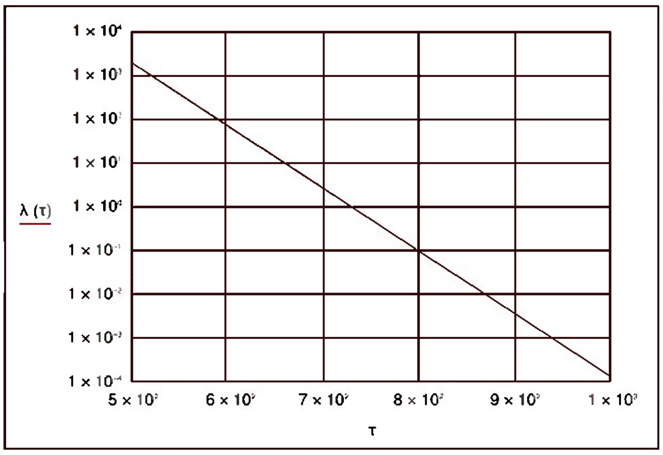


Figure 1—This chart shows the effect of testing on a predicted failure rate.

Assuming all the stabilization time and the reliability growth are essentially complete before the certification and deployment, the calculated reliability at that point is an unacceptable λ_{SW} = 801 failures per million calendar hours. If that is true, my career would have been over before it started.

The second example is based on a different model. It takes the development language, processor speed, and so forth, into consideration. The result, like RIAC-HDBK-217Plus, shows exponential reduction of remaining faults with testing time and corrections per equation:

$$\lambda_{\tau} = \lambda_{HS} \times e^{\frac{-\lambda_{HS}}{v_0} \times \tau} \qquad [2]$$

where λ_{HS} is the calculated failure rate at the start of testing, v_0 (Greek letter nu) the initial number of faults, and τ (Greek letter tau) the cumulative testing/running time. Figure 1 shows the resulting failure rate estimate λ_{τ} on a logarithmic scale plotted against testing time τ.

The testing time is cumulative. Thus, when several prototypes are running concurrently, the 770 h of testing presumably needed to achieve λ_{SW} = 0.33 doesn't take very long. For those wishing to dig deeper into this subject, publications regarding software reliability are listed in the References section of this article. Different mathematical models should be used for various systems. This may account for the difference in the results in my two examples.

Personally, I am skeptical about the usefulness of software reliability calculations. Controller hardware can fail long before its MTBF indicates, but only that one unit needs repair and that single failure does not invalidate the overall estimated product reliability λ. Just as well, our statistically excellent software development team can mess up once and its category rating will remain unaffected. But the produced software will contain more faults than calculated, will affect all built products and, if released on the strength of the statistical λ_{SW}, a massive recall or worse may result. Consequently, I don't believe the statistical λ_{SW} means much for an individual project.

TESTING & PROCEDURES

The two models suggest that the test duration is the primary driver for an organization's software reliability, which meets the defined process quality standards. Unfortunately, even in tightly controlled processes fitting into a statistical quality category, there is room for human error. There will be vast differences among companies and individuals working there, due to the vagaries of human behavior.

It has been mathematically demonstrated that 100% software test coverage is often practically impossible. That doesn't mean; however, that software cannot be thoroughly tested and be released with a small probability of a fault existence. To this end, software must be designed for testability from its very beginning, hand in hand with competent test cases and procedures. With execution of those procedures followed by equally well-prepared hardware/software and system-integration testing, the actual test time becomes inconsequential. It will merely be the time needed to successfully execute all those tests and will vary wildly, depending on the system. I'll address testability design in my next article.

For decades, a lot of work has been done to turn the art of software design into science. Great strides have been made, but there is a long way to go. By automating as much software development as possible, we can reduce our reliance on human contribution. There are marvelous (but costly) tools available to generate, simulate, and test code. Those tools help to eliminate defects caused by misunderstanding, incompetence, and mistakes. But until machines can replace us, if ever, we must depend on our own ingenuity and creativity to conceive new products and to keep human incompetence, momentary indisposition, and all those predictable moods in check. ■

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

RESOURCES

B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold, 1983.

Department of Defense, "Defense System Software Development," www.product-lifecycle-management.com/download/DOD-STD-2167A.pdf.

———, "Software Development and Documentation," 1994, www.abelia.com/498pdf/498-STD.PDF.

———, Information Analysis Center, "217Plus: RIAC's Reliability Prediction Methodology," www.theriac.org/productsandservices/products/217plus.

D. Dylis and M. Priore, "A Comprehensive Reliability Assessment Tool for Electronic Systems," the Illinois Institute of Technology (IIT) Research Institute (IITRI), www.theriac.org/productsandservices/products/217plus/rams01.pdf.

M. Friedman, P. Tran, and P. Goddard, "Reliability Techniques for Combined Hardware and Software Systems," RL-TR-92-15, 1992, www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA256347.

P. B. Lakey and A. M. Neufelder, "System Software and Reliability Assurance Notebook," Rome Laboratory, www.cs.colostate.edu/~cs530/rh/master01.pdf.

M. Lyu, *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.

J. McCall, W. Randell, J. Dunham, and L. Lauterbach, "Software Reliability, Measurement, and Testing: Software Reliability and Test Integration," Science Applications International Corp. (SAIC) and Research Triangle Institute (RTI), 1992, www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA256242.

J. McDermid and D. Pumfrey, "Software Safety: Why is There No Consensus?," University of York, www-users.cs.york.ac.uk/~djp/publications/ISSC_21_final_with_refs.pdf.

J. Pan, "Software Reliability," Carnegie Mellon University, 1999, www.ece.cmu.edu/~koopman/des_s99/sw_reliability.

Swedish Armed Forces, Defense Material Administration, "Handbook for Software in Safety Critical Applications," 2005, www.fmv.se/en/Our-activities/System-Safety/Handbooks-and-templates/Handbook-for-Software-in-Safety-Critical-Applications.

SOURCE

MPC555 Microcontroller
Freescale Semiconductor, Inc. | www.freescale.com

by Robert Lacoste (France)

Analyzing a Case of the Jitters

Tips for Preventing Digital Design Issues

Timing-signal transitions should occur at specific times. Jitter is the difference between the expected time and the actual time an event occurs. This article explains jitter's origins, two methods for measuring it and—most importantly—how it can negatively affect your high-speed digital designs.

I'm not that old, but I guess World War II-era electronics designers would be at ease with the majority of concepts we deal with today, at least on the analog side. However, there is one new subject of concern since the arrival of multi-megabit-per-second or even multi-gigabit-per-second digital transmissions: jitter.

The first real problems with jitter appeared in the 1970s. This was when the first all-digital telecommunication networks, such as plesiochronous digital hierarchy (PDH), were first unveiled. Just a couple of years later, companies (e.g., Hewlett-Packard) introduced the first dedicated jitter measurement equipment. Jitter is a 40-year-old subject, but you may not have encountered it in your designs.

WHAT IS JITTER?

Let's start with the basics. According to *Wikipedia*, "jitter is the undesired deviation from true periodicity of an assumed periodic signal."[1]

This is not incorrect, but I prefer the more generic definition provided by the International Telecommunication Union (ITU) in its G.810 recommendation: "Jitter (timing): The short-term variations of the significant instants of a timing signal from their ideal positions in time (where short-term implies that these variations are of frequency greater than or equal to 10 Hz)."[2]

I love these kinds of definitions. When you read them, you immediately know each word was carefully selected by experienced people.

First, jitter refers to timing signals (e.g., a clock or a digital control signal that must be time-correlated to a given clock). Then you only consider "significant instants" of these signals (i.e., signal-useful transitions from one logical state to the other). These events are supposed to happen at a specific time. Jitter is the difference between this expected time and the actual time when the event occurs (see Figure 1).

Last, jitter concerns only short-term variations, meaning fast variations as compared to the signal frequency (in contrast, very slow variations, lower than 10 Hz, are called "wander").

Figure 1—Jitter includes all phenomena that result in an unwanted shift in timing of some digital signal transitions in comparison to a supposedly "perfect" signal.

Figure 2—Cycle-to-cycle is the easiest way to measure jitter. You can simply trigger your oscilloscope on a signal transition and measure the dispersion of the following transition's time.

Why is jitter important? Take, for example, PDH telecom networks. On such networks—now nearly obsolete thanks to newer technologies such as synchronous optical networking (SONET), synchronous digital hierarchy (SDH), and Internet Protocol (IP) networks—all nodes are time-synchronized. This had a big advantage in terms of simplicity, as data frames usually didn't need to be buffered and could simply be forwarded their destination in real time with minimum time delay. But, in such systems, things can get weird if the bits arrive a little later (or sooner) than planned. PDH is nearly dead, but the same problem exists, and is even worse, in high-speed logic systems.

Take another example of a double data rate (DDR) chip or PCI bus running at a clock speed of hundreds of megahertz. If the delay between the active-clock transition and the read and write signal strobes is slowed by more than some nanoseconds, the memory access will probably fail.

Clock jitter is also a big concern for A/D conversions. Read my article on fast ADCs ("Playing with High-Speed ADCs," *Circuit Cellar* 259, 2012) and you will discover that jitter could quickly jeopardize your expensive, high-end ADC's signal-to-noise ratio.

CYCLE-TO-CYCLE JITTER

Assume you have a digital signal with transitions that should stay within preset time limits (which are usually calculated based on the receiver's signal period and timing diagrams, such as setup duration and so forth). You are wondering if it is suffering from any excessive jitter. How do you measure the jitter? First, think about what you actually want to measure: Do you have a single signal (e.g., a clock) that could have jitter in its timing transitions as compared to absolute time? Or, do you have a digital signal that must be time-correlated to an accessible clock that is supposed to be perfect? The measurement methods will be different. For simplicity, I will assume the first scenario: You have a clock signal with rising edges that are supposed to be perfectly stable, and you want to double check it.

My first suggestion is to connect this clock to your best oscilloscope's input, trigger the oscilloscope on the clock's rising edge, adjust the time base to get a full period on the screen, and measure the clock edge's time dispersion of the transition just following the trigger. This method will provide a measurement of the so-called cycle-to-cycle jitter (see Figure 2). If you have a dual time base or a digital oscilloscope with zoom features, you could enlarge the time zone around the clock edge you are interested in for more accurate measurements. I used an old Philips PM5786B pulse generator from my lab to perform the test. I configured the pulse generator to generate a 6.6-MHz square signal and connected it to my Teledyne LeCroy WaveRunner 610Zi oscilloscope. I admit this is high-end equipment (1-GHz bandwidth, 20-GSPS sampling rate and an impressive 32-M word memory when using only two of its four channels), but as I will show later, it enabled me to demonstrate some other interesting things about jitter. I could have used an analog oscilloscope to perform the same measurement, as long as the oscilloscope provided enough bandwidth and a dual time base (e.g., an old Tektronix 7904 oscilloscope or something similar). Nevertheless, the result is shown in Figure 3.

This signal generator's cycle-to-cycle jitter is clearly visible. I measured it around 620 ps. That's not much, but it can't be ignored as compared to the signal's period, which is 151 ns (i.e., 1/6.6 MHz). In fact, 620 ps is ±0.2% of the clock period. Caution: When you are performing this type of measurement, double check the oscilloscope's intrinsic jitter as you are measuring the sum of the jitter of the clock and the jitter of the oscilloscope. Here, the latter is far smaller.

TIME INTERVAL ERROR

Cycle-to-cycle is not the only way to measure jitter. In fact, this method is not the one stated by the definition of jitter I

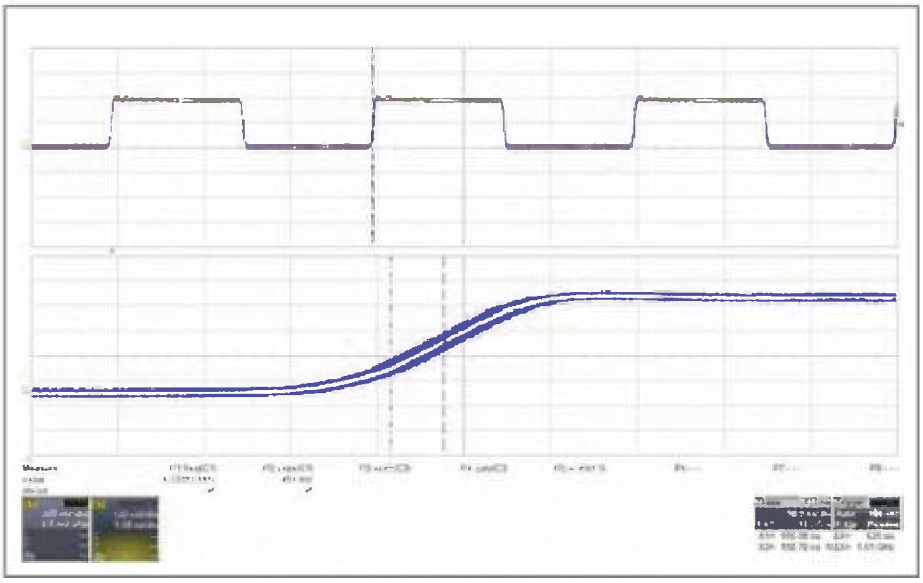


Figure 3—This is the result of a cycle-to-cycle jitter measurement of the PM5786A pulse generator. The bottom curve is a zoom of the rising front just following the trigger. The cycle-to-cycle jitter is the horizontal span of this transition over time, here measured at about 620 ps.

presented earlier. Cycle-to-cycle jitter is a measurement of the timing variation from one signal cycle to the next one, not between the signal and its "ideal" version. The jitter measurement closest to that definition is called time interval error (TIE). As its name suggests, this is a measure of a signal's transitions actual time, as compared to its expected time (see Figure 4).

It's difficult to know these expected times. If you are lucky, you could have a reference clock elsewhere on your circuit, which would supposedly be "perfect." In that case, you could use this reference as a trigger source, connect the signal to be measured on the oscilloscope's input channel, and measure its variation from trigger event to trigger event. This would give you a TIE measurement.

But how do you proceed if you don't have anything other than your signal to be measured? With my previous example, I wanted to measure the jitter of a lab signal generator's output, which isn't correlated to any accessible reference clock. In that case, you could still measure a TIE, but first you would have to generate a "perfect" clock. How can this be accomplished? Generating an "ideal" clock, synchronized with a signal, is a perfect job for a phase-locked loop (PLL). The technique is explained my article, "Are You Locked? A PLL Primer" (*Circuit Cellar* 209, 2007.) You could design a PLL to lock on your signal frequency and it could be as stable as you want (provided you are willing to pay the expense).

Figure 4—Time interval error (TIE) is another way to measure jitter. Here, the actual transitions are compared to a reference clock, which is supposed to be "perfect," providing the TIE. This reference can be either another physical signal or it can be generated using a PLL. The measured signal's accumulated plot, triggered by the reference clock, also provides the so-called eye diagram.

Moreover, this PLL's bandwidth (which is the bandwidth of its feedback filter) would give you an easy way to zoom in on your jitter of interest. For example, if the PLL bandwidth is 100 Hz, the PLL loop will capture any phase variation slower than 100 Hz. Therefore, you can measure the jitter components faster than this limit. This PLL (often called a carrier recovery circuit) can be either an actual hardware circuit or a software-based implementation (more on that later).

So, there are at least two ways to measure jitter: Cycle-to-cycle and TIE. (As you may have anticipated, many other measurements exist, but I will limit myself to these two for simplicity.) Are these measurement methods related? Yes, of course, but the relationship is not immediate. If the TIE is not null but remains constant, the cycle-to-cycle jitter is null. Similarly, if the cycle-to-cycle jitter is constant but not null, the TIE will increase over time. In fact, the TIE is closely linked to the mathematical integral over time of the cycle-to-cycle jitter, but this is a little more complex, as the jitter's frequency range must be limited.

JITTER CLASSIFICATION

Let's go back to the jitter. Things get interesting when it is closely examined. Imagine you measure a jitter in many successive clock periods and plot these measurements on a histogram. What will its shape be? If you are patient, you could use either a digital or memory-display analog type of standard oscilloscope to do the job. You could execute one measurement in single-sweep mode, plot the resulting measurement on the histogram, and redo the test many times. Fortunately, some digital oscilloscopes have such a feature built in.

In Figure 5 I have taken the same setup as was used for Figure 3 and launched one of LeCroy's JITKIT software packages. It provided me with a cycle-to-cycle jitter measurement, in terms of statistics (the peak to peak is, for example, 699 ps in this test, close to what was shown in Figure 3) and as a histogram.

Does this histogram's shape remind you of something? It should, because it's a perfect example of the classical Gaussian shape. Such a shape indicates that the jitter is a random

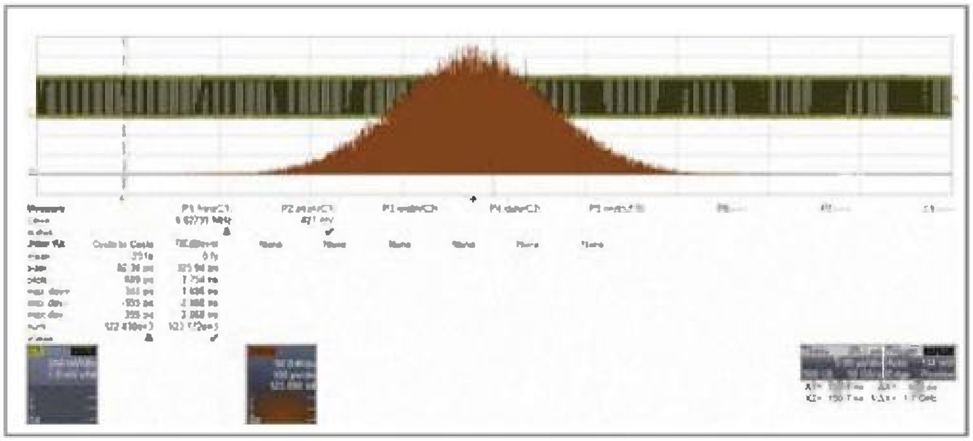


Figure 5—The histogram of the PM5786B pulse generator's jitter shows a clear Gaussian curve, which is a good indication of its random-noise nature.

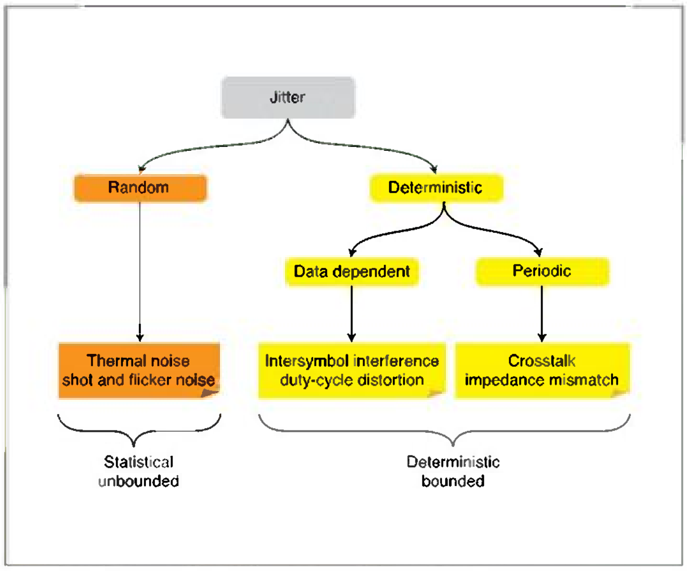


Figure 6: Jitter can be either random or deterministic.

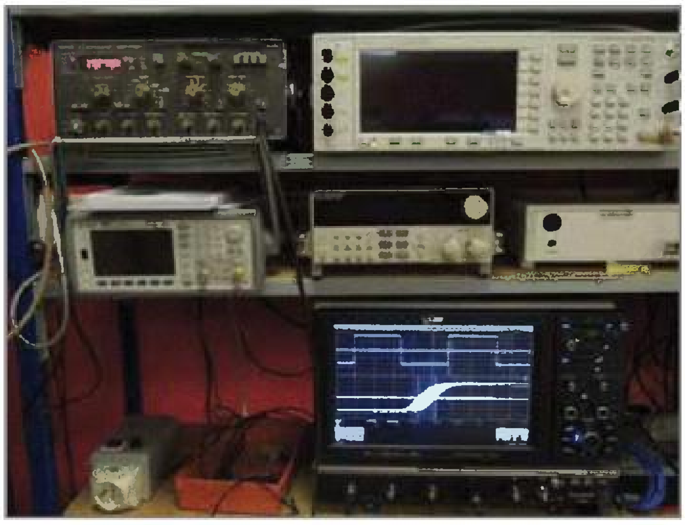


Photo 1—Here is the test setup. The partially faulty PM5134 generator is on the top left. The LeCroy WaveRunner oscilloscope, which shows a high level of jitter, is on the bottom.

noise-related jitter. This means it is caused by random fluctuations of voltages on the oscillator circuitry. To improve it, you would have to improve the design of the oscillator itself. You would have to use lower-noise silicon components, increase the working currents (as noise is lower when current is higher, at the expense of higher power consumption), use lower-value resistors (as thermal noise on resistors is proportional to their resistance), and so forth.

Such a random jitter is always present and is usually the root cause of the majority of all jitters. But it is not the only one!

Figure 6 provides a breakdown of the types of jitters you may encounter.

First, the jitter could be random by nature, as the jitter just discussed. In that case, it comes from noise in the electronics components and could be thermal, shot, or flicker noise, depending on the root physical cause. These later two jitters (shot and flicker noise) are linked more to the intrinsic properties of semiconductor materials, but usually all three contribute to the overall random noise of any circuit at different frequency offsets.

System on Module

- Atmel ARM9 400 MHZ Fanless Processor
- Up to 128 MB of DDR2 SDRAM
- Up to 1GB of NAND Flash
- Up to 8 MB Serial Data Flash
- 6 Serial Ports, 2 I2C and 2 SPI ports
- 2 Full Speed USB 2.0 Host ports
- 1 Full Speed USB 2.0 Device port
- CAN 2.0 B Controller, I2S Audio Port
- 10/100 BaseT Fast Ethernet with PHY
- Access to Processor Bus
- 5 Channels of 10-Bit A/D & 32 GPIO Lines
- SD/MMC Flash Card Interface
- System Reset, Real Time Clock
- Timers/Counters, PWM controller
- Small, 144 pin SODIMM form factor (2.66" x 1.50")

Wide Temperature
SoM-9x25

Windows CE

Designed and manufactured in the USA the SoM-9x25 uses the same small SODIMM form-factor utilized by other EMAC SoM modules and is the ideal processor engine for your next design. All of the ARM processor core is included on this tiny board including: Flash, Memory, Serial Ports, Ethernet, SPI, I2C, I2S Audio, CAN 2.0B, PWMs, Timer/Counters, A/D, Digital I/O lines, Clock/Calendar, and more. The SoM-9x25 is designed to plug into a custom or off-the-shelf carrier board containing all the connectors and any additional I/O components that may be required. The System on Module approach provides the flexibility of a fully customized product at a greatly reduced cost. Quantity 1 price begins at $180.

http://www.emacinc.com/som/som9x25.htm

Since 1985
OVER
28
YEARS OF
SINGLE BOARD
SOLUTIONS

EMAC, inc.
EQUIPMENT MONITOR AND CONTROL

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

THE ORIGINAL SINCE 1994
PCB-POOL®
Beta LAYOUT

FREE Stencil
with every prototype order

Embedded RFID
authenticate, track & protect your product
NEW!
www.magic-pcb.com

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

PCB-POOL® is a registered trademark of
Beta LAYOUT
create. electronics

www.pcb-pool.com

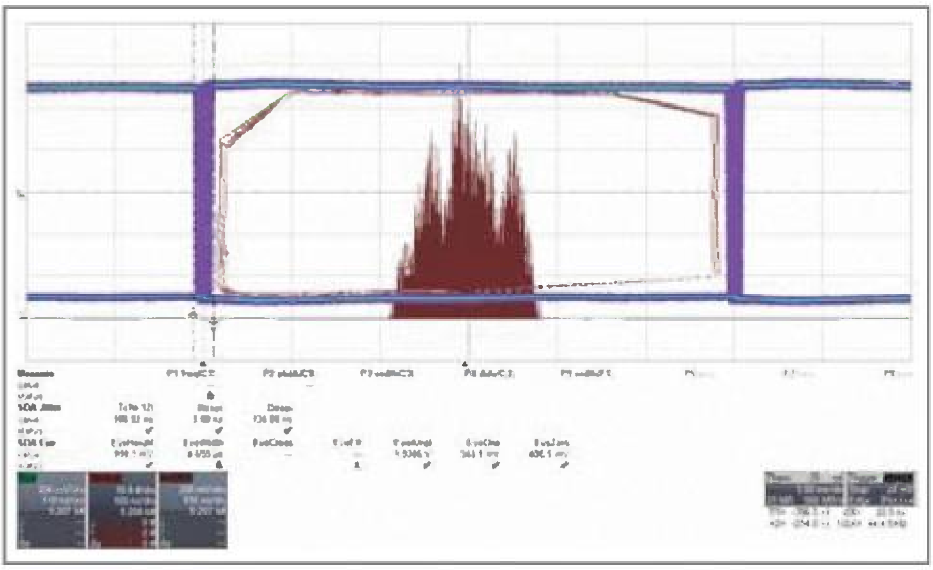


Figure 7—This analysis shows the PM5134 generator output's TIE. The eye diagram is superimposed on a histogram that is far from Gaussian.

Random-related jitter has two key properties: First, its shape is always more or less Gaussian and second (as a consequence, in fact), it is unbounded by nature. This means, with enough time, it will always exceed a given threshold. It's the same as playing the lottery: You can ensure you will win if you play long enough. For an engineer, that means a random jitter measurement should always be provided with the associated measurement duration.

But non-random jitter can also be present. These jitter contributors, called deterministic jitter, can be split into two categories: Data dependent and periodic. The first can't occur when you are only looking at a clock signal, but it is very frequent when measuring the jitter between a clock and a data or control line. In my article on emphasis and equalization ("High-Speed Signal Transmission," (*Circuit Cellar* 227, 2009) I referred to the so-called intersymbol interference (ISI). ISI appears when a signal behavior at a given time is partially modified by its value at previous time-steps. For example, a "zero" can be transmitted with slightly different timings and waveforms depending on the previously transmitted "ones" and "zeros." This is typically the root cause of data-dependent jitter: A signal's rising front may be faster or slower, depending on the previously transmitted data. This kind of jitter is deterministic and, therefore, bounded. The other kind of deterministic jitter is periodic jitter. In that case, there is an interfering signal somewhere, which modifies the signal timing you are seeking. Typical examples are either crosstalk between two clock signals on the same board or the effect of impedance mismatch on a line (which could generate spurious oscillations at precise frequencies).

A REAL-WORLD EXAMPLE

To show you some interesting jitter signals, I tested several of our lab signal generators and I found one. I set an old Philips PM5134 low-frequency signal generator to 100-kHz square wave mode, connected it to the oscilloscope, and measured a high level of jitter at ±100 ns, which is ±1% of the signal period (see Photo 1). The cycle-to-cycle jitter's histogram was not exactly Gaussian, but it was difficult to understand. It was time to test another high-end option installed on my Teledyne LeCroy oscilloscope, namely, the SDA-II data software package. This is a kind of extended version of the JITKIT I used before, with plenty of added features. In particular, it enables you to recover a reference clock, thanks to a software-based PLL, and thus, analyze the TIE without any external clock source.

Actually, this is quite simple to do by yourself, as long as you have a long-memory digital oscilloscope and a PC. Grab as many signal periods as possible (but keep a time base fast enough to measure the signal's transitions with enough accuracy), transfer them to a computer, write software to compare it with a software-generated clock at the same frequency as the signal, measure the phase difference, pass this difference through a digital low-pass filter, and use the result to correct the software reference clock.

As soon as you've done this, you can use the result to calculate the TIE and plot the signal's eye diagram, which is simply the accumulation of the signal's successive periods, triggered by the reference clock, which is supposed to be "perfect." Figure 7 shows the eye diagram and the TIE's histogram I produced during this test. The shape is clearly non-Gaussian and two peaks seem visible on both sides. What is it? What is the problem with this generator?

I pushed a couple more buttons and found another software feature. This enabled me to plot the time interval error over time, not on a histogram. Basically, I asked the oscilloscope to measure each signal period's TIE and plot it on the same scale as the signal itself. Therefore, the horizontal axis shows the time and the vertical axis shows the time interval error. The result is the blue curve shown in Figure 8, which is interesting. Two things are visible. First, there is a slow TIE oscillation. This oscillation's amplitude accounts for no less than 60% of the overall jitter. I measured the time interval between two peaks of this slow oscillation, which

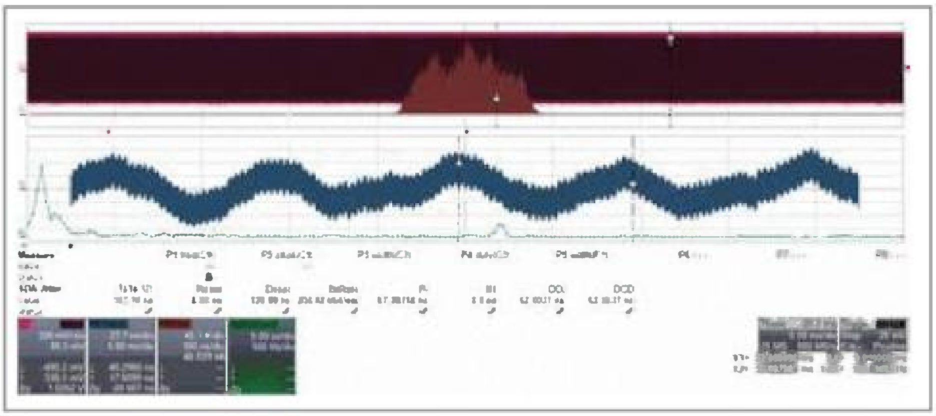


Figure 8—The plot of the TIE over time (blue) shows two interesting patterns: A 100-Hz slow oscillation and a faster 1.8-kHz oscillation, as confirmed by its Fourier transform (green).

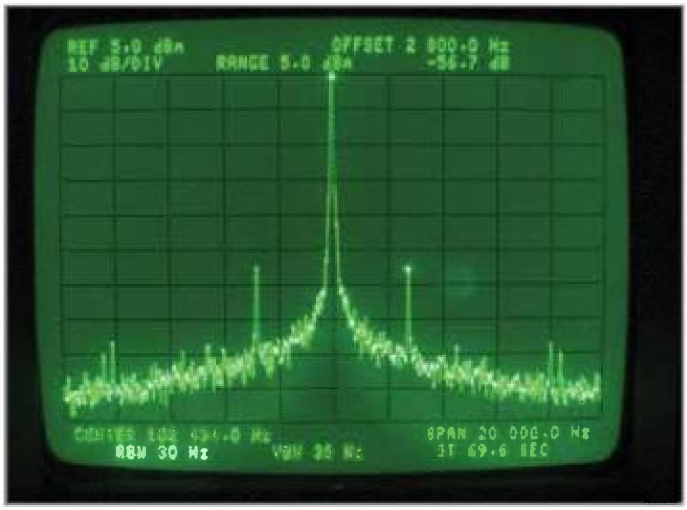


Photo 2—The spectrum analysis of the PM5134 output confirms the presence of a significant modulation, 1.8 kHz on each side of the 100-kHz carrier.

was 10 ms. Remember that I live in France; here the power line is 50 Hz. Bingo! This old generator's power supply probably has some faulty filtering capacitors, providing a 100-Hz noise on the DC power lines (thanks to the diode bridge), which produces a 100-Hz frequency modulation in the output. And a frequency modulation is a type of periodic jitter.

But the measurement of the TIE over time also shows a faster periodic variation, visible as small waves on the overall TIE waveform shown in Figure 7. What is it? As I began to get comfortable with the oscilloscope's features, I asked it to plot the TIE's Fourier transform over time and received the green curve shown in Figure 7. The 100-Hz peak is visible, but there is another peak around 2.8 kHz: Another spurious frequency modulation! I used a high-resolution Hewlett-Packard 3585A spectrum analyzer to verify my measurements. I connected the signal generator's output to the analyzer and displayed the signal spectrum around 100 kHz (see Photo 2). Bingo again! On each side of the signal 100-kHz carrier, there was a clear spur, 56 dB below the carrier. The frequency offset between the carrier and this spurious oscillation was, as expected, 2.8 kHz. This was another of my test generator's defaults. This time it was probably linked to a spurious oscillation of its internal circuits or unwanted modulation sources.

WRAPPING UP

I only superficially discussed jitter in this article, but I hope you found it interesting. As you have seen, jitter is one of the parameters you should consider when designing a project, especially when designing a high-speed digital system. Moreover, jitter investigation—performed either manually or with the help of proper measurement tools (e.g., from suppliers such as Teledyne LeCroy, Agilent, Tektronix, etc.)—can provide you with a thorough analysis of your product, even without opening the box. These tools can even use statistical analysis to break down the jitter into its different components.

I strongly encourage you to read the articles listed in the Resources section of this article, where you will find plenty of additional information about jitter. In particular, I have not even presented the interesting relationship between jitter and phase noise. This may be the subject of another article. In the meantime, you can read the Analog Devices MT-008 document. Have fun with jitter, which is no longer on your darker side!

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

REFERENCES

[1] Wikipedia, "Jitter," http://en.wikipedia.org/wiki/Jitter.

[2] International Telecommunication Union, "Transmission Systems and Media Digital Transmission Systems: Digital Networks: Design Objectives for Digital Networks," G.810 ITU-T Telecommunication Standardization Sector of ITU, 1996, www.itu.int/rec/T-REC-G.810-199608-I/en.

RESOURCES

Agilent Technologies, Inc., "Measuring Jitter in Digital Systems," http://cp.literature.agilent.com/litweb/pdf/5988-9109EN.pdf.

———, "Understanding Jitter and Wander Measurements and Standards," http://cp.literature.agilent.com/litweb/pdf/5988-6254EN.pdf.

Hewlett-Packard Journal, "Jitter Analysis of High-Speed Digital Systems," 1995, www.hpl.hp.com/hpjournal/95feb/feb95a8.pdf.

W. Kester, "Converting Oscillator Phase Noise to Jitter Time," MT-008 Tutorial, Analog Devices, Inc., www.analog.com/static/imported-files/tutorials/MT-008.pdf.

R. Lacoste, "Playing with High-Speed ADCs," *Circuit Cellar* 259, 2012.

———, "High-Speed Signal Transmission," *Circuit Cellar* 227, 2009.

———, "Are You Locked? A PLL Primer," *Circuit Cellar* 209, 2007.

National Instruments Corp., "Understanding and Characterizing Timing Jitter," 2012, www.ni.com/white-paper/14227/en.

Silicon Laboratories, "Clock Jitter Tutorial," http://pages.silabs.com/LP-Clock-Jitter-Tutorial.html.

SOURCES

3585A Spectrum analyzer
Hewlett-Packard, available through distributors such as eBay (www.ebay.com)

PM5134 Function generator and PM5786B pulse generator
Philips, available through distributors such as eBay (www.ebay.com)

WaveRunner 610Zi Digital oscilloscope, JITKIT jitter analysis package, and SDA-II data analyzer
Teledyne LeCroy | http://teledynelecroy.com

7904 Oscilloscope
Tektronix, Inc., available through distributors such as eBay (www.ebay.com)

by Jeff Bachiochi (USA)

Build a MIDI Communication Device (Part 1)

Introduction to MIDI

Designing and constructing a MIDI communication device doesn't have to be difficult. With a microcontroller and a little know-how, you can build a hardware circuit to monitor the MIDI traffic (message protocol) sent between MIDI devices. This article introduces the MIDI standard, the MIDI bus, and MIDI messages.

FROM THE BENCH

I recently acquired a used Harmonix drum accessory for an Xbox (see Photo 1). Microsoft's Xbox is based on a Windows machine and uses USB peripherals and an upgradable operating system. This keeps Xbox fresh and able to take on new peripherals (e.g., Kinect for "controller-free gaming"). The Xbox 360 has been able to hold its own against the top two competitors Sony's PlayStation 3 and Nintendo's Wii.

Rock Band and *Dance Dance Revolution* are two of the few non-fighting based video "games." The drum kit accessory is essentially a game controller with special drum pads that can trigger five game pad buttons. Other accessories (for *Rock Band*) include guitars, keyboards, and a microphone. In "Control Circuitry" (*Circuit Cellar* 214, 2008), I wrote about a guitar PlayStation 2 controller project. This article takes that project in a different direction. The unit I found has a USB interface, which I used to investigate the drum kit in Xbox's version of *Rock Band*.

Microsoft has a downloadable Windows driver that supports using an Xbox controller. With this driver installed, PC applications can take advantage of the controller as an input device. Andrew Rudson has a nice "drum machine" app, which is available on his website. With these (and speakers on your PC) the drum kit can trigger the drum machine and provide you with a usable set of "skins." With the drum kit you can assign .WAV files to each of the five trigger devices (i.e., four pads and a bass pedal).

It looks like each drumhead is in parallel with the X, Y, A, and B controller buttons. Photo 2 shows the PCB inside the drum kit. Figure 1 is a schematic of the trigger circuitry including the piezoelectric

Photo 1—Here is my Harmonix drum kit and my Yamaha SHS-10 keytar with MIDI-Out, which I use to investigate MIDI messages.

Photo 2—The drum kit's PCB is essentially a game controller with some extra triggering circuitry for each drum head. Its fifth input is a bass drum pedal.

device mounted inside each drum head. The controller buttons are a normal pulled-up input with a push button connected to the input and ground so the switch shorts the input to ground when it is pushed. The trigger circuit's output, a transistor, is used as a solid-state switch when it is placed in parallel with the controller's button. A significant signal on the transistor's base can turn the transistor on, shorting the input to ground just like the mechanical switch. When struck, the output device, used as a trigger sensor, develops enough energy to drive the transistor's base high.

As a *Rock Band* game controller, an input device must be simple. The game judges whether a user is pressing the right key or hitting the right pad at the right time. Expression is a quality that does not come into play. However, outside of gaming, when playing a real instrument, expression is an important part of the performance.

MIDI

What is the musical instrument digital interface (MIDI)? In the early 1990s, the MIDI Manufacturers Association (MMA) created the MIDI 1.0 specification providing a standardized design for synthesis technology. Musicians use a standardized way of notating music using a staff (pitch) and notes (duration) to produce sheet music. Any instrument can reproduce the melody from this sheet. It may have been intended for a specific instrument, which may be indicated on the sheet somewhere, but is not part of the music, per se. The MIDI standard takes into account not only the staff and notes, but also other details such as the intended instrument (general MIDI) and expression or dynamics. While sheet music visually communicates to a musician, MIDI communicates by passing instructions directly to specific instruments without the need of a musician.

Digital music files play back a recorded version of a song. MIDI files instruct the instruments how to play a song live. To use a MIDI file you need a device that can mimic every instrument required by the file. While this seems like it might be an unreachable end game, synthesis technology has come a long way. MIDI files are much smaller than audio files and only contain instructions. We may see them used more and more in the future. While a MIDI file contains MIDI messages, it is a format unto itself. (Refer to J. Glatt's "Standard MIDI File Format" in the Resources section at the end of this article.)

All MIDI messages or actions begin with a status byte followed by 1 or 2 data bytes (with the exception of SysEx). Every status byte has the most significant bit (MSB) set, while all remaining bytes have the MSB cleared. This enables you to easily sync up a status byte. MIDI messages are serially transmitted using a UART running at a 31,250-bps nonstandard baud rate. Yet the data format is the 1 start, 8 data bits, 1 stop, no parity with which we are all familiar. Electrically, the signal is in the form of a current loop to enable the required electrical isolation between devices using an optoisolator at each receiver. At the time the specification was developed, 1-MHz crystals were the norm and, being evenly divisible, 31.25 kbps was chosen as the MIDI specification's baud rate.

A MIDI controller module is a device that creates or documents its operation in real time, sending out MIDI messages (MIDI-Out) to indicate how a user is interacting with the device inputs. This is normally a digital or analog input, which might come from a switch or knob position (e.g., a key press, a trigger, a selector, or a sensor).

A MIDI sound module is a device that receives MIDI messages (MIDI-In) and interprets them into sound output. Sometimes a MIDI device will contain both a controller and sound modules. These devices may have both MIDI-Out and MIDI-In ports. Communication between two devices is a "one-way street" from a controller module (MIDI-Out) to a sound module (MIDI-In). A MIDI-Out or MIDI-Thru can be used to daisy chain multiple devices together to a MIDI-In port. A MIDI-Thru port is an optional port that passes on everything received on its MIDI-In port.

MIDI messages can be personally addressed to any of 16 channels. This means a MIDI controller module can separate instructions by instrument enabling a sound module set up for one instrument to ignore those instructions intended for a different instrument. To understand this more clearly, let's look at the MIDI message's makeup.

STATUS MESSAGES

If you did some math when I mentioned the MIDI message rule concerning the MSB, you probably thought about the limitations this rule puts on the data. Since every byte's MSB is pretty much spoken for, any remaining data is, therefore, limited to 7 bits (0–127). Large values require multiple bytes with values in 7-bit increments.

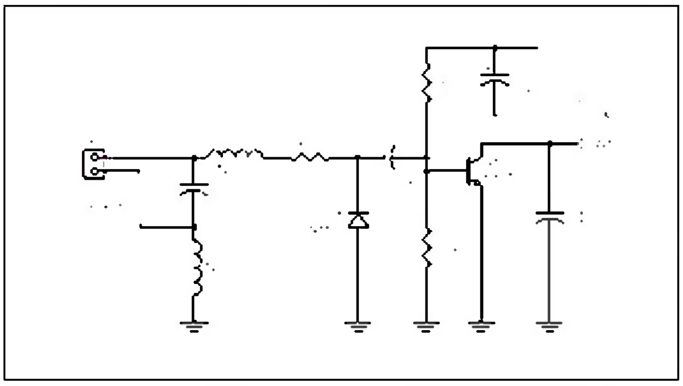


Figure 1—This schematic shows the triggering circuitry used for the piezoelectric device located in each drum head.

MSNibble Value	Status Command	Additional Data Bytes	Purpose
8 (0x8-)	Note on	2	Which key was pressed? With what velocity?
9 (0x9-)	Note off	2	Which key was released? With what velocity?
10 (0xA-)	After touch	2	Which key is involved? What is the change in pressure?
11 (0xB-)	Control change	2	Which controller is involved? What is the new value? See Table 2
12 (0xC-)	Program change	1	What program (instrument) should I use? See Table 3
13 (0xD-)	Channel pressure	1	What value should I use (i.e., volume or filter setting)?
14 (0xE-)	Pitch wheel	2	Raise or lower the pitch 0x2000 = no change, 0x1000 = lower pitch, 0x3000 = raise pitch
15 (0xF-)	SysEx	Multiple	See Table 5

Table 1—Every status byte's upper nibble indicates the instruction being issued. The lower byte indicates the channel or instrument for which the instruction is intended. A 0xF- status byte indicates the instruction is meant for everyone.

The MIDI message status byte (first byte, MSB = 1) carries two pieces of information with it, the upper nibble holds voice commands, and the lower nibble holds the channel number. The lower nibbles, Channel values 0–15, equal channel numbers 1–16. The MIDI message status byte 15 ("0xF-") is a special case that is used for messages that are meant for everyone. This is sort of like a conductor's general instructions to an orchestra. Table 1 is a list of the MIDI messages.

The data bytes, which follow a status instruction, are all 0–127 values because their MSB is always cleared. Once an instruction is completed, it may be repeated. In other words, the receipt of additional data bytes implies the same status byte thereby saving transmission time. The velocity (i.e., pressure used on a key) may be used to affect a note's volume or timbre (i.e., a light touch being low volume, while a heavy hit could be high volume). Touch-sensitive inputs are more expensive, so many devices use a fixed-velocity value. In fact, I've found many devices use only `Note On` messages where a "0" velocity value is used to turn the note off!

To add some expression to a played note, a musician may use the `Change Control` message to add a special effect like tremolo (i.e., vibrato) to the note (see Table 2). Some controls are a bit vague. These can be used for effects that are not covered by the specification.

At first release, there was no standard to indicate which instrument was being requested. As a result, the notes intended for a flute (see Table 3, Program 3), may be performed by a trombone if the sound module was designed using a trombone as the instrument for Program 3. The MIDI specification added an addendum to help standardize the use of program numbers for specific instruments.

One special case popped up as not fitting with the traditional instrument tuning of scaled keys and is often used in conjunction with most other instruments. It was decided that the percussive section should

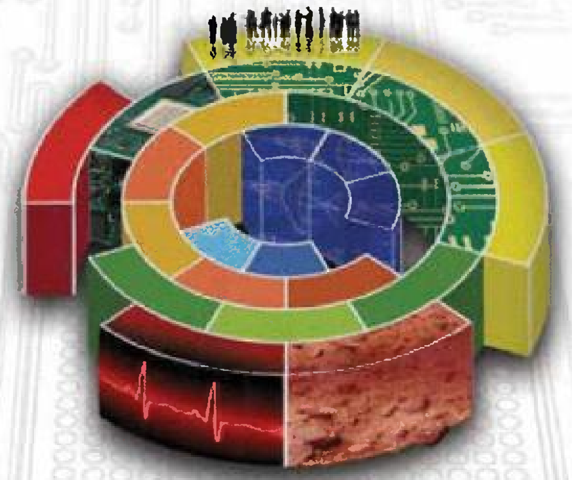
Control #	Function	Control #	Function
0	Bank select (coarse)	70	Sound variation
1	Modulation wheel (coarse)	71	Sound timbre
2	Breath controller (coarse)	72	Sound release time
4	Foot pedal (coarse)	73	Sound attack time
5	Portamento time (coarse)	74	Sound brightness
6	Data entry (coarse)	75	Sound control 6
7	Volume (coarse)	76	Sound control 7
8	Balance (coarse)	77	Sound control 8
10	Pan position (coarse)	78	Sound control 9
11	Expression (coarse)	79	Sound control 10
12	Effect control 1 (coarse)	80	General-purpose button 1 (on/off)
13	Effect control 2 (coarse)	81	General-purpose button 2 (on/off)
16	General-purpose slider 1	82	General-purpose button 3 (on/off)
17	General-purpose slider 2	83	General-purpose button 4 (on/off)
18	General-purpose slider 3	91	Effects level
19	General-purpose slider 4	92	Tremulo level
32	Bank select (fine)	93	Chorus level
33	Modulation wheel (fine)	94	Celeste level
34	Breath controller (fine)	95	Phaser level
36	Foot pedal (fine)	96	Data button increment
37	Portamento time (fine)	97	Data button decrement
38	Data entry (fine)	98	Nonregistered parameter (fine)
39	Volume (fine)	99	Nonregistered parameter (coarse)
40	Balance (fine)	100	Registered parameter (fine)
42	Pan position (fine)	101	Registered parameter (coarse)
43	Expression (fine)	120	All sound off
44	Effect control 1 (fine)	121	All controllers off
45	Effect control 2 (fine)	122	Local keyboard (on/off)
64	Hold pedal (on/off)	123	All notes off
65	Portamento (on/off)	124	Omni mode off
66	Sustenuto pedal (on/off)	125	Omni mode on
67	Soft pedal (on/off)	126	Mono operation
68	Legato pedal (on/off)	127	Poly operation
69	Hold 2 pedal (on/off)		

Table 2—The control instructions cover most of the sound parameters that go beyond starting and stopping a particular note. These are a combination of analog value) and digital (on/off) settings that could be used to modify each sound. While many analog controls have both course (MS7-bit) and fine (LS7-bit) adjustments, many devices will only use the course value.

designwest

Register today to attend the largest electronic engineering event in the United States.

150 conference sessions • 250+ exhibitors
3 keynotes • 80+ world class speakers



10,000 electronics design engineers, entrepreneurs, and technology professionals will be at DESIGN West.

Will you be there?

April 22-25, 2013
McEnery Convention Center
San Jose, CA

www.ubmdesign.com

Program (Value)	Instrument	Program (Value)	Instrument	Program (Value)	Instrument	Program (Value)	Instrument
	Piano		Chromatic Percussion		Organ		Guitar
1 (0x00)	Acoustic grand	9 (0x08)	Celesta	17 (0x10)	Drawbar organ	25 (0x18)	Nylon string guitar
2 (0x01)	Bright acoustic	10 (0x09)	Glockenspiel	18 (0x11)	Percussive organ	26 (0x19)	Steel string guitar
3 (0x02)	Electric grand	11 (0x0A)	Music box	19 (0x12)	Rock organ	27 (0x1A)	Electric jazz guitar
4 (0x03)	Honky-tonk	12 (0x0B)	Vibraphone	20 (0x13)	Church organ	28 (0x1B)	Electric clean guitar
5 (0x04)	Electric piano 1	13 (0x0C)	Marimba	21 (0x14)	Reed organ	29 (0x1C)	
6 (0x05)	Electric piano 2	14 (0x0D)	Xylophone	22 (0x15)	Accordian	30 (0x1D)	Overdriven guitar
7 (0x06)	Harpsichord	15 (0x0E)	Tubular bells	23 (0x16)	Harmonica	31 (0x1E)	Distortion guitar
8 (0x07)	Clavinet	16 (0x0F)	Dulcimer	24 (0x17)	Tango accordian	32 (0x1F)	Guitar harmonics
	Bass		Solo Strings		Ensemble		Brass
33 (0x20)	Acoustic bass	41(0x28)	Violin	49 (0x30)	String ensemble 1	57 (0x38)	Trumpet
34 (0x21)	Electric bass (finger)	42 (0x29)	Viola	50 (0x31)	String ensemble 2	58 (0x39)	Trombone
35 (0x22)	Electric bass (pick)	43 (0x2A)	Cello	51 (0x32)	Synth strings 1	59 (0x3A)	Tuba
36 (0x23)	Fretless bass	44 (0x2B)	Contrabass	52 (0x33)	Synth strings 2	60 (0x3B)	Muted trumpet
37 (0x24)	Slap bass 1	45 (0x2C)	Tremolo strings	53 (0x34)	Choir aahs	61 (0x3C)	French horn
38 (0x25)	Slap bass 2	46 (0x2D)	Pizzicato strings	54 (0x35)	Voice oohs	62 (0x2D)	Brass section
39 (0x26)	Synth bass 1	47(0x2E)	Orchestral strings	55 (0x36)	Synth voice	63 (0x3E)	Synth brass 1
40 (0x27)	Synth bass 2	48 (0x2F)	Timpani	56 (0x37)	Orchestra hit	64 (0x3F)	Synth brass 2
	Reed		Pipe		Synth Lead		Synth Pad
65 (0x40)	Soprano sax	73(0x48)	Piccolo	81 (0x50)	Lead 1 (square)	89(0x58)	Pad 1 (new age)
66 (0x41)	Alto sax	74 (0x49)	Flute	82 (0x51)	Lead 2 (sawtooth)	90 (0x59)	Pad 2 (warm)
67 (0x42)	Tenor sax	75 (0x4A)	Recorder	83 (0x52)	Lead 3 (calliope)	91 (0x5A)	Pad 3 (polysynth)
68 (0x43)	Baritone sax	76 (0x4B)	Pan flute	84 (0x53)	Lead 4 (chiff)	92 (0x5B)	Pad 4 (choir)
69 (0x44)	Oboe	77 (0x4C)	Blown bottle	85 (0x54)	Lead 5 (charang)	93 (0x5C)	Pad 5 (bowed)
70 (0x45)	English horn	78 (0x4D)	Skakuhachi	86 (0x55)	Lead 6 (voice)	94 (0x5D)	Pad 6 (metallic)
71 (0x46)	Bassoon	797(0x4E)	Whistle	87 (0x56)	Lead 7 (fifths)	95(0x5E)	Pad 7 (halo)
72 (0x47)	Clarinet	80 (0x4F)	Ocarina	88 (0x57)	Lead 8 (bass + lead)	96 (0x5F)	Pad 8 (sweep)
	Synth Effects		Ethnic		Percussive		Sound Effects
97 (0x60)	FX 1 (rain)	105(0x68)	Sitar	113 (0x70)	Tinkle bell	121(0x78)	Guitar fret noise
98 (0x61)	FX 2 (soundtrack)	106 (0x69)	Banjo	114 (0x71)	Agogo	122 (0x79)	Breath noise
99 (0x62)	FX 3 (crystal)	107 (0x6A)	Shamisen	115 (0x72)	Steel drums	123 (0x7A)	Seashore
100 (0x63)	FX 4 (atmosphere)	108 (0x6B)	Koto	116 (0x73)	Woodblock	124 (0x7B)	Bird tweet
101 (0x64)	FX 5 (brightness)	109 (0x6C)	Kalimba	117 (0x74)	Taiko drum	125 (0x7C)	Telephone ring
	FX 6 (goblins)	110 (0x6D)	Bagpipe	118 (0x75)	Melodic tom	126 (0x7D)	Helicopter
	FX 7 (echoes)	111 (0x6E)	Fiddle	119 (0x76)	Synth drum	127(0x7E)	Applause
	FX 8 (sci-fi)	112 (0x6F)	Shanai		Reverse cymbal	128 (0x7F)	Gunshot

Table 3—With the general MIDI addendum that assigns instruments to particular program values, every composer will be using the same set of instruments.

always be available to every program (instrument selection) by setting aside MIDI Channel 10 as the percussive section. This also would eliminate the need for "wasting" a program number for every possible percussive device. Table 4 shows how the percussive devices are assigned to specific keys on Channel 10 of every program selection. Each percussive device is triggered by its respective MIDI-assigned note. It should also be mentioned that there are percussive devices that are tuned to the scale, as seen in Table 3's chromatic percussive section.

The last type of status byte is the exception to the rule of MIDI messages dealing with a specific channel. Any status byte with an "F" in the upper nibble is treated as a system-wide message to all devices (see Table 5).

MIDI PROJECT

If you are a non-musician like me, you may find this MIDI stuff brand new. I thought a good place to start might be

Table 4—Since many percussion instruments do not play a particular note, each instrument can be assigned to a note and played by pressing that associated note. MIDI assigns Channel 10 of every program to this drum set, making it available with every instrument selection.

becoming more familiar with MIDI messages. I began my project with the circuitry necessary to connect to some MIDI communication and display the messages passed from one device to another. The MIDI specification not only defines the connectors used for MIDI-In, MIDI-Out, and MIDI-Thru, but also suggests opto-isolated circuitry that meets the specification's isolation requirement. I used a MIDI-In circuit connected to a microcontroller's serial port to read and interpret the communication bytes and display some text messages about what it saw. I wanted this to be a stand-alone device, and I expected the text messages to scroll rather quickly, so, a four-line LCD would not be sufficient. I needed to connect to my PC to take advantage of the large scrollable buffers offered by terminal programs. I decided a second serial port would be nice.

Microchip Technology's PIC24FV16KA302 microcontroller offers two serial ports and its I/O can be initialized to do inverse logic, which eliminates the need for Maxim Integrated Products MAX232 drivers/receivers. While the PIC24FV16KA302 is not RS-232 (±V) compliant, I found the logic level (0/+V) works with most serial interfaces. I used only two DIN connectors, MIDI-In and a shared MIDI-Thru/-Out that is configured with a jumper depending on how you use the project. Figure 2 shows the circuit I used for this first project phase. Note that the inverted RX2 input has a series resistor to limit any negative current applied to the pin from a –12-V RS-232 signal from the PC.

The MIDI selection jumper JP4 selects whether the MIDI-Out's driving source comes from the microcontroller or the MIDI-In connector. When the microcontroller is selected, the project became the source of MIDI messages, here I used MIDI-Thru, since I wanted the project to monitor and report all MIDI messages, yet enable them to pass through (i.e., daisy chain) to the next device.

This stage of the project only required the receipt of MIDI messages via UART1 and text transmission via UART2. However, it made sense to write the code to handle all four ring buffers required for the two UARTs. Two interrupt routines handle UART reception by using `RXBuffer HeadPointers` to place any received bytes into their respective buffers. Two more interrupt routines handle UART transmission by loading any available bytes from their respective buffers (`TXBufferTailPointers`) into their transmitters. I used the `main` program loop to add bytes to the TXBuffers and remove bytes from the `RXBuffers`. In the `main` loop, interrupts were temporarily disabled during any buffer operations to prevent a TX or RX interrupt from taking over and potentially destroying the Head and Tail pointers.

UART1 was initialized for 31,250 bps. UART2 was initialized for 115,200 bps and inverted logic. Other than that, the UART initialization and routines were identical. The `main` loop's function boiled down to a single operation, which was to check the RX1Buffer for a received character and determine what to do about it. Figure 3 shows a flowchart of the process. With the MSB set for all status bytes, it was easy to pick out each message's first byte. A bunch of comparisons determined which status message was being sent. The status byte could be determined from the upper nibble. The lower nibble provided the channel number. Each status byte has a fixed number of data bytes (1 or 2) that follow it. Most system messages have no data bytes. 0xF0 is an exception (start of SysEx message); it can have any number of data bytes. I used 3 as a data count and never decreased a count of 3. (This message's data ends

Status Command	Function
0xF0	SysEx start of message
0xF1	MTC Quarter frame message
0xF2	Song position pointer
0xF3	Song select
0xF4	(none)
0xF5	(none)
0xF6	Tune request
0xF7	SysEx end of message
0xF8	MIDI clock
0xF9	Tick
0xFA	MIDI start
0xFB	MIDI continue
0xFC	MIDI stop
0xFD	(none)
0xFE	Active sense
0xFF	Reset

Table 5—These system-exclusive status commands are intended for all MIDI channels. They can be used to send larger pieces of information and control MIDI files.

Figure 2—An opto-isolation circuit is used to receive MIDI messages and pass the message to the microcontroller and the MIDI-Thru output circuitry. The microcontroller determines the MIDI message and sends a text description to the secondary serial port for display on a PC.

when an 0xF7, end of SysEx message, is received.) While the SysEx messages can have many forms, for simplicity, I printed out each data byte within a wrapper "Start of SysEx message(...)End of SysEx message."

Photo 3 shows a display of the project's output when my Yamaha SHS-10 keytar was first turned on and the Demo button was pushed to play a multipart MIDI composition. My sign-on message is shown, followed by some status bytes that indicate which instruments should be played on which channels. Every 300 ms an "Active Sense" message was produced (if no other commands were sent) as an "I'm still alive" message. Once the Demo button is pressed, you can see other commands sent, notes played, and MIDI clocks that indicated time passing. The score's tempo in beats per minute (BPM) was used to define how often a MIDI clock message was sent. Quarter notes received 24 clocks. So, at 120 BPM (two beats per second), 24 clocks per 0.5 s indicate a quarter-note duration. Any particular note with 24 MIDI clocks between a Note On and its complement, a Note Off, would be a quarter note played for 0.5 s. Fewer MIDI clocks indicated faster notes (e.g., 8th, 16th, etc.).

WHAT'S NEXT?

Now that I have shown what kind of information is sent through a MIDI connection, you can appreciate how the MIDI standard has helped manufacturers get on the same page. MIDI controllers from multiple manufactures all look the same to every sound module used to reproduce the composition. While there are differences in the way instruments sound as interpreted by each manufacturer, at least when a French horn is called for, you don't get a cymbal crash.

In Part 2 of this article series, I'll finish up this project by adapting the *Rock Band* drum kit to supply the project with

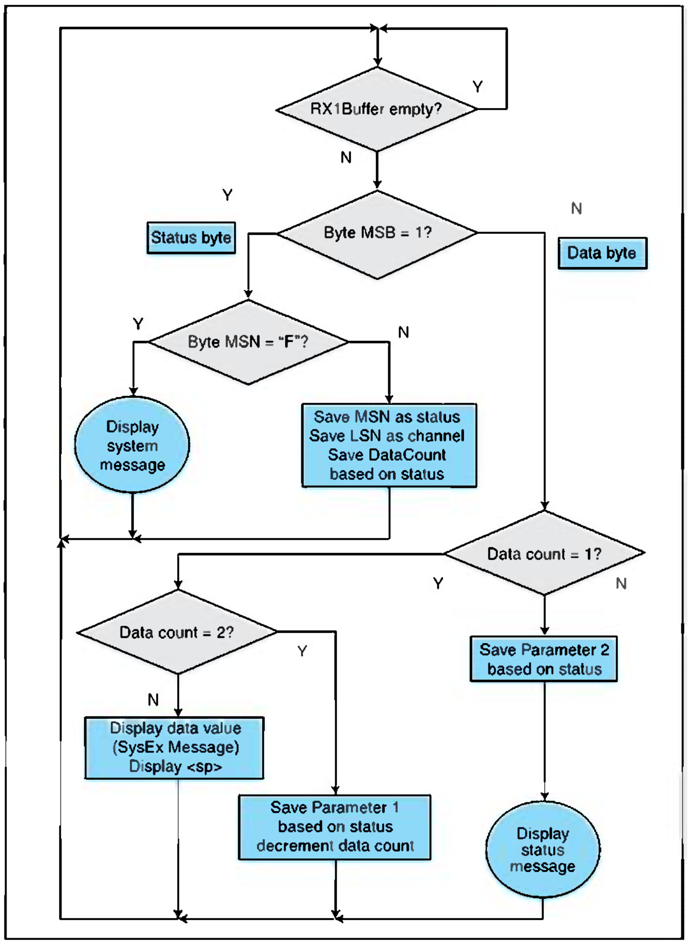


Figure 3—The `main` loop waits for MIDI bytes based on the MSB. Then it determines whether it's a new MIDI message or data bytes of the last message. 0xFx is a special case (SysEx) status byte, while 0x8x–0xEx are voicing status bytes. With the MSB cleared, the data byte values are 0x00–0x7F.

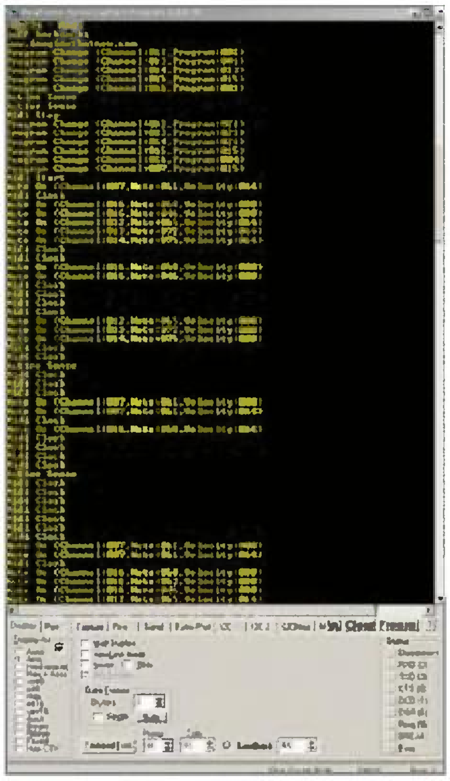


Photo 3—This screen shot shows the second serial port output displaying text messages describing each MIDI message received from the first serial port. The display's 115,200-bps rate enables descriptive text messages to keep up with the simple coded protocol of the MIDI port's 31,250-bps rate.

key (drum pad) triggers and produce MIDI-compliant output tha enables you to record your own MIDI tracks or play the drum set through your PC's sound system.

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

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2013/273.

RESOURCES

J. Bachiochi, "Control Circuitry," *Circuit Cellar* 214, 2008.

J. Glatt, "Standard MIDI File (SMF) Format," http://home.roadrunner.com/~jgglatt/tech/midifile.htm.

Microsoft Corp., Download Center, "Xbox 360 Controller for Windows Software," www.microsoft.com/en-us/download/confirmation.aspx?id=34001.

A. Rudson, "Drum Machine," http://andrewrudson.com/drummachine/main.php.

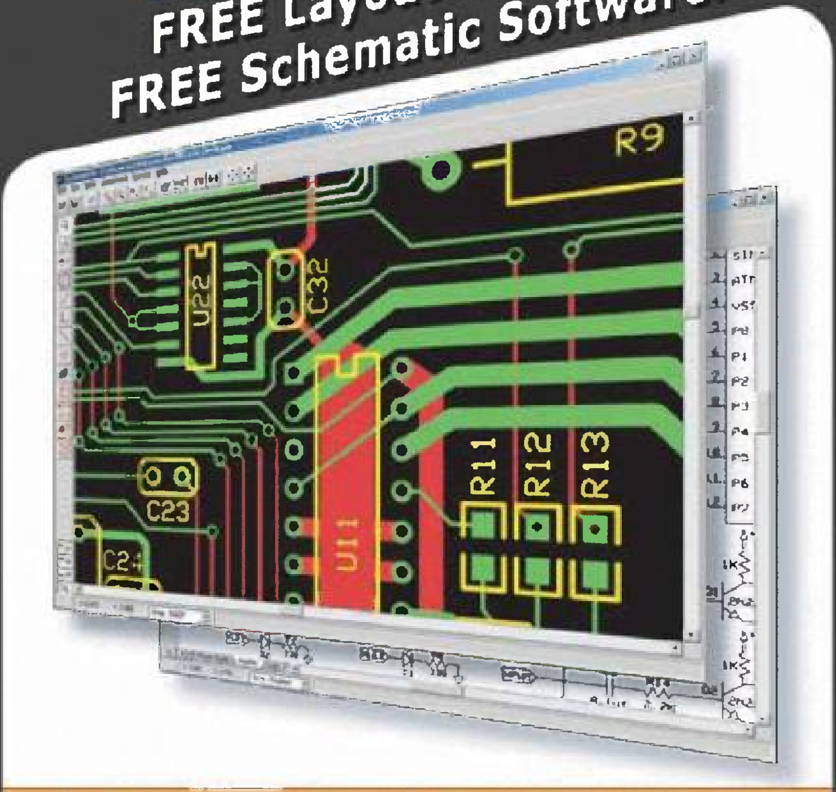
SOURCES

MAX232 Drivers/receivers
Maxim Integrated Products | www.maximintegrated.com

PIC24FV16KA302 Microcontroller
Microchip Technology, Inc. | www.microchip.com

$51 For 3 PCBs

FREE Layout Software!
FREE Schematic Software!



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

expresspcb.com

In celebration of *Circuit Cellar*'s 25th year, we're running an article from the archives each month that exemplifies something special about this magazine, its contributors, and its readers. We hope you'll enjoy reading (or perhaps rereading) these innovative projects as much as we did preparing them. This month, we feature a 2003 article about using Verilog HDL in your custom logic designs.

by Mark Balch
Circuit Cellar 158, 2003

Microprocessor Glue Logic with Verilog HDL

For reasons of availability and practicality, you may soon find it necessary to design custom logic for your digital projects. There are various design techniques to choose from, but you'll want one that suits your specific needs. Mark explains how Verilog HDL may prove to be the perfect solution for your more complex digital designs.

A typical digital system includes a microprocessor complemented by peripherals and miscellaneous glue logic that ties the components together. Many of the peripheral and logic functions are available in off-the-shelf ICs. A variety of UART ICs and DMA controllers are available, too. Simple address decoding is accomplished with 7400 devices such as the 74LS138. But, as digital systems become more complex, the chances increase that suitable off-the-shelf logic will become either unavailable or impractical. The answer is to design and implement custom logic rather than rely solely on a third party to deliver the perfect solution.

Logic design techniques differ according to the scale of logic that's implemented. If only a few gates are needed to implement a custom address decoder or timer, the most practical solution may be to write down truth tables, extract Boolean equations, select appropriate 7400 devices, and draw a schematic diagram. This used to be the predominant means to design logic for many applications. The original Apple and IBM desktop computers were designed in this way, as witnessed by their rows of 7400 ICs.

When functions grow more complex, it becomes awkward and often impossible to implement the necessary logic using discrete 7400 devices. Reasons vary from simple density constraints (How much physical area would be consumed by dozens of 7400 ICs?) to propagation-delay constraints (How fast can a signal pass through multiple discrete logic gates?). A common solution to these problems requires the implementation of application-specific logic in a programmable logic device, or PLD.

After you decide to implement logic within a PLD, you'll need a design methodology to move ahead and solve the problem at

Listing 1—Verilog gate, or instance, level design entails the manual connection of logical entities in a netlist-like form. A typical design uses this style for interconnecting hierarchical blocks rather than actually creating Boolean equations.

```
module my_logic (
  A, B, C, Y
);

input A, B, C;
output Y;

wire and1_out, and2_out, notA;

and_gate u_and1 (
  .in1 (A),
  .in2 (B),
  .out (and1_out)
);

not_gate u_not (
  .in  (A),
  .out (notA)
);

and_gate u_and2 (
  .in1 (notA),
  .in2 (C),
  .out (and2_out)
);

or_gate u_or (
  .in1 (and1_out),
  .in2 (and2_out),
  .out (Y)
);

endmodule
```

Listing 2—RTL level design is the principal method of inferring logic (e.g., Boolean equations and flip-flops). The same logic function (here a 2:1 multiplexer) may be implemented in multiple ways. Note the distinction between wire and reg types: wires are continuously assigned, and regs are assigned when discrete events (signal transitions) occur. Events are triggered through an always block's sensitivity list, which contains a list of the block's casual signals.

```
module my_logic (
  A, B, C, Y
);

input A, B, C;
output Y;
//Style 1: Continuous assignment
assign Y = (A && B) || (!A && C);
//Style 2: Behavioral assignment
reg Y;

always @(A or B or C)
begin
  Y = (A && B) || (!A && C);
end
//Style 3: If-then construct
reg Y;

always @(A or B or C)
begin
  if (A)
    Y = B;
  else
    Y = C;
end

endmodule
```

hand. It is possible to use the same design techniques as those used for discrete 7400 logic implementations. The trouble with graphical logic representations is that they are bulky and prone to human error. Hardware description languages (HDL) were developed to ease the implementation of more complex digital designs by representing logic with high-level semantic constructs found in mainstream computer programming languages. One of the major HDLs in use today is Verilog, which began as a proprietary product and was eventually transformed into an open standard.

Listing 3—Flip-flops are inferred using an edge-sensitive clock event in the sensitivity list. posedge and negedge are provided by Verilog to indicate a rising- or falling-edge sensitive flip-flop. When a reset signal is included in the sensitivity list, an asynchronous reset is inferred.

```
//Synchronous reset
always @(posedge CLK)
begin
  if (RESET)  //RESET evaluated only at CLK rising edge
    Q <= 1'b0;
  else
    Q <= D;
end
//Asynchronous reset
always @(posedge CLK or posedge RESET)
begin
  if (RESET)  //RESET evaluated whenever it goes active
    Q <= 1'b0;
  else
    Q <= D;
end
```

GATE-LEVEL DESIGN

HDLs enable logical representations that are abstracted to varying degrees. According to either your preference or contextual requirements, logic can be represented at the gate/instance level, register transfer level (RTL), or behavioral level.

Gate/instance-level representations involve the manual instantiation of each physical design element. An element can be an AND gate, flip-flop, multiplexer, or an entire microprocessor. These decisions are left up to you. In a purely gate/instance-level HDL design, the HDL source code is nothing more than a glorified list of instances and connections between the I/O ports of each instance. The Verilog instance representation of Y = A & B + A & C is shown in Listing 1. It is somewhat cumbersome but provides full control over the final implementation.

Listing 1 incorporates many of the basic pieces of a generic Verilog module. First, the module is named and declared with its list of ports. Following the port list, the ports are defined as either inputs or outputs. In this case, the ports are all single net vectors, so no indices are supplied. Next is the main body that defines the function of the module. Verilog recognizes two major variable types: wire and reg. A wire simply connects two or more entities together. A reg can be assigned values at discrete events. When ports are defined, they are assumed to be wire unless declared otherwise. An output port can be declared as a type other than wire.

This example is a gate/instance-level design, so all logic is represented by instantiating other modules that have been defined elsewhere. A module is instantiated by invoking its name and following it with an instance name. Here, the common convention of preceding the name with u_ is used. Following the name with a number differentiates multiple instances of the same module type. Individual ports for each module instance are explicitly connected by referencing the port name prefixed with a period and placing the connecting variable in parentheses.

Listing only the connecting variables in the order in which a module's ports are defined can implicitly connect ports. This is generally considered poor practice because it is prone to mistakes and difficult to read.

SYNTHESIS AND SIMULATION

HDL's textual representation of logic is converted into actual gates through a process called logic synthesis. A synthesis program parses the HDL code and generates a netlist that contains a detailed list of low-level logic gates and their interconnecting nets, or wires. Synthesis is usually achieved with a specific implementation target in mind because each implementation technology differs in the logic primitives it provides as basic building blocks. After synthesis is performed, the netlist can be mapped into a PLD.

A key benefit of HDL design methodology is the ability to thoroughly simulate logic before having to debug the circuit in the lab. Because HDL is a programming methodology, it can be arbitrarily manipulated in a software simulation environment. The simulator allows a test bench to be written in either the HDL or another language (e.g., C/C++) that is responsible for creating a stimulus to be applied to the logic modules.

A distinction is made between synthesizable and nonsynthesizable code when writing RTL and test benches. Synthesizable code represents the logic to be implemented in some type of chip. Nonsynthesizable code is used to implement the test bench and usually contains constructs specifically designed for simulation that cannot be converted into real logic through synthesis.

An example of a test bench for the preceding Verilog module may consist of three number generators that apply a pseudorandom test stimulus to the three input ports. Automatic verification of the logic would be possible by having the test bench independently compute the function Y = A & B + A & C and then check the result against the module's output. You can use such simulation, or verification, techniques to root out the majority of bugs in a complex design. This is a tremendous feature because it is usually faster to isolate and fix a bug in simulation than in the laboratory. In simulation, you have immediate access to all of the design's internal nodes. In the lab, such access may prove difficult to achieve.

Verilog supports simulation constructs that facilitate the writing of effective test benches. It is important to realize that these constructs are usually nonsynthesizable (e.g., a random number generator), and they should be used for writing test code rather than actual logic.

REGISTER TRANSFER LEVEL

Gate/instance-level coding is used to varying degrees in almost every design, but the real power of HDL lies at the RTL and behavioral levels. Except in rare circumstances when absolute control over gates is required, instance-level coding is mainly used to connect different modules together. Most logic is written in RTL and behavioral constructs, and they are often treated together, hence the reason that synthesizable HDL code is often called RTL.

Expressing logic in RTL frees you from having to break everything down into individual gates, and it transfers this responsibility to the synthesis software. The result is a dramatic increase in productivity and maintainability because logical representations become concise. You can rewrite Listing 1 in Verilog RTL in multiple styles, as shown in Listing 2.

Each of the three styles has its advantages, and each is substantially more concise and readable than the gate/instance-level version. You can freely mix the styles within the same module according to your preference. The first style is a continuous assignment; it makes use of the

Listing 4—Blocking and nonblocking assignments can result in different logic inferences. Nonblocking assignments are preferred for sequential logic (flip-flops), whereas blocking assignments are preferred for describing combinatorial logic paths.

```
//Non-blocking assignments: two flops inferred
always @(posedge CLK)
begin
  Q1 <= D;
  Q2 <= Q1;
end
//Blocking assignments: one flop inferred
always @(posedge CLK)
begin
  Q1 = D;
  Q2 = Q1;
end
```

Listing 5—This example of basic asynchronous address decoding uses the case construct to activate individual chip-selects when specific address ranges are active on the address bus. All driven signals are assigned default values so that each enumerated address range does not have to explicitly assign each signal. Undesirable latches are formed in a combinatorial always block when there exists at least one combination of inputs that does not result in a signal being assigned a value.

```
module GlueLogic (
  Addr,
  RomSel,
  CS0_,
  CS1_,
  CS2_,
  CS3_
);
input  [23:20] Addr;
input          RomSel;
output         CS0_, CS1_, CS2_, CS3_;

reg            CS0_, CS1_, CS2_, CS3_;
reg            IntSel;

always @(Addr or RomSel)
begin
  CS0_   = 1'b1;  //Establish default values to simplify case
  CS1_   = 1'b1;  //Statement and prevent formation of latches
  CS2_   = 1'b1;
  CS3_   = 1'b1;
  IntSel = 1'b0;

  case (Addr[23:20])
    4'b0000 : begin
                CS0_ = RomSel;
                CS1_ = !RomSel;
              end
    4'b0001 : begin
                CS0_ = !RomSel;
                CS1_ = RomSel;
              end
    4'b0010 : CS2_   = 1'b0;
    4'b0011 : CS3_   = 1'b0;
    4'b0100 : IntSel = 1'b1;
  endcase
end

endmodule
```

default wire data type for the output port. A wire is applicable here because it implicitly connects two entities: the logic function and the output port. Continuous assignments are useful in certain cases because they are concise, but they cannot get too complex without becoming unwieldy.

The second style uses the `always` block, which is a keyword that tells the synthesis and simulation tools to perform the specified operations whenever a variable in its sensitivity list changes. The sensitivity list defines the variables that are relevant to the `always` block. If all the relevant variables are not included in the list, incorrect results may occur.

Note that `always` blocks are one of Verilog's fundamental constructs. A design may contain numerous `always` blocks, each of which contains logic functions that are activated when a variable in the sensitivity list changes state. A combinatorial `always` block should normally include all of its input variables in the sensitivity list. Failure to do so can lead to unexpected simulation results because the `always` block will not be activated if a variable changes state and is not in the sensitivity list.

The third style also uses the `always` block, but it uses a logical `if-else` construct in place of a Boolean expression. Such logical representations are often preferable so you can concentrate on the functionality of the logic rather than deriving and simplifying Boolean algebra.

FLIP-FLOPS

The two examples I've shown you illustrate basic Verilog HDL syntax with combinatorial logic. Clearly, synchronous logic is critical to digital systems, and it is fully supported by HDLs. D-type flip-flops are most commonly used in digital logic design, and they are directly inferred by using the correct RTL syntax. As always, gate-level instances of flops can be invoked, but this is discouraged for the reasons already discussed. Listing 3 shows the Verilog RTL representation of two flops, one with a synchronous reset and the other with an asynchronous reset.

The first syntactical difference to notice is the Verilog keyword `posedge`. Note that `posedge` and its complement, `negedge`, modify a sensitivity list variable to activate the `always` block only when it transitions. Synthesis tools are smart enough to recognize these keywords and infer a clocked flop. Clocked `always` blocks should not include normal `reg` or `wire` variable types in the sensitivity list, because it is only desirable to activate the block on the active clock edge or when an optional asynchronous reset transitions.

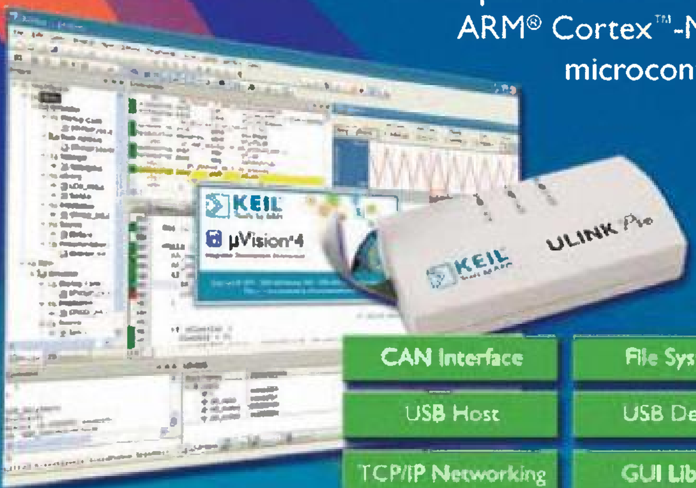
At reset, a default zero value is assigned to `Q`. Constants in Verilog can be explicitly sized and referenced to a particular radix. Preceding a constant with `'b` denotes it as binary. (`'h` is hex, and `'d` is decimal.) Preceding the radix identifier with a number indicates the number of bits the constant occupies.

Another syntactical difference to note is the use of a different type of assignment operator, `<=` instead of `=`. This is known as a nonblocking (<=) assignment as compared to a blocking (=) assignment. It is considered good practice to use nonblocking assignments when inferring flops because the nonblocking assignment doesn't take effect until after the current simulation time unit. This is analogous to the behavior of a real flop, where the output does not transition until a finite period of time has elapsed after its triggering event. Under certain circumstances, either type of assignment will yield the same result in both simulation and synthesis. In other situations, the results will differ, as shown in Listing 4.

In the first case, `reg` variable types `Q1` and `Q2` are tracked at two different instants in time. First, their current states are maintained as they were just prior to the clock edge for the purpose of using their values in subsequent assignments. Second, their new states are assigned as dictated by the RTL. When `Q2` is assigned,

Keil MDK-ARM

MDK-ARM™ is the complete development environment for ARM® Cortex™-M series microcontrollers.



CAN Interface	File System
USB Host	USB Device
TCP/IP Networking	GUI Library

ARM

www.keil.com/mdk

☎ +49 89 456040-20

©ARM Ltd.AD366 | 01.13

it takes the previous value of Q1, not the new value of Q1, which is D. Two flops are inferred.

In the second case, variables Q1 and Q2 are tracked at a single instant in time. Q1 is assigned the value of variable D, and then Q2 is assigned the new value of variable Q1. Q1 has become a temporary placeholder and has no real effect on its own; therefore, only a single flop, Q2, is inferred.

ADDRESS DECODING

Most digital systems require a quantity of miscellaneous glue logic to help tie a CPU to its memory and I/O peripherals. Some common support functions are address decoding, basic I/O signals, interrupt control, and timers.

Address decoding is usually a combinatorial implementation because many CPU interfaces are non-pipelined. When performing address decoding and other bus-control functions for a pipelined CPU bus, a more complex synchronous circuit is called for that can track the various pipeline stages and take the necessary actions during each stage. Basic combinatorial address decoding consists of mapping ranges of addresses to chip selects. Chip select signals are usually active low by convention and are numbered upwards from zero. For the sake of discussion, consider the 24-bit memory map in Table 1 to design an address decoder.

Four external chip selects are called out. Instead of using the asterisk to denote active-low signals, the underscore is used because an asterisk is not a valid character for use in a Verilog identifier. The first two chip selects are used for ROM (e.g., flash memory or EPROM), and their memory ranges are swappable according to the RomSel signal.

Sometimes it's useful to provide an alternate boot ROM that can be installed at a later date for various purposes such as a software upgrade. When boot ROM is implemented in flash memory, the CPU is able to load new data into its ROM. If there is no other way to send a new software image to the system, the image can be loaded onto a ROM module that is temporarily installed into the CS1_ slot. Then a jumper can be installed that causes RomSel to be asserted. When the system is turned on, RomSel=1 causes the ROM module to become the boot ROM, and new software can be loaded into CS0_ ROM.

The remainder of the address space is sparsely populated. Occupied memory regions are spread out to reduce the complexity of the decoding logic by virtue of requiring fewer address bits. If the UART was located immediately after the SRAM, the logic would have to consider the state of A[23:16] rather than just A[23:20]. The fifth and final used memory region is reserved for internal control and status registers. This decoding logic can be written in Verilog (see Listing 5).

The address decoding logic was written in behavioral form with a case construct. Case statements enable actions to be associated with individual states of a causal variable. Note that the chip select outputs are declared as regs even though they are not flops because they are assigned in an always block instead of in a continuous assignment.

Address Range	Qualifier	Chip Select	Function
0x000000-0x0FFFFF	RomSel=0	CS0_	1-MB default boot ROM
0x100000-0x1FFFFF	RomSel=0		
0x100000-0x1FFFFF	RomSel=1	CS1_	1-MB ROM module
0x000000-0x0FFFFF	N/A		
0x200000-0x21FFFF	N/A	CS2_	128-KB SRAM
0x220000-0x2FFFFF	N/A	None	Unused
0x300000-0x30000F	N/A	CS3_	UART
0x300010-0x3FFFFF	N/A	None	Unused
0x400000-0x4FFFFF	N/A	Internal	Control/status registers
0x500000-0xFFFFFF	N/A	None	Unused

Table 1—An example 16-MB memory map illustrates how address decoding can be performed. Two swappable ROM banks are present along with sparsely located peripherals for easier decoding.

Prior to the case statement, all of the always block's outputs are assigned to default inactive states. This prevents the synthesis software from inferring an unwanted latch instead of simple combinatorial logic. If a combinatorial always block does not assign a reg value for all input combinations, the synthesis tool determines that the reg should hold its previous state, thereby creating a latch. Latches are prevented by either exhaustively listing all combinations of inputs or assigning a default value to all variables somewhere in the always block.

Unintended latches are the bane of HDL design. There are valid instances when a latch is desired, but latches are often mistakenly inferred because the RTL code does not properly handle default cases where the variable is not assigned. Combinatorial logic must always assign values to variables regardless of the logic path. Otherwise, statefullness is implied. Verilog's blocking assignments enable multiple values to be assigned to a single variable in an always block, and the last assigned variable is the one that takes effect. Therefore, latches are avoided by assigning default values up front. An active-high signal, IntSel, is decoded but not used (yet) for selecting internal control and status registers.

BUS EXPANSION

When expanding a CPU bus, make sure that too many devices aren't on the bus, because output pins are only rated for certain drive strengths. As the interconnecting wires lengthen and the number of loads increases, it may become necessary to extend the CPU bus using bidirectional buffers.

Figure 1 shows how my hypothetical system might use such buffers to isolate the plug-in ROM module so that the electrical impact of connectors and a separate module are minimized. Control signals are not shown for clarity. A unidirectional buffer isolates the address bus, and a bidirectional buffer isolates the data bus. No control is necessary for the address buffer because it can be configured to always pass the address bus to the ROM module socket. However, the data buffers require control because they must direct data out to the module for writes and in from the module for reads. Therefore, the tristate control of the data buffer must be operated according to the address decode and read/write status.

The existing address decoding logic can be augmented to provide the necessary functionality. One additional input is required—the CPU's active-low read enable signal. An additional output is required to operate the data buffer's direction select signal. When high, the buffers pass data from the CPU side to the ROM; when low, the buffers drive the CPU data bus with data presented by the ROM. A second always block can be added (see Listing 6). New port and variable declarations are assumed.

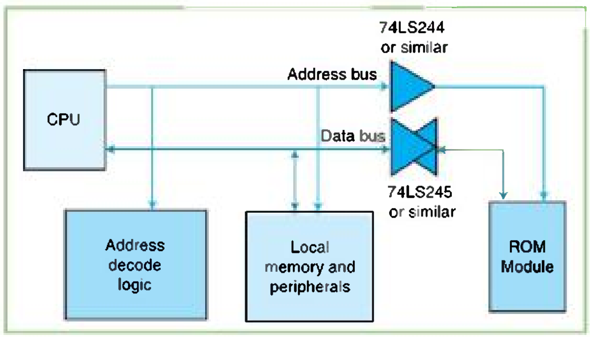


Figure 1—A CPU bus may need extension buffers if it is too long or there are more loads than the I/O drivers can handle. Data bus buffers must be bidirectional/tristate-capable to drive write data to the expansion module and drive read data to the CPU.

READ/WRITE REGISTERS

Another common function of support logic is to provide general I/O signals that the CPU can use to interact with its environment. Such interaction can include detecting an opening door and turning on an alarm. The opening door can be detected using a switch connected to an input signal. When the CPU reads the status of this signal, it can determine whether the switch is open or closed. An alarm can be turned on when the CPU sets an output signal that enables an alarm circuit. Control and status registers must be implemented to enable the CPU to read and write I/O signals.

In my example, I assume an 8-bit data bus coming from the CPU, as well as the need for eight input signals and eight output signals. Implementing registers varies according to whether the CPU bus is synchronous or asynchronous. Some older microprocessors use asynchronous buses requiring latches to be formed within the support logic. Listing 7 shows the implementation of two registers using the previously decoded IntSel signal in both synchronous and asynchronous styles. Again, the proper declarations for ports and variables are assumed.

An added level of address decoding is required in this to ensure that the two registers are not accessed simultaneously. The register logic consists of two basic sections: the write logic and read logic.

The write logic, which is only required for the control register that drives output signals, transfers the contents of the CPU data bus to the internal register when the register is addressed and the write enable is active. The Control RegSel signal is implemented in a case statement, but it can be implemented in a variety of ways. The asynchronous write logic infers a latch because not all permutations of input qualifiers are represented by assignments. If Reset_ is high and the control register is not being selected for a write, there is no specified action. Therefore, memory is implied, and, in the absence of a causal clock, a latch is inferred.

The synchronous write logic is almost identical, but it references a clock that causes a flop inference. Reset is implemented to provide a known initial state. This is a good idea so that external logic driven by the control register can be designed safely, assuming the operations begin at a known state. The known state is usually inactive so that peripherals do not start operating before the CPU finishes booting and can disable them.

The read logic consists of two sections: the output multiplexer and the output buffer control. The former simply selects one of the available registers for reading. It is not necessary to qualify the multiplexer with any other logic, because a read will not actually take place unless the output buffer control logic sends the data to the CPU. Rather than preventing a latch in ReadData by assigning it a default value before the case construct, the Verilog keyword default is used as the final case enumeration to specify default operation. Either solution will work; it is a matter of preference and style over which to use in a given situation.

Both read-only and writeable registers are included in the read multiplexer logic. Strictly speaking, it is not mandatory to have writeable register contents readable by the CPU, but it is a good practice. Years ago, when logic was extremely expensive, it was not uncommon to find write-only registers. However, there is a substantial drawback to this approach: you can never be sure what the contents of the register are if you fail to keep track of the exact data that has already been written!

Listing 6—A bidirectional data bus buffer can be controlled based on the chip select and read enable signals. When a read is performed, the buffer is turned to drive data toward the CPU. In all other circumstances, the buffer drives data from the CPU.

```
always @(CS1_ or Rd_)
begin
  if (!CS1_ && !Rd_)
   DataBufDir = 1'b0;
//Drive CPU bus when ROM selected for read,
else
   DataBufDir = 1'b1; //Otherwise always drive data to ROM
end
```

BIDIRECTIONAL PINS

Implementing bidirectional signals in Verilog can be achieved with a continuous assignment that selects between driving an active variable or a high-impedance value, Z. The asynchronous read logic is simple: whenever the internal registers are selected and read enable is active, the tristate buffer is enabled and the output of the multiplexer is driven to the CPU data bus. At all other times, the data bus is held in a high-impedance state. This works as expected because the value Z can be overridden by another assignment. In simulation, the other assignment may come from a test bench that emulates the CPU's operation. In synthesis, the software properly recognizes this arrangement as inferring a tristate bus.

The continuous assignment takes advantage of Verilog's conditional operator, ? : , which serves an if-else function. When the logical expression before the question mark is true, the value before the colon is used; otherwise, the value after the colon is used. A bidirectional port is declared using the Verilog inout keyword in place of input or output. The synchronous version of the read logic is similar to the asynchronous version, except that the outputs are first registered before being used in the tristate assignment.

Listing 7—Register read/write logic can be implemented in asynchronous or synchronous fashion according to the application's needs. In both cases, a read multiplexer selects a single register. The asynchronous circuit does not register the outgoing data bus and forms latches for the writeable register. The synchronous circuit registers outputs and uses flip-flops for the writeable register.

```
always @(Addr[3:0] or StatusInput[7:0] or ControlReg[7:0] or IntSel)
begin
  case (Addr[3:0])              //Read multiplexer
    4'h0    : ReadData[7:0] = StatusInput[7:0];    //External input pins
    4'h1    : ReadData[7:0] = ControlReg[7:0];
    default : ReadData[7:0] = 8'h0;     //Alternate means to prevent latch
  endcase

  ControlRegSel = 1'b0;         //Default inactive value
  case (Addr[3:0])
//Select signal only needed for writeable registers
    4'h1 : ControlRegSel = IntSel;
  endcase
end
//Option 1A: asynchronous read logic
assign CpuData[7:0] = (IntSel && !Rd_) ? ReadData[7:0] : 8'bz;

//Option 1B: synchronous read logic
always @(posedge CpuClk)
begin
  if (!Reset_)                  //Synchronous reset
    CpuDataOE <= 1'b0;
//No need to reset ReadDataReg and possibly save some logic
  else begin
    CpuDataOE    <= IntSel && !Rd_;
                                //All outputs are registered
    ReadDataReg[7:0] <= ReadData[7:0];
  end
end
assign CpuData[7:0] = CpuDataOE ? ReadDataReg[7:0] : 8'bz;
//Option 2A: asynchronous write logic
always @(ControlRegSel or CpuData[7:0] or Wr_ or Reset_)
begin
  if (!Reset_)
    ControlReg[7:0] = 8'h0; //Reset state is cleared
  else if (ControlRegSel && !Wr_)
    ControlReg[7:0] = CpuData[7:0];
//Missing else forces memory element: intentional latch!
end
//Option 2B: synchronous write logic
always @(posedge CpuClk)
begin
  if (!Reset_)                  //Synchronous reset
    ControlReg[7:0] <= 8'h0;
  else if (ControlRegSel && !Wr_)
    ControlReg[7:0] <= CpuData[7:0];
end
```

TIP OF THE ICEBERG

Basic microprocessor support functions including interrupt control and timers are the next step beyond what has been presented. Going further, multiphase bus protocols and data-processing algorithms are often implemented with finite state machines written and simulated in an HDL design environment. Choosing to develop logic with an HDL does not force a one-size-fits-all approach. Rather, it enables you to take advantage of specific HDL features that are appropriate for each situation. Small address decoders in a PAL-type device may not require the development overhead of a simulation test bench. Moderately complex glue logic and bus interfaces can be simulated to whatever detail is considered practical.

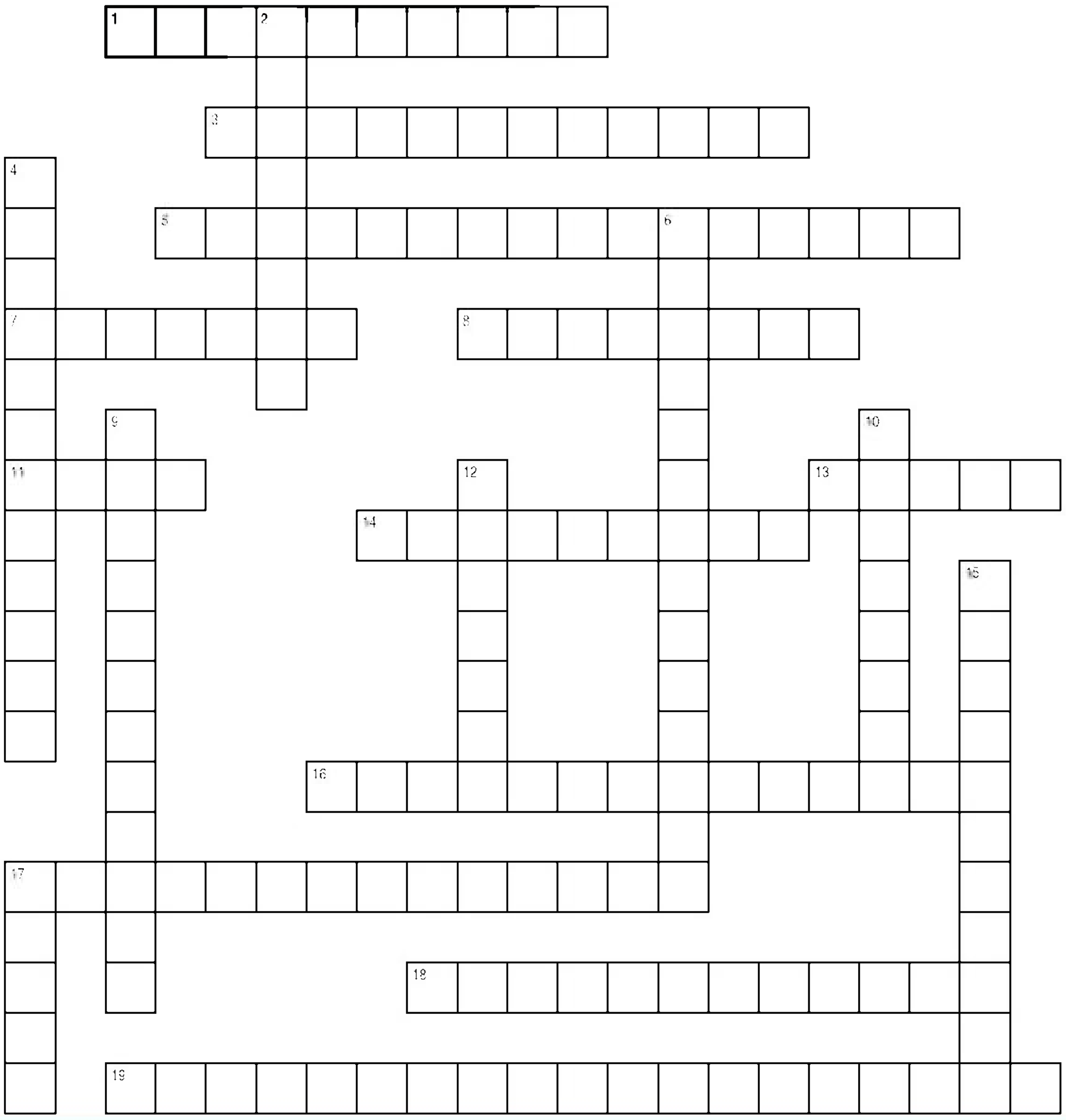
HDL simulation and synthesis tools are available in a wide range of prices and capabilities. Top-of-the-line tools run into the tens of thousands of dollars per seat. For those on a tight budget, these tools are commonly bundled into relatively low-cost development packages by PLD vendors to encourage the use of their chips. Such vendors include Actel, Altera, Cypress, Lattice, QuickLogic, and Xilinx. Once solely the domain of high-budget chip development, the HDL design methodology has become practical across the full spectrum of logic design.

Mark Balch is a Silicon Valley-based author and electrical engineer. He holds a BSEE from The Cooper Union in New York City. In addition to PCB, FPGA, and ASIC design, he has developed products for the HDTV, consumer electronics, and industrial computers industries. Currently, Mark designs high-performance computer-networking hardware. You may reach him at mark_balch@hotmail.com.

CROSSWORD

April 2013

The answers will be available in the next issue and are posted at circuitcellar.com/crossword



Across

1. Bits per inch of magnetic tape, for example [two words]
3. aka LQE [two words]
5. *Circuit Cellar* columnist Patrick Schaumont covers this topic with articles about authentication, encryption, and electronic signatures [two words]
7. Error-correcting code
8. Unintentional
11. Electronics and Computer Engineering professor at Cornell University and *Circuit Cellar* frequent contributor (many of his students contribute to *Circuit Cellar* as well)
13. American inventor who developed an oscillator frequency standard
14. Able to interact with nonmetallic circuit parts
16. A feedback structure with an odd number of digital inverters [two words]
17. A system that's not quite in synch
18. Takes electric current measurements
19. Created by Bell Labs physicist and researcher Frank Gray [three words]

Down

2. Potentiometer's part shortened by a tap [two words]
4. Electronics engineer and co-founder of the company that created Java
6. Digital fingerprints [two words]
9. Illuminating
10. Computer networking system from the 1970s
12. Where to find electronics engineer David L. Jones's off-the-cuff online videos
15. Solves equations [two words]
17. Helped make electromagnetic radio waves more useful

IDEA BOX

THE DIRECTORY OF PRODUCTS AND SERVICES

AD FORMAT: Advertisers must furnish digital files that meet our specifications (circuitcellar.com/mediakit).
ALL TEXT AND OTHER ELEMENTS MUST FIT WITHIN A 2" x 3" FORMAT. E-mail adcopy@circuitcellar.com with your file.
For current rates, deadlines, and more information contact Peter Wostrel at 978.281.7708 or peter@smmarketing.us.

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

ANALOG and DIGITAL
for PC/104

GPIO-104
8 ANALOG INPUTS
4 ANALOG OUTPUTS
24 DIGITAL I/O

Need More Analog I/O?

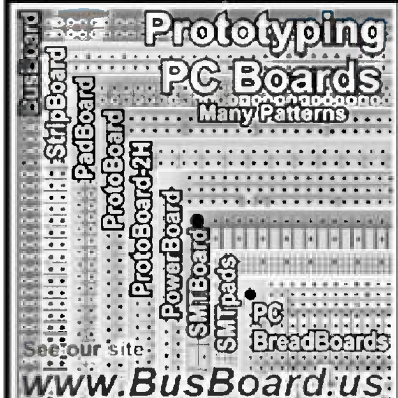
ADIO-104
16 ANALOG INPUTS
8 ANALOG OUTPUTS
24 DIGITAL I/O
PULSE ACCUMULATOR

SCIDYNE®

Also Available
• 96 DIGITAL I/O
• 24 HIGH POWER I/O
• RELAY/OPTO I/O
• EXTERNAL EXPANSIONS
• PC/104 PROTO. SERIES

1-877-SCIDYNE
info@scidyne.com
www.scidyne.com

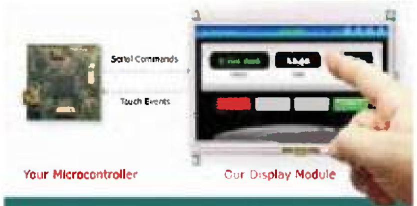
BPS BusBoard Prototype Systems



Prototyping PC Boards
Many Patterns

See our site:
www.BusBoard.us

Available From
JAMECO ELECTRONICS
MOUSER ELECTRONICS
amazon.com amazon.co.uk



Add A Touch Screen

- Get a Prototype Up in Days
- Move Smoothly to Production
- Development Kits Start at $349
- 100% Satisfaction Guarantee

Learn more at
www.reachtech.com
or contact us at
510-770-1417 or
sales@reachtech.com.

REACH TECHNOLOGY INC.

Development Kit

I²C™/SMBus

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

MCC
Micro Computer Control

www.mcc-us.com

Techsol Engineering

Commercial
Industrial
Medical
android
DEVICES

www.techsoleng.com

GHz Bandwidth Sockets
for DSP's in BGA

Industry's Smallest Footprint

- Pitch - 0.3mm to 1.27mm - BGA, QFN, MLF
- Bandwidth to 40+ GHz
- Six different lid options
- Optional 500,000 insertions
- Heatsinking to 100 watts

RoHS

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

ALL ELECTRONICS CORPORATION

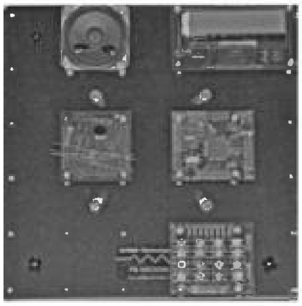
Electronic and Electro-mechanical Devices, Parts and Supplies. Wall Transformers, Alarms, Fuses, Relays, Opto Electronics, Knobs, Video Accessories, Sirens, Solder Accessories, Motors, Heat Sinks, Terminal Strips, L.E.D.S., Displays, Fans, Solar Cells, Buzzers, Batteries, Magnets, Cameras, Panel Meters,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

www.allelectronics.com

Free 96 page catalog

1-800-826-5432

MICROCONTROLLER KITS



LEARN
CREATE
EXPLORE

BASIC ON BOARD

ATRIA *Technologies Inc.*

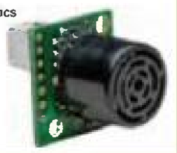
www.AtriaTechnologies.com

Ultrasonic Distance Sensing Made EZ

HRUSB-MaxSonar®-EZ™
- Multi-Sensor operation
- USB interface
- Easy integration
- 1 mm resolution
- Calibrated beam pattern
- Starting at $49.95

HRXL-MaxSonar®-WR™
- Incredible noise tolerance
- IP67 rated
- 1 mm resolution
- Multi-Sensor operation
- Calibrated beam pattern
- Starting at $109.95

I2CXL-MaxSonar®-EZ™
- Great for UAV's and robotics
- Incredible noise immunity
- I2C interface
- 1cm resolution
- Automatic calibration
- Starting at $39.95



www.maxbotix.com

Mini PC + Arduino: pcDuino

pcDuino is a mini PC platform that runs PC like OS (e.g. Ubuntu and Android ICS). It outputs screen to HDMI. It has headers interface compatible with Arduino.

Why Raspberry Pi? Choose pcDuino!

On sale now for $59 only!

- CPU: 1GHz ARM Cortex A8
- GPU: OpenGL ES2.0, OpenVG 1.1 Mali 400 core
- DRAM: 1GB
- Storage: 2GB Flash, SD card slot for up to 32GB
- Video Output: HDMI
- OS: Linux + Android
- Extension Interface: 2.54 mm Headers compatible with Arduino
- Network interface: RJ45 and USB WiFi Dongle

www.pcduino.com

microEngineering Labs, Inc.

www.melabs.com 888-316-1753

Programmers for Microchip PIC® Microcontrollers

Starting at $79.95

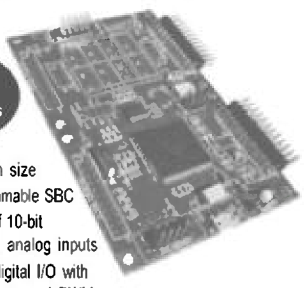
PC-Tethered USB Model (shown):
- Standalone software
- Command-line operation
- Hide GUI for automated use
- Override configuration with drop-downs

Stand-Alone Field Programmer:
- Power from target device or adapter
- Program file stored on SD-CARD
- Programming options stored in file
- Single-button operation

Program in-circuit or use adapters for unmounted chips.
Zero-Insertion-Force Adapters available for DIP, SOIC, SSOP, TQFP, and more.

PIC is a registered trademark of Microchip Technology Inc. in the USA and other countries.

Fast, Powerful, Low Cost Single Board Computer



Only $84/100s

- 2.5"x4" in size
- C-programmable SBC
- 16 lines of 10-bit resolution analog inputs
- 24 lines digital I/O with counter/timer and PWM
- 1 I²C communications port; I²C signals SDA and SCL
- 2 RS232 serial communications ports, 2 SPI ports
- Full featured development environment with device drivers for all I/O
- Prototyping area for custom circuitry

Mosaic Industries Inc.
tel: 510-790-1255 fax: 510-790-0925
www.mosaic-industries.com

B-Engine (BE)™

$69 OEM
$99 Qty 1

- *OEM CPU board with high speed ±10V bipolar 16-bit ADC (AD7606), DACs*
- *100M Base-T Ethernet, CompactFlash*
- *3.6" x 2.3", C/C++ programmable.*

100+ Low Cost Controllers with ADC, DAC, UARTs, 300 I/Os, solenoid, relays, CompactFlash, LCD, Ethernet, USB, motion control. Custom board design. Save time and money.

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

solve it!

C puzzler #3

```
void foo(void) {
    float f = 3.1415926;
    if (f + 0.1 != 3.2415926)
        fail();
    LCD_printf("%f", f);
    ...
```

Oh no! The comparison failed, and the LCD shows 3.141593! ...What happened??

To find out, AND get **$50 off** ImageCraft's professional and easy-to-use **AVR** and **Cortex-M C compilers** (regularly: $249) go to:

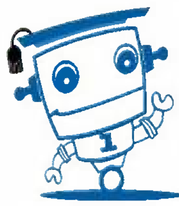
http://imagecraft.com/cci.html

Loadstar SENSORS

Low cost, digital load cells for force or weight sensing applications

- Serial TTL/USB/Analog output
- Compact miniature packaging
- Convenient mounting features

www.loadstarsensors.com
510-274-1872
sales@loadstarsensors.com



LOVE YOUR ROBOT!

We develop educational hardware and software that provides hands-on-fun for all ages

www.OneRobot.org

CONNECT WITH Circuit Cellar

For people who are passionate about hardware and software design, embedded development, and computer applications. Awesome projects, tutorials, industry news, design challenges, and more!

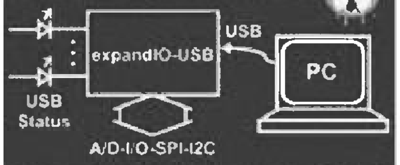
@CircuitCellar

www.facebook.com/circuitcellar

circuitcellar.com

CONNECT WITH US!

Revolutionary new expandIO-USB chip



- Ideal for adding USB to sensors & peripherals
- No drivers needed for Windows, Mac, Linux
- No microcontroller programming required
- Also check out our USB-232 USB to UART

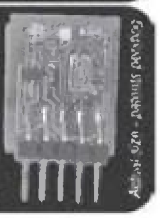
www.hexwax.com - Buy from Mouser & Farnell

Amazing PIC programmer

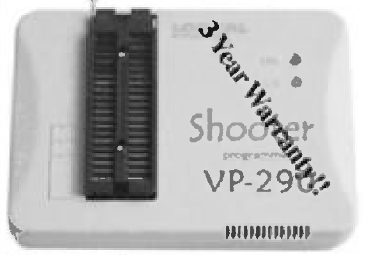
Most devices supported
ICSP, SQTP, & copy limits

$32 at Digikey & Mouser

www.flexipanel.com



LOGICAL DEVICES, INC.



Universal USB Programmer

Starting at **$195.00!!**

- UV Eraser $69.00
- PLD design Software $49.00
- Stand Alone Copier $175.00

Tel:303-923-8080
support@logicaldevices.com

www.logicaldevices.com

LISTEN TO YOUR MACHINES

Ethernet PLCs for OEMs

FMD88-10 and FMD1616-10

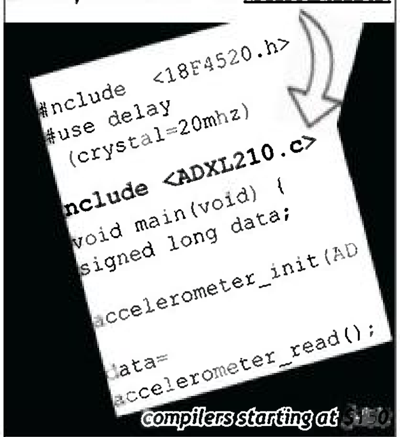
Integrated Features :
- ETHERNET / Modbus TCP/IP
- 16 or 32 digital I/Os
- 10 analog I/Os
- RS232 and RS485
- LCD Display Port
- I/O Expansion Port
- Ladder + BASIC Programming

$229 and $295
before OEM Qty Discount

tel : 1 877 TRI-PLCS
web : www.tri-plc.com/cci.htm

TRIANGLE RESEARCH INTERNATIONAL

PIC® C Compiler *to Accelerate your development with 165 device drivers*



```
#nclude <18F4520.h>
#use delay
 (crystal=20mhz)
nclude <ADXL210.c>
void main(void) {
signed long data;
accelerometer_init(AD
data=
accelerometer_read();
```

compilers starting at $150

www.ccsinfo.com/CC0413
sales@ccsinfo.com
262-522-6500 x 35

"Your reliable outsourcing partner"

LOW cost
with **HIGH quality**
PCB, PCBA and **More**

INSTANT QUOTE AT:
www.myropcb.com

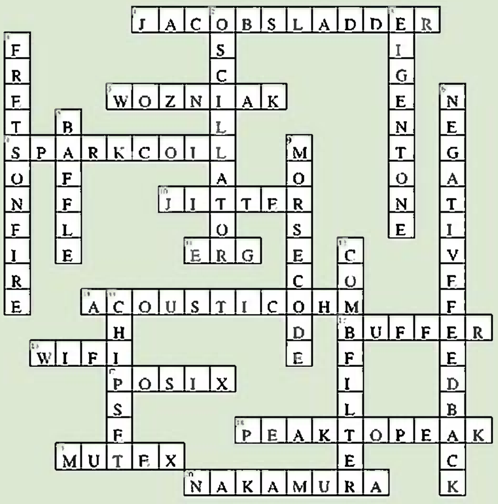
OR CALL:
1-888-PCB-MYRO

MyRO

CROSSWORD ANSWERS from Issue 272

Across
1. JACOBSLADDER—Climbing arc [two words]
5. WOZNIAK—Apple I
8. SPARKCOIL—Uses a low-voltage DC supply to create high-voltage pulses
10. JITTER—Creates an imperfect timing signal
11. ERG—Energy measurement
13. ACOUSTICOHM—Equivalent to µbar s/cm³ [two words]
15. BUFFER—Provides electrical isolation
16. WIFI—Provides movement to smartphones, PCs, and tablets
17. POSIX—An IEEE operating system compatibility standard
18. PEAKTOPEAK—Alterations between high and low values
19. MUTEX—Capable of ensuring atomic access to any shared resource
20. NAKAMURA—University of California, Santa Barbara professor credited with inventing the blue LED

Down
2. OSCILLATOR—American physicist George W. Pierce (1872–1956); piezoelectric
3. EIGENTONE—A pitch capable of resonance
4. FRETSONFIRE—Open-source gameplay for music lovers [three words]
6. NEGATIVEFEEDBACK—Type of amplifier invented in 1927 by Harold Black [two words]
7. BAFFLE—Sound wave obstruction
9. MORSECODE—A pre-texting means of communication [two words]
12. COMBFILTER—Capable of causing delay [two words]
14. CHIPSET—Intel created the first family of these



Can anyone deny that we're on the verge of some major breakthroughs in the fields of microcomputing, wireless communication, and robot design? Tech the Future is a recurring section devoted to the ideas and stories of innovators who are developing the groundbreaking technologies of tomorrow.

The Future of Open-Source Hardware for Medical Devices

by Fergus Dixon

Fergus Dixon holds a BE from Sydney University. After working for 10 years in various fields such as packaging, mining, and medical and control systems, he started Stirling Rock International (SRI), which provides custom electronic designs. After 14 years (and two Australian design awards), Fergus now runs several businesses, including Electronic System Design and Shields for Arduino. He recently designed the popular program "Simulator for Arduino." Fergus's article "Smart Switch Management: Construct an MCU-Based, 'Net-Enabled Controller" appeared in *Circuit Cellar* 263 (June 2012). *Circuit Cellar* is planning to publish his article about an embedded DNA sequencer next month.

Medical technology is changing at a rapid pace, but regulatory compliance is also becoming increasingly harder. Regulatory compliance can act as a barrier to innovation, but it is a necessary check to ensure quality medical care. For small companies, aligning innovation with regulatory compliance can only help.

When designing any new product, the FDA-recommended process is a great reference. First, the design input requirements must be written down. After the device has been designed and prototyped, verification and validation (V&V) will ensure that the device meets the design input. The device is then documented, creating the design output or device master record (DMR). Each device made is checked against the DMR and documented in the device history record (DHR). So all the details on how to make the device are contained in the DMR, and the results and traceability are recorded in the DHR.

My company recently asked an overseas company to design and manufacture an existing product. After many e-mails, the overseas company managed to build a working unit and immediately requested an order for 1,000. Before ordering even one unit, there was the matter of V&V. So what is V&V? Verification is the act of ensuring that the circuit acts as it should, as the circuit designer intended. This involves testing to a predetermined criteria, where the pass/fail is clearly defined. Testing happens by varying the inputs and checking the outputs to test the device as close to 100% as reasonably possible. When the inputs fall outside a normal range (e.g., a 10-VDC instead of 12-VDC battery voltage), the device must still work or it must provide a message showing why the device will not work (e.g., low battery light). Validation is the act of ensuring the circuit works as the customer or patient requires. This involves field testing, feedback, and rework—lots of it.

Working for medical device companies can be very rewarding. Smaller companies tend to work at the cutting edge. Larger companies are more secure and have stable products, but they can be less agile. With one company, we had a device that used smart batteries. During testing, we discovered that the batteries would not charge below 15ºC. After many meetings and e-mails to the manufacturers, the problem went to management, who decided to change the manual to say: "Do not charge below 15ºC." Smaller dynamic companies can attract the best scientists, which is great until a connector fails and there is a roomful of highly intelligent people with no soldering iron experience. Every technology company can benefit from having at least one experienced technician or engineer. A few hours spent playing with an Arduino is a great way to get this experience.

What about open-source hardware (OSHW) for medical devices? For home hobbyists and students, OSHW is great. There is free access to working circuits, programs, and sketches. C compilers, which once cost several thousand dollars, are mostly free. For the manufacturers, the benefits are plenty of feedback, which can be used to improve products. There is one roadblock, and that involves the loss of intellectual property (IP), which means anyone can copy the hardware. Creative Commons has addressed this with an agreement that any copies must reference the original work. Closed-source hardware can also be good and present fewer issues with losing IP. Apple is a great example. Rather than use feedback to improve products, it makes smart decisions about future products. The iOS vs. Android battle can be viewed as a closed-source vs. open-source struggle that still hasn't produced a winner. Medical devices and OSHW will have to meet up sometime.

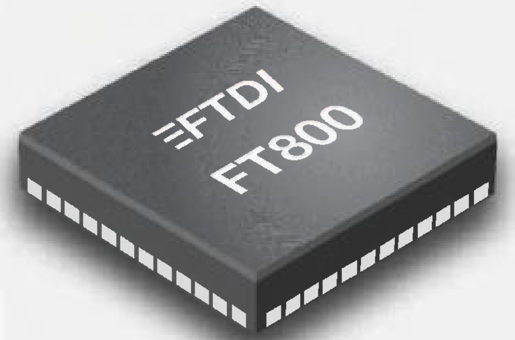
What about the future of medical devices? Well, the best is yet to come with brighter organic light-emitting diode (OLED) displays, a multitude of wireless connectivity options (all using the serial interface), and 32-bit ARM cores. DNA is gradually being unlocked with even "junk DNA" becoming meaningful. The latest hot topics of 3-D printing and unmanned aerial vehicles (UAVs) have direct medical applications with 3-D printed prosthetic ears and medical nanorobotics ready to benefit from UAV technology. Using a new sensor (e.g., a gyroscope) now means visiting an online seller such as Pololu, which offers ready-built development kits at reasonable prices. A recent design was a manually assisted CPR device project, which was abandoned due to lack of funding. How great would it be to have a device that could not only improve the current 10% survival rate with CPR (5% without CPR) but also could measure a patient's health to determine whether CPR was helping and, even more importantly, when to stop administering it? Now that would be a good OSHW project.

FTDI Chip

FTDI ENTERS THE **DISPLAY** MARKET

WITH

EVE EMBEDDED VIDEO ENGINE



3 in 1 Functionality – Display, Audio, and Touch

Complex Graphics Made Easy

Object Oriented Design Methodology

Low Cost BOM enables TFT Touch Display for New Markets

Eliminates Expensive Frame Buffer Memory

Allows Low Cost uC

Unparalleled Value in Embedded Display Technology

DISPLAY AUDIO TOUCH

FOLLOW US
@FTDICHIP

VISIT US
FTDICHIP.COM

WATCH OUR DEMOS ONLINE

Power of the Quoting Revolution in the palm of your hand!

At a click of a button we will find the best price in the market compiled in one easy to read spreadsheet ...just like your BOM.

Join the Revolution Happening Now @

www.PCBnet.com

847-806-0003   sales@PCBnet.com

ITAR, I SO 9001:2008, UL Approved