



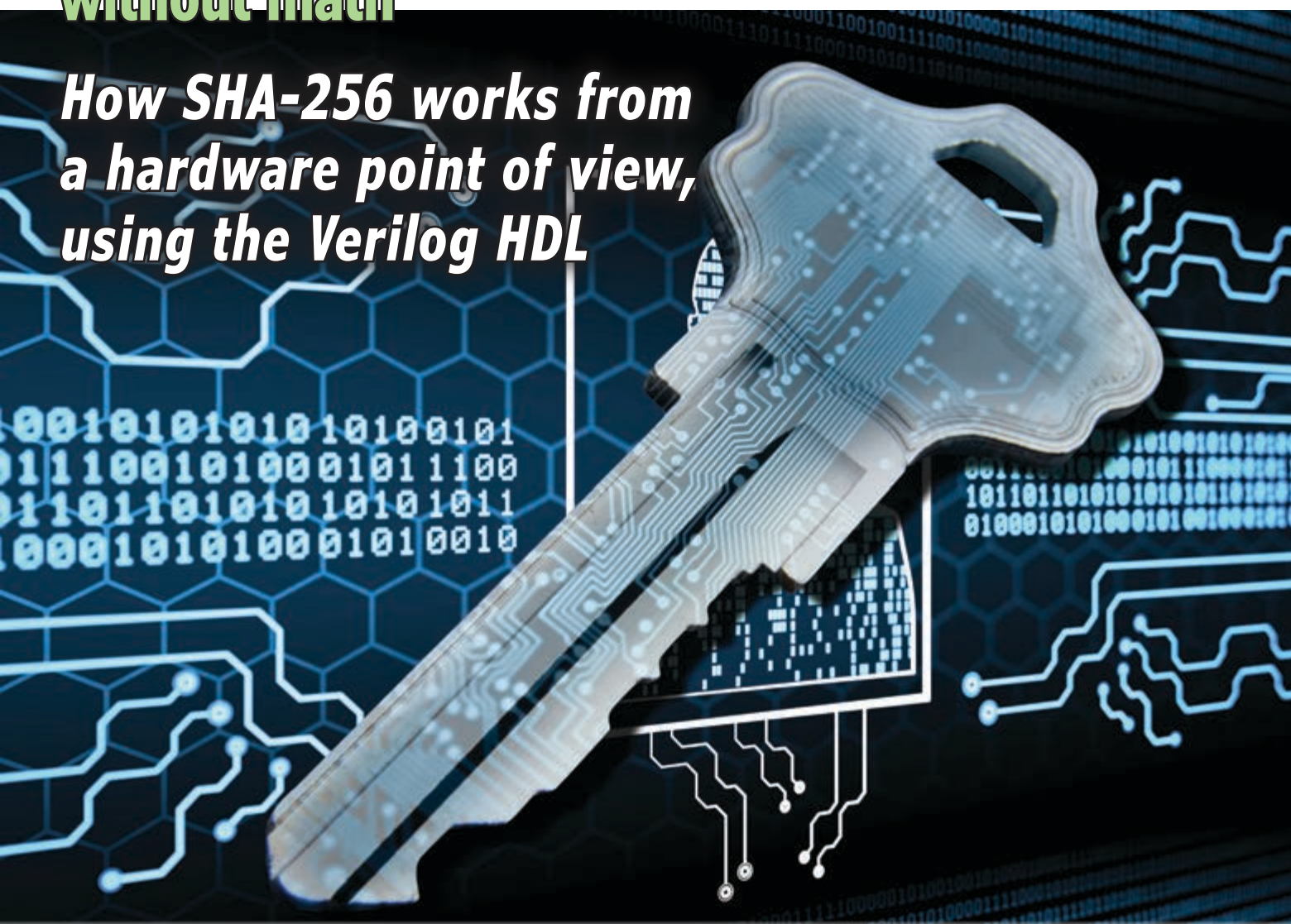
circuit cellar

A guide to the

SECURE HASH STANDARD

without math

*How SHA-256 works from
a hardware point of view,
using the Verilog HDL*



■ Editors' Picks: Embedded Solutions ■ Battery-Powered MCU Circuits |
Acoustic Recording App (Part 3) ■ Big Data Analysis in the Cloud |
Transformers 101 | Build a Radiation Monitor | MCU-Based Wiegand Data
Display System ■ The Future of Simulation & Systems Modeling

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), Polymide, 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

COME SEE US AT PCB WEST BOOTH 509

SUPERIOR **EMBEDDED** SOLUTIONS



DESIGN YOUR SOLUTION TODAY
CALL 480-837-5200

www.embeddedARM.com



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



Available w/ TS-8550
PC/104 Development Kit



TS-TPC-7990 Touch Panel PC

**COMING
SOON!**

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



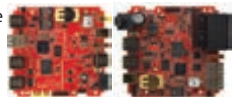
TS-7970 Single Board Computer

NEW!

Embedded Computer Version
of the TS-4900 i.MX6 CoM with
Dual Ethernet, Rugged Connector

- 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, 2x Gigabit Ethernet, 1x PCI Express Bus
- 1x microSD Socket, 1x SATA II, 4x USB Host, 1x USB OTG
- Daughter card interface for cell modem and more
- 40 °C to 85 °C Industrial Temperature Range
- HDMI, LVDS, and Audio In/Out Connections
- Runs Linux, Android, QNX, Windows

Starting at
\$169
Qty 100
\$214
Qty 1



Also available in this form factor are the
TS-7670 and TS-7680 with 454 MHz CPU



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



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

Issue 302 September 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

COVER PHOTOGRAPHY

Chris Rakoczy, www.rakoczyphoto.com

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®.

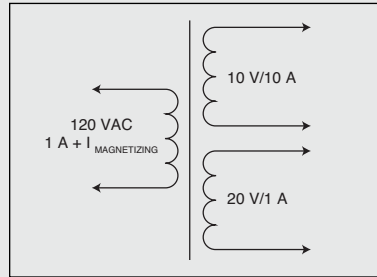
The information provided by Circuit Cellar® is for educational purposes. Circuit Cellar® makes no claims or warrants that readers have a right to build things based upon these ideas under patent or other relevant intellectual property law in their jurisdiction, or that readers have a right to construct 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

DATA GATHERING AND ANALYSIS

Data acquisition is essential to most, if not all, electronics applications. In this issue, we present a variety of articles about projects for which instantaneous information gathering is paramount.

Many consumer applications that store and manage data (e.g., MP3 players) run on battery power. In the first part of the series, "Running on Battery," Stuart Ball explains how to protect against reverse battery voltage when powering a small MCU circuit with batteries (p. 26).



Transformer balancing by ampere turns

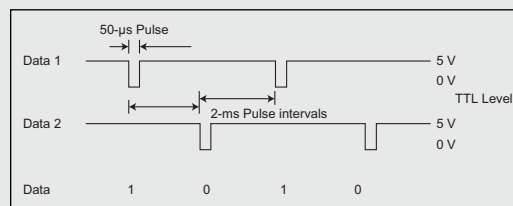
Mobile apps essentially serve as handy data acquisition tools. In the third part of their series, "Sound Ecology and Acoustic Health," Adrien Gaspard and Mike Smith present a quantitative application for their WAT_AN_APP app (p. 32).

Do you think current practices in big data analytics are sustainable? On page 52, Ata Turk addresses the topic of big data analysis in the cloud.

Turn to page 60 to read about an interesting DIY system for monitoring radiation levels. Ed Nisley built the small radiation monitor around an Arduino.

This issue also features articles on several other key electrical engineering topics. Let's review.

On page 14, Monte Dalrymple covers the Secure Hash Standard. He presents a design that implements the complete byte-oriented SHA-256 variant of the standard.



The typical Wiegand output for a Wiegand device

Turn to page 56 for a quick read about transformer basics. George Novacek covers their essential characteristics and reviews a typical power transformer.

In "Wiegand World" on page 68, Jeff Bachiochi introduces the physical layer and protocol. He also details the process of building a microcontroller-based Wiegand data display system.

We conclude the issue with R. Scott Coppersmith's essay, "The Future of Engineering Research and Environment Systems Modeling" (p. 80). He presents his thoughts on the future of application simulation and electronic system modeling.

C. J. Abate

cabate@circuitcellar.com

THE TEAM

EDITOR-IN-CHIEF

C. J. Abate

ART DIRECTOR

KC Prescott

ADVERTISING COORDINATOR

Kim Hopkins

PRESIDENT

Hugo Van haecke

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 Plane),

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

Accutrace	C2	Jeffery Kerr, LLC	79
All Electronics Corp.	79	Lemos International	31
AP Circuits	7	MaxBotix, Inc.	79
ControlByWeb.com	17	microEngineering Labs, Inc.	79
Custom Computer Services	79	MyRO Electronic Control Devices, Inc.	79
Elprotronic, Inc.	7	NetBurner, Inc.	13, 39
EMAC, Inc.	9	PCB West Conference & Exhibit	29
Front Panel Express	31	Pico Technology	35
HuMANDATA, Ltd.	9	Saelig Co., Inc.	63
IAR Systems	11	Scidyne Corp.	79
Imagineering, Inc.	C4	Technologic Systems	1
Ironwood Electronics	79		

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

SEPTEMBER 2015 • ISSUE 302



DATA ACQUISITION

```

/*****
/* Ch function
/*****
function [31:0] ch:
input [31:0] x, y, z:
begin
ch = (x & y) | (~x & z);
end
endfunction

/*****
/* Maj function
/*****
function [31:0] maj:
input [31:0] x, y, z:
begin
maj = (x & y) | (x & z) | (y & z);
end
endfunction

/*****
/* Big Sigma 0 function
/*****
function [31:0] bsigma_0:
input [31:0] x:
begin
bsigma_0 = {x[1:0], x[31:2]} ^ {x[12:0], x[31:13]} ^ {x[21:0], x[31:22]};
end
endfunction

/*****
/* Big Sigma 1 function
/*****
function [31:0] bsigma_1:
input [31:0] x:
begin
bsigma_1 = {x[5:0], x[31:6]} ^ {x[10:0], x[31:11]} ^ {x[24:0], x[31:25]};
end
endfunction

/*****
/* Little Sigma 0 function
/*****
function [31:0] sigma_0:
input [31:0] x:
begin
sigma_0 = {x[6:0], x[31:7]} ^ {x[17:0], x[31:18]} ^ {3'b000, x[31:3]};
end
endfunction

/*****
/* Little Sigma 1 function
/*****
function [31:0] sigma_1:
input [31:0] x:
begin
sigma_1 = {x[16:0], x[31:17]} ^ {x[18:0], x[31:19]} ^ {10'b0000000000, x[31:10]};
end
endfunction

```

AN OVERVIEW OF THE SECURE HASH STANDARD

INDUSTRY & ENTERPRISE

06 : PRODUCT NEWS

09 : PARTNER Q&A

The Great American Electronics Hobbyist Census
An Interview with Greg Harris (Jameco)

CC COMMUNITY

10 : EDITORS' PICKS

Embedded Solutions

Several of the *Circuit Cellar* team's favorite articles on embedded solutions

FEATURES

14 : Secure Hash Standard

By Monte Dalrymple

A look at the Secure Hash Standard from a hardware point of view

26 : Running on Battery (Part 1)

Battery-Powered Microcontroller Circuits

By Stuart Ball

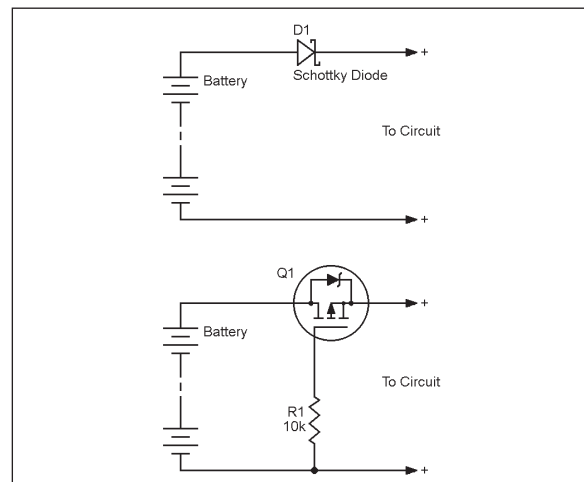
Protect against reverse battery voltage when powering a small MCU circuit with batteries

32 : Sound Ecology and Acoustic Health (Part 3)

A Quantitative Application for WAT_AN_APP

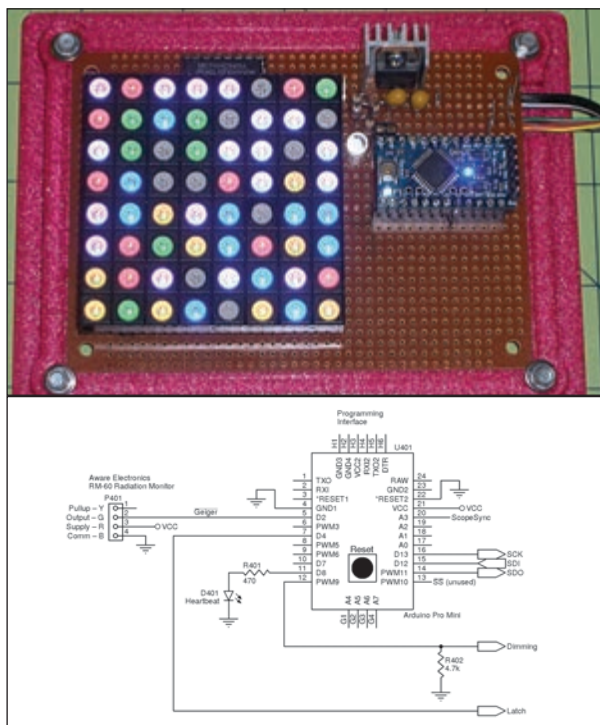
By Adrien Gaspard & Mike Smith

An audio record and analysis update for the app



PROTECT AN MCU CIRCUIT POWERED WITH BATTERIES

CONTENTS



ARDUINO-BASED DESKTOP RADIATION MONITOR



WIEGAND DATA DISPLAY PROJECT

CC REBOOT

44 : Power Over Ethernet Solutions

By Eddie Insam

A comprehensive introduction to powering devices over Ethernet

COLUMNS

52 : GREEN COMPUTING

Sustainable Big Data Analysis in the Cloud

By Ata Turk

Are current practices in big data analytics sustainable?

56 : THE CONSUMMATE ENGINEER

Transformers 101 (Part 1)

Essential Characteristics

By George Novacek

An introduction to transformers and a review of a typical power transformer

60 : ABOVE THE GROUND PLANE

Random LED Dots

By Ed Nisley

A DIY desktop radiation monitor built with an Arduino and a spare 8 x 8 RGB LED matrix

68 : FROM THE BENCH

Wiegand World

An Introduction to the Physical Layer & Protocol

By Jeff Bachiochi

Build a microcontroller-based Wiegand data display system

TESTS & CHALLENGES

77 : TEST YOUR EQ

78 : CROSSWORD

TECH THE FUTURE

80 : The Future of Engineering Research and Environment Sysetms Modeling

By R. Scott Coppersmith

Thoughts on the future of simulation and the importance of systems modeling



RESEARCH PROJECTS & SYSTEM MODELING

PRODUCT NEWS

TRACE32 SUPPORTS SPANSION HYPERFLASH MEMORY

Lauterbach recently announced its support for the Spansion HyperFlash Memory with the TRACE32 tools. HyperBus Interface was introduced by Spansion in 2014 as an improvement on today's low pin count memory interfaces and has been broadly implemented by the system-on-chip (SoC) manufactures.

HyperFlash Memory is based on the HyperBus interface and provides the important characteristics such as low latency, high read throughput, and space efficiency. TRACE32 tools support the HyperFlash memory with the intuitive, fast, and flexible Flash Programming feature that also provides you with control of reading, displaying, and erasing the content of the flash memory. The content is displayed in a standard hex dump, which allows the contents to be checked quickly. The tool supports the pairing of HyperFlash memory with the HyperBus interface and also with the ordinary Quad SPI controller.



Lauterbach | www.lauterbach.com

UNIVERSAL TRIGGER AND DECODER OPTION FOR R&S DIGITAL OSCILLOSCOPES

Rohde & Schwarz has expanded its range of trigger and decoder options for the R&S RTO and R&S RTE digital oscilloscopes. With the R&S RTx-K50, the oscilloscopes help you debug serial protocols that employ Manchester or NRZ coding. The option can be used with a variety of standardized buses (e.g., PROFIBUS, DALI, or MVB) as well as with proprietary serial protocols. Developers of products that use these types of interfaces can easily find implementation errors and so test and release their designs more quickly.

The option, which covers data rates of up to 5 Gbps, supports up to 50 different telegram formats, while the format of the serial bus can be configured flexibly. You can define your own preamble, frame ID, data, CRC and other telegram fields. Protocol decoding also takes Manchester code violations into account.

High acquisition rates and minimal blind times are provided by the hardware-based trigger implementation on the oscilloscopes. You can trigger on telegram and data content with the R&S RTx-K50 option. The decoded protocol content is displayed in an easy-to-read, color-coded format. Time correlation with the analog signal makes it easy to identify faults caused by signal integrity problems. A tabular list of the protocol contents is also provided. The standard mask test with up to 600,000 tests per second makes it possible to check the signal quality faster with an eye diagram than with any other solution. In addition, both oscilloscope series from Rohde & Schwarz support the option of decoding up to four different serial protocols in parallel.

Rohde & Schwarz | www.rohde-schwarz.com



PRECISION SET & READBACK PMBUS-COMPATIBLE UMODULE REGULATOR

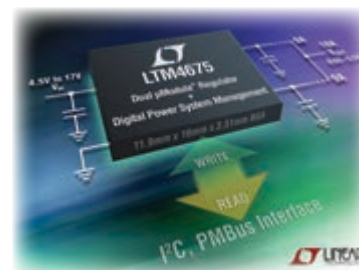
Linear Technology recently announced the LTM4675 dual 9-A or single 18-A, μ Module (micromodule) step-down DC/DC regulator with PMBus serial digital interface. It comes in a 11.9 mm \times 16 mm \times 3.51 mm BGA package. The I²C-based interface enables you to manage a system's power condition and consumption. Calibrated and guaranteed from -40°C to 125°C , output DC voltage accuracy is $\pm 0.5\%$ over line and load regulation, and load current readback accuracy is $\pm 2.5\%$ maximum.

The LTM4675 features EEPROM, power MOSFETs, inductors and supporting components. It is drop-in pin-compatible with the larger package (16 mm \times 16 mm BGA) higher power dual 13A LTM4676A, eliminating layout changes so that system designers can easily switch between the devices during the prototype phase. This eliminates redesign of power circuits if power requirements change during board prototyping. The LTM4675 has applications in optical transport systems, datacom and telecom switches and routers, industrial test equipment, robotics, RAID and enterprise systems where energy costs, cooling and maintenance are critical and must be continuously and precisely measured.

The LTM4675 operates from a 4.5-to-17-V input supply and steps down VIN to two outputs ranging from 0.5 to 5.5 V. Two channels can current share to provide up to 18 A (i.e., 9 A + 9 A as one output). Power-up turn-on time is 70 ms. To evaluate the performance of the LTM4675, the free LTpowerPlay GUI-based development system is available for download, and a USB-to-PMBus converter and demo kit are available.

The LTM4675 internal operating temperature range is from -40°C to 125°C . It costs \$24 in 1,000-piece units.

Linear Technology | www.linear.com



PRODUCT NEWS

HIGH-SPEED, CONDITIONED MEASUREMENTS WITH CHANNEL-TO-CHANNEL ISOLATION

Measurement Computing Corp. recently announced the release of the SC-1608 Series of USB and Ethernet data acquisition devices. The series features analog signal conditioning that enables you to measure voltage, thermocouple, RTD, strain, frequency, and current. Isolated analog output and solid-state relays make it a good solution for systems requiring flexible conditioning and low cost per channel.

There are four devices in the SC-1608 Series with sample rates up to 500 ksp/s. Each device accommodates up to eight 8B isolated analog signal conditioning modules and eight solid state relay modules.



Microsoft Windows software options for the SC-1608 include DAQami and TracerDAQ to display and log data, along with comprehensive support for C, C++, C#, Visual Basic, and Visual Basic .NET. Support is also included for DASyLab

and NI LabVIEW. UL for Android provides programming support for Android devices. Open-source Linux drivers are also available. The SC-1608 Series costs \$999.

Measurement Computing Corp. | www.mccdaq.com

RED EXPERT ONLINE DESIGN TOOL FOR PRECISE AC LOSS CALCULATION

Würth Elektronik recently published RED EXPERT, a new online tool you can use to simulate power inductors. With just a few clicks, you can select the power inductors and calculate the complete AC losses. RED EXPERT enables extremely precise loss calculation because it is not based on the known Steinmetz models with sinusoidal excitation. Instead, it is derived and validated from measurements of the power inductors in a switching controller setup. The losses determined with RED EXPERT are based on current and voltage waveforms typical in applications. Besides the core and winding losses, they also include the losses arising from the specific geometries of the inductance, such as the air gap.



Particular highlights of the RED EXPERT AC loss model are the range of duty cycles supported from 10% to 90% and the switching frequency range of 50 kHz to 5 MHz. This gives the RED EXPERT AC loss model a previously unattained precision.

RED EXPERT is freely available in German, English, Spanish, Japanese, Russian, and Chinese at www.we-online.com/redexpert.

Würth Elektronik | www.we-online.com

INDUSTRY & ENTERPRISE

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

MSP-GANG

USB-FPA

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

Programmers

USB-FPA-1	JTAG / SBW or BSL	MSP430Fxx
USB-FPA-2		MSP430Fxx
USB-FPA-3		MSP430Fxx
USB-FPA-8		MSP430Fxx

Elprotronic Incorporated
www.elprotronic.com

AP CIRCUITS
PCB Fabrication Since 1984

As low as...
\$9.95
each!

Two Boards
Two Layers
Two Masks
One Legend

Unmasked boards ship next day!

www.apcircuits.com

VISA MasterCard PayPal IPC MEMBER ASSOCIATION CONNECTING ELECTRONICS INDUSTRIES BBB

PARTNER Q&A



The Great American Electronics Hobbyist Census

An Interview with Greg Harris (VP, Marketing & Sales, Jameco)

A few months ago, Jameco Electronics surveyed over 1,600 hardcore US-based electronics enthusiasts about their hobbies, technical interests, projects, thoughts on the future of DIY, and much more. We recently asked Jameco's Greg Harris to tell us about the census and provide his thoughts on the results.

CIRCUIT CELLAR: Why did Jameco conduct the Great American Electronics Hobbyist Census?

HARRIS: Jameco has been selling to electronics hobbyists for over 40 years and while we know a lot about their projects and the components, tools and supplies they use, we wanted to better understand who they were and what this hobby really means to them.

The one thing that stood out was that while most treat a hobby as a casual activity, electronics hobbyists take it very seriously. In fact, we probably shouldn't even use the word "hobby." It's quite clear that this is something central to their lives. Many told us that this is something they think about every day. Some see it as a necessary exercise to keep the mind sharp, others described the unmatched sense of accomplishment they get from going from an idea or concept to something tangible. We can't leave out the added thrill of being able to push a button and watch lights blink.

CIRCUIT CELLAR: Tell us a bit about your methodology for the survey.

HARRIS: We ship hundreds of thousands of orders to hobbyists every year, but we didn't want to talk to causal hobbyists, so we culled our database to identify the most active hobbyists. Our goal was to get a true sense of the hobbyist from its most serious practitioners. We ultimately settled on a list of 10,000 individuals. We were a bit surprised that an email survey of this length could generate a 17% response rate but once we understood how much passion was rolled into this hobby, the incredibly high response rate made perfect sense.

CIRCUIT CELLAR: Share with our readers two or three of the most interesting results. What was most surprising?

HARRIS: We were very surprised we didn't have

a larger female response. While 19% of electrical engineering students are female, only 2% of serious hobbyists are women.

Electronics hobbyists are dripping with education, but surprisingly not as many studied engineering as we thought. Hobbyists are nearly twice as educated as average Americans with almost two-thirds graduating from a four-year college and an additional third completing a graduate degree. Yet the largest segment were self-taught and only about one-third had a formal degree in electrical engineering. We've concluded the hobby is for intellectuals, yet you don't need a degree to get hooked.

Another highlight that blew us away was that 77% of respondents reported blowing something up on accident while in pursuit of this hobby. That might dissuade some, but we sense that it had the opposite impact on true electronics hobbyists, because 33% also confessed to blowing something up on purpose.

CIRCUIT CELLAR: The respondents said that the microprocessor—not the PC or Internet—is the most important electronic invention of the past 50 years. Did this surprise the Jameco team? Why or why not?

HARRIS: We've seen sales of microprocessors explode in a very short period of time. In fact, there was a time not too long ago when experts predicted the demise of electronics as a hobby. Electronic components were getting smaller and it was increasingly difficult to complete projects without very expensive equipment. Yet in the last few years the microprocessor almost singlehandedly put new capabilities in the hands of hobbyists and that fueled a resurgence in the hobby. No, we weren't surprised that hobbyists selected microprocessors as the most important invention of the last 50 years. In fact, a few

PARTNER Q&A

Learn more about the Jameco census: <http://bit.ly/1LRuXi5>


suggested it might be the most important invention for the next 50 years as well.

CIRCUIT CELLAR: What's Jameco's biggest takeaway from the survey?

HARRIS: A hobby is something extra in life. For electronics hobbyists it's clear that much of their life revolves around electronics. The shows that everything from what they prefer to read to how they exercise their brain to how they get their thrills is frequently tied to this hobby. As one hobbyists described "traveling through the forest of frustration," to ultimately find a working solution, builds real character and marketable skills. This isn't just a hobby, it defines hobbyists.

This is consistent with what Jameco has seen since we asked if any of our readers would like to contribute to the newsletter. We've been flooded with a steady stream of contributions ever since, celebrating huge failures, big wins, unique projects and everything in between. It became clear that hobbyists like to interact with one another and that this is a community wishing to be as connected as the circuits they design.

CIRCUIT CELLAR: How will the survey results influence Jameco's future product offerings? What can electronics enthusiasts and engineers expect from Jameco moving forward?

HARRIS: We didn't conduct this study to drive product strategy, but rather to better understand our customers. We've been sharing these results with the manufacturers whose products we distribute and they have been most interested in the strong indication that this hobby is staged for an increased growth. Over half of respondents predicted growth in the number of projects that they personally will do while only about one in seven predicted doing fewer projects. Add to that the growth of the number of people engaging in this hobby and it's clear to us that many component manufacturers will be investing in hobbyist oriented products in the years to come. That wasn't the case five years ago. Hobbyists can expect Jameco to continue to provide a wide range of kits, hard to find components and other necessities to keep their tinkering going strong; not to mention a community where they can converge. 

HUMANDATA

FPGA Boards from JAPAN

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





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





PLCC68 Series

- Designed for 68-pin IC socket
- Very small size (25.3 x 25.3 [mm])
- 50 I/Os (External clock inputs are available)
- 3.3V single power supply operation

Find our various products@amazon!

Over 100 varieties of reliable FPGA boards are available at : www.hdl.co.jp/CC1509

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





EMAC, inc.

EQUIPMENT MONITOR AND CONTROL

Our Products Make Your Products Better™

618-529-4525 • info@emacinc.com • www.emacinc.com

30

YEARS OF EMBEDDED SOLUTIONS

EDITORS' PICKS

Embedded Solutions

A Bootloader for Blackfin

By David Tweed (Circuit Cellar 217, 2008)

David designed a two-stage bootloader that allows application firmware to be updated in the field to support bug fixes and additional features for specific end-user applications. It also adds capabilities to the native boot processing of the Blackfin chip. Although some details are specific to the Blackfin family of DSPs, some general features may be helpful on other CPUs. Tweed writes:

"Not long ago, I was working on an inertial measurement unit (IMU) that was based on the highly integrated ADIS16350 inertial sensor from Analog Devices that Tom Cantrell wrote about in his column in Issue 208 ('Thanks for the MEMS,' 2007). This is a six-axis MEMS sensor (three axes of angular rate and three axes of acceleration) in a compact and rugged package. My client wanted to marry an Analog Devices Blackfin DSP chip to it in order to create a self-contained inertial measurement solution ... A key aspect of the implementation was that the firmware would need to be updated in the field, after the unit had left the controlled environment of the factory, in order to support both bug fixes to the basic functionality and

additional features for specific end-user applications.

This article is about the two-stage bootloader that we developed that meets all our requirements and adds some capabilities to the native boot processing of the Blackfin chip. While much of this discussion will be specific to the Blackfin family of DSP chips, some aspects of it are more general and can be ported to other processors."

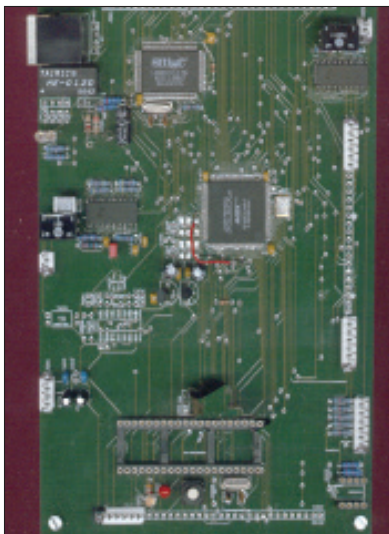


These articles and others on topics relating to embedded solutions are available at www.cc-webshop.com.

Interface Ethernet and Embedded Systems

By Eddie Insam (Circuit Cellar 172, 2004)

Fast Ethernet and small microcontrollers do not mix, or so they say. In this article, Eddie shows you how to add full-speed, 100-Mbps Ethernet to an embedded system. He



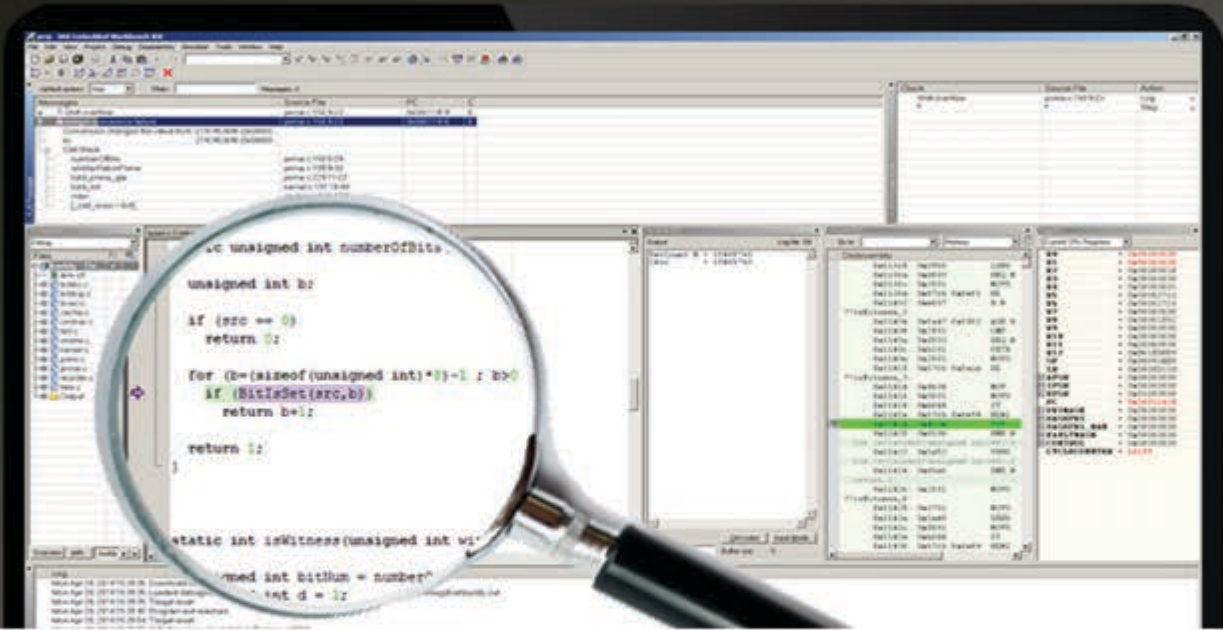
presents the supporting hardware that will help you get the job done. He writes:

"Another article about Ethernet and embedded systems? Well, yes, but here I'm talking about 100 Mbps. Yes, the fast version, not the 10-Mbps sloggers usually associated with small embedded systems. Who wants high-speed Ethernet anyway? I thought you might ask. If you need to feed data from a fast source such as a CCD camera, voice, or high-speed data converter, you'll need to use a high-speed method of getting it into your PC. FireWire and USB2 are possibilities, but Ethernet remains one of the comfiest methods for packing fast data into a PC. It also means your peripheral can be sited a long way away, something you just can't do with FireWire and USB.

Mind you, it's difficult enough to get a 10-Mbps Ethernet controller working anywhere near full speed when paired with a small microcontroller. These cronies can take an eternity to move data in and out of the line, and they do it mostly 1 byte at a time. Slap in a faster microcontroller? It won't necessarily help. You will need a pretty powerful 32 biter plus a good helping of side IC condiments before anybody notices the difference. This article is about modesty anyway. How can you stay below the clouds and still get the performance by using relatively cheap hardware?"

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



EDITORS' PICKS

Embedded Solutions

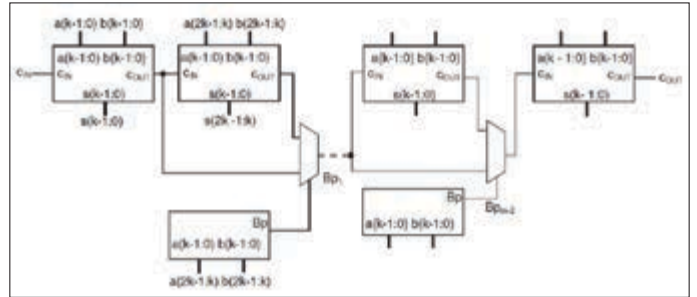
Efficient, Practical Adders for FPGAs

By Vitit Kantabutra, Pasquale Corsonello, Stefania Perri, and Maria Antonia Iachino (Circuit Cellar 148, 2002)

In the 1800s, Charles Babbage developed calculating engines that made addition more efficient. Today, carry-skip adders are used in many digital systems. A group of engineers took the technology a step further by designing efficient carry-skip adders for FPGAs. They write:

"Adders are a part of the critical path in virtually all practical digital systems, because every arithmetic operation requires one. Thus, the speed and area efficiency of adders is important when you're designing a circuit. A fairly large body of literature exists for adder design in today's standard cell and custom IC technologies, but little success has been reported in the realm of the increasingly important FPGA technologies.

The history of adder optimization long predates electronic computers. In his 1851 treatise on calculating engines, Charles Babbage wrote about the 'four cases of obstacles presenting the appearance of impossibilities' that he encountered as he designed the analytical engine, which was the mechanical prototype for modern computers. The first of



these difficulties, which he encountered in an early stage of the design process, was concerned with the efficiency of the addition process. So, Babbage went on to provide a practical solution that can be adapted for use in VLSI technology, including today's FPGA technology.

In this article, Babbage's solution, now called carry-skip or carry-bypass adders, will be explained, and we'll show you how we adapted them to be used for FPGAs. We focused on two families of FPGAs for this project, the Atmel AT40K and Xilinx Virtex."

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

SRAM: The New Embedded Solution

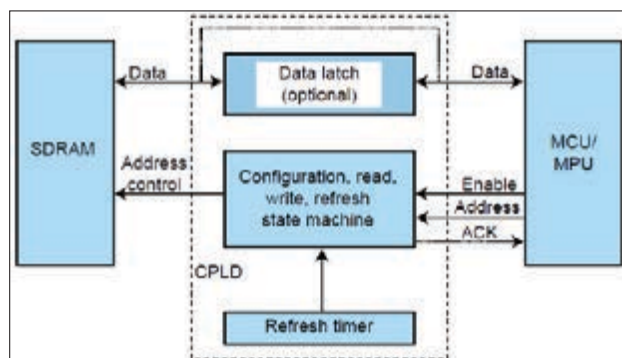
By Mark Balch (Circuit Cellar 125, 2000)

The next time you're designing a small, low-power embedded system, you might want to consider using an SDRAM controller. As Mark shows us, an embedded SDRAM controller just might be the simplest and most cost-effective solution. He writes:

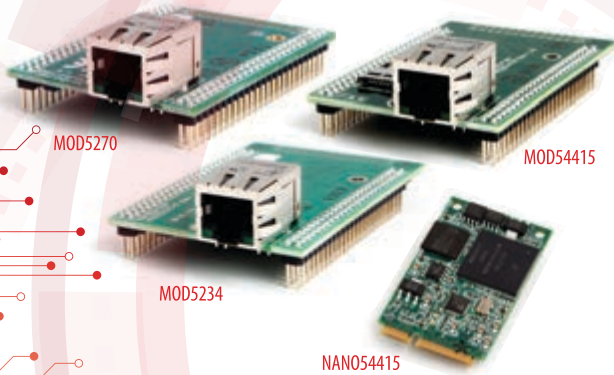
"The key to incorporating SDRAM into your embedded system is providing a simple memory controller to take care of the SDRAM housekeeping functions. The minimal set of

SDRAM control logic can fit into modest PLDs. Some basic features are powerup configuration, periodic refresh, single-word read, and single-word write.

That's it! All you need is control logic that can perform these four operations and you have a minimal SDRAM interface to your controller. There are as many ways to implement this basic idea as there are engineers. One technique is a single binary-encoded state machine inside a PLD. This state machine would be large, but if your system is only running at 12 MHz and you use a modern, mainstream PLD, the timing may be on your side."



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>

Secure Hash Standard

Understanding the Secure Hash Standard Without Math

FEATURES

Does the Secure Hash Standard confuse you? Try approaching it from a hardware point of view. The design presented here implements the complete byte-oriented SHA-256 variant of this standard.

By Monte Dalrymple (US)

The cryptographic hash functions specified in the Secure Hash Standard (SHS) are widely used in today's connected world, but in many cases, their use is not really visible to users. For example, both the Secure Sockets Layer (SSL) and the Transport Layer Security (TLS) protocols have either the option or the requirement to use a hash function from the SHS while setting up a secure link, but this fact is completely invisible to users. On the other hand, one obvious place where an SHS function is used is in the authentication of software distributions, where the correct hash value is published and can be used before the software is installed to verify that no alteration has taken place between the publisher and the user.

I've always wondered exactly how these hash functions work, and I recently had the opportunity to investigate the details myself. While there is just one SHS, there are seven different algorithms specified in the standard, with varying levels of complexity. In this article I'll show you how one of the algorithms, called SHA-256, really works from a hardware standpoint, using the Verilog hardware description language.

HIGH-LEVEL VIEW

A cryptographic hash function is most often used to transform an arbitrary-length message into a fixed-size representation called a message digest (or hash value). This transformation is one-way, which means that given a message digest it is impossible to

reverse the process and recreate the original message. The transformation is also very nonlinear, in the sense that even a small change to the original message, such as flipping the state of a single bit, will lead to a very different message digest. Another property of this transformation is that given a large fraction of a message, and the message digest, it is still infeasible to compute the missing part of the message. All of these features make the cryptographic hash function indispensable to digital signature and message authentication algorithms. The highly nonlinear nature of secure hash algorithms also makes them useful for generating random numbers or random bits.

One way to look at a cryptographic hash function is to think of the input message as a very big binary number, and the message digest as a fixed-sized binary number that is somehow computed from the input number. In the case of the SHA-256 the input number can be up to 2^{64} bits in length, and the message digest is 256 bits long. Given the size difference in these two numbers, there are clearly a huge number of messages with the same message digest! So the secret to a good hash algorithm is to make sure that no similar messages can lead to the same hash value.

There are different ways to construct a secure hash algorithm, but those specified in the SHS all share the same basic steps and have similar requirements. The primary requirement is that the input message has to be a multiple of the block size. For SHA-256 this


```

/*****
/* Ch function */
/*****
function [31:0] ch;
    input [31:0] x, y, z;
    begin
        ch = (x & y) | (~x & z);
    end
endfunction

/*****
/* Maj function */
/*****
function [31:0] maj;
    input [31:0] x, y, z;
    begin
        maj = (x & y) | (x & z) | (y & z);
    end
endfunction

/*****
/* Big Sigma 0 function */
/*****
function [31:0] bsigma_0;
    input [31:0] x;
    begin
        bsigma_0 = {x[1:0], x[31:2]} ^ {x[12:0], x[31:13]} ^ {x[21:0], x[31:22]};
    end
endfunction

/*****
/* Big Sigma 1 function */
/*****
function [31:0] bsigma_1;
    input [31:0] x;
    begin
        bsigma_1 = {x[5:0], x[31:6]} ^ {x[10:0], x[31:11]} ^ {x[24:0], x[31:25]};
    end
endfunction

/*****
/* Little Sigma 0 function */
/*****
function [31:0] sigma_0;
    input [31:0] x;
    begin
        sigma_0 = {x[6:0], x[31:7]} ^ {x[17:0], x[31:18]} ^ {3'b000, x[31:3]};
    end
endfunction

/*****
/* Little Sigma 1 function */
/*****
function [31:0] sigma_1;
    input [31:0] x;
    begin
        sigma_1 = {x[16:0], x[31:17]} ^ {x[18:0], x[31:19]} ^ {10'b0000000000, x[31:10]};
    end
endfunction

```

LISTING 1

SHA-256 uses six simple functions to mix the data in various ways. Two functions operate on three inputs, while the rest operate on a single input.

LISTING 2

A 32-bit constant is inserted into the calculation during each round. The organization shown here, with sixteen 128-bit words and a four-input multiplexer, was chosen for coding convenience.

```

/*****
/* k constants
*****/
function [31:0] k256_const;
    input  [5:0] step;
    reg [127:0] const_rom;
    begin
        case (step[5:2])
            4'b0000: const_rom = 128'h428a2f9871374491b5c0fbcfe9b5dba5;
            4'b0001: const_rom = 128'h3956c25b59f111f1923f82a4ab1c5ed5;
            4'b0010: const_rom = 128'hd807aa9812835b01243185be550c7dc3;
            4'b0011: const_rom = 128'h72be5d7480deb1fe9bdc06a7c19bf174;
            4'b0100: const_rom = 128'he49b69c1efbe47860fc19dc6240ca1cc;
            4'b0101: const_rom = 128'h2de92c6f4a7484aa5cb0a9dc76f988da;
            4'b0110: const_rom = 128'h983e5152a831c66db00327c8bf597fc7;
            4'b0111: const_rom = 128'hc6e00bf3d5a7914706ca635114292967;
            4'b1000: const_rom = 128'h27b70a852e1b21384d2c6dffc53380d13;
            4'b1001: const_rom = 128'h650a7354766a0abb81c2c92e92722c85;
            4'b1010: const_rom = 128'ha2bfe8a1a81a664bc24b8b70c76c51a3;
            4'b1011: const_rom = 128'hd192e819d6990624f40e3585106aa070;
            4'b1100: const_rom = 128'h19a4c1161e376c082748774c34b0bcb5;
            4'b1101: const_rom = 128'h391c0cb34ed8aa4a5b9cca4f682e6ff3;
            4'b1110: const_rom = 128'h748f82ee78a5636f84c878148cc70208;
            4'b1111: const_rom = 128'h90bffffaa4506cebbef9a3f7c67178f2;
        endcase
        case (step[1:0])
            2'b00: k256_const = const_rom[127:96];
            2'b01: k256_const = const_rom[95:64];
            2'b10: k256_const = const_rom[63:32];
            default: k256_const = const_rom[31:0];
        endcase
    end
endfunction

```

block size is 512 bits, or sixteen 32-bit words. Since it is unlikely that any given message will meet this requirement, the standard specifies exactly how a message must be padded to be a multiple of 512 bits. It is worth noting that the standard allows for arbitrary bit length messages, even though most messages will be multiples of a byte.

Once the padding is done, each message block is expanded into a “message schedule” of 64 32-bit words. The first 16 of these words are just the original message, while the remainder are combinations of the words in the original message.

The words in the message schedule are then cycled through eight “working variables” which further combine them so that each word in the message schedule contributes to the final values of each of these working variables. At the end of these mixing operations the values in the working variables are added to the existing hash value. At the end of the message this gives the final hash value.

The SHS specifies exactly how the message schedule and working variables are calculated, but provides no insight into the mathematical

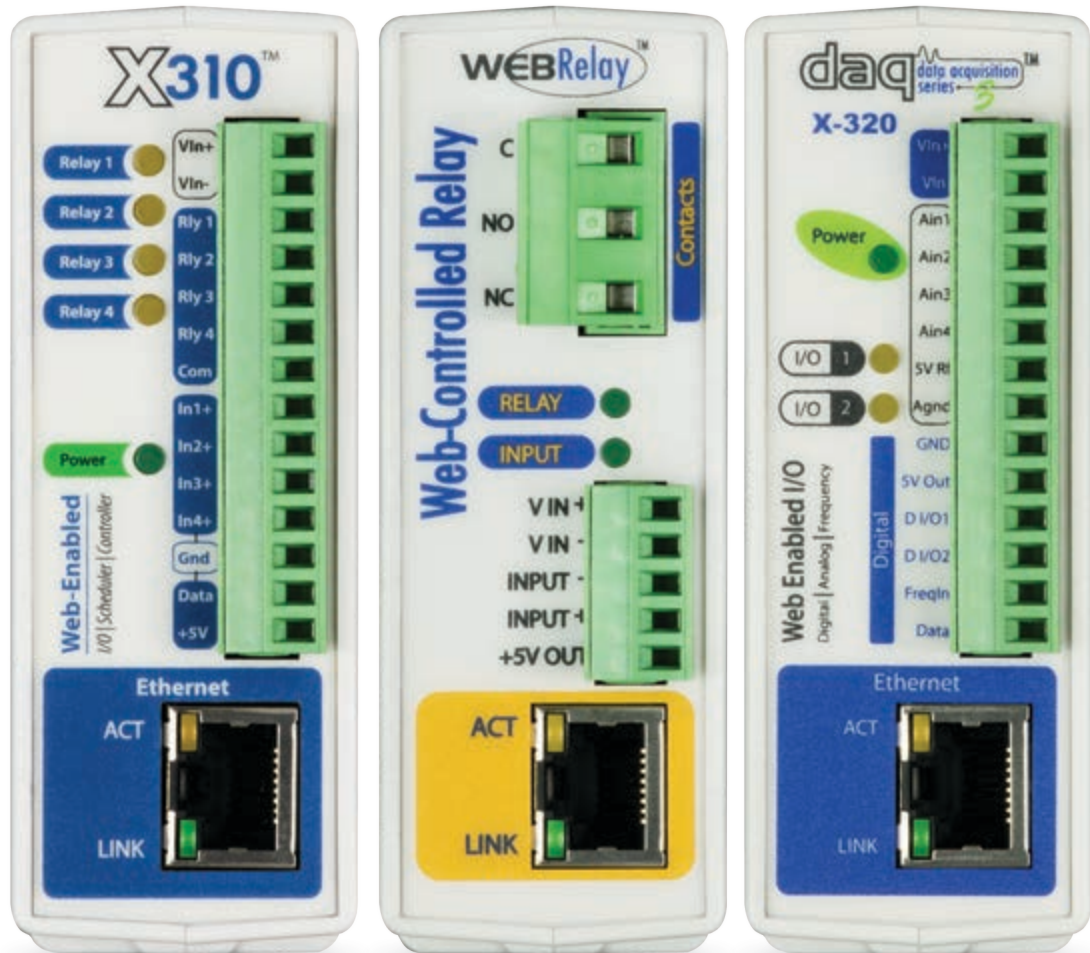
basis for the different operations. This is quite different from the Advanced Encryption Standard (AES), where the mathematics are explained in detail in the standard. Personally, I would have preferred at least some clues as to how the mixing functions were created.

DETAILED OPERATION

The design I describe here is a complete implementation of SHA-256 that uses one clock cycle per step. I chose the one clock cycle per step to make the logic easy to follow, while at the same time minimizing the amount of hardware used. If higher performance is required it is certainly possible to apply pipelining techniques to the design, but that will increase the amount of hardware required.

The design uses a 32-bit data path, which is what the standard envisions. Other widths are possible, either wider or narrower. A 64-bit data path would give higher performance, at the cost of more hardware. A narrower data path would obviously give poorer performance, and probably also lead to more hardware, because of the need to store

Your Ultimate Drop-In Solution



Web-Based I/O Control & Monitoring

Web-Enabled: Relays | Digital Inputs | Analog Inputs & Outputs | Temperature/Humidity

CONTROL by
WEBTM
www.ControlByWeb.com

ABOUT THE AUTHOR

Monte Dalrymple (monted@systemyde.com) has been designing integrated circuits for over 35 years. He holds a BSEE and MSEE from the University of California at Berkeley and holds seventeen patents. He is the author of the book *Microprocessor Design Using Verilog HDL* (Circuit Cellar, 2012). Not limited to things digital, he holds both amateur and commercial radio licenses.

intermediate results and the need for more complex control. Keep this in mind if you are thinking about using a different data path width.

The six logical functions that are used to mix the bits of the message data are shown in **Listing 1**. Two of the functions—called Ch (which is probably short for “choose”) and Maj (which is probably short for “majority”) in the standard—take three input words and provide one output word. The ch function uses the bits of one input word to control the selection of the corresponding bit from the other two input words, while the Maj function sets the output bit if the corresponding bits in two or more of the input words are set. The other four functions—called in this design *bsigma_0*, *bsigma_1*, *sigma_0*, and *sigma_1*—mix the individual bits of the input word in different ways by adding together shifted versions of the word (hence the “sigma” in the name).

During each of the 64 steps in the

algorithm, a 32-bit constant is used in the calculation. According to the standard, these constants are “the first thirty-two bits of the fractional parts of the cube roots of the first sixty-four prime numbers.” There is nothing special about this choice, except that it makes it easy for anyone to verify that there is no backdoor hidden in the constants. Listing 2 shows how these constants are generated. The choice of sixteen 128-bit words and a four-input multiplexer is somewhat arbitrary, although with this choice each bit of the *const_rom* variable can be created directly using a four-input LUT in an FPGA.

The top-level module interface and definitions are shown in **Listing 3**. This is a simple synchronous interface that uses a 32-bit data bus and a write strobe. One complication arises because the message being hashed may not be a multiple of 32-bit words. So the last write must be tagged with a data width of 1, 2, 3, or 4 bytes. A second complication is that the standard

```

module sha256_top (bufr_full, hash_done, hash_reg, clk, resetb, start_pls, wr_bus,
                  wr_pls, wr_type);

    input          clk;           /* main clock*/
    input          start_pls;     /* ok to start hash operation*/
    input          resetb;       /* async master reset*/
    input          wr_pls;       /* write buffer*/
    input [2:0]    wr_type;      /* write operation type*/
    input [31:0]   wr_bus;       /* write data bus*/

    output         bufr_full;     /* buffer is full*/
    output         hash_done;     /* hash_reg is valid*/
    output [255:0] hash_reg;     /* hash result*/

    /******
    /* write_tag definitions*/
    /******
    `define TYPE_LAST0 3'b000      /* only for zero-length msg */
    `define TYPE_LAST1 3'b001      /* last word - one byte only */
    `define TYPE_LAST2 3'b010      /* last word - two bytes */
    `define TYPE_LAST3 3'b011      /* last word - three bytes */
    `define TYPE_LAST4 3'b100      /* last word - full word */
    `define TYPE_WORD  3'b111      /* normal word */

    /******
    /* initial hash values */
    /******
    `define IHV_0 32'h6a09e667
    `define IHV_1 32'hbb67ae85
    `define IHV_2 32'h3c6ef372
    `define IHV_3 32'ha54ff53a
    `define IHV_4 32'h510e527f
    `define IHV_5 32'h9b05688c
    `define IHV_6 32'h1f83d9ab
    `define IHV_7 32'h5be0cd19
  
```

LISTING 3

The interface uses a 32-bit bus, with each write tagged with a type. This is also where the Initial Hash Value is defined.

```

/*****
/* host interface */
/*****
assign wr_fifo = wr_p1s && !(wr_type == `TYPE_LAST0);

always @ (wr_type or wr_bus) begin
  case (wr_type)
    `TYPE_LAST1: wr_data = {4'h8, wr_bus[31:24], 24'h800000};
    `TYPE_LAST2: wr_data = {4'h8, wr_bus[31:16], 16'h8000};
    `TYPE_LAST3: wr_data = {4'h8, wr_bus[31:8], 8'h80};
    `TYPE_LAST4: wr_data = {4'h8, wr_bus};
    default: wr_data = {4'h0, wr_bus};
  endcase
end

fifo_x36 FIFO ( .almost_empty(almost_empty), .almost_full(),
               .data_out(fifo_out), .empty(fifo_empty), .full(buf_full), .clk(clk),
               .data_in(wr_data), .read_en(rd_fifo), .resetb(resetb), .write_en(wr_fifo) );

always @ (wr_type) begin
  case (wr_type)
    `TYPE_LAST0: bcnt_inc = 3'b000;
    `TYPE_LAST1: bcnt_inc = 3'b001;
    `TYPE_LAST2: bcnt_inc = 3'b010;
    `TYPE_LAST3: bcnt_inc = 3'b011;
    default: bcnt_inc = 3'b100;
  endcase
end

always @ (posedge clk or negedge resetb) begin
  if (!resetb) begin
    bcnt_reg    <= 40'h0;
    empty_dly   <= 1'b1;
    length_reg  <= 40'h0;
    msg_null    <= 1'b0;
    msg_walign  <= 1'b0;
    msgdone_reg <= 1'b0;
    msgstrt_reg <= 1'b0;
    wr_length   <= 1'b0;
  end
  else begin
    if (wr_p1s || wr_length) bcnt_reg <= (wr_length) ? 40'h0 : (bcnt_reg + bcnt_inc);
    if (wr_length) begin
      length_reg <= bcnt_reg;
      msg_walign <= ~|bcnt_reg[1:0];
    end
    empty_dly   <= fifo_empty;
    msg_null    <= !msg_done && ((wr_length && ~|bcnt_reg) || msg_null);
    msgdone_reg <= !msg_done && (wr_length || msgdone_reg);
    msgstrt_reg <= !rd_word  && (start_p1s || msgstrt_reg);
    wr_length   <= wr_p1s && !(wr_type == `TYPE_WORD);
  end
end

assign blk_start = msg_null || (msgdone_reg && !fifo_empty) || !(almost_empty || fifo_empty);
assign msg_start = msgstrt_reg && blk_start;
assign msg_lastw = !empty_dly && fifo_out[35];

```

LISTING 4

Most FPGA families provide 36-bit wide FIFO primitives, so that is what I use to hold the message data. Part of the message padding is done on the input side of the FIFO.

allows for zero-length messages, so there has to be some way to communicate this. A write with the `WRITE_LAST0` type signals that the message to be hashed has zero length. This degenerate case is explicitly allowed in the standard, and requires special handling in the remainder of the design.

Rather than return the calculated hash value as words over a data bus, I have chosen to simply output the full 256-bit hash along with a one-clock pulse that indicates that the hash result is valid. This structure makes simulation and testing easier, because the intermediate hash value is available after each block.

The initial hash values are also defined in this listing. According to the standard these constants are “the first thirty-two bits of the

fractional parts of the square roots of the first eight prime numbers.” This is another case where the values seem to have been chosen to eliminate any suspicion of a backdoor.

The host interface is shown in **Listing 4**. This interface uses a FIFO that is 36 bits wide and at least 16 words deep. Most FPGA families provide macros meeting these specifications as part of their block RAM functionality. The details will vary depending on vendor, so I have merely instantiated a generic version, which will need to contain the vendor-specific module name and connections. This design assumes non-pipelined timing for the FIFO. That is, the read enable is sampled on one clock edge and the output data will be available for sampling by the next clock edge.

The 36-bit wide FIFO holds the 32-bit data,

```

/*****
/* pad insertion
/*****
assign msg_done = &pad1_reg;
assign pad_pls = pad_strt || (rd_dly && padding_reg) || (msg_null && msg_done);
assign pad_strt = rd_dly && !padding_reg && (msg_null || msg_lastw);
assign pad1_lsb = length_reg[5:2] + !msg_walign;
assign rd_fifo = rd_word && !padding_reg && !msg_null && !pad_strt && !padding_reg;

always @ (length_reg) begin
    casex (length_reg[5:0])
        6'b1110xx,
        6'b111100: pad1_msb = 1'b0;
        default: pad1_msb = 1'b1;
    endcase
end

always @ (posedge clk or negedge resetb) begin
    if (!resetb) begin
        pad1_reg <= 5'h0;
        pad2_reg <= 1'b0;
        padding_reg <= 1'b0;
    end
    else if (pad_pls) begin
        pad1_reg <= (pad_strt) ? {pad1_msb, pad1_lsb} : (pad1_reg + 1'b1);
        pad2_reg <= pad_strt && msg_walign && !msg_null;
        padding_reg <= !msg_done && (pad_strt || padding_reg);
    end
end

always @ (msg_null or pad1_reg or pad2_reg or fifo_out or length_reg) begin
    casex ({msg_null, pad2_reg, pad1_reg})
        7'b0000000: msg_data = fifo_out[31:0];
        7'b0011110: msg_data = {21'h0, length_reg[39:29]};
        7'b0011111: msg_data = {length_reg[28:0], 3'b000};
        7'b01xxxxx,
        7'b1x0xxxx: msg_data = 32'h80000000;
        default: msg_data = 32'h00000000;
    endcase
end

```

LISTING 5

The pad insertion state machine does the bulk of the message padding, and handles the case where the message is word-aligned or aligned to a 512-bit block boundary.

Verilog HDL

With the right tools

such as this book,

**designing a microprocessor
can be easy.**

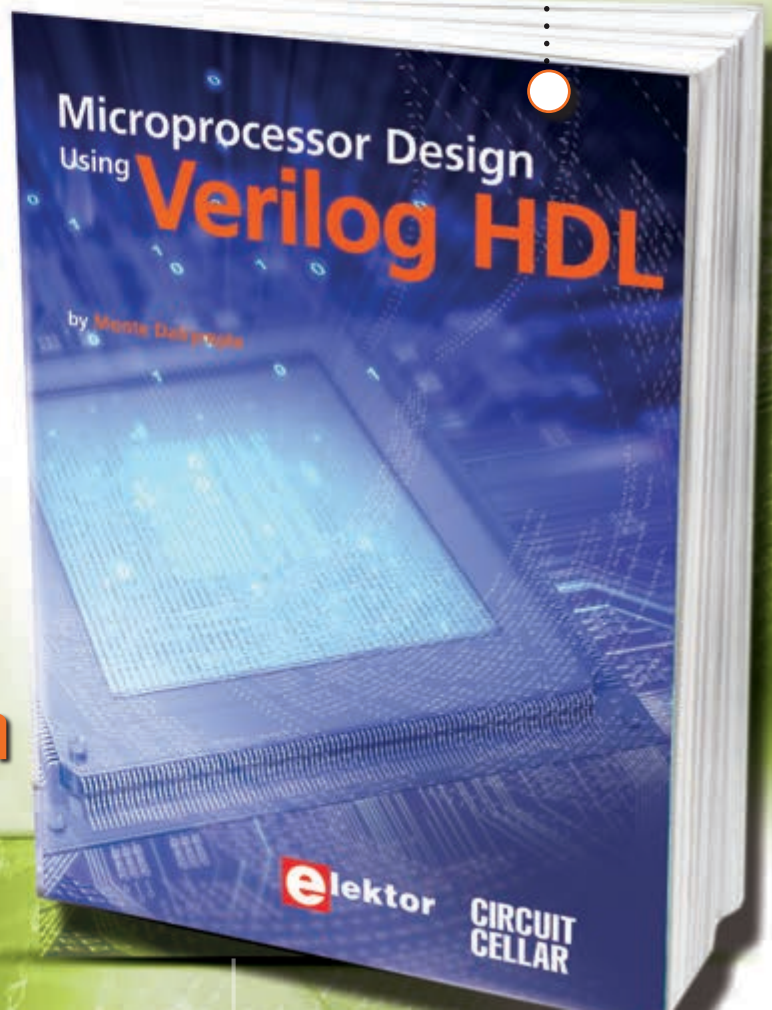
Okay, maybe not easy, but certainly less complicated. Monte Dalrymple has taken his years of experience designing embedded architecture and microprocessors and compiled his knowledge into one comprehensive guide to processor design in the real world.

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!



```

/*****
/* hash state machine
/*****
assign rd_word  = ini_blk || ini_blk_dly ||
                 (runblk_reg && ~|step_reg[5:4] && ~&step_reg[3:1]);
assign step_done = &step_reg;

always @ (posedge clk or negedge resetb) begin
  if (!resetb) begin
    hash_done  <= 1'b0;
    ini_blk    <= 1'b0;
    ini_blk_dly <= 1'b0;
    ini_msg    <= 1'b0;
    ini_msg_dly <= 1'b0;
    ld_hash   <= 1'b0;
    rd_dly    <= 1'b0;
    runblk_reg <= 1'b0;
    runmsg_reg <= 1'b0;
    step_reg  <= 6'h00;
  end
  else begin
    hash_done <= ld_hash && !runmsg_reg;
    ini_blk   <= !ini_blk && !ini_blk_dly && !runblk_reg && ((blk_start && runmsg_reg) ||
                                                             (msg_start && !runmsg_reg) ||
                                                             padding_reg);

    ini_blk_dly <= ini_blk;
    ini_msg    <= !ini_msg && !ini_msg_dly && !runblk_reg && msg_start && !runmsg_reg;
    ini_msg_dly <= ini_msg;
    ld_hash   <= step_done;
    rd_dly    <= rd_word;
    runblk_reg <= !step_done && (ini_blk_dly || runblk_reg);
    runmsg_reg <= !msg_done && (ini_msg_dly || runmsg_reg);
    step_reg  <= (runblk_reg) ? (step_reg + 1'b1) : 6'h00;
  end
end
end

```

LISTING 6

The hash state machine reads 16 words from the input FIFO and keeps track of the 64 rounds of the hash algorithm. FIFO reads are pipelined to account for the timing of common FPGA FIFO primitives.

plus one extra bit that is set only for the last word in a message. Tagging the last word in a message makes it very easy to communicate the “last word of the message” information to the other side of the FIFO.

Because the hash state machine requires a complete block of 16 32-bit words, the FIFO

must be configured to assert the `almost_empty` signal as long as there are 15 or fewer words in the FIFO. This is what holds off the start of a hash calculation. Again, this is a standard feature on FPGA FIFO macros.

The host interface is the ideal place to do the first part of the padding operation,

SHA-256 & BITCOINS

SHA-256 is central to the operation of the Bitcoin digital currency. The bitcoin blockchain, which can be thought of as a ledger recording all bitcoin transactions, must be periodically verified. Bitcoin miners do this verification using SHA-256.

The way this works is that a nonce (number used once), plus all of the new bitcoin transactions, are appended to the existing blockchain and the resulting hash value is calculated. This process continues, with a different nonce, until a hash value is found that meets certain requirements, namely that it begins with some number of zero bits. The bitcoin miner who first finds a nonce that results in such a hash value is rewarded with some newly-created bitcoins

and the blockchain is then considered verified to that point. A short time later, this process is repeated.

Finding a hash value that meets the leading-zeros requirement requires a brute-force search, using a lot of nonces. Early bitcoin miners used regular computers to do the searching, but the process quickly migrated to dedicated FPGA-based machines and then to ASIC-based machines. Where a CPU-based machine might do 1 million hashes per second (MHash/s), FPGA machines might reach one GHash/s and an ASIC machine might reach 1,000 GHash/s. Multiply these rates by the number of bitcoin miners, and that’s a lot of hashes.

```

/*****
/* message schedule
/*****
assign wi_nxt = sigma_1(wim1_reg) + wim6_reg + sigma_0(wim14_reg) + wim15_reg;

always @ (posedge clk) begin
  if (ini_blk_dly || runblk_reg) begin
    wi_reg   <= (rd_dly) ? msg_data : wi_nxt;
    wim1_reg <= wi_reg;
    wim2_reg <= wim1_reg;
    wim3_reg <= wim2_reg;
    wim4_reg <= wim3_reg;
    wim5_reg <= wim4_reg;
    wim6_reg <= wim5_reg;
    wim7_reg <= wim6_reg;
    wim8_reg <= wim7_reg;
    wim9_reg <= wim8_reg;
    wim10_reg <= wim9_reg;
    wim11_reg <= wim10_reg;
    wim12_reg <= wim11_reg;
    wim13_reg <= wim12_reg;
    wim14_reg <= wim13_reg;
    wim15_reg <= wim14_reg;
  end
end

```

when the last word, or part of a word, of the message is written. Even though the standard allows for completely arbitrary message lengths, this design only handles byte-aligned messages. It's easy to modify the design to account for arbitrary bit length messages, but I'll leave that as an exercise for the reader, because the feature is rarely necessary.

The host interface side of the FIFO is also the best place to implement the counter for the message length. The standard provides for a 64-bit field for message length, but I've only implemented a 40-bit counter here. That is sufficient for a terabyte message length, so it should be good enough.

The bulk of the pad insertion is done in the logic shown in **Listing 5**. The unaligned last-word cases were handled in the bus interface, but the case of a word-aligned message must be accounted for in this section, along with case of a zero-length message. A simple state machine is required to keep track of the inserted padding, and providing the pad data to the hash state machine, while at the same time intercepting the FIFO read strobe from the hash state machine. The padding information finishes with the value from the message length counter, which complicates things if the message ends very close to a 512-bit block boundary. The `pad1_msb` signal is required to detect this condition. The message is padded even if it naturally ends on a block boundary.

Compared to the bus interface and the pad

insertion logic, the hash state machine shown in Listing 6 is pretty simple. It's basically just a 6-bit counter that tracks the sixty-four steps of the hash algorithm. The extra registers are required to properly start up the state machine at the beginning of a message or the beginning of a block, and to account for the timing of the FIFO read operations.

The standard talks about a "message schedule of sixty-four 32-bit words," which can be very disconcerting until you realize that only 16 of these values need to be available for any given step of the algorithm. The logic for the message schedule is shown in **Listing 7**. This is just a recirculating buffer, where the `wi_reg` register holds the message schedule word for the current step, and the rest of the buffer holds the values for the previous 15 steps. During the first 16 steps of the algorithm this buffer is loaded with the 16 words of the message block to be hashed, and for the remaining 48 steps the modified message data recirculates through this buffer. At each recirculation step the `wi_reg` register is loaded with a combination of four of the other entries in the message schedule. Two of these entries are scrambled using two of the sigma functions shown earlier, and then the four values are added together to form the new word of the message schedule.

The bulk of the hash algorithm is implemented in the working registers, shown in **Listing 8**. This listing also contains the actual 256-bit hash register. The eight

LISTING 7

The message schedule is loaded with the message data during the first 16 rounds, and then the first stage of mixing occurs during the remaining rounds.

LISTING 8

The eight working variables are arranged as a recirculating buffer, and this is where the bulk of the mixing occurs. At the end of the 64 rounds the resulting value is added to the previous hash value.

```

/*****
/* working variables
*****/
assign t_1 = h_reg + bsigma_1(e_reg) + ch(e_reg, f_reg, g_reg) +
           k256_const(step_reg) + wi_reg;
assign t_2 = bsigma_0(a_reg) + maj(a_reg, b_reg, c_reg);

always @ (posedge clk) begin
  if (ini_msg_dly || ini_blk_dly || runblk_reg) begin
    a_reg <= (ini_msg_dly) ? `IHV_0 :
             (ini_blk_dly) ? hash_reg[255:224] : (t_1 + t_2);
    b_reg <= (ini_msg_dly) ? `IHV_1 :
             (ini_blk_dly) ? hash_reg[223:192] : a_reg;
    c_reg <= (ini_msg_dly) ? `IHV_2 :
             (ini_blk_dly) ? hash_reg[191:160] : b_reg;
    d_reg <= (ini_msg_dly) ? `IHV_3 :
             (ini_blk_dly) ? hash_reg[159:128] : c_reg;
    e_reg <= (ini_msg_dly) ? `IHV_4 :
             (ini_blk_dly) ? hash_reg[127:96] : (d_reg + t_1);
    f_reg <= (ini_msg_dly) ? `IHV_5 :
             (ini_blk_dly) ? hash_reg[95:64] : e_reg;
    g_reg <= (ini_msg_dly) ? `IHV_6 :
             (ini_blk_dly) ? hash_reg[63:32] : f_reg;
    h_reg <= (ini_msg_dly) ? `IHV_7 :
             (ini_blk_dly) ? hash_reg[31:0] : g_reg;
  end
end

/*****
/* hash value
*****/
always @ (posedge clk) begin
  if (ini_msg_dly || ld_hash) begin
    hash_reg[255:224] <= (ini_msg_dly) ? `IHV_0 : (hash_reg[255:224] + a_reg);
    hash_reg[223:192] <= (ini_msg_dly) ? `IHV_1 : (hash_reg[223:192] + b_reg);
    hash_reg[191:160] <= (ini_msg_dly) ? `IHV_2 : (hash_reg[191:160] + c_reg);
    hash_reg[159:128] <= (ini_msg_dly) ? `IHV_3 : (hash_reg[159:128] + d_reg);
    hash_reg[127:96] <= (ini_msg_dly) ? `IHV_4 : (hash_reg[127:96] + e_reg);
    hash_reg[95:64] <= (ini_msg_dly) ? `IHV_5 : (hash_reg[95:64] + f_reg);
    hash_reg[63:32] <= (ini_msg_dly) ? `IHV_6 : (hash_reg[63:32] + g_reg);
    hash_reg[31:0] <= (ini_msg_dly) ? `IHV_7 : (hash_reg[31:0] + h_reg);
  end
end

```

working registers are also arranged as a recirculating buffer, although modified values are injected into this buffer at two points. These modified values are created from two

temporary variables, called t_1 and t_2 . One of these temporary variables involves a five-input adder, which I have chosen to code directly even though this may or may not be the best approach for logic synthesis. These temporary variables use the remainder of the functions shown earlier, along with the step-specific constant.

The working registers only use the current message schedule value, wi_reg , which is why the working registers can operate during the first sixteen steps while the message schedule is being loaded with the message data. The working registers and the hash register are loaded with the initial hash values at the start of a message, and the working registers are



circuitcellar.com/ccmaterials

RESOURCES

L. Bassham and T. Hall, "The Secure Hash Algorithm Validation System (SHAVS)," NIST, 2014, <http://csrc.nist.gov/groups/STM/cavp/documents/shs/SHAVS.pdf>

Information Technology Laboratory, "Secure Hash Standard (SHS)," NIST, 2012, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

loaded with the current hash value at the start of a new 16-word block. At the end of each 16-word block, the contents of the working registers are added, word by word, to the value in the hash register. As you can see, the hash calculation itself is really pretty simple, with the message data flowing through a pair of recirculating buffers to create the hash value.

VERIFICATION

Even though this implementation of the hash algorithm is pretty simple, verifying that the implementation is correct can be a real challenge, because it isn't easy to figure out what the correct hash value should be for an arbitrary message. To help with this process NIST provides a set of sample messages along with the correct hash value. The test bench that is available on the *Circuit Cellar* FTP site exercises this design with a subset of this set of messages and automatically checks the resultant hash value against the published hash value.

NIST has also set up the Secure Hash Standard Validation System (SHAVS) for verifying implementations of the standard. With this system, a testing laboratory generates a series of messages, which are hashed by the implementation being tested. These hash values are returned to the testing laboratory, which verifies correctness, and certifies the implementation. The series of messages provided includes all short messages from zero to the block size in length, a series of long messages up to one hundred blocks in length, and a very long message one hundred thousand blocks in length. A hardware implementation like the one presented here really only needs to be checked using the short and long messages. The other cases tested by SHAVS are only necessary for software implementations where buffer issues might be present. Even so, to be certified an implementation must correctly generate the hash for every test message.

SO, DOES IT WORK?

Given the lack of a mathematical foundation for the algorithms in the standard, I was curious about how well SHA-256 actually worked. So I instrumented the simulation test bench and did some testing.

In the first set of tests I checked the hash values for every possible bit combination in each individual byte position in a word. The results of these tests are shown in **Table 1**. The "byte 0" case is for a single-byte message, the "byte 2" case is for a 16-bit message of the form 0x00nn, the "byte 3" case is for a message of the form 0x0000nn, and the

	Mean	Std Dev	Median	Minimum	Maximum
byte 0	128.7	7.7	129	103	149
byte 1	128.1	8.1	129	104	148
byte 2	127.5	7.8	127	104	158
byte 3	128.1	8.3	128	99	147

TABLE 1

The statistics for the number of one's in the message digest for all 256 possible values in each individual byte position.

	Mean	Std Dev	Median	Minimum	Maximum
byte 0	127.7	7.6	127	110	148
byte 1	128.2	7.7	128	109	155
byte 2	127.9	8.7	128	108	158
byte 3	127.5	8.3	128	108	153

TABLE 2


The statistics for the number of bit positions in the message digest that toggle for all 256 possible single-bit toggles in each individual byte position.

"byte 4" case is for a message of the form 0x000000nn. Since the number of ones and zeros in the resulting hash should be fairly evenly distributed, I counted the number of ones in each hash result. In all cases the average was close to half the bits in the resulting hash value being one. Clearly SHA-256 does a very good job of mixing up the results across the entire 32-bit word.

For the second set of tests, I used the same set of input messages, with Gray code ordering, and then checked for the number of bit positions in the resulting hash values that changed with each one-bit change in the input message. The results of these tests are shown in **Table 2**. Again, SHA-256 does a very good job, with each single-bit change in the input message leading to an average of half of the bits in the resulting hash value changing.

One final test that would be interesting would be to try every possible 256-bit input message and check for identical hash results. Ideally there should not be any, but since this would involve 2^{256} simulation runs, I'll have to skip this test.

WRAPPING UP

For all of its power, the SHA-256 algorithm is fairly easy to understand once the individual operations and data flow are clear. However, the SHS standard specifies the operations with no reference to how the algorithm is implemented, which obscures what is going on during the hashing process. Hopefully I've made this hashing process a little clearer, and now you can be confident that your data really is protected. 

Running on Battery (Part 1)

FEATURES

Battery-Powered Microcontroller Circuits

Sometimes you might need to power a small microcontroller circuit with batteries. In this article, Stuart explains how to protect against reverse battery voltage, select a microcontroller, and manage power.

By Stuart Ball (US)

Most of the time, we power our projects from the AC line. Whether it is a one-off prototype or a simple product, the easiest way to power a small microcontroller circuit is with a wall-wart DC supply and a regulator of some kind on the board. But what about those cases where you need to be able to operate from batteries? More and more consumer electronics, from games to smartphones, are operated from battery.

Microcontroller applications that need battery operation can range from MP3 players to hand-held temperature monitor to a remote device that has to wake up once a day to log and possibly transmit instrument readings to a monitoring station. How do you power such devices? What things do you need to consider in designing for that environment?

Battery-based operation is a big topic, so in the first part of this article series, I will look at three aspects of it: reverse battery protection, microcontroller selection, and power management.

REVERSE BATTERY PROTECTION

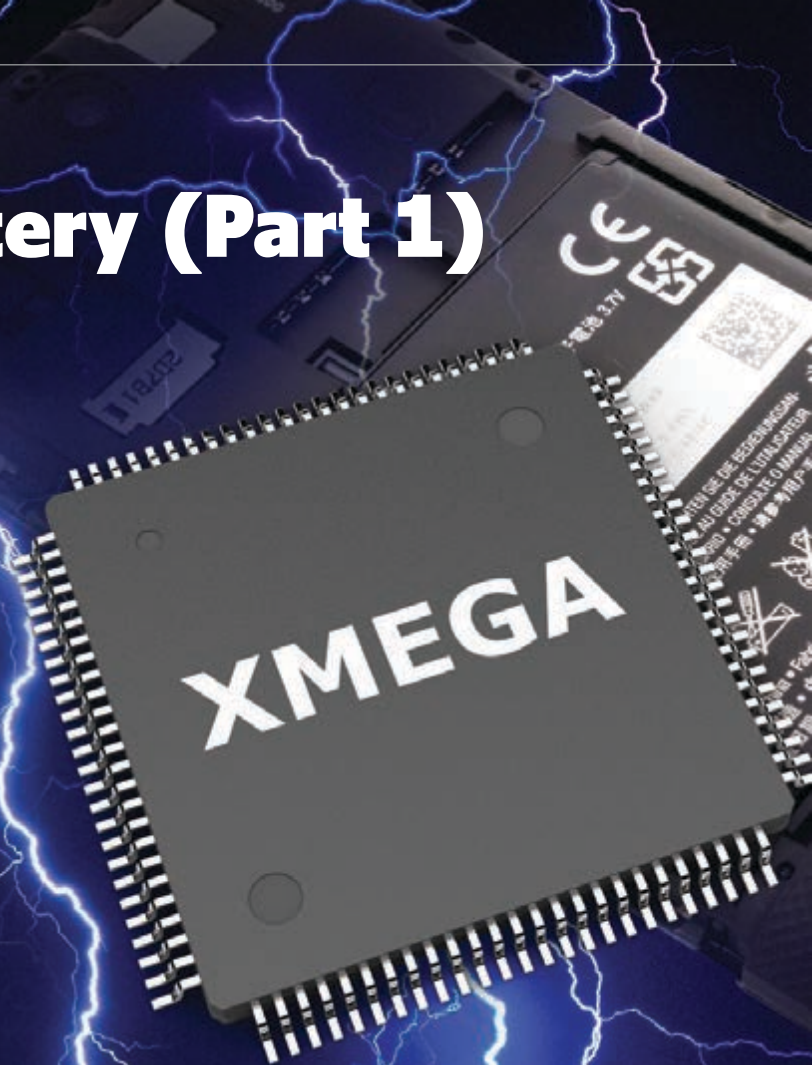
If you run something on batteries, sooner or later someone is going to try to install a battery backwards. Although your 40-year-old transistor radio might easily survive reversing the battery, modern microcontroller electronics is not quite so forgiving. A circuit

run on batteries has to protect in some way from battery reversal. There are a few ways to do that. Let's consider each one.

Solder the battery into the circuit: Although it may seem extreme, this absolutely prevents battery reversal unless the user tries to replace the battery by unsoldering it. It also implies the need for a rechargeable battery and an accompanying charging circuit, and some kind of external power jack to charge the battery. For a remote site, it probably means a solar charging system. All that is beyond the scope of this article, but there may be situations where a soldered rechargeable battery is the simplest solution, especially if you don't have to provide fast recharge of the battery.

In a very low-power circuit intended for a single use (think of a remote data logger that will log data until the battery is exhausted and then be retrieved later), you may not need the charging circuit. But such applications are rare. Usually you will want to be able to replace or recharge the battery.

Use a battery that can only be installed one way: This is what most cell phones do. The battery pack is not symmetrical and the battery can only be inserted one way that will allow it to make contact with the circuit. Again, a simple solution, but one that requires a custom battery. It is really only suitable for



high-volume devices, unless you want to use a battery from a real cell phone. That might be a reasonable solution if you only need a handful of them so you can swap the battery at a remote site once a month.

There are some off-the-shelf solutions that get close to this. Tadiran and Dantona (and others) make battery packs that have wire leads and polarized connectors. These aren't cheap, and you aren't going to pick up a replacement battery at your local Walmart. But they solve the battery reversal issue, and might be a solution for something that isn't going to need battery replacement very often.

Use a coin cell: An example would be the standard CR2302 battery. Some coin cell holders have a contact on the top and bottom, which does allow the battery to be inserted so that the polarity is reversed. But if the positive contact is on the side of the holder instead of on the top, then it makes contact with the edge of the battery instead of the top surface. In those holders, a reversed battery is not an issue; both contacts connect to the same terminal of the battery if it is inserted backward. The BS-7 from Memory Protection Devices is an example of such a battery holder. Renata and others make similar parts. Of course, your circuit must be capable of running from one or more coin cells to make this approach practical.

Use a diode: This is the most foolproof way to prevent damage due to battery reversal if you are using standard AA, AAA, C, or similar batteries. A Schottky diode is placed in series with one of the battery leads (usually the positive) so that it inhibits current flow in the reverse direction. The catch is that the diode will have a forward voltage drop typically between about 0.3 and 0.5 V. For a microcontroller operating from a 9-V battery and a regulator, this probably won't be an issue. But if you are operating from a couple of AA batteries, this is a significant portion of the available battery voltage.

Use a MOSFET: When the battery is inserted correctly, the MOSFET is turned on and passes current to the circuit. If the battery is inserted backward, the gate is reverse-biased and the MOSFET is off. This approach is great if you are operating from a battery of 9 V or greater. The problem with operating from batteries in the 3-V range is finding a MOSFET that has sufficiently low on-resistance and sufficiently low gate-source threshold voltage. MOSFETs that have low gate-source threshold voltage typically have relatively high on-resistance, so the drop across the MOSFET can approach that of a Schottky diode. However, this approach may work if your operating current is low enough. There are a few MOSFET transistors, such as the Fairchild FDN306P that are rated

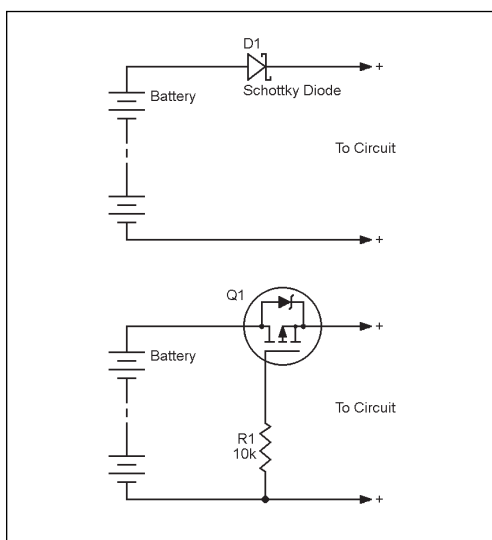


FIGURE 1

Either a series Schottky diode or a P-channel MOSFET can be used to protect the circuit against a reversed battery.

for 1.8-V operation with reasonably low on-resistance. **Figure 1** shows both a series diode and a P-channel MOSFET used for reversal protection.

MICROCONTROLLER SELECTION

The key to battery operation is to reduce the current drawn by the circuit as much as possible. You also need to accommodate the unique characteristics of a battery-operated environment. Let's review the key requirements for the microcontroller.

Low-power operation: You want the microcontroller to operate at minimal current, consistent with the requirements of the application. Availability of low-power sleep modes may be important; more about that later.

Ability to operate over a range of voltages: The Atmel ATXmega324A operates from 1.6 to 3.6 V. This allows the part to work as the battery ages and accommodates an external diode (if needed) for battery reversal protection. Some microcontrollers have tight operating voltage ranges, such as 3.3 V only. These can be difficult to use in a battery circuit.

Low-power clock: Many microcontrollers have low-power internal clocks. Many will operate using an external 32-kHz watch crystal. In general, parts that operate with high-speed external clocks are going to use significant power even in a low-power state. Again, using the ATXmega324A as an example, the maximum idle current using a 32-kHz external crystal and 3-V supply is 4 μ A. Using a 2-MHz external crystal, the maximum idle current is 390 μ A. Although that isn't a lot, it's a difference of almost 100:1.

Use of a low-power, external clock either means low-frequency operation or an internal PLL to multiply the clock to a reasonable

internal operating frequency. Many microcontrollers, including the ATXmega series, do have internal PLLs that can multiply an external 32-kHz clock to a much faster operating frequency.

POWER MANAGEMENT

Some battery-powered devices are only used intermittently. An example would be a piece of test equipment such as a DVM or some piece of portable RF test gear. In that case, the easiest way to manage the power is to use an on-off switch; you turn the unit on when you want to use it. For devices like that, you could use a 9- or 12-V battery and a regulator to supply power to the circuit.

On the other hand, imagine that you had to actually turn your smartphone off when you weren't using it. Aside from the problem of not being able to receive incoming calls and messages, you would have to wait for the phone to start up, register with the cell towers, and get ready to operate when you wanted to use it.

A smartphone is an extreme example since it contains a complete operating system and a number of internal peripherals that have to be initialized. But the principle applies to other cases. Sometimes you want to leave the device powered on all the time. The remote data logger that I mentioned earlier is one example. For that kind of application, you want to get as many hours of operation as possible from a set of batteries. This typically means being deliberate about power usage. Some power management considerations include the following.

Sleep mode: Many microcontrollers have multiple sleep modes. For example, the ATXmega32A4 has five different sleep modes with varying degrees of power usage and varying capabilities. In Power-down mode, the internal real-time clock will not wake the device up but an external interrupt, such as a button push, will. This would be suitable for a device that needs to wake up when the user presses a key.

In the ATXmega32A4 Power-save mode, the RTC still operates and can wake the device up. This mode would be suitable for regular sampling of temperature or some other external sensor, where there is no user to press a button. The point is that the part has varying power modes, with various power consumption levels. Selection of a part with power modes suitable to your application can have significant impact on the power consumption.

Peripheral power-down: Again, using the ATXmega as an example, the part has sleep modes with specific functionality. But it is also possible when actively operating to shut

down specific peripherals such as the analog-to-digital converter. This allows the part to reduce operating current when in an active mode. In some situations, sleep mode may not be needed if the active mode current can be sufficiently reduced.

Avoid power-hogging design techniques: Some things we take for granted in AC-powered designs turn out to be power hogs in battery-operated designs. For example, it is easy to generate a reference voltage lower than the supply voltage with a resistive voltage divider. But any resistor connected directly between the battery terminals is going to draw current, regardless of the microcontroller state. Either avoid these or give the microcontroller the ability to remove power from the voltage divider when it goes into a low power mode.

Make sure that output drive is removed when entering a reduced-power mode. For example, you might drive a relay using an NPN transistor with a resistor in the base, which is in turn driven by a microcontroller output pin. Unless the pins float in sleep mode, leaving this pin high will leave the transistor turned on and the relay energized. You would want to configure that pin as a low output before entering sleep mode. Even if the transistor is just driving a logic signal, such as a case where it is translating between 3.3 and 5 V, the current drain from a high microcontroller output through the limiting resistor can be many times the sleep mode current of the microcontroller.

Low battery sensing: The easiest way to detect a low battery is to use a microcontroller with an internal analog-to-digital converter and read the battery voltage. This can be problematic on some microcontrollers if they don't have an internal voltage reference for the ADC. Ideally, you want a microcontroller with an internal reference that is lower than the battery voltage. Alternatively, you can use an external reference and turn it off when you aren't reading battery voltage.

Usually microcontroller ADCs will not work if the input voltage is higher than the reference. So you can use a voltage divider to reduce the battery voltage to a level the ADC input will accept. You will want to switch the positive side of the divider off to avoid permanent current drain. The microcontroller can drive a transistor to switch the high side of the voltage divider on and off. Depending on the accuracy you need, you may need to compensate for the transistor voltage drop. Some microcontrollers include a brown-out sensing circuit to detect low supply voltage. Typically this generates an interrupt or reset.

PWM relays: If you are driving a relay, drive it with 100% duty cycle to pull it in, and then

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.

PCB WEST 2015

Conference & Exhibition

CONFERENCE: September 15 - 17

EXHIBITION: Wednesday, September 16

SANTA CLARA CONVENTION CENTER, CA

PCB WEST OFFERS:

- A comprehensive three-day conference, with several affordable packages
- More than 65 presentations in all, the largest PCB event in the Silicon Valley
- Targeted conference sessions for all levels of experience and training, from novice designer and engineer to seasoned pro
- Covering topics such as RF/microwave/PI/SI, printed electronics/flex circuits and next-gen components
- Exhibit hall featuring the industry's leading suppliers and services in a one-day exhibition
- 10+ Free technical sessions PLUS networking events - all on the exhibit floor - lunch, afternoon breaks and an evening reception
- and more...

Register for the conference by August 14th and save up to \$100

www.pcbwest.com

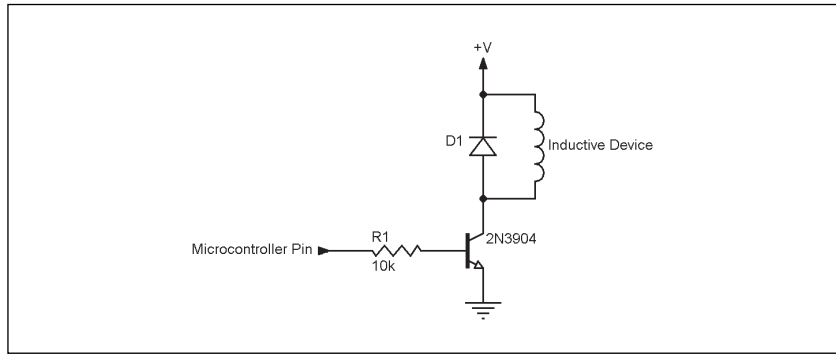


FIGURE 2

Flyback protection using a diode. The positive supply must be able to dissipate the energy from the inductive device without generating excessive voltage to the microcontroller.

reduce the duty cycle to hold the relay. Most relays require significantly less current to hold them closed than to close them. You may want to adjust the duty cycle to compensate for battery voltage as the battery drains. The same concept applies to any other peripheral component that is needed intermittently; turn it off when not actually in use.

Compensating for battery voltage loss: Say you are driving some LEDs in your circuit. If they are visual indicators, the decrease in brightness as the battery loses voltage may not be enough to notice. But if the LED is used to drive an optical sensor, the reduction in drive may cause false readings. You could fix this by driving the LED with a constant current source, or by characterizing the LED (and sensor, if needed) over the operational battery voltage range, then use a look-up table in the firmware to compensate for changing battery voltage. Obviously, you need to be able to read the battery voltage for this to work.

Battery swapping: In a device that is always on and isn't using rechargeable batteries, the user will eventually change the battery. This can produce a lot of make-break cycles of the power as the old battery is removed and the new battery is inserted. So, depending on your application, when power is applied, you might need to introduce

a significant delay of half second or more to let everything stabilize. You don't want to be writing information to the microcontroller EEPROM when power is suddenly removed. One way around this is to have a large enough capacitor to hold the voltage up for a bit, and then sense the voltage in the firmware. If you see it drop rapidly, go into some kind of safe mode.

Active duty cycle: The battery drain in the various sleep modes of a microcontroller is low. But battery life is dependent on the average current drain, which includes the time that the microcontroller is active. Minimizing the active time and maximizing the intervals between active times will produce the longest battery life. You can't control this interval if the active time is initiated by a user input. But if an internal RTC or other timing event initiates the active state, then you have some control over the average current drain. Waking up the microcontroller 10 times a second will take less total power than waking it up 100 times per second, if the application requirements give you a choice.

OTHER CONSIDERATIONS

Let's consider some additional problems and fixes.

Amplifiers: In the old days of transistor electronics, it wasn't uncommon to see inexpensive radios and walkie-talkies make a "motorboating" sound when the battery got low. The audio circuits depended on the low impedance of the battery to prevent the battery from becoming part of an oscillator circuit. When the battery (typically a 9-V battery) aged, the circuit would become a blocking oscillator.

The point is that if you have amplifiers in your circuit, you want to be sure that you have enough capacitance across the positive and negative supply leads to allow the circuit to operate without oscillating even when the battery is weak. And if you are using a diode or MOSFET for reverse polarity protection, the capacitance goes on the circuit side of the protection, not on the battery side.

Drive levels: In an AC-powered circuit, we can easily generate 5 V or some other voltage to drive things like FETs. In a battery-operated circuit, it may be difficult to generate enough voltage to drive the gate of a MOSFET to its switching threshold voltage. What happens to the ability to drive the gate of a MOSFET when the battery is nearly drained? You can add a voltage doubler or other DC-DC converter if the drive level is marginal, but that adds complexity, cost, and additional current drain. This sort of problem isn't insurmountable, but you may need to take it into account.

Flyback diodes: In the unusual event that



SOURCE

ATXmega324A Microcontroller
Atmel Corp. | www.atmel.com

you are driving a relay, solenoid, motor, or other inductive electromechanical device, a diode is often used from the driving device (usually a transistor) to the supply (see **Figure 2**). When the transistor is turned off, the flyback voltage from the inductor can damage the transistor; the diode limits this voltage to the supply voltage. The problem is that in a battery-operated circuit with a diode or MOSFET for battery reversal protection, the energy has no place to go and can cause a rise in the supply voltage to the microcontroller, high enough to damage it.


You can fix this with a sufficiently high capacitance from the supply to ground, high enough to absorb the energy without excessive voltage rise. You could also use a Zener diode from the supply to ground if you can select a value that is higher than the maximum battery voltage but lower than the safe voltage of the circuit. A microcontroller that can operate from 2.7 to 5 V, for example, could operate at 3 V on battery, but have a 5-V Zener diode to protect against excessive flyback transients.

A similar situation could occur with an external input (such as an RS-232 signal) that is clamped to the positive supply. In such cases, you would want to clamp to a fixed

voltage (such as a Zener diode) rather than the positive supply voltage. The important thing is to compensate for situations that can produce excessive voltages on the positive supply. The positive supply in electronic circuits is often treated as a low impedance to ground; there are situations where this is not the case, especially in battery-operated circuits.

Testing: You will want to test your circuit with dead batteries, nearly dead batteries, and new batteries. You want to be sure it works over the entire range you intend. An AC-powered circuit just needs to work if the power supply is good; a battery circuit needs to operate over some range of power supply voltage. An AC-powered circuit has a supply that is either on or off; a battery-powered circuit has a “supply” that degrades over time.

BATTERY POWER

Running on batteries is more complicated than using AC power. But for applications that demand it, a careful design makes it possible. In the next part of this article series, we'll consider the topics of combined AC/battery operation, single-cell operation, rechargeable batteries, and battery life estimation. 

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

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



LEMOS INTERNATIONAL

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

Sound Ecology and Acoustic Health (Part 3)

A Quantitative Application for WAT_AN_APP

FEATURES

Last month, Adrien and Mike tackled the topics of recording and playing back audio .3GPP files. In this article, they get quantitative with an audio record and analysis update for the WAT_AN_APP application.

By Adrien Gaspard and Mike Smith (Canada)

As we detailed the first article in this series, we can boast to our local teenager that “WE” have developed an Android application. Last month, we explained how we added “just enough additional code” (JEAC) to record and play back .3GPP files. As a result, we could say the following to a neighbor at a backyard BBQ: “We’re not imagining things. Look, there are really local urban noise nuisances.”

This time we are going quantitative with an audio record-and-analysis update of the WAT_AN_APP application for “Things that Go Boom at Night” (TGBN). This enables us either to request lots of commiseration as the community noises are really bad or boast that we have more ghosts going “BOO(m)” at night than anybody else!

QUICK RECAP

The idea is to leave the TGBN device running over a weekend. We want to capture a 7-s sound recording if the community noise sound level gets above background level. When the sound stays high in intensity, a warning message (Android toast) appears on our device’s screen. As a precaution, a scary “Boo” kind of sound will be output to intimidate any ghosts that might be present as we head off in the opposite direction.

The key elements of our app so far are outlined in **Listing 1**. We kept the line numbers used in the previous articles to help you find the things that need to

be updated. We’ve extended the `MainActivity` class to add the new `SoundAnalysis` activity for the TGBN update (Lines 30–33). To activate the TGBN section, there is another button to our screen layout file `activity_main.xml` (see **Listing 2**, Lines 130–139).

SCREEN, SOUND, & COMPATIBILITY

Figure 1a shows a screen with text to indicate the numbers of records remaining (see **Listing 3**, Lines 1710–1716) and TGBN sounds detected (Lines 1720–1727) together with the recording time remaining (Lines 1730–1739). These `TextView` widgets are updated by the activity’s code.

Figure 1b shows that still with four recording sessions to go, our sound analysis indicates that one TGBN sound has been detected. To avoid setting off false alarms, we must remain “quiet and collected.” When ghost hunting, does “collected” mean in complete control or huddled together in a corner?

Each time we find that we have recorded a sound much larger than the background level, we will generate the “BOO” sounds to intimidate the local ghosts. Finally, we have the application display a message “Captured” and report the results of our sound analysis. If you feel in a Halloween mood, use our current app to record your own “Boo” sound in a .3GPP format. Otherwise, you can grab scary sounds from various website such as caseyscaverns.com/4/halloween5.html. For compatibility with our code, the sound file needs to be stored in a new `WAT_AN_APP` folder named “raw.”

Before proceeding, check the Android Manifest file in the project to make sure that

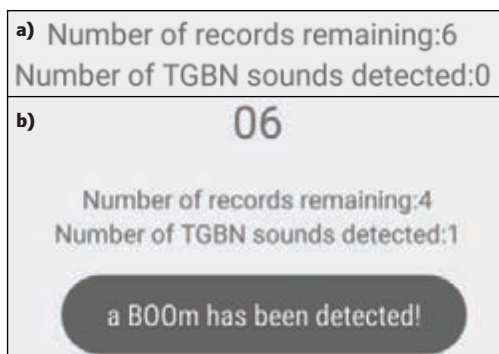


FIGURE 1

Here you see the (a) the initial sound analysis screen and (b) the detection of a BOOm using the TGBN detector.

the android minimum SDK version of our application is set to 14 for compatibility issues with the code we are going to implement. (Refer to Part 2 of this article series: Listing 7, Line 3006 of WAT_AN_APP\AndroidManifest.xml.)

TGBN IMPLEMENTATION

Implementing our new SoundAnalysis activity requires a number of steps. To prevent carpal tunnel syndrome, remember to sit up straight in your chair and to practice the finger yoga exercises noted in the previous article. An alternative is to access the code on the *Circuit Cellar* FTP site.

A new ints.xml file in the WAT_AN_APP\res\values folder (see **Listing 4**) shows the parameters needed to control our widgets. These are the recording time (Line 2822) and the number of recordings (Line 2830). To avoid just collecting background noises, set the impulse threshold for when to start recording (Line 2840). You will have to play around with the sound threshold for when you consider that a TGBN sound has been detected (Line 2841). The parameters needed for our next article are defined Lines 2850 to Line 2852.

The actual SoundAnalysis activity is implemented in **Listing 5**. After having imported all the libraries, we pop up the widgets to display the number of records remaining (Line 551), TGBN sounds detected (Line 552), and recording time remaining (Line 553). A variable captureAudio is defined to capture the audio (Line 555), and a MediaPlayer mediaPlayer is used to play the scary sound when a TGBN sound has been detected (Line 556). The scary sound "boo.3GPP" from the "raw" folder is linked to the MediaPlayer in the onCreate() method (Line 603). All the methods that are going to be used are summed up in Listing 5.

The article 2 activity used a MediaRecorder recorder to allow us to capture and listen to a sound. The recorder tracking TGBN sound is different than the one implemented in the previous article, as it is used to record a sound in a room and store its data into a buffer. We are no longer just interested in listening to the recorded sound. With this quantitative analysis we need to measure the number of bizarre louder sounds. If you are interested in listening to the captured TGBN sounds then you could do that with the use of an AudioTrack. This class allows streaming of the PCM audio buffer to the audio sync for playback as explained on developer.android.com/reference/media/AudioTrack.html.

The onStart() method from **Listing 6** contains the code required to initialize the three TextView widgets, which display the number of records available (6)

```
package com.wat_an_app: // MainActivity.java
// SAME AS Article 1, Listing 1, Lines 2 to 5

// Cause display of MainActivity screen layout
public class MainActivity extends Activity{
// SAME AS Article 1, Listing 1, Lines 11 to 15
    public void AudioRecordPlayback(View v){
// SAME AS Article 1, Listing 1, Lines 21 to 23

// New SoundAnalysis Activity
30.     public void SoundAnalysis(View v){
31.     Intent beginSoundAnalysis =
// SAME AS Article 1, Listing 1, Lines 21 to 23
        new Intent(this, SoundAnalysis.class);
32.     startActivity(beginSoundAnalysis);
33.     }
}
```

LISTING 1

New SoundAnalysis activity, which is called from MainActivity.java in the WAT_AN_APP\src\ folder

```
<!--Used by MainActivity -->
<RelativeLayout
xmlns:android=http://schemas.android.com/apk/res/android
... <!-- SAME AS Article 1, Listing 2, Lines 102 to 105 -->

    <!--Greeting text-->
    <TextView
        <!-- SAME AS Article 1 Listing 2 Lines 111 to 116-->
    />

    <!--Start activity button for AudioRecordPlayback-->
    <Button
        <!--SAME AS Article 1 Listing 2 Lines 121 to 128 -->
    />

    <!--New Button to start SoundAnalysis activity -->
130. <Button
131.     android:id= "@+id/start_detecting_TGBN"
132.     android:layout_width="wrap_content"
133.     android:layout_height="wrap_content"
134.     android:layout_centerVertical="true"
135.     android:layout_centerHorizontal="true"
136.     android:layout_below="@id/start_WAT_AN_APP"
137.     android:text="@string/press_to_start_detecting_TGBN"
138.     android:onClick="SoundAnalysis"
139. />

</RelativeLayout>
```

LISTING 2

New layout details to be inserted into activity_main.xml file in the WAT_AN_APP\res\layout folder

and TGBN sound detected (0) when the activity starts. We execute the captureAudio class from this method (Line 758). The onPause() method (Lines 800–804) is called when the activity goes into the background, but has not been killed (stopped) and turned into a spirit yet. The audio capture is cancelled in

LISTING 3

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

```

<!--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- 105 -->

1710. <TextView
1711.     android:id="@+id/number_records"
1712.     android:layout_width="wrap_content"
1713.     android:layout_height="wrap_content"
1714.     android:layout_centerHorizontal="true"
1715.     android:layout_centerVertical="true"
1716.     />

1720. <TextView
1721.     android:id="@+id/number_tgbn_sounds"
1722.     android:layout_below="@+id/number_records"
1723.     <!-- COPY FROM Article 3, Listing 2 Lines 1712 to 1715-->
1727.     />

1730. <TextView
1731.     android:id="@+id/textViewTime"
1732.     android:layout_above="@+id/number_records"
1733.     <!--COPY FROM Article 3, Listing 2 Lines 1712 to 1715 -->
1737.     android:layout_marginBottom="17dp"
1738.     android:textSize="25sp"
1739.     />

1799. </RelativeLayout>

```

```

2800. <?xml version="1.0" encoding="utf-8"?>
2801. <resources>

    <!-- AudioRecord parameters -->
2810.     <integer name = "sample_rate">8000</integer>
2811.     <integer name = "num_channels">1</integer>

    <!-- Defines the buffer length for sound detection -->
2820.     <integer name = "detect_buffer_length">256</integer>
2821.     <!-- Defines the capture time after impulse is detected -->
2822.     <integer name = "capture_time">7</integer>

    <!-- Number of sounds we want to record -->
2830.     <integer name = "num_records">6</integer>
2831.     <integer name = "num_tgbn">1</integer>
2832.     <integer name = "number_tgbn_scratch">0</integer>

    <!-- Threshold at which we detect a sound above back-ground
        and at which we detect a TGBN sound -->
2840.     <integer name = "detect_threshold_impulse">10000</integer>
2841.     <integer name =
                "detect_threshold_tgbn_sound">25000</integer>

    // SPOILER ALERT - Needed in Articles 4 and 5
2850.     <integer name = "samples_per_bin_freq">32</integer>
2851.     <integer name = "samples_per_bin_time">4</integer>
2852.     <integer name = "playChirp">1</integer>

2899. </resources>

```

LISTING 4

Creation of the ints.xml file in the WAT_AN_APP\values folder

PICOSCOPE 4000 SERIES



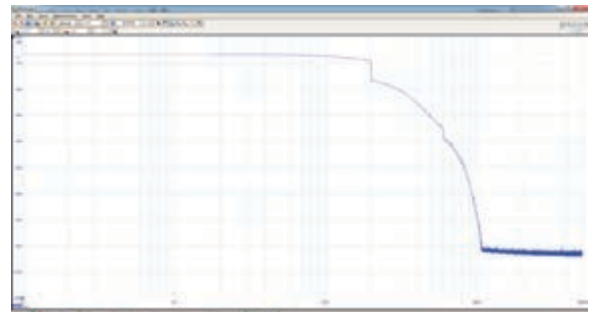
PicoScope® 4262

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


```

500. package com.wat_an_app;
501. import java.util.concurrent.TimeUnit;
502. import android.media.AudioFormat;
503. import android.media.AudioRecord;
504. import android.media.MediaPlayer;
505. import android.media.MediaRecorder.AudioSource;
506. import android.os.AsyncTask;
507. import android.os.Bundle;
508. import android.os.CountDownTimer;
509. import android.support.v7.app.ActionBarActivity;
510. import android.widget.TextView;
511. import android.widget.Toast;
512. import com.wat_an_app.R;

550. public class SoundAnalysis extends ActionBarActivity{
551.     TextView TextHandleNumberRecords;
552.     TextView TextHandleNumberTGBN;
553.     TextView textViewTime;
554.     AudioRecord recorder;
555.     CaptureAudio captureAudio;
556.     private MediaPlayer mediaPlayer;
557.     final CounterClass timer = new CounterClass(8000,1000);

600.     @Override protected void onCreate(Bundle savedInstanceState) {
601.         super.onCreate(savedInstanceState);
602.         setContentView(R.layout.activity_sound_analysis);
603.         mediaPlayer = MediaPlayer.create(this, R.raw.booy);
604.     }

    // Methods and classes associated with public class SoundAnalysis

    // SPOILER ALERT - Leaving 605 - 749 for next article's neat stuff

    // @Override protected void onStart()
    // Method details in Listing 5 Lines 750 to 759
    // protected void onPause(){
    // Method details in Listing 5 Lines 800 to 804
    // public class CounterClass extends CountDownTimer {
    // Method details in Listing 6 Lines 850 to 862

    // private class CaptureAudio
    // extends AsyncTask<Void, Integer, Integer> {
    // protected void onPreExecute()
    // Method details in Listing 7 Lines 910 to 919
    // protected Integer doInBackground(Void ... params)
    // Method details in Listing 8 Lines 950 to 987
    // protected void onProgressUpdate(Integer ... data)
    // Method details in Listing 9 Lines 1200 to 1212
    // protected void onPostExecute(Integer data)
    // Method details in Listing 9 Lines 1250 to 1255
    // protected void onCancelled()
    // Method details in Listing 10 Lines 1300 to 1305
    // protected boolean detectImpulse(short[] samples)
    // Method details in Listing 10 Lines 1350 to 1356
    // protected boolean detectTGBN(short[] samples)
    // Method details in Listing 10 Lines 1400 to 1406
1449.     }

```

LISTING 5

SoundAnalysis.java – Definition of the variables for onCreate() method

this method (Line 801).

We based the countdown indicator from one on the Internet (see **Listing 7**), which will set a text “Captured,” indicating that the record is over (Listing 7, Lines 850–877). The `onTick()` method fires at regular intervals to show the number of seconds remaining until the recording finishes. `onFinish()` is called when the time is up to set the text “Captured” on the screen. Note that we will use an `onTick()` `Article4` method in our next article (Line 860), which has not been implemented yet. More details of the public constructors and methods used in this class are at [developer.android.com/reference/android/os/CountDownTimer.html#onTick\(long\)](http://developer.android.com/reference/android/os/CountDownTimer.html#onTick(long)).

There’s a special health feature available with this app! Please note that the recorder only starts recording after a sound impulse above the background noise level has been detected. You can validate your finger yoga practice by showing you can snap your fingers together loud enough to test the app.

Now let’s consider some tricks or treats for Android.

ASYNCTASK: PRE-EXECUTION

A process can run on multiple threads within the Android system. When the application first runs, it will use the User Interface (UI) thread to controls everything we see on the screen. While doing shorter operations on this thread is acceptable, doing longer operations may cause the system to stop responding to user interaction, causing the user to think that the program is running slowly or has stopped running.

To fix this, Android uses the `AsyncTask` class so that you can shift longer operations to different threads and keep the main UI thread running smoothly. An asynchronous task is defined on Android using three types: `Async<Params, Progress, Result>`, as well as four steps: `onPreExecute`, `doInBackground`, `onProgressUpdate`, and `onPostExecute`.

`onPreExecute` in **Listing 8** is the first step to be invoked and sets up our `SoundAnalysis` activity. This task initializes the `AudioRecorder` using the following format: `AudioRecord(int audioSource, int sampleRateInHz, int channelConfig, int audioFormat, int bufferSizeInBytes)` (Line 912).

Our audio source is the device’s microphone. The sample rate and number of channels have been configured in the `ints.xml` file. The audio format “`ENCODING_PCM_16BIT`” means that our audio buffer will be filled with signed integer values ranging from the maximum value of `-32767` to a minimum value of `32768`.

Every time a media recorder or player is initialized, it is really important to verify that

```

750.     @Override protected void onStart(){
751.         TextHandleNumberRecords =(TextView) findViewById
              (R.id.number_records);
752.         TextHandleNumberRecords.setText(getResources().
              getString(R.string.number_records_remaining)
              +Integer.toString(getResources().
              getInteger(R.integer.num_records)));
753.         TextHandleNumberTGBN=
              (TextView)findViewById (R.id. number_tgbn_sounds);
754.         TextHandleNumberTGBN.setText(getResources().
              getString(R.string. number_tgbn_recorded)
              +Integer.toString(getResources().getInteger
              (R.integer.number_tgbn_scratch)));
755.         super.onStart();
756.         textViewTime = (TextView)findViewById
              (R.id.textViewTime);
757.         captureAudio = new CaptureAudio();
758.         captureAudio.execute();
759.     }

800.     protected void onPause(){
801.         captureAudio.cancel(false);
802.         super.onPause();
803.         finish();
804.     }

```

LISTING 6

SoundAnalysis.java — `onStart()` and `onPause()` methods

```

850.     public class CounterClass extends CountDownTimer {
851.         public CounterClass(long millisInFuture,
              long countDownInterval) {
852.             super(millisInFuture, countDownInterval);
853.         }

854.         @Override public void onFinish(){
855.             textViewTime.setText("Captured");}

860.         public void onTick_Article4 // SPOILER ALERT
              (long millisUntilFinished){} //USED IN ARTICLE 4

870.         @Override public void onTick(long millisUntilFinished){
871.             long millis = millisUntilFinished;
872.             String hms = String.format("%02d",
              TimeUnit.MILLISECONDS.
              toSeconds(millis) - TimeUnit.MINUTES.toSeconds
              (TimeUnit.MILLISECONDS.toMinutes(millis)));
873.             System.out.println(hms);
874.             textViewTime.setText(hms);
875.             onTick_Article4(millisUntilFinished);
876.         }
877.     }

```

LISTING 7

SoundAnalysis.java — `CounterClass` class

it has been initialized correctly. If another activity, such as our audio record playback, is already using the media recorder, we won’t be able to initialize it again. The initialization

LISTING 8

SoundAnalysis.java — Pre-execution

```

900. private class CaptureAudio extends AsyncTask<Void,
      Integer, Integer>{
910.     protected void onPreExecute(){
911.         int bufferSize = 2 * AudioRecord.getMinBufferSize
              (getResources().getInteger(R.integer.sample_rate),
              getResources().getInteger(R.integer.num_channels),
              AudioFormat.ENCODING_PCM_16BIT);
912.         recorder = new AudioRecord(AudioSource.MIC,
              getResources().getInteger(R.integer.sample_rate),
              getResources().getInteger(R.integer.num_channels),
              AudioFormat.ENCODING_PCM_16BIT,bufferSize);
913.         if(recorder.getState() !=
              AudioRecord.STATE_INITIALIZED){
914.             Toast.makeText(SoundAnalysis.this, getResources().
              getString(R.string.recorder_init_fail),
              Toast.LENGTH_LONG).show();
915.             recorder.release();
916.             recorder = null;
917.             return;
918.         }
919.     }

```

will then fail but we won't be warned. In most cases, the activity crashes the application when it runs, with error messages appearing on the LogCat. We want to save this precious time for hunting ghosts instead of debugging the application, so we create a message that pops up on the screen if the recorder initialization fails (Line 914). If the initialization fails, the warning toast message is displayed and the audio recorder is released (Line 915). Watch the order for the different initializations to avoid the Android equivalent of the Blue Screen of Death: "Illegal State Exception."

ASYNCTASK: DO IN BACKGROUND

In theory, all we need to do now is to start recording the noise. The buffer, which is internal to the AudioRecord instance, will be filled up with data. While recording, we need a number of operations to update the user interface. Once again, these operations don't have to be done in parallel with the recording task, which takes the most time.

Ghosts are not the only ones to do tasks in the background. We can also perform several tasks that don't have to be all synchronized with one another, using Android AsyncTask.

The `doInBackground` method is called to operate the background computation, which can take time (see **Listing 9**). We use it to initialize the number of records and TGBN sounds and the buffers (Line 952 to 958) before starting the record using the previously initialized `MediaRecorder recorder` (Line 959). Until the number of records remaining reaches 0, we save the recorded data into a buffer and make sure that this buffer is full before computing a result (Lines 963 and 964).

If an impulse above the background sound level is detected we start to collect data, "detectImpulse", decrease the number of records remaining, update the UI thread and fill the buffer `sampleBuffer` with the current data plus the data that have just been captured (Lines 965 to 972). During this recording, if a TGBN sound has been detected, "detectTGBN"

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.

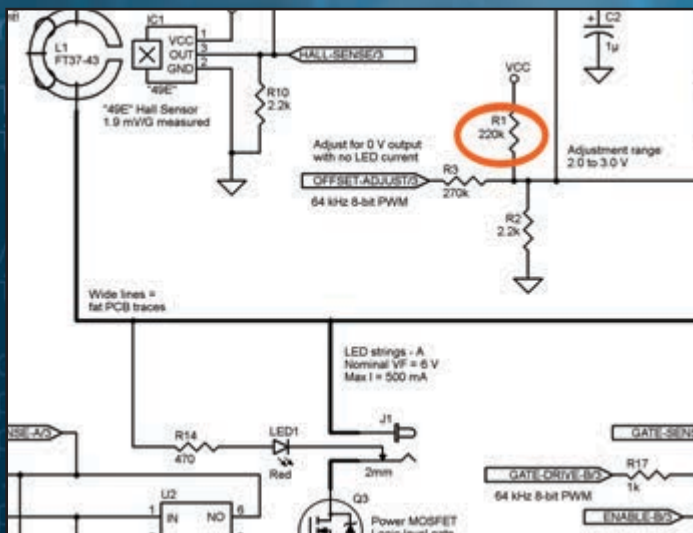
Mike Smith (Mike.Smith@ucalgary.ca) has been contributing to *Circuit Cellar* since the 1980s. He is a professor of Computer Engineering at the University of Calgary, Canada. Mike's main interests are in developing new biomedical engineering algorithms and moving them onto multi-core and multiple-processor 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.



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

```

950. protected Integer doInBackground(Void ... params){
951.     if(recorder == null) {return -1;}
952.     int remainingRecords =
953.         getResources().getInteger(R.integer.num_records);
954.     int numberTGBN=
955.         getResources().getInteger(R.integer.num_tgbn);
956.     int detectBufferLength = getResources().getInteger
957.         (R.integer.detect_buffer_length);
958.         //length = sampleRate * recordTime
959.     int sampleBufferLength = getResources().getInteger
960.         (R.integer.sample_rate) * getResources().getInteger
961.         (R.integer.capture_time);
962.         // SPOILER ALERT! UNCOMMENTED IN ARTICLES 4 and 5
963.     //sampleBufferLength=
964.     //nearestPow2Length(sampleBufferLength);
965.     short[] detectBuffer = new short[detectBufferLength];
966.     short[][] sampleBuffer = new short
967.         [remainingRecords][sampleBufferLength];
968.     recorder.startRecording();

969.     while(remainingRecords > 0){
970.         publishProgress(-1, -1, -1, -1, -1, -1);
971.         int samplesRead = 0;

972.         while(samplesRead < detectBufferLength)
973.             samplesRead += recorder.read(detectBuffer,
974.                 samplesRead, detectBufferLength - samplesRead);

975.         if(detectImpulse(detectBuffer)) {
976.             remainingRecords--;
977.             publishProgress(-1, remainingRecords, -1,0, -1, -1);
978.             System.arraycopy(detectBuffer, 0,
979.                 sampleBuffer[remainingRecords], 0,
980.                 detectBufferLength);
981.             samplesRead = detectBufferLength;
982.             while(samplesRead < sampleBufferLength)
983.                 samplesRead += recorder.read(sampleBuffer
984.                     [remainingRecords], samplesRead,
985.                     sampleBufferLength - samplesRead);
986.         }
987.         if(detectTGBN (detectBuffer)){
988.             numberTGBN++;
989.             publishProgress(1, remainingRecords,
990.                 numberTGBN,-1, -1, -1);
991.         }
992.         if(isCancelled()){
993.             detectBuffer = null; sampleBuffer = null;
994.             return -1;
995.         }
996.     } //end while(remainingrecord> 0)
997.     detectBuffer = null;
998.     sampleBuffer = null;
999.     if(recorder != null){recorder.release(); recorder = null;}
1000.     if(!isCancelled()){publishProgress(-1, -1, -1, -1, -1, -1);}
1001.     return 0;
1002. }

```

LISTING 9

SoundAnalysis.java — CaptureAudio -
doInBackground()

```

1200. protected void onProgressUpdate(Integer ... data){
1201.     if(data[0] == 1){
1202.         mediaPlayer.start();
1203.         Toast.makeText(SoundAnalysis.this, getString(R.string. TGBN_detected),
                        Toast.LENGTH_SHORT).show();}
1204.     if(data[1] != -1)
1205.         TextHandleNumberRecords.setText(getResources().getString
                (R.string.number_records_remaining)+ Integer.toString(data[1]));
1206.     if(data[2] != -1)
1207.         TextHandleNumberTGBN.setText(getResources().getString
                (R.string. number_tgbn_recorded)+ Integer.
toString(data[2]-1));
1208.     if(data[3] != -1)
1209.         timer.start();
1210.     if (data[4 ]!=-1) { /*IMPLEMENTED IN NEXT ARTICLE */ }
1217.     if (data[5] !=-1) { /*IMPLEMENTED IN NEXT ARTICLE */ }
1220.     }

1250. protected void onPostExecute(Integer data){
1251.     if(recorder != null){
1252.         recorder.release();
1253.         recorder = null;
1254.     }
1259.     }

```

LISTING 10

SoundAnalysis.java — CaptureAudio – onProgressUpdate()

```

1300. protected void onCancelled(){
1301.     if(recorder != null){
1302.         recorder.release();
1303.         recorder = null;
1304.     }
1305.     }

1306. int threshold = getResources().getInteger(R.integer.detect_threshold_impulse);
1307. int threshold_tgbn = getResources().getInteger (R.integer.
        detect_threshold_tgbn_sound);

1350. protected boolean detectImpulse(short[] samples){
1351.     for(int k = 0; k < samples.length; k++){
1352.         if((samples[k] >= threshold) &&(samples[k]< threshold_tgbn))
1353.             return true;
1354.     }
1355.     return false;
1356.     }

1400. protected boolean detectTGBN (short[] samples){
1401.     for(int k = 0; k < samples.length; k++){
1402.         if(samples[k] >= threshold_tgbn) {return true;}
1403.     }
1404.     return false;
1405.     }
1406.     }
1449. }

```

LISTING 11

SoundAnalysis.java — CaptureAudio – Detecting Impulses

LISTING 12

Preset string values must be set in strings.xml from WAT_AN_APP\res\values

```

200. <?xml version="1.0" encoding="utf-8"?>
201. <resources>
205. <!-- SAME AS Article 1 Listing 3, Lines 205 to 214 -->
220. <!-- SAME AS Article 2 Listing 4, Lines 220 to 224 -->
230. <!-- String required for the third part of the application, record and
      output Boo sound if TGBN sound is detected -->
231. <string name="error">An error has occurred</string>
232. <string name="recorder_init_fail">Recording device initialization
      failed</string>
233. <string name="press_to_start_detecting_TGBN">Press to start
      detecting TGBN</string>
234. <string name="press_to_start_WAT_A_RECORDER">Press to start
      recording</string>
235. <string name="number_records_remaining">Number of records
      remaining: </string>
236. <string name="number_tgbn_recorded">Number of TGBN sounds
      detected: </string>
237. <string name="TGBN_detected">a BOOm has been
      detected!</string>
238. <string name="title_activity_sound_analysis">Sound
      Analysis</string>
249. </resources>

```

is called. The counter tracking the number of TGBN sounds detected is increased by one and the scary sound is outputted (Lines 973 to 976).

To update the records and TGBN counters, we use “publishProgress” to publish the activity progresses on the UI. publishProgress works with OnProgressUpdate (see Listing 10), which is detailed later. The functions that detect the impulses, “detectImpulse” and “detectTGBN” are given an inputted short

type table containing all our samples (Lines 965 and 973). These two functions are also detailed later.

ASYNCTASK: UPDATE THE APP PROGRESS

onProgressUpdate displays the task progress on the UI, while the background task is still executing (see Listing 10). This method works in parallel with “publishProgress.” The values passed in parameters in

IMPLEMENTATION ISSUES

As of April 2015, we have tested out our code on two phones running on different versions of Android. The TGBN detector works perfectly on an older Samsung phone, which runs on Android KitKat (API 19) and has an SD memory card. We have also tried with a Google Nexus 5 phone running on two versions of Lollipop, API 21 (Lollipop 5.0.1) and API 22 (Lollipop 5.1). That phone does not possess a SD memory card needed to save data on external memory.

The reason why we mention recording for just “7 seconds” in this article is that the following often occurs on the Google phone running Android Lollipop 5.0.1 (API 21) after that time:

```
Error: fatal signal 11 (SIGSEGV) code 1 fault
address 0xa146f00c in tid 260 7(AsyncTask #3)
```

This problem seems to have occurred for a lot of developers given the net discussions. SIGSEGV is caused by a segmentation fault or access violation raised by hardware with memory protection. This error APPEARS fixed with the most recent Lollipop 5.1 (API 22).

Being only able to record 7 s for TGBN ghost BOOm sound busting is no big deal. After all, six recordings will give you 42, the answer to everything. However, our long-term plans are to record the Acoustic Ecology simultaneously in several home over a weekend or a week. For that, we need more equipment than our local acoustic firm friends have available to loan. We offer two solutions if you encounter the same issue using a phone running Lollipop 5.0.1.

There is a tool to help you fixing this kind of access violation errors. “NDK-STACK” allows analyzing stack traces as they appear in the output of the LogCat. Tutorials are available online (e.g., <https://yssays.wordpress.com/2011/12/27/android-ndk-stack-tool/>).

The easiest fixes: Hunt your old phone out of its drawer, keep using the KitKat OS, and keep searching the net. Or upgrade your more recent phone to the latest version of Android—Android 5.1. This bug is really annoying, and somebody with more skills will find a solution soon.

We are on our way to come full circle back to the start of Part 1 in this series. Thank heaven for that local teenager down the road!

`publishProgress` are stored in a data array that is given to the `onProgressUpdate` method. `publishProgress` is then presented as follows: `publishProgress(data[0], data[1], data[2], data[3])`.

We use `onProgressUpdate` to start the media player if a TGBN sound is detected and to display the message “a BOOm has been detected!” on the screen (data [0], Lines 1201 to 1203). It also decreases the number of impulses remaining if an impulse is detected and recorded (data [1], Lines 1204 and 1205), increases the counter if a TGBN sound has been detected (data [2], Lines 1206 and 1207) and starts the countdown once an impulse has been detected (data [3], Lines 1208 and 1209). The empty curly brackets in Lines 1210 and 1211 are needed to keep the compiler happy until we use the last two data items, `data[4]` and `data[5]`, passed to `publishProgress()` in the next article.

ASYNCTASK: POST EXECUTION

`onPostExecute` is invoked after the background task finishes (see Listing 9, Lines 1250–1254). We use it to release the recorder, as we are not recording any more at that point and to verify that no error has occurred during the asynchronous operations.


Listing 11 describes the `onCancelled()` method (Lines 1300–1305). If a task cancels, we simply need to release the recorder if it has not been done before. While working with asynchronous tasks, a task can be cancelled at any time by invoking `cancel(boolean)`. Invoking such a method implies that `onCancelled()` will be invoked instead of `onPostExecute()` after `doInBackground()` returns.

The functions used to detect impulses (Lines 1350–1356) and to detect a TGBN sound (Lines 1400–1406) are also described in Listing 11. To detect an impulse and start recording, we process our samples one by one and compare them to two thresholds set to 10000 and 25000 (Lines 1351–1354). The samples must be within this range for the recorder to start. At the end of each recording, if these data are greater than or equal to a threshold set to 25000, Line 1402, the `detectTGBN` function is called and we have detected one of the ghosts mucking about in the room. Our detector then increases the TGBN sound counter and outputs the scary sound.

To avoid compiler errors, update the Preset Strings in `strings.xml` as shown in **Listing 12**. For more information, functions and examples concerning asynchronous tasks on Android are available at developer.android.com/reference/android/os/AsyncTask.html.

SPECTRAL ANALYSIS

We are now done with developing a long-term, sound-gathering device for our community noise nuisance task. By side-tracking into our ghost detection mission and practicing some finger yoga activities, we hope you have had a few chuckles while getting familiar with the world of Android development.

Nevertheless, our job as noise detectives has just started. In the next articles, we are going to work on “spectral” rather than “sound” analysis. We will implement a fast Fourier transform (FFT) algorithm to transform our recorded data into the frequency domain, before graphing the results. This will enable us to develop a tool displaying the time and frequency domain analysis of a sound. After tracking sounds and proving their existence in a room, we can now start studying them in detail. Perhaps finding those room resonances we mentioned in Part 1 of this series are just another way of detecting the nests that ghosts build in our homes! 

Authors’ Note: Recently, Orchisama Das (Jadavpur University, India) joined Mike and Adrien as a summer research student with travel funds from the Mitacs Globalink international exchange program (www.mitacs.ca/en/programs/globalink). Her work was to upgrade the WAT-AN-APP “Sound Ecology” App to use a database to store and analyze multiple sound records. (A forthcoming Circuit Cellar articles series is in the works for 2016.)

To help proofreading the WAT-AN-APP Circuit Cellar articles, Orchi produced a Java program running inside Eclipse to allow the readers to cut-and-paste the code from listings in Circuit Cellar article PDF files directly into .jar files and automatically remove the line number comments. The “WANT-A-FASTER way to code WAT-AN-APP” can be found on the Circuit Cellar FTP site.

Then interesting things started to happen! To publicize the work of their not-for-profit organization, Mitacs put out a press release about the neat projects undertaken by the 40-plus students in their 2015 international exchange program at the University of Calgary. Suddenly, Orchi and Mike were being interviewed for TV, radio, and in hardcopy (front page coverage) about the current and future plans for the Android Sound Capture WAT-AN-APP and the “realness or otherwise” of local community noise nuisances! Interested readers can find details of local and national coverage on the “Ranchlands’ Hum” tab at People.Ucalgary.ca/~TheHum or by typing “Ranchlands Hum Smith Orchi” into a search engine.

Power Over Ethernet Solutions



Powering devices over Ethernet cabling seems easy, but there's more to it than meets the eye. Eddie explains how it all works.

By Eddie Insam (England)

Editor's Note: This article first appeared in Circuit Cellar 195, 2006.

So you've designed a brand new Ethernet-based device. Perhaps it's a clock, a weather sensor, or an industrial controller device. You plan to hang it proudly on your wall and connect it to a RJ-45 wall socket. But how are you going to power it? Where will the system get its juice? Surely, you aren't going to disgrace your design with a brick wart. There must be a better way!

Why not feed power over the CAT-5 cable? Well, you're not the first person to consider this technique.

Standard CAT-5 cable has four pairs, and only two are used for data in a typical 10- or 100-Mbps installation (see **Figure 1a**). So, it sounds obvious to stick a few DC volts down the spare pairs. Oh, yes. But hang on, life is never so simple. This is technology, remember? There has to be a catch somewhere. So, sit down and relax, I have the story.

It may not come as a surprise that the wise men at the IEEE thought about this for a while and came up with a standard (IEEE 802.3af). This standard has been around since 1999, but progress has been relatively slow. It started to take off only recently, mainly

because of the availability of inexpensive specialist components. Tom Cantrell and Jeff Bachiochi have covered some of the available components and modules (*Circuit Cellar* 165 and 187). A wide range of parts are now available, including dedicated switching transistors, isolation transformers, and high-quality nonsaturating magnetics, making power over Ethernet (PoE) a practical proposition.

TECHNICALITIES

The IEEE document covers two main methods for sending power down the CAT-5 wire. One involves using the spare pairs. The other involves sharing with the existing data lines using center-tapped transformers (see **Figures 1b and 1c**). The latter method is beneficial when spare cable capacity isn't available.

The method involving spare pins allows a decent amount of current to be drawn because the two spare pairs are paralleled together to increase capacity by reducing the total DC resistance. The present IEEE specifications allow up to 13 W of power to be transferred

this way. This may not be enough for some heavy-duty devices, but it's quite acceptable for medium-size and small items such as TV cameras and VoIP phones. An updated PoEPlus standard is currently being considered. This will allow for up to 30-W capacity, while still remaining backwards compatible.

Transmitting power with center-tapped transformers is more limited. Pulse transformers and other magnetics in the Ethernet controller must be designed to take the full DC power load current without saturating. That isn't an easy task for miniature surface-mounted components. The advantage of this alternative is that it leaves the extra pairs alone, an essential consideration in higher-speed gigabit Ethernet, which requires all four pairs to carry data.

POWER SUPPLY

Why can't you just stick any old power supply across the spare wires? Because you don't know what's at the remote end, and you may run the risk of blowing up sensitive equipment. If you don't believe me, take a look at **Figure 2**, which is a typical Ethernet terminator. This kind of circuitry is sometimes contained within a single metal enclosure called a MagJack. Note the two 50-W resistors R3 and R4 across the center taps of transformers T3 and T4. They are branched in series to form an effective 150-W DC load across the input lines. Also note the two 50-W resistors R1 and R2 right across pins 7 and 8 and 4 and 5. These present a controlled impedance load to the otherwise non-terminated wires. They are there for robustness and noise reduction. This hookup is sometimes known as a Bob Smith termination.

If you connect a 48-VDC raw supply into such a socket, you will be driving a good third of an amp through these tiny resistors. This is guaranteed to vaporize them to kingdom come. Tiny SMD resistors are not built for such treatment.

Admittedly, some terminators and MagJacks have extra series capacitors to protect the resistors, and not all Ethernet devices use such extra networks. However, you don't want the power supply to blow the other devices that have them.

There are other potential problems that can be blamed on bad design or pure accident. For example, a wireman could accidentally short or swap the CAT-5 pairs. All possibilities have to be considered, and many are mentioned in a 1999 IEEE report entitled "DTE Power Problem Set and Solution Methodology."

Needless to say, the good people at the IEEE have devised cunning schemes to preempt the aforementioned challenges. In

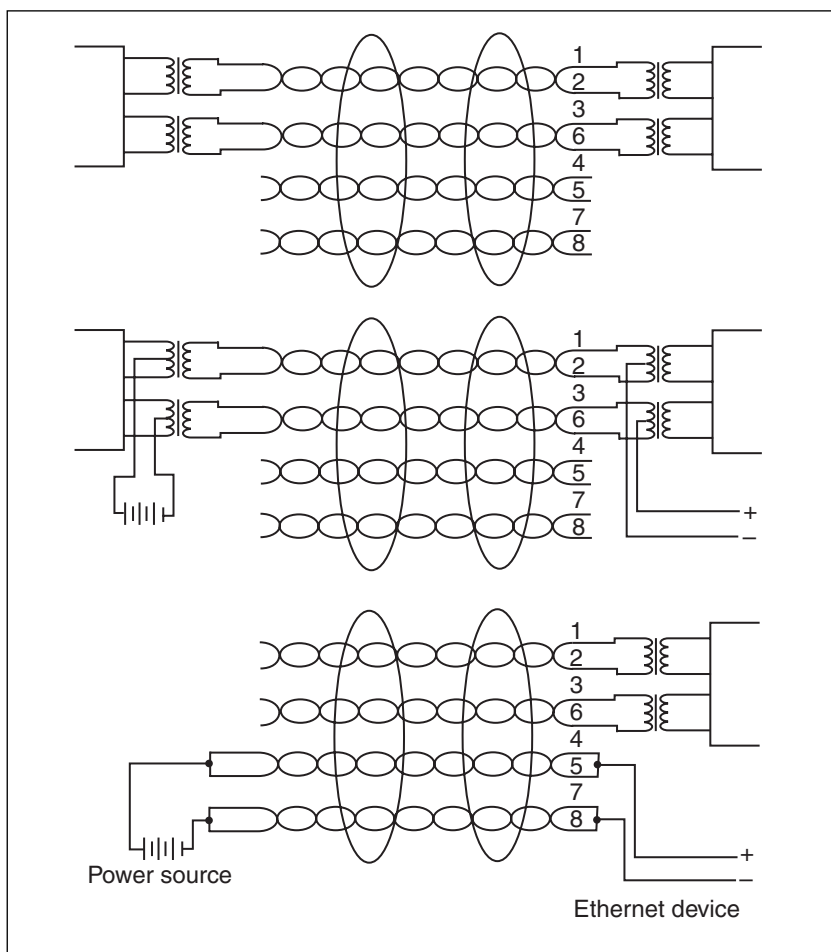


FIGURE 1 Standard 10- and 100-Mbps Ethernet devices use just two of the four available pairs. The spare wires can be used to transmit power to the remote. Two possible methods are shown (b and c). But watch out! The power source must be smart enough to detect shorts and overloads and to avoid damaging components at the far end.

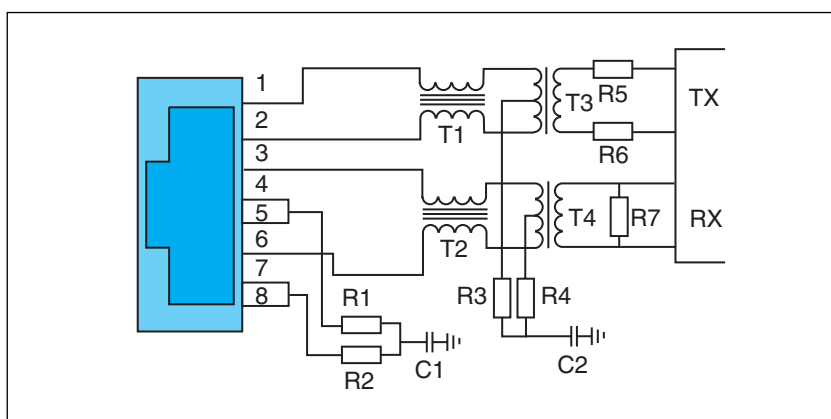


FIGURE 2 This is a typical Ethernet termination. The resistors strapped to the spare data pins and center taps are there to balance the line and to reduce noise. They can quickly flash to smithereens in true Harry Potter style if any unmanaged DC power is placed on the cable.

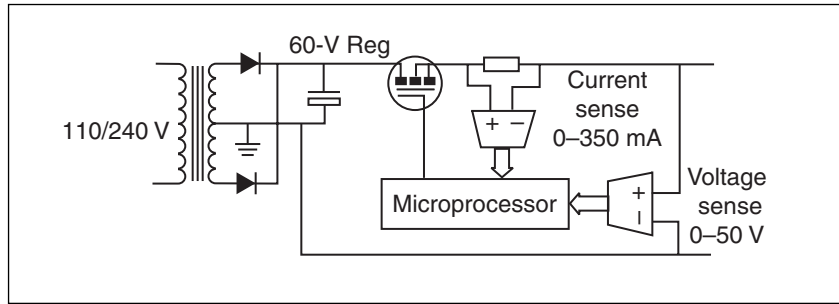


FIGURE 3

A power supply will include a microcontroller in a standard design configuration to sense load current and generate output voltage levels accordingly.

simple terms, the smart power supply can figure out what's happening at the load end. It does this by taking a number of graded impedance measurements before applying full power. These impedance signatures tell the supply whether or not it's safe to apply full power. Full power is applied only when it's safe to do so. Furthermore, the load is regularly monitored during normal operation to ensure nothing drastic has happened. This allows the supply to turn off the wick if it detects any suspicious problems, when the load fails, or when it is disconnected. This arrangement, of course, needs cooperating equipment at the load end to provide the right dummy impedances at the right time.

Apart from the safety factor, the IEEE standard helps to reduce overall energy loss, because only those sockets that have a valid load can be programmed with power. During sensing, the supply knows the range of power loading taken by a load, and it ensures that

the correct amount of current is delivered (within a reasonable range). No more, no less.

PC-CONTROLLED POWER?

So, does the power supply need to be computer controlled? Well, yes, but what isn't nowadays?

The operating algorithm is relatively straightforward, and even the tiniest microprocessor can handle it. You just need a power supply that can deliver a programmable voltage between 2 and 48 V, a means of sensing the load current, and a means of measuring its output voltage from which you can compute the load impedance and various other parameters. The rest is just software. Mind you, and as you would suspect, the IEEE standard is not that straightforward. Many options are included to cater for all eventualities. For example, there are options for sensing an AC load as well as the DC load, but many of these are just optional enhancements. You can get away with just sensing a plain DC resistive load. **Figure 3** shows what the supply looks like.

How about the load end? The power source does its validation by sensing the impedance of the load at different source voltage levels. While this is taking place, the load needs to behave a bit like a nonlinear resistor, which is otherwise called a signature impedance (see **Figure 4**). The circuitry to do this is relatively simple, and there are a number of ICs that will do the job for you. The basic circuit is best

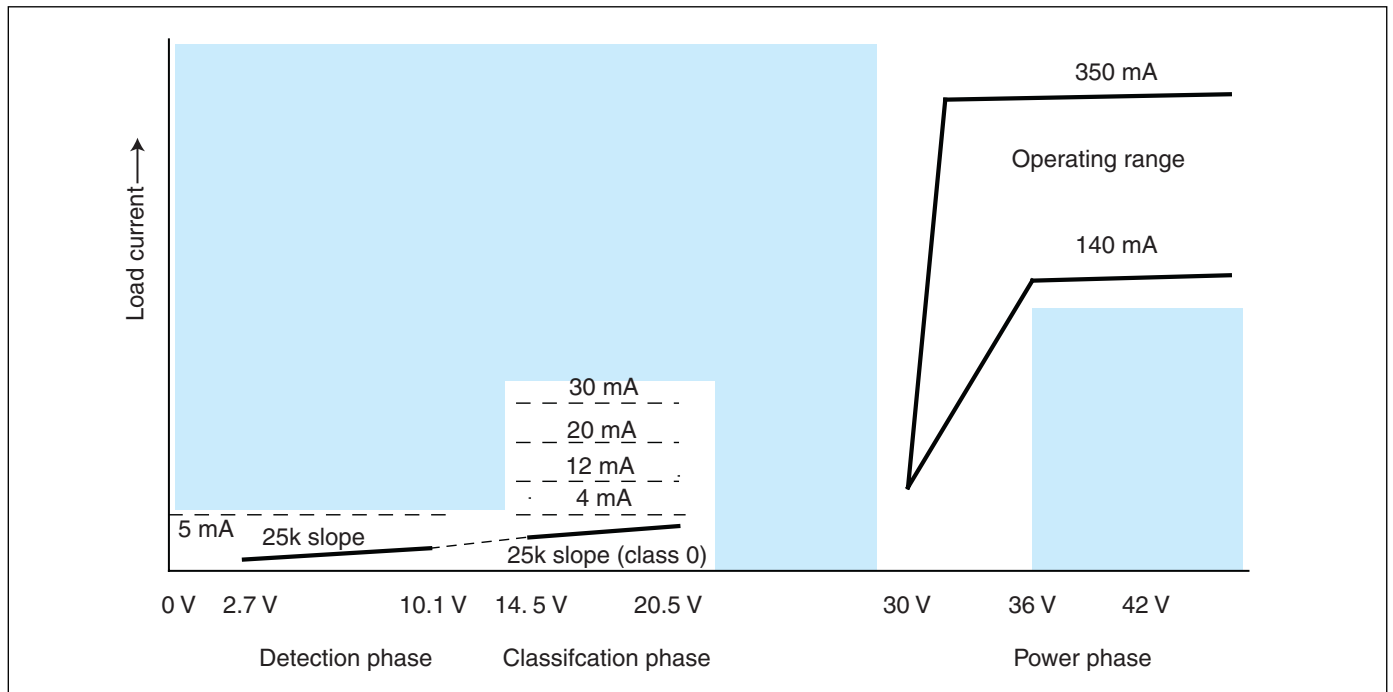
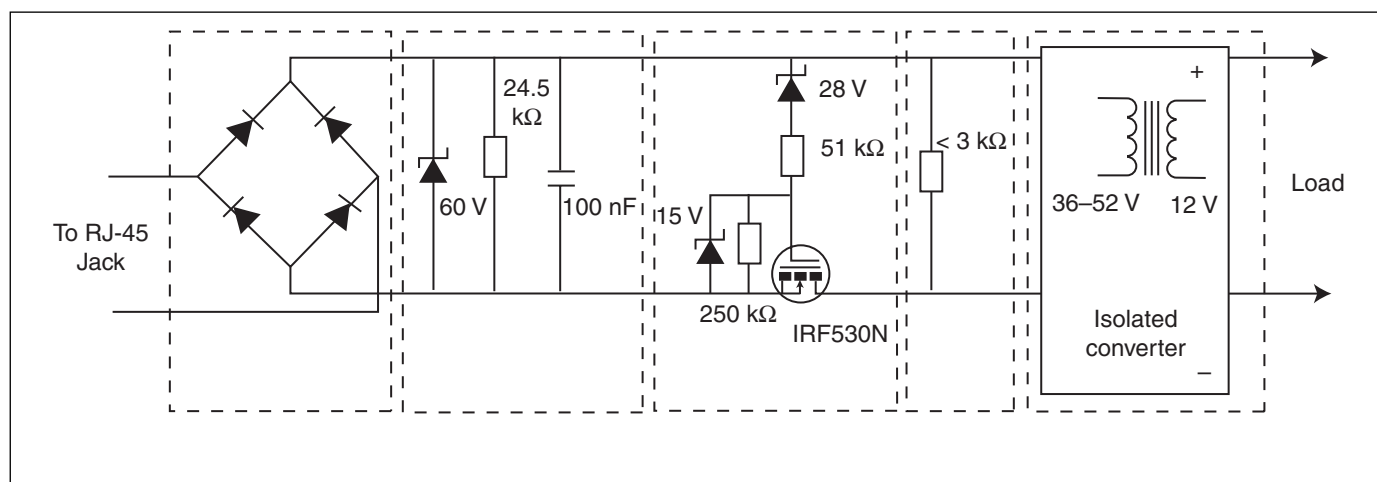


FIGURE 4

There are three distinct phases. The simplest of loads will present a 24.5-k Ω resistance until the input voltage rises above 30 V, at which point the actual driven circuit will be switched into operation.



described in terms of discrete components. **Figure 5** shows the basic principle.

HOW IT WORKS

First, I need to introduce some jargon. Don't forget that I'm talking about IEEE standards, so the use of jaw-churning technospeak is essential.

Power sourcing equipment (PSE) is a term for the source end, or power supply. Powered device (PD) is the equipment at the user end or load. An endpoint feed describes the arrangement or situation where the power supply is fitted inside the source box (e.g., inside an Ethernet router), so only one cable link is needed between the router and the PD.

A midspan feed unit (MFU) is a separate box that's added somewhere between the router and the PD to provide the power. This necessitates two CAT-5 links, one between the router and the MFU, and another between the MFU and the PD. You need to buy an MFU if you already have a router that doesn't provide PoE. If you start from scratch, you may prefer to buy a router with a built-in endpoint feed. Are you still with me? Don't go away. There's more.

The voltage level at the power supply is specified as between 44 and 57 V, whereas this is widened to 36 and 57 V at the user end to allow for reasonable ohmic drop down the CAT-5 cable. The PSE is allowed to supply up to 15.4 W of power with a maximum current limit of 350 mA. The maximum power consumption at the PD is about 13 W, which corresponds to a nominal current of 270 mA at 48 V. CAT-5 runs can be considerably long, and a lot of ohmic loss can be expected. This is one of the reasons why the standards suggest that pairs 4 and 5 and 7 and 8 should be paralleled together to halve the cable's resistance.

Although the specifications define which pin should be positive and negative, the load must not assume anything. Murphy's law! The PD must also ensure that the internal supply is

floating with respect to the input power feed. So, it needs to include a bridge rectifier on the input plus a floating transformer-isolated power converter.

So, how does it work? Let's take it in stages. Take a look at Figure 4. When there is no load applied (i.e., when the user end PD is disconnected or during first power on), the source (PSE) repeatedly sits in a short loop sensing the line for an ohmic signature. This is the detection phase. It does this by placing at least two spot voltage levels between 2 and 10 V and then measuring the line currents drawn at these points. The current difference is taken rather than the absolute values because this makes for a more precise derivation of the signature impedance. It also compensates for fixed losses such as diode drops. A current limiter on the line ensures the load can draw no more than 5 mA just in case there is a short or similar problem.

The two test voltages are changed relatively slowly to avoid any glitches. The specifications suggest between 2 and 500 ms between readings. During the detection phase, the load has to present a 24.5-kW resistive component in parallel with a 0.1-mF capacitor. This is not a real component value; it's a theoretical average. You can't buy 24.5-kW resistors in the shops. To be more precise, any load between 23.75 and 26.25 kW is considered valid. Loads below

Class	Load by PD	Usage	Power range
0	0–4 mA	Default	0.44 to 12.95 W (full range)
1	9–12 mA	Optional	0.44 to 3.84 W
2	17–20 mA	Optional	3.84 to 6.49 W
3	26–30 mA	Optional	6.49 to 12.95 W

TABLE 1

Take a look at the PD power classification scheme. This allows the supply to provide only as much power as the device demands.

FIGURE 5

Take a look at the operation of a typical PD in stages.



PHOTO 1

The D-Link DWL-P50 is a ready-to-go module. Ethernet in, Ethernet out, and a choice between 12- and 5-VDC outputs.

15 kW or above 33 kW are considered invalid. Loads outside of these two ranges are in no man's land and may or may not indicate the presence of a (possibly faulty) PD.

If this all sounds confusing, it's because this is the way standards tend to specify things that need to lie in ranges. Mere mortals like us need to know only that the resistance needs to be about 25 kW. The capacitor is required for an optional alternative AC load sensing method. I'll cover this later.

When the 24.5-kW resistor is detected, the PSE proceeds to the next stage: the classification phase. If at any point the load measures too low or too high, the PSE assumes there is no valid termination and removes the power altogether. It then waits a couple of seconds and then starts again from the detection phase, repeating the cycle forever. In the worst case, an incompatible or bad PD will see a maximum of 10 V or 5 mA applied across it and no harm will be done. This is somewhat more preferable than being hit with 48 V at full current!

The purpose of the classification phase is to determine the range of load currents the user device will need. In other words, the PD tells the PSE how much current it is going to need. The use of limited power ranges could be useful for loads that need critical monitoring or to avoid users connecting unauthorized devices to certain sockets. A main application for this is to allow limited resource PSEs to allocate different power levels to different outlets or to allow the PSE to enable only certain PDs in case of an emergency or other priority. In practice, however, this may create more problems than it can solve. **Table 1** shows some of the available options.

During the classification phase, the PSE applies two or more voltages between 15.5

and 20.5 V (current limited to 100 mA) and measures the new signature impedance. The PD recognizes these new voltage levels and switches in a suitable load resistor according to its expected needs. Note that if the PD retains the original 24.5-kW resistor, it will be classified as Class 0 and default to full-power range, which is very convenient. In other words, the simple do-nothing option will give you the full power range. Who says committees never come up with practical ideas? The PSE will have a further chance of detecting improper loads or shorts during this stage. It will remove the power altogether if anything feels suspicious.

Having passed the classification phase, the PSE can now slowly ramp up to full power, so the voltage now goes up to the 48 V per 300 mA current limit. At the same time, the PD will connect the line to its internal circuits powering the user electronics. After this new stage and while providing full power, the PSE will constantly monitor the load for current drawn. The PD will guarantee to sink a defined maintain power signature (MPS). In other words, if the load current rises above 400 mA at any time or drops below 10 mA for than 250 ms, the PSE will assume the load has gone funny, kill the supply, and revert to its detection phase as before. There is a defined back-off period of 2 s to avoid the entire thing going into wild oscillations.

A well-known scenario to be avoided is when a valid PD device has just been unplugged from an Ethernet wall socket and a legacy device is plugged in immediately after. If the PSE doesn't recognize this situation quickly, it can damage the legacy device because full power is still being applied to the line. This is where the alternative AC sensing method scores. A 500-Hz AC common-mode signal is superimposed on the DC. Any AC disconnection can be detected immediately, whereas a DC disconnection has to rely on slow voltage decays before it can be correctly detected. Note that the supply can optionally use either AC or DC sensing, but the load must include methods for supporting both. In practice, this is just a 100-nF capacitor in parallel with our beloved 24.5-kW resistor.

THE LOAD'S JOB

During initialization, the PD presents a variable impedance to the supply depending on the input voltage across its input pins. Between 0 and 10 V, the load looks like a 24.5-kW resistor (plus the voltage drop effects of the bridge rectifier). Between 10 and 20 V, it can still be a simple resistor, but it's calculated to give the current load specified in Table 1. Alternatively, it can keep the same 24.5 kW to respond for Class 0 and the full power

When it comes to robotics, the future is now!



From home control systems to animatronic toys to unmanned rovers, it's an exciting time to

be a roboticist. *Advanced Control Robotics* simplifies the theory and best practices of advanced robot technologies, making it ideal reading for beginners and experts alike. You'll gain superior knowledge of embedded design theory by way of handy code samples, essential schematics, and valuable design tips.

With this book, you'll learn about:

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

ADVANCED CONTROL ROBOTICS

HANNO SANDER

AD

Urbus a
ndpapae
berum de
mi nes de
lique sum
ad et ut re
ta nos dolo
voluptas as me

Os praes ma
sum as sim re
rum ne aliqu
eae part et que
nis ut accusa
ent magna
vellest unsect
tu?

Obis si emende
enim doloipon
teci audore
audand primis
de perunt rem
accatus sin et
consed que no
forum pendi
tequi ad et sim
item quibus
ad bla naton

HANNO SANDER

start
start task **async**

repeat
start task **sync** and wait

task **sync**
print text " "
wait **1000**

task **async**
repeat
print text " "
wait **1000**

Get it today at cc-webshop.com.

range. As the input voltage ramps up between 30 and 42 V, the user load is switched in. If during full power the input falls below 36 V, the PD disconnects itself from the supply. This is known as under voltage lock out (UVLO).

It's the PD's responsibility to ensure that the load doesn't take more than the rated power or less than a minimum threshold current to make sure it doesn't get turned off. This minimum current is specified as 10 mA for at least 75 ms in every 325 ms. Unplugging the PD can then be easily recognized by the PSE as it sees the current drop below 10 mA.

The disadvantaged products in this scheme are low-power devices that need to include a bleed resistor just to ensure that the minimum current threshold is met. So much for energy conservation!

ABOUT THE AUTHOR

Eddie Insam (edinsam@eic.co.uk) lives next to the Thames in southern England. He has been designing specialist signal processing and telecom systems for more than 20 years.



circuitcellar.com/ccmaterials

RESOURCES

J. Bachiochi, "Power Over Ethernet Primer," *Circuit Cellar* 187, February 2006.

T. Cantrell, "Powered Points," *Circuit Cellar* 165, April 2004.

IEEE, IEEE Standards Interpretation for IEEE Std 802.3af-2003, IEEE, New York, NY, 2005.

Maxim Integrated Products, "MAX5941A/MAX5941B: IEEE 802.3af-Compliant Power-Over-Ethernet Interface/PWM Controller for Power Devices," 19-3069, rev. 4, 2006, <http://pdfserv.maxim-ic.com/en/ds/MAX5941AMAX5941B.pdf>.

M. McCormack, "DTE Power Problem Set and Methodology," 1999, www.ieee802.org/3/power_study/public/nov99/mccormack_1_1199.pdf.

SOURCES

DWL-P50 PoE Adapter
D-Link Corp. | www.dlink.com

LTC4257 PoE Interface controller
Linear Technology Corp. | www.linear.com

LM5070/71 PoE PD Interface and PWM controller
National Semiconductors | www.national.com

Power modules
Recom International Power | www.recom-international.com

HV110 PD Controller IC
Supertex, Inc. | www.supertex.com

TPS2370 Power interface switch
Texas Instruments, Inc. | www.ti.com

TYPICAL PD

Figure 5 shows a most basic PD. It has been divided into sections to show the relative responsibilities. Figure 5a shows a bridge rectifier. It's always good practice to use a bridge in case the wires have been swapped around. A PD can make use of both alternative sources by having two bridges, each connected to the two power options shown in Figures 1b and 1c.

Figure 5b shows the main 24.5-kW signature-sensing resistor and a 100-nF capacitor to provide an AC signature load. There is also a 60-V Zener diode to provide some sort of overall protection. (An extra fuse connected between this and the input line would not come amiss.) In this simplified circuit, the classification phase is also managed with the same 24.5-kW resistor, classifying the unit as Class 0.

Figure 5c is a simple gated switch that turns the load on when the input voltage reaches about 30 V. Figure 5d denotes that the load has to sink at least 10 mA. Figure 5e represents a 36- to 42-V converter, which must be floating (e.g., transformer isolated). Modules in the RECOM International Econoline series are typical examples. They are small potted modules (e.g., an RS4805 that takes 36 to 72 DC input and 5 V at 200-mA output all in a small SIL footprint).

TYPICAL PSE

Figure 3 shows a basic PSE design. Of course, you'll usually use a premade PSE rather than make your own.

The supply consists of a decent 48-VDC power supply and a series regulator controlled by the D/A output from a microprocessor (possibly via a PWM output). The series resistor emulates the current limit and a place to take a sample of the current drawn. One such controller is needed for each Ethernet line or RJ-45 outlet.

The design is pretty straightforward because accuracy is not primordial. One tricky part of the design is the wide-ranging metering of the output current, which needs to cover a range of 100 mA to more than 300 mA. This necessitates either a high-resolution (14-bit) ADC or a means of switching in different shunt resistors for the different ranges.

Note that the series pass transistor won't need much heat sinking. It will normally be operating either fully on (when delivering full power) or at limited current during the initialization phases. The software consists of a simple timed loop to cover the detection, classification, and power delivery phases one at a time. The IEEE 802.af document describes procedures for implementing a version of this flowchart if you have the time and inclination

to decipher the gripping notation and methodologies used.

INTEGRATING POE

Of course, you may not be interested in making your own circuits. There are plenty of ready-made chip and module solutions available out there to make it all easier. But understanding the principles involved will ensure that you won't get caught in many gotchas!


The MAX5940/1 was one of the first kids in the block. These chips provide all of the 802.3af interface detection, classification, and switching facilities. One of the chips is normally used in conjunction with a separate Maxim 48-V switching down regulator (MAX5014) to provide a complete power supply function.

National Semiconductor's LM5070, LM5071, and LM5072 are typical of the all-in-one-chip solutions. They integrate a current-mode DC-to-DC controller, user-programmable under-voltage threshold, a fault current control loop, and many other functions. The LM5071 and LM5072 can accept power from an external AC/DC adapter (a wall wart).

The Texas Instruments TPS2370, TPS23750, and TPS23770 are also big

contenders. They combine the functionality of the older TPS2375 controllers and need a minimum number of external components. Similar devices are also available from Linear Technology (LTC4257) and Supertex (HV110).

Chip solutions are also available for the PSE end. Some of these have multiple controllers, which allow four, eight, or even 12 power supply controllers from one chip. Current devices are the Maxim MAX5945, the Texas Instruments TPS2383, the Linear Technology LTC4258, and the PowerDsine PD640xx series. For instant satisfaction, check out the PowerDsine 3001 (a single port mid-span supply) and the corresponding D-Link DWL-P50 end load adapter, which are considered complete modules. The latter comprises a floating supply that can generate either 5 or 12 VDC at the flick of a switch (see **Photo 1**). This pair can provide a relatively inexpensive solution for small PoE needs. Similar products are also available from suppliers such as Hyperlink Technologies.

You're sure to see many more PoE solutions in VoIP phones, CCTV cameras, and industrial Ethernet applications. Integrating a PoE supply into a module will be commonplace. 

Get PUBLISHED. Get NOTICED. Get PAID.

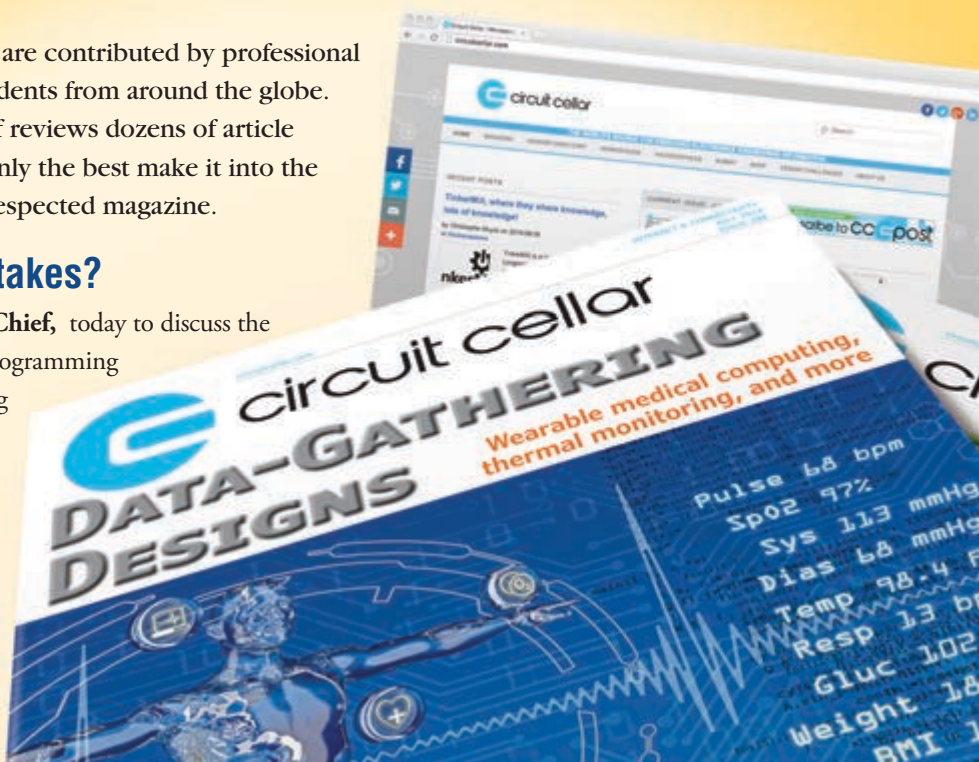
Circuit Cellar feature articles are contributed by professional engineers, academics, and students from around the globe.

Each month, the editorial staff reviews dozens of article proposals and submissions. Only the best make it into the pages of this internationally respected magazine.

Do you have what it takes?

Contact **C. J. Abate, Editor-in-Chief**, today to discuss the embedded design projects and programming applications you've been working on and your article could be featured in an upcoming issue or online at circuitcellar.com.

Email: editor@circuitcellar.com



GREEN COMPUTING

Sustainable Big Data Analytics in the Cloud

By Ata Turk (US)

Data is the oldest renewable resource and thanks to the increase in our computing efficiency and capacity, we finally started to harvest and process it in big quantities. It is imperative to come up with sustainable processing mechanisms to make the most out of this valuable resource and if possible use its power to improve sustainable practices. This article investigates whether the current practices in big data analytics are sustainable or not.

Data is a renewable resource. As long as the universe and the intelligent mind to observe it exist, this resource will not deplete. Moreover, it is infinite. The more you dig, the more you get; but as with all resources, there is a cost/benefit cut point. In this aptly named "Information Age" where we are generating and harvesting data in an unprecedented rate, making the most benefit while processing as much as possible data in the most efficient manner is paramount.

Unfortunately, we are also in an age where the level of contamination in the world has observed unprecedented increases and reached to critical levels. It is clear that we cannot maintain our current development and processing methods and very likely, the era following the Information Age will be the "Sustainability Age." All main industries, including the data processing industry, will have to develop and adopt sustainable methods.

BIG DATA ANALYTICS & CLOUD COMPUTING

The process of extracting gains from huge quantities of data is called "big data analytics" and delivery of on-demand computing resources over the Internet on a pay-for-use basis is called "cloud computing." Big data analytics and cloud computing are beyond their hype stage and well into their maturity. They ascended separately in popularity in similar timeframes by providing positive feedback to each other's import.

Big data analytics was initially driven by the massive increase in the enterprise data

volume caused by the shift to mobile and social mediums and the commercial desire to make use of this large volume of data, but it has now gained wide acceptance in various other fields such as oil drilling, telecommunication, credit management, architecture, and datacenter design. Similarly, cloud computing was initially spurred by the growth in Internet-based communication, commerce, entertainment, and management solutions, but it has also penetrated diverse markets such as healthcare, automotive design, financial services, and governance.

There is a gift relationship and constant forward-backward feedback between these two game changing paradigms. As big data analytics solutions promise untapped gains to be obtained from existing enterprise data sets, many large companies started to employ various sized big data analytics frameworks and most of these frameworks are running on cloud computing infrastructures. Similarly, as cloud computing solutions made collecting, storing, and processing of large quantities of data cheaper and available, many large entities started to store previously discarded datasets, which are then used as motivations for employment of big data analytics frameworks.

BIG DATA ANALYTICS & SMART POWER

Energy industry is among the biggest industries that started to utilize the power of big data analytics for intelligent decisions and improvements in efficiency. The available

capacities for new renewable energy resources such as solar and wind are too low and also the current cost of ownership for these resources are relatively high making their adoption difficult. Currently, solutions that improve energy efficiency offer the largest possible gains for attaining sustainability in energy usage.

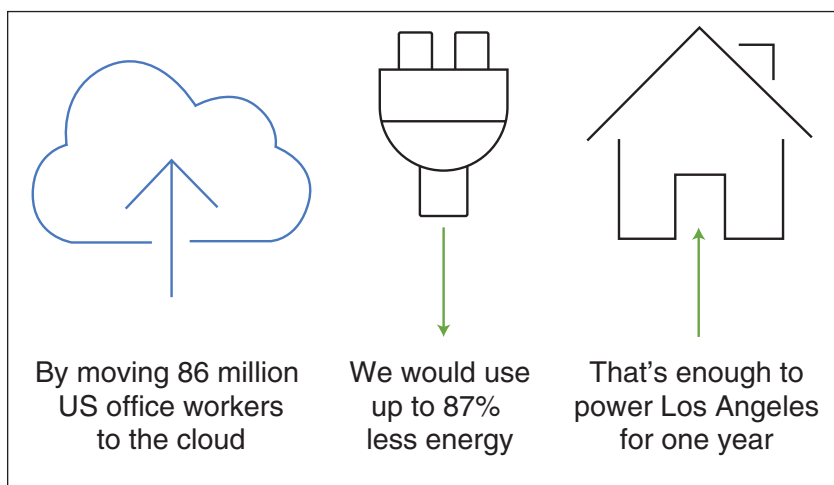
By making use of smart meters and smart grids and intelligently processing the enormous amount of data created by these technologies, energy providers can now do the following: forecast demand and determine the need for new investments; predict and prevent outages by identifying stressed portions of the energy grid and organize maintenance activities accordingly; and devise and offer incentives for shaping customer usage by increasing user energy usage awareness and offering varied energy pricing mechanisms to cost-savvy customers. Some example practices employed by energy providers include modeling wind patterns to optimize the location and design of new wind turbine sites, developing apps that tell users how and where they use electricity, and letting users compare their electricity usage with their neighbors to encourage them reduce wasted energy.

SUSTAINABLE BIG DATA ANALYTICS

Even though big data analytics can offer improvements in energy efficiency, it still requires huge amounts of data to be stored and processed, which itself requires energy. The environmental impacts of the infrastructures used for supporting big data analytics and cloud computing solutions increase dramatically as the number and capacity of datacenters providing the necessary services for these products increase with their popularity.

It is undisputable that datacenters are major power hogs. According to Department of Energy, there are already more than 3 million datacenters in US, and these datacenters consume 2% of all US electricity use. According to GreenPeace, datacenters all around the world are responsible from 2% of the global carbon emission, a carbon footprint paralleling that of the aviation industry.

Note that computing in the cloud generally provides better utilization of resources and hence can be argued to lead to lowered environmental impact. Cloud computing is applauded for enabling: resource scaling with changing demand; improved utilization with virtualization and multi-tenancy; and economies of scale that lead to reduced management overhead, reduced power loss, and hence reduced costs. A recent study



from Lawrence Berkeley National Laboratory indicates that the amount of energy saved by businesses moving to the cloud can be as high as 87% (see **Figure 1**).^[1]

Furthermore, most major cloud providers started to engage in transforming the energy used in their datacenters to greener renewable energy solutions. As an example, Apple's cloud datacenters are 100% powered by renewable energy sources. Google engages in long-term power purchase agreements with renewable energy providers that are in the same region with their datacenters. This approach enables financing of new renewable energy installation projects by providing long-term financial guarantees. Microsoft and Intel are running carbon neutral by procuring green power equal to the volume of their consumption of electricity as well and other cloud providers such as Rackspace and Amazon also made commitments to becoming carbon neutral and switching to 100% renewable energy sources within few years. As seen in these examples, the energy spent in the datacenters act as locomotion for the investments made in renewable energy sources.

In addition, cloud datacenters can participate in power regulation programs of electricity providers and help service operators in matching electricity supply with the power demand. As datacenters are huge energy consumers and some of the computations performed in them have relatively flexible timing constraints, they can regulate their power usage to a certain extent to participate in these programs to gain monetary advantages and to ease the burden on infrastructure providers. Check Ayse K. Coskun's articles "Data Centers in the Smart Grid" (*Circuit Cellar* 286, 2014) and "Application-Aware Power Capping" (*Circuit Cellar* 280, 2013) for more details on the power regulation opportunities on datacenters.

FIGURE 1

Moving to the cloud can save up to 87% of IT energy.^[1]

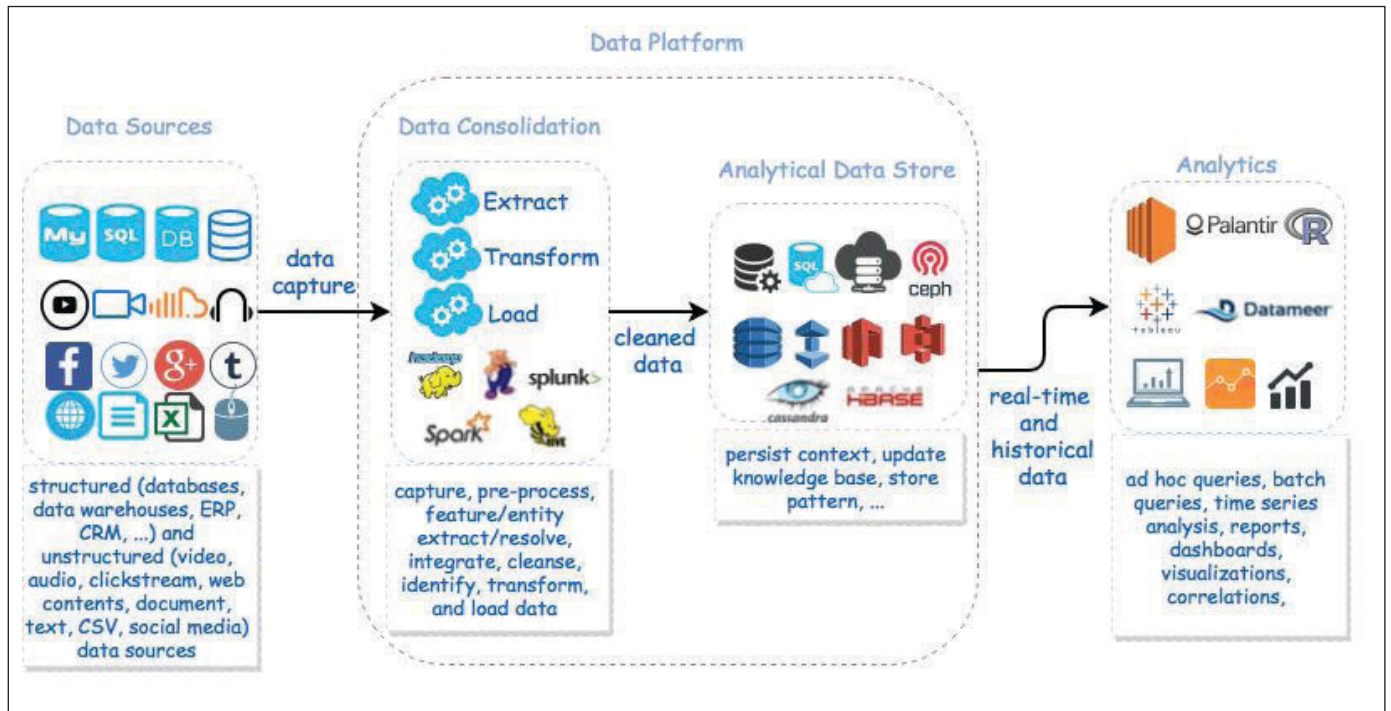


FIGURE 2

A standard big data analytics platform workflow: Structured and unstructured data sources are gathered and consolidated into an analytical data store. Analytics tools use real-time and historical data coming from the data platform for providing insights.

Regardless of all these positive side effects, researchers and datacenter managers keep looking for more energy efficient and environmental friendly datacenter management schemes, as energy costs are still a major component of datacenter costs. There are many studies that focus on improving the power usage effectiveness of datacenters. Power usage effectiveness is the ratio of total amount of energy used by a datacenter to the total amount of energy used for computation.

The difference power between datacenter and compute power usage is the facility overhead power spent to datacenter services such as cooling, power distribution, lighting, etc. The current average datacenter power usage effectiveness is 1.7. Companies such as Google and Facebook developed various novel datacenter designs empowered with ingenious cooling mechanisms and publicized these designs. State-of-the-art datacenters built according to these designs can achieve power usage effectiveness ratios as low as 1.1, meaning that these datacenters can operate with overheads as low as 10%.



circuitcellar.com/ccmaterials

REFERENCES

[1] E. Masanet, et al, "The Energy Efficiency Potential Of Cloud-Based Software: A U.S. Case Study," Lawrence Berkeley National Laboratory, 2013.

[2] R. McDougall, et al, "Towards an Elastic Elephant: Enabling Hadoop for the Cloud," *VMware Technical Journal*, Vol. 2, No. 2, 2013.

[3] Í. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "GreenHadoop: leveraging green energy in data-processing frameworks," in Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12), 2012.

BIG DATA ANALYTICS FRAMEWORK HADOOP

After investigating the sustainability and energy efficiency of the underlying infrastructures sustaining big data analytics, let's have a look at the energy efficiency of software frameworks for big data analytics. Hadoop is the most popular data processing framework that is commonly used in all stages of big data analytics as it can easily act as a data extraction, transformation, and loading framework, or as a data warehouse that harbors various data sources in one place and enables appropriate mappings and joins among these sources, or as a query

processing and analytics framework that draws the final value added analytics results. Hadoop implements the MapReduce parallel processing paradigm and incorporates a distributed file system component (Hadoop Distributed File System) to be able to scale to very large datasets.

The term Hadoop interchangeably is also used for referring to the applications in the “Hadoop ecosystem,” which include applications such as Apache Pig, Apache Hive, Apache HBase, and Apache Spark that can run on top of or alongside Hadoop. With the increase in big data analytics requests of many businesses and the advancements in cloud computing that enable easy access to large-scale processing and storage capacities on demand, utilization of applications in the Hadoop-ecosystem keep observing a steady increase as well.

Figure 2 showcases a classic big data analytics platform workflow. The platform is fed by structured and unstructured data sources. Data is gathered and consolidated into an analytical data store. Analytics tools use the real time and historical data coming from the data platform for providing insights.

As big data analytics and cloud computing keep on converging and merging, many cloud providers such as Amazon Web Services, Google Compute Engine, Rackspace, and Microsoft Azure started to offer various “big data as a service” solutions that run on VMs on the cloud, starting with offering Hadoop as a service solutions. These services automate provisioning of Hadoop clusters on the cloud to ease data analytics deployments and Hadoop clusters running on virtualized environments are starting to become the norm rather than the exception. Arguably, even though there are many companies specialized on Hadoop services such as Cloudera, Hortonworks, and MapR, Amazon is the company that makes most money from Hadoop with its Elastic Map Reduce solution.

Big data analytics solutions evolved separately from cloud computing and initial big data applications assumed that they would run on separate dedicated clusters that will only cater to them. As such, Hadoop assumes that its data storage and data processing services will be collocated in the same physical machine, and it tries to maximize data locality in its data placement and replication strategies to obtain the best I/O performance. However, in cloud-based Hadoop deployments, collocating data and compute services in the same set of servers is not preferred and these services are separated. This enables migrating live VMs and spinning up/down compute VMs as demand increases/decreases without causing costly


ABOUT THE AUTHOR

Ata Turk (ataturk@bu.edu) is a postdoctoral researcher in the Electrical and Computer Engineering Department at Boston University. He received MS and PhD degrees in Computer Science from the Bilkent University, Turkey. His research interests include information retrieval, data analytics, and combinatorial optimization for performance, energy, and cost improvements in cloud computing applications. He worked at Yahoo Labs in Barcelona, Spain, prior to his current position at Boston University.

data migrates.^[2] It also enables standard energy efficiency mechanisms to be applied even on Hadoop clusters as compute tasks now can be treated as independent stateless processes that can be easily migrated and packed into fewer number of physical servers if capacity and SLAs permit. Furthermore, as some of the workloads processed in Hadoop deployments are less time sensitive, Hadoop can enable participation in energy regulation programs or increase green energy usage.^[3]

Separation of data and compute services comes at the cost of a loss in data locality and performance degradation, but recent improvements in networking technologies can alleviate some of the adverse effects of remote data access and there are studies indicating that the importance of data locality under these newer networking technologies is negligible. Still, Hadoop requires storage in various stages of computation and organization of these storage sources can have an impact on the performance and energy utilization of a virtualized deployment.^[2] Furthermore, storage technologies keep on advancing as well, and with the widespread adoption of flash disks and ramdisks, data locality will start to matter again as the difference between the network I/O and storage I/O capacity will widen again. The performance and power implications of such novel deployments are still not well investigated and remain as an open research topic.

SUSTAINABILITY SUPPORTED

The largest public clouds supporting many big data analytics frameworks and datasets are running on energy efficient cloud datacenters that support investments into renewable energy sources. Furthermore, big data analytics is actively used by the energy industry for making intelligent decisions and improvements in efficiency. Finally, big data analytics frameworks such as Hadoop can moderate their computation to support green computing solutions and/or power provider energy regulations, although energy efficiency improvements for these frameworks especially under new hardware infrastructures is open for investigation. According to these observations it is safe to claim that current practices in big data analytics support sustainability. 

THE CONSUMMATE ENGINEER

Transformers 101 (Part 1)

Essential Characteristics

During the past year, George has presented introductions to a variety of essential electronics topics, including resistors and capacitors. In this series, he tackles the subject of transformers.

By George Novacek (Canada)

Before I delve into the subject of transformers, let's refresh what we know about inductors because they form the basis for all transformers. Defined by Faraday's Law, an undulating current i flowing through a coil whose inductance is constant will generate a back EMF voltage across the coil's terminals.

$$v = -L \left(\frac{di}{dt} \right)$$

This is a defining equation for inductors. The energy stored in the magnetic field of an ideal coil is:

$$P = \frac{1}{2} \times LI^2$$

Let me emphasize: an ideal coil has zero resistance of the winding and no parasitic capacitance between the turns. The fundamental characteristic of an ideal inductor is that it is a reactive element with frequency-dependent impedance Z .

$$Z = j\omega L = j2\pi fL$$

MAGNETISM

When voltage is applied across an inductor's terminals, a current flows through the coil. This generates magnetic field and, consequently, magnetic flux. These elements are responsible for generating the back electromagnetic force (EMF) which works to slow down the rate of rise of the current through the inductor. With a DC voltage step

(closing the switch in **Figure 1**) the current eventually saturates at V_{BAT}/R , where R represents the resistance of the winding plus any other resistance added in series, such as the wiring.

The relationship between the inductor's current I (in amperes, A), its inductance L (in henries, H) and the magnetic flux F is expressed as:

$$L = \frac{\Phi}{I} \times 10^{-8}$$

When two (or more) inductors operate close to each other, such as in transformers, their respective magnetic fluxes affect each other through mutual inductance M , which is sometimes expressed as a coefficient of coupling (k). For two coupled coils, mutual inductance M is:

$$M = \frac{\Phi_{2,1}}{I_1} \times 10^{-8} = \frac{\Phi_{1,2}}{I_2} \times 10^{-8}$$

$F_{2,1}$ is the magnetic flux in the second coil caused by the current in the first coil. $F_{1,2}$ is the magnetic flux in the first coil caused by the current in the second coil. And I_1 and I_2 are the respective currents in the first and the second coil. Mutual inductance and the degree of coupling k affect the transformer's characteristics. To achieve high efficiency in power and signal transformers, the coupling needs to be maximized. The maximum mutual inductance between two coils L_1 and L_2 occurs when magnetic flux of one coil crosses all the turns of the second coil and vice versa.



The maximum achievable mutual inductance value is:

$$M_{MAX} = \sqrt{L_1 \times L_2}$$

And the coefficient of coupling k equals:

$$k = \frac{M}{M_{MAX}}$$

k is the ratio of an actual (measured) mutual inductance M divided by calculated M_{MAX} . It has no dimension; its maximum value is 1. What applies for power and some signal transformers does not necessarily apply to high frequency resonating signal transformers in applications such as band-pass or band-reject filters, coupling transformers for tuned intermediate frequency (IF) filters and so forth. There, for example, the degree of coupling affects the frequency response and the slope of the filter. This is a separate topic, beyond the scope of this article.

A current flowing through a wire generates a circular magnetic field that's perpendicular to the wire. The total flux is the product of the field strength and loop area, perpendicular to the loop. In a solenoid (a coil) each wire loop contributes to the flux and, consequently, field lines inside the solenoid become nearly straight lines. All of the flux passes through the solenoid, perpendicular to its axis. The magnitude of the flux is proportional to the electrical current through the solenoid.

The magnetic field can be described by the right hand rule. Imagine you hold a wire in your right hand such, that the thumb points in the direction of the current flow. The magnetic flux curls like your fingers around the wire. The magnetic flux, also called induction field has intensity and direction in every point in space, forming continuous loops in space.

The magnetic field is characterized by its field strength H, which relates to the electrical current flow only. When a permeable core is inserted in the solenoid, the induction field quantity called B enters the picture. The relationship between B and H is based on the relative permeability of the magnetic core, where μ_0 is the permeability of air or vacuum, equaling to 1. With no ferromagnetic core $B = H$.

$$B = \mu_0 \times \mu_r \times H$$

This relationship is depicted in **Figure 2**. The flux B within an air-core solenoid versus its field strength H would be just a straight line, crossing where the axes cross. With a permeable core the flux B is not linear and exhibits hysteresis. When a solenoid is energized by AC current, the initial magnetization flux moves first from point A to

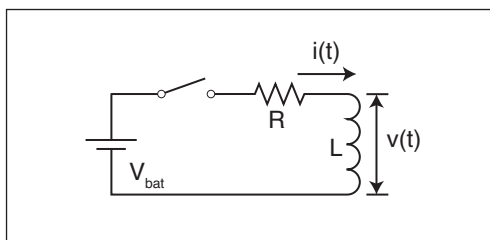


FIGURE 1
Back EMF works against the current supplied by the battery.

C. With the magnetic field H reducing and then inverting its polarity, the flux B moves along the left curve C to D and then again from D to C on the right. The characteristics of the curve depend on the permeable material of the core as well as the range of H.

Manufacturers provide B/H relationship for sinusoidal excitation. The nonlinear relationship of B/H varies with peak B values, material, frequency and waveform. Cores for operation above 20 kHz are often characterized with square wave excitation.

Transformers are often loaded with nonlinear loads, such as rectifiers with capacitor output filters. Consequently, the selection of a core material becomes very important. Power and audio transformers' cores are built as stacked laminations, torroids, and other designs to reduce losses, such as those due to eddy currents. Ferrite cores regularly use an air gap. For most applications it is crucial to avoid core's magnetic saturation and to operate within the linear B/H region.

Magnitude of the H field is reduced by the permeability factor μ_r . With the exception

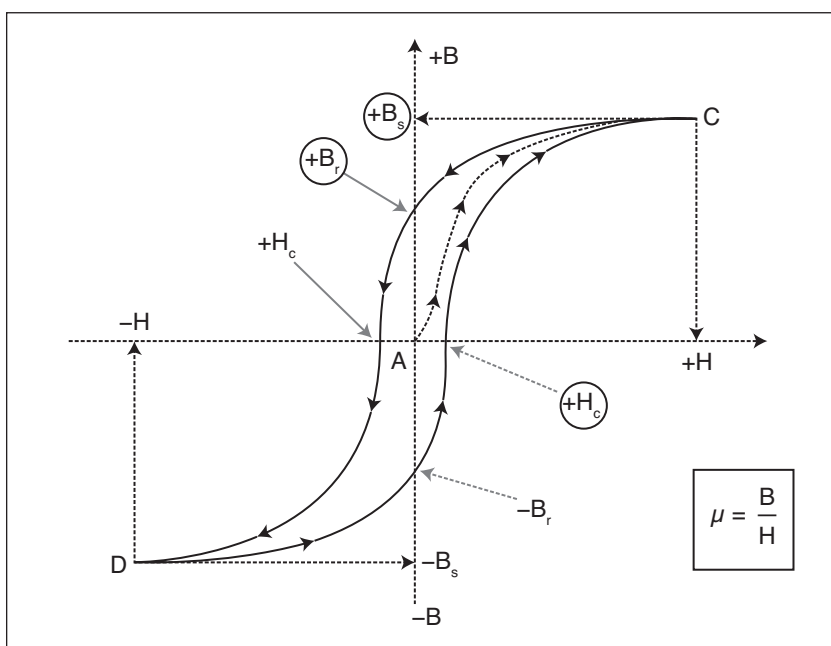
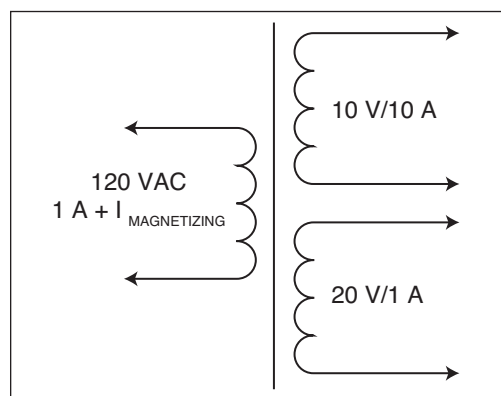


FIGURE 2
Magnetic field vs. magnetic flux and hysteresis

FIGURE 3

Transformer balancing by ampere-turns



of very high-frequency applications, most transformers need a ferromagnetic core to increase the permeability and, thus, decrease the necessary number of turns of the winding. The flux concentrates in the core. If there is a small air gap in the core, the field H will equal to B , while in the core it will be $H = B/\mu_r$. This results in a discontinuity of the H field lines, now reaching outside the core. This effect has been used in devices such as erase, record and playback heads in tape recorders, hard drives, etc.

POWER TRANSFORMER

Let's consider a typical 60-Hz power transformer. Transformers have a primary winding and one or more secondary windings



Circuit Cellar between 1999 and 2004. Contact him at gnovacek@nexicom.net with "Circuit Cellar" in the subject line.

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.

placed around a magnetic path. The primary winding takes energy from a power source, such as 120VAC and transforms it via secondary winding into a desired voltage.

The current flowing through n turns of wire in the primary winding, called magnetizing current, establishes the H field. Without a core this would have to be huge. Adding a ferromagnetic core such as iron reduces the current by the permeability of iron to a reasonable magnitude.

In *Grounding and Shielding Techniques* (Wiley-IEEE Press, 2007), Ralph Morrison provides example of a 60Hz transformer with a magnetizing current of 50mA. Permeability of the core $\mu_r = 20,000$. Should the core be removed, the required magnetizing current would jump to 1,000A.


The H field created by the primary winding generates magnetic flux B through the core. Secondary coils added around the path of the flux are magnetically coupled to it. According to Faraday's law, the secondary winding voltage will follow the B field changes as follows, where $F = BA$.

$$V = n \times A \times \frac{d\Phi}{dt}$$

A is the area of the core and n is the number of turns. Voltages on all windings, therefore, will be proportional to each other by the number of turns, ohmic losses due to the wire resistance notwithstanding. The windings are balanced by their ampere-turns. If the primary has 1,000 turns and the secondary 100 turns, for instance, the load current in the primary coil will be 10% of the secondary current. This is illustrated in **Figure 3**.

Notice that because of the need for the magnetizing current, a transformer cannot achieve full 100% efficiency. Transformers may have numerous windings and each winding may have many taps. Transformers can also provide isolation of many thousands volts between all their windings. They also serve to suppress common mode interference from communication signals, such as Ethernet, MIL-STD-1553B and others. One exception is an autotransformer, where only one winding with taps exists. While it is less costly to build, it provides no isolation or common mode interference reduction.

TRANSFORMER DESIGN

In the second part of this article series, I'll detail some aspects of the transformer design. There are many empirical equations for calculating the transformer characteristics. Relying on them in the course of your design will achieve something within a wide ballpark of the measured values at best. 



circuitcellar.com/ccmaterials

Have you stopped by the shop lately?

Circuit Cellar is always adding new items to help with your design projects. Stop by often for the latest deals.

books
audio
back issues

contest cds

archive cds



www.cc-webshop.com

ABOVE THE GROUND PLANE

Random LED Dots

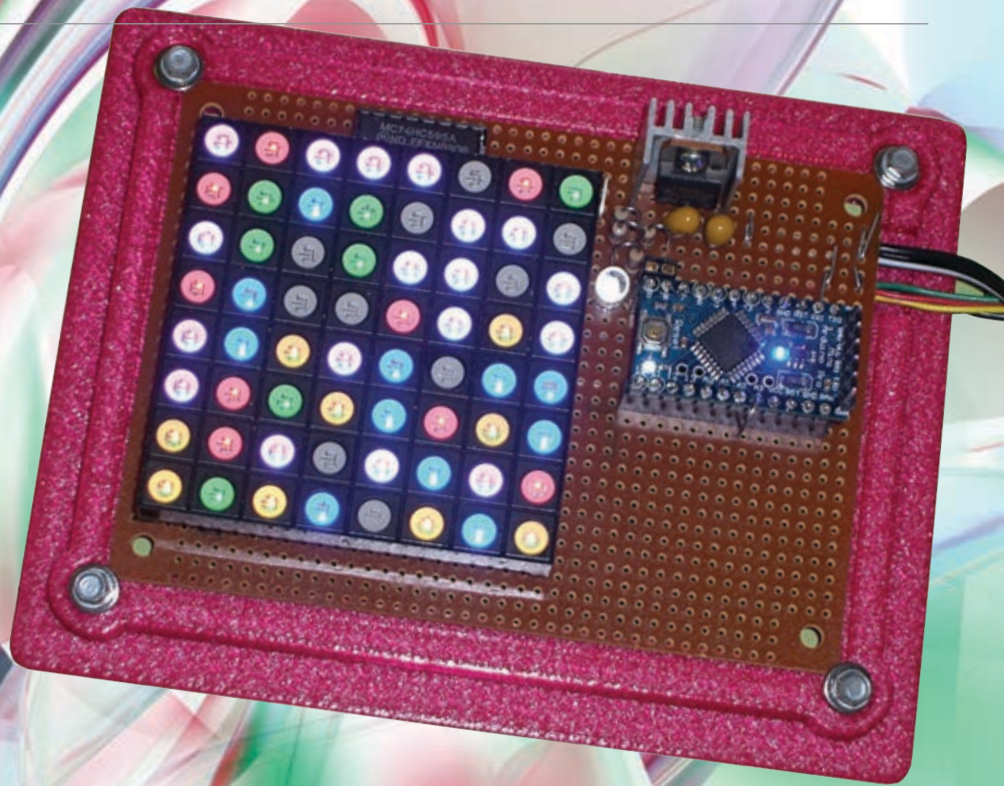


PHOTO 1

A simple handwired protoboard works well enough with a 1 MHz shift clock. Connections to the 9 VDC wall wart and Geiger monitor emerge from the 3D printed board holder at the upper right. The LED matrix covers most of the circuitry.

Ed was recently challenged to build something with an unclaimed 8 × 8 RGB LED matrix. Never one to pass on an engineering challenge, he built an interesting Arduino-based desktop radiation monitor.

By Ed Nisley (US)

A while ago, Sophi dumped a pile of tech gadgetry on the work table at the local makerspace and asked all present to make it vanish. When an 8 × 8 RGB LED matrix remained unclaimed after the initial scrum, she flicked it across the table and asked me if I wanted to make something of it. Combined with a few bits from my heap and an idea from one of her spectacular Burning Man installations, the prototype of a small desk toy shown in **Photo 1** took shape.

The white LED to the upper left of the Arduino Pro Mini flashes whenever an external Geiger-tube radiation sensor detects a beta or gamma particle, whereupon one of the RGB LEDs changes color. The background radiation on my desk ticks along at a dozen counts per minute, producing a slowly changing display.

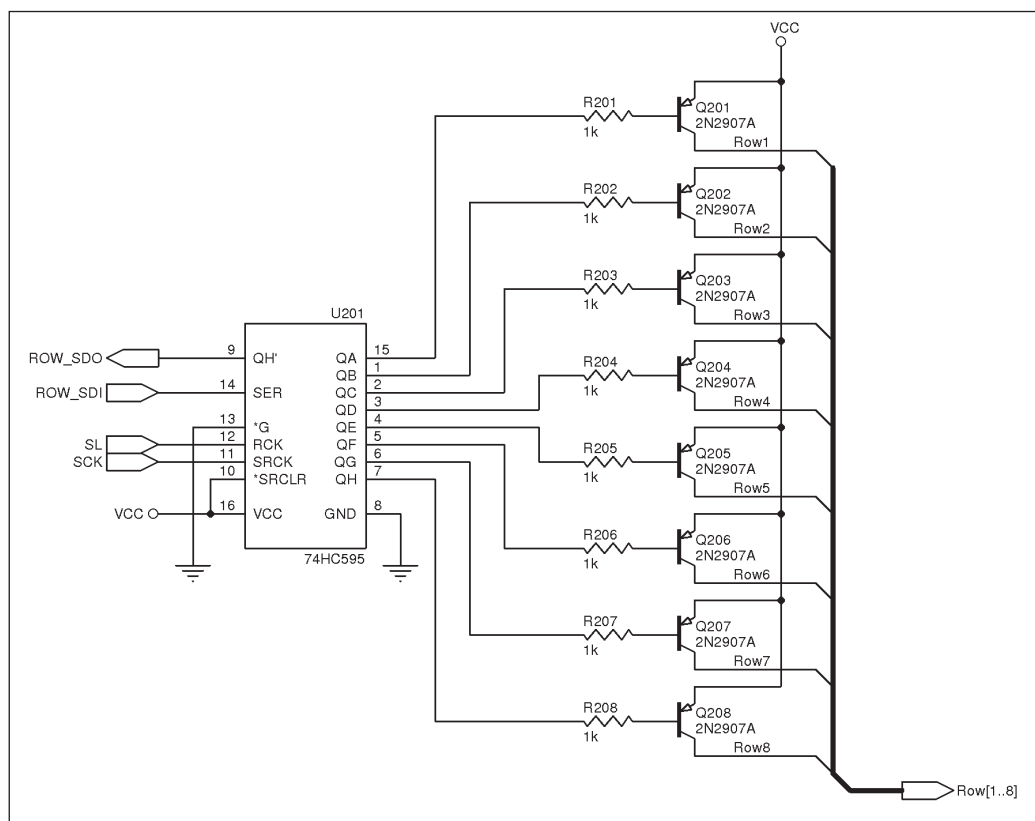
In this column, I'll describe how to multiplex an LED matrix with simple shift registers and transistors, as well as explain why specialized hardware makes more ambitious projects feasible. You'll also see how to generate reasonably random numbers from randomly timed pulses. An Arduino can certainly handle more complex display algorithms, but that's a simple matter of software.

MULTIPLEXING LEDs

The 8 × 8 RGB LED matrix contains 192 diodes in a package with only 32 pins, so it obviously requires external multiplexing hardware. The one I used has common anode matrix connections: each of the eight rows has one pin connected to all 24 LED anodes along that row. Each of the eight columns has three pins, one apiece for the cathodes of the eight red, green, and blue LEDs in that column. This type of display depends on the ability of human eyes to see rapidly blinking lights as steady sources, because only one row of LEDs will be illuminated at any time. You see the entire panel glowing at once, as shown in Photos 1, 2, and 4, as do cameras with relatively long exposure times.

The persistence of vision fails when your eyes move, because each blink activates different cells in your retina. The effect becomes more striking in dim illumination: you've probably noticed how LED automobile taillights form dotted contrails as your gaze shifts between traffic, the dashboard, roadside signs, and back to traffic. I find LED taillights distracting, although some people can't distinguish them from incandescent bulbs.

Because each LED emits light only 1/8 of

**FIGURE 1**

Properly multiplexing the LED matrix requires activating each row in sequence, so the firmware ensures the 74HC595 shift register contains only a single 0 bit. Each PNP transistor must supply enough current for all 24 LEDs in its row.

the time, the matrix will be 1/8 as bright as it would be with constantly lit LEDs. The usual remedy requires driving the LEDs at their maximum pulsed-current rating, with results that I explored in my July and September 2013 columns on LED characterization. Large arrays intended for outdoor use require high-power drive circuitry and elaborate thermal control, but a desk toy can produce enough light with very low currents and low-cost generic hardware.

The diodes in the LED matrix have no memory, which means the driver must maintain a bitmap of the entire display state and periodically refresh the matrix. An eight-color RGB display requires one bit per LED, so the 8x8 matrix I'm using requires 24 bytes of RAM. The amount of data grows as the square of the matrix size: a 16x16 matrix fills 96 bytes and a 32x32 display soaks up 384 bytes. The Atmel ATmega 328 microcontroller on the Arduino Pro Mini has 2 KB of RAM, of which about 1.5 KB will be available for simple programs, putting an obvious upper limit on the display size.

If the display hardware supports per-LED brightness control, then the memory use grows accordingly. Sixteen brightness levels requires four bits for every LED, which means the bitmap for a 32x32 matrix will blot up all of the Arduino's RAM. Obviously, Arduino-class microcontrollers are best suited for small displays with relatively few colors. The

simple hardware I used allows on-off control of each LED, for a total of eight colors.

Another limit comes from the output bandwidth required to transfer the data fast enough to reduce visible flicker. I set the ATmega328's hardware SPI to shift data at 1 μ s/bit, which is a reasonable compromise

```
typedef struct {
    const byte Row;
    byte ColR;
    byte ColG;
    byte ColB;
} LED_BYTES;

#define NUMROWS 8
#define NUMCOLS 8

LED_BYTES LEDs[NUMROWS] = {
    {0x80,0.0,0.0},
    {0x40,0.0,0.0},
    {0x20,0.0,0.0},
    {0x10,0.0,0.0},
    {0x08,0.0,0.0},
    {0x04,0.0,0.0},
    {0x02,0.0,0.0},
    {0x01,0.0,0.0},
};

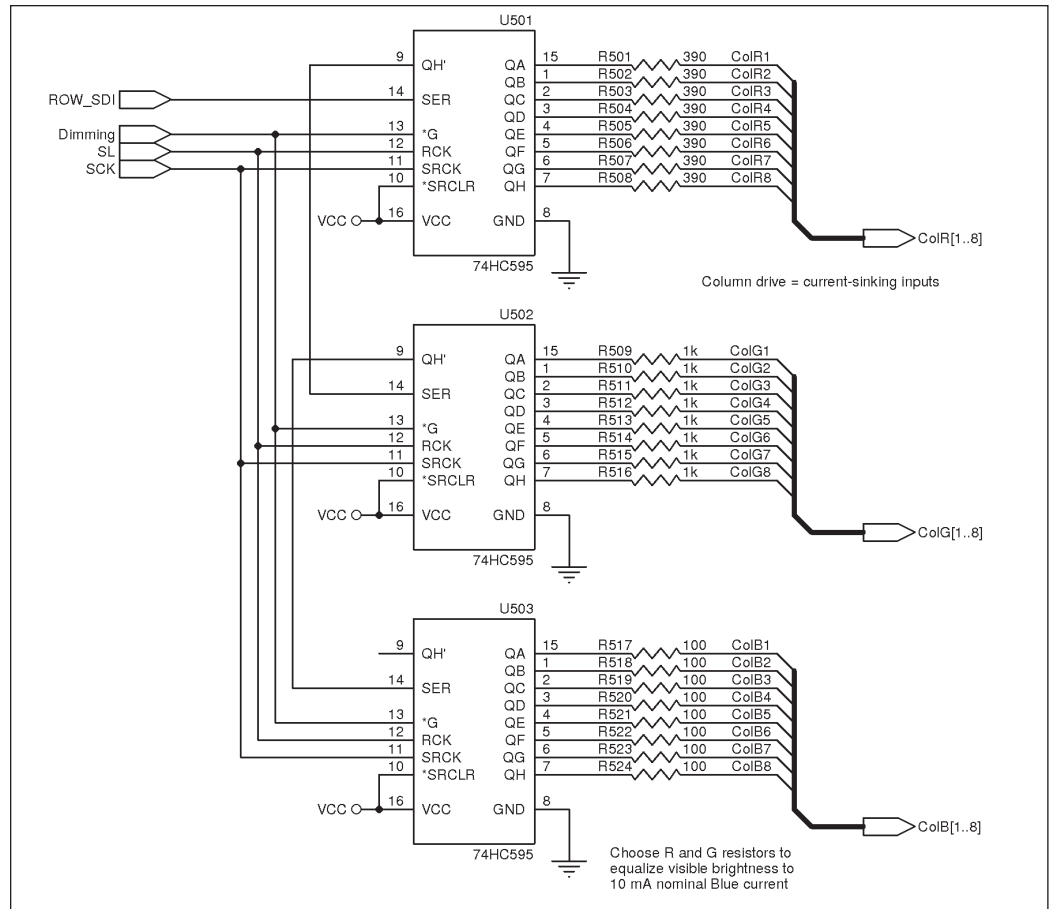
byteRowIndex;
```

LISTING 1

Each array element contains the bit patterns for one row of the LED matrix: illuminating the entire matrix requires sending all eight elements in rapid succession. The function handling the SPI output inverts the bit patterns to match the active-low circuitry driving the matrix.

FIGURE 2

The 74HC595 shift registers act as current sinks for the LEDs in each row, with the resistors limiting the current. Because the blue LEDs have the lowest luminous efficiency, they require the highest current. The other resistors set the red and green LED current for equivalent brightness, so that turning on all three LEDs produces white light.



between speed and circuit design. Although the SPI clock can run up to 8 MHz, the per-byte software overhead doesn't decrease

**PHOTO 2**

A close look shows the individual RGB LED chips in each site wash the white walls with their color. As a result, you can't get perfect white light no matter how carefully you balance the LED currents.

and the overall refresh rate won't improve as much as you might expect. In addition, the simple circuit layout and power distribution I used won't support such high bit rates.

With a 1 MHz data rate and nothing else on its mind, the Arduino can refresh the entire matrix every 800 μs. At that pace, even 1/8 duty cycle LEDs don't flicker at all.

Homework: figure the duty cycle and data rate for a 128x128 RGB matrix with 256 brightness levels for each LED, refreshed at 100 Hz. Compare that with the LCD panel in your phone to understand the limitations of small microcontrollers.

BINARY IN AN ANALOG WORLD

The LED matrix has 32 connections: eight current sources for the anodes in each row and 24 current sinks for the RGB LED cathodes along each column. Because ATmega 328 microcontrollers don't have that many I/O pins, I used four 74HC595 parallel-output shift registers to provide direct control over each signal.

The row drivers must source enough current to light up all 24 LEDs in each row at the same time. Although I used relatively low LED currents, as I'll describe later, the total current for one row can exceed 100 mA, far

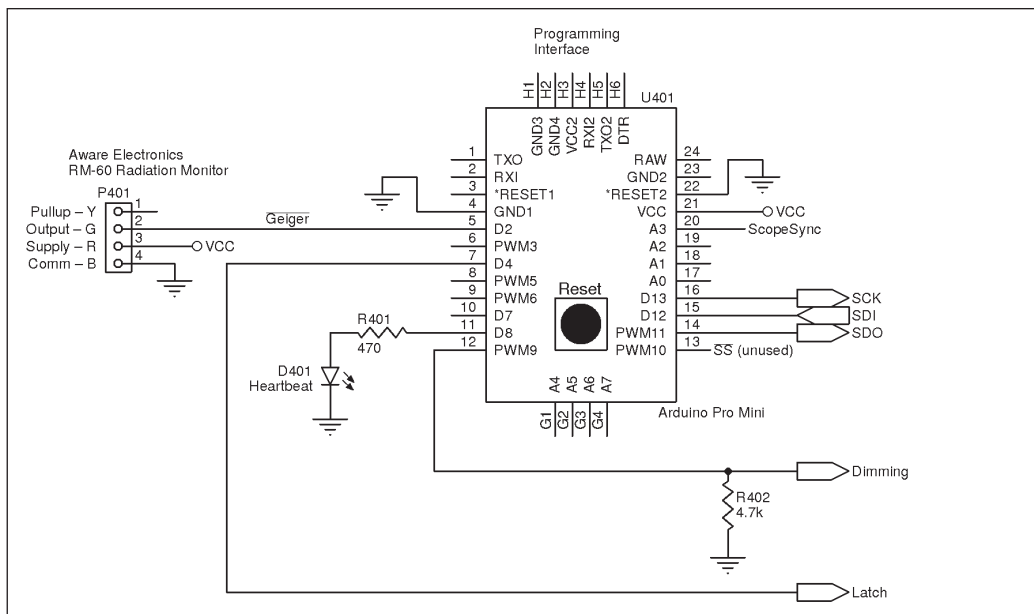


FIGURE 3

An Arduino Pro Mini refreshes the LED matrix using hardware-assisted SPI. Pulses from a Geiger-tube radiation sensor generate random numbers to update the LED colors.

exceeding the microcontroller’s 40 mA per-pin absolute maximum current rating. As a rule of thumb for most digital logic chips, you shouldn’t exceed 20 mA per pin without carefully considering the part’s data sheet: obviously, some buffering was in order.

I used ordinary 2N2907 bipolar

transistors as current amplifiers for the LED anodes, as shown in **Figure 1**, with the shift register outputs acting as analog current sinks. Each shift register output will pull 4 mA through the base of its 2N2907A PNP transistor when it goes low, due to the 1 kΩ resistor. The 2N2907A data sheet specifies

COLUMNS

A Guide to Powerful Programming for Embedded Systems

Assembly Language Essentials

Larry Cicchinelli

Shop for this book, and others, at www.cc-webshop.com

CIRCUIT CELLAR

The Lowest Prices on the Best Scopes

- Passport-Size PC Scopes** \$129+
Great scopes for field use with laptops. Up to 200MHz bandwidth with 1GSa/s, high speed data streaming to 1MSa/s, built-in 1GSa/s AWG/function gen. PS2200A series
- 30MHz Scope** \$289
Remarkable 30MHz, 2-ch, 250MS/s sample rate scope. 8-in color TFT-LCD and AutoScale function. Includes FREE carry case and 3 year warranty! SDS5032E
- 50MHz Scope** \$399
50MHz, 4-ch scope at 2-ch price! Up to 1GSa/s rate and huge 12Mpts memory! Innovative "UltraVision" technology for real time wfm recording. FREE carry case! DS1054Z
- 100MHz Scope** \$399
Best selling 100MHz, 2-ch scope with 1GSa/s rate plus huge 10MSa memory! 8-in color TFT-LCD. Includes FREE carry case and 3 year warranty! SDS7102V
- 200MHz - 1GHz Scopes** \$1,500+
2/4 channel, 8 or 12 bit with long memory, powerful debug capabilities (Teledyne LeCroy) WaveAce 2000 Series

- Free Technical Support
- Excellent Customer Service

```

void WaitSPIF(void) {
  while (! (SPSR & (1 << SPIF))) {
    continue;
  }
}

byte SendRecSPI(byte Dbyte) { // send one byte, get another in exchange
  SPDR = Dbyte;
  WaitSPIF();
  return SPDR; // SPIF will be cleared
}

void UpdateLEDs(byte i) {

  SendRecSPI(~LEDs[i].ColB); // low-active outputs
  SendRecSPI(~LEDs[i].ColG);
  SendRecSPI(~LEDs[i].ColR);
  SendRecSPI(~LEDs[i].Row);

  analogWrite(PIN_DIMMING,LEDS_OFF); // turn off LED to quench current
  PulsePin(PIN_LATCH); // make new shift reg contents visible
  analogWrite(PIN_DIMMING,LEDS_ON);

}

```

LISTING 2

These three functions handle the SPI hardware interface. Sending all four bytes with a 1 MHz clock requires 35 μ s.

a minimum DC current gain of 100, so the transistor will remain saturated for collector currents under 400 mA. At $I_C = 100$ mA, the transistors act as switches with V_{CE} around half a volt.

For proper multiplexing, the Arduino

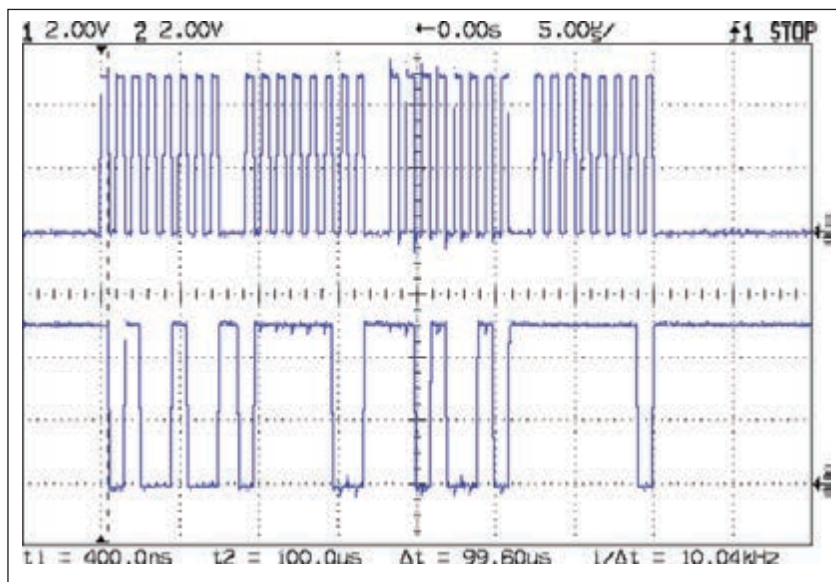
program must ensure that only one shift register output bit will be active at any time. The data structure and initialized array definition in **Listing 1** set up the proper bit patterns, with the `const` byte attribute telling the compiler to prevent inadvertent changes to the row selection bits during execution. I use active-high bit patterns in the array, because they're easier to work with, and invert the bits before sending them to the SPI hardware.

Pop Quiz: Is the array initialization sufficient to prevent multiple row activation?

With one output low and its PNP transistor turned on, the column drivers must limit the current through the 24 LEDs in that row. The three groups of resistors shown in **Figure 2** allow simple on-off control with a fixed brightness for all eight LEDs of each color, with the shift register outputs acting as analog current sinks.

Each LED color requires a different ballast resistor, chosen for the LED's efficiency at converting current to light and the human eye's response to that color. Because blue LEDs have the lowest efficiency, red LEDs the highest, eyes respond best to green, and I wanted more-or-less white light when all three LEDs turned on, choosing the resistors required some trial-and-error.

Data sheets for 74HC595 shift registers

**PHOTO 3**

Each rising edge of the 1 MHz SPI clock (upper trace) clocks data into the 74HC595 shift registers. The last eight data bits (lower trace) control the row drivers, so only one bit will be low. The code in Listing 2 produces the gaps between the bytes as it loads the next set byte into the SPI hardware.


```

if (GeigerTicked) {
    digitalWrite(PIN_HEARTBEAT,HIGH);    // show a blip
    analogWrite(PIN_DIMMING,LEDS_OFF);  // turn off LED array to prevent bright glitch

    Hash = jenkins_one_at_a_time_hash((char *)&GeigerTime,4); // whiten the noise

    GeigerTicked = false;                // flag interrupt handler to resume recording
    SetLED(Hash);
}

UpdateLEDs(RowIndex++);
if (RowIndex >= NUMROWS) {
    RowIndex = 0;
    PulsePin(PIN_SYNC);
}

digitalWrite(PIN_HEARTBEAT,LOW);        // always turn off the blip

```

LISTING 3

This code executes on each pass through the main loop. When GeigerTicked is true, the conditional converts the timestamp of the new Geiger event into four bytes of white noise that sets the color of a single LED. The loop iterates at 10 kHz and refreshes the entire LED matrix at 1.3 kHz, which is fast enough to require turning the LEDs off so that lengthy computations don't hold a single row on long enough to produce a visible flash.

specify 35 mA maximum for any output pin and 70 mA maximum through the IC's ground pin. The ground pin current limit poses the tightest constraint when using the outputs as current sinks: all of the current from eight LEDs must pass through that single pin.

Because blue LEDs have the lowest visual response, I set them to a nominal 10 mA and picked the red and green LED currents to produce equivalent brightness, with the result shown in **Photo 2**. Obviously, the light from the RGB site in the lower left corner isn't pure white, but it's close enough for my purposes.

Note that the 74HC595 chip must sink 80 mA when all the blue LEDs turn on, somewhat in excess of its maximum rating. In comparison, the green LEDs require 7 mA and the red LEDs only 2 mA, so those 74HC595 chips operate well within their limits, even with all eight LEDs turned on.

Because I'm setting the LEDs with random data, however, it's very rare for all the blue LEDs in a single row to be on at the same time and, in fact, you'd expect only half of them to be on at once. A slight change to the program could slant the odds by not turning on the eighth LED in any row.

Pop Quiz: How many blue LEDs are on in Photo 1? Which row has the maximum?

The TLC5916 LED driver I used for the Totally Featureless WWVB Clock might seem to be a better choice than these shift registers, as its drivers allow a *maximum*

120 mA per LED. However, it also has a 5 mA *minimum* regulated output, twice what the red LEDs draw, so using it would require a neutral-density filter atop the LED matrix to produce a similar intensity.

Homework: Adapt the design to use the specialized LED driver IC of your choice.

The closeup view in Photo 2 shows the three individual LED chips in each site, lined

```

void SetLED(unsigned long Value) {
    byte Row = (Value >> 8) & 0x07;
    byte Col = (Value >> 16) & 0x07;
    byte Color = (Value >> 24) & 0x07;

    byte BitMask = (0x80 >> Col);

    LEDs[Row].ColR &= ~BitMask;
    LEDs[Row].ColR |= (Color & 0x04) ? BitMask : 0;

    LEDs[Row].ColG &= ~BitMask;
    LEDs[Row].ColG |= (Color & 0x02) ? BitMask : 0;

    LEDs[Row].ColB &= ~BitMask;
    LEDs[Row].ColB |= (Color & 0x01) ? BitMask : 0;
}

```

LISTING 4

Setting the color of a single LED requires splitting the value into three separate bits, then inserting each bit into the proper location in the column elements of that row.

PHOTO 4

An underexposed video frame captures the full LED array. The Arduino Pro Mini's white LED glows brightly, because that pin is also the SPI hardware shift clock.



up at the base of the conical white plastic reflector that increases the visual effect. Viewing the matrix at close range, as you would for a desk toy, reveals the red and blue LEDs wash the sides of the “white” reflector with color, an effect that even carefully adjusted LED currents can't eliminate. Fortunately, I'm not trying to achieve color-corrected photorealistic results.

The Arduino Pro Mini in **Figure 3** controls the LED matrix using only five outputs: unlike most Arduino projects, it has plenty of

spare I/O pins. The SPI hardware interface supports both output (SDO) and input (SDI) bits, although SDI isn't used here. The -SS pin must remain high when the Arduino is in SPI Master mode, so the code sets it to be an output with no connection. The Latch signal pulses high after shifting all four bytes into the shift registers.

The Dimming output connects to the 74HC595 column driver -G inputs. I use it as a binary control: high to disable and low to enable the driver output pins. You could also use it as a PWM output, with increasing PWM values reducing the overall LED matrix brightness.

With that in mind, the UpdateLEDs function in **Listing 2** sends a single row of data from the LED bitmap array to the shift registers. The code inverts each byte before sending it, because the bitmap array represents selected rows and LEDs with active-high (1) bits, while the drivers require active-low (0) bits.

Photo 3 shows the SPI clock and data signals that transfer a single row to the shift registers. The 1 MHz SPI clock in the upper trace transfers each byte in 8 μ s and the complete transfer occupies 35 μ s, with the gaps between the bytes corresponding to the code in Listing 2. The rightmost byte, sent by the last SendRecSPI call in Listing 2, goes to the row driver in Figure 1 and has an obvious only-one-low-bit pattern.

The program's main loop calls UpdateLEDs during each iteration to send successive rows. Sending the entire contents of the bitmap array to refresh all the LEDs requires eight passes.

The next step: put some interesting data



circuitcellar.com/ccmaterials

RESOURCES

3D Printed Protoboard Holder, <http://softsolder.com/2015/05/14/proto-board-holder-80x110-mm-version/>

Hardware Random Number Generation, http://en.wikipedia.org/wiki/Hardware_random_number_generator

Jenkins One-At-A-Time Hash Function, http://en.wikipedia.org/wiki/Jenkins_hash_function

More on Random Noise, <http://softsolder.com/2015/05/21/random-led-dots-radioactive-noise/>

Sophi's Art Projects, <http://mix-engineering.com/>

SOURCES

Adafruit RGB LED matrix display
Adafruit Industries | www.adafruit.com/categories/327

8 x 8 Common-Anode RGB LED Matrix
Seed Development | www.seeedstudio.com/depot/60mm-square-88-led-matrix-super-bright-rgb-p-113.html?cPath=163_165

Arduino Pro Mini microcontroller board
SparkFun (distributor) | www.sparkfun.com/products/11113

TLC5916 LED Driver
Texas Instruments | <http://focus.ti.com/docs/prod/folders/print/tlc5916.html>

into the bitmap array.

TIMESTAMPED RANDOMNESS

A microcontroller program with no inputs is, essentially by definition, a deterministic process: it will produce exactly the same outputs after each reset. The Arduino library's `random` function actually returns a fixed sequence of pseudorandom numbers, unless you set the initial seed with a truly random value from an input.

The `-Geiger` input in Figure 3 injects true randomness into this program, because it comes from an Aware Electronics RM-60 Geiger sensor that detects beta and gamma radiation. The background radiation on my desk produces about 12 pulses each minute and, for more activity, a watch with a radium dial positioned against the Geiger tube produces 160 pulse/min.

Generating cryptographic-quality random numbers from a radiation detector, as described in the references, requires far more effort than seemed warranted for this gadget. Although radiation pulses occur at truly random intervals, the absolute times when those pulses occur form an ascending sequence that's obviously not random. Finding the interval between two events removes the sequence, but those intervals have a relatively small average range that depends on the radiation intensity.

John von Neumann described how to eliminate the bias, at the cost of producing less than one random bit for every four pulses. That algorithm would require about three minutes of background radiation to generate the nine random bits that update a single RGB LED, far longer than seemed reasonable.

Because all I needed was a series of reasonably random numbers (and I was unwilling to increase the background radiation level around my desk), I used an interrupt handler to record the absolute time in microseconds when each pulse occurs, then passed those ascending values into a hash algorithm that produces four bytes of randomized bits with no obvious sequence.

A hash function summarizes a (possibly lengthy) stream of input numbers or characters by producing a fixed-length output number. Because hash functions are deterministic computer programs, hashing the same stream always produces the same output number, a characteristic of the digital signature that verifies a larger message. In this case, the hash function must map 32 bits of input data into 32 bits of output data.

I used the Jenkins One-At-A-Time hash, a simple algorithm that thoroughly mixes its input bits and runs quickly, so that feeding

it with absolute time values produces a corresponding series of mixed bits that resemble white noise. Cryptographically secure hash functions, such as those in the SHA family, require elaborate computations that aren't well-suited for Arduino-class microcontrollers, so, even though the Jenkins hash may not be crypto grade, it's good enough for my purposes.

The interrupt handler sets the `GeigerTicked` boolean variable code to indicate that it has recorded the time of a new pulse in `GeigerTime`. When `GeigerTicked` becomes true, the code in **Listing 3** passes `GeigerTime`, the pulse's four-byte timestamp, through the Jenkins hash function to produce `Hash`. It then calls the `SetLED` function in **Listing 4** to extract a trio of three-bit values from the hash that select an LED by row and column, then set the color of that LED in the bitmap array.

A four-byte integer can count 71.6 minutes of microseconds before it wraps to zero. At 12 events per minute, each of those 860 events will (almost certainly) occur at a different one of the 4.3×10^9 microseconds, so the hash function will deliver unique values for each event.

I haven't bothered to run the tests showing exactly how random these random numbers might be, because it doesn't matter.

Photo 4, a somewhat gritty video image capture, shows another one of the 6.3×10^{57} possible color arrangements: so far, they've all been pleasing to the eye. The pulses arrive slowly enough that the display never seems to change when you're watching it, yet it's completely different after ten minutes.

After I got everything running, however, I noticed that sometimes the white LED would flash to indicate a pulse from the Geiger sensor, but the LED matrix wouldn't change. It turns out that the randomly chosen new color for an LED may be the same as the old color: even though the code updates the bitmap with every pulse, 12.5% of those pulses produce no visible difference.

CONTACT RELEASE

The magenta 3D printed board holder sufficed to get the circuit and firmware running, but doesn't provide enough protection for a desk toy. I originally planned to conceal all the hardware inside a black box with a smoke-gray window over the matrix, but perhaps showing off the tech with a transparent box would be more appropriate. It's time for some rapid prototyping!

You can download the Arduino program and the complete schematic in KiCad format from the *Circuit Cellar* FTP site. [E](#)



ABOUT THE AUTHOR

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

FROM THE BENCH

Wiegand World

An Introduction to the Physical Layer & Protocol



Wiegand technology has been around since the 1970s. Jeff Bachiochi covers the history of the technology, and covers the Wiegand interface and protocol. He concludes with details about a microcontroller-based Wiegand data display project.

By Jeff Bachiochi (US)

PHOTO 1

This typical harsh environment keypad outputs data using the standard Wiegand formatted output. A minimum connection includes +12 V, Common, and Wiegand 0 and 1 outputs. The outputs are open collector and multiple units can coexist on the same connections. While there is no protection against collision, this will most likely cause the transmitted data to fail proper formatting.

While working to improve the ignition system of the automobile during the early 1970s, John Richard Wiegand discovered a physical phenomenon we now know as the Wiegand effect. The ability for a material to retain a bistable magnetic state and change state when exposed to an opposing magnetic field. The material is produced in the form of wire that has been first annealed to produce a soft core and then cold-worked producing a hard shell. This gives it a unique nonlinear magnetic property.

The outer shell has high coercivity, or the ability to withstand an external magnetic field without becoming demagnetized, while the inner core has a low coercivity. So the shell resists any opposing magnetic field until its magnetic hysteresis level is reached; then the material rapidly changes its magnetic polarity.

This flip/flop occurs in just a few microseconds.

When used as the heart of a sensor consisting of a short length of the material surrounded by a coil, the rapid change in the magnetic field induces a current spike in the external coil. The Wiegand sensor will produce both positive and negative spikes depending on the polarity of the magnetic change. The high repeatability threshold of the magnetic field makes the Wiegand effect useful for positional sensors and the like.

KEYCARDS

Keycards have been around for a long time. In a September 1954 *Popular Mechanics* article, "A House of Magic," Thomas E. Stimson details the use of key cards to open a gate at an automated parking lot. Prior to the new craze of smartcards and RFID technology, popular

types of keycards included the mechanical punched hole, barcode, and magnetic stripe cards. Wiegand wire embedded cards became popular because they were difficult to duplicate. Since the key code is permanently set into the card at manufacture by the positions of the wires, Wiegand cards can't be erased by magnetic fields or reprogrammed as magnetic stripe cards can. So, in high-security areas, Wiegand cards were king.

The Wiegand plastic keycard has a series of short lengths of Wiegand wire embedded in it, which encodes the key by the presence or absence of wires. A second track of wires provides a clock track. The card is read by pulling it through a slot in a reader device, which has a fixed magnetic field and a sensor coil. As each length of wire passes through the magnetic field, its magnetic state flips, which indicates a 1, and this is sensed by the coil. The absence of a wire indicates a 0.

With the popularity of Wiegand on an upturn, secure points of entry needed to be tied to a central computer as building systems grew in complexity over local control right at each door. A new protocol was developed for transmitting data over extended distances from the point of entry to the central control system. By the 1980s, the Wiegand interface became a de facto wiring standard.

WIEGAND PHYSICAL LAYER

The Wiegand interface uses three wires: a common ground and two data transmission wires usually called DATA0 and DATA1. When idle, both DATA0 and DATA1 are pulled up to the "high" voltage level, usually +5 VDC. When a 0 is sent, the DATA0 wire is pulled to a low voltage (DATA1 stays high). When a 1 is sent, the DATA1 wire is pulled to a low voltage (DATA0 stays high). An advantage of the Wiegand signaling format is that it allows very long cable runs, far longer than other interface standards of its day allowed.

Wiegand card readers pretty much defined the Wiegand communication protocol, as it followed the physical manufacturing format of

the embedded card data. Using a card as the only access vehicle was secure, but a physical keypad had wide spread appeal. Building off Wiegand card success, the already established communication format stuck for other types of entry devices.

Photo 1 pictures a typical harsh-environment keypad. These are typically in the standard 3 × 4 key format consisting of 10 digits (0 to 9), plus asterisk (*) and pound (#) keys. Most of the industry operates at a supply voltage of 12 VDC. This allows for a bit of drop in the supply lines, affording sufficient overhead to an onboard 5-V regulator. While many devices aren't considered "smart," they all need some amount of smarts to interface the user entry device (keypad) with a communication protocol. Key entering consists of a number of digits followed by the "#" used as an enter/finished key. The "*" key is often a cancel, flushing out any digits entered.

WIEGAND PROTOCOL

Based on the Wiegand swipe card technology, the data format is presented in a 26-bit format, one even parity bit, 8 bits of facility code, 16 bits of ID code, and a trailing odd parity bit. The first parity bit is calculated from the first 12 bits of the code and the trailing parity bit from the last 12 bits. Most access control system manufacturers adopted Wiegand technology, but its limitations of only 8 bits for facility codes (0–255) and 16 bits for card ID (0–65535) caused some to design their own formats with varying complexity of field numbers and lengths and parity checking. The physical size limitations of the card dictated that a maximum of 37 Wiegand wire filaments could be placed in a standard card, as dictated by CR80 or ISO/IEC 7810 standards. Therefore, most Wiegand formats used in physical access control are less than 37 bits in length. While this doesn't change the physical communications layer, propriety data formats don't lend themselves to universal use.

Let's take a look at the typical Wiegand output for a Wiegand device. As you can see

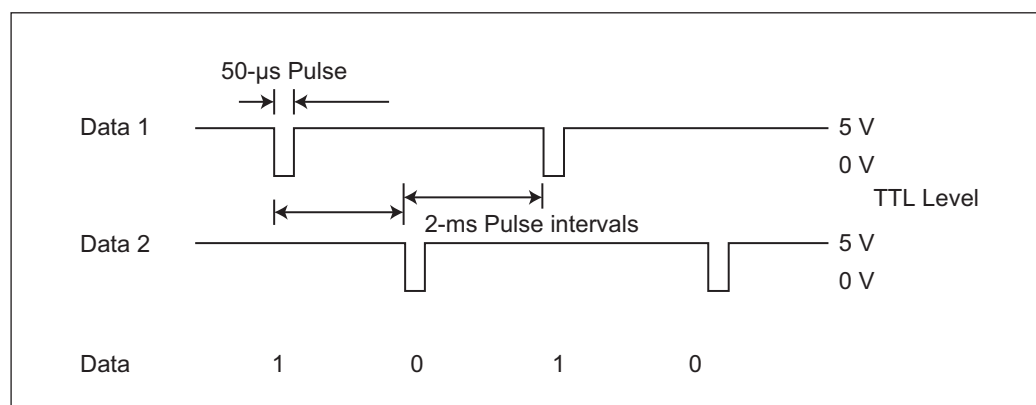
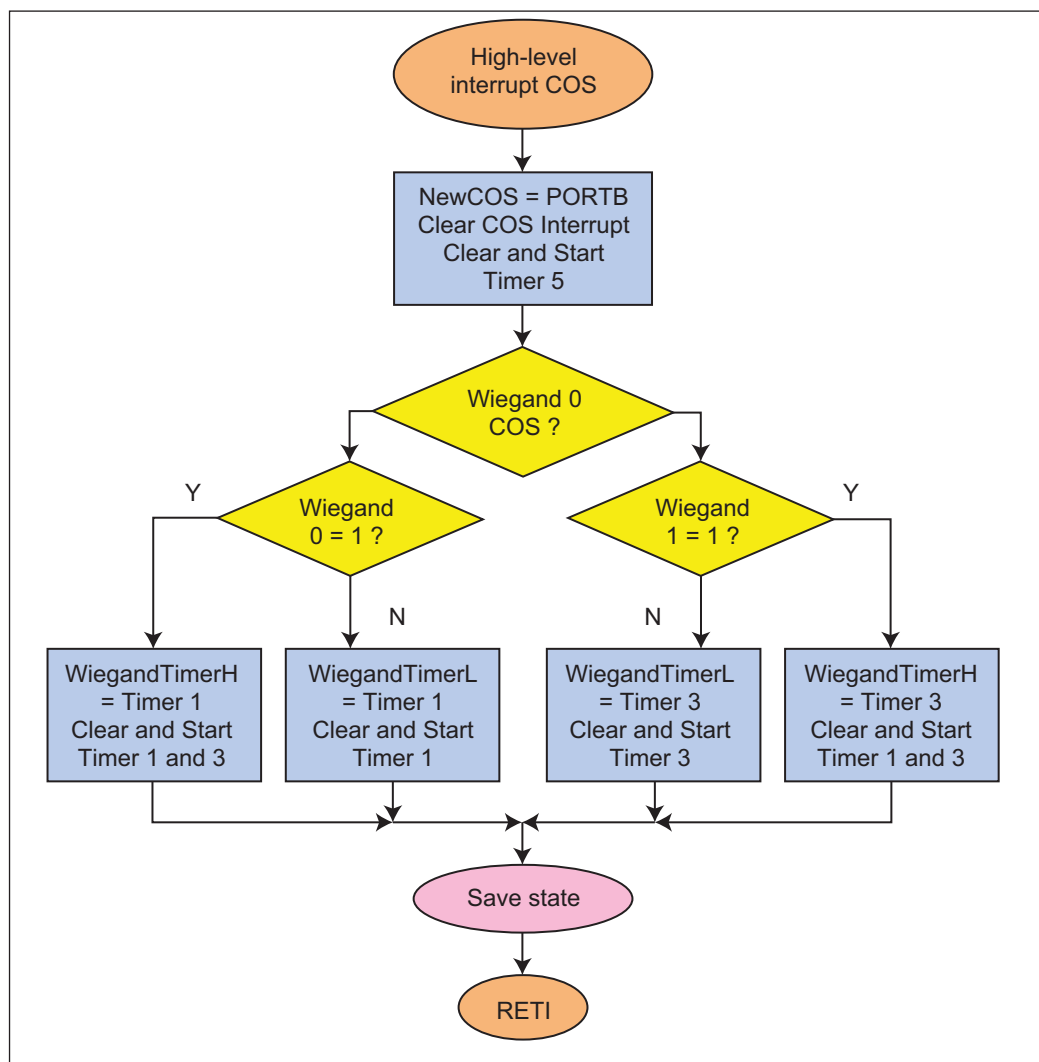


FIGURE 1

The standard pulse width is typically 50 μs for either Wiegand 0 or Wiegand 1 data with an inter-pulse or delay time of 2 ms.

FIGURE 2

The highest level interrupt collects any changes of state (COS) that are seen on either of the Wiegand data lines. Related Pulse or Delay times since the last COS are recorded as well so that later these can be verified against the actual Wiegand specs.



in **Figure 1**, the Data1 pulses on the top trace and Data0 pulses on the bottom trace do not coincide—that is, they occur independently. Open-collector drivers for each line are pulled up to VCC with resistors. The pulse width is typically 50 μ s, but it can be between 20 and 200 μ s. However, while the time between bits has a delay time of 2 ms, it might be as short as 200 μ s or as long as 10 ms.

A keypad device that is used to collect “ID code” data will normally have the “facility code” data preprogrammed so the user need only enter the ID code consisting of up to five digits followed by the “#” (enter key, the “*” is a clear entry key). Upon pressing “#,” the logic either limits the entry or appends zeros to the left if necessary to create a five-digit number. This number is limited to between 0 and 65535 so that it fits within a four-digit hex value. Entering “1234” will be sent as “04D2” (hex) because the Wiegand format limits the ID code to four hexadecimal digits. Let’s say the facility code is set to 170 (that’s “AA” in hex), then the six-digit (hex) data would be “AA04D2.”

Once this data has the even and odd parity bit added, it will be ready to output. If we look at this data in binary, it is much easier to see the parity: 1010-1010-0000-0100-1101-0010. There are four 1 bits amongst the first 12 bits. That’s an even number, so the even parity must be “0” to keep the parity even. There are five 1 bits in the last 12 bits, and that’s already odd, so again the odd parity bit must be a “0” to keep the parity odd. The parity bit is set to whatever state is necessary for the total set of 12 data bits plus 1 parity bit to be the required parity. With the complete 26-bit Wiegand format established, this string of data is output through the Wiegand interface.

```

010101010 00000100110100100
  A A 0 4 D 2
  
```

WIEGAND DECODING

On the outside, it may look like a no-brainer to decode a Wiegand communication string. For the most part you’d be correct. You would just watch the Data0 and Data1

outputs for any low pulse and log each pulse as a 0 or 1. The transmission should have 26 data bits. However, if you want to assure the transmission is legal, there is more to check than just the number of bits received. Besides the parity bit data checks, there is the width of each pulse, the time between non-concurrent pulses. We can set up minimum and maximum pulse and delay times (as mentioned above) and check the incoming data to make sure it fits within these specifications.

There are various ways to implement this. I chose to use the change of state (COS) interrupt originally intended for capturing a key press during a powered down sleep state. You designate which inputs are to be used and an interrupt is generated whenever there is a change from present state. It is important to note that the present state is updated when you read the state, so it's important to do this right away, which will allow additional changes to issue another interrupt (even if mid interrupt). In this application, two inputs are designated, one for Wiegand 0 and one for Wiegand 1. A copy of the last read is saved, so we have a reference to determine which input has changed and to which state it has changed.

The **Figure 2** shows the interrupt routine used for each COS interrupt. Timer 5 is used to determine a time out for the Wiegand communication. Similar to a watchdog timer it is reset each time a COS occurs. As long as this continues the timer will not reach its time out. Upon each COS the state (input and level) is recorded along with the appropriate timer count (in microseconds). We are not verifying anything here, just keeping track of what is happening at the inputs. Once Wiegand data ceases, the timer 5 will overflow, setting a flag which is being tested in the main loop (see **Figure 3**).

The main loop has two functions handling user requests, commands received from the UART, and Timer 5 timeouts, indicating that a Wiegand data transfer has occurred. A terminal program connected to the serial port allows commands to be entered by the user and Wiegand results to be displayed. The command structure is simple. Most commands are single letter entries: D, V, S, W, enable/disable (toggle) Debug data, enable/disable (toggle) Verbose data, display Status, and send a Wiegand command. (It seemed a shame not to include this along with Wiegand decoding.) This last command requires two additional commands to set the facility data (8-bit) and ID data (16-bit) using the format Fxx and Ixxxx. Debug displays Wiegand data as bits instead of bytes. Verbose data displays the timing info recorded for each COS.

All the heavy lifting occurs once Timer 5 overflows. **Figure 4** outlines the process of

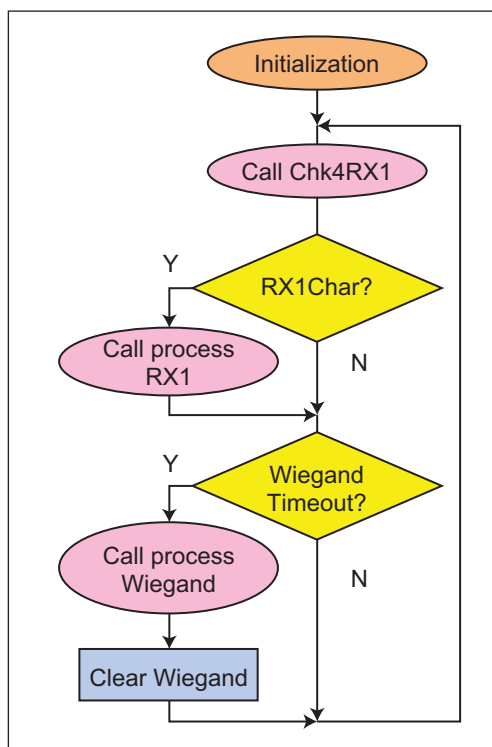


FIGURE 3

The applications Main loop consists of handling user requests and processing any Wiegand data that has been captured. While not necessary to the final application, user commands allow some flexibility in what is displayed.

reviewing the state and timing data recorded prior to Timer 5's overflow. The process steps through this list looking for the rising COS for each input. This indicates the end of each data pulse. The appropriate logic state—"0" for Data 0 and "1" for Data 1—is shifted into a FIFO to collect the transmitted data states. Every entry (whether it's a pulse or delay) is checked for proper timing based on preconfigured constants for minimum and maximum times for pulses and delays. Actually, the list's first entry skips a timing check, as the timer is free running at this point (and meaningless.)

Once all the entries have been processed, we can now determine if the data shifted into the FIFO is legal. For this application, I am just interested in the standard 26-bit Wiegand format. Any other bit count indicates an illegal Wiegand format. I've left sufficient wiggle room for nonstandard formats. These could be implemented but since many are proprietary, I'll leave this up to the reader.

The even and odd parity bits that bookend the data help to indicate a potential error in the data. Parity bits are the simplest form of error detecting. They cannot be used to correct any errors, as there is no way to determine which particular bits are corrupted. Therefore, if parity is determined to be incorrect, the entire transmission is considered bad and is tossed into the bit bucket. The parity bit is used to force the data's total parity to conform to the chosen parity, either even or odd. If the data's parity is even and we want even parity, then the parity bit is 0, which does not affect the



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@imaginethatnow.com or at www.imaginethatnow.com.

total parity. If the data's parity is even and we want odd parity, then the parity bit is 1, which changes the total parity.

As I mentioned earlier, the first 12 data bits (that's the facility code and the first 4-bits of the ID code) use a leading even parity bit and the last 12 data bits (ID code) use a trailing odd parity bit for a total of 26-bits. If the first 13 bits do not conform to even parity and the last 13-bits do not conform to odd parity, then a bad parity message is displayed. Otherwise, the Wiegand data is displayed in either bit or byte form.

WIEGAND DATA DISPLAY

My first thought was to make this a standalone project with a small LCD to display each six-digit hex code picked off the Wiegand data lines. When I was deciding how to build

this, I came across some old prototype PCBs I had left over from a previous Bluetooth project. I decided to use it just for that reason. I could eliminate both the display and the wired connection to my computer. This would allow the project to connect directly to the four-wire Wiegand bus and act as both a wireless Wiegand logger and keypad emulator.

With this arrangement, I only needed to bring my laptop (or Android phone) in the vicinity to make a connection and monitor Wiegand usage or enter a Wiegand code right from my device. Using my laptop and Realterm (terminal emulation program), I get output as shown in **Photo 2**. You'll notice some disturbing messages prior to the timing for each COS. The pulse and delay timing is outside of the maximum allowable times in the Wiegand specifications. I must have made some error in

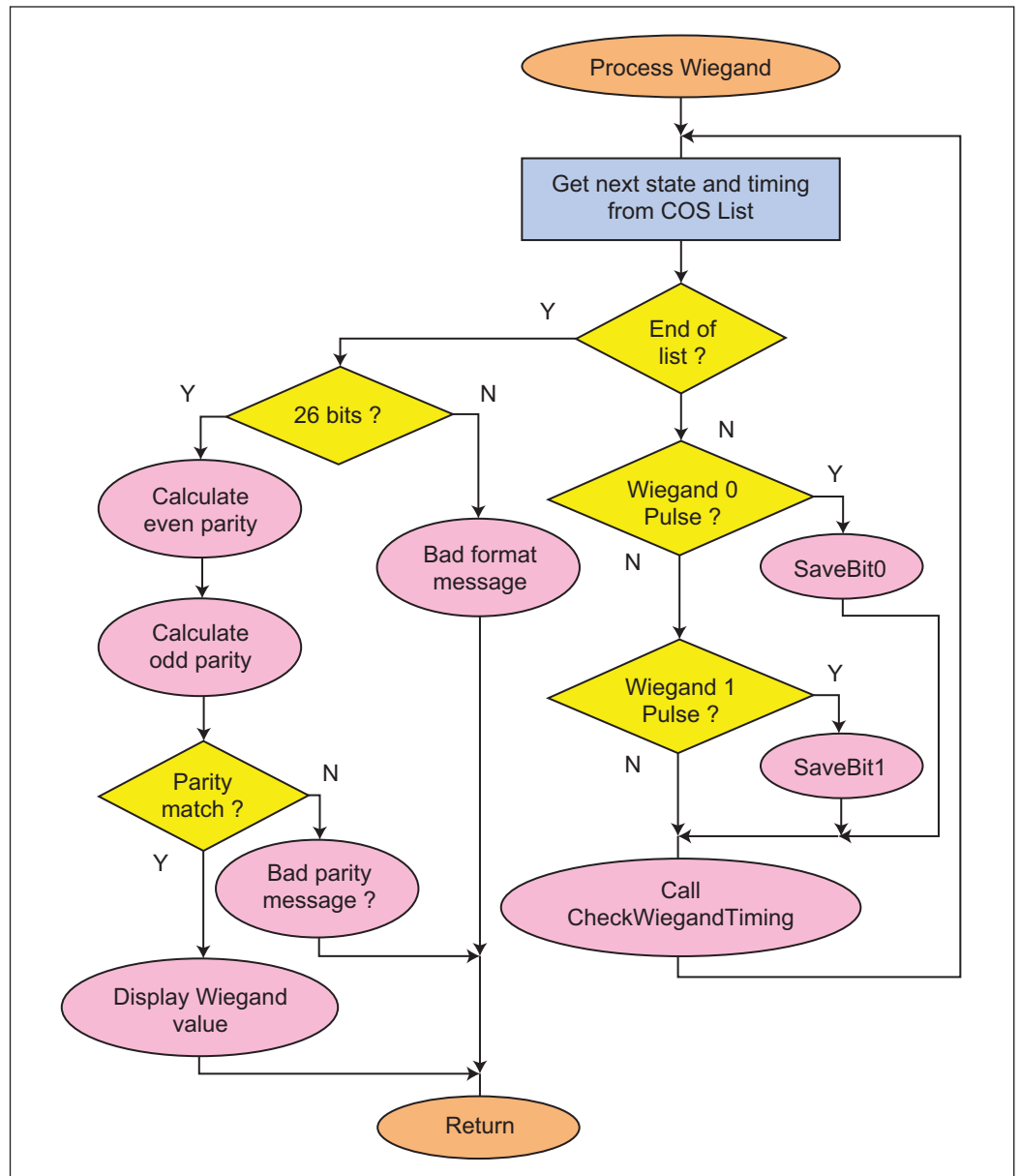
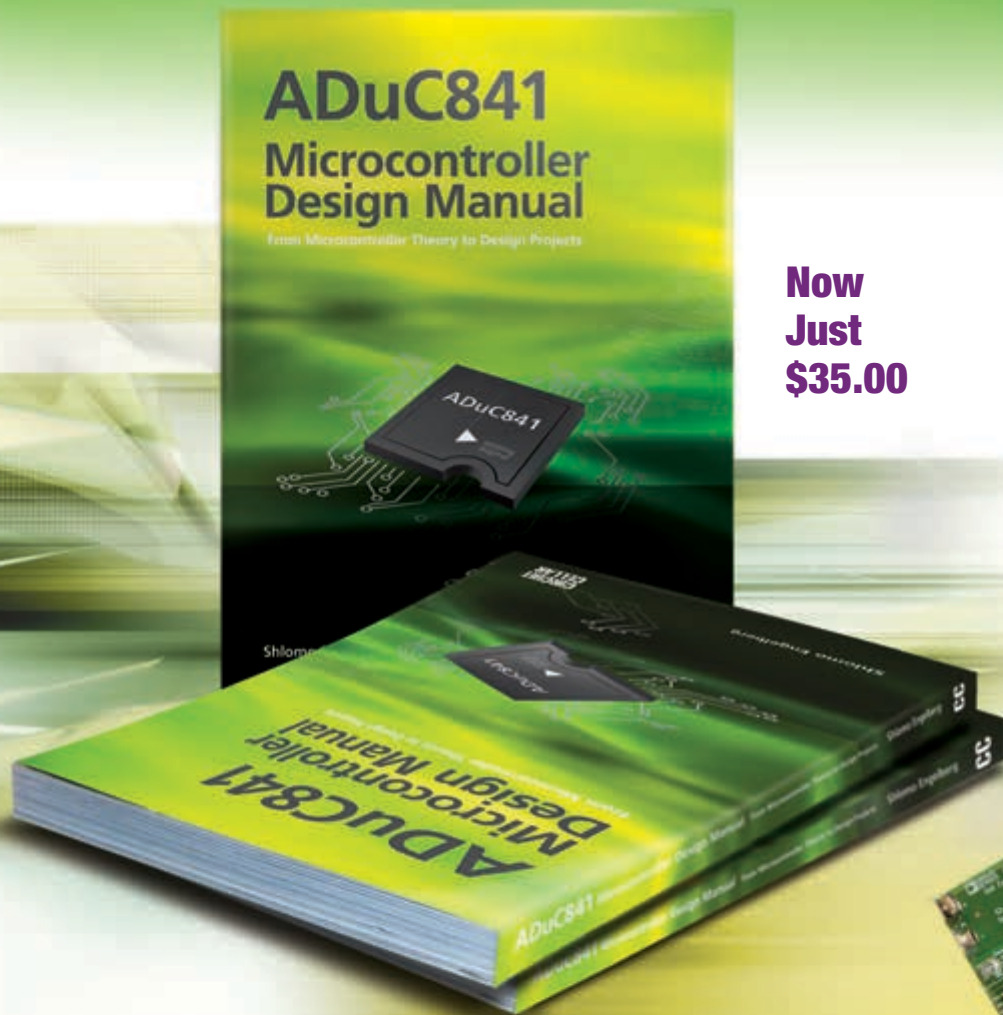


FIGURE 4

While you can receive Wiegand data without verifying specification timing, recording timing information allows you to compare and reject any transmission that does not adhere to the spec.

ADuC841 Microcontroller Design Manual: From Microcontroller Theory to Design Projects

If you've ever wanted to design and program with the ADuC841 microcontroller, or other microcontrollers in the 8051 family, this is the book for you. With introductory and advanced labs, you'll soon master the many ways to use a microcontroller. Perfect for academics!



**Now
Just
\$35.00**

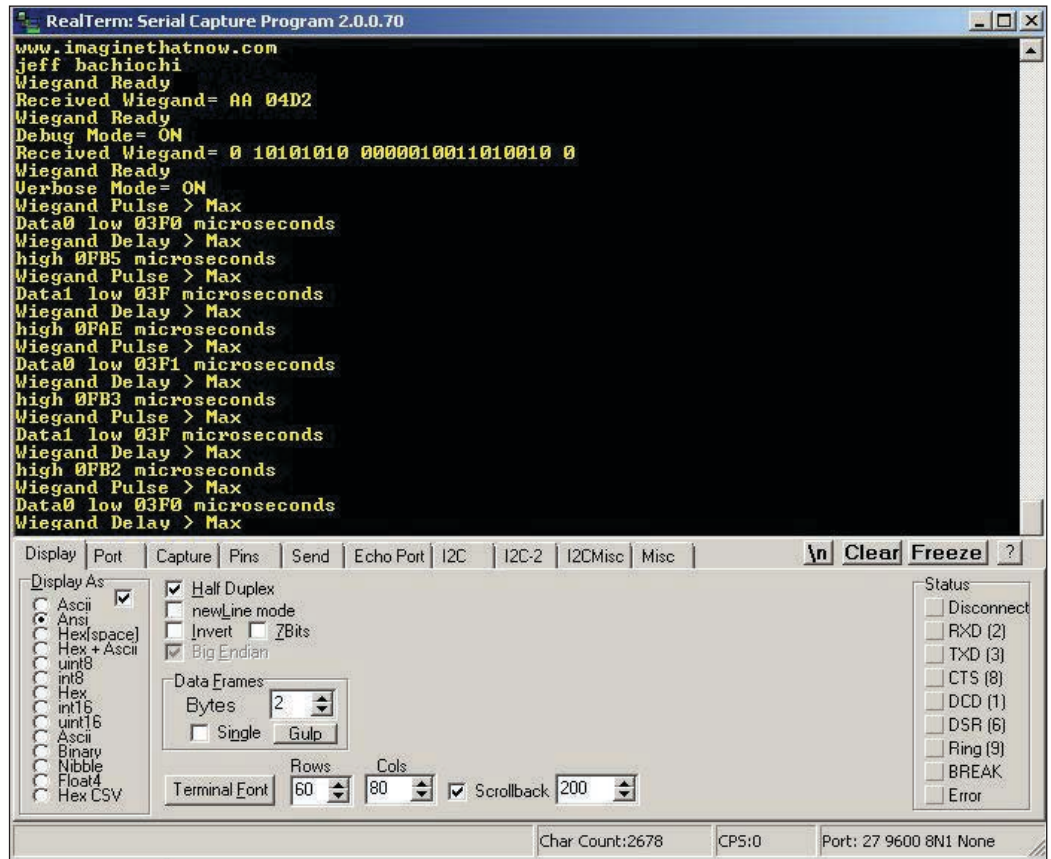
Buy it today!

www.cc-webshop.com



PHOTO 2

Communication with the project is over a Bluetooth serial connection. Here you can see a sign-on message and the first Wiegand entry "1234" is displayed as "AA 04D2." Entering "D" toggles debug mode on. A second Wiegand entry "1234" is now displayed in bit form. Entering "V" toggles verbose mode on. Now we see timing information on a third Wiegand entry.



a routine somewhere. The resultant Wiegand code displayed at the bottom of the screen is correct. Hmm.

I connected a scope to the Wiegand bus to see the actual timing. **Photo 3** shows what I found. The scope's sweep parameter verified that my timing information was correct. This keypad was not within the standard specification. How was that possible? I Googled the keypad part number and was directed to Essex Electronics (www.keyless.com). My keypad didn't seem to be in their line-up, so I contacted them. I received an immediate reply to assure me that my keypad was in fact a customized device. I was assured that

other than the relaxed specs, the keypad was identical to the company's standard product, KTP-103. The keypad is heavy. In fact, I was so curious that I weighed it found it was 1 lb. So, the mystery was solved and my application was in fact giving me the correct information.

To send a Wiegand transmission doesn't require all the hoop jumping I had to implement here to read a Wiegand transmission. A single timer can be used to alternately time the duration of each of my pulses and inter-pulse delays. I have commands set up to allow facility and ID codes to be entered and one to send a Wiegand transmission using those codes. To keep the open collector hardware bus architecture intact, we can't just turn the Wiegand inputs to outputs and drive the bus high or low. An open collector output is not driven high, it's pulled high by an external resistor. This allows any device on the bus to pull the bus low at any time. Any logic low wins over a logic high. Without open collector output drivers you need to play games. To output a logic high (idle), the I/O is treated as an input. To output a logic low (pulse), the I/O is treated as an output with a logic low applied. In fact we can leave the output loaded with a "0" and just change the direction of the I/O bit. That is, for a logic high the port bit remains as a input, the external pull-up creates the logic high on the bus, for a logic low the port bit



circuitcellar.com/cmaterials

J. Wiegand, "Bistable Magnetic Device," US Patent US3820090 A, 1974, www.google.com/patents/US3820090.

SOURCE

PIC18F23K22 Microcontroller
Microchip Technology | www.microchip.com

RESOURCES

T. E. Stimson, "A House of Magic," *Popular Science*, August 1954.



PHOTO 3


The scope doesn't lie. This Wiegand transmission is out of spec. The low pulse is measuring 1.4 ms with a delay of 6.2 ms.

is changed to an output and the output driver pulls the external pull-ups to ground. The fact that a logic low remains on the port bits output latch is a 'don't care' when that port bit is defined as an input. So here we are not forcing an output high and low, just configuring the bit as an input (for a 1) and output (for a 0).

IMPLEMENTATION

While this application was built using a 28-pin Microchip Technology PIC18F23K22 microcontroller, there are 16 I/Os that are not used (see **Figure 5**). Certainly, a smaller

microcontroller could be used here, unless you wish to add a keypad and/or display to make a full-fledged input device. There is plenty of room left for additional functions.

You can search the internet and find a plethora of Wiegand interfaced devices still being produced. While many manufacturers have options available for other communication medium, Wiegand is still king. With a vast installed base, Wiegand is not going away any time soon. 

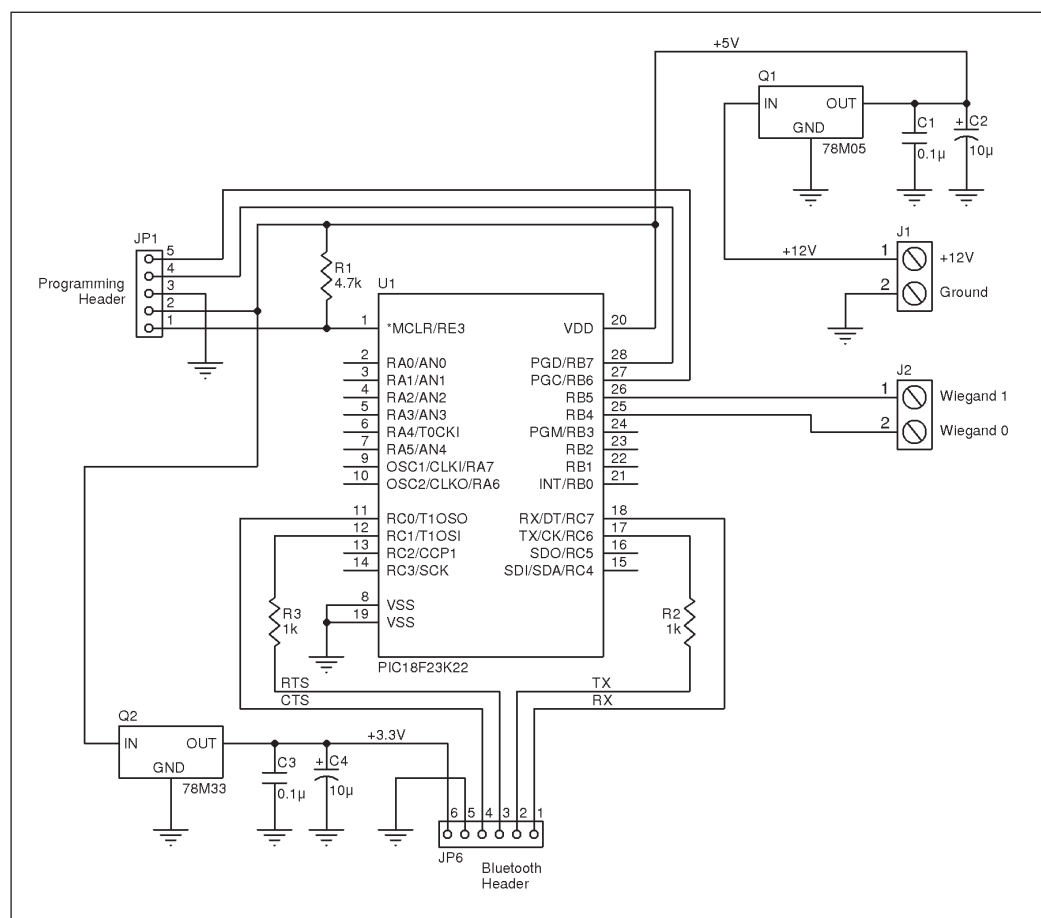


FIGURE 5

This application only requires about six I/Os. The majority of room is taken up by connectors as seen in here. Note that 5-V logic is used to remain compatible with the Wiegand bus. Most bluetooth modules require the use of 3.3 V, so series resistors are in series with any driven lines to protect the Bluetooth inputs.

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 ADUC841 MICROCONTROLLER DESIGN MANUAL

This book presents a comprehensive guide to designing and programming with the Analog Devices ADuC841 microcontroller and other microcontrollers in the 8051 family. It includes a set of introductory labs that detail how to use these microcontrollers' most standard features, and includes a set of more advanced labs, many of which make use of features available only on the ADuC841 microcontroller.

The more advanced labs include several projects that introduce you to ADCs, DACs, and their applications. Other projects demonstrate some of the many ways you can use a microcontroller to solve practical problems. The Keil μ Vision4 IDE is introduced early on, and it is used throughout the book. This book is perfect for a university classroom setting or for independent study.

Author: Shlomo Engelberg

Item #: CC-BK-9780963013347



4

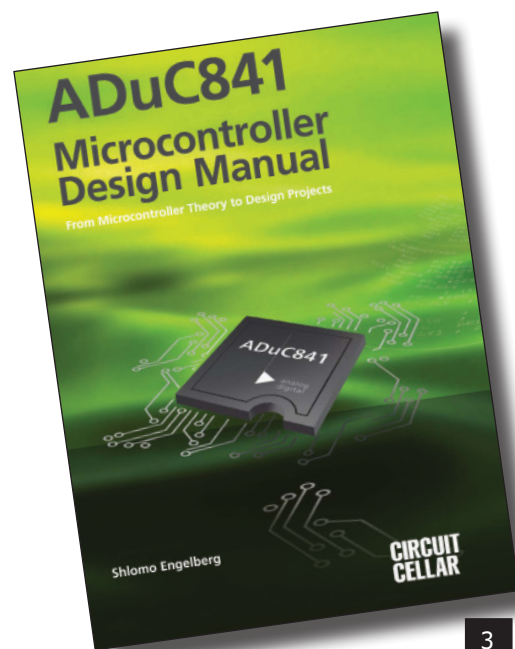
1 CC VAULT

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

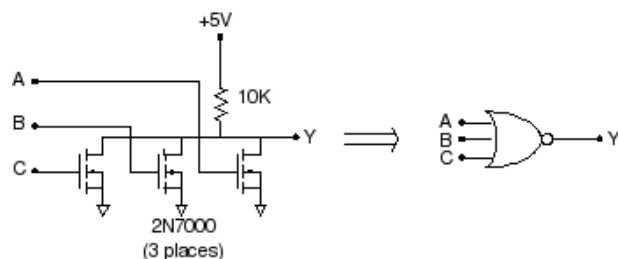
Further information and ordering: www.cc-webshop.com

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

TEST YOUR EQ

Contributed by David Tweed

PROBLEM 1—You have decided to build a small computer from discrete transistors as a demonstration. After researching the available technologies, you have decided to base your design on NMOS logic, using a 3-input NOR gate as your basic building block, as shown below.



Each gate uses three 2N7000 N-channel MOSFETs as pull-down transistors, and a 10-k Ω resistor as a passive pullup. You figure that you'll need somewhere between 500 and 1000 of these gates to build a useful computer—after all, the original PDP-8 12-bit minicomputer CPU was built with only about 519 gates.

Approximately how fast will you be able to clock this computer?

PROBLEM 2—Assuming a supply voltage of 5 V, about how much power would you expect this computer to consume?

PROBLEM 3—How many three-input gates does it take to construct an edge-triggered (master-slave) D flip-flop?

PROBLEM 4—What famous computer was built using NOR gates exclusively for the logic?

The answers to these EQ problems will appear in *Circuit Cellar 303* (October 2015) and at CircuitCellar.com.

Sign up for the FREE Circuit Cellar Newsletter!

You'll receive electrical engineering tips, interesting electronics projects, embedded systems industry news, and exclusive product deals via e-mail to your inbox on a regular basis. If you're looking for essential electrical engineering-related information, we've got you covered: microcontroller-based projects, embedded development, programmable logic, wireless communications, robotics, analog techniques, embedded programming, and more!

Subscribe now to stay up-to-date with our latest content, news, and offers!

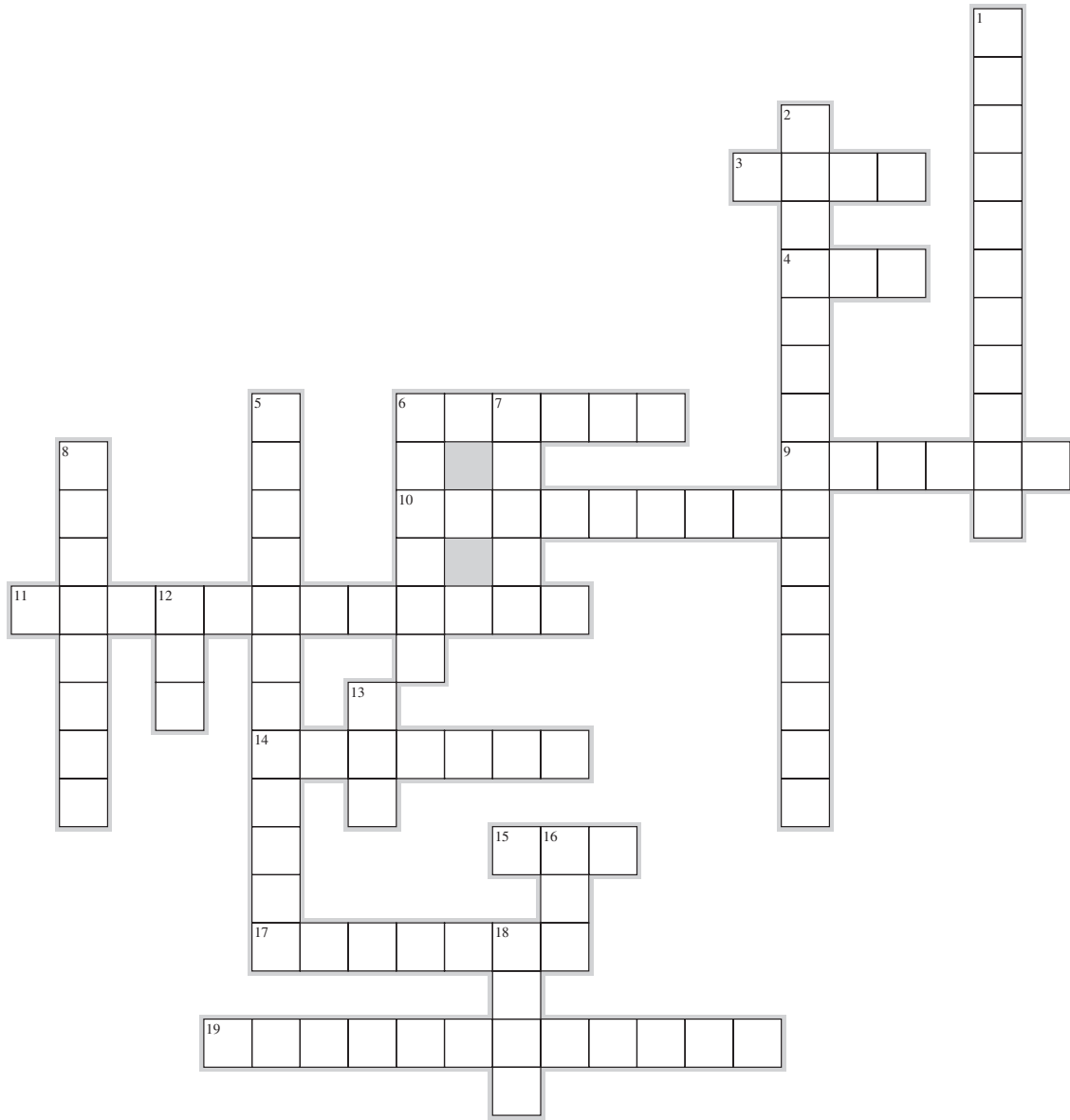
circuitcellar.com



CROSSWORD

SEPTEMBER 2015

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



ACROSS

3. Decrease amplitude
4. Light intensity
6. Quantum theory
9. The Great Explainer
10. Leyden jar
11. Measures small electrical currents by means of deflecting magnetic coils
14. Metal containing iron
15. Big Blue
17. EW
19. 3.00×10^8 m/s

DOWN

1. ENUM
2. GaAs
5. $F = qE + qv \times B$
6. Chunk of data
7. Introduced the LISA computer in 1983
8. 1 quadrillion bytes
12. Range = 30 and 300 MH
13. Error-detecting code
16. Batch file
18. Thinner than QFP

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

PLCC Emulation Plug

Upgraded modules to PLCC Socket

- True J-lead emulation
- Low mass for easy assembly
- 20, 28, 32, 44, 52, 68 and 84 pin
- Available with alignment pins
- Volume pricing available
- <50 milliohm contact resistance



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

ARDUINO Peripherals

XMEM+

External Memory plus Parallel Bus Expansion


- For Mega 2560 and Mega ADK
- Adds Memory Space, Seamlessly
- 512K SRAM (32K x 16 banks)
- On-Board High-Speed Logic
- Buffered Address, Data, Control
- Supports 3.3V and 5V Circuitry



DIO24-ARD

Digital Interface

- 24 Digital I/O Channels
- 85ma Output Sink Current
- Uses Standard SPI Library Routines
- Connects to I/O Racks, LEDs, Switches, Relays
- Industrial Operating Temperature Range



Learn More Details at . . .

SCIDYNE® Call Toll-Free 1-877-SCIDYNE (1-877-724-3963)

NEW!

Built-in PID Functions

Included in CCS C Compilers

Fast Calculations with PID Module


Setup PID in 3 steps:

- Setup K values
- Enable module & result functions
- Input new data to the system & get the output

setup_pid(PID_MODE_PID, K1, K2, K3)

FREE 45 Day Demo:
ccsinfo.com/CC915

sales@ccsinfo.com 262-522-6500 x 35



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

microEngineering Labs, Inc.

www.melabs.com 888-316-1753

Programmers for Microchip PIC® Microcontrollers



Starting at \$79.95

PC-Tethered USB Model (shown):

- Standalone software
- Command-line operation
- Hide GUI for automated use
- Override configuration with drop-downs

Stand-Alone Field Programmer:

- Power from target device or adapter
- Program file stored on SD-CARD
- Programming options stored in file
- Single-button operation

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

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

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 Engineering Research and Environment Systems Modeling

By R. Scott Coppersmith



R. Scott Coppersmith earned a BSc in Electrical Engineering at Michigan Technological University. He held several engineering positions in the automotive industry from the late 1980s until 2010 when he joined the University of Notre Dame's Civil Engineering and Geological Sciences department as Research Engineer to help build an Environmental Fluid Dynamics laboratory and assist students, faculty, and visiting researchers with their projects. Scott also teaches a variety of engineering courses (e.g., Intro to Microcontrollers and Graphic Communication for Manufacturing) at Ivy Tech Community College.

So many bytes, so little time. Five years ago, I found myself looking for a new career. After 20 years in the automotive sector, the economic downturn hit home and my time had come. I was lucky enough to find a position at the University of Notre Dame designing and building lab instrumentation and data acquisition equipment in the Department of Civil and Environmental Engineering & Earth Sciences, and teaching microprocessor programming in the evenings at Ivy Tech Community College. The transition from industry to the academic world has been challenging and rewarding. Component and System modeling using computer simulation is an integral part of all engineering disciplines. Much of the industry simulation software started out in a university computer lab.

A successful computer simulation of a physical phenomenon has several requirements. The first requirement is a stable model based on a set of equations relating to the physics and scale of the event. For complex systems, this model may not exist, and a simplified model may be used to approximate the event as close as possible. Assumptions are made where data is scarce. The second requirement is a set of initial conditions that all the equation variables need to start the simulation. These values are usually determined by running real-world experiments and capturing a "snapshot" of the event at a specific time. The quality of this data depends on the technology available at the time. The technology behind sensors and data acquisition for these experiments is evolving at an incredible rate. Some sensors that may have cost \$500 10 years ago are available now for \$5 and have been miniaturized to one tenth of its original size to fit into a cell phone or smart band. Equipment that was too large to be used out of a lab environment is now pocket sized and portable. Researchers are taking advantage of this, and taking much more data than ever imagined.

So how will this affect the future of simulation? Multicore processors and distributed computing are allowing researchers to run

more simulations and get results quicker. Our world has become Internet driven and people want data immediately, so data must become available as close to real-time as possible. As more and more sensors become wireless, low cost, energy efficient, and "smart" due to the


Internet of Things movement, empirical data is available from places never before conceived. Imagine the possible advancements in weather modeling and forecasting if every cell phone in the world sent temperature, humidity, barometric pressure, GPS, and light intensity data to a cloud database automatically. More sensors lead to higher simulation resolution and more accuracy.

A popular saying, "garbage in = garbage out," still applies, and is the bane of the Internet. Our future programmers must be able to sift through all of this new data and determine the good from the bad. Evil hackers

enjoy destroying databases, so security is a major concern. Some of this new technology that could be useful in research is being rejected by the public due to criminal use. For example, a UAV "drone" that can survey a farmer's crop can also deliver contraband or cause havoc at an airport or sporting event.

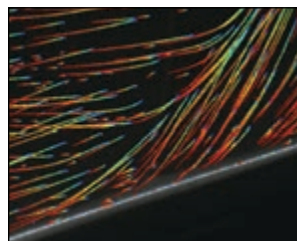
While these issues are tackled in the courtroom and the FAA, researchers are waiting to take more data.

Simulation is still only a guess at what may happen under specific conditions based on assumptions of how our world works. The advancements in sensor and data acquisition technology will continue to improve the accuracy of these guesses, as long as we can de-

pend on the reliability of the input sources and keep the evil hackers out of the databases. Schools still need to train students on how to determine good data from questionable data. The terabyte question for the future of simulation is whether or not we will be able to find the data we need in the format we need, searching through all these new data sources in less time than it would take to run the original experiments ourselves. So many bytes, so little time. 



Laser Particle Image Velocimetry (PIV) Tank measuring 2-D flow through a long cylinder



Flow movement due to heat flux on inclined surface. Simulation of a heated mountainside from the sun.

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

We bring the full range of *Electronic Contract Manufacturing services to your fingertip!*

FABRICATION



ASSEMBLY



KEYPADS



ENCLOSURES



This is the only place where you would put all your eggs in one basket to get fastest time to market. From concept design to prototype to full turnkey production on all your electronic products.



imagineering inc. www.PCBnet.com

**847-806-0003 sales@PCBnet.com
Certified Woman-Owned Small Business**