



circuit cellar

TAKE CONTROL

WIRELESS HOME AUTOMATION

A look at the evolution of wireless home automation technology and exciting new options for innovation

```
100 Module Attributes
101 'Activity module
102 Sub Process_Globals
103     Public serial1 As Serial
104 End Sub
105 Sub Globals
106 End Sub
107 Sub Activity_Create(FirstTime As Boolean)
108     If FirstTime Then
109         serial1.Initialize("serial1")
110     End If
111     Activity.LoadLayout("1")
112 End Sub
113 Sub Activity_Resume
114 End Sub
115 Sub Activity_Pause (UserClosed As Boolean)
116     If UserClosed = True Then
117         serial1.Disconnect
118     End If
119 End Sub
120 Sub btnConnect_Click
121     Dim PairedDevices As Map
122     PairedDevices = serial1.GetPairedDevices
123     Dim l As List
124     l.Initialize
125     For i = 0 To PairedDevices.Size - 1
126         l.Add(PairedDevices.GetKeyAt(i))
127     Next
128     Dim res As Int
129     res = InputList(l, "Choose device", -1) 'show list with paired devices
130     If res <> DialogResponse.CANCEL Then
131         serial1.Connect(PairedDevices.Get(l.Get(res))) 'connect to selected device and connect
132     End If
133 End Sub
134 Sub Serial1_Connected (Success As Boolean)
135     ProgressDialogHide
136     Log("connected: " & Success)
137     If Success = False Then
138         Log(LastException.Message)
139         ToastMessageShow("Error connecting: " & LastException.Message, True)
140     Else
141         StartActivity(ChatActivity)
142     End If
143 End Sub
144 End Sub
```



- Editors' Picks: Signal Processing Solutions ■ DIY Footlight Project |
- Room Acoustics Analysis | Battery Operation ■ Transformer Design |
- FPGA Design in Python | Connect Wirelessly with a Microcontroller |
- Tips for Selecting an Operational Amplifier | X-10 and Beyond
- The Future of Commodity Hardware Security





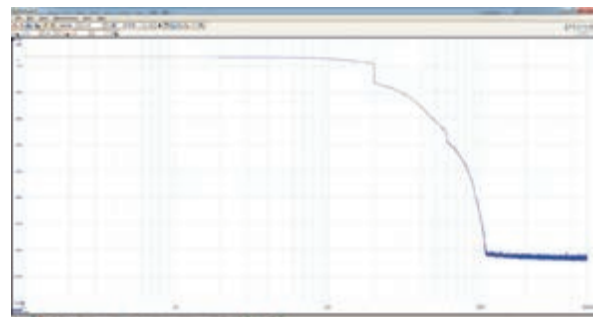
PicoScope® 4262

HIGH-RESOLUTION OSCILLOSCOPE

A Digital Oscilloscope for the Analog World

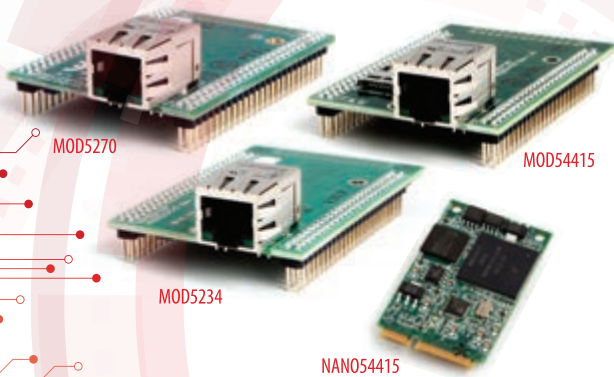
16 bit

- 102 dB SFDR • Log x Log FFT View
- Low noise • Two channels
- 16 MS buffer • 16-bit resolution
- 10 MS/s sampling • 5 MHz bandwidth
- Advanced digital triggers
- Low-distortion signal generator
- Arbitrary waveform generator
- USB powered
- SDK including LabVIEW and MATLAB
- Mac, Linux and Windows



For more information call 1-800-591-2796 or visit:
www.picotech.com/pco542

Ethernet Core Modules with High-Performance Connectivity Options



- **MOD5270**
147.5 MHz processor with 512KB Flash & 8MB RAM · 47 GPIO · 3 UARTs · I²C · SPI
- **MOD5234**
147.5 MHz processor with 2MB flash & 8MB RAM · 49 GPIO · 3 UARTs · I²C · SPI · CAN · eTPU (for I/O handling, serial communications, motor/timing/engine control applications)
- **MOD54415**
250 MHz processor with 32MB flash & 64MB RAM · 42 GPIO · 8 UARTs · 5 I²C · 3 SPI · 2 CAN · SSI · 8 ADC · 2 DAC · 8 PWM · 1-Wire[®] interface
- **NANO54415**
250 MHz processor with 8MB flash & 64MB RAM · 30 GPIO · 8 UARTs · 4 I²C · 3 SPI · 2 CAN · SSI · 6 ADC · 2 DAC · 8 PWM · 1-Wire[®] interface

Add Ethernet connectivity to an existing product, or use it as your product's core processor



The goal: Control, configure, or monitor a device using Ethernet

The method: Create and deploy applications from your Mac or Windows PC. Get hands-on familiarity with the NetBurner platform by studying, building, and modifying source code examples.

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

The NetBurner Ethernet Core Module is a device containing everything needed for design engineers to add network control and to monitor a company's communications assets. For a very low price point, this module solves the problem of network-enabling devices with 10/100 Ethernet, including those requiring digital, analog and serial control.

MOD5270-100IR.....\$69 (qty. 100)	NNDK-MOD5270LC-KIT\$99
MOD5234-100IR.....\$99 (qty. 100)	NNDK-MOD5234LC-KIT\$249
MOD54415-100IR.....\$89 (qty. 100)	NNDK-MOD54415LC-KIT\$129
NANO54415-200IR...\$69 (qty. 100)	NNDK-NANO54415-KIT.....\$99

NetBurner Development Kits are available to customize any aspect of operation including web pages, data filtering, or custom network applications. The kits include all the hardware and software you need to build your embedded application.

➤ **For additional information please visit**
<http://www.netburner.com/kits>

Issue 303 October 2015 | ISSN 1528-0608

CIRCUIT CELLAR® (ISSN 1528-0608) is published monthly by:

Circuit Cellar, Inc.
111 Founders Plaza, Suite 904
East Hartford, CT 06108

Periodical rates paid at East Hartford, CT, and additional offices.
One-year (12 issues) subscription rate US and possessions \$50, Canada \$65, Foreign/ ROW \$75. All subscription orders payable in US funds only via Visa, MasterCard, international postal money order, or check drawn on US bank.

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

ADVERTISING

Strategic Media Marketing, Inc.
2 Main Street, Gloucester, MA 01930 USA

Phone: 978.281.7708

Fax: 978.281.7706

E-mail: circuitcellar@smmarketing.us
Advertising rates and terms available on request.

New Products:

New Products, Circuit Cellar, 111 Founders Plaza, Suite 904
East Hartford, CT 06108, E-mail: newproducts@circuitcellar.com

HEAD OFFICE

Circuit Cellar, Inc. 111 Founders Plaza, Suite 904
East Hartford, CT 06108
Phone: 860.289.0800

COPYRIGHT NOTICE

Entire contents copyright © 2015 by Circuit Cellar, Inc. All rights reserved. Circuit Cellar is a registered trademark of Circuit Cellar, Inc. Reproduction of this publication in whole or in part without written consent from Circuit Cellar, Inc. is prohibited.

DISCLAIMER

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

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

© Circuit Cellar 2015 Printed in the United States

IN MEMORIAM—HUGO VAN HAECKE (1951–2015)

Hugo Van haecke, president and publisher of Circuit Cellar, Inc. and Segment LLC, passed away on August 19, 2015, in Denver, CO. A publishing industry veteran and an exceptional manager, Hugo was instrumental in the transition of our business from its early foundations into a future-ready organization—managing acquisitions, mergers, and restructuring the companies to ensure the continuity and evolution of the titles (*audioXpress*, *Voice Coil*, *Loudspeaker Industry Sourcebook*, and *Circuit Cellar*) as well as the company's book publishing business.

Born May 5, 1951, in Antwerp, Belgium, Hugo was the second of four children born to Henri and Alice Van haecke-Verrycken. He was an eager learner from the start, and through the teachings of his older brother, Alex, knew how to read, write and do basic math before he entered the first grade. During his formative educational years, Hugo was a passionate student with a desire to learn, but struggled with the restrictive nature of the educational environment.



In 1973, Frank de Winter gave Hugo a job at Old Charley, a wine wholesale company, to assist in bookkeeping, accounts, and customer relations. Frank taught Hugo everything there was to know about accounting, bookkeeping, and finances. Hugo excelled in this environment and the experience reinforced his lifelong belief that wisdom and success are fostered through real word experiences.

Hugo moved to the US with his family in 1999, while working for Wolters Kluwer, a publishing group based in the Netherlands. During his career, he managed businesses and

companies around the world. In 2005, Hugo started his own venture, working with several companies as a consultant, a financial advisor, and president.

Hugo had semi-retired to devote more time to his family and especially his granddaughter, while continuing to be at the helm of our business. A lover of family, life, and friends, Hugo was a great manager, a proud Million Miler traveler, and someone who inspired us all and will be greatly missed.

Hugo Van haecke is survived by his wife, Erna Van Meerbergen; his children Margo Valaika and husband Chris Valaika; Thomas Van haecke and wife, Emilie Van haecke; his granddaughter, Alyse Valaika; his brother, Alex Van haecke; his sister Lieve Van haecke; and numerous other relatives.

We dedicate this issue to Hugo.

The Circuit Cellar Staff

THE TEAM

EDITOR-IN-CHIEF

C. J. Abate

ART DIRECTOR

KC Prescott

ADVERTISING COORDINATOR

Kim Hopkins

PUBLISHER

Dan Rodrigues

COLUMNISTS

Jeff Bachiochi (From the Bench), Ayse K. Coskun

(Green Computing), Bob

Japenga (Embedded in Thin Slices), Robert Lacoste (The Darker Side), Ed Nisley (Above the Ground Plance), George Novacek (The Consummate Engineer), and Colin O'Flynn (Programmable Logic in Practice)

FOUNDER

Steve Ciarcia

PROJECT EDITORS

Chris Coulston, Ken Davidson, and David Tweed

OFFICE ASSISTANT

Debbie Lavoie

OUR NETWORK



SUPPORTING COMPANIES

13th International SoC Conference	53	Imagineering, Inc.	C4
Accutrace	7	Ironwood Electronics	79
AES - Audio Engineering Society	63	Jeffery Kerr, LLC	79
All Electronics Corp.	79	Lemos International	11
Custom Computer Services	79	MaxBotix, Inc.	79
Elprotronic, Inc.	11	Measurement Computing Corp.	55
EMAC, Inc.	9	MyRO Electronic Control Devices, Inc.	79
Front Panel Express	9	NetBurner, Inc.	1, 59
General Circuits Co. Ltd.	45	Pico Technology	C2
HuMANDATA, Ltd.	15	Saelig Co., Inc.	33
IAR Systems	21	Technologic Systems	31

NOT A SUPPORTING COMPANY YET?

Contact Peter Wostrel (circuitcellar@smmarketing.us, Phone 978.281.7708, Fax 978.281.7706)
to reserve your own space for the next edition of our members' magazine.

CONTENTS

OCTOBER 2015 • ISSUE 303

SIGNAL PROCESSING**INDUSTRY & ENTERPRISE**

06 : PRODUCT NEWS

09 : CLIENT PROFILE
Percepio (Västerås, Sweden)**CC COMMUNITY**

10 : EDITORS' PICKS

Signal ProcessingSeveral of the *Circuit Cellar* team's favorite articles on signal processing-related topics**FEATURES**12 : **The Footlight Project (Part 1)**

Circuit Board Design

By Tom Struzik

Details of the circuit board development process for a footlight project

18 : **Sound Ecology and Acoustic Health (Part 4)**

Room Acoustics Analysis

By Adrien Gaspard & Mike Smith

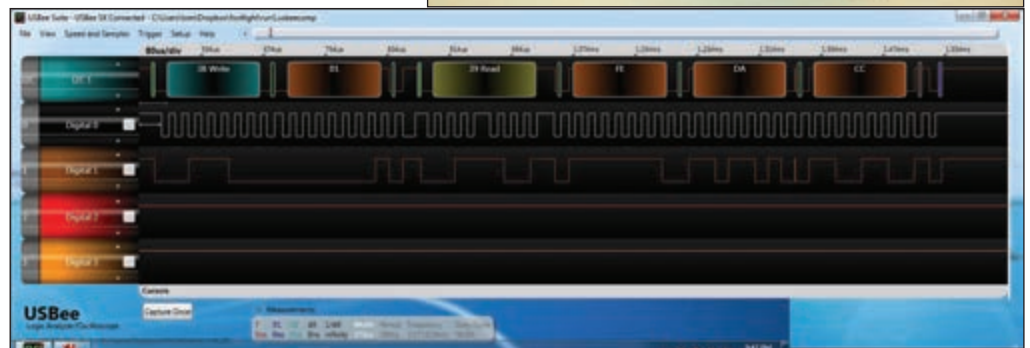
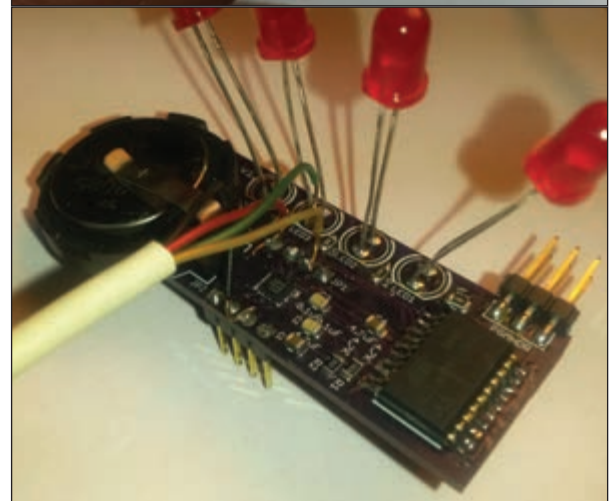
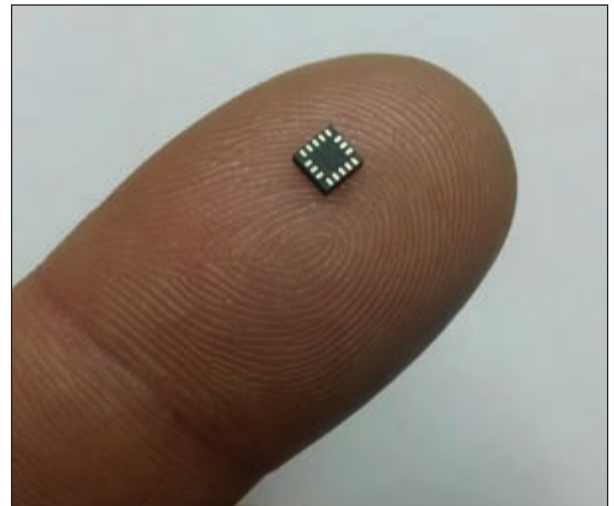
How to code a room acoustics analysis

28 : **Running on Battery (Part 2)**

Battery Operation

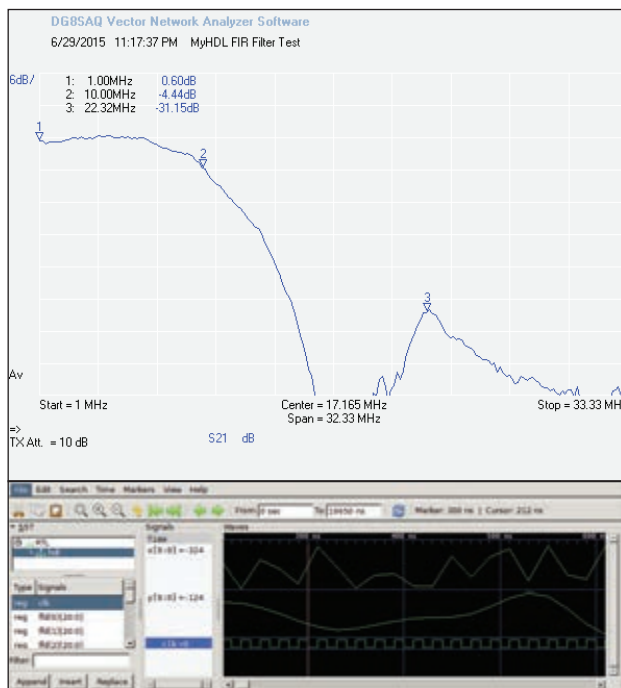
By Stuart Ball

An examination of combined AC/battery operation, single-cell operation, rechargeable batteries, and more

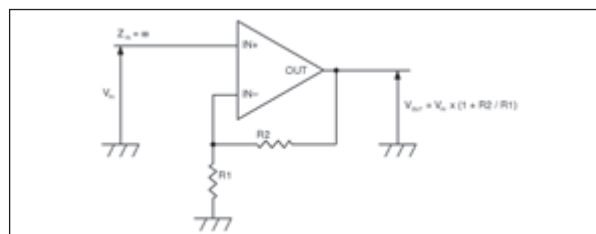


FOOTLIGHT PROJECT: CIRCUIT DESIGN

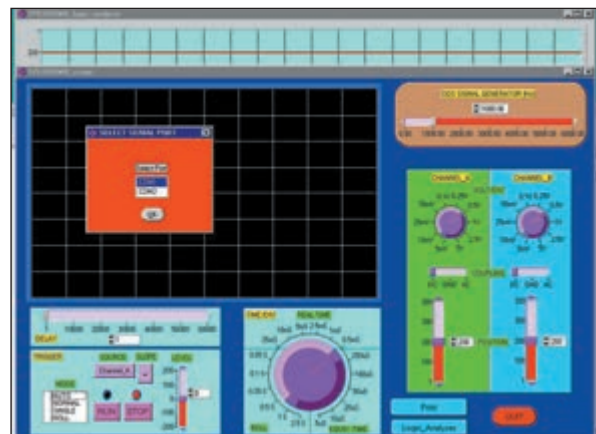
CONTENTS



WORKING WITH MyHDL



TIPS FOR SELECTING OP-AMPS



DIY 3-IN-1 MEASUREMENT SYSTEM

CC REBOOT

34 : Build a Three-in-One Measurement System

By Salvador Perdomo

How to build a measurement system comprising a signal generator, logic analyzer, and digital oscilloscope

COLUMNS

42 : THE CONSUMMATE ENGINEER

Transformers 101 (Part 2)

Transformer Design

By George Novacek

A review of essential transformer design principles

46 : PROGRAMMABLE LOGIC IN PRACTICE

Rapid FPGA Design in Python Using MyHDL

By Colin O'Flynn

Use MyHDL to leverage the power of Python for designing, simulating, and verifying FPGA designs

56 : EMBEDDED IN THIN SLICES

The Internet of Things (Part 3)

Connect Wirelessly with a Microcontroller

By Bob Japenga

Tips for connecting your devices wirelessly to the Internet

60 : THE DARKER SIDE

How to Select an Operational Amplifier

By Robert Lacoste

The key characteristics of op-amps and suggestions for choosing the right one

68 : FROM THE BENCH

Wireless Home Automation (Part 1)

X-10 and Beyond

By Jeff Bachiochi

The evolution of wireless technology and a look at exciting new home automation options

TESTS & CHALLENGES

77 : TEST YOUR EQ

78 : CROSSWORD

TECH THE FUTURE

80 : The Future of Commodity Hardware Security and Your Data

By Alexandra Mellen

Thoughts on the future of hardware design and the problem of underestimated security vulnerabilities



THE FURUTE OF HARDWARE SECURITY

PRODUCT NEWS

NEW 32-BIT MCU SERIES FOR EMBEDDED CONTROL & TOUCH

Microchip Technology recently announced a new series within its PIC32MX1/2 32-bit microcontroller family that features a 256-KB flash configuration and 16-KB of RAM. The microcontrollers provide flexibility to low-cost applications that need complex algorithms and application code. More specifically, they are intended to help designers looking to develop products with capacitive touch screens or touch buttons, as well as USB device/host/OTG connectivity.

The PIC32MX1/2 MCU series provides up to 50 MHz/83 DMIPS performance for executing advanced control applications and mTouch capacitive touch sensing. In addition, it has an enhanced 8-bit Parallel Master Port (PMP) for graphics or external memory, a 10-bit, 1-Msps, 13-channel ADC, support for SPI and I2S serial communications interfaces, and USB device/host/On-the-Go (OTG) functionality.

Microchip's MPLAB Harmony software development framework further simplifies designs by integrating the license, resale, and support of Microchip and third-party middleware, drivers, libraries and Real-Time Operating Systems (RTOS). Specifically, Microchip's readily available software packages—including USB stacks and Graphics and Touch libraries—can greatly reduce the

development time of applications such as consumer, industrial and general-purpose embedded control.

These latest PIC32MX1/2 MCUs are available now in 28-pin QFN, SPDIP, and SSOP packages and 44-pin QFN, TQFP and VTLA packages. Pricing starts at \$1.91 each, in 10,000-unit quantities.

Microchip Technology | www.microchip.com



ANALOG AMPLIFIER PROVIDES PRECISE CURRENT SHUNT MEASUREMENT

Silicon Labs has introduced a new isolated current sense amplifier with industry-leading signal bandwidth (up to 750 kHz) that ensures rapid, precise DC current measurement and accurate representation of the primary signal and harmonics. The Si8920 isolated amplifier provides an ideal current shunt measurement solution for power control systems operating in harsh environments (e.g., hybrid vehicles, industrial motor drives, and high-voltage power converters).

The Si8920 isolated amplifier uses Silicon Labs' proven, CMOS-based isolation technology, supports up to 5 kV withstand and 1200 V working voltage, and offers a wide operating temperature range, noise immunity, and long lifetimes. With exceptionally low 1- μ V/ $^{\circ}$ C offset drift, you gain stable performance over diverse operating conditions.

The Si8920 isolated amplifier is available in standard SOIC and DIP packages. Pricing in 10,000-unit quantities starts at \$2.39. The

\$29 Si8920ISO-KIT evaluation kit enables you to connect quickly to a shunt resistor to evaluate Si8920 analog isolation functionality including low-voltage differential input, response times, offset and gain characteristics.

Silicon Labs | www.silabs.com



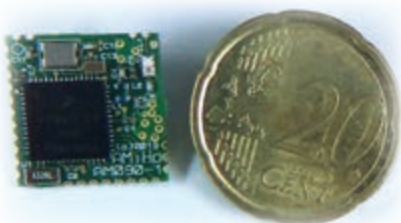
NEW ULTRA-COMPACT WIRELESS M-BUS MODULE

AMIHO Technology recently announced an ultra-compact and cost effective Wireless Meter-Bus module. The AM090 is intended primarily for connecting smart meters and Internet of Things (IoT) devices. At just 15 x 15 mm, the AM090 works well with small

sensors and other IoT end points.

Fully compliant with the European standards (EN13757), the AM090 features Freescale's Kinetis family of ARM cortex MCUs and operates at 868 MHz. The module includes a comprehensive and optimized software stack, which can be licensed as a stand-alone product for integration into other designs.

AMIHO Technology | www.amihotechnology.com



PRINTED CIRCUIT BOARDS

THINK YOU CAN FIND PCB PRICES THAT BEAT OURS?

WE DARE YOU.

If you do, than we
will match the price
AND give you \$100
towards your
next order!

THERE ARE NO GAMES INVOLVED IN OUR PRICING



Our Capabilities:

- From same day quick turn prototype to production in under 10 days
- Full CAD and CAM review plus design rule check on ALL Gerber files
- Materials: Fr4, Rigid, Flex, Metal Core (Aluminum), Polyimide, Rogers, Isola, etc.
- HDI Capabilities: Blind/Buried Microvias, 10+N+10, Via-in-Pad Technology, Sequential Lamination, Any Layer, etc.
- Our HDI Advantage: Direct Laser Drilling, Plasma De-Smear Technology, Laser Microvia, Conductive Plate Shut.

Take the Accutrace Challenge and see WHY OUR PRICING CANNOT BE BEATEN

Accutrace inc.

www.PCB4u.com sales@PCB4u.com

PRODUCT NEWS

NEW BATTERY PACK MONITOR PROTECTS MULTI-CELL LI-ION BATTERIES

Intersil Corp. recently announced the ISL94203 3-to-8 cell battery pack monitor that supports lithium-ion (Li-ion) and other batteries. The ISL94203 can monitor, protect, and cell balance rechargeable battery packs to maximize battery life and ensure safe charging and system operation. It works as a stand-alone battery management system for rechargeable Li-ion battery packs. The ISL94203's internal state machine has five preprogrammed stages that accurately control each cell of a battery pack to extend operating life. In addition, it integrates high-side charge/discharge FET drive circuitry.

Notable features and specifications:

- Eight cell voltage monitors support Li-ion CoO₂, Li-ion Mn₂O₄, Li-ion phosphate and other battery chemistries
- Operates as a standalone solution or with a microcontroller
- Integrated charge pump controls cutoff FETs used to charge/discharge battery pack
- Multiple cell voltage protection options up to 4.8 V
- Programmable detection/recovery times for over-voltage, under-voltage, over-current, and short circuit conditions
- Open wire detection
- EEPROM storage for device configuration
- Power saving algorithm activated when pack is not in use

The ISL94203 battery pack monitor is available now through

Intersil's worldwide network of authorized distributors. The ISL94203 comes in a 6 mm × 6 mm, 48-lead TQFN package, and is priced at \$2.19 in 1,000-piece quantities. The ISL94203EVKIT1Z evaluation kit (\$328) includes an evaluation board, interface board with USB-to-I2C interface, and software GUI that supports stand-alone operation or an external microcontroller.

Source: Intersil Corp. | www.intersil.com



SENSOR INTERFACE CONNECTS MULTIPLE SENSORS TO MCUs OR FPGAs

Exar Corp. has announced the XR10910, a new sensor interface analog front end (AFE) for the calibration of sensor outputs. The XR10910 features an onboard 16:1 differential multiplexer, offset correction DAC, programmable gain instrumentation amplifier, and voltage reference. In addition, it provides 14-bit signal path linearity and is designed to connect multiple bridge sensors to a microcontroller or FPGA with an embedded ADC. Operating from from 2.7- to 5-V supplies, the XR10910 has a wide digital supply range of 1.8 to 5 V. It typically consumes 457 μ A of supply current and offers a sleep mode for reducing the supply current to 45 μ A.

The XR10910 is available in a 6 mm × 6 mm QFN package. Pricing starts at \$8.10 each for 1,000-piece quantities.

Exar Corp. | www.exar.com



MINIATURE 9.7 × 7.5 MM OCXO

IQD's latest Oven-Controlled Crystal Oscillator (OCXO), the IQOV-71 series, is housed in four-pad plastic package with a fiber glass base. Despite its 9.7 × 7.5 mm size, it offers very low frequency stabilities down to ± 10 ppb over an operating temperature range of -20° to 70° C or ± 20 ppb over -40° to 85° C.

The available standard frequencies include 10 MHz, 12.8 MHz, 19.2 MHz, 20 MHz, 24.576 MHz, 25 MHz, 30.72 MHz, 38.88 MHz, 40 MHz, 49.152 MHz, and 50 MHz, which will satisfy most applications. Other frequencies in the range of 5 to 50 MHz can be developed for commercially viable quantities. Power consumption is typically less than 1 W during the warm up phase, which only takes approximately 3 minutes, and less than 0.4 W once the device

has reached steady state. Frequency aging is less than 2 ppb per day and a maximum of 3 ppm over a 10-year period.

IQD | www.iqdfrequencyproducts.com



CLIENT PROFILE

Percepio

Location: Västerås, Sweden

Web: www.percepio.com

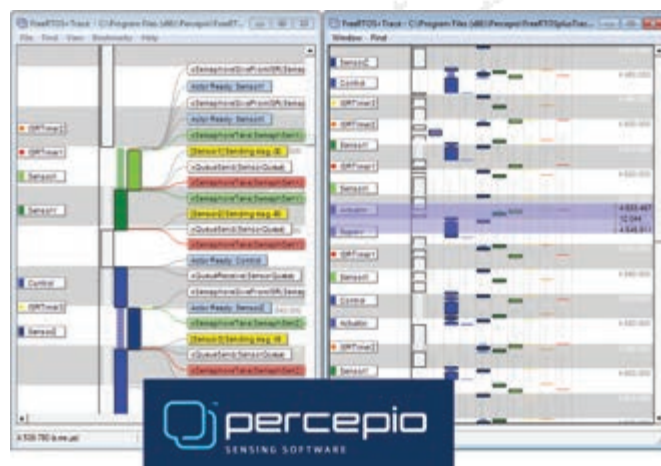
Contact: Mike Skrtic (mike.skrtic@percepio.com)

EMBEDDED PRODUCTS

Percepio Tracealyzer provides an unprecedented level of insight into the run-time world of RTOS or Linux-based software systems. Tracealyzer allows you to solve complex software problems in a fraction of the time otherwise needed, develop more robust designs to prevent future problems, and find new ways of improving your software performance. (More info: <http://percepio.com/tz/>)

WHY SHOULD CC READERS BE INTERESTED?

In order to solve a bug, you first have to see it. Percepio Tracealyzer is world class visualization software that allows you see what's going on inside your RTOS-based system. Tracealyzer visualizes the run-time behavior through more than 20 innovative views that complement the debugger perspective. The views are interconnected in intuitive ways, which makes them very powerful and easy to navigate. Tracealyzer is available for the following real-time operating systems: FreeRTOS, Micrium, embOS, Linux, VxWorks, SafeRTOS, and On Time RTOS-32



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.

INDUSTRY & ENTERPRISE

Industrial IoT

- Embedded Systems
- Engineering
- Prototyping
- Manufacturing
- Application Development

Qty 100 Starts at \$89

SoM-9G25

- Atmel ARM9
- AT91SAM9x25 400MHz
- 64MB DDR
- 32MB Serial Data Flash
- Up to 512 NAND (option)
- Ethernet
- A/D, SPI, I2C, I2S
- PWM, GPIO, CAN
- 6x Serial Ports, SDIO Port
- Wide Temp -40 to +85

ENAC OEM MADE IN USA

ENAC, inc.

EQUIPMENT MONITOR AND CONTROL

Our Products Make Your Products Better™

30
YEARS OF
EMBEDDED
SOLUTIONS

The Easiest Way to Design Custom Front Panels & Enclosures

Free Front Panel Designer

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

EDITORS' PICKS

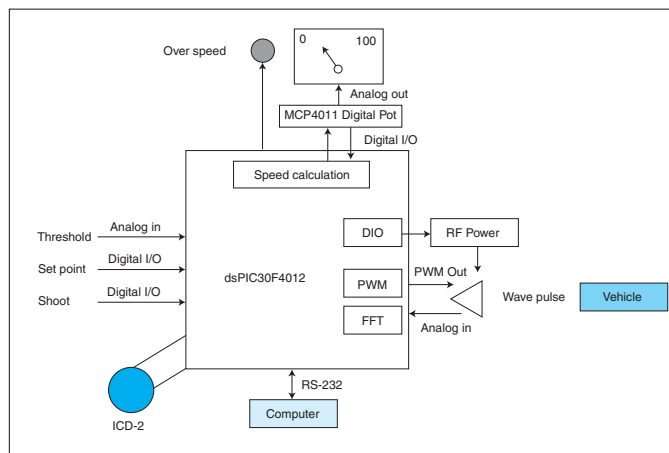
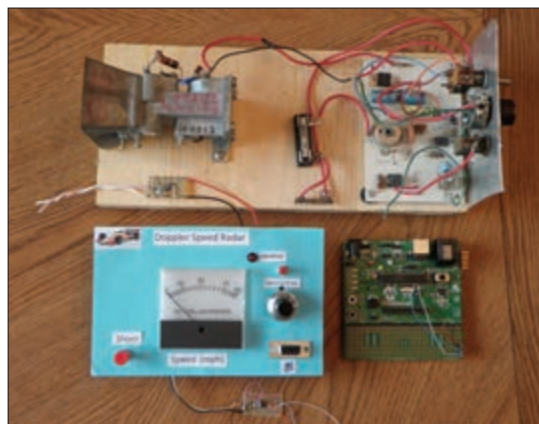
Signal Processing

Doppler Radar Design

By Steve Lubbers (*Circuit Cellar 243, 2010*)

A Doppler radar system can enable you to track speeding vehicles. Steve Lubbers's design generates a microwave energy burst with a 10-GHz transceiver. The microwave signal bounces off a moving vehicle and the frequency shift is measured to determine its speed.

Lubbers writes: "My Doppler radar required a microcontroller with DSP capabilities. The microcontroller features enabled the I/O and control features required by the project. The DSP capabilities were required to perform signal processing on the received radio signal. The dsPIC30F4012 CPU fulfilled all of the processor requirements. It features an ADC for RF input capture, PWM output for transmit signal modulation, and general-purpose I/O to control the remaining hardware features. Software development and debugging was eased by using the Microchip ICD2 interface. I chose the dsPIC's 28-pin DIP package so I could use a Microchip 28-pin starter board as the base of the digital hardware... The primary output of my Doppler radar is the target's speed displayed on an analog meter. An LED accompanies the meter to indicate if a preset threshold has been exceeded, and the target is 'speeding.' The analog display is driven by an MCP4011 64-position digital pot, which the dsPIC controls using two digital output lines."



These articles and others on topics relating to signal processing are available at www.cc-webshop.com.

Signal Generation Solution

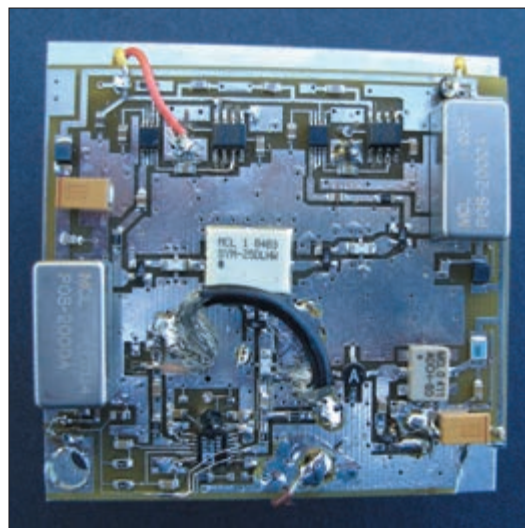
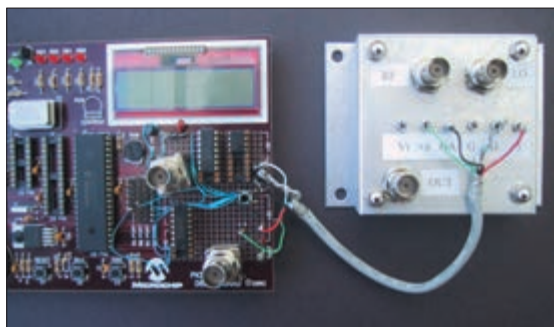
Build an Inexpensive RF Signal Generator

By Neal Martini (*Circuit Cellar 182, 2005*)

Tired of going to a local university lab to use a signal generator, Neal Martini designed his own. In this article he explains how he built the PIC16F877A-based controller and RF module.

Martini writes: "With many years of professional and hobby experience in lower frequency digital and analog systems, I decided to venture out and build my own RF signal generator... My goal for this project was to design a signal generator

that produces sine waves from 10 to 600 MHz at a constant output power level of 5 dBm. Let's take a look at how I did it. The assumed load is 50 Ω , which is typical for RF systems. Talking about signal levels in terms of decibels relative to 1 mW (dBm) is common when you're dealing with RF systems."



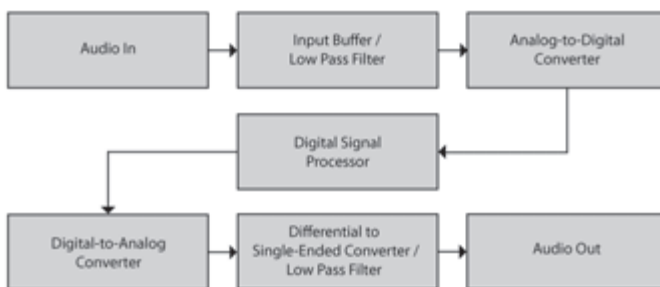
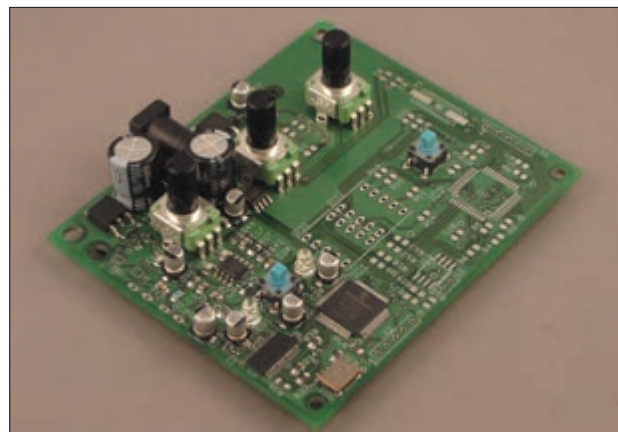
EDITORS' PICKS

Digital Stompboxing

An Easy-to-Use Digital Signal Processing Platform
 By Kit Church (Circuit Cellar 220, 2008)

Are you ready to venture into the world of digital signal processing effects? This article is the perfect introduction. Kit Church describes a high-quality DSP platform and presents sample code.

Church writes: "To make up for a lack of 'hackable' digital effects, I created a high-quality, easy-to-understand DSP platform. In this article, I'll describe the design and provide sample code that will enable you to venture into the world of DSP-based effects... To some of you, the thought of assembling a DSP-based effects platform may seem intimidating. But the hardware for this project is relatively simple. It includes an ultralow-distortion Texas Instruments OPA2134 dual op-amp, a Texas Instruments PCM3060 stereo audio codec (ADC and DAC combined into one chip), a Microchip Technology dsPIC33FJ64GP206 microcontroller, and a handful of passive components... To control the effects processing, I've attached three potentiometers and two momentary switches to the microcontroller as well as a few LED indicators. The dsPIC33FJ64GP206 has a maximum of 53 I/O pins, including 18 analog channels, so there's also plenty of room for expansion for your own effects and configurations."



COMMUNITY

These articles and others on topics relating to signal processing are available at www.cc-webshop.com.

FlashPro430
FlashPro-CC
FlashPro2000
GangPro430
MSP-GANG
GangPro-CC

NEW

FlashPro-CM
FlashPro-LM
FlashPro-STM

USB Flash and Gang Programmers for Texas Instruments' MCUs MSP430, Chipcon CCxx, C2000, Stellaris LMxx and ST-Microelectronics MCUs - STM32xx (ARM)

* can assign unique serial number
 * up to 64 USB-FPA programmers can be connected to one PC and program target devices simultaneously

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

Programmers: USB-FPA-1, USB-FPA-2, USB-FPA-3, USB-FPA-8

Connectors: JTAG / SBW or BSL

Chips: MSP430Fxx

Elprotronic Incorporated
www.elprotronic.com

RF Specialists

"Making your RF ideas into profitable products."

FCC Part 90 Compliant

USX2 - NBFM Multi-Channel UHF Transceiver with Programmable RF Power

MURS (Multi-Use Radio Service)

SHX1 - Long Range, High Power MURS Band Transceiver

ZigBee Pro

OEM Modules and USB ZigBee Sticks, Mesh Networks

Industrial Bluetooth

OEM, Modules, Wireless Device Servers, RS-232 Long range options, low cost

Ultra-Low Power Wi-Fi networking module and Eval board

The AMW006 'Numbat' module is an ultra-low power Wi-Fi networking module with full regulatory certification.

RF Design Services

Prepared to work with your in-house engineers, or support your RF project from initial design to implementation.

Industrial • Military • Space • Medical • Smart Grid Metering • SCADA • Lighting Control

LE MOS
INTERNATIONAL

Tel: 1.866.345.3667
 orders@lemosint.com
www.lemosint.com

The Footlight Project (Part 1)

Circuit Board Design

It's unrealistic to expect every project to be a simple success. Here's a project to build a "foot headlight" that does not come out as expected. This article details the circuit board development process.

By Tom Struzik (US)

PHOTO 1

This is the Freescale Semiconductor MMA8453QT accelerometer in a QFN-16 footprint. Certainly, it's at the limit of my simple hotplate reflow soldering capabilities!

Mistakes, failures, and unexpected outcomes can be the source of great innovation. As Henry Ford once said, "The only real mistake is the one from which we learn nothing."^[1] Most of the project articles I've seen have been about project successes. Rarely do I see an article about a project that failed. To the unassuming reader, it must appear that every project is a success. To help correct that misinterpretation, I wrote an article about a project that failed and what I learned during the process. Those learnings made the project a success despite all the failures along the way. From this perspective, no project is ever a true failure.

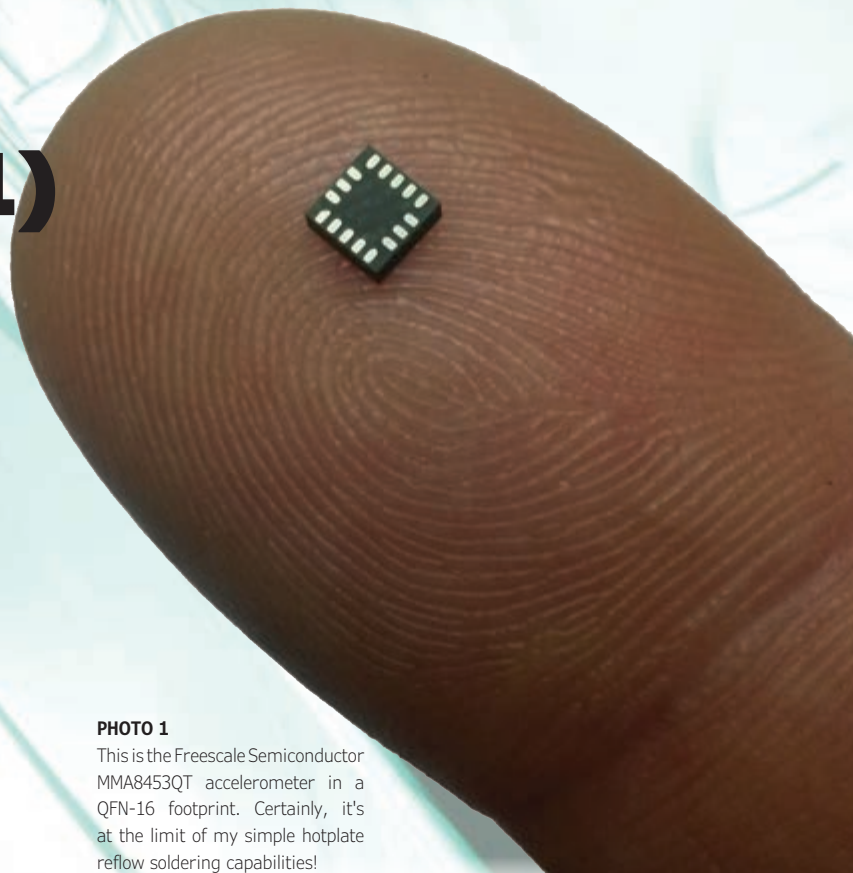
This project came about as a result of a bad fall I took on an early morning jog when I tripped over an unseen crack in the sidewalk. The idea came straight away: I needed headlights for my shoes! Not just a "dumb" LED that spent the majority of its time pointed the wrong direction, but rather a controlled array of LEDs that would keep the resulting light beam pointed at the ground in front of me. The result was a "foot headlight," or footlight.

FOOTLIGHT PROTOTYPE

From the start, I decided the footlight to use an XYZ accelerometer and an Atmel ATtiny AVR processor to turn on only the LED that was at the proper angle to illuminate the

ground in front of my feet. I had numerous options for accelerometers: one-, two-, or three-axis options, as well as analog or digital outputs, and various G range measurements. After comparing options, I decided on a three-axis accelerometer that gave direct digital measurements via an I²C interface with a range of approximately 3Gs. This, combined with an AVR processor with hardware support for implementing I²C, seemed like the quick-and-easy approach to meet the project's goals. Thus, the footlight comprises a Freescale Semiconductor MMA8453QT accelerometer and an Atmel ATtiny 281 running at 8 MHz. Yes, the MMA8453QT has a QFN-16 footprint, which might be pushing the limits of my hotplate reflow technique, but I knew I could make it work (see **Photo 1**).

Since I hadn't worked with an accelerometer before, I thought it best to put together a quick prototype to verify my ability to read the acceleration data via the I²C interface while managing basic LED selection. The goal for the prototype was to be able to keep illuminated the LED currently pointing up. The initial code was very simple: read the I²C data, and based only on the range of the acceleration data, select the most upward-facing LED. I designed a quick circuit board and sent it off to OSH Park for production. Soon thereafter I found that nothing is ever as easy it initially appears.



Once the prototype board arrived, it was off to the hotplate to reflow solder mount the components for the prototype. With its QFN-16 footprint, the accelerometer was a bit of a stretch using this method, but it visually appeared to have successfully mounted. I wrote and programmed the prototype software into the ATtiny, power was applied, and ... nothing, nada, zilch. I then pulled out my CWAV USBee-SX and started snooping the I²C signals. Again, nothing. So began the first of a long string of failures on this project. However, this also brought the first three learnings: using the USBee-SX to sniff and decode I²C signals; the criticality of bringing important signals, like the I²C bus, out to an easily accessible debug header; and finally the need to make the pads on tiny SMT parts longer than the manufacturer's recommended profile to facilitate hand-soldering when necessary. At this point, you can see the prototype board in **Photo 2** with added wires to enable access to the I²C bus.

While I had used a software I²C interface before, I had not used this specific hardware TWI/ I²C interface. I spent several hours poring over the Atmel documentation on the hardware serial interface trying to determine why the interface was not working. After multiple failures, I finally broke down and simply switched to my familiar software I²C library so I could at least start making progress again. Now I was able to use the USBee-SX to observe I²C query and response traffic, thus confirming that I finally had communication established between the ATtiny and the accelerometer. Certainly the software I²C library was a larger code base than the hardware I²C implementation, and it took more processing power—but, for the time being, that was acceptable. Yes, I still had to categorize the hardware I²C as a fail, but that could go on the bin list for learning later. At least I had confirmed that the reflow mounting of the accelerometer had worked.

CODING & CONFIGURATION

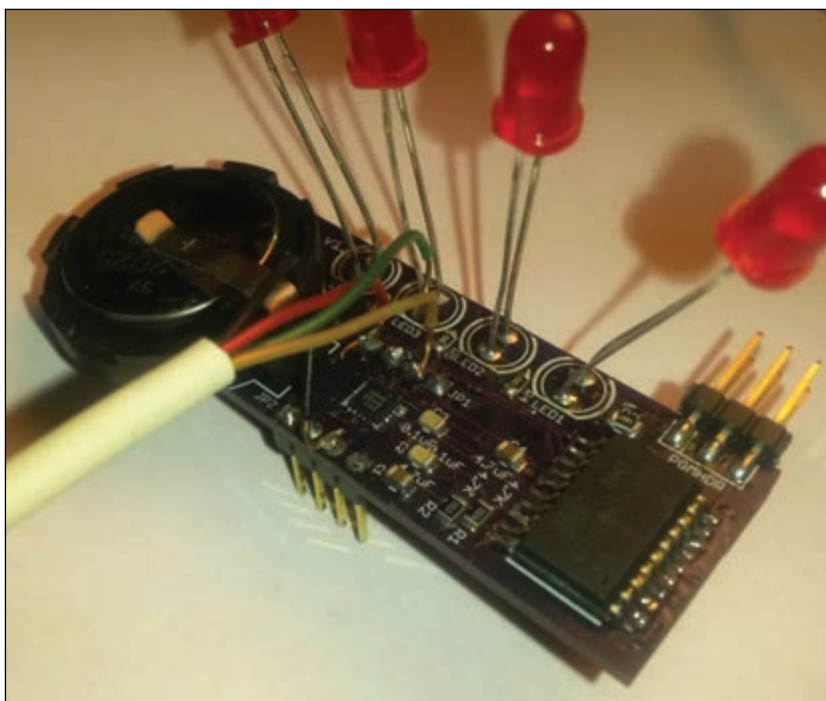
Next, I started the truly interesting coding, configuring the accelerometer and reading the acceleration data. Once the accelerometer was configured, the acceleration data could be accessed from three consecutive device registers. These were easily accessed via sequential I²C reads. The acceleration data was presented in Gs as a two's complement scaled number, so I had to do a quick refresher on interpreting binary data as two-complement integers. I was using the USBee-SX to track the acceleration data changes via decoded I²C signals, but that quickly became painful due to the amount of rapidly changing data (see **Photo 3**). I realized then that,

since I had an I²C bus available, I could also connect a small I²C based LCD. Now if I had just included that dang I²C debug header... For now I hand soldered some add-on wires to facilitate connecting in the I²C LCD—which, unfortunately, ran at 5 VDC while the rest of the protoboard's components required 3.3 VDC. Out of this, I learned that for the LCD I had, even though the board required 5 VDC for its supply, it continued to operate properly with 3.3-VDC-level I²C signals. Probably the biggest learning at this point was that, if possible, I should always include a three-pin header, exposing two available I/O pins and ground for a makeshift I²C debug interface on every ATtiny design. Having the LCD available as a software troubleshooting aid was invaluable.

At that point, the ATtiny could read acceleration data over the I²C bus from the accelerometer and, using a simple value mapping, turn on specific LEDs. However, the prototype implementation (see Photo 2) was not intended to be accurate enough for the actual footlight implementation I'd envisioned. I wanted to be able to smoothly transition from LED to LED. This would require using PWM to control the relative brightness between adjacent LEDs. That control would be based on the board's actual angle. To be accurate enough, the software needed to be able to calculate the angle of the board and thus the angle of the user's foot as they walked/ran. I thought this would be relatively easy given the plethora of open-source drone software available today, which used three-axis accelerometer data for heading calculations.

PHOTO 2

Here is the prototype board showing the QFN-16 accelerometer chip and the hand wiring to more easily expose the I²C signals for debugging.



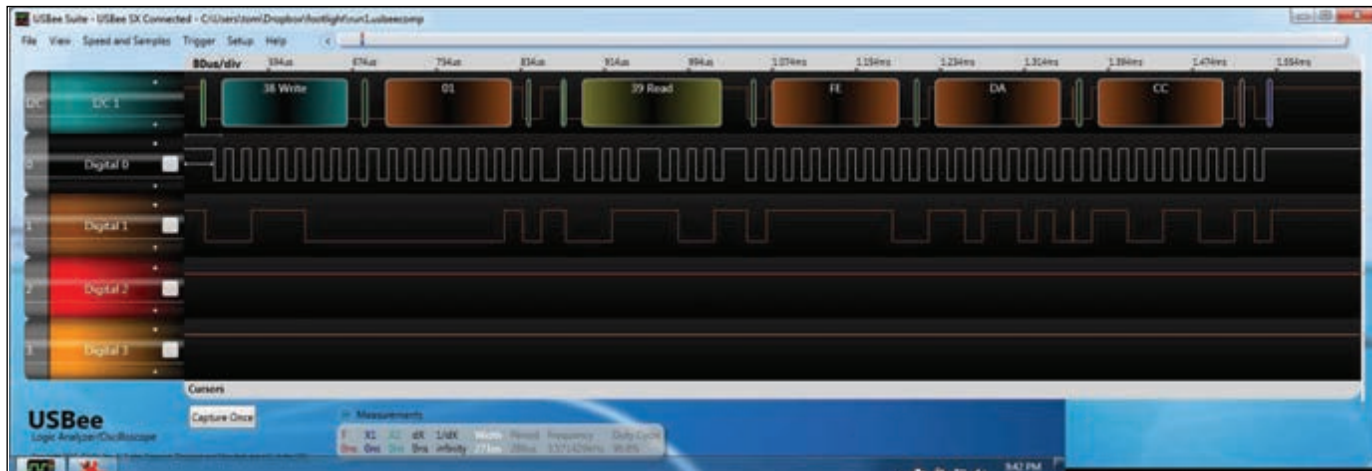


PHOTO 3

The ATtiny is reading the XYZ acceleration data via I2C as displayed by the USBee Suite's protocol decode mode.

Google came to the rescue again and turned up these relatively simple equations for determining heading from X/Y/Z accelerometer data (see **Equation 1**). I only needed to calculate the pitch angle and not the roll or yaw, so I only needed the equation for the Y Angle. Given I had written the prototype code in AVR GCC, it was a simple matter to translate the equation to C, and off we'd go. Simple heading calculation:

$$\begin{aligned} x\text{Angle} &= \text{atan}(ax/(\text{sqrt}(\text{square}(ay) + \text{square}(az))))); \\ y\text{Angle} &= \text{atan}(ay/(\text{sqrt}(\text{square}(ax) + \text{square}(az))))); \\ z\text{Angle} &= \text{atan}(\text{sqrt}(\text{square}(ax) + \text{square}(ay))/az); \end{aligned}$$

Sure enough, the code compiled the first time and loaded right into the ATtiny. I applied power, and... nothing happened. After much head scratching and several failed attempts to get any information written to the I²C LCD, I finally noticed that the compiler was reporting that both the program and data memory were over 200% full! No error messages, no red flags—it just quietly produced code too large for the device and then programmed that into the device with no errors. One great big learning here, always, always check the percent memory fill. I now moved to slash-and-burn coding to attempt to make the code small enough to fit the ATtiny261's limited memories.

My first attempt was to try to simplify the equation itself. Attempting to remove the SQRT, SQR, and/or ATAN was the obvious approach. Google again came to the rescue, turning up the fact that $\sqrt{(x^2 + y^2)}$ was actually a "distance" calculation and could be approximated by $\text{MAX}(|x|,|y|) + \text{MIN}(|x|,|y|)/2$.^[2]

This was a good lesson. It's not exact, but it's close enough and there are no squares or square root to deal with. Switching to the approximation formula improved things somewhat. The code did get much smaller, but it was still too big. Worse, while this reduced the program memory, it had no impact on the mysterious excessive data memory usage, so that was where I turned next.

While trying to figure out if removing the ArcTan function would reduce the code size, I happened to notice that when I left the ArcTan function call out, the data memory usage went almost to zero. It turned out that the AVR GCC ArcTan library function is a data memory hog—a real pig. Again, back to Google for ArcTan replacements, which turned up a piecewise linear curve fitting algorithm. This algorithm could be used to approximate any function.^[3]

One specific advantage of this function was that it makes no assumptions about the spacing of the control points. I could include more control points where the target function changed rapidly and fewer points where the function was more linear. That worked well for approximating the ArcTan function which included very long linear sections with 2 sharp curves in the middle. Throwing the whole thing into Excel let me adjust the number of points to manage the approximation error while still limiting the overall table size. After several attempts, I eventually got the approximation function down to a 33-element look-up table, requiring 66 numbers in total. I included my Excel spreadsheet on the *Circuit Cellar* FTP site along with the other code (see **Figure 1**). Thus, two major learnings here. First, the ArcTan function is a big memory

ABOUT THE AUTHOR


Tom Struzik has been building and taking things apart from an early age. He built his first Heathkit project at age 12 and sold his first computer program at age 16. Tom has a BSEE from Purdue University.

FPGA Boards from JAPAN

SAVING COST=TIME with readily available FPGA boards

XILINX Series Kintex-7 Artix-7 Spartan-6 Virtex-5

Kintex-7

- 3.3V single power supply operation
- Plenty of user I/Os: 100 or 128
- Line up 10 types of proven boards
- RoHS compliant 



DDR3 MRAM



DDR3



DDR3 MRAM USB Comm

ALTERA Series MAX 10 MAX V MAX II Arria II GX Cyclone V Cyclone IV

MAX 10

- Brand new MAX10 FPGA board
- Integrated A/D converter
- Integrated configuration memory
- RoHS compliant 



MRAM microSD



SPI-Flash
PLCC 68

PLCC68 Series Stamp size FPGA/CPLD Module



Easy and Quickly Mountable on 68-pin IC socket

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

ALTERA Series

XILINX Series



RoHS compliant 

Cyclone V FRAM Cyclone III

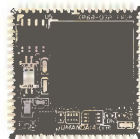


RoHS compliant 



RoHS compliant 

MAX V



RoHS compliant 

Spartan-6



RoHS compliant 

Spartan-3AN



RoHS compliant 

Spartan-6

Find our various products @amazon!!

all ▾

HUMANDATA



Go

HUMANDATA LTD.

TEL : +81-72-620-2002 (Japanese) FAX : +81-72-620-2003 (Japanese/English)
E-Mail : s2@hdl.co.jp URL : http://www2.hdl.co.jp/en/

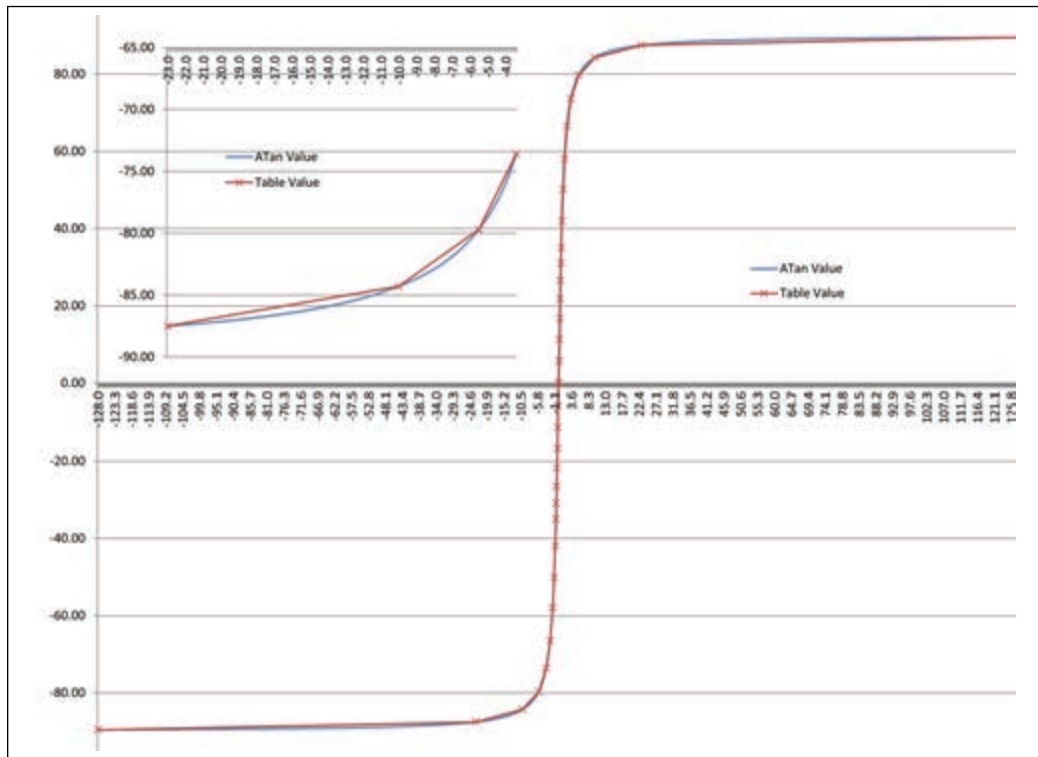


See all our products, A/D D/A conversion board, boards with USB chip from FTDI and accessories at :

www.hdl.co.jp/CC1510

FIGURE 1

This is the ArcTan approximation function. The large chart shows the overall situation. The small chart is zoomed in on one knee. The red line is the linear approximation. The blue is the actual Excel ATAN function result.



hog. And second, I now have an easy to use routine that can approximate any function. Good so far, the data memory usage was

again reasonable, but I was still in a failed state because the program memory usage was still too large.

After looking at the code yet again, I finally realized that I would just need to give up floating point altogether. The AVR GCC floating-point library was just too big. I bit the bullet and began work on switching to integer math but of course the value ranges meant I still needed to work with numbers less than 1. In fact, looking at the ArcTan function, the most critical range is between -1 and 1 so obviously integer math was going to be interesting. It was time to brush off my fixed-point arithmetic skills from long ago college days.^[4] Fixed-point arithmetic had multiple benefits. That divide by 2 in the distance approximation became a shift and the ArcTan table could be converted to directly return the correct fixed-point numbers—no conversion necessary. However, unlike floating point where step order did not matter, with fixed-point, step order was critical. The calculation steps had to ensure that no intermediate value dropped off significant digits or overflowed. For example, while $(A + B)/C$ would work just fine in floating point, with fixed point, that intermediate $A + B$ could overflow even though the ultimate result would still have been in range. Fixing this required implementing the steps as $(A/C + B/C)$ in order to keep the intermediate values within range of the chosen fixed-point format.

Woo-hoo! At that point the rotation angle calculation worked and fit into the ATiny261's



circuitcellar.com/cmaterials

Arithmetic," http://en.wikipedia.org/wiki/Fixed-point_arithmetic.

RESOURCES

Atmel, "AVR310: Using the USI Module as a I2C Master," 2561C-AVR-12/2013, 2013.

D. McFarland, "Simple Accelerometer Data Conversion to Degrees," 2013, <http://wizmoz.blogspot.com/2013/01/simple-accelerometer-data-conversion-to.html?m=1> Simple heading calculations.

SOURCES

ATtiny281 Microcontroller
Atmel | www.atmel.com

123D 3-D Design Software
AutoDesk | www.123dapp.com

USBee-SX Test Pod
CWAV | www.usbee.com

MMA8453QT Accelerometer
Freescale Semiconductor | www.freescale.com

I2C Master Library
Pete Fleury | <http://homepage.hispeed.ch/peterfleury/avr-software.html>

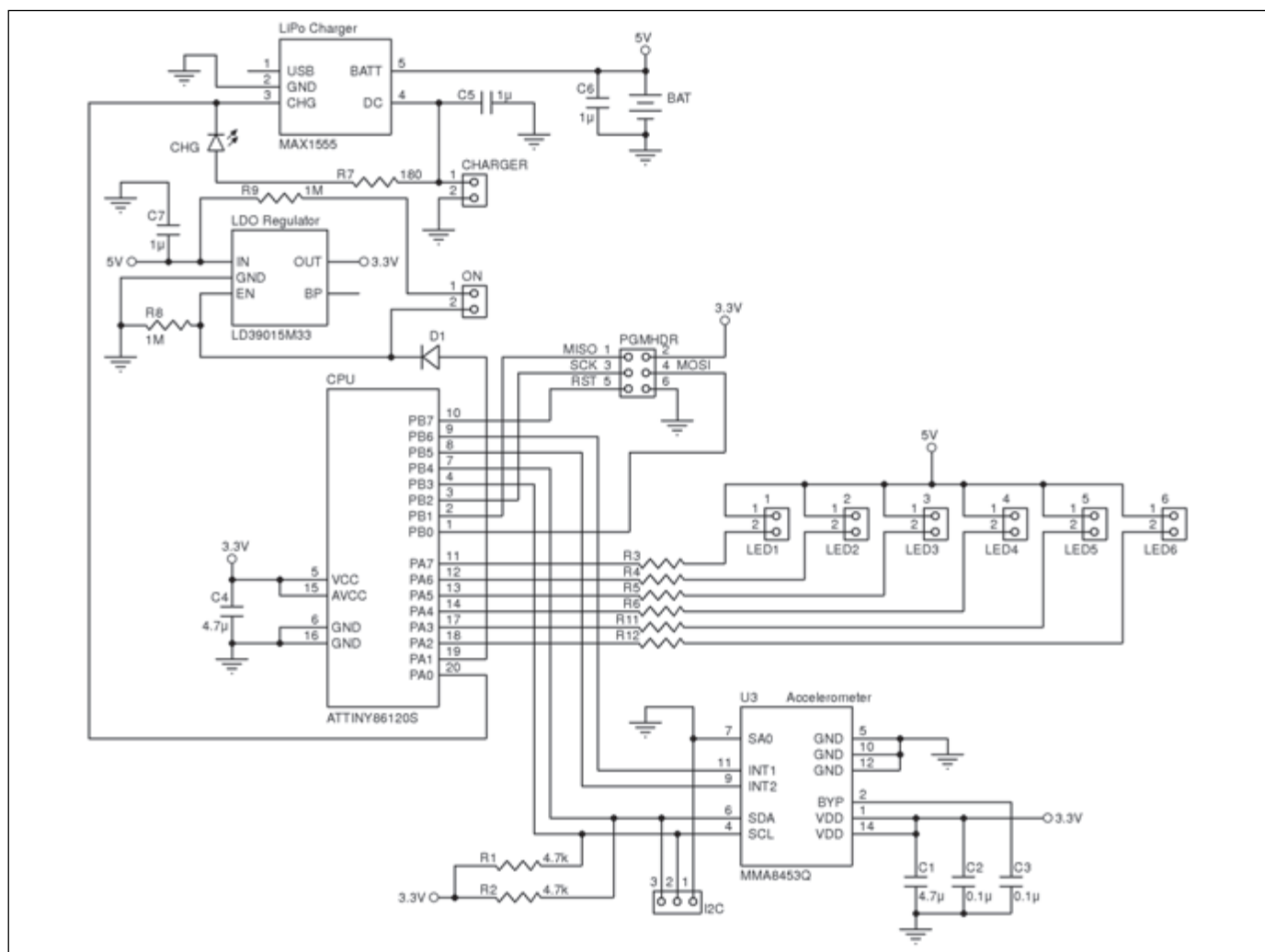
REFERENCES

[1] E. Walter, "30 Powerful Quotes on Failure," Forbes.com, 2013, www.forbes.com/sites/ekaterinawalter/2013/12/30/30-powerful-quotes-on-failure/.

[2] R. Baptista, "Fast Approximate Distance Functions," Flipcode.com, 2003, www.flipcode.com/archives/Fast_Approximate_Distance_Functions.shtml

[3] Linear Interpolation Function, www.ozgrid.com/forum/archive/index.php/t-69973.html.

[4] Wikipedia, "Fixed-Point



limited memory. I finally felt like this project finally had a chance to succeed—at least that was until I realize I had not yet implemented the code for either the LED PWM code or the user interface. I pressed forward and worked on implementing the LED PWM code and quickly discovered, yet again, that the code no longer fit into the ATtiny261's limited memory. I didn't think there was any way left to reduce the code any more. I finally threw up my hands and decided to switch to an ATtiny861, which had more data and program memory. But I still felt confident about being able to move the project forward. I believed all the outstanding issues were manageable.

While waiting for the shipment of an ATtiny861, I decided the project was far enough along to design what hopefully would be the final circuit board. I needed to add four more LEDs, design some type of on/off switch, and add a charger for a LiPo battery. This was relatively straightforward, but somewhat tedious because of the small size of the board required to fit the intended case. However, with a bit of diligence and after detangling

some signal lines, I finally succeeded in fitting everything within the required space, so again off to OSH Park for a board (see **Figure 2**).

LESSONS LEARNED

While waiting for OSH Park to return the new board, I thought it was time to take a step back and reflect on what I'd learned so far. I had learned that I could reliably mount a QFN-16 component (or so I thought). I learned how to read accelerometer data and convert that to a heading, all without using floating-point math. I also discovered an approximation function that fit into an ATtiny and yet would yield a good-enough ArcTan function, again all without floating-point math. I also determined that it would probably be a good idea to include an I²C port in future designs to support a “debug” LCD.


Next month, I'll detail how things really turned out and if this was the end of the learnings or simply the beginning of a much more challenging effort. Until then, keep trying new things. You never know what you might learn. 

FIGURE 2
This is the final circuit design. Or is it?

Sound Ecology and Acoustic Health (Part 4)

Room Acoustics Analysis

FEATURES

Last month, Adrien and Mike got quantitative with an audio record and analysis update for the WAT_AN_APP application. This month they discuss coding a room acoustics analysis.

By Adrien Gaspard and Mike Smith (Canada)

We have spent the last while working towards a mobile phone application to help identify a local noise nuisance problem. We joked with Mike Smith's neighbor's kids that the record and playback .3GPP file WAT_AN_APP application was developed to impress them that, "Without Any Teenage Assistance

Necessary, we could write an Android APP." We then added just enough additional code (JEAC) to store an audio record for later analysis. To continue the friendly tease presented in the first three parts of this article series, we pretended that the project code was actually designed to detect "Things that Go BOOM at

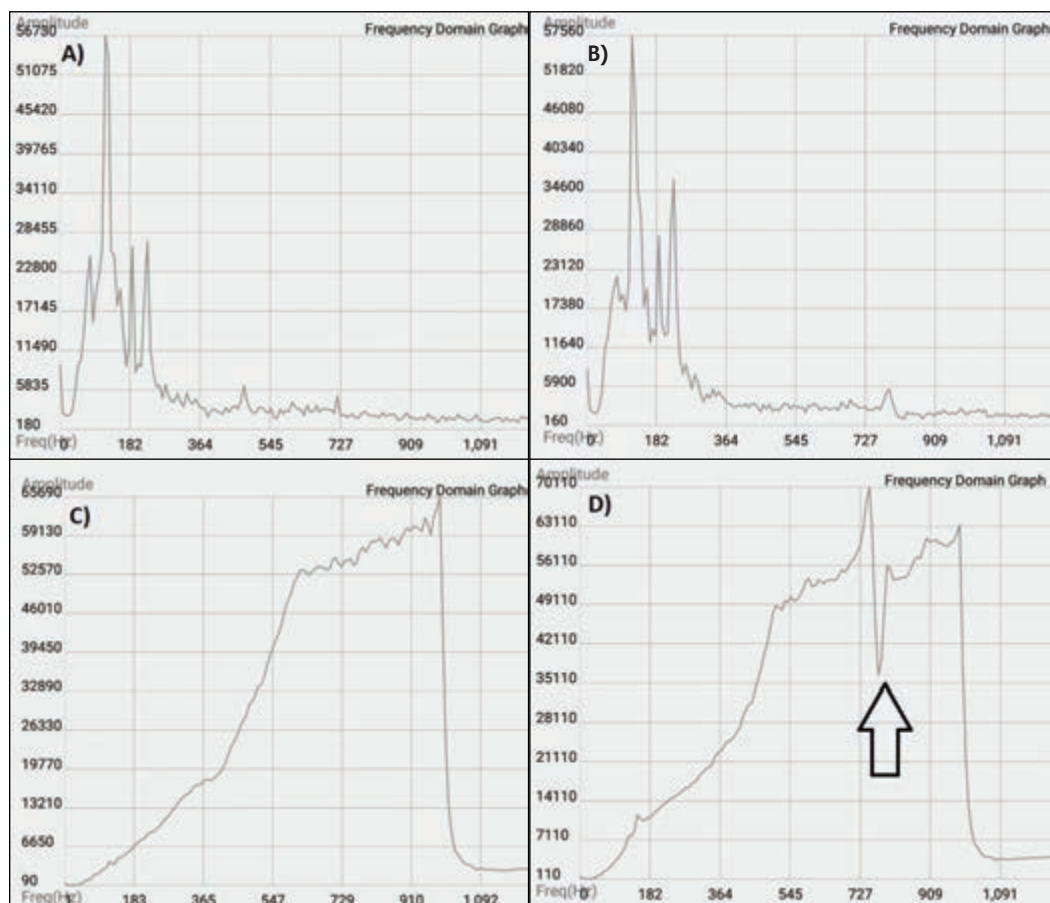


FIGURE 1

After adding the graphics capability from Article 5, we found that recording the frequency characteristics of our lab's background noise level indicates that we might be getting closer to a "727-Hz ghost" after moving between positions (a) and (b). However, the frequency characteristics of a Chirp sound burst (c) really change when we wake up the ghost and it flees the room (d).

Night" or how many "TGBN ghosts" are in the neighborhood.

As they say "Be careful what you wish for!" Our neighbors got interested in the community noise issues we were really trying to measure. They had their teenagers explore the acoustic health of their home using our work in progress. Late yesterday, a knock on the door revealed our neighbors asking for help. Their eldest teenager had gone to the University of Pennsylvania. According to the Penn Arts and Sciences website (sites.sas.upenn.edu/ghosts-healing), a group of scholars from literature, art history, nursing, archaeology, religious studies, science, and medicine wants to take research on ghosts seriously. So our neighbor's kid decided to volunteer with this group. This turned into a term project that involved analyzing room acoustics as a possible source of "that friendly spectral feeling." Hence, the frantic email message they wanted to pass on: *Term's nearly over! Could you please get Mike to hurry up and fulfill his promise in that first Circuit Cellar article of providing enough information to do some "real" digital signal processing (DSP) analysis? While he was at it—could he get Adrien to add some graphics' capability to display the frequency characteristics of the sounds in a room to make my term report more interesting!*

In Canada, it's always good to keep on the right side of the neighbor's kids as they are a good (inexpensive) labor source for shoveling snow off sidewalks. So, we decided to write a RoomAcoustics Analysis Capability addition. (Actually, we wanted to be able to say that we had Penn-ed some code. Sorry for the pun.)

First, we will explain how to excite a room resonance that can be captured by our existing TGBN detector code. We will graphically display the room audio signal to give us a first chance to compare resonance characteristics in different rooms. However, we found that looking for small differences in the captured signals displayed as a function of time meant working (slowly) with a lot of data. So we added a way to generate frequency information signal of captured signals using a discrete Fourier transform (DFT) algorithm code we grabbed from the web. **Figure 1a** shows the background noise recorded in our university lab. Having noticed a possible small 727-Hz ghost sleeping next to our desk, we tried to move around the room to better record its characteristic (see Figure 1b). The frequency characteristics of our two records look too similar for us to be sure that we have a non-snoring ghost close by.

We decided to wake it up by outputting a 3-s Chirp, a sound burst from 50 to 1,000 Hz. Figure 1c show the frequency response of the

Chirp signal, but there is not much there other than showing the poor low frequency of our phone's speaker. However, we accidentally got close enough that we woke up the sleeping ghost which significantly changed the frequency response of the room (see Figure 1d).

Want a ghost portrait? Sorry, you'll have to wait to the end of Article 5 to see a picture of the ghost we persuaded to live behind a van der Vaal's force field.

```

<!--Used by SoundAnalysis activity -->
1700.<RelativeLayout
1701. xmlns:android="http://schemas.android.com/apk/res/android"
1702. <!-- COPY FROM Article 1, Listing 2 Lines 102 to 105-->
1710. <TextView
1711. <!-- COPY FROM Article 3, Listing 3 Lines 1711-1716 -->
1720. <TextView
1721. <!-- COPY FROM Article 3, Listing 3 Lines 1721-1727 -->
1730. <TextView
1731. <!-- COPY FROM Article 3, Listing 3 Lines 1730-1739 -->
1740. <ProgressBar
1741. android:id="@+id/computation_progress"
1742. style="?android:attr/progressBarStyleHorizontal"
1743. android:layout_width="wrap_content"
1744. android:layout_height="wrap_content"
1745. android:layout_centerHorizontal="true"
1746. android:layout_centerVertical="true"
1747. android:indeterminate="false"
1748. android:max="100"
1749. android:progress="0"
1750. android:visibility="invisible"
1751. />
1760. <Button
1761. android:id="@+id/start_graph_time"
1762. android:layout_width="wrap_content"
1763. android:layout_height="wrap_content"
1764. android:layout_below="@id/number_tgbn_sounds"
1765. android:layout_centerHorizontal="true"
1766. android:layout_centerVertical="true"
1767. android:text="@string/button_start_graph_time"
1768. />
1770. <Button
1771. android:id="@+id/start_graph_freq"
1772. android:layout_width="wrap_content"
1773. android:layout_height="wrap_content"
1774. android:layout_below="@id/start_graph_time"

```

LISTING 1

The activity_sound_analysis.xml layout file from the WAT_AN_APP\res\layout folder

EXCITING A ROOM RESONANCE

We explained to our next-door neighbors how to excite a room resonance to our next door neighbors at the last BBQ before the snow fell. We lined up 10 glasses on a hard surface, each filled with different levels of water. You can generate a sound impulse if you clap your hands together. All of the glasses should have resonated, tinkled, as an impulse contains all possible frequencies in theory. However, persuading the BBQ group to sing "Do-re-mi" at the glasses proved a better way of getting enough sound energy at a particular frequency. Once your group has found a note

that starts a glass to sympathetically vibrate, then you can adjust their singing and get the "note" just right for a resonance. We were not sure about suggesting that our neighbor's teenager persuade the Penn Choral Society members to join his term project and sing in each room while he recorded them with the TGBN part of our WAT_AN_APP! Instead we explained how to use a more systematic and controlled approach—generating a Chirp signal—a longer duration sound containing all frequencies.

The history of Chirps is neat. In the early stages of AM radio, you could hear

LISTING 2

The prologue of the SoundAnalysis.java file (WAT_AN_APP\src\ folder) sets up the UI. The onCreate() method enables the UI composed amongst other of two buttons (Lines 605 and 606). The other methods in this class are detailed in the different listings.

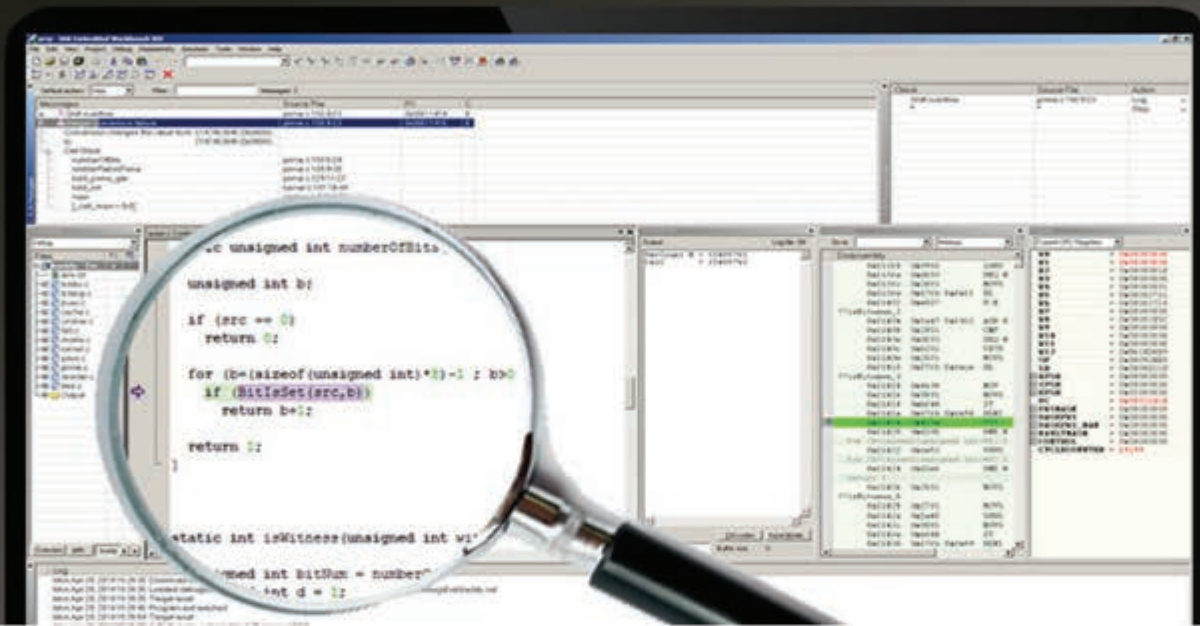
```

package com.wat_an_app;
// NEW CODE - INSERT AFTER Article 3, Listing 5 Lines 501- 512
513. import android.content.Intent;
514. import android.os.Environment;
515. import android.util.Log;
516. import android.view.Menu;
517. import android.view.MenuInflater;
518. import android.view.MenuItem;
519. import android.view.View;
520. import android.widget.Button;
521. import android.widget.ProgressBar;
522. import java.io.IOException;
    public class SoundAnalysis extends ActionBarActivity{
//NEW CODE - INSERT AFTER Article 3, Listing 5 Lines 551- 556
557. final CounterClass timer = new CounterClass(5000,250);
558. private static final double REFSPL = 0.00002; // Hearing reference level
559. private Button button_graph_time; private Button button_graph_freq;
560. private MediaPlayer mPlayer=null;
600. @Override protected void onCreate(Bundle savedInstanceState) {
//NEW CODE - INSERT AFTER Article 3, Listing 5 Lines 601- 603
//Delete Article 3 Listing 5 Line 604
605. button_graph_time = (Button) findViewById(R.id.start_graph_time);
606. button_graph_freq = (Button) findViewById(R.id.start_graph_freq);
607. }
// Some methods from Article 3, others described in Article 5
//protected void onStart() //Article 3 Listing 6 Lines 750 to 759
//protected void onPause() //Article 3 Listing 6 Lines 800 to 804
//public class CounterClass // Article 3 Listing 6 Lines 850 to 877
//protected void onPreExecute() //Article 3 Listing 8 Lines 910 to 919
//protected void onCancelled() //Article 3 Listing 11 Lines 1300 to 1305
//protected boolean detectImpulse() // Article 3 Listing 10 Lines 1350 to 1356
//protected boolean detectTGBN( ) // Article 3 Listing 11 Lines 1400 to 1406
//NEW AND MODIFIED METHODS
//public boolean onCreateOptionsMenu(Menu menu) //Listing 3 Lines 610 to 615
//public void startPlaying() //Listing 3 Lines 650 to 657
//public boolean onOptionsItemSelected(MenuItem item) //Listing 3 Lines 700 to 708
//public void onTick_Article4(long millisUntilFinished) // See Article 4 Lines 860 to 865
//protected Integer doInBackground() // See Article 4 Lines 910 to 1199
//protected void onProgressUpdate(Integer ... data) // See Article 5 Lines 1200 to 1220
//protected void onPostExecute(Integer data) // See Article 5 Lines 1250 to 1259
//protected int doubleFFT(double[][] samples, int numRecords, int sampleSize)
// See Article 5 Lines 1450 to 1468
//public static int nearestPow2Length(int length) // See Article 5 Lines 1500 to 1505
//public void DisplayGraph(View v)
// See Article 5 Lines 1600 to 1603

```


EXPLORE C-RUN FOR ARM

Runtime Analysis Simplified



C-RUN is a high-performance runtime analysis add-on product, fully integrated with world-leading C/C++ compiler and debugger tool suite IAR Embedded Workbench.

C-RUN performs runtime analysis by monitoring application execution directly within the development environment.

The tight integration with IAR Embedded Workbench improves development workflow and provides each developer with access to runtime analysis that is easy-to-use.

www.iar.com/crun



LISTING 3

Details of the methods dealing with the Action Bar used to display an icon outputting a Chirp sound saved in the phone memory

```

610. public boolean onCreateOptionsMenu(Menu menu) {
611. // Inflate the menu items for use in the action bar
612. MenuInflater inflater = getMenuInflater();
613. inflater.inflate(R.menu.menu_sound_analysis,menu);
614. return super.onCreateOptionsMenu(menu);
615. }
650. public void startPlaying() {
651. mPlayer = new MediaPlayer();
652. try {
653. mPlayer.setDataSource(Environment.getExternalStorageDirectory().
getAbsolutePath()+"/MySounds/Chirp_50_1000Hz.wav");
654. mPlayer.prepare();
655. mPlayer.start();
656. } catch (IOException e) {}
657. }
700. public boolean onOptionsItemSelected(MenuItem item) {
// Handle presses on the action bar items
701. switch (item.getItemId()) {
702. case R.id.GenerateChirp:
703. startPlaying();
704. return true;
705. default:
706. return super.onOptionsItemSelected(item);
707. }
708. }

```

the interference of lightning strikes some distance away as a crackle on the loudspeaker. If you slightly mistuned the radio, the crackle changed into something else. What happened was that the lightning strike, an electromagnetic (EM) impulse, generated a signal containing "all" EM frequencies. Signals at different frequencies traveled at different speeds through the air because of an effect called EM dispersion, and arrived at the radio at different times. Analog down-sampling within the AM radios caused the different EM frequencies to turn into different audio signals. So you heard what sounded like a bird whistle starting at low audio frequencies and going up to high. Hence the name Chirp. You can generate an audio Chirp using a pedestrian underpass. If you clap your hands near the entrance, then you get a "doo-WEE" sound rather than a single clap echo. The "doo" sound is the reflection of low frequency sounds arriving back at your ears at an earlier time than the high frequency "WEE" reflections. Our preferred way of generating a 3-s Chirp sound burst is using

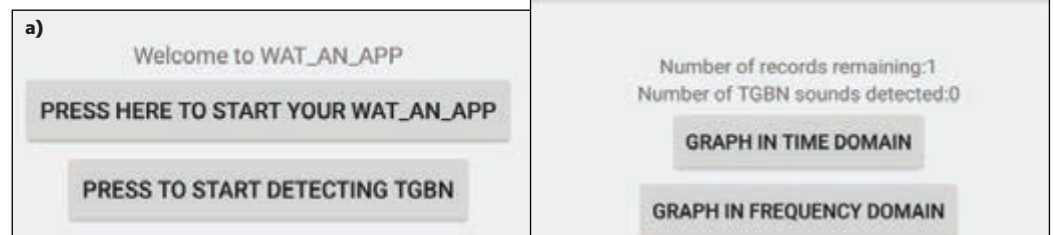
Audacity (audacity.sourceforge.net) on a laptop.

In this article, we make an application capable of both outputting a Chirp and capturing the room sound ecology. However, it is not our favorite approach as generating a Chirp on a mobile has issues. The mobile audio electronics are not really designed to handle low frequency output or provide a lot of audio power without distortion (see Figures 1c and 1d). However, plugging in an external speaker to boost the output cuts out the microphone input—a classic "CATCH-22" situation! We can fix that by capturing and then playing back the laptop Chirp as a .3GPP file on another Android phone using external speakers. By modifying the code in this article to output a NADA.3GPP file of "the sound of silence" rather than a stored Chirp you can have the best of both worlds!

Our plan for running the extended WAT_AN_APP is to press the "Press To Start

FIGURE 2

We can start the sound analysis from the MainActivity (a). The TGBN SoundAnalysis activity allows us to output a Chirp, perform our calculations, and call the activity that displays time and frequency graphs (b).



Detecting TGBN" button from the MainActivity screen that pops up when the application starts (see **Figure 2a**) to start the TGBN SoundAnalysis activity. To ensure that the sound capture starts immediately modify the ints.xml file from the "WAT_AN_APP\res\values" folder and set the threshold at which we start the recording from "10000" to "0" (see Article 3, Listing 4, Line 2840). As we will explain later, the DSP frequency analysis code works best with a 5-s sound sampling so change the capture time from "7" to "5" in the ints.xml file.

As shown in Figure 2b, we can press the play icon located in the action bar (top right corner of the screen) to output a three seconds Chirp signal. After the sound capture, you can display the audio information as either an audio time graph or an audio frequency graph.

GETTING STARTED

This article is an extension of Part 3 of this series, so you will see a lot of similar file names and line numbers. The code for the MainActivity is identical to Listing 1 and Listing 2 in Article 3. As a reminder, the setContentView() makes use of the layout file activity_main.xml to generate a screen with a message welcoming the user in the application, and two buttons to start the audio record/playback and audio analysis activities. When we called the SoundAnalysis activity in Article 3, we initialized the recorder, detected a sound, and recorded it into a local array. Then we did some simple DSP analysis—check to see if the recorded data was above a certain threshold. This time, we are going to manipulate the recorded data using some DSP program before graphing the results using the GraphView library.

THE SOUND ANALYSIS ACTIVITY

SoundAnalysis uses an asynchronous task which offloads the computations to a worker thread. This is necessary to not stall out the user interface (UI) thread which could cause the UI to stop responding. The SoundAnalysis activity's layout, activity_sound_analysis.xml is described in **Listing 1**. The TextView displaying the number of records remaining and the number of TGBN sounds detected (Lines 1710 to 1739) are identical to Listing 3 in Article 3. We simply add a progress bar on the screen, translating the background FFT calculations (Lines 1740 to 1751), as well as two buttons, to start displaying graphs in the time (Lines 1760 to 1768) or frequency domain (Lines 1770 to 1778).

Listing 2 shows code to add the time and frequency display buttons, Lines 559, 605 and 606. Overviews of all the methods we need to develop are given at the end of Listing 2. The

```
2500.<item
2501. android:id="@+id/GenerateChirp"
2502. android:icon="@drawable/ic_action_play"
2503. android:showAsAction="always"
2504. android:title="@string/audio_play"
2505. />
```

LISTING 4

menu_sound_analysis.xml file from WAT_AN_APP\res\menu folder to setup the action bar in the activity

SoundAnalysis activity covers a lot of code. (To avoid typing, cut and paste the line from the listings in Circuit Cellar electronic version, or visit the Circuit Cellar FTP site.)

The three methods in **Listing 3** set up an action bar so that we can press an icon that output a Chirp sound which the application will record. The run-once onCreateOptionsMenu() method, Line 610, is responsible for handling the content of the activity's menu that appears on the action bar. As this is our first action bar, we need to set up a "menu" folder in "WAT_AN_APP\res\menu. Add the "menu_sound_analysis.xml" file to this folder and insert the code given in **Listing 4** (Lines 2500 to 2505). In order that Listing 4 (Line 2502) can display a play icon we need to add an "ic_action_play.png" file in the "WAT_AN_APP\res\drawable-mdpi" folder. This icon can be obtained from a "...\Action Bar Icons\holo_dark\09_media_play\drawable-mdpi" folder, following the Quick Help Guide from Article 2 to have more information on adding an icon.

Lines 700 to 708 in Listing 3 show the onOptionsItemSelected() hook called whenever an item in the option menu is selected. Line 703 activates the startPlaying() method which outputs a Chirp to play a sound from 50 to 1,000 Hz. The startPlaying() method, Lines 650 to 657, outputs the Chirp_50_1000Hz.wav file located in a folder "MySounds" that we must add into the root directory of our phone's internal memory. We found that pushing the Chirp play-button by hand meant that we often lost the last

```
860. public void onTick_Article4(long millisUntilFinished) {
861.     int capture_time_ms=getResources().
           getInteger(R.integer.capture_time) * 1000;
862.     if(millisUntilFinished> (capture_time_ms - 650) &&
           millisUntilFinished< (capture_time_ms - 400)){
863.         int playChirp =getResources().getInteger(R.integer.playChirp);
864.         if (playChirp==1) startPlaying();
865.     }
866. }
```

LISTING 5

onTick_Article4() from the CounterClass

LISTING 6

Saving recorded data into the time domain using the `doInBackground()` step from the `CaptureAudio` class

```

//COPY FROM Article 3, Listing 6 and Listing 7
private class CaptureAudio extends AsyncTask<Void, Integer, Integer>{
//NEW CODE - INSERT AFTER Article 3, Listings 8 and 9 Lines 910- 981
982. detectBuffer = null;
//TO UPDATE FROM Article 3
//sampleBuffer = null;
984. if (recorder != null) { recorder.release(); recorder = null; }
985. if (!isCancelled()) publishProgress(-1, -1, -1, -1, 0, -1);
//return 0;} //Delete 986 and 987 to allow extended background task to work
990. final int numRecords = getResources().getInteger(R.integer.num_records);
// copy the buffer into a buffer of double
991. double[][] samples =new double[numRecords][sampleBufferLength];
992. double max = 0;
993. for(int n = 0; n < sampleBufferLength; n++){
994.     samples[0][n] = (double) sampleBuffer[0][n]; // Identify maximum value
995.     if(max < Math.abs(samples[0][n])) {max = samples[0][n];}
996. }
997. for(int h = 0; h < sampleBufferLength; h++) {samples[0][h] /= max;}
// Grab first record for analysis and display
1000. double[] toStorage_time = new double[sampleBufferLength];
1001. for (int n = 0; n < sampleBufferLength; n++) {
1002.     toStorage_time[n] = samples[0][n] / REFSPL;
1003. }
1004. if (isCancelled()) {return -1;}
// reduce the size of our sample so the graph can load in a normal amount of time
1005. int samplesPerPoint = getResources().getInteger(R.integer.
    samples_per_bin_time);
1006. int width_time = toStorage_time.length / samplesPerPoint ;
1007. int samplerate = getResources().getInteger(R.integer.sample_rate);
1008. double maxYval_time = 0;
1009. final double[] tempBuffer_time = new double[width_time];
1010. for (int k = 0; k < tempBuffer_time.length; k++) {
1011.     for (int n = 0; n < samplesPerPoint; n++){
1012.         tempBuffer_time[k] += (samples[0][k*samplesPerPoint + n] / REFSPL);
1013.     }
1014. tempBuffer_time [k] /= (double) samplesPerPoint;
1015. if (maxYval_time < tempBuffer_time [k]){
1016.     maxYval_time = tempBuffer_time [k];}
1017. }
// scaling the x "time" values stored into xVals
1018. final double[] xVals_time = new double[tempBuffer_time.length];
1019. for (int k = 0; k < xVals_time.length; k++) { // xVals.length=512
1020.     xVals_time [k] = k * (1.0*samplesPerPoint) / (samplerate);
1021. }
//Adding properties to clicking on the "GRAPH IN TIME DOMAIN" button
1025. button_graph_time.setOnClickListener(new View.OnClickListener() {
1026.     public void onClick(View arg0) {
1027.         String which_button_pressed = "1";
1028.         Bundle extras_time_values = new Bundle();
1029.         extras_time_values.putDoubleArray("key_x_time", xVals_time);
1030.         extras_time_values.putDoubleArray("key_y_time", tempBuffer_time);
1031.         extras_time_values.putString("button_pressed", which_button_pressed);
1032.         Intent intent_graph_time = new Intent(SoundAnalysis.this,DisplayGraph.class);
1033.         intent_graph_time.putExtras(extras_time_values);
1034.         intent_graph_time.putExtras(extras_time_values);
1035.         intent_graph_time.putExtras(extras_time_values);
1036.         startActivity(intent_graph_time);
1037.     }
1038. }); // Continues in Listing 7

```

part of the Chirp. Rather than increasing the recording and analysis time we modified the `onTick_Article4()` mentioned in Article 3 to automatically play the Chirp about 400 ms after recording had started **Listing 5** (Lines 862 to 865).

TIME DOMAIN ANALYSIS

Our Article 3 code allowed the capture of up to six sound records. For calculation time and other issues, we decided to analyze just

one record by changing the variable “`num_records`” (Article 3 Listing 4, Line 2830) from “6” to “1” in the `ints.xml` file. As you can see in Figure 1a, one record provided frequency domain signals with a good signal-to-noise (SNR) ratio. You will have to synchronize the start of the Chirp output capture to better than one sample period (1/8000 s). This is not straightforward when you have Android tasks running in addition to our `WAT_AN_APP`.

In Article 3, we mentioned how the

```

//Call the function that process the FFT
1100. int error = doubleFFT(samples, numRecords, sampleBufferLength);
1101. if (error == -1) {
1102.     if (!isCancelled())
1103.         publishProgress(-1, -1, -1, -1, -1, 0); //display error message
1104.     sampleBuffer = null;
1105.     return -1;
1106. }
// Grab first record for analysis and display
1107. double[] toStorage_freq = new double[sampleBufferLength];
1108. for (int n = 0; n < sampleBufferLength; n++) {
1109.     toStorage_freq[n] = (samples[0][n] / REFSPL);;
1110. }
1111. if (isCancelled()) {return -1;}
// reduce the size of our sample so the graph can load in a normal
1116. int samplesPerPoint_freq = getResources().getInteger(R.integer.samples_per_bin_freq);
1117. int width_freq = toStorage_freq.length / samplesPerPoint_freq / 2;
1118. double maxYval_freq = 0;
1119. final double[] tempBuffer_freq = new double[width_freq];
1120. for (int k = 0; k < tempBuffer_freq.length; k++) {
1121.     for (int n = 0; n < samplesPerPoint_freq; n++)
1122.         tempBuffer_freq[k] += toStorage_freq[k * samplesPerPoint_freq + n];
1123.     tempBuffer_freq[k] /= (double) samplesPerPoint;
1124.     // Log.d("ADebugTag", "Value of tempBuffer: " + Double.toString(tempBuffer[k]));
1125.     if (maxYval_freq < tempBuffer_freq[k]) {maxYval_freq = tempBuffer_freq[k];}
1126. }
// Save X data
1130. final double[] xVals_freq = new double[tempBuffer_freq.length];
1131. for (int k = 0; k < xVals_freq.length; k++)
1132.     xVals_freq[k] = k * samplerate / (2 * xVals_freq.length);
1135. button_graph_freq.setOnClickListener(new View.OnClickListener() {
1136.     public void onClick(View arg0) {
1137.         String which_button_pressed = "2";
1138.         Bundle extras_freq_values = new Bundle();
1139.         extras_freq_values.putDoubleArray("key_x_freq", xVals_freq);
1140.         extras_freq_values.putDoubleArray("key_y_freq", tempBuffer_freq);
1141.         extras_freq_values.putString("button_pressed", which_button_pressed);
1142.         Intent intent_graph_freq = new Intent(SoundAnalysis.this, DisplayGraph.class);
1143.         intent_graph_freq.putExtras(extras_freq_values);
1144.         intent_graph_freq.putExtras(extras_freq_values);
1145.         intent_graph_freq.putExtras(extras_freq_values);
1146.         startActivity(intent_graph_freq);
1147.     }
1148. });
1149. return 0;
1199. }// Continues in Article 5

```

LISTING 7

Saving recorded data into the frequency domain using the `doInBackground` step from the `CaptureAudio` class

SoundAnalysis activity uses an asynchronous task, CaptureAudio, to record and analyze the sound, and to update the activity's UI with this class that makes use of four steps. The same first step, onPreExecute, performs any necessary setup. The second step, doInBackground, is invoked to perform any background computations that take a long time. doInBackground needs to be extended by adding the **Listing 6** code to the TGBN detector doInBackground code from Article 3 Listing 9.

We need to do some housekeeping steps to allow the extended background activity to execute: after having inserted the code from Article 3 Listings 8 and 9, uncomment the line 956, as it will make sure that the sample buffer we are working with has a length that is a power of 2 for the FFT calculations. If you forget this step, the FFT calculations won't be processed and an error message will appear on the screen. Remove the setting of sampleBuffer pointer to null in Line 983, remove the return statement in Line 986 and the curly bracket for Line 987.

Preparing data for graphing can be time consuming unless the mobile CPU has lots of horsepower. There are efficient Android graphing packages available if the data is stored within a SQL database. As employing those will take another series of articles, we have taken a straightforward, bull-at-gate approach. You have to handle possible overflow issues when doing integer DSP calculations. So in Lines 991 to 997, we convert the audio record to doubles, allowing graphing of the recorded sound. Lines 1000 to 1003 allows grabbing the record for future analysis and display. The samples are stored


into an array toStorage_time and divided by a constant REFSPL, which is the threshold of the human hearing, used here as the reference sound pressure level and equal to 0.00002 (Line 1002).

Displaying the sound signal generates another CATCH-22 scenario. If you display all the information, then you have 4 s off sound (32,000 point) displayed on a small screen. It takes forever to zoom in. We decided to speed the time display and zooming by doing a rough form of down sampling, Lines 1005 to 1017. If you try this with the Chirp signal, then you will find that your display does not show constant sound amplitude. The strong amplitude variation shows that you are no longer satisfying the Nyquist sampling rate and you get "display aliasing."

The "x-axis" time values are generated through Lines 1018 to 1021. We make the button "Graph in Time domain" clickable by using setOnClickListener(), Line 1025. Line 1036 calls the graphing DisplayGraph activity we'll discuss later. This call requires the graph's "x" and "y" values to be passed from the SoundAnalysis activity to the DisplayGraph one using a bundle, Line 1028. This bundle makes use of two strings, "key_x_time" and "key_y_time", Lines 1029 and 1030 to pass the arrays "xVals_time" and "tempBuffer_time". We put the string containing the value of the button that has been pressed to start the DisplayGraph activity into a key "button_pressed", Line 1031. We then declare an intent that will start the DisplayGraph activity, and pass our two arrays and one string via the bundle "extras_time_values", Lines 1033, 1034, as well as the button that has been pressed to call the graphing activity, Line 1035. The data in the time domain can now be graphed.

The Quick Guide shows a way to write array's values into an external .txt file stored in the phone internal memory for debugging analysis off the phone. If you are not interested in saving the values into a file, you can also display the content of the tempBuffer and xVals arrays in the LogCat by using the "Log" API that sends log output in the LogCat window, as we show in **Listing 7** Line 1124. This line of code displays the recorded data amplitude's values in the frequency domain.

A GHOST TO COME

We are about half-way there to getting the full DSP code ready to assist out our neighbor's kid with his volunteer work. In the final article of this series, we will tackle getting the fast Fourier transform (FFT) code to handle spectral analysis. Then we head out into the world of Android graphics, and provide that promised picture of a ghost. 

ABOUT THE AUTHORS

Adrien Gaspard (gasp.adrien@gmail.com) earned a Masters of Engineering from CPE Lyon, France, in February 2015. He tackled his final practicum as an exchange student in Electrical and Computer Engineering at the University of Calgary. He undertook self-directed term projects directed towards the possible use of noise cancelling to solve the community noise problem in Calgary community of Ranchlands. Adrien intends to focus his career in the fields of embedded systems and wireless telecommunications.

Mike Smith (Mike.Smith@ucalgary.ca) has been contributing to *Circuit Cellar* since the 1980s. He is a professor of Computer Engineering at the University of Calgary, Canada. Mike's main interests are in developing new biomedical engineering algorithms and moving them onto multi-core and multiple-processor embedded systems in a systematic and reliable fashion. He is a recent convert to the application of agile methodologies in the embedded environment. Mike has been an Analog Devices University Ambassador since 2001.

INSTALL AN APK FILE ON AN ANDROID DEVICE

The Android application package file (APK) is the format of installable files on Android platform. In order to install an .apk on your phone, enable the “unknown sources” on Android (settings, security and check the box next to “unknown sources”). A dialog box pops up asking you to confirm the action, knowing that “your phone and personal data are more vulnerable to attack by apps from unknown sources.” Tap OK if you want to allow the .apk from the *Circuit Cellar* FTP site to run, but make sure to always know the source of the .apk files you install on your device. Connect your phone to the computer, and in the “USB Storage” folder, create a “My Applications” folder. Get the application .apk from the *Circuit Cellar* FTP site and copy it from your computer’s download folder (“C:\Users\ajfgaspa\Downloads” by default) to the “My Applications” directory on your phone using your computer. From your phone, using the Google Play store, download and install an application as “file commander” (play.google.com/store/apps/details?id=com.mobisystems.fileman&hl=en) to take control over the files stored in your phone. Start file commander, tap on the “USB storage” folder, and then on “My Applications” folder: the application .apk file should be here. Tap it. A window pops up asking you to “complete action using.” Select “Package installer”, and confirm that you want to install this application. A few second later, the application has been installed. Click on “Open.” The application starts!

DISPLAY VALUES IN THE LOGCAT

To display buffer values on the LogCat—e.g., tempBuffer_time with amplitudes “y” values—add import android.util.Log in Listing 2 after Line 522, and then between Line

```
// To make this output work, you will need to add these imports to
// Listing 2 after line 522 -- import android.net.Uri; import java.io.File;
// import java.io.FileNotFoundException; import java.io.FileOutputStream;
// import java.io.OutputStreamWriter; import java.util.Arrays;
// Insert this code in Listing 6 after Line 1021 and before Line 1025
String file_path4 = Environment.getExternalStorageDirectory()
    .getAbsolutePath() + "/Android/";
//Store file in the Android folder from the phone Internal memory
File file4 = new File(file_path4 + "/YValuesfromSOUNDANALYSIS.txt"); //file's name
sendBroadcast(new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE,
    Uri.fromFile(file4))); //force the file to be displayed
FileOutputStream fos4 = null;
try { fos4 = new FileOutputStream(file4); }
catch (FileNotFoundException e1) { //exception: file not found
    e1.printStackTrace(); }
OutputStreamWriter osw4 = new OutputStreamWriter(fos4);
try {
    String b4 = Arrays.toString(tempBuffer_time);
    //display the array "tempBuffer_time" in a text file as a string
    osw4.write(b4); osw4.flush();osw4.close();
}
catch (FileNotFoundException e) { e.printStackTrace(); }
catch (IOException e) { e.printStackTrace(); }
```

LISTING 1

This code can be used to save the amplitudes of the time domain recorded data (referred as « tempBuffer_time ») into an external .txt file, stored into the phone memory. Similar code will also store the “xVals_time” in a text file as a string.

LISTING 2

Modifying AndroidManifest.xml file will cause graphs to appear in “easy-to-read” landscape mode and stop the keyboard from hiding parts of the graph.

1013 and 1014 in Listing 6 add: Log.d(“ADebugTag”, “Value of tempBuffer_time: “ +Double.toString(tempBuffer_time[k])):.

STORING AUDIO VALUES IN A .TXT FILE

To save data in an external .TXT file on a phone’s memory, refer to **Figure 1** and **Listing 1**.

FORCE AN ACTIVITY TO START IN LANDSCAPE MODE

We always want to start up displaying graphs in the landscape screen orientation. Generate the graphing DisplayGraph activity as demonstrated in Article 3 to prepare for the graphing activities in Article 5. Now, modify the AndroidManifest.xml file where you see the activity line with the words android:name =“.DisplayGraph”. Add the lines for android:screenOrientation and android:configChanges as shown in **Listing 2** to cause landscape mode and stop the phone’s keyboard from hiding parts of the graph.

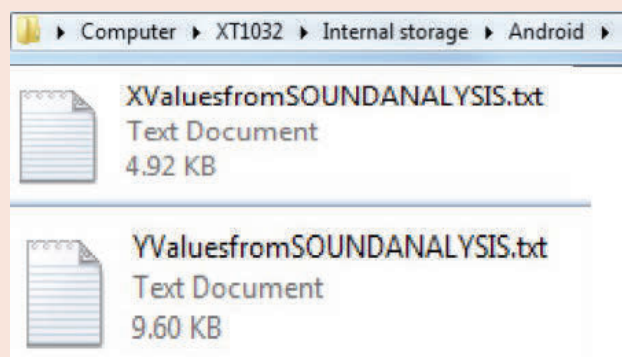


FIGURE 1

Saving data into an external .TXT file on the phone’s memory

```
<activity
    android:name=".DisplayGraph"
    android:screenOrientation="landscape"
    android:configChanges="orientation|keyboardHidden"
    android:label="@string/title_activity_display_graph">
</activity>
```

Running on Battery (Part 2)

FEATURES

Battery Operation

In the first part of this series, Stuart explained how to protect against reverse battery voltage, select a microcontroller, and manage power. Here he examines combined AC/battery operation, single-cell operation, rechargeable batteries, and estimating battery life.

By *Stuart Ball (US)*

In the first part of this article series, we looked at protecting against reverse battery voltage, selecting a microcontroller, and power management. In this article, we will look at combined AC/battery operation, single-cell operation, rechargeable batteries, and estimating battery life.

COMBINED AC/BATTERY OPERATION

In many applications, you want to operate your circuit from either AC (from a wall outlet) or batteries. AC operation typically involves a wall mount or other external power supply with a DC output. You could also embed an AC supply in the housing for the circuit. You might do that if your circuit normally operates from AC with battery only as backup. I won't go into power supply design here.

Typically, when you want to operate from the AC supply, you plug in the AC/DC adapter and the circuit switches to use the external supply. To do that, you need to disconnect the battery from the circuit when the AC supply is in use. Usually, the point of AC operation is to avoid battery drain, and possibly to charge the battery. You can make the changeover from battery to the external supply using a

mechanical or electronic switch.

Some coaxial DC power adapters have a pair of contacts that open when the external DC plug is inserted. The battery lead (usually the negative side) is run through these contacts so that the battery is disconnected when the external DC is plugged in. **Figure 1** shows this configuration. The figure shows a coaxial DC jack, but other types of connectors could also be used—0.125" audio jacks are common. Obviously, this won't work if the supply is internal since there is no external DC plug in that case. For an internal supply, if you want automatic switching, you have to use an electronic switch.

The drawbacks to using the switched jack method are, first, that the contacts on the connector are an added failure point. Connectors and switches tend to be a high failure item in electronics. The second issue is that there may be momentary loss of power while the DC plug is being inserted. This happens if the battery contacts open before the external supply is fully connected.

Another method, also shown in Figure 1, is to use a Schottky diode in series with the external DC supply, and a second diode in series with the positive lead of the battery.

This eliminates the extra set of contacts used in the switched jack method, and it eliminates the potential for momentary power loss. However, this approach does lose around 0.3 V from the battery and the DC supply due to the drop across the diodes.

For diode-based electronic switching, the Schottky diode in the battery circuit also prevents reverse battery polarity. (See Part 1 of this article series.) If you were going to use a diode for reverse polarity protection anyway, you don't need an additional diode in series with the battery for isolation with an external supply; the same diode will do both jobs.

Figure 1 indicates a 12-V external DC power supply voltage with a 9-V battery. The specific voltages are not important, but this circuit will only work if the external supply voltage is higher than the battery voltage. If the external supply voltage is lower than the battery voltage, D2 will be reverse-biased and the battery will still be powering the circuit. If you are using a 6-V battery, you could use any DC supply greater than 6 V. Common voltages are 7.5, 8, and 9 V.

Many wall mount-type supplies output greater voltage than the rated voltage at low loads, so you might be able to use a 9-V supply with a circuit using a 9-V battery. I have one wall mount supply that produces over 12-V unloaded. But if you try that, test it in an operating circuit or check the manufacturer's specifications for output voltage vs. current. The voltage may drop below the battery voltage under load.

A third way to handle AC/battery switching is to use a part such as the Linear Technology LTC4412. This IC, coupled with an external MOSFET, allows external supply operation from 3 to 28 V, and battery operation from 2.5 to 28 V. It includes reverse battery protection, so there is no need for an additional Schottky diode for the battery.

The LTC4412 has a logic output to indicate when the external supply is connected. The microcontroller can sense this to enable an LCD backlight, battery-charging circuit, or some other feature that is normally disabled during battery operation. The LTC4412 still requires that the external DC voltage be greater than the battery voltage to enable the switch from battery to external DC power.

There are other, similar battery switch parts, including the Intersil ICL7673. The ICL7673 switches the output to whichever input has the higher voltage. It is available in both SMT and DIP configuration, making prototyping easy.

Often the battery or power supply is followed by a regulator to provide the correct voltages for the circuit. You want to be sure

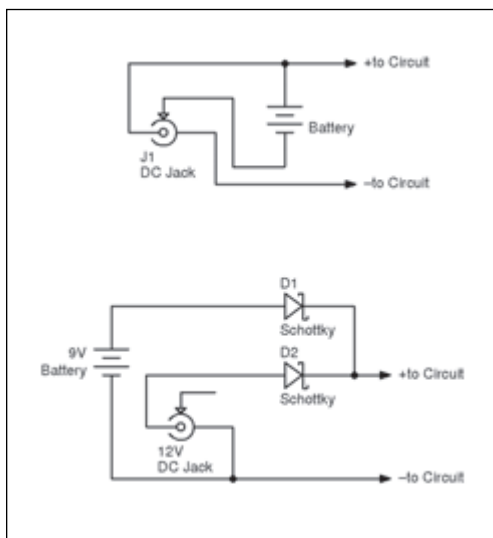


FIGURE 1

This is how to use of a power plug with an extra contact in the negative terminal to isolate the battery when an external DC supply is used. I also show how to use Schottky diodes to isolate battery when external DC supply is used.

that the regulator, if one is used, is capable of handling the additional power when the external supply is connected. For example, take the simple case of a 9-V battery with a 78L05 regulator to produce a 5-V output. If the circuit draws 20 mA, the dissipation in the regulator on battery is 80 mW ($9\text{ V} - 5\text{ V} \times 20\text{ mA}$). If an external 12-V supply is used for AC operation, the regulator dissipation goes to 140 mW ($12\text{ V} - 5\text{ V} \times 20\text{ mA}$).

For most battery circuits, you wouldn't use a simple 78L05 because the quiescent current is too high. But the same principle applies to any regulator; make sure that the dissipation isn't exceeded while operating on the higher DC voltage from the AC supply.

SINGLE-CELL OPERATION

In some cases you want to operate your circuit from a single 1.5-V battery. This allows for a smaller, lighter design. But there is less voltage available to drive LEDs or audio outputs or LCD bias inputs.

In many single-cell applications, the microcontroller voltage will need to be higher than the battery. One way to do this is to use a DC-DC converter to step the battery voltage up to a higher voltage such as 3.3 V. There are numerous DC-DC boost converter ICs that are designed for battery operation, such as the Linear Technology LTC3525. This part is a burst mode regulator, so if the microcontroller is in sleep mode to save power, the regulator will draw only enough current from the battery to keep the output voltage constant.

One problem with an inductor-switching boost converter is the current draw during regulation. This reduces battery life. A switched-capacitor charge pump such as the Linear Technology LTC1502 has lower current drain but significantly less output current capability (10 mA in the case of the LTC1502).

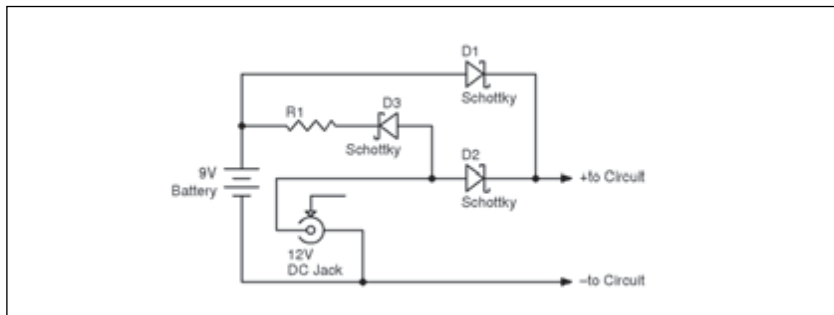


FIGURE 2

Simple recharging circuit based on diode-switched AC adapter

One way to minimize battery drain is to operate the microcontroller from a low-current charge pump boost converter, and run any higher current devices, such as LEDs, from a switched-inductor boost converter. During sleep mode, the microcontroller shuts down the switched-inductor converter to minimize current draw, but the switched-capacitor converter keeps the microcontroller powered.

Both types of converters draw some current all the time, so you have to decide whether the total power usage, based on the time-averaged total power, is less with two converters or with one switched-inductor converter. An additional feature of using two converters is that two output voltages can be generated, one for the microcontroller and the other for any components that require higher operating voltage.

There are microcontrollers, such as the Atmel ATtiny43U and the Silicon Labs C8051F9xx, with internal boost converters for single-cell operation. Since the boost converter is part of the microcontroller, more efficient operation is possible. For example, the ATtiny43U has a low-current mode of operation where the boost converter regulation is relaxed to minimize power consumption.

As I mentioned in Part 1 of this article series, when using batteries, you will want to monitor the battery voltage to avoid erratic operation when the voltage goes low. With single-cell operation, this is even more important due to the reduced usable voltage margin of the single cell. The Atmel parts contain a brown-out detector that can reset the microcontroller when the supply voltage is too low, but you may not want to wait until that point to go into some kind of safe mode.

BATTERY SELECTION AND RECHARGING

What kind of battery do you use in your project? As I mentioned in Part 1, you might

be limited to off-the-shelf batteries because you want to use coin cells or standard AA, AAA, or 9-V batteries. This lets your user buy batteries almost anywhere. But he has to replace the batteries when they are exhausted.

If you don't want to replace batteries, you need to think about rechargeable batteries. Obviously this implies a recharging circuit. There are different types of rechargeable batteries; a detailed analysis is beyond the scope of this article. For our purposes here, we're interested in recharging the batteries, not in the battery technology.

The simplest recharge circuit is shown in **Figure 2**. This circuit is a modification of the diode-based battery switching circuit of Figure 1. The circuit in Figure 2 includes an additional blocking diode (D3) and a resistor. The resistor charges the battery when the AC adapter is plugged in. This approach is only suitable for batteries such as Nickel-Cadmium (Ni-Cd) or Nickel Metal-Hydride (Ni-MH) that can withstand continuous trickle charging.

The resistor must be chosen to limit the maximum charge current to no more than the continuous charging current for the selected battery. This is typically the capacity in milliamp-hours (mAh) divided by 10, but varies by battery. Also be sure to select diode D3 to handle the charging current; this will probably be significantly higher than the normal operating current of the circuit.

The charging current is the difference between the external DC supply voltage and the battery voltage, divided by the resistance. This can be a problem if you are using, say, a 9-V wall mount DC supply and a user plugs in a 12-V supply instead. The circuit will probably still work, but the battery charging current may be too high.

The advantage of trickle charging is simplicity; the drawback is charge time. A discharged battery can take many hours to recharge at the maximum trickle charge rate; longer if lower current is used. For a device that is normally plugged into the external supply and only occasionally run on batteries, or can be routinely charged overnight, this may be an acceptable trade-off.

Another advantage of trickle charging is that it's suited to charging from a solar cell. This only works if the solar cell can charge the battery faster than the circuit discharges it. But in a circuit intended to last for days or weeks, solar trickle charging is a viable option since the trickle charge current will be higher than the drain current. In effect, the circuit operates from the solar cell when sunlight is available and from the battery the rest of the time.

ABOUT THE AUTHOR

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

SUPERIOR **EMBEDDED** SOLUTIONS



DESIGN YOUR SOLUTION TODAY
CALL 480-837-5200

www.embeddedARM.com

Embedded Products for All Needs



TS-4900 Computer on Module

Industrial High Performance
i.MX6 Module with Wireless
Connectivity and Flash Storage

- 1 GHz Solo or Quad Core Freescale i.MX6 ARM CPU
- 512 MB, 1 GB, or 2 GB DDR3 RAM and 4 GB eMMC Flash Storage
- Wireless 802.11 b/g/n and Bluetooth 4.0 Soldered Module
- 4k LUT FPGA, 1x Gigabit Ethernet, 1x PCI Express Bus
- 1x microSD Socket, 1x SATA II, 1x USB Host, 1x USB OTG
- 70x DIO, 4x I2C, 1x I2S, 2x SPI, 2x CAN
- 40 °C to 85 °C Industrial Temperature Range
- Runs Linux, Android, QNX, Windows
- QT, OpenGL, DirectFB, GNU Tool Kit, and More

Starting at
\$89
Qty 100
\$122
Qty 1



TS-7970. SBC Version of
the TS-4900



TS-7250-V2 Single Board Computer

Extensible PC/104 Embedded
System with Customizable
Features and Industrial Temps

- 800 MHz or 1 GHz Marvell PXA166 ARM CPU
- 512 MB DDR3 RAM and 2 GB SLC eMMC Flash Storage
- PC/104 Connector with FPGA Driven Pins (8k or 17k LUT FPGA)
- 2x 10/100 Ethernet, 1x microSD Socket, 2x USB Host
- 75x DIO, 5x ACD, 3x RS232, 3x TTL UART, 1x RS485, 1x CAN
- 40 °C to 85 °C Industrial Temperature Range
- Preinstalled Debian Linux OS and Utilities

Starting at
\$165
Qty 100
\$199
Qty 1



Available with TS-ENC720 enclosure



TS-TPC-7990 Touch Panel PC

7" High End i.MX6 Mountable
Panel PC with Dev Tools Such
as Debian GNU and QtCreator

- 1 GHz Solo or Quad Core Freescale i.MX6 ARM CPU
- 7 Inch or 10 Inch Touch Panel PC
- Resistive and Capacitive Screens
- Linux, Android, QNX, and Windows
- QtCreator, GTK, DirectFB, GNU Tool Kit, and More
- Runs Yocto, Debian, Ubuntu Distributions

Starting at
\$299
Qty 100
\$342
Qty 1



Enclosed TPCs
Also Available

TRUE CUSTOM SOLUTIONS Accelerated Time to Market

Partner with our 31 years of embedded expertise,
flexible manufacturing, long lifecycle guarantee,
to reduce risk and accelerate time-to-market.



We've never
discontinued a
product in 30
years



Embedded
systems that
are built to
endure



Support every
step of the way
with open
source vision



Unique embedded
solutions add
value for our
customers

www.embeddedARM.com

For faster charging, or for Lithium-ion (Li-Ion) batteries, a more complex circuit is used. Fast charge circuits monitor the battery for full charge and then cut off the charge current to prevent overcharging. The easiest way to implement a fast-charge circuit is to use an IC that is designed for that purpose. Typical examples would be the Linear Technology LT1571 or the Maxim MAX712. There are many different battery charge management ICs to choose from.

Texas Instruments has a good application note on these three battery technologies (Nickel-Cadmium, Nickel Metal-Hydride, and Lithium-ion) and the trade-offs in charging them. You can find it at www.ti.com/lit/an/snva557/snva557.pdf.

ESTIMATING BATTERY LIFE

In most cases, you will want to have some idea how long your battery will last in the circuit. In very low-power circuits or consumer electronics, it may not matter that much. The batteries in the remote control for your television can last months. You may not care whether it is 12 or 18 months. But for a remote data logger application you would want to know how often to schedule battery replacement.

The first component of battery life is the current drawn by the electronics. For battery operation, this will be a combination of the current drawn in sleep mode and in active modes. (These modes are described in Part 1 of this article series.) To build a power profile, you will want to add the current drawn in sleep mode and the current drawn in active modes.

Note that you could have more than one active mode. For example, a remote data logger might take readings once per second, but it might transmit those readings via cell modem once per hour or once per day. Obviously, using the cell modem is going to require significantly more current than just taking readings, in most applications.

In a typical example, say that we have a remote data logger that takes a reading every second, and it takes 100 ms to wake up and capture all the information. One of the sensors we are reading requires that an infrared LED be turned on; maybe we're reading the

opacity of water in a pipe to measure the amount of suspended sediment. So our power usage looks like this:

- Microcontroller current during active state: 10 mA
- LED current: 20 mA
- Microcontroller current during sleep state: 25 μ A

Since we make a measurement once per second, and each measurement takes 100 ms, then the average active mode current draw is $(10 \text{ mA} + 20 \text{ mA}) \times 100 \text{ ms}/1 \text{ s}$, or 3 mA. Running this for an hour consumes a charge of 3 mAh from the battery. The sleep mode consumption is 22.5 μ Ah. The total of active and sleep mode is 3.0225 mAh/hour. Clearly, the sleep mode contributes little to the battery drain, which is why we want the microcontroller to spend as much time as possible in that state.

An alkaline AAA battery has a capacity of about 1,250 mAh, so the battery lifetime will be $1,250 \text{ mAh}/3.0225 \text{ mA} = 413 \text{ h}$. This will vary with the type of battery: alkaline vs. Ni-Cd vs. zinc-carbon, and so on. It also varies with the amount of voltage drop in the battery our circuit can tolerate. Even the quality of the battery will affect it. I got a consumer product once that came with batteries, but the manual said that the included batteries could be expected to have shorter life than purchased batteries. Like those small capacity "starter" ink cartridges that some printer manufacturers include with their printers.

Switching to an alkaline AA battery with about 2,000-mAh capacity gives about 661 h of operation. All the same conditions and caveats apply.

Generally, we want to run a battery-powered microcontroller at the lowest possible clock rate for minimum power. For many microcontrollers, this means a 32-kHz watch-type crystal. If your application is like that theoretical data logger, where the microcontroller is just waking up to take readings for a fixed time, that is usually what you would do. A device like a television remote control, where the microcontroller wakes up in response to button presses by the user, is another example of such an application.

There are some applications where the slowest clock rate isn't the best choice. If the active mode of the microcontroller requires complex calculations that take significant time, then the slower clock rate may not result in the lowest possible average power. But using this same method of calculation will let you determine that, based on the active and idle current used by your microcontroller. Be sure to compare the power calculation



circuitcellar.com/ccmaterials

SOURCE

ATXmega324A Microcontroller
Atmel Corp. | www.atmel.com

for both clock rates at the same supply voltage. If the microcontroller can operate at multiple supply voltages, the datasheet may list the active and idle current for different frequencies and at different supply voltages.

As an example, let's take the same data logger, but say that the calculations required to make all the readings (maybe compensating for the ambient light and temperature) is what makes the sample take 100 ms. Say that if we ran at 2 MHz, the reading would only take 1.6 ms ($100 \text{ ms} \times 32 \text{ kHz}/2 \text{ MHz}$). Say that at 2 MHz, the sleep mode current is 200 μA and the active mode current for the microcontroller is 20 mA. In that case, the average current consumption for the circuit would be about 0.26 mA and the faster clock actually results in lower average power.


This example is a bit extreme because of the relatively high LED current during the sampling interval. But it is not uncharacteristic of the trade-offs involved in comparing the slowest possible crystal to a faster clock rate in applications where the processor has to spend time calculating things. In most applications the slower crystal speed will give the lowest current—but not always.

If you use a microcontroller with an

internal PLL, you may get the best of both worlds. Run the microcontroller on a 32-kHz clock in sleep mode and reprogram the PLL for a faster clock when the microcontroller wakes up. Obviously, this approach limits your microcontroller selection to those with all the low power and other features you need, plus an internal programmable PLL.

When estimating battery life, and if you are using rechargeable batteries, you will want to estimate battery life when the batteries are nearing their end of usable life. For some applications, you might not care, especially if the rechargeable battery can be easily replaced. But in other applications, such as the remote data logger, you want to know how long a weak battery will last. You don't want to have to make an extra trip to replace the data logger because old batteries didn't last long enough.

DESIGN FOR OPERATION

Although this article series can't cover every aspect of battery operation, they should get you started on battery-based design. Taking these things into account will help you avoid some of the potential pitfalls of designing for battery operation. 

When it comes to robotics, the future is now!

Advanced Control Robotics simplifies the theory and best practices of advanced robot technologies, making it ideal reading for beginners and experts alike.

With this book, you'll learn about:

- Communication Technologies
- Control Robotics
- Embedded Technology
- Programming Language
- Visual Debugging... and more

ADVANCED CONTROL ROBOTICS
HANNO SANDER

start task **async**
repeat start task **sync** and wait
task **async**
task **sync**
print text
wait 1000
repeat print text
wait 1000

Get it today at ccwebshop.com

The Best Test Equipment at the Lowest Prices

Scopes

Signal Generators

EMC Test Equipment

• Free Technical Support
• Excellent Customer Service

Saelig.com
unique electronics
585-385-1750

Build a Three-in-One Measurement System

No home electronics lab is complete without a signal generator, logic analyzer, and digital oscilloscope. But why purchase the measurement devices separately, when you can build one system that houses all three? Salvador shows you how.

By Salvador Perdomo (Spain)

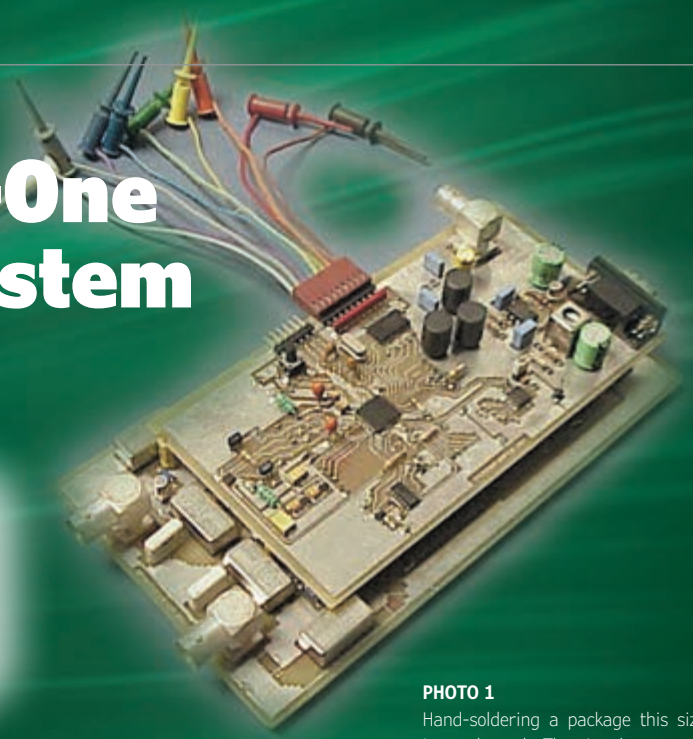


PHOTO 1

Hand-soldering a package this size is tough work. The signal-generator filter has bulky coils. In contrast, the MSP430F149's PQFP64 is tiny.

Editor's Note: This article first appeared in Circuit Cellar 156, 2003.

This article deals with some of the most important measurement instruments needed for a general-purpose electronic laboratory. It should prove to be a useful

resource for electronic enthusiasts and engineers working in their homes, where signal generators, logic analyzers, and digital oscilloscopes are unavailable.

I've built an inexpensive and versatile measurement system that contains a signal generator, logical analyzer, and digital oscilloscope. If you build your own, you'll be able to address many of the problems typically encountered on test benches.

The system is not PC-bus connected. Instead, it's external to the computer, making use of the RS-232 serial port shown in **Figure 1**. Also, it doesn't have a power supply input, so the same serial cable feeds it. Because the computer's serial connection provides limited power, low power consumption is a fundamental requirement.

The low-power goal is achieved with a small number of components—the fewer the better. So, I quickly became interested in the MSP430F149, which is a highly integrated device with low power consumption. Note that everything is integrated except the oscilloscope analog chain (coupling and programmable amplifier), part of the trigger circuit, and the input buffer for the logic analyzer. The microcontroller works with an 8-MHz crystal oscillator.

This application uses the register bank, the entire RAM (2 KB), and nearly all of the peripherals. The peripherals used include the 16-bit TimerA and B, ADC, analog comparator, multiply accumulate, and one USART with modulation capability. Only the second USART is spared.

The system has several main features. You can control and display on the PC by running

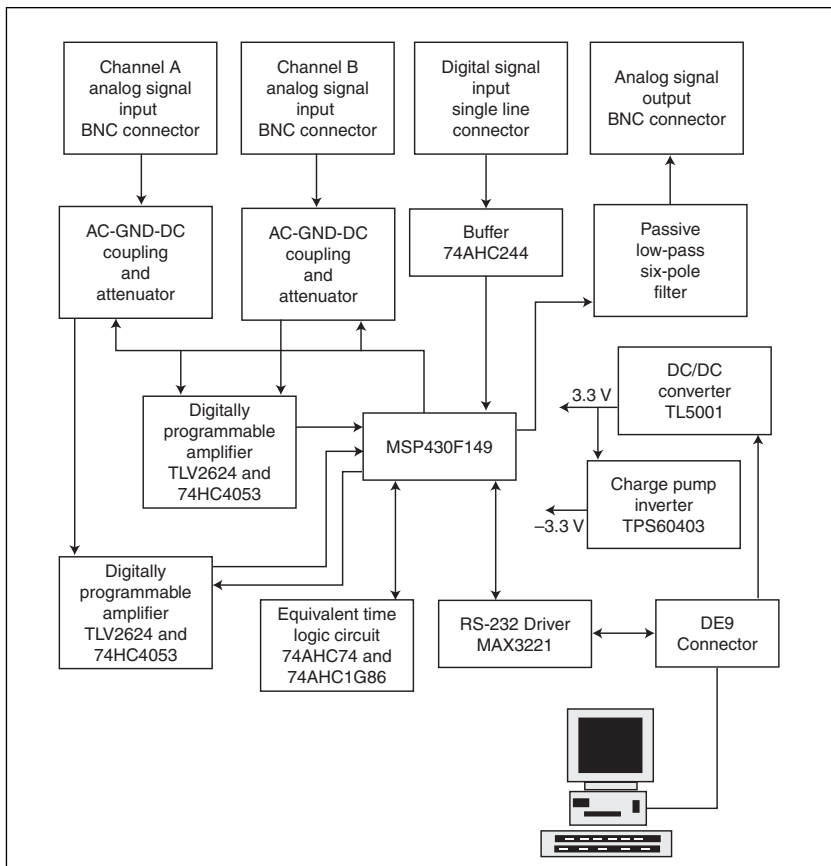


FIGURE 1

It is of interest to have your test benches as clear as possible to search for the faulty part of your design. So, a small measurement system is highly recommended. It's better if it isn't connected to the mains.

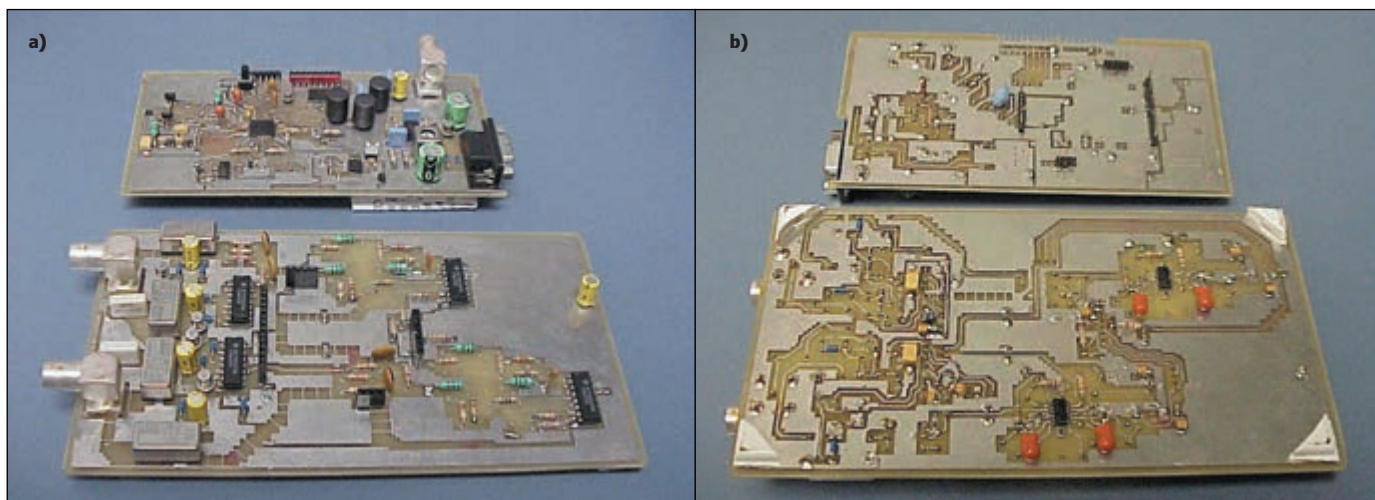


PHOTO 2
a—You can replace the relays in the coupling section and the driver circuit with solid-state relays if you can find ones with low leakage current. **b**—The op-amp’s SMD packages are best viewed from the bottom. The larger board is populated on both sides. Note the importance of the parasitic coupling of the PWM D/A outputs to the input of the amplifiers.

software implemented on LabWindows/CVI. In addition, it has a signal generator based on the direct digital synthesis method and a frequency of up to 6 kHz with 0.3-Hz resolution. The

output voltage reaches a peak of 1.3-V (± 2 dB) fixed amplitude. The signal generator works simultaneously with the oscilloscope and logic analyzer (but not these two).

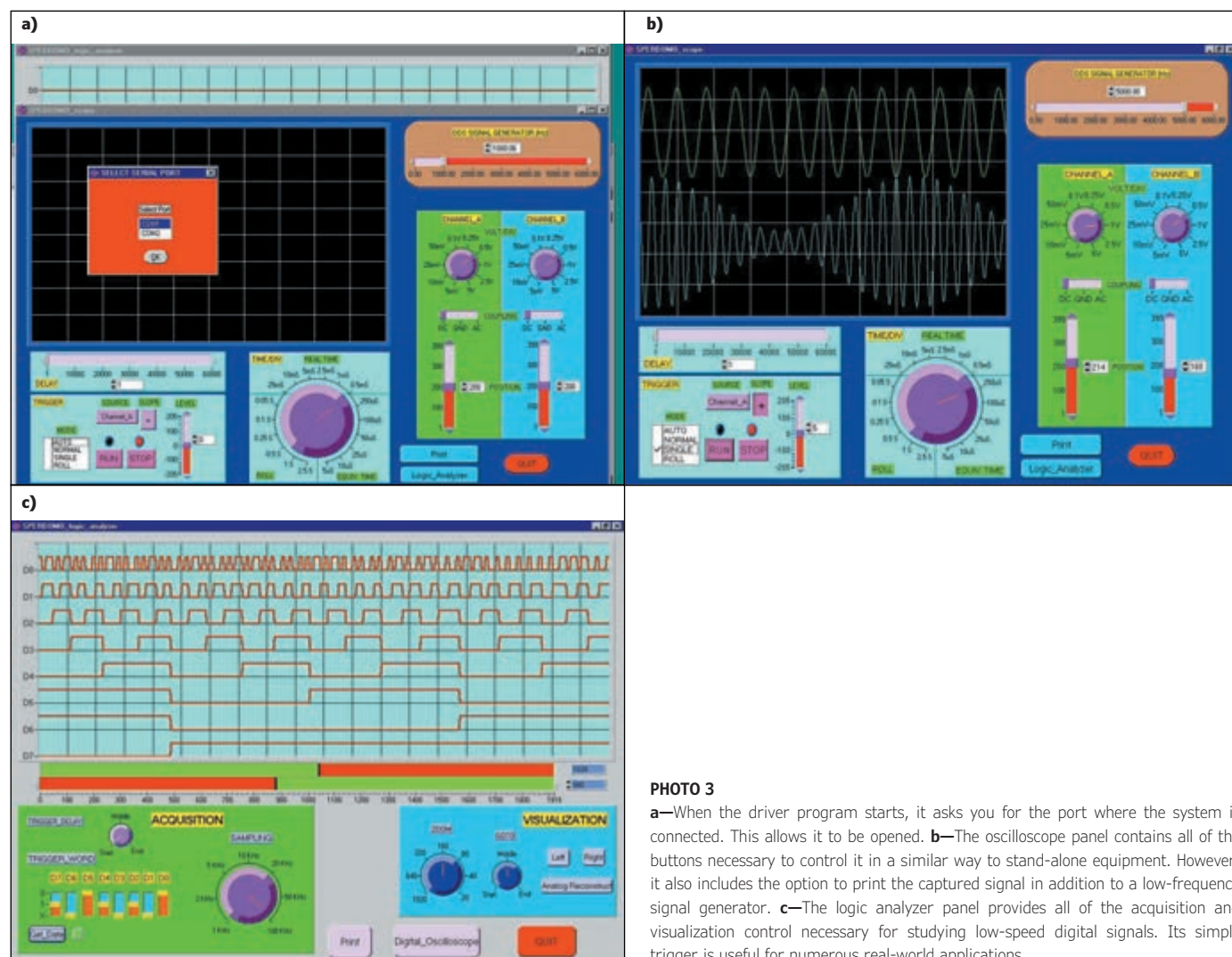


PHOTO 3
a—When the driver program starts, it asks you for the port where the system is connected. This allows it to be opened. **b**—The oscilloscope panel contains all of the buttons necessary to control it in a similar way to stand-alone equipment. However, it also includes the option to print the captured signal in addition to a low-frequency signal generator. **c**—The logic analyzer panel provides all of the acquisition and visualization control necessary for studying low-speed digital signals. Its simple trigger is useful for numerous real-world applications.

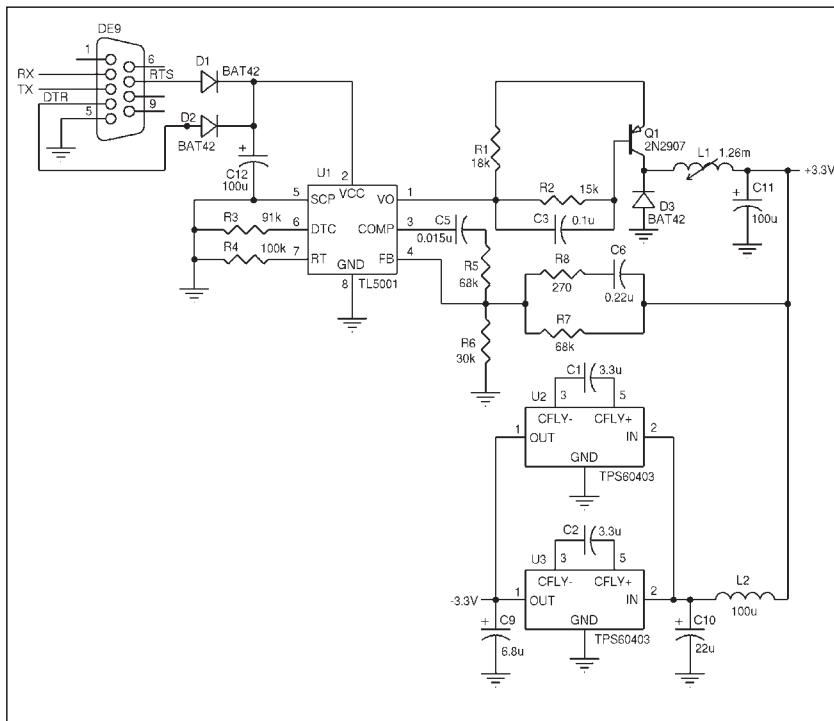


FIGURE 2
Thanks to the selected components' low power, I was able to eliminate the system's independent power. For this project, a DC/DC converter is indispensable.

I included a digital oscilloscope with two channels that have 1-MHz bandwidth, 8 bits of resolution, and 401 words of memory per channel. There are 10 amplitude scales from 5 mV to 5 V per division and 18 timescales from 5 μs to 2.5 s per division. Note that there are four working modes: Auto, Normal, Single, and Roll.

The logic analyzer has eight channels, 1920 words of memory per channel, and sampling from 1 to 100 kS/s. It is trigger-delay selectable between 0, 50, and 100% of memory length.

Looking at **Photo 1**, you see that the system's hardware consists of two separate boards that are attached to each other. Photo 2a shows the tops of the boards, and Photo 2b shows the bottoms. The larger board

contains the oscilloscope analog chain: BNC connectors, relays (and circuit controller) for DC-GND-AC in the coupling section, and the digital programmable attenuator/amplifier. The top board contains the DC/DC converter power supply, charge-pump inverter, serial-communication driver, low-pass filter, trigger (real and equivalent time sampling) circuit, channel-trigger selector, and the microcontroller.

CONNECTION AND POWER

The RTS and DTR DE9 connector pins feed the system. (I didn't use the hardware handshake.) The Rx(2) and Tx(3) pins are used for communication between the microcontroller and PC. This is achieved with the USART0 and a MAX3221 driver.

First, I tested the RTS and DTR pins' I-V curves in order to know how much power was available. The curves are similar, and they resemble a PMOS device connected to a 12-V supply coming down to 10 V at 10 mA of current consumption and approaching 0 V at 14 mA. So, each pin produces a maximum power of approximately 100 mW (200 mW for both of them).

In order to feed the system with 3.3 V from the serial port, I used a 3.3-V output DC/DC converter (see Figure 2). The controller is based on a TL5001. (In the photos, it's near a radio-IF filter can that contains the converter's coil.) Also, a couple of TPS60403 charge-pump voltage inverters were used in parallel to feed the analog portion of the system with ±3.3 V.

Although I could have bought a 1.2-mH unit for the coil construction, I decided to build it with an old 10.7-MHz radio IF filter core. The coil construction took 180 turns of 4-mils (0.1 mm) diameter wire and gave an adjusted range value from 0.5 to 1.26 mH. The coil's DC resistance is 4.1 Ω.

The circuit was measured, and it performed well up to a current consumption of 42 mA (a power of 138 mW), which is enough to feed the entire system. The RTS and DTR pins remain at -12 V when the serial port is closed. When you call the PC driver program, the first panel appears (see Photo 3a). When you select the serial port where the system is connected, the port opens (increasing the RTS and DTR voltage to 12 V), configures a protocol with 115,200 bps, 8N1, and reserves a receiving buffer of 2000 bytes. Also, an interrupt for receiving_buffer_full (802 bytes) is prepared to let the main program know that a datastream has arrived. The LabWindows statements include the following two lines of code:

a)

$$\text{SIN_ROM} = \text{round} \left\{ 199 \times \sin \left[\frac{2\pi(0:255)}{256} \right] + 200 \right\}$$

b)

$$\text{COSIN_ROM} = \text{round} \left\{ 199 \times 2\pi \cos \left[\frac{2\pi(0:255)}{256} \right] \right\}$$

c)

$$\text{SINE} = \text{SIN_ROM}(\text{Phase_H}) + \text{COSIN_ROM}(\text{Phase_H}) \times \text{Phase_L}$$

FIGURE 3
a—The first equation extends the values from 1 to 399 to cover the dynamic range of the DAC. b—Now, the values extend from -1250 to 1250. c—Using the software DDS, you can create a sine function.

```
OpenComConfig (COMx, " ",
115200, 0, 8, 1, 2000, 30)
```

```

add    FREQ,&PHASE           //Phase accumulate from 0 to 65535
mov.b  &(PHASE+1),TABLE_S    //PHASE_H is used to read tables.
rla    TABLE_S              X2 accesses word tables.
mov    #8000h,&RESLO         //To round the MACS to the 16-bit
                                nearest integer.

mov    SIN(TABLE_S),&RESHI
mov    COSIN(TABLE_S),&MACS
mov.b  &PHASE,TABLE_S        //OP2 is a 16-bit SFR, so go throw
                                a register to translate PHASE_L
                                to OP2.

mov    TABLE_S,&OP2
mov    &RESHI,&TBCCR2        //Got new sample. Update TB2_PWM.

SIN    DW    200,205,210,215,220,224,229,234,239,244,248,253
        DW    258,262,267,272,276,281,285,289,294,298,302,306, ...

COSIN  DW    1250, 1250, 1249, 1247, 1244, 1241, 1237, 1232, 1226, 1220
        DW    1213, 1205, 1197, 1187, 1177, 1167, ....
    
```

```

InstallComCallback(COMx, 15,
    802, 0, RxBuffer_full, 0)
    
```

When the microcontroller receives power, it starts from reset and configures all of the necessary peripherals: the comparator, ADC, TimerA and B, USART0, I/O pins, and the interrupts. Afterwards, it stays in a default state, waits for a PC command, and attends to the TimerB signal generator interrupts because it starts generating a 1-kHz sine wave by default.

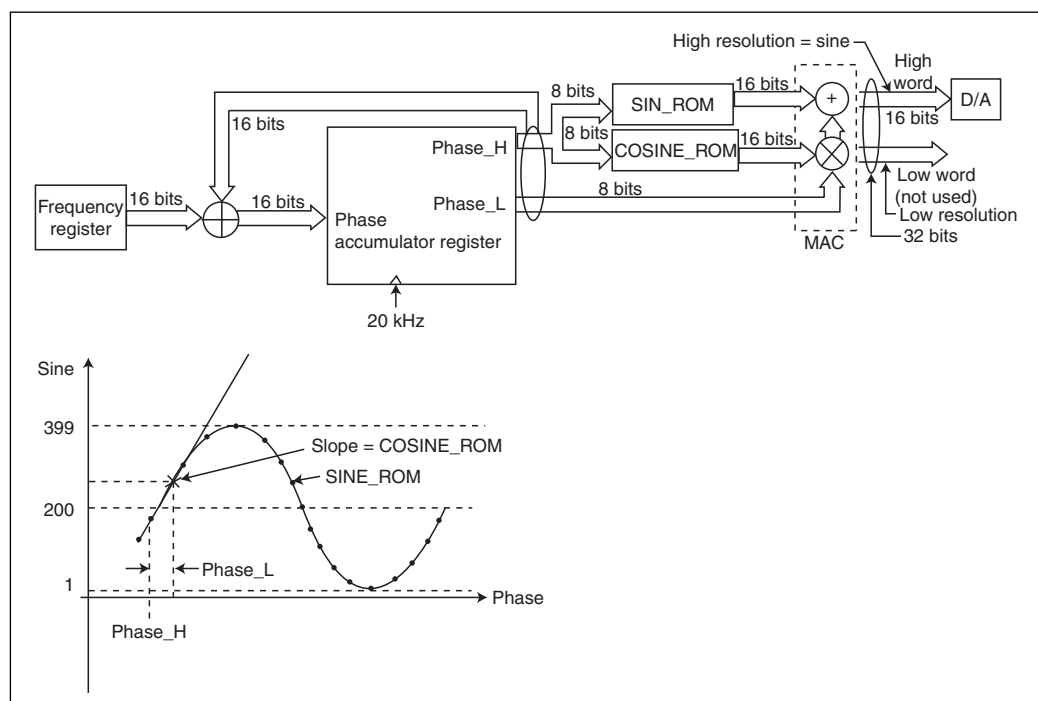
THE SIGNAL GENERATOR

The signal generator is intended to provide the signal $A \times \text{sine}(wt)$, where the amplitude,

A , is fixed near $VCC/2$, and the frequency is programmable with 0.3-Hz (20 kHz/65,536) resolution and up to 6 kHz.

The signal generator's hardware consists of a D/A converter made with TimerB PWM output TB2 (pin 4.2), an external six-pole RLC low-pass passive filter, and a BNC connector. To save more components, the necessary D/A converters are carried out with the PWM of TimerB. A passive filter loads the output transistors and produces distortion in the signal. Active filters are recommended instead. You may download a schematic of the signal generator's hardware from the Circuit Cellar ftp site.

The PWM-D/A converter has a 20-kHz



LISTING 1

Now that you're familiar with the software DDS, you can generate the sine function in Figure 3c.

FIGURE 4

The DDS technique for synthesis is a recent development. Today, the IC-form approach has a 1-GHz sampling frequency with phase accumulators of 32 bits or more. I found the performance to be a modest 20 kHz with a 16-bit phase accumulator.

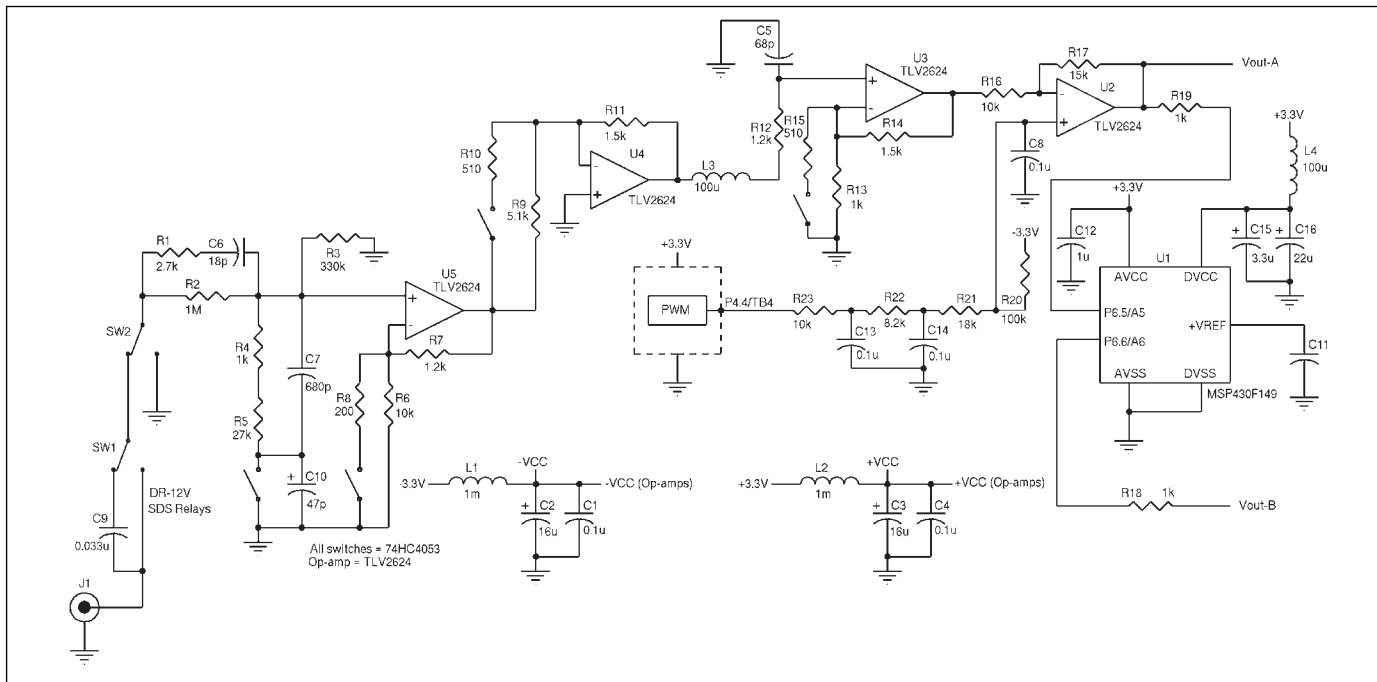


FIGURE 5
The analog conditioning chain must have digital programmable gain capability in order to adjust the voltage range of the input signal to the ADC input voltage range.

sampling frequency. This value is the result of several trade-offs among resolution, time spent attending to its interrupt and peripherals, and the maximum generated frequency. The converter has the following resolution:

$$\frac{8 \text{ MHz}}{20 \text{ kHz}} = 400$$

which is equivalent to 8.64 bits.

As you can see in Photo 1, the filter’s inductors are power chokes. Their series resistance was measured, and it changes with the frequency in the following way:

$$R_s = 27 + 0.9e - 3f + 0.5e - 6f^2$$

This behavior results from the losses in the ferromagnetic core. It was taken into account in the filter design as well as the resistance of the PWM (pin 4.2) output MOS transistors. The filter was designed after measuring the coils with a MATLAB-based program that took into account the coil-loss variations with the frequency. You may download graphs of the frequency response from the Circuit Cellar ftp site. The graphs represent the response to the sampling frequency and the passband details.

The distortion of the PWM as a D/A converter is by far the biggest source of spurious signals, mainly because the output MOS transistors must supply the analog current to the passive filter. So, I recommend using an active filter because it won’t load the PWM, and it will save you from using bulky coils.

The rest of the signal generator is based on a software digital synthesizer (DDS) composed of a 16-bit frequency register, 16-bit phase accumulator register, and two look-up tables (SIN_ROM and COSIN_ROM). Each of these

tables is 256 words long and 16 bits wide. The values are computed in MATLAB using the equation shown in Figure 3a. By using the equation in Figure 3b, the values extend from -1250 to 1250. This equation represents the time derivative (slope) of the SIN_ROM table multiplied by 256. Note that 0:255 (i.e., 0 through 255) is simply a way to create an array of numbers in MATLAB.

Figure 4 depicts a function diagram of the software DDS. It uses the microcontroller’s multiply-accumulate capability to generate the sine function in the equation shown in Figure 3c, which is easily carried out with the code in Listing 1.

THE DIGITAL OSCILLOSCOPE

The oscilloscope panel incorporating the signal generator’s controls is shown in Photo 3b. The brown box in the upper-right corner is the DDS control, which controls the generated frequency. Everything else pertains to the visualization and control of the digital oscilloscope. As you can see, the panel is visualizing two signals: a 5-kHz sine DDS generated in channel_A (green trace), and a 10-kHz AM modulated signal generated by a commercial generator in channel_B (blue trace). The digital oscilloscope’s hardware consists of a configurable analog chain that drives the ADC, RAM, a trigger circuit, and a display.

Each of the digital oscilloscope’s channels has a configurable coupling stage (DC, GND, AC) made with two low-power (high-resistance, 1400-W) DR-12V monostable relays from SDS-Relais—a company that’s now called Matsushita Electric Works UK. Its pick-up

voltage is 9.6 V, and its dropout voltage is 1.2 V. So, the circuit that drives its coil is a little tricky when you're trying to engage it with only ± 3.3 V. To produce a transient response bigger than the available power supply, you must rely on reactive components (i.e., coils, capacitors, or both). In this system, the charge stored in a capacitor is used as a floating battery that's added to the fixed power supply. You may download a diagram of the circuit from the Circuit Cellar ftp site.

When the microcontroller pin changes from a high level to a low level, a pulse that's long enough and close enough to 12 V is applied to the coil to pick it up. Afterwards, it continues applying approximately 3.3 V, which keeps it engaged (neglecting the voltage drop in the Schottky diode and the transistor saturation voltage).

Nearby, there is a digitally controlled attenuator and amplifier around the low-power, high-bandwidth CMOS op-amp (TLV2624) and 74HC4053 multiplexer. Of the 16 possible switch combinations, only 10 are used to obtain 10 different gains (from 30 to 0.03) corresponding to 10 different input ranges (i.e., oscilloscope sensitivity from 5 mV to 5 V per division).

The bandwidth achieved is always better than 1 MHz. The signal path that runs from the BNC connector to the ADC input for one channel is shown in Figure 5. As I calculated its values, I took into account the pin's capacitance, the op-amp frequency response, and the 74HC4053 switch's on resistance (approximately 70 Ω). The ADC had 12 bits of resolution, but I used only the eight higher bits that were sent to the PC. The internal 1.5-V reference voltage fixes the input range.

In order to control the position of the channel A and B traces in the screen and the offset of the amplifiers, two PWM DACs and a passive

low-pass filter—which are based on TimerB PWM outputs TB4 (pin P4.4) and TB3 (P4.3)—are provided (see Figure 5). The amplifiers' outputs go to the micro's ADC inputs—A5 (pin 6.5) and A6 (P6.6)—and to a trigger-channel selector made with a 74LVC2G66, which feeds the MSP430F149 analog comparator connected with positive feedback (Smith trigger). This is achieved in such a way that it has 30-mV hysteresis, or 2% of 1.5 V (the full range of the ADC).

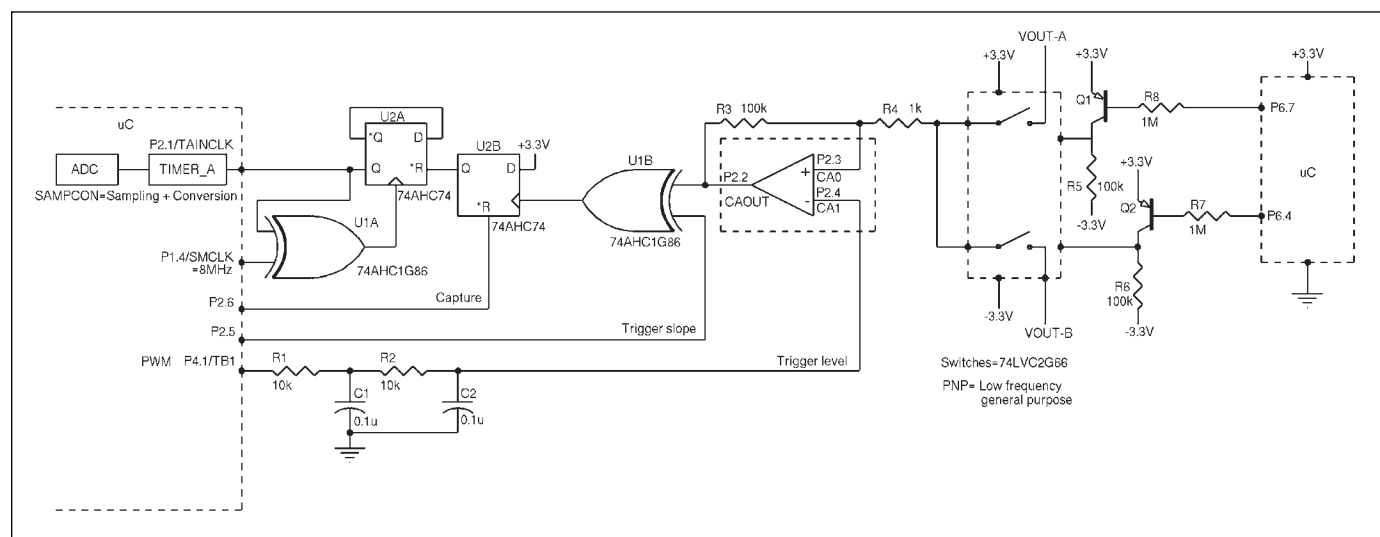
The comparator also receives the output of another PWM DAC and passive low-pass filter based on the TimerB PWM output TB1 (pin P4.1) that establishes the oscilloscope trigger level. After the comparator, the 74AHC1G86 exclusive OR gate is used to select the trigger slope.

Figure 6 depicts this portion of the hardware with the rest of the trigger circuit, which makes possible the sequential equivalent time-sampling technique. It's composed by one 74AHC74 (a couple of D flip-flops) and the 74AHC1G86 exclusive OR gate.

The trigger circuit and TimerA collaborate in order to make the oscilloscope work with this technique. It happens as soon as you select the time bases from 5 to 250 μ s per division, and it is transparent. In this way, the oscilloscope bandwidth is only limited by the analog bandwidth and comparator precision, and not by the ADC maximum frequency conversion (Nyquist criteria).

The only requirement to function with this technique is that the input signal must be periodic during the acquisition. For instance, given the faster time base of 5 μ s per division, it is necessary to capture 401 samples (400 intervals of 125 ns) to complete 50 μ s of the signal (i.e., 5 μ s per division \times 10 divisions). But the MSP430F149 ADC's maximum sampling frequency is limited to 200 kS/s because of the

FIGURE 6
The trigger is probably the key section in an analog or digital oscilloscope. To start the capture, it must provide a clean and precise point in the signal.



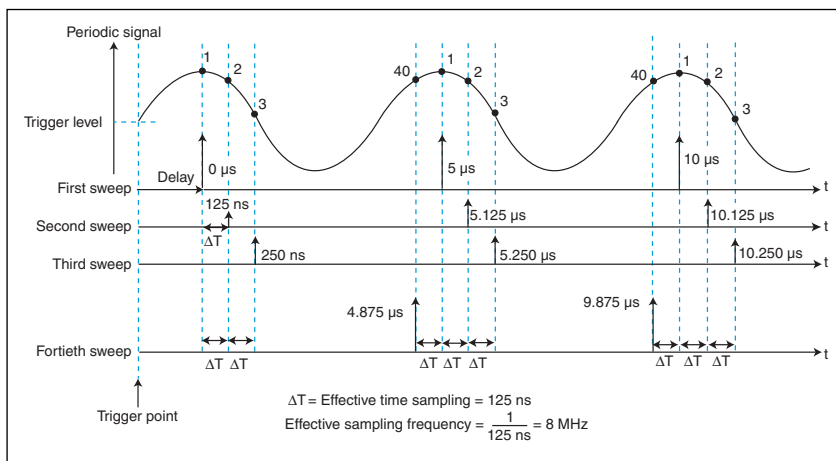


FIGURE 7

If the ADC has a limited conversion speed and its analog bandwidth is higher than the Nyquist criteria enforces, some kind of equivalent time sampling can be applied. This figure explains one of the techniques—sequential time sampling.

5- μs conversion time (per channel).

The signal is periodic, so it is possible to make successive trigger-capture cycles or sweeps capturing only a portion of the signal on each sweep. They would have to be incrementally delayed with respect to the trigger point, as illustrated in Figure 7. TimerA is in charge of the delay. For the 5- μs -per-square time base, the ADC is programmed to acquire 11 successive samples (5 μs apart) on each sweep. Forty successive sweeps, which are incrementally delayed 125 ns, are performed to total 440 samples. Note that only the first 401 are sent to the PC.

Real time covers the time bases from 500 μs to 2.5 s per division, and it implies only one sweep capturing 401 samples per channel in both channels simultaneously (Nyquist criteria applies). In practice, there is a delay of one

sample between the two channels because there is only one sample-hold (actually the channels are converted interlaced), but it isn't noticeable.

Equivalent time and real time (depending on the time base that's selected) are the ways the hardware works when you select Auto mode, Normal mode, or Single mode from the PC's oscilloscope mode control. Now, let's take a look at each one.

In Auto mode, an automatic trigger will occur if there is not a trigger within a fixed 0.2-s interval. This fixed time is commanded by the PC if it does not receive the samples it is waiting for from the previous Acquire command. To produce the automatic trigger, the microcontroller changes pin P2.5 (trigger slope) twice in order to assure that the first flip-flop in Figure 6 is set. When the 802 samples arrive (401 + 401), another Acquire command is released.

In Normal mode, a trigger event is necessary to acquire data. Only after all of the 802 samples have arrived does the PC release another Acquire command.

Single mode is similar to Normal mode, but there is one major difference. Basically, after all of the samples have arrived, the PC stops waiting for another user command.

Roll mode is only selected from 0.05 to 2.5 s per division. It is different from the other modes because it doesn't use a trigger event to start acquisition. Instead, it is continuous, and the microcontroller doesn't wait to acquire 401/channel samples before they are sent to the PC.

In Roll mode, a smaller number of samples (depending on the time base) are acquired and sent. For instance, at 2.5 s per division, only four samples per channel are acquired before they are sent to the PC. When the PC receives them, the old samples are shifted to make room for the new ones and are shown on the screen. This produces an effect of picture displacement known as the roll effect. Of course, because the datastream length changes with respect to the other modes, the LabWindows receiving_buffer_full interrupt has to be adapted correspondingly (function InstallComCallback).

THE LOGIC ANALYZER

The logic analyzer panel is shown in Photo 3c. There is no direct access from this panel to the signal generator, but it keeps generating a signal with the frequency previously fixed in the oscilloscope panel. Photo 3c shows only the central 160 samples per channel (zoom applies) of the 1920 samples per channel captured.

The hardware is easily built with a 74AHC244 buffer and a pull-down array of eight 1-MW resistances. The 74AHC244 buffer makes the



circuitcellar.com/ccmaterials

PROJECT FILES

To download the code and additional files, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2003/156.

RESOURCES

Texas Instruments, "MSP430 Bug list," www.ti.com/sc/cgi-bin/buglist.cgi.

-----"MSP430x1xx Family User's Guide," SLAU049A, 2001.

SOURCES

MATLAB
MathWorks, Inc. | www.mathworks.com

Monostable relays
Matsushita Electric Works UK | www.matsushita.co.uk

LabWindows/CVI
National Instruments Corp. | www.ni.com

74AHC244 Buffer
Philips Semiconductors | www.semiconductors.philips.com

MSP430F149 Microcontroller
Texas Instruments, Inc. | www.ti.com

system 5-V, TTL-compatible, and is connected to port 5 on the microcontroller. The rest of the logic analyzer (i.e., the sampling frequency, triggering, and trigger delay) is software-based. Also note that it's 8 bits wide with a 1920-KB memory depth and an acquisition frequency range from 1 to 100 kS/s.

The trigger delay is user-selectable, which enables pre-triggering, middle triggering, and post-triggering. Because of the asynchronous sampling of the data, the visualization is only available as a timing diagram.

Concerning the triggering, when you activate the Get_Data control, the PC's main program extracts two bytes—ID (ID7 through ID0) and IDE (IDE7 through IDE0)—from the states of the trinary switches, D7 through D0. D7 through D0 define the trigger word with three possible values (0, 1, and x) bit by bit. After the microcontroller receives the IDE and ID bytes from the PC, the sampled input data is masked (masked_DATA = DATA logical AND with IDE), making zero the don't care bits (option x) selected by the user. This masked input data is compared to the ID byte (masked_DATA is CMP with ID). A match validates the trigger. If there is a match, the trigger delay is counted and the acquisition is completed. Afterwards, the data is sent to the PC. You may download a diagram of this process from the Circuit Cellar ftp site. Note that the process is used for the trigger word depicted in Photo 3c.

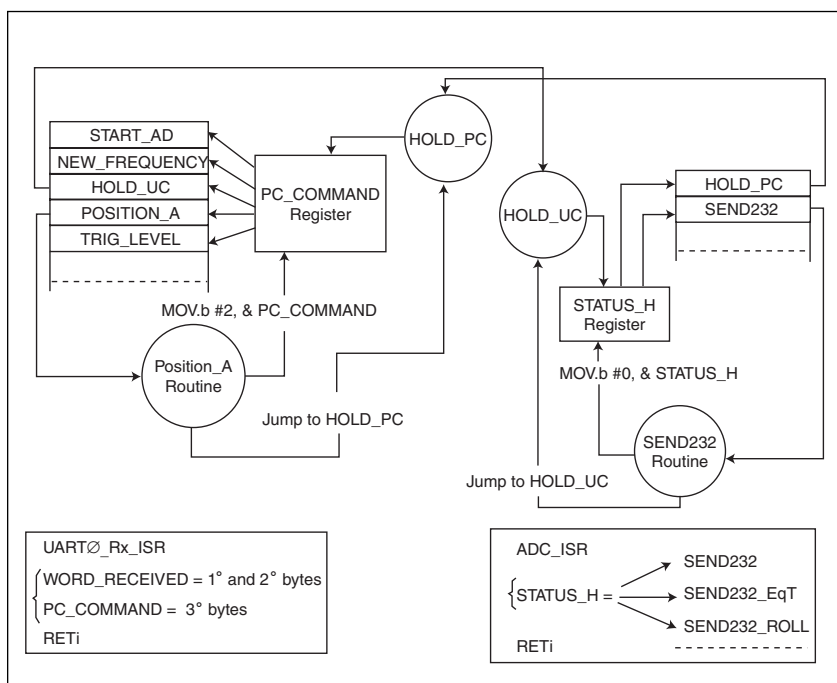
The simplest event that can trigger a logic analyzer is the coincidence of a data with a word you have selected. This coincidence must be bit by bit. To define the trigger word, some switches are provided so you can set each bit to 0, 1, and x (don't care).

THE PC-MICRO COMBO

By default, the microcontroller attends to the TimerB signal-generator interrupt every 50 μ s (20-kHz sampling frequency), and the subroutine lasts 7.875 μ s including the latency time. So, the remainder of the time is available for the received PC commands or the interrupts and commands released by other peripherals.

The PC commands are composed of 3 bytes: two data bytes and one command byte. When the USART0 received data interrupt is attended, the number of data bytes received are counted in order to correctly deposit them in three registers, including word_received (16 bits) and PC_command (8 bits). Back in the main program, the PC_command register is used to make a table-based indexed branch to the routine that serves the intended command.

The reason for accompanying the byte command with two data bytes is self-explanatory. To change the frequency of the signal generator, it is necessary to load a new 16-bit value in the frequency register in Figure



4, and to change the trigger level or change the channel trace position. Another example is that the acquisition command, START_ADQ, is accompanied by a number indicating the time division to program TimerA in order to fix the ADC acquisition frequency. Another number indicates how the samples have to be dealt with (e.g., equivalent time, real time, or roll). For other commands (e.g., TRIG_SLOPE and TRIG_SOURCE), they're unnecessary and filled with dummy data.

Otherwise, there is another microcontroller register called STATUS_H, which keeps track of the peripherals' jobs. For instance, when the ADC routine has loaded 401 samples per channel in the RAM memory, it deposits a peripheral command such as SEND232 in this register. Thus, when back in the main program, the STATUS_H register is used to make a table-based indexed branch to the routine that serves the intended command. (In this example, the acquired data is sent to the computer.)

A helpful flow chart is depicted in Figure 8. You may download several lines of code representing these ideas from the Circuit Cellar ftp site.


Keep the following advice in mind when you're changing from the oscilloscope to the logic analyzer (and vice versa): You can send a command to make the microcontroller return to a default state (waiting for a PC command and attending the TimerB signal-generator interrupts), and stop sending data if it is doing it. Simultaneously, the PC data queue must be cleared. 

FIGURE 8

Two different registers control the microcontroller's program flow. The PC writes one (PC_COMMAND), and the other (STATUS_H) is written by the peripheral when their interrupts are attended.

ABOUT THE AUTHOR

Salvador Perdomo received a degree in Telecommunications Engineering from the Universidad Politécnica de Madrid, Spain. He has lectured at the Universidad de Las Palmas de Gran Canaria, Spain. His interests include analog and digital electronics. You may reach him at sperdomo@det.ulpgc.es.

THE CONSUMMATE ENGINEER

Transformers 101 (Part 2)

Transformer Design

COLUMNS

In the first part of this article series, George presented the transformer and its essential characteristics. In this article, he covers the basics of transformer design.

By George Novacek (Canada)

Last month, I covered the fundamental theory behind transformers. Now let's continue by considering the basic aspects of transformer design.

In low-power supplies—such as those for laptop computers, radios, battery chargers, and so forth—built-in transformers as we've known them are a dying breed. Manufacturers of electrical appliances have been, whenever possible, replacing their internal transformers with plug-in “wall wart” adapters. Those are either transformers with AC output (see **Figure 1a**) or DC supplies containing a rectifier and a capacitor (see Figures 1b–c).

One reason for this trend has been the avoidance of the costly safety certification of every equipment model, required in every country where the product was to be sold. Low-volume, high-mix product and slightly different regulations among countries cause the cost of certification to be a major issue. A mass-produced plug-in wall wart supply, already certified, is the answer. The traditional DC wall wart supplies (see Figures 1b–d), notorious for their poor power factor, are being replaced by high-frequency switching regulators with many benefits. Their design is not the subject of this series. Because their transformers operate at high frequencies, they are smaller, lighter, and less expensive.

A wide range of input voltage, output voltage regulation, and excellent power factor provide additional benefits. The power factor correction (PFC) is now mandatory in many countries.

One disadvantage of the switching wall wart supply as compared with a traditional transformer type is its inherently lower reliability, due to the number and type of components in it. However, I have found those supplies to be of excellent quality and reliability, while I have seen far too many “classic” wall wart supplies fail due to their cheap design or shoddy workmanship.

At one time, engineers designed and built their own transformers. Today, there are so many off-the-shelf options that the need for “rolling your own” has virtually disappeared. When you need a transformer not readily available, you should have it designed. It takes a lifelong experience to become a competent transformer designer, but there are many expert companies to help you.

With that said, it is nevertheless a good idea for an engineer to be familiar with transformer design basics. At the very least, you'll be knowledgeable enough to interface with your supplier and will appreciate the potential design constraints. What transformer requirements do you need to specify?



REQUIREMENTS

Apart from the mechanical issues—such as the size, weight, mounting arrangement, and environmental conditions, including operating temperature range, vibration, and so forth, which are generally up to the mechanical designers to address—the electrical engineer's responsibility is to define the transformer's electrical characteristics. First, you need to know the primary voltage and frequency. Then, you must know the secondary windings' requirements. How many secondary windings? What are their voltages and maximum currents? What is the required regulation (i.e., the secondary voltage fluctuations caused by a varying load)? Any taps? Windings' insulation voltages. Primary to secondary insulation. Maximum parasitic capacitive coupling. Are there any shielding requirements to contain the electromagnetic field within the device?

All these requirements affect how the coils are designed, the type of the permeable core used, additional magnetic shielding, and so forth.

The secondary windings are expected to deliver power: $P = V_2 \times I_2 + \dots + V_n \times I_n$. P is the total maximum power to be drawn from the secondary windings. The input power to the primary winding P_T is then: $P_T = P/\eta$, where η stands for the transformer efficiency. This is always less than 100% due to the magnetization current as we saw in the first part of this article series, parasitic capacitance, losses caused by the ohmic resistance of the windings, and so on. Having established the input power P_T and the operating frequency, an appropriate core can be selected. Here's where experience becomes crucial. For one, magnetic induction B needs to stay within the optimum range. The core size is also affected by the expected size of the bobbins to hold the windings, high voltage and electrostatic insulation, wire size to carry the required current, etc. Core manufacturers' data sheets are rarely exhaustive enough for a beginner to make the right choice.

Once the core has been selected, its characteristics determine the required number of turns per one volt. Then the turns for each winding based on their required voltages can be calculated as: $n = k \times V \times V_v$, where n is the number of turns. V is the desired voltage in volts. V_v is the number of turns per volt. k is a constant, which is to compensate for the secondary losses caused by the magnetic flux dispersion and ohmic losses at the maximum rated output power. For the primary winding $k = 1$. Therefore, the primary winding is not changed to avoid modification of the established magnetic flux in the core. Once again, transformer

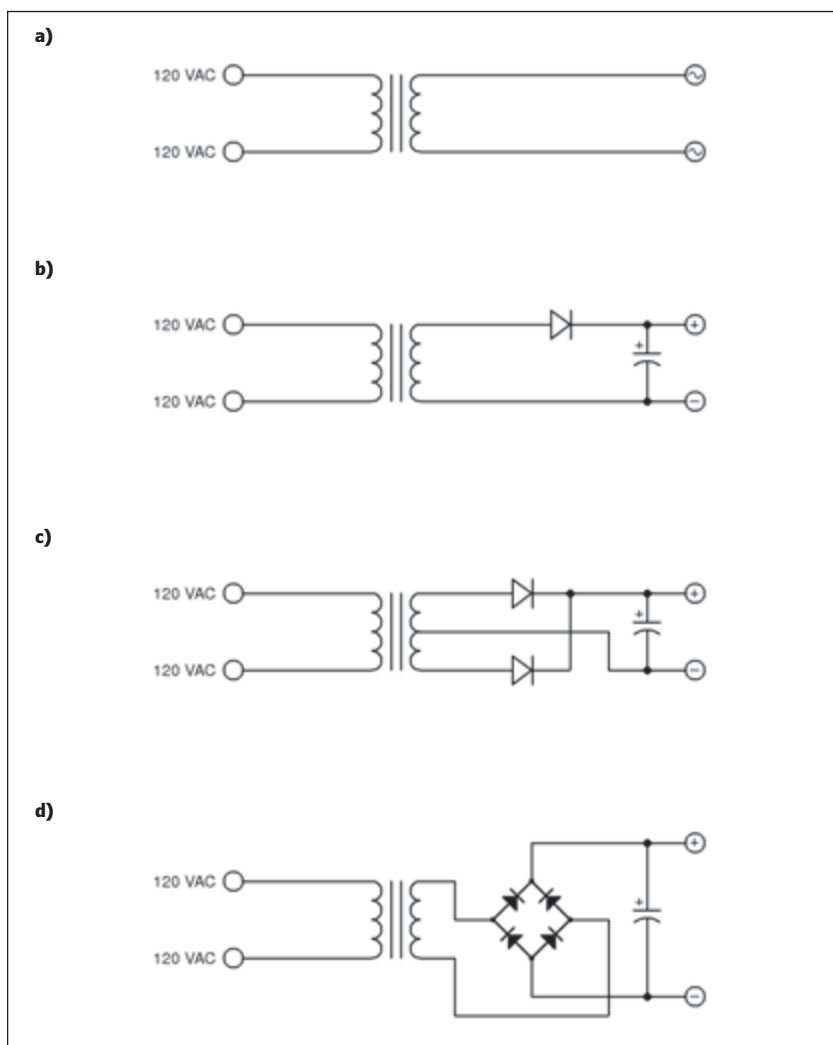


FIGURE 1

Common wall wart power supplies. While the first (a) is still very much in use, the other power supplies (b–d) are gradually disappearing.

specialists have experience, historical data and establishing k is usually not much of a problem for them. For a transformer design novice, establishing k means a lengthy process of trial and error.

The wire sizes are based on the windings' currents and the current density s , which in turn depends on the operating temperature and cooling effects of the core. Old, conservative design tables recommended s to be in the range of 2 to 3 A, sometimes 4 A per 1 mm² of the wire cross-section. Modern designs—usually to cut cost, weight, and size—sometimes exceed this by more than an order of magnitude, relying on the better quality core materials, reliability sacrifice caused by the higher operating temperature and acceptance of higher ohmic losses. These are sometimes intentional to limit the

maximum output current, such as in cheap battery chargers or due to expectations that the voltage fluctuations caused by a varying load will be compensated for by a following voltage regulator.

Most signal and low-power transformers in electronics work as single-phase devices, although there are special applications in power distribution where multiple phase primary or secondary or both windings are needed. Once I participated in design of a relatively high-power (60-kVA) control system supplied from a three-phase, 200-V/400-Hz generator. Building a transformer with a nine-phase secondary, followed by rectifiers with capacitive filters, improved the power factor such that reduced the power quality requirement was satisfied without a PFC. Multiple phases were a smaller, lighter, and less expensive solution.

The same transformer design principles apply throughout the power and frequency spectrum. The major difference is made by the required core. Transformers working in the audio spectrum, for instance, require flat frequency response characteristics from typically 20 Hz to 20 kHz and a minimum


harmonic distortion. Except for low, usually a single frequency, such as for 60-, 50-, or 400-Hz power supplies, transformers need specialized cores designed to optimally handle the given frequency spectrum of the signals and their waveforms.

SPECIALIZED DEVICES

There are also many specialized transformer-based devices, such as pulse transformers or transformers for switching power supply applications, whose input is not sinusoidal. Some must handle a DC bias. There are also magnetic amplifiers, ferroresonant voltage regulators, transformers using magnetic flux nonlinearity, including saturation to perform special functions too exotic to address in this short article series. Prior to the invention of the vacuum tube, transformers were the only component capable to modify signal levels.

It should be remembered that other than supplying desired voltages, transformers can match different impedances by the ratio of their turns. This, coupled with their primary to secondary insulation and suppression of common mode interference, is used in distribution of numerous data communication systems (e.g., Ethernet and MIL-STD-1553).

As the frequency increases, the magnetic cores become smaller and somewhere around 100 MHz can be eliminated. The obvious advantage of the air core, provided the frequency is high enough and the coils can be reasonably small, is their linear B/H relationship and, thus, no need to worry about its nonlinearity. At high-megahertz frequencies, transformers can be created by transmission lines and once we get into the gigahertz, with waveguides. But that, while interesting, is a different topic for another time.

For completeness I should mention "electronic transformers." These are similar to the DC-producing switching wall warts. Light, reasonably efficient and less costly to manufacture than magnetic transformers, their output is their internal switching frequency amplitude modulated by double the input AC (50 or 60 Hz) frequency. Popular with some appliance manufacturers, they have limitations. More about them next month in the final part of the series when we'll also look at some less common transformer types. 



ABOUT THE AUTHOR

George Novacek 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. Contact him at gnovacek@nexicom.net with "Circuit Cellar" in the subject line.

RESOURCES

R. Lee, L. Wilson, and C. E. Carter, *Electronic Transformers and Circuits*, Wiley Interscience, 1988.

R. Morrison, *Grounding and Shielding Techniques*, Wiley-IEEE Press, 2007.

G. Novacek, "Inductors 101," *Circuit Cellar* 292, 2014.

R. Schmitt, *Electromagnetics Explained*, Newness, 2002.



circuitcellar.com/ccmaterials



PCBCART

High-Value PCB Fabrication & Assembly from Industry's leading supplier



Save **15% Off** Your First Order use code **PCBCC**

➔ www.pcbcarts.com

We offer:

- ✓ Professional quality products
- ✓ Quick turnaround times
- ✓ Affordable competitive pricing
- ✓ Superior customer service

- ✓ PCB fabrication up to 32 layers
- ✓ Min. tracing/spacing to 3mil/3mil
- ✓ Min. microvias to 0.1mm
- ✓ Enhanced custom PCB features
- ✓ Special PCBs-Aluminum, flex and HDI
- ✓ Prototype to mass production
- ✓ Full turnkey PCB assembly



sales@pcbcarts.com

PROGRAMMABLE LOGIC IN PRACTICE

Rapid FPGA Design in Python Using MyHDL



COLUMNS

MyHDL is an alternate hardware description language (HDL) that allows you to leverage the power of Python for designing, simulating, and verifying FPGA designs. Colin explains how MyHDL works and describes a FIR filter he created with C/C++ HLS in his February 2014 article to compare the toolchain flow.

By Colin O'Flynn (Canada)

Back in February 2014, I took you through the use of C/C++ High Level Synthesis (HLS) as a design language for a FPGA. This article is designed to introduce you to another option for a design language, this time using Python. Once again I'll demonstrate that directly writing Verilog or VHDL is not always the most efficient option.

I'm going to follow the Finite Impulse Response (FIR) filter example from my February 2014 column, which allows you to directly compare the design process. One of the major advantages of using MyHDL compared to C/C++ HLS is that you can pull upon a huge library of existing Python modules to help generate and validate your design.

In the C/C++ HLS example, I used external tools to generate the FIR coefficients. In the MyHDL example, they are generated automatically from my filter specifications. This makes it easy to validate the fixed-point implementation, and compare the filter results to the "ideal" filter result. I'll get into more details later, but before that I want to present an overview of MyHDL.

I should also mention this column owes a great debt to Christopher Felton, who's presentation at DesignWest 2013 on MyHDL is what originally turned me on to the use of MyHDL. I've based the FIR filter example in this column on some of his examples. (For more of his examples, refer to www.fpgarelated.com/blogs-1/nf/Christopher_Felton.php.) I've also linked to his work from ProgrammableLogicInPractice.com, which includes a few other sites besides FPGArelated.com.

INTRODUCING MyHDL

Even if you haven't heard about MyHDL before, it's been in development for some time. It was created by Jan Decaluwe, and released to the world in September 2003. MyHDL allows you to use Python as a hardware description language (HDL). Like other high-level synthesis tools, you must remember it is not designed to convert arbitrary software code into FPGA modules. It won't make an FPGA designer out of a Python programmer, but might make a FPGA designer want to pick up Python for improved productivity.

If you are familiar with Python, you will know that it doesn't natively support all the features required in a HDL. But with a handful of extensions, we can emulate the required features such as ports, signals, and concurrent blocks. For synthesis MyHDL operates at the same Register Transfer Level (RTL) as Verilog or VHDL. This makes it easy to automatically convert from MyHDL to Verilog or VHDL. The resulting Verilog or VHDL files can either be synthesized directly by your FPGA toolchain, or integrated into your existing project (which will again be synthesized by your FPGA toolchain).

Let's jump right into a simple example.

Listing 1 shows a simple implementation of a counter with programmable maximum. **Listing 2** shows the resulting Verilog code. The `ctr_hdl()` block is the main module. One of the first things to note is the module follows some Python-centric themes. For example, there is no explicit type (such as integer bit-width) in the module definition. Instead, the module is able to pull attributes such as the input/output port widths directly from the objects themselves.

The combinational logic (`@always_comb`) and sequential logic (`@always_seq`) blocks will be familiar to any FPGA designer. Like with Verilog or VHDL, a process sensitivity list can be used to determine when the blocks run. You will start to notice the simple use of class attributes, such as the rising edge being defined as part of the `Signal()` class from MyHDL. As well when dealing with the assignment of the future value of the signal once this block executes, we use the `.next` attribute instead of requiring a special operator (such as `<=` in Verilog).

This simple example also takes advantage of the use of the `ResetSignal()` object type. This special signal makes working with resets easier. Notice I never define the reset behavior in my MyHDL `@always_seq` block. Instead the reset signal will automatically reset any used signals to their "default" state (which was declared when I defined those signals). This helps make the code clearer. Often we don't need to see all the reset logic, but still want signals to start at a known value.

Of course, MyHDL doesn't force its reset handling down your throat. Another form of the sequential block allows you to explicitly define the reset behavior. This allows you to reset signals to other values or perform additional actions within the reset block.

So far, I've concentrated mostly on the synthesizable aspects of MyHDL. But much of the "more interesting" aspects of MyHDL are the ability to use it for both simulation and verification of your hardware cores. Whereas Verilog or VHDL have somewhat limited I/O facilities and external libraries, Python has

almost limitless potential when it comes to I/O facilities and external libraries.

In fact, MyHDL can even be used in combination with a Verilog simulator. This means you are not simulating the MyHDL code, but actually simulating the Verilog code generated by MyHDL. The advantage is that by using MyHDL (and Python), you are able to perform complex verification tasks with ease, while still validating your Verilog implementation.

MyHDL also makes problems such as conditional instantiation (selecting which version of a core to use) trivial. MyHDL passes instances of the HDL object, and doesn't

```
from myhdl import *

def ctr_hdl(clk,reset,prog_max,cnt):
    #Define local signal with sizes based on port
    intcnt = Signal(intbv(0,min=cnt.min, max=cnt.max))

    #Example combinational block
    @always_comb
    def copy_out():
        cnt.next = intcnt

    #Example sequential block - reset code generated
    #automatically
    @always_seq(clk.posedge, reset=reset)
    def cnt_main():
        if cnt < prog_max:
            intcnt.next = (intcnt + 1)
        else:
            intcnt.next = 0

    return instances()

##Example of instantiating module, here used just
##for Verilog conversion

#Simple boolean signal
clk = Signal(False)

#Reset signal gets special treatment, makes it easier
#to change reset parameters around
reset = ResetSignal(0, active=1, async=True)

#bit-vector types, specify default value along with min/max
prog_max = Signal(intbv(0,min=0, max=4000))
cnt = Signal(intbv(0,min=0, max=4000))

toVerilog(ctr_hdl, clk, reset, prog_max, cnt)
```

LISTING 1

A simple counter implemented in MyHDL. This code is sufficient to describe the counter and convert it to Verilog, the resulting Verilog being shown in Listing 2.


```

module cntr_hdl (
    clk,
    reset,
    prog_max,
    cnt
);

input clk;
input reset;
input [11:0] prog_max;
output [11:0] cnt;
wire [11:0] cnt;

reg [11:0] intcnt;

always @(posedge clk, posedge reset) begin:
    CNTR_HDL_CNT_MAIN
        if (reset == 1) begin
            intcnt <= 0;
        end
        else begin
            if ((cnt < prog_max)) begin
                intcnt <= (intcnt + 1);
            end
            else begin
                intcnt <= 0;
            end
        end
    end
end

assign cnt = intcnt;

endmodule

```

LISTING 2

The Verilog output of MyHDL for the input given in Listing 1. The direct conversion can easily be seen in this case, although MyHDL has handled some features for us such as resetting signals to default values that Verilog requires us to explicitly specify.

**ABOUT THE AUTHOR**

Colin O'Flynn (coflynn@newae.com) has been building and breaking electronic devices for many years. He is currently completing a PhD at Dalhousie University in Halifax, NS, Canada. His most recent work focuses on embedded security, but he still enjoys everything from FPGA development to hand-soldering prototype circuits. Some of his work is posted on his website at www.colinoflynn.com.

require you to define the entire port map as Verilog or VHDL would need.

While I don't have time to cover all these aspects, I want to talk you through at least a simple example of MyHDL simulation and implementation. To do this I'll be replicating the FIR filter from the February 2014 column.

ANOTHER FIRRY EXAMPLE

The FIR filter is not particularly exciting, but it does show off the use of MyHDL and Python to simplify your entire development. If you want to follow along, the easiest method is using a Python distribution such as WinPython on Windows. You can then install MyHDL using the pip tools, as described in the MyHDL documentation. This is all that is required to run the examples, which will be posted on ProgrammableLogicInPractice.com if you don't want to type everything in from the listing.

The MyHDL code for the FIR filter is shown in **Listing 3**, and the Verilog code generated by this is shown in **Listing 4**. Note the code in Listing 3 doesn't show the external interface or coefficient generation. I'll talk about that in a moment.

Comparing Listing 3 and Listing 4, you can note the similarity between the two code bases. One difference between the HLS C/C++ example from my previous column is that loop unrolling is not handled by MyHDL. Instead as MyHDL is operating at a similar level to Verilog or VHDL it relies on the synthesizer to perform the loop unrolling. Future version of MyHDL may support loop unrolling, but one could argue that perhaps this is not the job of the HDL, but instead the job of the hardware designer using the HDL.

Regardless of philosophical arguments, this does mean you are unable to automatically perform tasks such as tuning the trade-off between usage of hardware resources and throughput by asking the tools to unroll or not unroll a specific loop. The C/C++ HLS examples from my previous columns could be optimized for area or speed by a simple `#pragma` due to the support of C/C++ HLS to tune loop unrolling.

One thing I haven't explicitly mentioned until now is that MyHDL is entirely open-source (and free), whereas the C/C++ HLS has a \$2,000 yearly license fee and is proprietary. Thus, while I will compare the two for regular usage, it's worthwhile to also consider both the up-front cost, and the ability to modify the tools for your own use. MyHDL easily wins on both of those fronts!

But the real triumph of MyHDL can be seen once I introduce the complete simulation and generation environment. This is shown in **Listing 5**. The HDL code from Listing 3 is

not repeated, but you can consider the two listings are combined in the final program. In the C/C++ FIR example I required the use of external tools for filter design—with MyHDL, it's built right into the tools.

MyHDL is really just calling standard Python libraries, which have extensive tools for filter design. Thus, I could easily generate FIR or IIR filters of almost any order and type. The filter implementation itself is

fixed-point, and the Python code converts the floating-point types to the integer (fixed-point) notation in use. Full fixed-point support is still not present in the latest MyHDL release as of this column (0.8), but is on the roadmap for a future version.

Even without fixed-point support, the simulation environment of MyHDL pulls it ahead of C/C++ HLS. This makes it easy to verify correct operation of complex modules

LISTING 3

The core of the FIR module in MyHDL is given here. Note this snippet requires instantiation to declare signal widths and the filter constants.

```
# Based on IIR Filter code, which is Copyright Christopher Felton
# and released under the LGPL license.

from myhdl import *

def sfir_hdl(
    # ~~ Ports ~~
    clk,          # Synchronous clock
    x,            # Input word, fixed-point format described by "W"
    y,            # Output word, fixed-point format described by "W"

    # ~~ Parameters ~~
    B=None,      # Numerator coefficients, in fixed-point specified
    W=(24,0)     # Fixed-point description, tuple,
                # W[0] = word length (wl)
                # W[1] = integer word length (iwl)
                # fraction word length (fwl) = wl-iwl-1
):
    # Make sure all coefficients are int, the class wrapper handles all float to
    # fixed-point conversion.
    rB = [isinstance(B[ii], int) for ii in range(len(B))]
    assert False not in rB, "All B coefficients must be type int (fixed-point)"

    # We use a double-precision parameters as the result of the multiplication
    # will be 2x the input bit width. Define double width (precision ) max and min
    _max = 2**(2*W[0])
    _min = -1*_max

    Q = W[0]-1
    Qd = 2*W[0]

    # Delay elements, list of signals (double precision for all)
    ffd = [Signal(intbv(0, min=_min, max=_max)) for ii in range(len(B))]

    @always(clk.posedge)
    def rtl_fir():
        ffd[0].next = x
        for i in range(1, len(B)):
            ffd[i].next = ffd[i-1]

        yacc = 0
        for i in range(0, len(B)):
            b = B[i]
            yacc += b * ffd[i]

        # Double precision accumulator
        y.next = yacc >> Q

    return instances()
```

LISTING 4

The resulting FIR filter in Verilog, based on Listing 3. Again, note the fairly straightforward conversion from MyHDL to Verilog.

```

`timescale 1ns/10ps

module sfir_hdl (
    clk,
    x,
    y
);

input clk;
input signed [9:0] x;
output signed [9:0] y;
reg signed [9:0] y;

reg signed [20:0] ffd [0:19-1];

always @(posedge clk) begin: SFIR_HDL_RTL_FIR
    integer i;
    integer yacc;
    integer b;
    ffd[0] <= x;
    for (i=1; i<19; i=i+1) begin
        ffd[i] <= ffd[(i - 1)];
    end
    yacc = 0;
    for (i=0; i<19; i=i+1) begin
        case (i)
            0: b = 1;
            1: b = 2;
            2: b = 1;
            3: b = (-5);
            4: b = (-15);
            5: b = (-15);
            6: b = 13;
            7: b = 69;
            8: b = 128;
            9: b = 153;
            10: b = 128;
            11: b = 69;
            12: b = 13;
            13: b = (-15);
            14: b = (-15);
            15: b = (-5);
            16: b = 1;
            17: b = 2;
            default: b = 1;
        endcase
        yacc = yacc + (b * ffd[i]);
    end
    y <= $signed(yacc >>> 9);
end

endmodule

```

compared to C/C++ HLS, mostly as MyHDL is able to use the huge selection of Python modules to do everything from FFTs to graphing to I/O handling.

The simulation itself is performed in the `TestFreqResponse()` function. You will notice again the MyHDL-specific extensions

used here (such as the `@always` block to generate a clock signal). But we can use Python libraries as part of our test bench—appending data to lists or performing FFTs of data before saving.

In this example the function `PlotResponse()` generates the “expected”

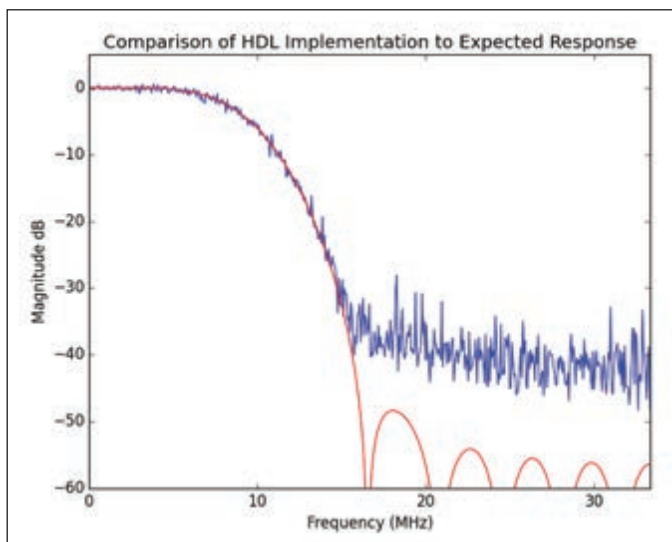


FIGURE 1 This shows the comparison of the expected FIR filter (in red) to the 10-bit fixed point implementation in blue. The fixed-point frequency response is obtained through a simulation in MyHDL.

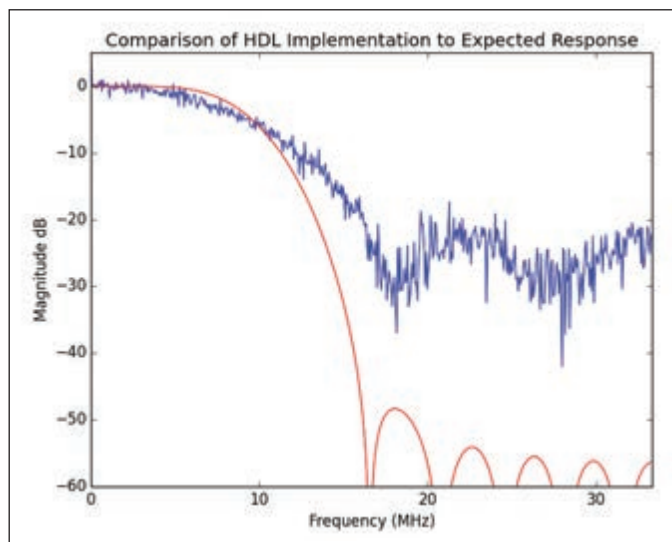


FIGURE 2 Compared to Figure 1, this shows what happens if we use only a 5-bit signal instead of a 10-bit signal. The loss of correct filter response can be seen by comparing the fixed-point response (in blue) to the expected response (in red).

frequency response of the filter based entirely on tried-and-true Python libraries, and compares it to our fixed-point results. The results of this are shown in **Figure 1**. Notice that the frequency response generally follows the expected response. This is using 10-bit inputs (the same as the ADC/DAC on my test board) and 20-bit intermediate values.

I could easily change the HDL to use 5-bit integers for the input values, which causes some additional divergence of my filter frequency response to the ideal filter. This frequency response of this fixed-point implementation is shown in **Figure 2**.

As a final test I've implemented the FIR filter in a Spartan 3 device, with an ADC and DAC running at 66.67 MHz. The FIR filter has been inserted between the ADC and DAC, and the frequency response is plotted in **Figure 3**. The analog path isn't perfect here which accounts for some of the errors, but you can see the response falls within "expected" bands compared to Figure 1.

MyHDL made it trivial to entirely describe a filter which can be synthesized onto a FPGA. Unlike the C/C++ HLS example, I was able to use Python tools to include the entire filter design specifications into the source file.

EVEN MORE THROWN IN

While this brief introduction to MyHDL won't do it full justice, there are a few more things worth mentioning. One thing I can't miss is highlighting the ability to perform unit testing in MyHDL. When designing hardware modules it can be a hassle to ensure you have tests for every module, and let alone attempting to script those tests to continuously run.

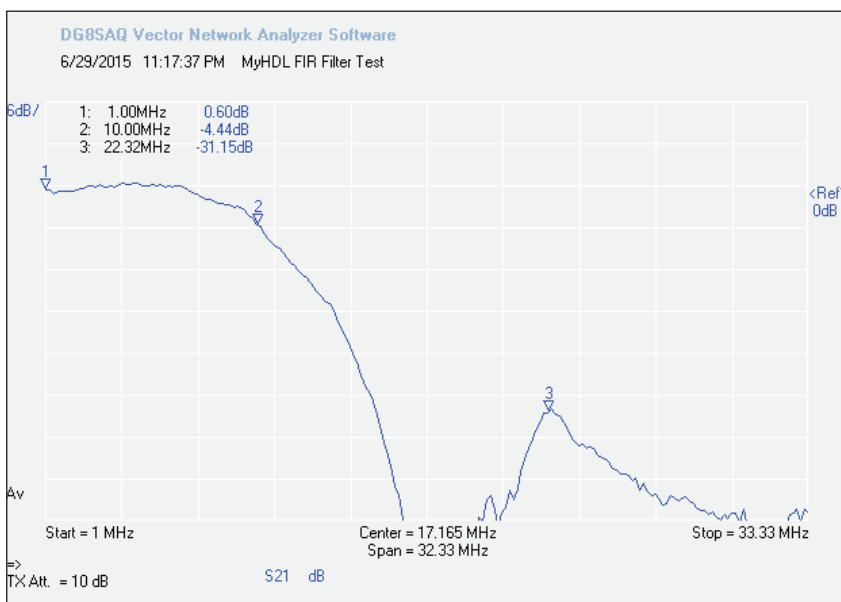


FIGURE 3 The implemented FIR filter using 10-bit integer inputs is tested on a FPGA, where the input and output of the filter is an ADC and DAC respectively sampling at 66.67 MHz. This figure has numerous sources of error due to the introduction of an analog signal path, but it can be seen to generally match the expected FIR filter response.

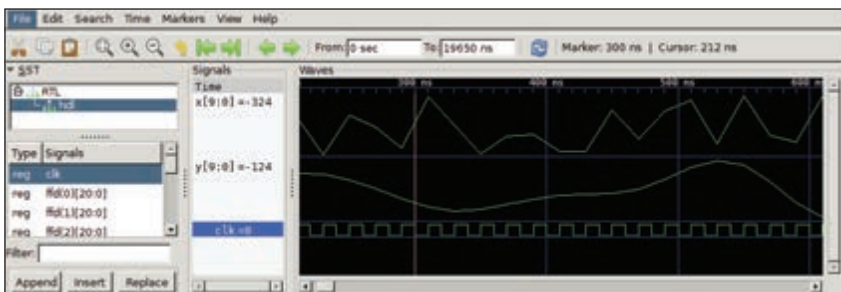


FIGURE 4 MyHDL also makes it easy to trace signal changes with time, by writing all signal changes to a VCD file. Here I'm inspecting the input and output of the filter for the frequency-bandwidth test used in generating Figures 1 and 2.

```

# Based on IIR Filter code, which is Copyright Christopher Felton
# and released under the LGPL license.

from myhdl import *
import numpy as np
from numpy import pi, log10
from numpy.fft import fft
from numpy.random import uniform
from scipy.signal import firwin, freqz
import pylab

class SFIR():
    def __init__(self,
                 Fc=10E6,      # cutoff frequency
                 Fs=66.66E6,   # sample rate
                 W=(24,0)      # Fixed-point to use
                 ):
        # The W format, intended to be (total bits, integer bits,
        # fractional bits) is not fully support.
        # Determine the max and min for the word-widths specified
        self.W = W
        self.max = int(2**(W[0]-1))
        self.min = int(-1*self.max)

        # Filter Design
        N = 19
        Wn = 0

        # Define the cutoff as a fraction of the nyquist
        Wn = float(Fc)/(Fs/2.0)
        self.b = firwin(N, Wn)

        # fixed-point Coefficients for the FIR filter
        self.fxb = np.round(self.b * self.max)/self.max

        # Create the integer (fixed-point) version
        self.fxb = tuple(map(int, self.fxb*self.max))

        print "FIR w,b", Wn, self.b
        print "FIR fixed-point b", self.fxb

    def Convert(self, W=None):
        clk = Signal(False)
        x = Signal(intbv(0,min=-2**(self.W[0]-1), max=2**(self.W[0]-1)))
        y = Signal(intbv(0,min=-2**(self.W[0]-1), max=2**(self.W[0]-1)))

        toVerilog(sfir_hdl, clk, x, y, B=self.fxb, W=self.W)

    def TestFreqResponse(self, Nloops=3, Nfft=1024):
        self.Nfft = Nfft
        Q = self.W[0]-1
        clk = Signal(False)
        x = Signal(intbv(0,min=-2**Q,max=2**Q))
        y = Signal(intbv(0,min=-2**Q,max=2**Q))
        xf = Signal(0.0)

        dut = traceSignals(self.RTL, clk, x, y)

        @always(delay(10))
        def clkgen():

```



13th International System-on-Chip (SoC) Conference, Exhibit & Workshops

October 21 & 22, 2015

University of California, Irvine - Calit2

www.SoCconference.com

EARLY BIRD REGISTRATION IS NOW OPEN!

Platinum Sponsors



- Sub 10nm Designs & Beyond
- Analog & Mixed-Signal Designs
- SoC Design & Verification
- 3-D ICs Designs
- IC Security & Challenges
- Innovative EDA Tools
- Complex IP Subsystems
- Low-Power Techniques
- Memory Trends & Technologies
- Table-Top Exhibit (Free Passes)

- Complex Mixed-Signal SoCs
- SOI vs. CMOS
- RF Design
- FPGAs –Trends & Designs
- High-Speed I/Os
- Smarter Mobile Devices
- Informative Panels
- Network-on-Chips (NoCs)
- Networking Opportunities
- And Much More . . .

The Most Informative, Targeted & Educational IC & IP Design Conference, Exhibit & Workshops of the Year!

Keynote Speakers



Intel
Dr. Jeff Parkhurst, Science & Technology Center & Technology Center Program Director.



University of California, Irvine (UCI)
Dr. G.P. Li, Calit2 Director, Professor of Engineering.



Microsemi
Jim Aralis, Chief Technology Officer (CTO), and Vice President of R&D.



Skyworks
James P. Young Vice President, Advanced Development.

Selected Participating Companies & Universities



For More Information or Questions, Please Contact the SoC Conference Organizing Committee at:

SoC.Conf.Update@Gmail.com or (949) 981-1837

www.SavantCompany.com & www.SoCconference.com


```

        clk.next = not clk

    @always(clk.posedge)
    def ist():
        xi      = uniform(-1,1)
        x.next  = int(self.max*xi)
        xf.next = xi

    @instance
    def stimulus():
        ysave   = np.zeros(Nfft)
        xsave   = np.zeros(Nfft)

        self.yfavg = np.zeros(Nfft)
        self.xfavg = np.zeros(Nfft)

        for ii in range(Nloops):
            for jj in range(Nfft):
                yield clk.posedge
                xsave[jj] = float(xf)
                ysave[jj] = float(y)/self.max

                self.yfavg = self.yfavg + (abs(fft(ysave, Nfft)) / Nfft)
                self.xfavg = self.xfavg + (abs(fft(xsave, Nfft)) / Nfft)

            raise StopSimulation

    return instances()

def RTL(self, clk, x, y):
    hdl = sfir_hdl(clk, x, y, B=self.fxb, W=self.W)
    return hdl

def PlotResponse(self):
    # Plot the designed filter response
    pylab.ioff()

    Fs = 66.66E6

    # plot the simulated response
    # -- Fixed Point Sim --
    xa = (2*pi * np.arange(self.Nfft)/self.Nfft) / (2*pi) * Fs
    H = self.yfavg / self.xfavg
    pylab.plot(xa, 20*log10(H), 'b' )

    w, h = freqz(self.b)
    # pylab.hold(True)
    pylab.plot((w/(2*pi))*Fs, 20 * np.log10(abs(h)), 'r')

    pylab.ylabel('Magnitude dB');
    pylab.xlabel('Frequency (MHz)')
    pylab.axis([0, Fs/2, -60, 5])
    pylab.xticks([0,10E6,20E6,30E6], ['0', '10', '20', '30'])
    pylab.title('Comparison of HDL Implementation to Expected Response')

```

```

pylab.savefig("firtest.png")

if __name__ == '__main__':
    # Instantiate the filter and define the Signal
    W = (10,0)
    flt = SFIR(W=W)

    flt.Convert()

    tb = flt.TestFreqResponse(Nloops=3, Nfft=1024)
    sim = Simulation(tb)
    print "Run Simulation"
    sim.run()
    print "Plot Response"
    flt.PlotResponse()

```

LISTING 5

The real power of MyHDL occurs once we start building systems around our cores. Here I'm using the SciPy library to automatically generate FIR filter coefficients, given the sampling frequency and desired cut-off. I can also compare the output of the fixed-point filter implementation to an idea FIR filter for the selected fixed-point bit width.


Once again MyHDL pulls on existing work in Python to simplify our test cases. It uses the `unittest` module from Python to allow you to easily generate test cases. Such test cases can validate a range of inputs—including testing options such as various bit widths to your module. These tests can be strung together with regular Python code, a task it excels at.

When it comes to debugging or documenting the code, MyHDL can automatically trace into a module and save signal waveforms to a `.vcd` file. Such a file can be opened by a universal viewer, such as `gtkwave`, which I show plotting the input and output of my FIR filter in **Figure 4**. The trace statement itself can be seen in **Listing 5**, as the call to `traceSignals()`. This was all done without any additional Verilog simulator, but as part of the regular MyHDL development process.

MYHDL SUPERHERO

MyHDL presents a number of credible reasons it can be taken seriously as a hardware description language (HDL). Most critically, it doesn't try to be "too clever", but instead inserts itself at about the same level as your existing Verilog or VHDL code. But by using the power of Python, MyHDL greatly simplifies aspects such as simulation and unit testing of your design, while improving many aspects that affect your synthesizable module such as clarifying reset signal handling and improving parametrized port definitions.

If you want to learn more about MyHDL, I'll have some examples (such as the FIR filter in this column) and links to other resources at the `ProgrammableLogicInPractice.com` website. But there is extensive documentation online at the MyHDL project webpage (`MyHDL.org`), which also includes a few examples. Christopher Felton has a number of additional well-documented examples such as FFTs, IIR filters, and more. You'll find further examples on various webpages such as everything from simple counters to Kalman filters implemented in MyHDL.

Considering MyHDL is free and easily available, there's nothing to stop you from giving it a spin. I think you'll find it has the right combination of familiar constructs that get you up to speed quickly with the new language, but adds enough new functionality to improve your overall productivity and enjoyment of FPGA development. Have fun! 

MEASUREMENT COMPUTING

MCC Continues to Lower the Cost of DAQ

- Easy to Use
- Easy to Integrate
- Easy to Support



USB-230 Series From \$249

- 8 SE/4 DIFF analog inputs
- 16-bit resolution
- Up to 100 kS/s sample rate
- 8 digital I/O
- One 32-bit counter
- Two analog outputs
- Included software and drivers



OEM board-only versions are also available

mccdaq.com/USB-230-Series

**MEASUREMENT
COMPUTING™**

Contact us

1.800.234.4232

©2015 Measurement Computing Corporation
info@mccdaq.com

EMBEDDED IN THIN SLICES

The Internet of Things (Part 3)

Connect Wirelessly with a Microcontroller

COLUMNS

In the second part of this article series, Bob provided tips on choosing a carrier for a 'Net-connected embedded system. In this article he details how to connect simple devices wirelessly to the Internet.

By Bob Japenga (US)

In our first installment, we talked about the various ways we have connected our embedded systems wirelessly to the Internet. In Part 2, I covered the various decisions that need to be made in choosing a carrier for your embedded system. This month we will look in detail at how we have connected simple devices wirelessly to the Internet.

Yesterday, we had a new potential client fly in to see us to discuss their need to create an Internet of Things (IoT) device for their very non-electronic company. This company has been in business for over 50 years making and innovating extremely low-tech products. Now they wanted their extremely low-tech product to be wirelessly connected to the Internet. They came to us with a prototype developed by their general manager using an Arduino and a development kit. This was a man who had no electrical engineering background. It was quite impressive. He did not have any complications sending the data to the cloud. He did struggle with designing the sensor to obtain the data he desired. With the feasibility behind them and after becoming knowledgeable enough to know what it takes to join the IoT revolution, they wanted us to

turn it into a product. This shows how easy it is to take a simple microcontroller, talk to a complex cell module, and create a device that will soon join the IoT revolution.

Many of our systems are similar to theirs and use a very simple microcontroller with very little memory. Some cell modules offer a good assortment of options for connecting to the web. Since some of the module manufacturers we use require an NDA to obtain their documentation, we will only talk about our experience with a Swiss company called u-blox who freely publishes their technical documentation on-line. In particular we will talk about the features of the LISA C200, which supports CDMA. Other modules have similar functionality. **Figure 1** shows the basic system architecture that we will be reviewing this month.

AT COMMAND SET

Most of the module manufacturers provide an AT command set to configure the module and initiate communications over the network. Developed by Dennis Hayes for the 300-baud Hayes SmartModem (ah, those were the days!), this simple command and response



protocol is used on a lot of communications devices. This is the primary way a small microcontroller will talk to the cell module. With the AT command set, the microcontroller can easily use: FTP, HTTP, UDP, TCP/IP, and SMS. Let's look briefly at how easy this is to do.

FILE SYSTEM

All of the networking commands can use an on-board flash file system on the cell module. This flash file system has about 1 MB of disk space available for you to use. There are rudimentary file system commands to allow you to read, write, get stats, or delete (ASCII or binary) files from this flash file system. Since the files are read over a serial port it is not like reading from a classic disk controller. Any read over the serial port could be corrupted. Thus we always provide some method of validation that what you have read or written over the serial port is indeed what you intended.

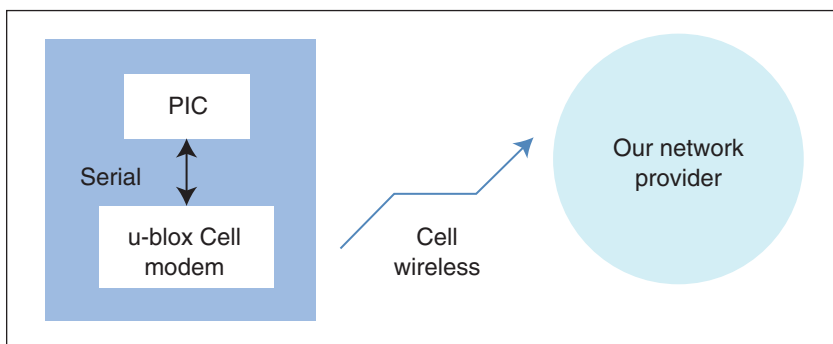
FTP

FTP is a file transfer protocol that allows you to send files unencrypted (including the password) over the Internet to and from your system. If you want to send or receive files over the Internet to or from an FTP server, you need an ftp client on your device. The cell module contains that ftp client for you and provides an AT command interface to that client.

There are just a few AT commands required to send a file. The first one defines the username and password as well as the URL (either named or IP address). Then you log in with another AT command. You then can either obtain one file from the server (get) or send one file to the server (put). Each command provides a response in a file on the file system as to the success or failure of the operation. Additional commands may be required to obtain the status of the network. Once you have completed sending or receiving files, you can log off with another AT command.

The LISA C200 supports an FTP client which supports the FTP commands listed in **Table 1**. A number of standard FTP functions are not supported (like verbose) because they don't apply. There are others that would be nice to have (like the append command or mput and mget for sending and receiving multiple files) but are not provided.

In today's world, FTP should rarely be used because it is so insecure. We had one customer many years ago who, when we offered both FTP and secure FTP (SFTP), wanted us to delete FTP off his devices because he did not want his customers to use



it. If you are on a private network, FTP would work and maintain the security you need.

FIGURE 1
The basic system architecture

HTTP

The cell module supports the following HTTP methods: GET, PUT, POST, HEAD, and DELETE. The process is quite simple and does not require a lot of code to implement on your little microcontroller. You can set up a number of profiles which contain exactly where the data is going with a series of AT commands. Once a profile is set, you can issue another AT command to initiate the specific HTTP method associated with that profile. For example, if you are POSTing to multiple URLs, you can set up a separate profile for each. The response from the server comes back as a file on the local flash file system that you specified. You can read the file using the file system AT read command.

As with FTP, it would be ideal if HTTPS (secure HTTP) was supported. Without HTTPS, on one project we needed to encrypt everything that is included in the payload of the HTTP request and decrypt everything that comes back. For this application we used Advanced Encryption Standard (AES) 128, which uses a symmetric private key algorithm.

AES 128 is a specific instance of an encryption standard established by the National Institute of Standards and Technology

TABLE 1
The FTP client supports several ftp commands

u-blox FTP Command	Windows FTP	Description
0	Quit	Log out
1	User	Log in
2	Delete	Delete file from the server
3	Rename	Rename file on the server
4	Get	Retrieve file from the server
5	Put	Send a file to the server
8	Cd	Change the working directory
10	Mkdir	Make a directory on the server
11	Rmdir	Remove directory on the server
13	N/A	Obtain stats on a file on the server
14	Dir	List the filenames on the server



ABOUT THE AUTHOR

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.

in the United States. It uses a 128-bit key. The encryption and decryption algorithms are public knowledge. There are thus 3.4×10^{38} possible keys. If you tried to decrypt the encrypted data by brute force by trying a different key a trillion times per second, it would still take 10^{18} years. To date, no one has found any way to crack the code other than brute force.

TCP/IP

On one system, we initially had a diagnostic mode where we were sending the contents of a screen every 10 seconds (the embedded version of Google Hangout) using HTTP. When we specified it, we knew that HTTP and the server could handle the data throughput, but the cell module just could not keep up with sending 1k of data every 10 seconds in separate HTTP POSTs. As a result we implemented the feature using the cell module's TCP/IP capability.

The cell module provides AT commands to create the socket, to open the socket, to send the data, to close the socket, and to set options for the socket. When a message comes back from the socket, an asynchronous response is provided so your microcontroller does not need to poll the server for the results.

The cell module can handle up to 6 TCP/IP sockets open at a time. It also supports sending and receiving binary or ASCII data.

SMS

A Short Message System (SMS)—or what we call “texting”—client can be invaluable if you wish to provide a relatively responsive system while keeping your data plan costs low. For example, let's say that you want to send up data once per day. Occasionally you would also like to be able to command the device to do something and have it respond within a few seconds. Since the overhead of an HTTP POST can be greater than 1 KB, checking the server every minute for some kind of request would use up more than one megabyte of data per day from your data plan. With SMS, the server can just send the embedded system an unsolicited message with specific instructions. The text will be stored in a file and the presence of that file can be polled (at no cost to your data plan).

A couple of AT commands will set up

your cell modem for sending or receiving text messages. The microcontroller can periodically send another AT command to see the file you specified to receive SMS messaging with no cost to your data plan except for the SMS message. If you plan to use this approach, make sure your data plan supports SMS.

PROBLEMS WITH THIS APPROACH

No cell module will implement the full HTTP or FTP specification. Let's review some of the more serious shortcomings that we uncovered.


With the u-blox module, we found that both the FTP and HTTP implementation did not support sending multiple files with a single command. This actually resulted required us to rewrite some of our server code. So if you are going to be using your microcontroller to talk to an existing cloud server that may be expecting multiple files, you are out of luck.

Another shortcoming is that the HTTP and FTP responses are slower than when you connect these cell modems via the Point to Point Protocol (PPP) (if the cell modem supports PPP) or Ethernet (some cell modules simulate an Ethernet connection eliminating the overhead of PPP).

We also found that we wanted to tweak the HTTP headers and this was also not allowed on the u-blox module. We used a serial port (although all of the cell modems we have used have USB) on one of the two u-blox implementations. Even at 115,200 baud, this is a relatively slow interface for accessing 0.5-MB files. If your microcontroller can support USB, then I would recommend that you talk to the modem via USB.

Finally, we mentioned the lack of HTTPS and SFTP support previously. As one of our engineers said, HTTP is being deprecated across the Internet. Even Google search engine ratings are lowered if your web site doesn't support HTTPS. Not that this applies to our M2M world, but it does speak to a growing trend away from HTTP.

ON TO CERTIFICATION

Connecting your device to the Internet has become much simpler with the design of a number of the cell modules on the market. Using simple AT commands over a serial port or a USB port, you can connect your device to the Internet and join the Internet of Things revolution. Next time we will discuss certification options for your embedded systems. Certification is a big topic so I can guarantee that we will approach it in thin slices. 



circuitcellar.com/ccmaterials

SOURCE

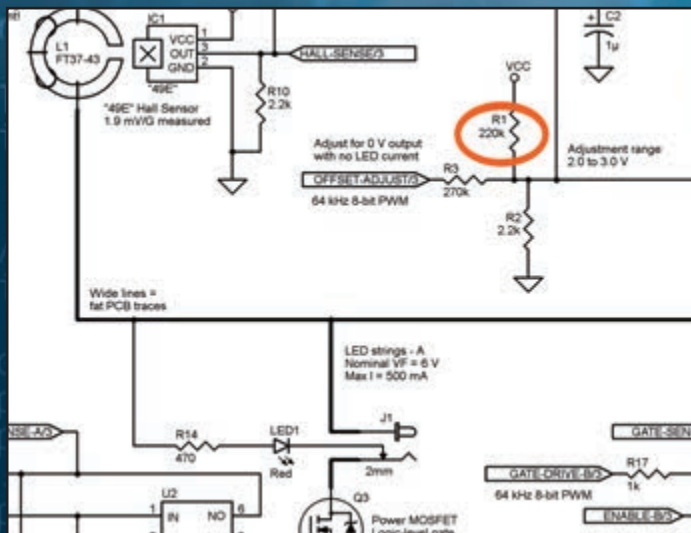
LISA-C200 CDMA 1xRTT Module
ublox | www.u-blox.com



MONTHLY ENGINEERING CHALLENGE

Each month, you're challenged to find an error in a schematic or in code that's presented on the challenge webpage. Locate the error for a chance to win prizes and recognition in Circuit Cellar magazine!

Prizes such as a NetBurner MOD54415 LC Development kit or a Circuit Cellar subscription will be announced each month.



```
1 #include <stdio.h>
2
3 int main()
4 {
5     int first, second, sum;
6     int *p, *q;
7
8     printf("Enter two integers to add: \n");
9     scanf("%d%d", &first, &second);
10
11     p = &first;
12     q = &second;
13
14     sum = p + q;
15
16     printf("Sum of entered numbers = %d\n", sum);
17
18     return 0;
19 }
```

Participate: circuitcellar.com/engineering-challenge-netburner

Launch: 1st of each month

Deadline: 20th of each month

No purchase necessary to enter or win. Void where prohibited by law. Registration required. Prizes subject to change based on availability. Review these terms before submitting each Entry. More info: circuitcellar.com/engineering-challenge-netburner-terms

THE DARKER SIDE

How to Select an Operational Amplifier

COLUMNS

Op-amps are essential components that have replaced transistors in many designs. This month, Robert explains the key characteristics of op-amps and provides tips for choosing the right one.

By Robert Lacoste (France)



Welcome back to The Darker Side. In the early years, electronic engineers excelled at analog circuit design. Each transistor had to be selected based on its physical characteristics and, of course, the intended application. They had to be biased with care and powered on with even more care. Then integrated circuits (ICs) were invented. (Jack Kilby, working for Texas Instruments, manufactured the first actual chips in 1958, even though the concept was patented 10 years earlier by Werner Jacobi.) Soon after that, in 1963, the first monolithic integrated operational amplifier was designed by Bob Wildar at Fairchild Semiconductors. The μ A702 was born. Operational amplifiers (op-amps for short) existed before that. **Figure 1** shows a well-known example. However, with such an integration, their success took another dimension. The same Bob Wildar then designed the μ A709 in 1965, and moved to National Semiconductors to design the LM101. As a response, in 1968, Fairchild launched the well-known μ A741, which we still use more than 50 years later! I will stop with the history at this point, but if you are interested,

check out Walt Jung's interesting document titled "Op Amp History," which is listed in the Resources section of this article.

Op-amps are great building blocks that have replaced transistors in many designs. Compact, inexpensive, simple, and flexible, op-amps sure offer many advantages. However, designers sometimes forget that these small chips are nothing more than a set of interconnected transistors. They are not magic black boxes. They have electrical characteristics and limitations. So, you must select them with as much care as our predecessors selected their transistors.

This month, I will detail an op-amp's key characteristics. My aim is to convince you that the figures and graphs that fill tens of pages on the datasheets are actually useful, even if they are ignored 90% of the time. You will then know what's critical for your design and ultimately how to select the relevant op-amp for your application. The μ A741 is a great chip, but it might not be the best one for you!

Of course, in such an article, I can only scratch the surface of the subject. However, there is a must-read for

this topic (as for plenty of others): *The Art of Electronics*, by Paul Horowitz and Winfield Hill. The book is not cheap, but you definitely need it in your library. I promise you that it's worth reading the long book, including the 200 pages or so dealing with operational amplifiers. By the way I just bought the newly released third edition, and it is still on my bedside table.

OP-AMP BASICS

Let's start by refreshing your memory. Basically, an op-amp is a DC-coupled differential amplifier (see **Figure 2**). It has one noninverting input (IN+), one inverting input (IN-), and one output (OUT). The output voltage is simply the voltage difference between the two inputs, multiplied by a given open-loop gain: $V_{OUT} = GAIN_{OPENLOOP} \times (V_{IN+} - V_{IN-})$. However, any op-amp has a very high open-loop gain. When I say "very high," I mean that a gain of 1,000,000 is not uncommon. So as soon as the voltage on the noninverting input exceeds the voltage of the inverting one, even by some tens of microvolts, the output voltage will jump as high as possible. This means the op-amp will saturate with an output voltage close to its positive $V+$ power supply. Similarly, if the voltage on the inverting input is higher than the noninverting input, then the output will jump down to $V-$. As a consequence, an op-amp is nearly never used in such an open-loop configuration. It would be just a voltage comparator (and dedicated chips do exist for that). There is always a feedback loop around the op-amp, as I will show you in a minute. For the moment, keep in mind the first rule of an ideal op-amp.

Rule 1: *The voltage between the two inputs of a non-saturated op-amp is always zero.*

Another key characteristic of any op-amp is that the currents flowing through the inputs are very low. Once, again I mean very low—that is, some nano-amperes are common. This leads to the second golden rule, once again for an ideal op-amp.

Rule 2: *The current through the inputs of an op-amp is null.*

Lastly, an op-amp needs to be powered, of course. It has two power input pins, positive and negative. You can connect the $V+$ and $V-$ pins to a symmetrical power source like ± 10 V, or you can ground the $V-$ pin and wire the $V+$ pin to a positive power supply. Don't be fooled by datasheets stating that a specific chip is designed for "bipolar power" or "unipolar power." An op-amp can't tell the

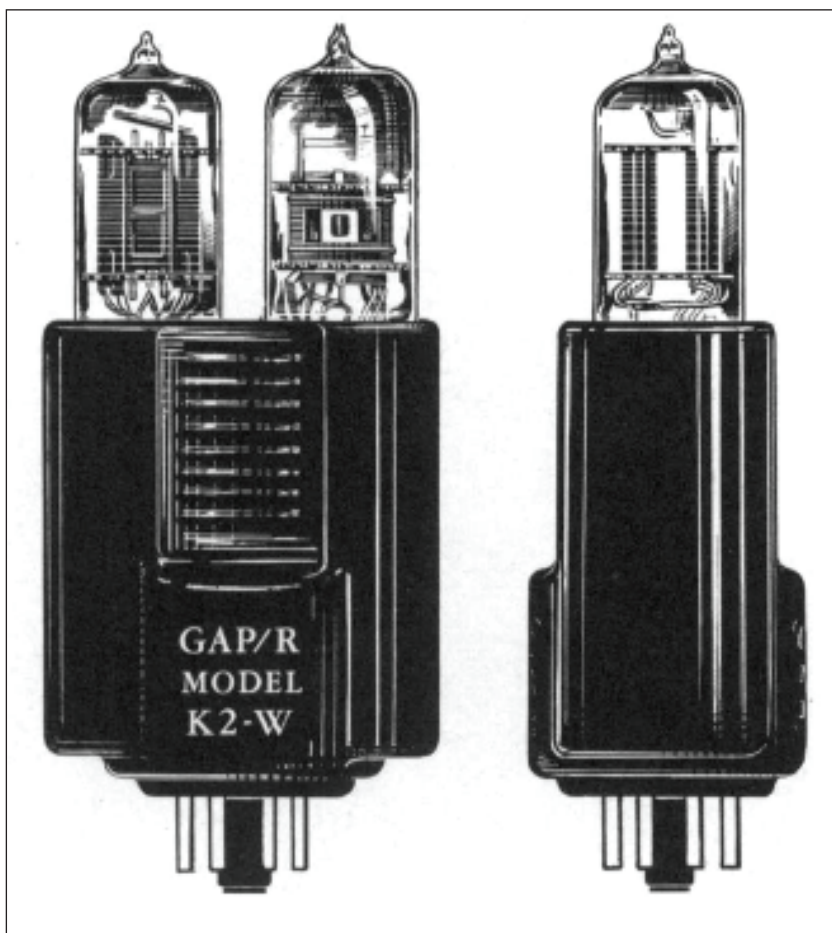


FIGURE 1

The K2-W was the first commercial "integrated" operational amplifiers (1952). It was designed by George A. Philbrick Researches (GAP-R), Inc. (Source: www.philbrickarchive.org)

difference between a ± 10 -V power supply and a 0/20-V one, because it doesn't have a ground connection!

TYPICAL DESIGNS

You will find plenty of designs around an op-amp in books or on the Internet: amplifiers, differentiators, summaters, etc. The three most basic examples are illustrated in **Figure 3**. Even though they are simple, it is useful to know how to calculate such circuits using the two rules I listed above. Let's start with the voltage follower. Here

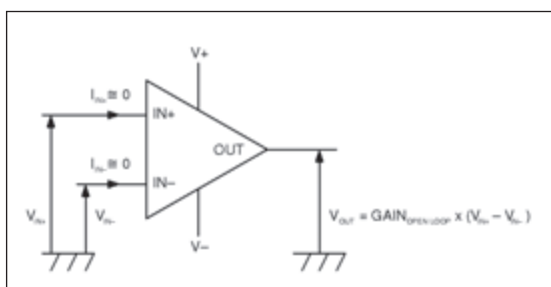


FIGURE 2

An op-amp is simply a differential amplifier. Its key characteristics are a very high open-loop gain and very low input bias currents.

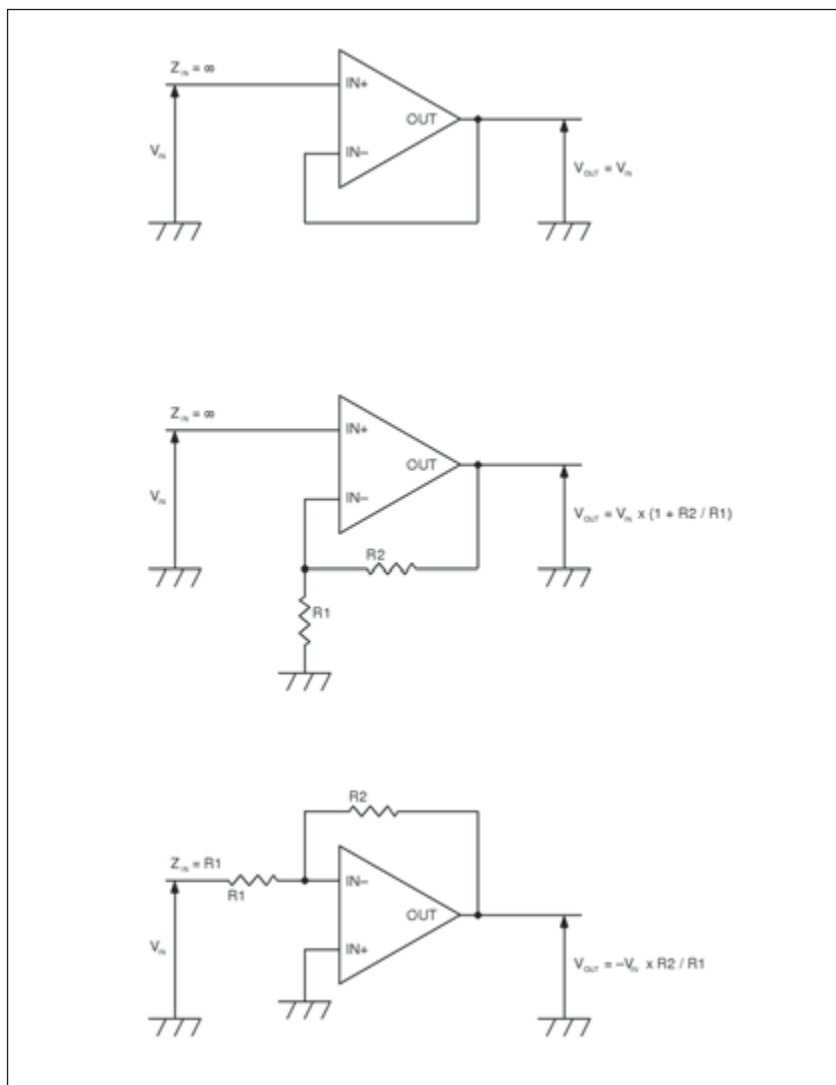


FIGURE 3

These are some typical applications for an op-amp. As shown, there is always an external feedback loop around the amplifier.

the input voltage is simply connected to the noninverting input, and the inverting input is connected to the output. Remember Rule 1? The voltage between the two inputs must be zero; therefore, the output voltage is identical to the input voltage. Moreover, as the input current is null (thanks to Rule 2), the input impedance is virtually infinite. We have a perfect voltage follower (see Figure 3a).

The noninverting amplifier design (see Figure 3b) is nearly identical. This time, two additional resistors are used. The input voltage is still connected to the noninverting input, so the voltage at the inverting input must also be V_{IN} . But this inverting input is connected to the ground through a resistor $R1$. Remember Ohm's law? The current

through $R1$ must be $I = U/R = V_{IN}/R1$. But, wait, no current is flowing through this input as stated by Rule 2. So the same current must also go through $R2$, which is wired between the inverting input and the output. The voltage drop through $R2$ is then simply $U = R \times I = R2 \times (V_{IN}/R1)$. So what is the voltage at the output of the op-amp? It is equal to the voltage on the inverting input (which is equal to V_{IN}), plus this voltage drop across $R2$ that we have just calculated. So we have $V_{OUT} = V_{IN} + (R2 \times V_{IN}/R1)$, which you can rewrite as $V_{OUT} = V_{IN} \times (1 + R2/R1)$. Here it is: we have a noninverting amplifier and its gain is fixed by $R1$ and $R2$.

I hope you grabbed that all these circuits can be easily calculated using just the two golden rules. As an exercise, I encourage you to do the same simple calculation with the inverting amplifier (see Figure 3c). It's easy if you understand the basic idea.

WHEN ZERO IS... NOT NULL

Up to now, I've assumed that the op-amp was perfect. But life would be boring if everything were perfect. Let's assume that you're using our old friend, the UA741, an equivalent of the $\mu A741$. Refer to **Table 1**, which includes data from STMicroelectronics. What do all the numbers mean?

The first line of this data sheet is labeled "input offset voltage." This is a measure of how good this op-amp is, as compared to the first golden rule stated above. As the circuit is not perfect, the voltage difference between the two inputs is not exactly null. Here the typical error is specified as 1 mV. The manufacturer states that this error can go up to 5 mV, and even 6 mV when the temperature is not 25°C, but I will just consider the typical value. Is 1 mV of error low enough? It depends on your application. This can be neglected, or it can transform your project into a useless bunch of wires and silicon. Imagine that you're designing an amplifier for a thermocouple sensor. The input voltage would be in the 10-mV range, so a 1-mV offset error would introduce a 10% error on the output! That's why dedicated op-amps exist for such applications. These amplifiers, often referred as "zero-drift," are specifically designed to reduce the offset voltage to a minimum, with built-in offset canceling circuits. Do you want an example? Find datasheet on the web for the Texas Instruments TLC2652AC. The chip provides a maximum offset error of 1 μV , and an even more impressive temperature variation of 0.003 $\mu V/^{\circ}C$. That's better than the UA741, isn't it?

Going back to the datasheet (see **Table 1**), the third line is called "Input Bias Current."



139TH

AUDIO ENGINEERING SOCIETY
INTERNATIONAL CONVENTION

NEW YORK CITY, NEW YORK
JACOB JAVITS CENTER
CONFERENCE: OCT 29 – Nov 1, 2015
EXHIBITS: OCT 30 – Nov 1, 2015

If It's About *Recording, Live Sound & More*, It's At **AES139**

Join the AES in NYC this fall for **THE Audio Event of the Year**

The **139th Audio Engineering Society Convention** will offer three **FREE*** days of interactive, immersive, hands-on experience with the latest audio gear, along with special events and showcases from leading manufacturers and audio-related services. Our **Project Studio Expo** and **Live Sound Expo** are returning with a jam-packed comprehensive program, where attendees can learn with and from the movers and shakers in the industry. **AES139** will also feature four days of workshops, technical papers and program content from those driving innovation in the audio industry. **The AES Convention** in NYC will be **THE Audio Event of the Year** and is not to be missed.

TOPICS WILL INCLUDE:

- Recording and Production
- Mastering
- Broadcast and Streaming
- Game Audio
- Product Design
- Networked Audio
- Live Sound
- Archiving and Restoration



Photo courtesy of Laz Harris

If It's About **AUDIO**, It's At **AES!**

Visit <http://www.aes.org/events/139> for the latest **Registration, Travel** and **Technical Program** information for the **AES139 Convention**.



#AES139



Facebook.com/AESorg

***FREE** advance registration includes exhibits, special events,
Project Studio Expo & Live Sound Expo.

Symbol	Parameter	Min.	Typ.	Max.	Unit
V_{io}	Input offset voltage ($R_S \leq 10 \text{ k}\Omega$) $T_{amb} = +25 \text{ }^\circ\text{C}$ $T_{min} \leq T_{amb} \leq T_{max}$		1	5 6	mV
I_{io}	Input offset current $T_{amb} = +25 \text{ }^\circ\text{C}$ $T_{min} \leq T_{amb} \leq T_{max}$		2	30 70	nA
I_{ib}	Input bias current $T_{amb} = +25 \text{ }^\circ\text{C}$ $T_{min} \leq T_{amb} \leq T_{max}$		10	100 200	
A_{vd}	Large signal voltage gain ($V_O = \pm 10 \text{ V}$, $R_L = 2 \text{ k}\Omega$) $T_{amb} = +25 \text{ }^\circ\text{C}$ $T_{min} \leq T_{amb} \leq T_{max}$	50 25	200		V/mV
SVR	Supply voltage rejection ratio ($R_S \leq 10 \text{ k}\Omega$) $T_{amb} = +25 \text{ }^\circ\text{C}$ $T_{min} \leq T_{amb} \leq T_{max}$	77 77	90		dB
I_{CC}	Supply current, no load $T_{amb} = +25 \text{ }^\circ\text{C}$ $T_{min} \leq T_{amb} \leq T_{max}$		1.7	2.8 3.3	mA
V_{icm}	Input common mode voltage range $T_{amb} = +25 \text{ }^\circ\text{C}$ $T_{min} \leq T_{amb} \leq T_{max}$	± 12 ± 12			V
CMR	Common mode rejection ratio ($R_S \leq 10 \text{ k}\Omega$) $T_{amb} = +25 \text{ }^\circ\text{C}$ $T_{min} \leq T_{amb} \leq T_{max}$	70 70	90		dB
I_{OS}	Output short circuit current	10	25	40	mA
$\pm V_{opp}$	Output voltage swing $T_{amb} = +25 \text{ }^\circ\text{C}$ $T_{min} \leq T_{amb} \leq T_{max}$				V
	$R_L = 10 \text{ k}\Omega$	12	14		
	$R_L = 2 \text{ k}\Omega$	10	13		
	$R_L = 10 \text{ k}\Omega$ $R_L = 2 \text{ k}\Omega$	12 10			

TABLE 1

This information from the the UA741 datasheet shows all the amplifier's key DC characteristics (Source: www.st.com)

As you can imagine, this one is related to the second golden rule. The currents flowing through the inputs are not exactly null. They are stated to be lower than 10 nA typically, but can go up to 200 nA. Moreover, the current flowing through both inputs are not equal. The difference, called "Input Offset Current" (second line of Table 1), is about 2 nA. These are low currents, but are they low enough? Once again, this depends on your application. If you design a noninverting amplifier with two external resistors, then this small leakage will introduce a little error on the theoretical gain. The 10 nA will roughly double the 1-mV voltage offset error if the resistors are in the range of $R = U/I = 2 \text{ mV}/10 \text{ nA} = 200 \text{ k}\Omega$. This will not be a

problem if you use low-value resistors, but may become an issue if you require large resistors. Another source of concern is that the same leakage current will be applied on your circuit input. Note that 10 nA will usually be negligible, but this is not the case if you want to amplify a very small current coming, for example, from a sensitive photodiode. Once again, there are specialized op-amps for such applications, referred as "low bias op-amps." They are designed to reduce this input bias current as much as possible. For example, look at the LTC6268 from Linear Technologies. Its typical bias current is 3 fA. Yes, three femtoamps. That's 3 million times better than the UA741.

Lastly, the datasheet's fourth line, which is

Symbol	Parameter	Min.	Typ.	Max.	Unit
GBP	Gain bandwidth product $V_i = 10 \text{ mV}$, $R_L = 2 \text{ k}\Omega$, $C_L = 100 \text{ pF}$, $f = 100 \text{ kHz}$	0.7	1		MHz
THD	Total harmonic distortion $f = 1 \text{ kHz}$, $A_v = 20 \text{ dB}$, $R_L = 2 \text{ k}\Omega$, $V_o = 2 V_{pp}$, $C_L = 100 \text{ pF}$, $T_{amb} = +25^\circ \text{ C}$		0.06		%
e_n	Equivalent input noise voltage $f = 1 \text{ kHz}$, $R_s = 100 \Omega$		23		$\frac{\text{nV}}{\sqrt{\text{Hz}}}$
ϕ_m	Phase margin		50		Degree
SR	Slew rate $V_i = \pm 10 \text{ V}$, $R_L = 2 \text{ k}\Omega$, $C_L = 100 \text{ pF}$, unity gain	0.25	0.5		V/ μs
t_r	Rise time $V_i = \pm 20 \text{ mV}$, $R_L = 2 \text{ k}\Omega$, $C_L = 100 \text{ pF}$, unity gain		0.3		μs
K_{ov}	Overshoot $V_i = 20 \text{ mV}$, $R_L = 2 \text{ k}\Omega$, $C_L = 100 \text{ pF}$, unity gain		5		%

reproduced in Table 1, is called “Large signal voltage gain.” This is the open-loop gain I was talking about at the beginning of this article. For the UA741 it is typically equal to 200 V/mV, which means a gain of 200,000. This is high, but not infinite. Once again, it may or may not be enough, depending on your design, but usually this is not a concern for DC signals, we will see the AC case a little later.

OTHER ERRORS

As described in my theoretical presentation above, the output voltage of an op-amp is proportional to the difference of voltage between its two inputs and nothing more. But this isn't exactly the case. What are the sources of errors?

The first one is that the output voltage is also a little dependent on the power supply voltage. This dependency is also specified in the datasheet as “Supply voltage rejection ratio.” As you can see in Table 1, this is 90 dB for the UA741. This supply rejection ratio can be referred either to the output or the input of the op-amp and it should be in the datasheet. Let's assume it is input-related. This means that if the amplifier is configured for a gain of 1 (voltage follower), then the voltage error on the output for a 1-V voltage change on the power supply will be $1 \text{ V} \times 10^{-90/20}$, or 31 μV . But this error would be 100 times higher for a gain of 100, and 3 mV starts to be significant. Supply voltage rejection could in particular be a problem for systems directly powered by an unstable battery voltage. Be sure that this phenomenon could also induce nasty oscillation problems if the power supply

is not properly decoupled.

The other error source, “Common mode rejection ratio,” is also specified to 90 dB for the UA741. It is simply stating that the output voltage will not be the same if the two inputs are respectively at 2 V and 2.001 V, or at 3 V and 3.001 V. It should, as the op-amp just measure their difference, but it isn't because the op-amp is not perfect. So-called instrumentation amplifiers are usually quite good in terms of common mode rejection. For example, the Analog Devices AD8422 has a specified 150-dB CMRR at DC.

RAIL TO RAIL?

Now let's focus on power supply-related characteristics. Of course, each op-amp is specified for a given nominal power supply voltage and will probably die if you exceed its absolute maximum specification. But what about the inputs and outputs? Look one more time at the UA741 specification (see Table 1). The “Input common mode voltage range” parameter is specified as $\pm 12 \text{ V}$ for a $\pm 15\text{-V}$ power supply. What does this mean? Simply that this op-amp will not work properly if the voltage applied on any of its inputs is too close to the -15- or 15-V power voltages. It has to stay at least 3 V away from these upper and lower bounds. Now imagine that you use this UA741 with a unipolar voltage supply coming from a 9-V battery. You will connect GND to its V- power pin and the 9-V source to +V. Then, for optimal performance, you must ensure that the input voltage never goes lower than 3 V (i.e., $0 + 3$), and never higher than 6 V (i.e., $9 - 3$). That's a small usable range! You

TABLE 2

The op-amp datasheet also includes AC characteristics. Once again, this is the venerable UA741.

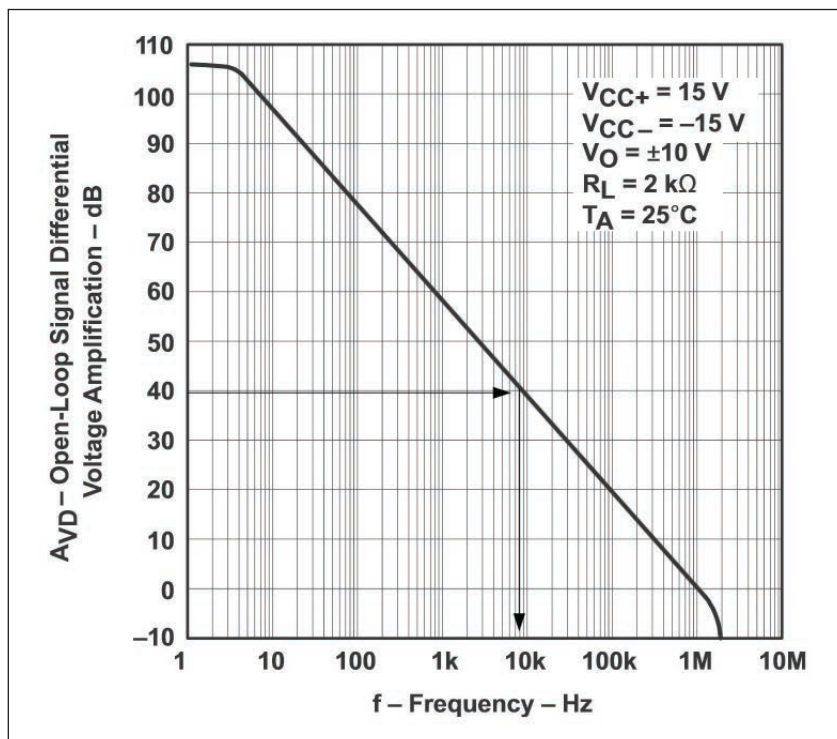


FIGURE 4

This graph shows you the open loop gain versus frequency of this op-amp. If you need to build a 40dB closed-loop amplifier it also shows you that its cut-off frequency will be 10 kHz. (Source: Texas Instruments UA741 datasheet)



circuitcellar.com/ccmaterials

RESOURCES

GAP/R archive, <http://www.philbrickarchive.org>

P. Horowitz and W. Hill, *The Art of Electronics*, 3rd Edition, Cambridge University Press, 2015.

Invention of the Integrated Circuit, https://en.wikipedia.org/wiki/Invention_of_the_integrated_circuit

W. Jung, "Op-Amp History," Analog Devices, www.analog.com

com/library/analogDialogue/archives/39-05/Web_ChH_final.pdf

STMicroelectronics, "UA741: General-Purpose Single Operational Amplifier," 2013, www.st.com/web/en/resource/technical/document/datasheet/CD00001252.pdf

SOURCES

AD8422 Precision instrumentation amplifier Analog Devices | www.analog.com

LT1783 Op-amp and LTC6268 op-amp Linear Technology | www.linear.com

MCP6001 Op-amp Microchip Technology | www.microchip.com

UA741 operational amplifier STMicroelectronics | www.st.com

LMH5401 Amplifier and TLC2652AC op-amp Texas Instruments | www.ti.com

might find my explanations repetitive, but once again, there are operational amplifiers specifically designed for an as large an input voltage range as possible. These so-called rail-to-rail input op-amps accept an input common mode voltage range going from V^- (or GND) up to V^+ . There are even "above the rail" op-amps. One example is the LT1783 from Linear Technologies. This tiny micropower chip accepts input voltages from 0 to 18 V, even if it is powered by a 2.5-V voltage source.

I just talked about the inputs, but the same problem exists on the output side. Our dear friend the UA741 is specified for an "output voltage swing" of ± 13 V with a 2-k Ω load. That means that the output will never come closer than 2 V to V^+ or V^- , even if the op-amp is saturated. I've seen several poorly working designs from designers who forgot to read this line of the spec. Imagine an op-amp powered from a unipolar source ($V^- = \text{GND}$), with its output driving a common-emitter NPN transistor. If the output voltage swing of the op-amp doesn't go lower than 0.6 V, then the transistor will always stay on. You've probably anticipated that so-called rail-to-rail output amplifiers have outputs very close to V^+ and V^- . For example, Microchip Technology MCP6001's output can get as close as 25 mV to both power rails.

FREQUENCY CONCERNS

Thus far, I've covered the key DC parameters of an op-amp. But what about its behavior with AC signals? The key AC specifications for the UA741 are reproduced on **Table 2**. The first one is the well-known "Gain-bandwidth product," or GBP. For the UA741, this is 1 MHz. To understand this parameter, you must know that the open-loop gain of an op-amp is roughly a straight line on a logarithmic scale. The open-loop gain is reduced by 6 dB each time the frequency is doubled (at least for a majority of op-amps). Therefore, there is a frequency at which the open-loop is down to 1. This frequency is by definition the GBP.

The interesting thing is that the same plot is very useful when you design a closed-loop amplifier. Just draw a horizontal line at the closed-loop gain of the amplifier. Determine the frequency at which it intersects the open-loop gain. This will provide the 3-dB low-pass frequency of the amplifier. This is illustrated in **Figure 4**. On this example I used the graph from the UA741 data sheet. I assumed that you want to design a 40-dB amplifier—meaning, a voltage gain of 100. If you aren't sure, calculate $20 \times \log_{10}(100)$ and it will give you 40. Draw a horizontal line at 40 dB, up to the intersection with the open loop

gain, then down to the frequency axis and you find 10 kHz. That means that a UA741-based amplifier with a voltage gain of 100 will not be helpful above 10 kHz. As always, if you need a faster op-amp, use a model designed for high-frequency applications, like the Texas Instruments LMH5401. This monster has an 8-GHz gain-bandwidth option. I guess this would be enough for any reasonable application.

One parameter closely related to the GBP is the “slew rate” of the op-amp. It is 0.5 V/ μ s for the UA741, but 17,500 V/ μ s for the LMH5401. This number indicates how fast the output of the amplifier can swing. For example, 17,500 V/ μ s means that its output can go from 1 to 3 V in about 120 ps. Not too bad.

WRAPPING UP


Selecting an op-amp just because you already used that reference somewhere else is not a good idea. Firstly, check what parameters are critical for the performances of your project. Offset voltage? Bias current? Common-mode voltage range? Rail-to-rail characteristics? Frequency performance? Or maybe noise factor? Define the specifications, prioritize them, and lastly use the good



ABOUT THE AUTHOR


Robert Lacoste lives in France, near Paris. He has 25 years of experience in embedded systems, analog designs, and wireless telecommunications. A prize winner in more than 15 international design contests, in 2003 he started his consulting company, ALCIOM, to share his passion for innovative mixed-signal designs. His book (*Robert Lacoste's The Darker Side*) was published by Elsevier/Newnes in 2009. You can reach him at rlacoste@alciom.com. Don't forget to put “darker side” in the subject line to bypass spam filters.

parametric search tools proposed by all op-amp suppliers on their websites to find the best chip for your application. Just don't forget to add the unit price and minimum order quantity to the search criteria.


I'm out of space, but I have the feeling that I have only scratched the surfaced of the subject. In particular, I didn't cover a key concern related to amplifiers built using a feedback loop: stability. If you have already built an amplifier behaving more like an oscillator, raise your hand. OK, I'll cover that in my next column. 

Circuit Cellar 2014 Digital Archive

With this digital subscription, you have access to all 12 issues of Circuit Cellar 2014 from any computer or tablet at anytime. Readers can explore project ideas, bookmark pages, and make annotations throughout each issue.



Circuit Cellar 2014 CD
CD includes 12 issues of Circuit Cellar in PDF format along with related article code.



Order yours today

cc-webshop.com

FROM THE BENCH

Wireless Home Automation (Part 1)

X-10 and Beyond

COLUMNS

X-10 is considered the Grandfather of Home Automation technologies. Jeff covers the evolution of wireless technology and highlights some exciting new home automation options, such as the ZBPLM.

By Jeff Bachiochi (US)

X-10 Wireless Technology, Inc. (WTI) has officially gone belly-up. For over 40 years, X-10 led the charge in home automation. Unfortunately, the company could wait no longer for the market to catch up with its dream of the future. In 2013, it ceased production. My X-10 appliances still work fine. Sure, there are issues, but they gave us techies stuff to play with when no one else had the vision.

Authinx, Inc., which is the largest distributor of X-10 home automation products in North America, purchased the x10.com domain name in the fall of 2013. It promises to continue improving functionality, reliability, and even aesthetics of the X-10 switches, modules, and controllers. One of most popular items for those interested in experimenting with X-10 is the computer interface. This began with PL513 that had an isolated interface giving your computer the ability to send X-10 commands. The TW523 followed offering reception as well as transmission of these codes.

Over the years, you've seen lots of projects in Circuit Cellar involving the X-10 protocol. Because of the loss of RS232 ports on computers, the X-10 interfaces have been updated to move away from the barebones carrier-initiating interface to include a USB smart interface. However, for experimenters, the computer interface still remains the staple, as they are the easiest to interface with a microcontroller. These products have been

reabeled as PSC04 and PSC05.

Because the X-10 is a protocol that sends information over the existing power lines, data is sent during each half cycle of the 60-Hz (or 50-Hz) 120-VAC waveform. Zero crossings are used as a reference for when to expect the data. Data is sent three times per each half cycle to coincide with a three-phased power system. This assures that data is available after a zero crossing of any phase. A data "1" is indicated by the presence of a 1-ms pulse of 120-kHz carrier imposed on the power line. A lack of carrier during the expected time specifies a data "0." Check out the reference for more information on this protocol.

Past projects included smart interfaces that translate ASCII commands into proper zero crossing timed carrier-initiating interfaces to the PL513 and TW523. These allowed an experimenter to deal with simple commands to control X-10 devices, as opposed to the lower level data timing required by the above interfaces. This month I want to cover the process of sending and receiving X-10 commands using an Android smartphone or tablet over a Bluetooth link. But wait, surely several companies have jumped into the home automation ring by now, so what's new?

LOOK AT ME

While most communication protocols are touting "Look At Me" to control the world, few

manufacturers are stepping up to endorse any specific protocol for the long range. From the cry for an updated and secure power distribution infrastructure, we see both wired and wireless communications used in smart-metering, giving both providers and users demand and cost information. Security companies and cable providers are beginning to offer some level of control and monitoring services. Many are keeping the specifics about their propriety systems under wraps.

From a device point-of-view, thermostats, lighting, door locks, video cameras, and the like usually have some communication logo on the packaging. This gives the user some indication on what it takes to use the device. Meanwhile, system integrators are having nightmares trying to support every possible device and make their control and monitoring software all encompassing.

Back in January 2011, I devoted on column to the Smart Network Access Point (USNAP). The USNAP Alliance was stepping up and trying to create the interface connection between devices, allowing all to communicate with one another via all possible communication medium. I loved this idea and the interface was free and open to all. Since then, this interface has gone through growing pains and the physical interface has changed. USNAP members and other industry stakeholders joined with the Consumer Electronics Association (CEA) to create a new Modular Communications Interface. This new interface combines the previous USNAP standard and the EPRI AC driven version together with additional improvements. While I wish them well, you can obtain the ANSI/CEA-2045 standard only through the CEA website, which costs money.

RF JOINS PL AS CO HOSTS

Towards the end of the 1990s, SmartLabs Technology began developing its own "Linc" series of Power Line (PL) wired products. Their goal was to improve on the reliability of the X-10 protocol. It soon became clear that RF wireless communications could add a much boost to limited repertoire of wired control. Of the wireless communication protocols that had existed, most were either too complex, expensive, or range limited. As an alternative,

SmartLabs partnered with Integration Associates to develop the family of Insteon integrated circuits.

The Insteon RF protocol uses the 915-MHz band. With boundaries of 902 and 928 MHz, the 915-MHz Industrial Scientific Medical (ISM) band is one of the ISM bands set aside for unlicensed operation in the Americas. Together with a power line protocol using 131.65-kHz Binary Phase Shift Keying (BPSK) modulation on the AC line, Insteon RF provides complementary paths which assures complete device coverage.

To be fair, Insteon competes with a number of other systems like Wink, SmartThings, Lowes Iris, and WeMo. As you can see in **Table 1**, most system integrators just make use of other manufacturer's devices. While some offer APIs, most require going through some cloud.

TAKING CONTROL

Past X-10 projects were popular because they showed how to make use of the technology in your own applications. This was due to X-10's willingness to open their technology to those interested in exploiting it in their own way. In an effort to continue the great tradition of the DIYer, Insteon offers a bridge that not only allows contact with the past but also the future. The ZBPLM is a combination of X-10, Insteon, ZigBee interface to an RS-232 connection. While this sounds like the PL513 or TW523 originating from X-10, with the ZBPLM you are not required to build PL transmissions via monitoring the zero-crossings. All communications use a single command set for all the supported interfaces.

Should you choose to use a PC for your application, the included cable with a DE-9 can plug right in, or you may have to use a USB to serial converter available less than \$10 on the web. I picked up this device as it was not USB, so I could easily interface it with a microcontroller. It also has the benefit of providing 300 mA at 12 V for powering your project. It does cost \$120, but considering the interfaces supported and ease of use, I believe it is still cost effective. Note that a USB version is also available.

For this column I will be connecting the ZBPLM to a Bluetooth module through a

TABLE 1

Most home automation systems make use of other manufacturer's technologies and require going through the cloud in order to communicate with a device.

	Insteon	Wink	SmartThings	Lowes Iris	WeMo
Communication	Radio Frequency + Powerline	Radio Frequency	Radio Frequency	Radio Frequency	Radio Frequency
Technology	X-10, Insteon, ZigBee	WiFi, ZigBee, Z-Wave, Bluetooth LE, Lutron ClearConnect	WiFi, ZigBee, Z-Wave	Z-Wave	WiFi
API	Yes	Yes	Yes	n/a	n/a

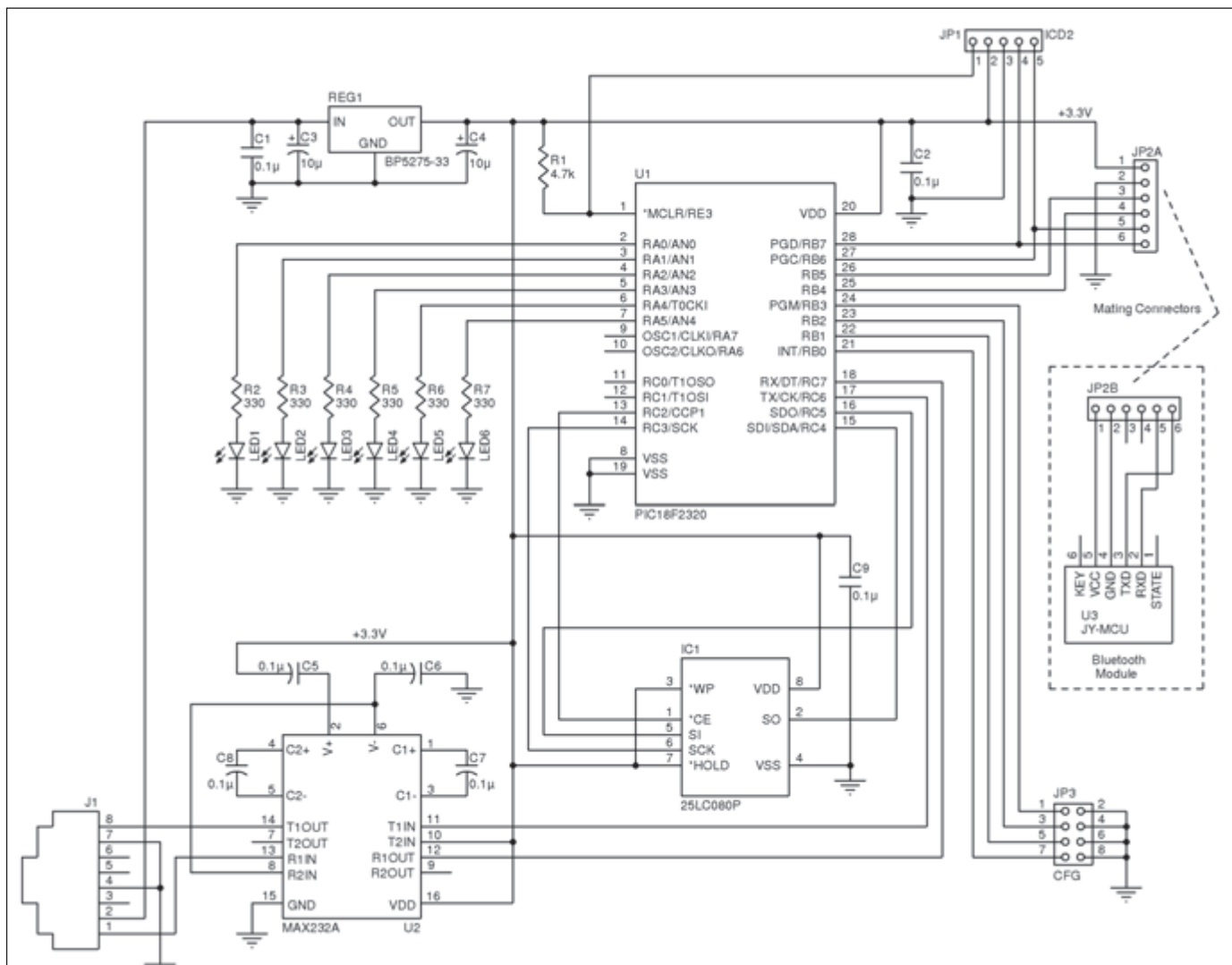


FIGURE 1

The project is based on this schematic, which for all intents and purposes merely passes data back and forth between the ZBPLM module and whatever serial device is hung on the 6-pin serial connector. ZBPLM output is RS-232 so this must be converted to TTL for the microcontroller. This project uses a JY-MCU Bluetooth module for linking wirelessly to any Android device. The microcontroller passes data, initializes the Bluetooth (if necessary), and handles the external EEPROM, which will be used in the future.

PHOTO 1

The Lamp (and Appliance) module use rotary switches to set the House (A-P) and Unit (1-16) codes for each receiver. This becomes the address for the device.



microcontroller. You might think that this could be done without the use of a microcontroller, and you'd be mostly correct. The data to and from the ZBPLM could be shuffled to an application on the opposite end of the Bluetooth; however, I will ultimately want the real smarts to be in this interface. For now the circuit presented **Figure 1** will not be translating anything other than a baud rate change. The same circuitry will allow this application to evolve into a more sophisticated one, as you will see.

To give this month's column a sense of completion, the application will allow just X-10 control from an Android smartphone. For the purpose of this simple illustration, we only need two commands.

```
0x02 0x52 <Raw X-10><X-10 Flag>
X-10 Received
0x02 0x63 <Raw X-10><X-10 Flag>
Send X-10
```

These commands contain Raw X-10 data, which is a byte holding the "Housecode" in

the upper nibble (0-15) and the "UnitCode" or "Function" in the lower nibble (0-15). The X-10 Flag merely defines whether the lower nibble is a "UnitCode" or a "Function". The "Housecode" is always sent as the upper nibble of every Raw X-10 transmission.

X-10 receivers have two little dials on them allowing a user to set a "HouseCode" (A-P) and a "Unit Code" (0-15) for that module (see **Photo 1**). X-10 receiver modules will only pay attention to a "Function" transmission where the last "Unitcode" transmission matches its dial settings. An X-10 transmitter must send a "Unitcode" transmission to wake up those receivers with a matching setting. Any "Function" transmission will be executed by that module, until a new "Unitcode"

transmission is made.

X-10 contains only 16 possible "Housecodes", "Unitcodes", and "Functions" (see **Table 2**). Note that the value (0-15) for "Housecode"=A and "Unitcode"=1 are not in logical order; in this case, both are equal to "6." The Raw X-10 value for sending a "Unitcode" transmission of "Housecode"=A (6) and "Unitcode"=1 (6) is the value 0x66. The Raw X-10 value for sending a "Function" transmission of "Housecode"=A (6) and "Function"=ON (2) is the value 0x62. When the X-10 Flag value is 0x00 the Raw X-10 data is interpreted as a "Unitcode" and when the X-10 Flag value is 0x80 the Raw X-10 data is interpreted as a "Function".

The smartphone application must do three things. First, it must be able to find and

TABLE 2

Here are the nibble values of House, Unit, and Function codes used in X-10 commands. The House and Unit codes are usually set by rotary switches on Lamp and Appliance modules as seen in Photo 1.

COLUMNS

HOUSE CODE	Bit3	Bit2	Bit1	Bit0	Decimal
A	0	1	1	0	6
B	1	1	1	0	14
C	0	0	1	0	2
D	1	0	1	0	10
E	0	0	0	1	1
F	1	0	0	1	9
G	0	1	0	1	5
H	1	1	0	1	13
I	0	1	1	1	7
J	1	1	1	1	15
K	0	0	1	1	3
L	1	0	1	1	11
M	0	0	0	0	0
N	1	0	0	0	8
O	0	1	0	0	4
P	1	1	0	0	12

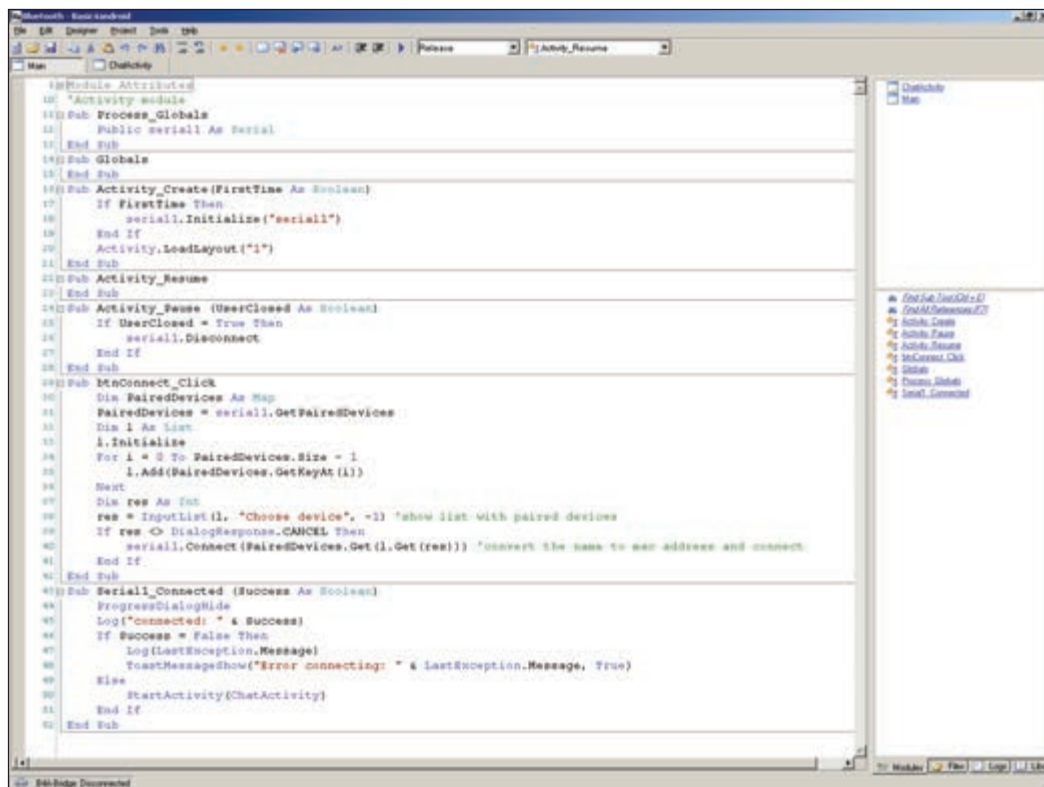
13	0	0	0	0	0
14	1	0	0	0	8
15	0	1	0	0	4
16	1	1	0	0	12

UNIT CODE	Bit3	Bit2	Bit1	Bit0	Decimal
1	0	1	1	0	6
2	1	1	1	0	14
3	0	0	1	0	2
4	1	0	1	0	10
5	0	0	0	1	1
6	1	0	0	1	9
7	0	1	0	1	5
8	1	1	0	1	13
9	0	1	1	1	7
10	1	1	1	1	15
11	0	0	1	1	3
12	1	0	1	1	11

FUNCTION CODE	Bit3	Bit2	Bit1	Bit0	Decimal
ON	0	1	1	0	6
OFF	1	1	1	0	14
DIM	0	0	1	0	2
BRIGHT	1	0	1	0	10
All Lights ON	0	0	0	1	1
All Units OFF	1	0	0	1	9
All Lights OFF	0	1	0	1	5
EXTENDED CODE	1	1	0	1	13
HAIL REQUEST	0	1	1	1	7
HAIL ACK	1	1	1	1	15
EXTENDED CODE	0	0	1	1	3
UNUSED	1	0	1	1	11
EXTENDED CODE	0	0	0	0	0
STATUS "ON"	1	0	0	0	8
STATUS "OFF"	0	1	0	0	4
STATUS REQUEST	1	1	0	0	12

FIGURE 2

The IDE for B4A (Basic for Android) looks like many similar tools—for instance, those familiar with VB6. I started my learning process by going through a sample program that shows how Bluetooth is used to create a chat application between two Android devices.



connect to the Bluetooth device on our circuitry, establishing a serial link. Second, it must provide a way for the user to select a "Housecode", a "Unitcode" and a "Function", and then create a command string containing the appropriate data, and send it over Bluetooth. Finally, there must be some feedback to the user. This is a monitoring of data sent to the smartphone in the form of a response to the requested command or some other X-10 activity. I've dreaded writing smartphone apps in the past because the Bluetooth communications usually

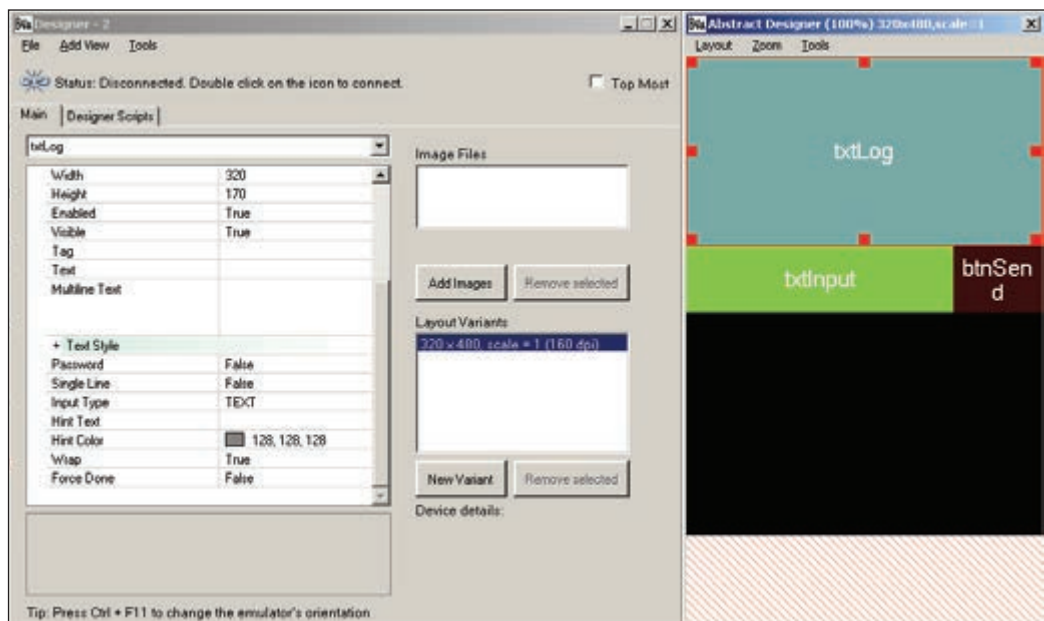
meant more complexity than I bargained for. I found that B4A from Anywhere Software allowed me to accomplish what I wanted with little effort. "B4" versions are available for 'A'ndroid, 'I'OS, and 'j'ava (for Windows, Mac, Linux, and ARM systems). Let's look at the process used for this project.

B4A

Rapid Application Development (RAD) tools for native Android, iOS, and desktop applications are based on Java and require

FIGURE 3

The Designer tool lets you create screen layouts for your application and show the result on a virtual device (shown) or directly on your Android device. Here we see a button and two text boxes used by the "Bluetooth Chat" application.



Verilog HDL

With the right tools

such as this 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.

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.**

cc-webshop.com

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!

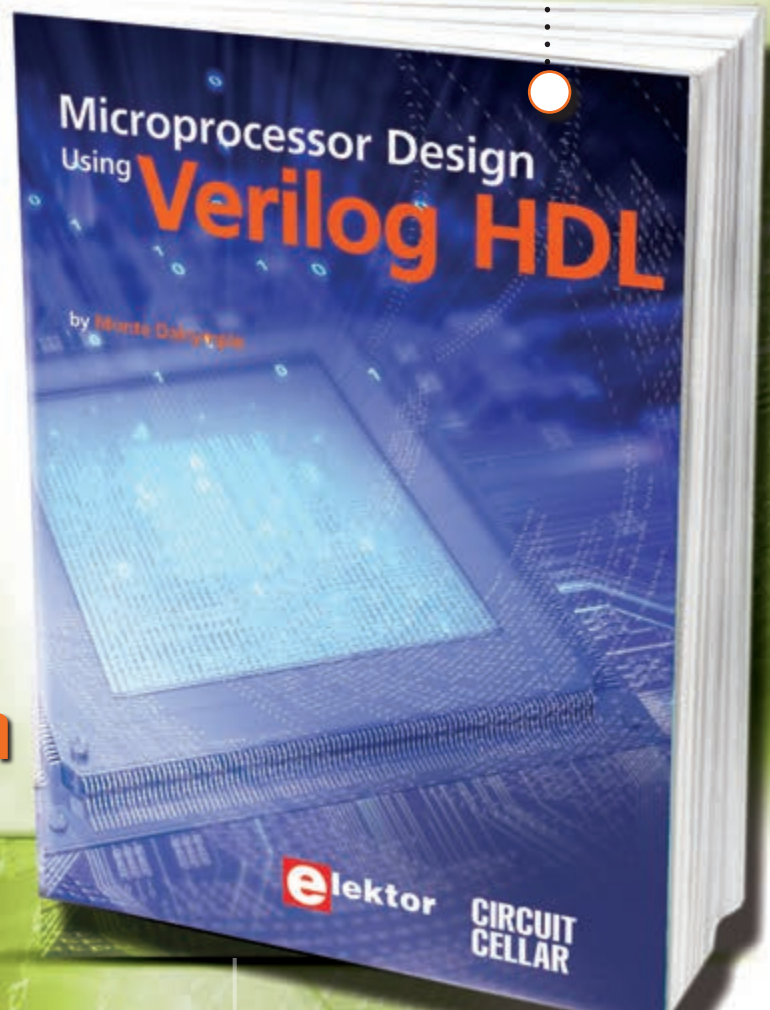
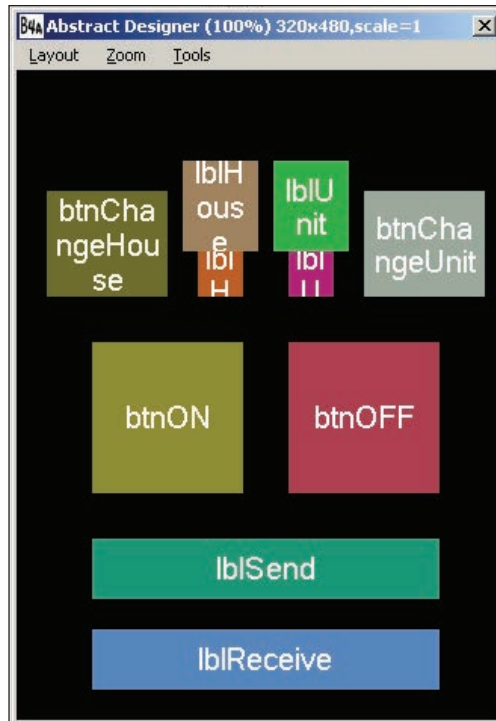


FIGURE 4

For my application I've placed 6 text labels and 4 buttons using the Designer. The two upper buttons will be used to change the House or Unit code, while the central buttons are used to send ON or OFF commands to an X-10 device.



the installation of the Java Development Kit (JDK). As my interest is using my Android smartphone, I choose to use B4A. The standard versions of B4a and B4i are \$59; however, you can try the B4j version for free. Here's a brief overview of the IDE.

I find the best way to help me speed up the learning curve is to choose a prewritten application that is in some way related to what I am trying to accomplish. In this case one of the example applications I found in the website forum is a Bluetooth Chat program. I extracted the contents of this Bluetooth.zip file to my projects folder, then loaded Bluetooth.b4a into the IDE using the File and then Open Source pull-down selection. The IDE (see **Figure 2**) shows the first of two modules "Main" and "Chat activity". These are shown as tabs below the ToolBar as well as at the top of the modules stub on the right side of the page. The left side of the IDE displays the code for the selected module (Main).

Creating a new activity automatically begins five subroutines: `Process_Globals`, `Globals`, `Activity_Create(FirstTime As Boolean)`, `Activity_Resume`, and

`Activity_Pause (UserClosed As Boolean)`. `Process_Globals` are the only "public" variables that can be accessed from all modules in the application. `Globals` are the only "public" variables that can be accessed from within an application module. `Activity_Create` is run when an activity is created—that is, when the application is first launched, the device configuration has changed (e.g., a user rotated the device and the activity was destroyed), or the activity was in the background and the OS decided to destroy it in order to free memory. This sub's purpose might be to load or create the layout and initialize variables declared above. `Activity_Resume` will be called before the activity moves from the background into the foreground either from an `Activity_Create` or an `Activity_Pause`. If the user presses the Home or Back button or another activity wishes to take over the foreground task, the `Activity_Pause` allows any clean up or saving of task data before it is moved to the background (possibly being destroyed).

As you can see in **Figure 2**, two additional subroutines have been added to this module, `btnConnect_Click` and `Serial_Connected (Success As Boolean)`. The `btnConnect_Click` routine is called upon a button press and `Serial_Connected (Success As Boolean)` upon a connection or disconnection of a serial operation. Let's start with the button as it will demonstrate an important tool in the design process. Previously, I mentioned loading a layout or screen. Most applications require a way to display or collect information. This might be text entered or displayed in a window, a graphic, or button and the layout of these can be predetermined using the screen Designer.

The Designer allows the generation of layouts using either the Emulator (a virtual device) or a real device (like your smartphone). With the Designer you can add a number of items: `Button`, `CheckBox`, `EditText`, `ImageView`, `Label`, `ListView`, `Panel`, `ProgressBar`, `RadioButton`, `ScrollView`, `SeekBar`, `Spinner`, `TabHost`, `ToggleButton`, and `WebView` to your layout all of whose properties can be manipulated through the Designer.

In this application the Main module uses a layout containing a single button labeled `btnConnect`. It is loaded when this activity begins. Other than leaving this application the only choice for the user is to click on this button. When clicked we have the stimulus for the `btnConnect_Click` routine. This sub is using a powerful object in the Serial library (previously initialized), which will handle all of the necessary tasks for searching out, pairing with, and connecting to an external Bluetooth device that is within range. The outcome of this



circuitcellar.com/ccmaterials

SOURCES

PIC18F23K22 Microcontroller
Microchip Technology | www.microchip.com

ZBPLM (ZigBee/InsteOn/X-10 Multi-Protocol
Powerline Modem)
Smartenit | www.smartenit.com

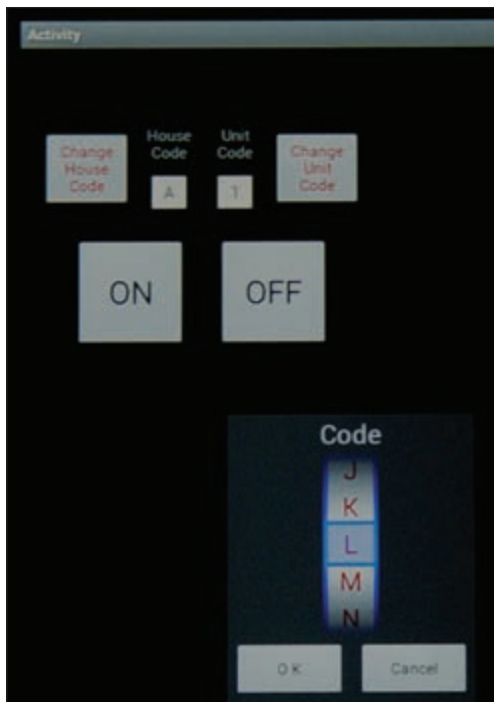
object can be connected or failed. This in turn calls the `Serial1_Connected` routine with the result. When this fails the library function `ToastMessageShow` displays a popup message, if the connection was a success, then a new module `ChatActivity` is called.

Now that the Bluetooth hardware has been initialized, we can make use of the connection after one last detail. The `RandomAccessFile` library offers a way to read from an `InputStream` and write to an `OutputStream` in the background without blocking the main thread by using the object `AsyncStreams`. Besides setting up `AsyncStreams`, `ChatActivity` loads a new screen layout which will provide two edittext boxes and a button. **Figure 3** shows the Designer and virtual device (on the right) with three defined views. The top edittext box, `txtLog`, will display the text messages as sent and received between Bluetooth connected devices. The lower edittext box, `txtInput`, is used to compose a message to send. Finally, the button, `btnSend`, is used to send a complete message. As an alternate to physically pressing `btnSend`, `txtInput` has an `EnterPressed` event that can be used call the `btnSend` object.

BLUETOOTH PLM

You can see that this chat application has most of the Bluetooth code I needed for my application. I needed to substitute my own PLM module for the `ChatActivity` module. Using the Designer, I created a new layout with six label views and four buttons as seen in **Figure 4**. `lblHouse` and `lblUnit` hold the constants 'House Code' and 'Unit Code'. `lblH` and `lblU` display the selected House and Unit Codes. `lblSend` and `lblReceive` display the command codes sent or those received. `btnON` and `btnOFF` will send two commands `<0x02 0x63 HouseCodeUnitCode 0x00>` and `<0x02 0x63 HouseCodeFunctionCode 0x80>` where Function code is 'ON' or 'OFF'. Finally, `btnChangeHouse` and `btnChangeUnit` will invoke the `CLsWheel` class to display and allow user data selection input using a wheel, which is sort of a fancy listbox. This is a pop-up view that can't be placed using the Designer. **Photo 2** shows my Acer 100 tablet displaying the PLM screen once the `btnChangeHouse` is pressed. The scroll wheel shows either the House Code choices (A-P) or Unit Code choices (1-16). Once you scroll the wheel to the appropriate selection, the OK button will exit the class and place the selection appropriately in either `lblW` or `lblU`.

As I noted earlier, an X-10 command requires both a `HouseCode/UnitCode` transmission and a `HouseCodeFunctionCode` transmission. Each successful transmission has a return message. I found it wasn't good



enough to just wait for the return message before transmitting the second transmission. I had to delay the transmission slightly before the second command would be accepted. So I implemented the first transmission upon the button press and set a flag to indicate the second transmission was necessary. When the `AStream_NewData()` event finds a response message and the flag set, the second transmission is sent after a 0.5-s delay. Any message that is received is decoded and displayed in `lblReceive`. It should be noted here that commands sent from, say, a manual controller, will be forwarded to the ZBPLM, so all activity is received.

BLUETOOTH MODULE

I've standardized on a six-pin header for interfacing serial to various devices. For this project, I wired the JY-MCU Bluetooth module (CSR BlueCore-4 chipset) to a mating connector. This way I can plug this into any project using the interface. You can find the JY-MCU on the Internet for less than \$10. I have version 1.05 and this 3.3-V module requires only four connections, power plus TX and RX. It uses an AT-style command set to access both command and data modes for easy connections using SSP protocol.


This project was a no-frills introduction to using an Android device for controlling X-10 modules using the ZBPLM interface. In the next part of this series, I'll go into the two other interfaces available on the ZBPLM, Insteon, and ZigBee. 

PHOTO 2

I chose to use some fancy wheel views for selecting House or Unit codes. This photo of my Acer 100 shows the popup that shows when the 'Change House Code' button has been pressed. The wheel popup is handled by a Class module and called from both Change House and Unit code buttons.



ABOUT THE AUTHOR

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@imagine-thatnow.com or at www.imaginedthatnow.com.

CC SHOP



1

2 CC 2014 CD

2014 was an exciting year for electronics engineers! The continued success of open-source solutions, Internet of Things (IoT) revolutions, and green-energy consciousness has changed the face of embedded design indefinitely. In *Circuit Cellar's* 2014 archive CD, you can find all of these hot topics and gain insight into how experts, as well as your peers, are putting the newest technologies to the test. You'll have access to all articles, schematics, and source code published from January to December 2014.

Item #: CD-018-CC2014

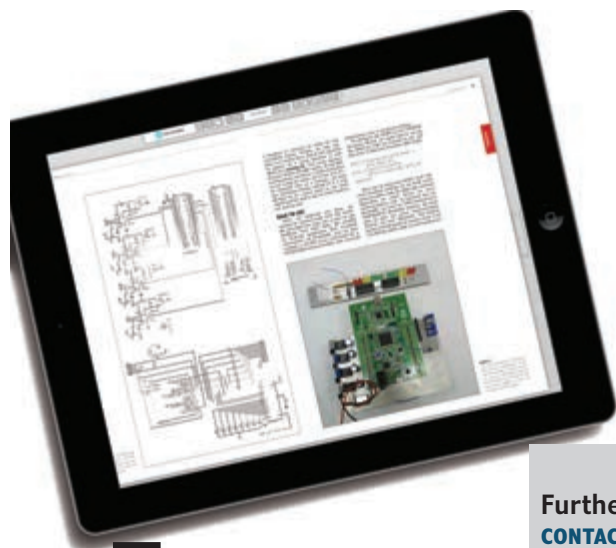
Previous Years Also Available

3 MICROPROCESSOR DESIGN USING VERILOG HDL

After years of experience, Monte Dalrymple has compiled his knowledge of designing embedded architecture and microprocessors into one comprehensive guide for electronics engineers. *Microprocessor Design Using Verilog HDL* provides you with microarchitecture, writing in Verilog, Verilog HDL review, and coding style that enables you to depict, simulate, and synthesize an electronic design on your own.

Author: Monte Dalrymple

Item #: CC-BK-9780963013354



4

1 CC VAULT

CC Vault is a pocket-sized USB that comes fully loaded with every issue of *Circuit Cellar* magazine! This comprehensive archive provides an unparalleled amount of embedded hardware and software design tips, schematics, and source code. CC Vault contains all the trade secrets you need to become a better, more educated electronics engineer!

Item #: CCVAULT



2



3

4 CC 2014 DIGITAL ARCHIVE SUBSCRIPTION

Just when you thought it couldn't get any easier than a thumb drive...you can now access a full year of *Circuit Cellar* from any device connected to the Internet! (2014: 12 issues)

You get all the benefits of a printed copy—bookmark pages, make annotations, and write in the margins—combined with the digital advantages of easy storage, zoom, links, and search features.

Item #: CC-DA-2014

Further information and ordering: www.cc-webshop.com

CONTACT US: Circuit Cellar, Inc. | Phone: 860.289.0800 | E-mail: custservice@circuitcellar.com

The answers to the EQ problems that appeared in *Circuit Cellar* 302 (September 2015).

ANSWER 1—The timing will depend primarily on the capacitive load on each logic gate, which would include both the wiring capacitance and the capacitance of the MOSFET gate(s) you're driving.

For example, the 2N7000 has an input capacitance of 20 pF typical (50 pF max). If your average fanout is 3, plus some wiring capacitance, that gives you a typical load of 100–200 pF. With a 10-k Ω pullup, that gives you an R-C time constant of 1–2 μ s. You'd probably need to allow at least two time constants for one "gate delay" for reliable switching, so we're talking about 2–4 μ s per gate.

To get useful work done, you'll need to allow some maximum number of gate delays per clock period. This will depend on your specific design, but a number like 6 to 10 would be typical. So now we're talking about a clock period of 12–40 μ s, or frequencies in the range of 25–80 kHz.

Switching to a 1-k Ω pullup resistor would allow the frequency to scale up by roughly a factor of 10.

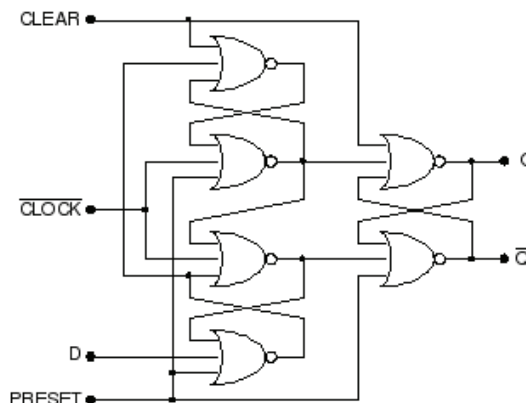
ANSWER 2—You can assume that roughly half of the gates will be active (outputs low) at any given moment, with current passing through their pullup resistors. Each resistor passes $5\text{ V}/10\text{ k}\Omega = 0.5\text{ mA}$, and if there are 1,000 gates, this represents an worst-case current of 0.5A, giving a power consumption of $5\text{ V} \times 0.5\text{ A} = 2.5\text{ W}$. If only about half the gates are active, then the average power will be about 1.25 W.

Switching to a 1-k Ω pullup resistor will raise this average static power consumption to roughly 12.5 W (5 A, or 25 W, worst-case).

TEST YOUR EQ

Contributed by David Tweed

ANSWER 3—Six three-input NOR gates can be used to build a master-slave D flip-flop. Note that the active edge of the clock is the falling edge.



ANSWER 4—The original Cray-1 supercomputer was constructed using a single type of IC for the logic that contained one four-input and one five-input NOR gate. This IC used ECL (emitter-coupled logic) technology and the machine ran with a cycle time of 12.5 ns (80 MHz). About 200,000 gates were required to implement the CPU.



Sign up today and SAVE 50% • Sign up today and SAVE 50% • Sign up today and SAVE 50%

Now offering student SUBSCRIPTIONS!

When textbooks just aren't enough, supplement your study supplies with a subscription to *Circuit Cellar*. From programming to soldering, robotics to Internet and connectivity, *Circuit Cellar* delivers the critical analysis you require to thrive and excel in your electronics engineering courses.

Sign up today and SAVE 50% • Sign up today and SAVE 50% • Sign up today and SAVE 50%

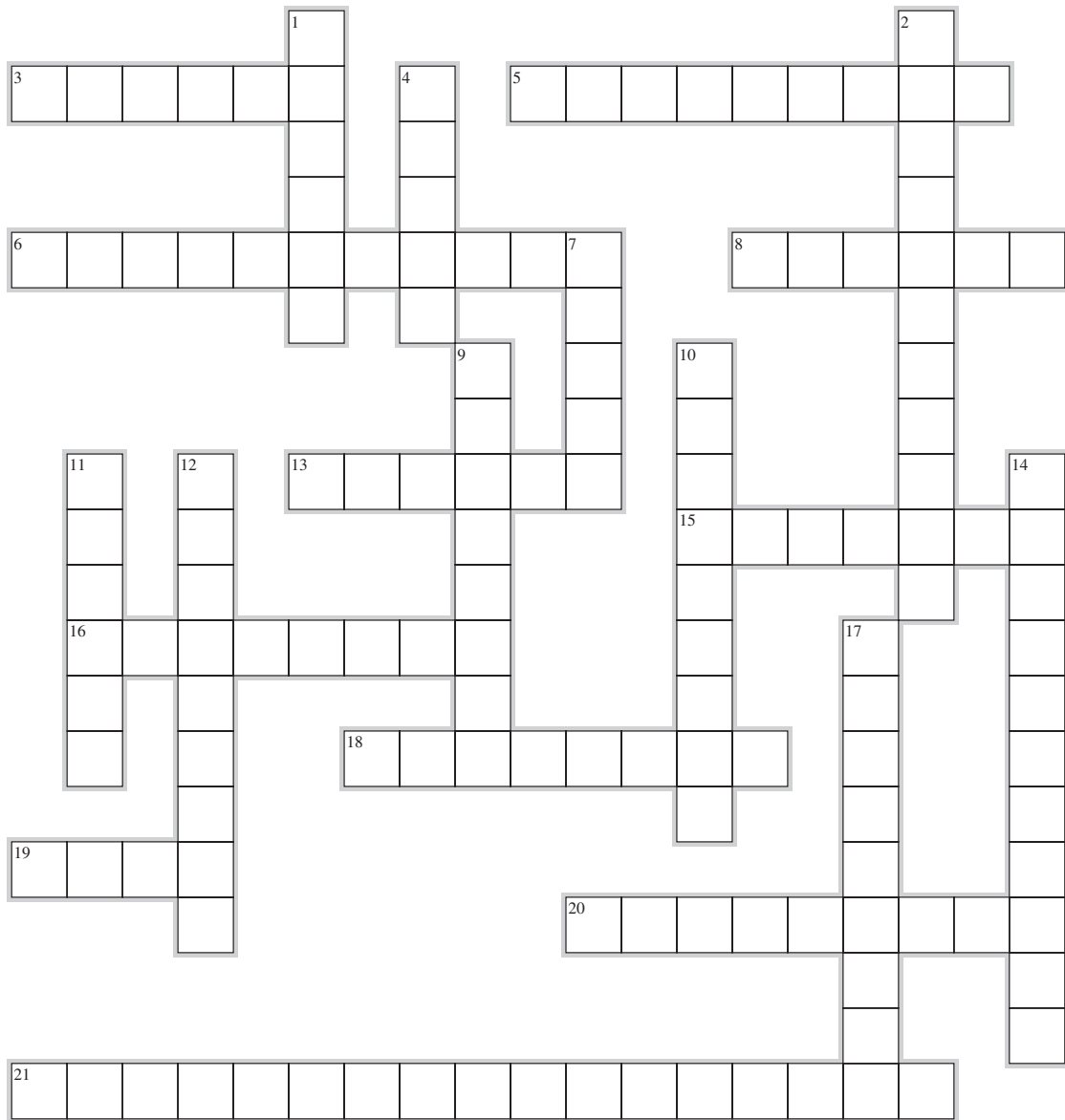
www.circuitcellar.com/subscription



CROSSWORD

OCTOBER 2015

The answers will be available at circuitcellar.com/category/crossword/



ACROSS

3. Discrete particles
5. Box
6. Bending of energy waves
8. Offs and ons
13. Physically coupled to work in unison
15. Vacuum tube with a plate, control grid, screen grid, and cathode
16. Coil
18. DIY
19. Point of maximum electrical polarity
20. Depicts a sequence of operations
21. PoE

DOWN

1. 1 newton/cm²
2. Dissemination of energy
4. One quadrillionth
7. Rechargeable battery
9. 10⁻⁴ micrometer
10. Diode used to convert AC to DC
11. Equal to
12. Soldered permanently
14. Circuit that extracts modulations from an RF signal
17. 180° out of phase

IDEA BOX the directory of PRODUCTS & SERVICES

For current rates, deadlines, and more information contact Peter Wostrel at 978.281.7708 or circuitcellar@smmarketing.us.

\$20 for 5 PCBs

Standard PCB

- 2 layer, 4"x4", FR4(RoHS), 0.063", 1 oz
- 2LPI, Green, Lead free HASL



PCB PCBA

Small to Mass QTY
Instant Quote at:
www.myropcb.com
Or Call 1-888-PCB-MYRO

GHz Bandwidth Sockets for BGA & QFN

Industry's Smallest Footprint

- Pitch - 0.5mm to 1.27mm
- Six different lid options
- BGA - QFN (MG.F)
- Optional 500,000 insertions
- Bandwidth to 40+ GHz
- Heatlinking to 700 watts



RoHS

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

MaxBotix®

High Performance Ultrasonic Rangefinders



Call Carlson today for application specific sensors

Phone: 218-454-0766
Email: info@maxbotix.com
www.maxbotix.com

Connect with engineers seeking your design solutions.

Magazine, e-newsletters, website, advertorials, and more.

RESERVE ADVERTISING SPACE TODAY!

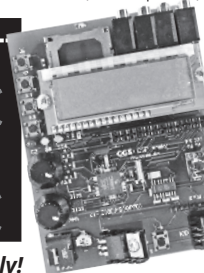
Strategic Media Marketing, LLC
978.281.7708
circuitcellar@smmarketing.us
www.smmarketing.us

DSP Analog on a PIC® MCU

DSP features on an audio processing board

DSP Analog Dev Kit provides line in, line out, microphone in, & amplified audio out (for headphones)

-----SALE-----
H/W Only Kit:
~~\$124~~ ~~\$149~~
Kit with IDE Compiler:
~~\$449~~ ~~\$474~~



Limited Time Only!
www.ccsinfo.com/CC1015
sales@ccsinfo.com
262-522-6500 x 35

CCS


ALL ELECTRONICS CORPORATION

Electronic and Electro-mechanical Devices, Parts and Supplies. Many unique items.

We have what you need for your next project.



www.allelectronics.com
Free 96 page catalog 1-800-826-5432



ASSEMBLY LANGUAGE ESSENTIALS

A matter-of-fact guide that introduces the most fundamental programming language of a processor.

Complete with downloadable Assembler program, important algorithms, and more!

Only \$47.50
www.cc-webshop.com

PIC-SERVO MOTION CONTROL

MOTION CONTROLLERS FOR BRUSH, BRUSHLESS AND STEPPER MOTORS.

- controller chips
- controller boards

www.picservo.com
JEFFREY KERR, LLC

The Future of Commodity Hardware Security and Your Data

By *Alexandrea Mellen*



Alexandrea Mellen is the founder and chief developer at Terrapin Computing, LLC, which makes mobile applications. She presented as a briefing speaker at Black Hat USA 2015 (“Mobile Point of Scam: Attacking the Square Reader”). She also works in engineering sales at The Mellen Company, which manufactures and designs high-temperature lab furnaces. She has previously worked at New Valence Robotics, a 3-D printing company, as well as The Dorm Room Fund, a student-run venture firm. She holds a BS in Computer Engineering from Boston University. During her undergraduate years, she completed research on liquid metal batteries at MIT with Group Sadoway. See alexandreamellen.com for more information.

TECH THE FUTURE

The emergence of the smartphone industry has enabled the commodity hardware market to expand at an astonishing rate. Providers are creating cheap, compact, and widely compatible hardware, which bring about underestimated and unexplored security vulnerabilities. Often, this hardware is coupled with back end and front end software designed to handle data-sensitive applications such as mobile point-of-sale, home security, and health and fitness, among others. Given the personal data passed through these hardware devices and the infancy of much of the market, potential security holes are a unique and growing concern. Hardware providers face many challenges when dealing with these security vulnerabilities, foremost among them being distribution and consequent depreciation issues, and the battle of cost versus security.


An important part of designing a hardware device is being prepared for a straightforward hardware depreciation. However, this can be a thorn in a provider’s side, especially when dealing with widespread production. These companies create on the order of millions of copies of each revision of their hardware. If the hardware has a critical security vulnerability post-distribution, the provider must develop a way to not only deprecate the revision, but also fix the problem and distribute the fix to their customers. A hardware security vulnerability can be very detrimental to companies unless a clever solution through companion software is possible to patch the issue and avoid a hardware recall. In lieu of this, products may require a full recall, which can be messy and ineffective unless the provider has a way to prevent future, malicious use of the insecure previous revision.

Many hardware providers have begun opting out of conventional product payments and have instead turned to subscription or use-based payments. Hence, the provider may charge low prices for the actual hardware, but still maintain high yields, typically through back end or front end companion software. For example, Arlo creates a home security camera with a feature that allows users to save videos through their cloud service and view

the videos on their smartphone. The price of the camera (their hardware) is mid-range when measured against their competitors, but they charge a monthly fee for extra cloud storage. This enables Arlo to have a continual source of income beyond their hardware product. The hardware can be seen as a hook to a more stable source of income, so long as consumers continue to use their products. For this reason, it is critical that providers minimize costs of their hardware, even down to a single dollar—especially given their large-scale production. Unfortunately, the cost of the hardware is typically directly related to the security of the system. For example, a recent vulnerability found by me and my colleagues in the latest model Square Reader is the ability to convert the Reader to a credit card skimmer via a hardware encryption bypass. This vulnerability was possible due to the placement of the encryption chip on a ribbon cable offset from the magnetic head.

If the encryption chip and magnetic head had been mounted to the Reader as an assembly, the attack would not have been possible. However, there is a drastic difference in the cost, on the order of several dollars per part, and therefore security was sacrificed for the bottom line. This is the kind of challenging decision every hardware company has to make in order to meet their business metrics, and often it can be difficult to find a middle ground where security is not sacrificed for expense.

New commodity hardware will continue to integrate into our personal lives and personal data as it becomes cheaper,

more compact, and universally compatible. For these reasons, commodity hardware continues to present undetermined and intriguing security vulnerabilities. Concurrently, hardware providers confront these demanding security challenges unique to their industry. They face design issues for proper hardware depreciation due to massive distribution, and they play a constant tug-of-war between cost constraints and security, which typically ends with a less secure device. These potential security holes will remain a concern so long as the smartphone industry and commodity hardware market advance. 



The Square Reader’s encryption chip is located in the bottom right-hand corner instead of on the magnetic head. This drastically reduces the cost of the device.



*Alexandrea Mellen's
Black Hat 2015 Brief*

CC Vault

Unlock the power of embedded design.



This pocket-sized vault comes fully loaded with every issue of Circuit Cellar magazine and serves as an unparalleled resource for embedded hardware and software design tips, schematics, and source code.

From green energy design to 'Net-enabled devices, maximizing power to minimizing footprint, CC Vault* contains all the trade secrets you need to become a better, more educated electronics engineer.

A vault of need-to-know information in the fields of embedded hardware, embedded software, and computer applications

*CC Vault is a 16-GB USB drive.



Order yours today! cc-webshop.com

LONG LEAD TIMES FOR PCB ASSEMBLY IS HISTORY!!!

R.P.A.S

Rapid Prototype Assembly Service

Is The **FUTURE!**

STENCIL-LESS JET PRINTING

Our Stencil-less jet printer eliminates the need for stencils. This allows us to turn your assemblies around in a day! No more waiting 1-3 days for stencils!

MULTI-HEAD SMT EQUIPMENT

State of the art equipment allows quick changeovers, eliminating downtime needed to swap parts for jobs in line.

QUOTING REVOLUTION

Our proprietary Quoting software gives full turnkey quotes (Boards, Components and Labor) online within minutes...not hours, not days.

Multiple day lead times are History! With our capabilities, we can turn around assemblies in 24 Hours*

PCBs

ADVERTISED SPECIALS

\$10

PER PIECE

Try Imagineering's Triple Play
and experience the Future!

Industry leader in Printed Circuit Board Manufacturing
and Assembly Services since 1985.

ASSEMBLY

\$250 UP TO
250 PLACEMENTS
PER ORDER

Maximum 10 pieces

**COMPONENTS
& PCBs**

FREE UP TO
\$500
VALUE



imagineering inc

☎ 847-806-0003

🌐 www.PCBnet.com

✉ sales@pcbnet.com

*Does not apply to this offer