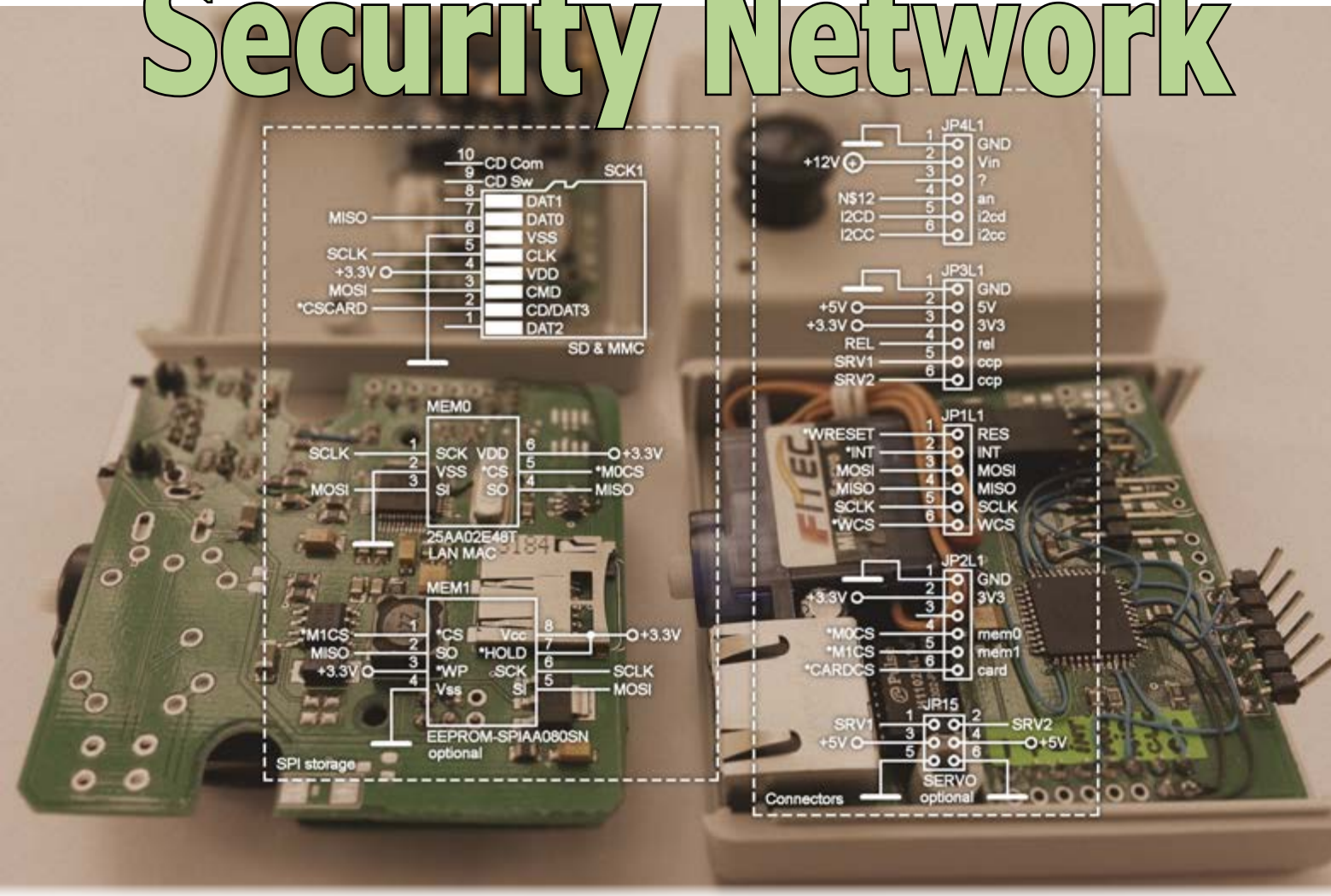
circuitcellar.com

INTERNET & CONNECTIVITY
JULY 2015
ISSUE 300

circuit cellar

'Net-Connected Security Network

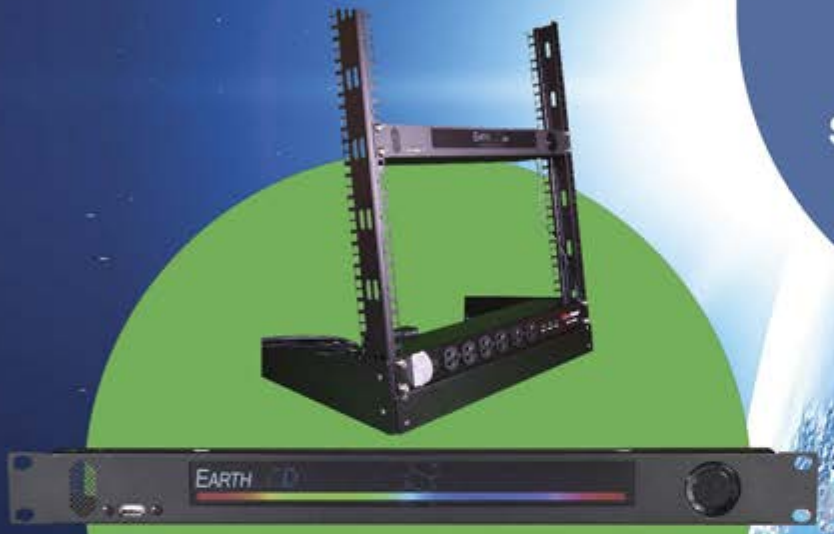


Editors' Picks: Internet & Connectivity Projects | DIY Acoustic Recording App | Microcontroller-Based WAVE Player | How to Build a Multi-Node Security System | Sustain Mobile System Performance | Magnetic Shielding Explained | Touch Screen Control Panel Project | LIDAR 101 | What Are Interconnect Defects?

$9.00US $10.00CAN
07
0 74470 75349 0

Making LCDs Work for 21 Years

ezLCD
Smart, All-In-One, Color TFT
LCD Touchscreen Modules
from $99.00
ezLCD.com



10" x 1" TFT 1U Rack Appliance
1 GB Linux CPU
Open Source Hardware
LCD Available Separately
Pi-RAQ.com

Multi-Touch 15" Atom PanelPC
OF-15.0-VGA-ET
EarthIPC.com

EARTHLCD.com

"The Smart LCD Company"

SUPERIOR EMBEDDED SOLUTIONS

Technologic Systems

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

Starting at
$89
Qty 100
$122
Qty 1

- 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 ºC to 85 ºC Industrial Temperature Range
- Runs Linux, Android, QNX, Windows
- QT, OpenGL, DirectFB, GNU Tool Kit, and More

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

Starting at
$299
Qty 100
$342
Qty 1

- 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

Enclosed TPCs
Also Available

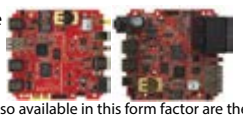
TS-7970
Single Board Computer

NEW!

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

Starting at
$169
Qty 100
$214
Qty 1

- 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 ºC to 85 ºC Industrial Temperature Range
- HDMI, LVDS, and Audio In/Out Connections
- Runs Linux, Android, QNX, Windows



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

(Shown with optional microSD card)

TS-7250-V2
Single Board Computer

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

Starting at
$165
Qty 100
$199
Qty 1

- 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 ºC to 85 ºC Industrial Temperature Range
- Preinstalled Debian Linux OS and Utilities

Available with TS-ENC720 enclosure

LONG LIFE — We've never discontinued a product in 30 years

RUGGED — Embedded systems that are built to endure

OPEN — Support every step of the way with open source vision

ORIGINAL — Unique embedded solutions add value for our customers

www.embeddedARM.com

Issue 300 July 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

ENGINEERING CONNECTED SOLUTIONS

Most consumers take it for granted that the majority of the electronic devices they purchase will be connected to the Internet. In fact, the average consumer would be hard pressed to come across a new electronic system of any value that isn't Internet capable. Many new watches, glasses, audiovisual systems, home appliances, cameras, and health-related systems are 'Net-connected. But electrical engineers don't take this for granted. You've dedicated your careers to designing, building, and testing the connected products that are changing the ways we all interact with machines.

One the many perks associated with working at Circuit Cellar is that our staffers get to see how the world's top engineers develop new embedded technologies. In this issue, we feature some of the most interesting projects we've seen to date.

As part of a project at the University of Calgary, Adrien Gaspard, Mike Smith, and Nicholas Lepine created a way to analyze the frequency response of their surroundings, including resonances. Starting on page 16, they describe their "recording app" with a number of custom DSP extensions for resonance identification.

On page 26, Wangcheng Zhou explains how he built a cost-effective, Atmel AVR-based wave player featuring CD-quality sound. The design can play 8-/16-bit mono/stereo standard RIFF wave files.

Interested in building an Internet-connected security network? Claudiu Chiculita and Liviu Ene designed an innovate system comprising several nodes that collect and process information independently, generate alarms, and examine threats from multiple angles (p. 40).

Mobile technologies are rapidly changing the ways we interact with the people and machines around us. On page 48, Ayse Coskun tackles the subject of modern smartphone thermal management.

Turn to page 54 for the final article in George Novacek's series, "Shielding 101." This month he covers magnetic shielding, absorption, and rules for cabinet design.

Ed Nisley continues to investigate electrical engineering topics by upgrading his Kenmore 158 sewing machine. On page 58 he explains how he has the machine's UI running on an Arduino Mega with a TFT LCD and a resistive touch screen overlay.

Interested in learning about Light Detection and Ranging (LIDAR) technology? Jeff Bachiochi has you covered (p. 66). He explains how to use it to measure distances and why you might implement it instead of IR or ultrasonic sensors.

We wrap up the issue with Doug Trobough's introduction to interconnect defects (ICDs). In addition to detailing the differences between debris-based and copper bond failure ICDs, he provides some thoughts on the future of PCB design.

C. J. Abate
cabate@circuitcellar.com

THE TEAM

OUR NETWORK

circuit cellar audioxpress VOICE COIL

SUPPORTING COMPANIES

NOT A SUPPORTING COMPANY YET?

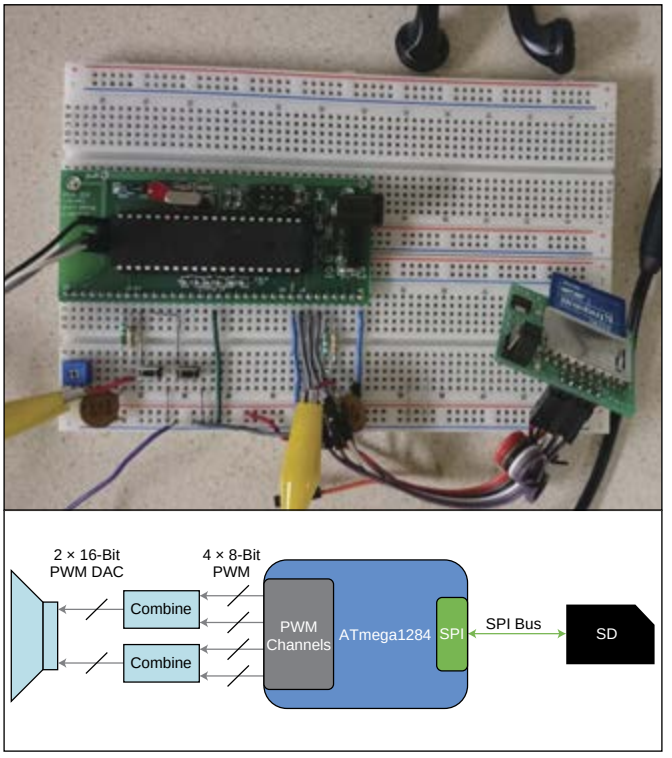
Contact Peter Wostrel (circuitcellar@smmarketing.us, Phone 978.281.7708, Fax 978.281.7706) to reserve your own space for the next edition of our members' magazine.

CONTENTS

JULY 2015 • ISSUE 300

INTERNET & CONNECTIVITY

circuit cellar

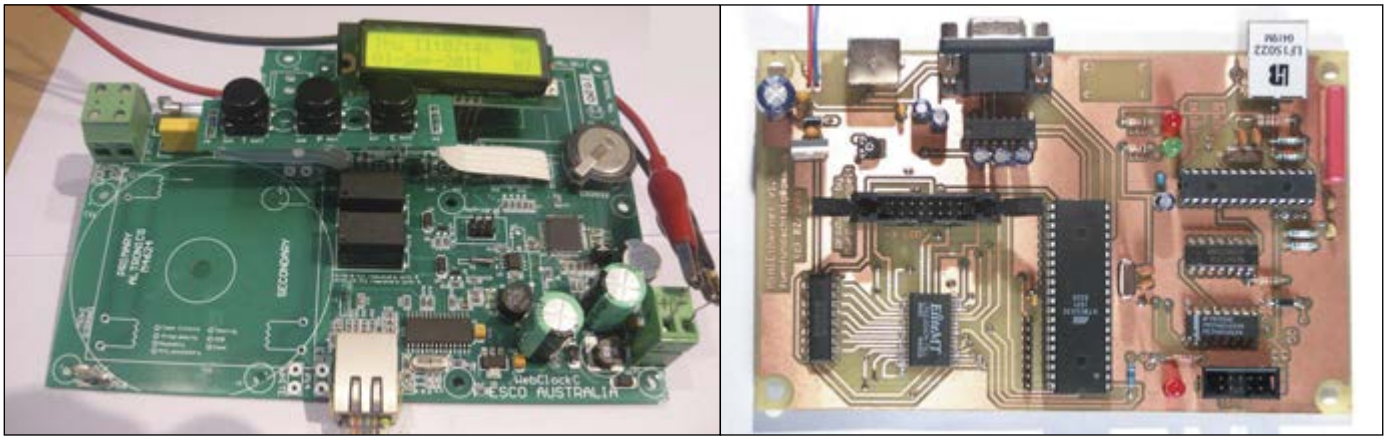


AVR-BASED WAVE PLAYER

CC COMMUNITY

INDUSTRY & ENTERPRISE

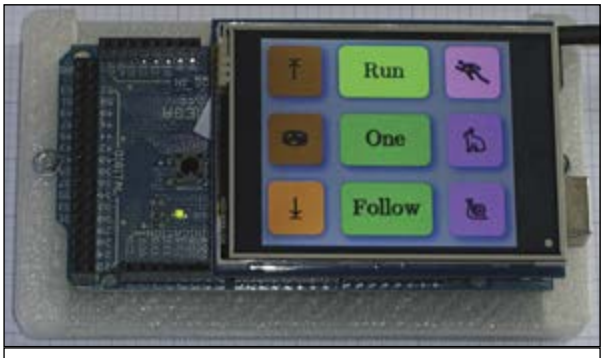
FEATURES



EDITORS' PICKS: INNOVATIVE INTERNET-CONNECTED DESIGNS

CONTENTS

MULTI-NODE, 'NET-CONNECTED SECURITY NETWORK



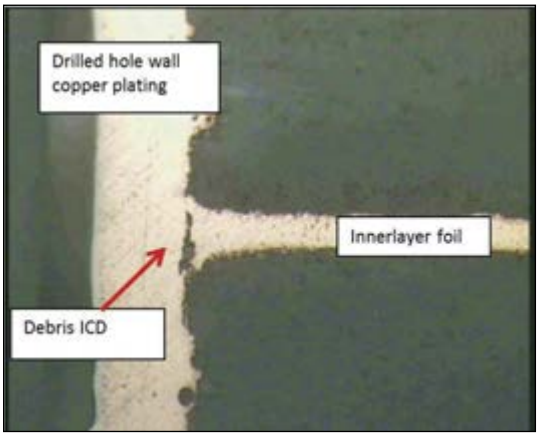

```
#define MAXCMDLEN 2

enum bstatus_t {BT_DISABLED,BT_UP,BT_DOWN};

typedef void (*pBtnFn)(byte BID);  // button action function called when hit

struct button_t {
    byte ID;                    // button identifier, 0 unused
    byte Group;                 // radio button group, 0 for none
    byte Status;                // button status
    word ulX,ulY;               // origin: upper left
    word szX,szY;               // button image size
    pBtnFn pAction;             // button function
    char Cmd[MAXCMDLEN + 1];    // command string
    char NameStem[9];           // button BMP file name - stem only
};

struct button_t Buttons[] = {
    { 1,    1, BT_UP,     0,0,      80,80, DefaultAction, "Nu", "NdUp"},
    { 2,    1, BT_UP,     0,80,     80,80, DefaultAction, "Na", "NdAny"},
    { 3,    1, BT_DOWN,   0,160,    80,80, DefaultAction, "Nd", "NdDn"},

    { 4,    2, BT_DOWN,   80,0,     120,80, DefaultAction, "Pr", "PdRun"},
    { 5,    2, BT_UP,     80,80,    120,80, DefaultAction, "P1", "PdOne"},
    { 6,    2, BT_UP,     80,160,   120,80, DefaultAction, "Pf", "PdFol"},

    { 7,    3, BT_DOWN,  200,0,     80,80, DefaultAction, "Sh", "SpMax"},
    { 8,    3, BT_UP,    200,80,    80,80, DefaultAction, "Sm", "SpMed"},
    { 9,    3, BT_UP,    200,160,   80,80, DefaultAction, "Sl", "SpLow"},
};

byte NumButtons = sizeof(Buttons) / sizeof(struct button_t);
```

THE LOGIC BEHIND A USER INTERFACE

COLUMNS

TESTS & CHALLENGES

TECH THE FUTURE



A LOOK AT DIFFERENT INTERCONNECT DEFECTS

EDITORS' PICKS

Internet & Connectivity

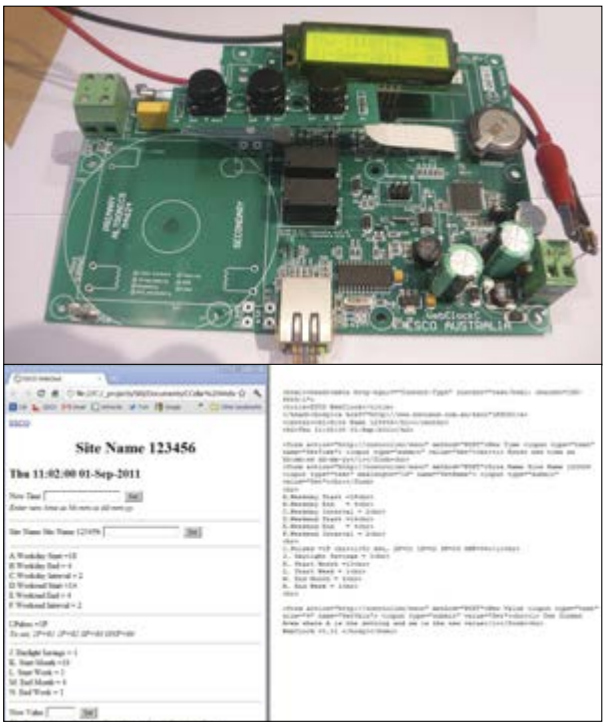
Smart Switch Management

Construct an MCU-Based, 'Net-Enabled Controller

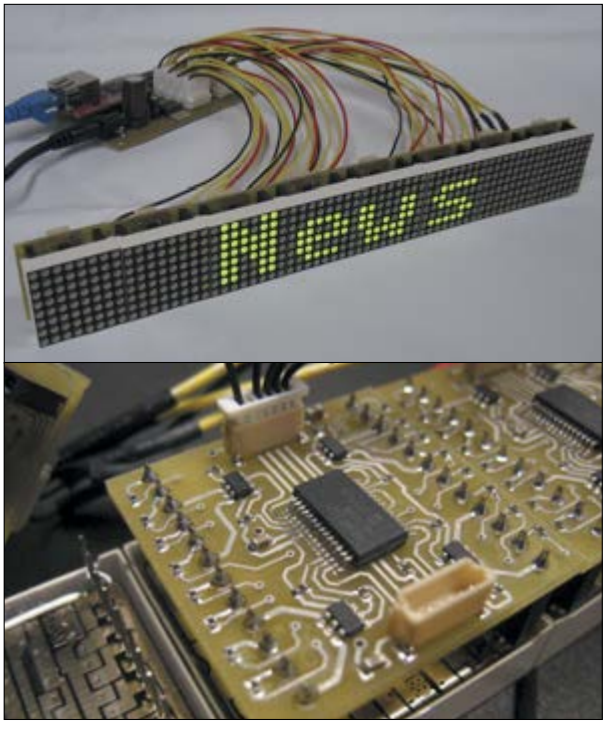
By Fergus Dixon (Circuit Cellar 263, 2012)

Smart switches are handy energy-saving devices that simplify energy conservation. This microcontroller-based, 'Net-connected controller enables you to manage up to 50 switches with ease. Fergus writes:

"This project was to design a controller for up to 50 smart switches. Smart switches are energy-saving devices installed in office blocks to automatically turn off the lights at the end of the day to conserve energy. The controller needed an accurate real-time clock (RTC) that would pulse a 24-V AC line once or twice to turn off the smart switches at the end of the working day, and repeat at two-to-three hour intervals in case the lights were turned on. I added the Ethernet interface after the first prototype was finished."



These articles and others on topics relating to Internet & Connectivity are available at www.cc-webshop.com.



Content Collection and Display

Build an Internet-Connected News Ticker

By James Blackwell (Circuit Cellar 218, 2008)

James no longer has to turn on a TV or computer to get news updates. His innovative design retrieves news headlines from RSS feeds and constantly scrolls them across a dot-matrix LED display. The system checks for updates every 15 minutes. James writes:

"To retrieve the news headlines, I wrote a cut-down HTTP client and XML parser. I wanted the device to operate without any action by the user, so I implemented a domain name service (DNS) and dynamic host configuration protocol (DHCP) client in order to use dynamic IP addressing … Every 15 min., the device renews its client IP, requests the IP of the BBC News RSS server, parses the XML for news headlines, and scrolls them across the display. The system features a Microchip Technology PIC18F2525, PIC18F2221, and a WIZnet WIZ810MJ Ethernet module. In this article, I'll describe how I used these building blocks to design the news ticker."

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 Mocrovias, 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

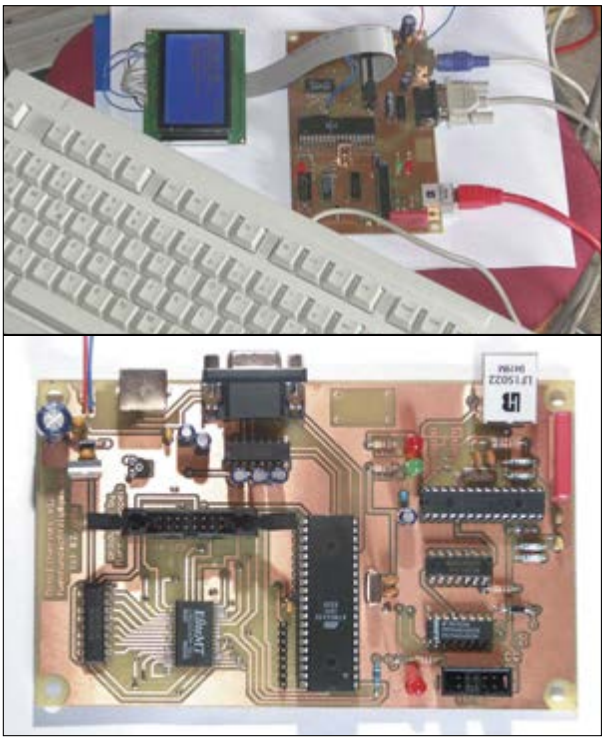
EDITORS' PICKS

Internet & Connectivity

MiniEmail

A Compact MCU-Based Mail Client

By Alexander Mann (Circuit Cellar 204, 2007)



Alexander's mail client system uses an Atmel ATmega32 microcontroller and a Microchip Technology ENC28J60 Ethernet controller to continually check for e-mail. When mail arrives, you can immediately read it on the system's LCD and respond with a standard keyboard. Alexander writes:

"My MiniEmail system is a compact microcontroller-based mail client. The silent, easy-to-use system doesn't require a lot of power and it is immune to mail worms. Another advantage is the system's short start-up time. If you want to write a quick e-mail but your PC is off, you can simply switch on the miniature e-mail client and start writing without having to wait for your PC to boot up and load the necessary applications. All you need is an Ethernet connection and the MiniEmail system … The main components are an Atmel ATmega32 microcontroller and a Microchip Technology ENC28J60 Ethernet Controller."

These articles and others on topics relating to Internet & Connectivity are available at www.cc-webshop.com.

Build Your Own 8051 Web Server

By Jim Brady (Circuit Cellar 146, 2002)

Building your own web server can be a difficult task, especially if you proceed without proper direction and the right parts for the job. Fortunately, Jim has finished an 8051 server and he's eager to walk you through his project. With this tutorial, you can avoid common difficulties. Jim writes:

"This article grew out of an experiment to see how hard it would be to build an 8051 web server and write a minimal TCP/IP stack. It seems like everything is serving web pages these days, so why not an 8051? It was not easy, but it was a fun project. After a few months of studying ARP and TCP, I had something up and running. In this article, I'll explain how I built an 8051 web server and describe what I learned along the way. I'll also discuss timing and performance."

circuit cellar

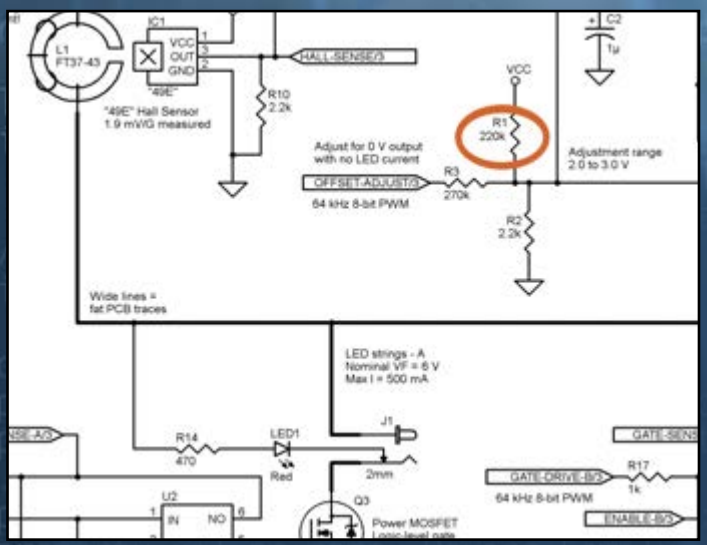
Sponsored by NetBurner

NetBurner

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

PRODUCT NEWS

ARTIK PLATFORM ACCELERATES IoT DEVELOPMENT

Samsung Electronics Co. recently announced the Samsung ARTIK platform to allow faster, simpler development of new enterprise, industrial, and consumer applications for the Internet of Things (IoT). ARTIK is an open platform that includes a best-in-class family of integrated production-ready modules, advanced software, development boards, drivers, tools, security features and cloud connectivity designed to help accelerate development of a new generation of better, smarter IoT devices, solutions and services.

All members of the Samsung ARTIK family incorporate unique embedded hardware security technology, on-board memory and advanced processing power in an open platform. Security is also a key element of the advanced software integrated into the platform, along with the ability to connect to the Internet for cloud-based data analytics and enhanced services. As an open platform, Samsung ARTIK can be easily customized for more rapid deployment of IoT devices and the services that can be delivered using them.

The Samsung ARTIK platform comes in a variety of configurations to meet the specific requirements of a wide range of devices from wearables and home automation, to smart lighting and industrial applications. Initial members of the ARTIK family include:

ARTIK 1, the smallest IoT module currently available at 12 mm × 12 mm combines Bluetooth/BLE connectivity and a nine-axis sensor with superior compute capabilities and power consumption. It is specifically designed for low-power, small form factor IoT applications.

ARTIK 5 delivers an outstanding balance of size, power and price-performance and is ideal for home hubs, drones and high-end wearables. It incorporates a 1-GHz dual-core processor and on-board DRAM and flash memory.

ARTIK 10 delivers advanced capabilities and high-performance to IoT with an eight-core processor, full 1080p video decoding/encoding, 5.1 audio and 2-GB DRAM along with 16-GB flash memory. The Samsung ARTIK 10 includes Wi-Fi, Bluetooth/BLE and ZigBee connectivity and is designed for use with home servers, media applications, and in industrial settings.

Additional technical highlights include:

Security and privacy: The ARTIK platform offers a best-in-class, end-to-end security solution. At the hardware level, ARTIK contains am embedded secure element that goes beyond software-based encryption solutions alone. At the application level, ARTIK is equipped with a machine learning based anomaly detection system. This allows the user to identify abnormalities and unusual behavior in order to address possible hacking or intrusion activity.

IoT Software Stack: Samsung's ARTIK platform comes with an extensive IoT software stack and tools needed to accelerate product development. Developers can go directly to application framework development, instead of spending time building low-level software libraries.

Local Storage and Computational Capability: ARTIK supports unique local storage and computational capabilities that in most current IoT environments are generally only addressable by large-scale cloud servers. Depending on user requirements, data can be managed locally or in the cloud in encrypted or unencrypted formats.

Low-Power Architecture: The ARTIK platform offers best-in-class power consumption to enable longer battery life for battery operated IoT devices like wearables. The platform includes a tiered architecture that allows applications and tasks to run at the right power-optimized performance and memory utilization.

*Small Form Fact*or: All ARTIK modules offer the smallest form factor in their class. Certain ARTIK modules use Samsung's next generation embedded Package-on-Package (ePoP) technology. Samsung Electromechanics, a global leader in component manufacturing, played a key role in developing advanced packaging technology for ARTIK.

Connectivity: Depending on the configuration, the ARTIK family offers all major connectivity protocols including Wi-Fi, Bluetooth (including BLE) and ZigBee. More information about the ARTIK platform and development tools may be found at www.artik.io.

Samsung Electronics Co. | www.samsung.com/us/

F-RAM EXPANDS THE DENSITY RANGE OF ENERGY-EFFICIENT NONVOLATILE RAMs

Cypress Semiconductor Corp. has introduced a family of 4-Mb serial Ferroelectric Random Access Memories (F-RAMs), which are the industry's highest density serial F-RAMs. The 4-Mb serial F-RAMs feature a 40-MHz SPI, a 2-to-3.6-V operating voltage range and are available in industry-standard, RoHS-compliant package options. All Cypress F-RAMs provide 100 trillion read/write cycle endurance with 10-year data retention at 85°C and 151 years at 65°C.

Cypress F-RAMs are ideal solutions for applications requiring continuous and frequent high-speed reading and writing of data with absolute data security. The 4-Mb serial F-RAM family addresses mission-critical applications such as industrial controls and automation, industrial metering, multifunction printers, test and measurement equipment, and medical wearables.

The 4-Mb serial F-RAMs are currently sampling in industry-standard 8EIAJ and 8TDFN packages. Production expected in the fourth quarter of 2015.

Cypress Semiconductor | www.cypress.com

FPGA/CPLD Boards from JAPAN

HUMANDATA

SAVING COST=TIME with readily available FPGA boards

XILINX FPGA Board (Kintex-7 Series)

Kintex-7 FBG484 FPGA board
XCM-022 series

Kintex-7 | 5W
MRAM | LED
DDR3 | I/O:100
RocketIO | SIF40

- FPGA : XC7K70T-1FBG484C XC7K160T-1FBG484C
- Board size : 86 x 54 [mm]

RoHS compliant

Kintex-7 FBG484 FPGA board
XCM-112 series

Kintex-7 | 5W
RocketIO | LED
DDR3 | I/O:128

- FPGA : XC7K70T-1FBG484C XC7K160T-1FBG484C
- Board size : 43 x 54 [mm]

RoHS compliant

Kintex-7 USB-FPGA board
EDX-008

Kintex-7 | 5W
MRAM | LED
DDR3 | I/O:100
USB Config.
USB Comm. | HI-SPEED

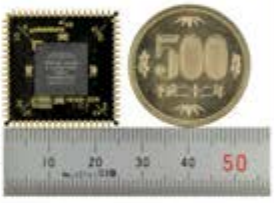
- FPGA : XC7K70T-1FBG484C XC7K160T-1FBG484C
- Board size : 86 x 54 [mm]

RoHS compliant

PLCC68 Series — Stamp size FPGA/CPLD Module

Easy and Quickly Mountable on 68-pin IC socket

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



ALTERA Series

Cyclone V PLCC68 FPGA Module
AP68-06 Series

- FPGA : 5CEBA4U15C8N
- FRAM
- 3.3 V single power supply operation
- On-board oscillator, 50MHz
- Configuration Device
- 8 Layer PCB

RoHS compliant

Cyclone III PLCC68 FPGA Module
AP68-03 Series

- FPGA : EP3C10U256C8N
- 3.3V single power supply operation
- On-board oscillator, 50MHz
- Configuration Device
- 8 Layer PCB
- Board size : 25.3 x 25.3[mm]

RoHS compliant

MAX II PLCC68 CPLD Module
AP68-01 Series

- CPLD : EPM240F100C5N EPM570F100C5N
- 3.3V single power supply operation
- Board size : 25.3 x 25.3[mm]
- 4 Layer PCB

RoHS compliant

XILINX Series

Spartan-6 PLCC68 FPGA Module
XP68-03 Series

- FPGA : XC6SLX45-2CSG324C
- 3.3V single power supply operation
- on-board oscillator, 50MHz
- Configuration Device
- Board size : 25.3 x 25.3[mm]
- 6 Layer PCB

RoHS compliant

Spartan-3AN PLCC68 FPGA Module
XP68-02 Series

- FPGA : XC3S200AN-4FTG256C
- 3.3V single power supply operation
- on-board oscillator, 50MHz
- Board size : 25.3 x 25.3[mm]
- 6 Layer PCB

RoHS compliant

Spartan-6 PLCC68 FPGA Module
XP68-01 Series

- FPGA : XC6SLX16-2CSG225C
- 3.3V single power supply operation
- on-board oscillator, 50MHz
- Configuration Device
- Board size : 25.3 x 25.3[mm]
- 6 Layer PCB

RoHS compliant

We also have many other products
All stocked items are ready to be shipped immediately

Humandata Products are in stock@amazon!!

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

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

www.hdl.co.jp/CC1507

PRODUCT NEWS

HAPPY GECKO MCU FAMILY SIMPLIFIES USB CONNECTIVITY

Silicon Labs recently introduced new energy-friendly USB-enabled microcontrollers (MCUs). Part of its EFM32 32-bit MCU portfolio, the new Happy Gecko MCUs are designed to deliver the lowest USB power drain in the industry, enabling longer battery life and energy-harvesting applications. Based on the ARM Cortex-M0+ core and low-energy peripherals, the Happy Gecko family simplifies USB connectivity for a wide range of Internet of Things (IoT) applications including smart metering, building automation, alarm and security systems, smart accessories, wearable devices, and more.

Happy Gecko USB MCUs feature an advanced energy management system with five energy modes enabling applications to remain in an energy-optimal state by spending as little time as possible in active mode. In deep-sleep mode, Happy Gecko MCUs have an industry-leading 0.9-µA standby current consumption (with a 32.768-kHz RTC, RAM/CPU state retention, brown-out detector and power-on-reset circuitry active). Active-mode power consumption drops down to 130 µA/MHz at 24 MHz with real-world code (prime number algorithm). The USB MCUs further reduce power consumption with a 2-µs wakeup time from Standby mode.

Happy Gecko MCUs feature many of the same low-energy precision analog peripherals included in other popular EFM32 devices. These low-energy peripherals include an analog comparator, supply voltage comparator, on-chip temperature sensor, programmable current digital-to-analog converter (IDAC), and a 12-bit analog-to-digital converter (ADC) with 350-µA current consumption at a 1-MHz sample rate. On-chip AES encryption enables the secure deployment of wireless connectivity for IoT applications such as smart meters and wireless sensor networks.

The Happy Gecko family's exceptional single-die integration enables developers to reduce component count and bill-of-materials (BOM) cost. While typical USB connectivity alternatives require external components such as crystals and regulators, the highly integrated Happy Gecko MCUs eliminate nearly all of these discretes with a crystal-less architecture featuring a full-speed USB PHY, an on-chip regulator and resistors. Happy Gecko MCUs are available in a choice of space-saving QFN, QFP and chip-scale package (CSP) options small enough for use in USB connectors and thin-form-factor wearable designs.

The Happy Gecko family is supported by the Silicon Labs Simplicity Studio development platform, which helps developers simplify low-energy design. Developers can download Simplicity Studio and access Silicon Labs's USB source code and software examples at no charge at www.silabs.com/simplicity-studio.

To help developers move rapidly from design idea to final product, the Happy Gecko family is supported by the ARM mbed ecosystem, which includes new power management APIs developed by Silicon Labs and ARM.

The Happy Gecko family includes 20 MCU devices providing an array of memory, package and peripheral options, as well as pin and software compatibility with Silicon Labs's entire EFM32 MCU portfolio. Samples and production quantities of Happy Gecko MCUs are available now in 24-pin and 32-pin QFN, 48-pin QFP and 3 mm × 2.9 mm CSP packages. Happy Gecko MCU pricing in 10,000-unit quantities begins at $0.83. The Happy Gecko SLSTK3400A starter kit costs $29.

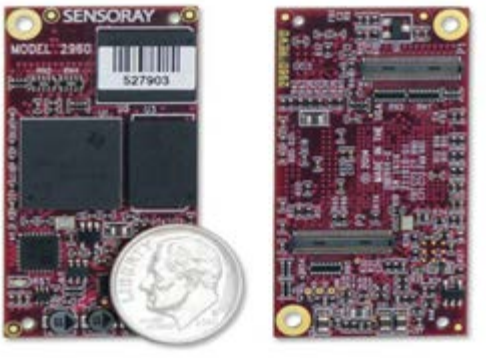
Silicon Labs | www.silabs.com

ULTRA-COMPACT AUDIO/VIDEO PROCESSING ENGINE

Sensoray's 2960 Dragon is an ultra-compact lightweight board with extreme computing power and flexible architecture for HD/SD video and audio processing systems. Using the 2960 Dragon as a building block, Sensoray experts collaborate with customers to create complex custom-configured audiovisual processing systems with extremely short development times.

The 2960 Dragon is only 1.9″ × 1″ and weighs in at 0.22 oz (6.2 g). It captures up to 1920 × 1200 video at 30 fps and JPEG snapshots at up to 4096 × 3104. Packed onto its tiny footprint is a powerful, highly flexible, and configurable audio/video processing engine. The 2960 Dragon features a controller and stream router, SD card interface, six GPIOs, USB (device or host mode), Ethernet, serial COM, and I2C communication interfaces. It is also equipped with one input and one output for HD/SD digital video, a stereo digital audio input and output, and a composite NTSC/PAL output. If the USB interface operates in device mode, the board can be completely powered from USB. Sensoray develops fully customized solutions by designing firmware and integrating it with a carrier board that holds the 2960 Dragon, the connectors, and any other circuitry specified.

Sensoray | www.sensoray.com



60-V LED DRIVER WITH INTERNAL 4-1 SWITCH & PWM GENERATOR

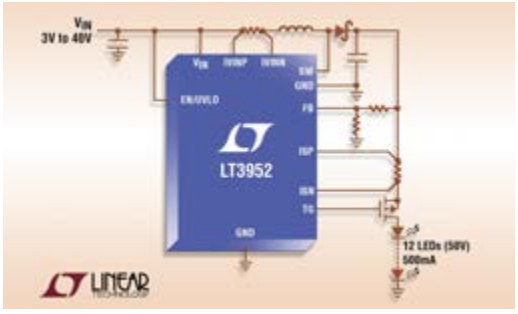
Linear Technology's LT3952 is a current mode step-up DC/DC converter with an internal 60-V, 4-A DMOS power switch. It is specifically designed to drive high power LEDs in multiple configurations. It combines input and output current regulation loops with output voltage regulation to operate as a flexible current/voltage source. The LT3952's 3-to-42-V input voltage range makes it ideal for a wide variety of applications, including automotive, industrial, and architectural lighting.

The LT3952 can drive up to 16 350-mA white LEDs from a nominal 12-V input, delivering in excess of 15 W. It incorporates a high side current sense, enabling its use in boost mode, buck mode, buck-boost mode or SEPIC topologies. Internal spread spectrum frequency modulation minimizes EMI concerns. The LT3952 delivers efficiencies of over 94% in the boost topology, eliminating the need for external heat sinking, and internal LED short-circuit protection enables added reliability required in most applications. A frequency adjust pin permits the user to program the switching frequency between 200 kHz and 3 MHz, optimizing efficiency while minimizing external component size and cost. The LT3952 delivers over 90% efficiency while switching at 2 MHz in a tiny solution footprint. The LT3952 provides a very compact high power LED driver solution in a thermally enhanced TSSOP-28E package.

The LT3952 has a gate driver for a PMOS LED disconnect switch, delivering dimming ratios of up to 4,000:1 using an external PWM signal. For less demanding dimming requirements, the CTRL pin can be used to offer a 10:1 analog dimming range and an internal PWM generator can be used for 5:1 dimming. The LT3952's fixed frequency, current-mode architecture offers stable operation over a wide range of supply and output voltages. Output short-circuit protection and open LED protection enhance system reliability. Other features include frequency synchronization, spread spectrum frequency modulation, programmable VIN undervoltage and overvoltage protection, and an input current limit and monitor.

The LT3952EFE is available in a thermally enhanced 28-lead TSSOP package. Three temperature grades are available, with operation from –40°C to 125°C (junction) for the extended, and industrial grades, and a high temperature grade of –40°C to 150°C. Pricing starts at $3.95 each in 1,000-piece quantities and all versions are available from stock.

Linear Technology | www.linear.com



PRODUCT NEWS

NEW USB3.0 SMART HUB FAMILY

Microchip Technology's USB5734/44 USB3.0 Smart Hub family enables host and device port swapping, I/O bridging, and other serial communication interfaces. The USB5734 and USB5744 devices feature an integrated microcontroller that creates new functionality for USB hubs while lowering overall BOM costs and reducing software complexity.

The new USB3.0 Smart hubs enable an upstream host controller to communicate to numerous types of external peripherals beyond the USB connection through direct bridging from USB to I2C, SPI, UART, and GPIO interfaces. This eliminates the need for an additional external microcontroller, while providing improved control from the USB host hardware.

Microchip's FlexConnect technology enables the USB5734 Smart Hub to dynamically swap between a USB host and a USB device through hardware or software system commands giving the new USB host access to downstream resources. The FlexConnect technology can also switch common downstream resources between two different USB hosts. Incorporating FlexConnect into a system simplifies the overall software requirements of the primary host, as class drivers and application software stay local to the Device-turned-Host.

Available 56-pin, 7 x 7 mm package, the USB5744 is the industry's smallest USB3.0 Hub for applications where board space is important. You can use the USB5734 and USB5744 USB3.0 controller hubs for a variety of applications (e.g., computing, embedded, medical, industrial, and networking markets).

The USB5734 and USB5744 are supported by Microchip's $399 USB 3.0 Controller Hub Evaluation Board (EVB-USB5734) and $299 USB 3.0 Small Form Factor Controller Hub Evaluation Board (EVB-USB5744). The former includes mezzanine cards that can be used as preset application configurations for easy testing and development of a USB5734 system.

The USB5734 is available in 64-pin QFN (9 × 9 mm) packages starting at $4.20 each in 10,000-unit quantities. The USB5744 is available in 56-pin QFN (7 × 7 mm) packages starting at $3.75 each in 10,000-unit quantities.

Microchip Technology | www.microchip.com

COST-EFFECTIVE, LONG-RANGE, LOW-POWER IoT CONNECTIVITY

SIGFOX and Texas Instruments are now working together to increase Internet of Things (IoT) deployments using the Sub-1 GHz spectrum. Customers can use the SIGFOX network with TI's Sub-1 GHz RF transceivers to deploy wireless sensor nodes that are lower cost and lower power than 3G/cellular connected nodes, while providing long-range connectivity to the IoT.

Targeting a wide variety of applications ranging from environmental sensors to asset tracking, the SIGFOX and TI collaboration maximizes the benefits of narrowband radio technology. It also reduces barriers to entry for manufacturers interested in connecting their products to the cloud. Using the SIGFOX infrastructure reduces the cost and effort to get sensor data to the cloud and TI's Sub-1 GHz technology provides years of battery life for less maintenance and up to 100 km range.

SIGFOX's two-way network is based on an ultra-narrowband (UNB) radio technology for connecting devices, which is key to providing a scalable, high-capacity network with very low energy consumption and unmatched spectral efficiency. That is essential in a network that will handle billions of messages daily.

TI's CC1120 Sub-1 GHz RF transceiver uses narrowband technology to deliver the longest-range connectivity and superior coexistence to SIGFOX's network with strong tolerance of interference. Narrowband is the de facto standard for long-range communication due to the high spectral efficiency, which is critical to support the projected high growth of connected IoT applications. The CC1120 RF transceiver also provides years of battery lifetime for a sensor node, which reduces maintenance and lowers the cost of ownership for end users.

Sub-1 GHz networks operate in region-specific industrial scientific and medical (ISM) bands below 1 GHz including 169, 315, 433, 500, 868, 915 and 920 MHz. The networks are proprietary by nature and provide a more robust IoT connection, which is why the technology has been used for smart metering, security and alarm systems and other sensitive industrial systems. Additionally, the technology is low power, enabling years of battery life to reduce service and maintenance requirements.

SIGFOX-certified modules based on TI's CC1120 were demonstrated at Mobile World Congress 2015 and are currently available.

SIGFOX | www.sigfox.com
Texas Instruments | www.ti.com

CLIENT PROFILE

Jameco Electronics

Location: Belmont, CA (USA)
Web: www.jameco.com
Contact: Angela Rolls
Email: arolls@jameco.com

EMBEDDED PRODUCTS

Jameco Electronics has been a leading electronics component distributor for over 40 years and offers businesses, educational institutions, and hobbyists alike more than 50,000 of the industry's most popular name brand components in addition to a large selection of in-stock house and generic brands.

WHY SHOULD CC READERS BE INTERESTED?

Jameco's product categories range from power to testing to security to robotics and more. Jameco is not only a great source for hard to find electronic components, Jameco offers more electronics kits than can be found anywhere else. Club Jameco, a hobbyist driven DIY electronics projects site, features kits for all age and skill levels created by designers and electronics enthusiasts from around the world. Designers earn a royalty on every kit sold. Jameco also provides a variety of free content ranging from DIY videos to tech tips to product reviews including the Great American Electronics Hobbyist Census—a study uncovering surprising results regarding electronics hobbyists across the United States.

Jameco is committed to exceeding industry standards and only sells high-quality, reliable products. All Jameco branded products are backed with a 30-day money back guarantee and a 90-day warranty. More than 99% of Jameco's catalog items are same-day ready to ship and Jameco's in-house technical support provides expertise to help troubleshoot and answer questions.

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

FEATURES

Sound Ecology and Acoustic Health (Part 1)

Develop a Basic Android Application

Is the urban noise nuisance you hear at home a community problem or a personal issue? Perhaps it's your house's poltergeist humming to announce its displeasure with the new exterior paint colors? Build a "Without Any Teenager Assistance Necessary" Android app (WAT_AN_APP) and find out!

By Adrien Gaspard, Mike Smith, and Nicholas Lepine (Canada)

A background level of urban noise is, unfortunately, one of those inconveniences that we must put up with to live in a community. You and your neighbors can sympathize with each other at a BBQ about being awakened at dawn by a chorus of birds or early morning commuter traffic noise. You can band together to solve that problem with a joint big win on the local lottery so you can buy new houses elsewhere.

Sometimes, the noise can be such an issue across a wide community that even federal politicians can be stirred from their comfortable seats to assist in the investigation as in the case of the Windsor Hum (www.zugislanddocumentary.com). However, the BBQ conversation might go along these lines. You say: "That noise at 2 AM every morning last week … Hasn't it been driving you crazy?" Your neighbor responds: "What noise?"

After such a conversation, it is worthwhile getting your hearing tested. There are conditions affecting the ears, such as Tinnitus or "ringing in the ears," which can be alleviated with professional help. However, with that issue ruled out, you now have a noise in your house that nobody else in your

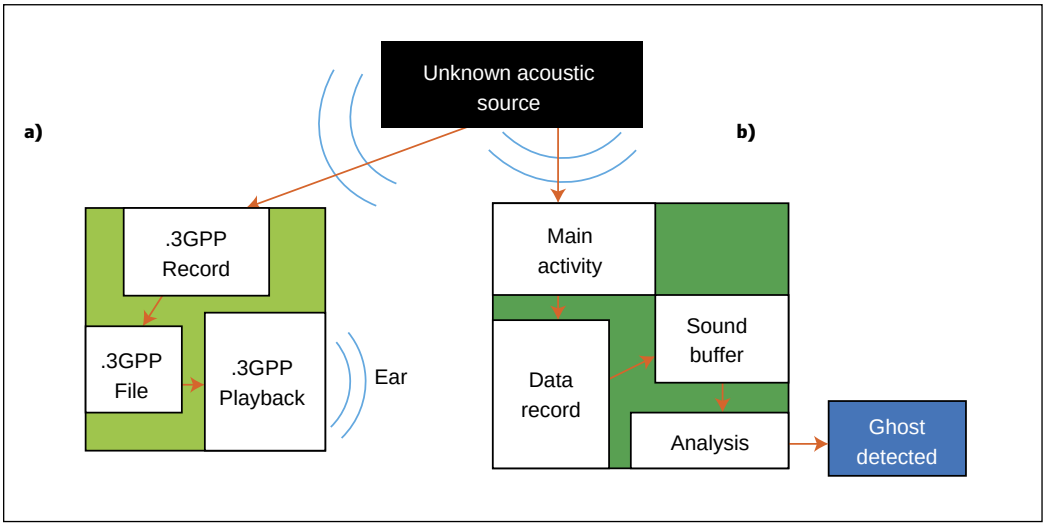


FIGURE 1
The Android WAT_AN_APP for **(a)** audio .3GPP record/playback and **(b)** data record/ghost analysis

neighborhood seems to have. It's time to become a noise detective.

There are apps to record your prowess at "singing in the shower" and those would be useful (to a limited degree) for capturing the noise. If your neighbor agrees, check his house or garden. Everybody's hearing is different. It is possible that your neighbor isn't sensitive to the noise you hear.

Recording is good. It confirms that something physical is present in several locations. However, we wanted a little more than that. While gathering information on these topics, we consulted local acoustics firms and learned that "home resonances" can contribute to situations in which you hear noise and your neighbors don't. We have all seen neighborhood kids playing with resonances. For instance, by making tiny, gentle leg kicks at just the right time, you can cause a playground swing to reach great heights. In a similar way, parts of your house (e.g., a wall, basement concrete pad, or a heating duct) can be constructed in such a way that they can vibrate with, and amplify, some tiny, gentle, totally insignificant incoming vibration through the air or ground.

To analyze the frequency response of a house including resonances, we needed a recording app with a number of custom DSP extensions for resonance identification. Rather than deal with the hassle of accessing and then modifying an existing app's source code, we decided on a more pragmatic approach. We figured that if the local teenagers can build their own Android apps, we surely could too! We set out to build our own recording app from online Android tutorials (http://developer.android.com/training/index.html).

Figure 1a outlines the plan for our first Android project. Our goals were to be able to record and play back an audio .3GPP file using a modified Android MediaRecorder app and prove the existence of noise problems we could hear. We also wanted to use our own DSP experience from work to add special analysis features. In future articles, we will explain how to use a different Android activity approach to capture sound in an uncompressed data buffer.

Figure 1b demonstrates the use of DSP "spectral" analysis to identify an unwelcome "spectral" presence—a ghost. As you'll see, we can extend the app to perform a more sophisticated analysis that uses graphics to display home resonances that might be impacting your home's acoustic health.

COMMUNITY PROBLEM?

Alright, we must now provide an answer to the following question: "What DSP app features do you and your neighbors need to help identify any poltergeists in your community?"

Actually, the proposed "house resonance identifier" WAT_AN_APP should be sufficient. Some hauntings are associated with wind-induced resonances in a house structure according to the literature. The strong vibrations in the air produced by resonance reduce the local temperature. This lowers the dew point, causing water condensation to occur in the anti-nodes of the resonance, producing a white ghost cloud. If this explanation is not true, then at least we will have shown you how to build an app to compare screams as you and your neighbors discretely leave the area!

If you are going to get serious about becoming a noise detective, grab an app—something equivalent to Google Calendar perhaps—and start taking notes when you are bothered by the noise. A list can help you see if there is a time pattern—say, the annoyance always starts around 2:30 AM. That information could be a possible lead to matching your noise nuisance to something unintentionally generated by a local firm. Remember that if the local ground structure in your area is just so, the sound (especially low-frequency sounds) can travel many kilometers before entering your house and inducing a resonance.

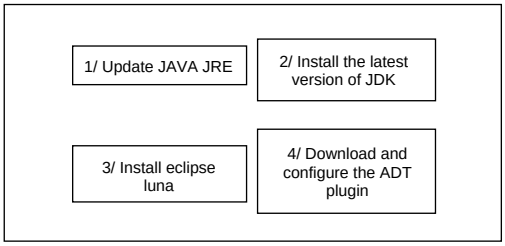


FIGURE 2
The installation steps for the Android WAT_AN environment

WAT_AN_APP DEV ENVIRONMENT

If you don't have a "tame" teenager in the house already developing Android apps, refer to **Figure 2** for the installation steps for the environment needed to develop, debug, and

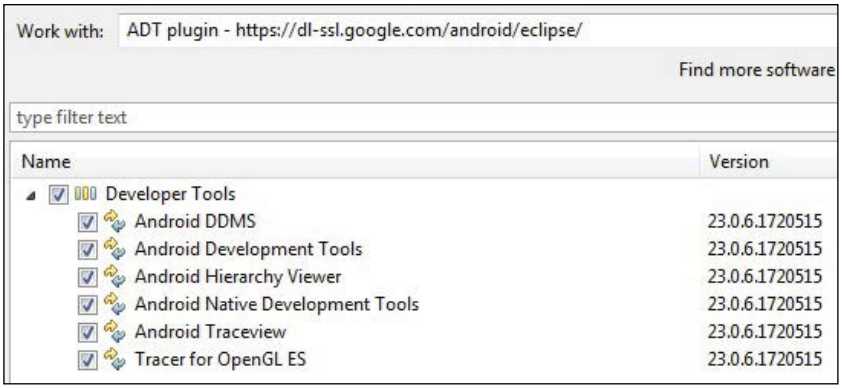


FIGURE 3
Selecting the ADT plugins needed for Eclipse Android development

run the code for our Android WAT_AN_APP application. To get the latest version of Java Runtime Environment (JRE) running on your system, go to www.oracle.com/technetwork/java/index.html. As JRE is only the Java runtime environment, it does not contain the definitions required to build a new Java file. You need the Java Developer Kit (JDK), which is available at www.oracle.com/technetwork/java/index.html.

To get started we read the Eclipse plugin tutorial at developer.android.com/sdk/installing/installing-adt.html. The Eclipse Luna IDE is available at www.eclipse.org/downloads/. Choose Eclipse IDE for Java Developers for your operating system (e.g., Windows 64 Bit). Click on one of the mirror sites to download the Eclipse .zip file. Unzip the file into the destination path C:\. Start Eclipse by clicking on eclipse.exe in the new C:|eclipse folder. Click Run when the standard install security warning pops up. You will then be asked to choose a default workspace. We placed the WAT_AN_APP folder in Documents (e.g., C:\Users\Documents\WAT_AN_APP). A few more steps are needed before Eclipse is ready for use.

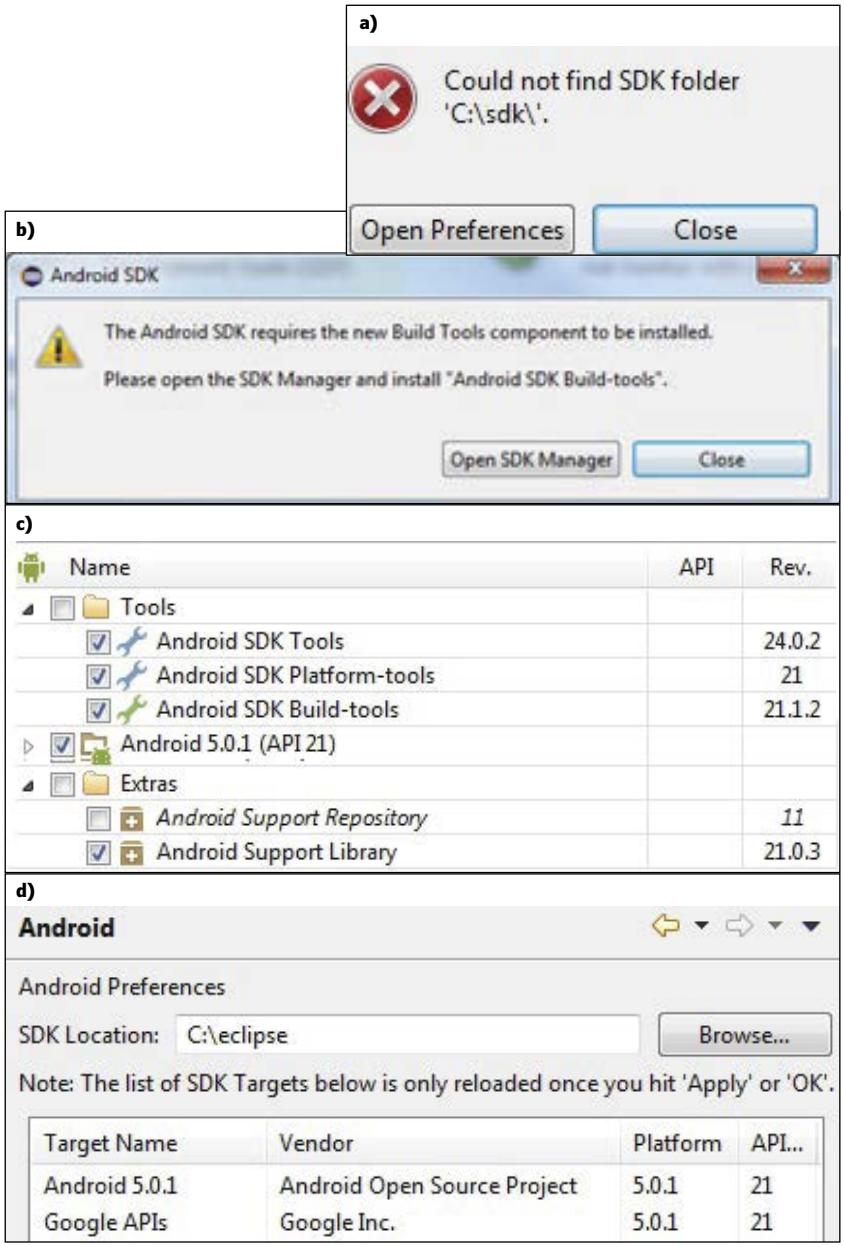


FIGURE 4
The following messages and data appear after you restart Eclipse: **(a)** Android SDK error: could not find the SDK folder; **(b)** Android SDK warning: the SDK required components must be installed; **(c)** the required Android SDK Tools, API and library; and **(d)** a screen dump of the Android environment needed for WAT_AN_APP development.

You now need to install Android Development Tools (ADT) plugins for the Eclipse IDE listed in **Figure 3**. These plugins extend Eclipse's capabilities so you can set up new Android projects, create Android applications, or debug them using the Software Development Kit (SDK) tools.

To add the ADT plugin, select Help from Eclipse toolbar then Install New Software. After clicking Add, enter the name "ADT plugin" and the following URL for the location: https://dl-ssl.google.com/android/eclipse/. As you can see in Figure 3, check the Developer Tools box, click Next, and then click Next again to install the six new items. Accept the license agreement and click Finish. The software installation process then begins. Eclipse always seems to need something else from the web, so when asked, allow communication access to the domain networks. Restart Eclipse when prompted, select the default workspace, and receive your reward—the fatal Android Software Development Kit (SDK) error window stating that the SDK folder could not be found (see **Figure 4a**). Adding this kit gives access to the Application Programming Interface (API) libraries as well as the developer tools required to build, test, and debug applications.

To install the SDK tool, close Eclipse, go to developer.android.com/sdk/index.html#Other, accept the terms and conditions, and then download installer_r24.2-windows.exe. Double click on it to start the installation. When asked, choose C:\eclipse\ as an installation path. Restarting Eclipse generates a new complaint (see Figure 4b). You need to install the SDK build tools required for building Android application code. Click on Eclipse's Open SDK Manager menu option to bring up the options shown in Figure 4c. Select to install the Android SDK Tools, the Android SDK Platform-Tools, the highest version of the Android SDK Build-tools (currently 21.1.2), and the Android support Repository and Library package. Accept the license agreements, and go for a coffee since

it takes a while for all the packages to install. Once all the packages are downloaded and installed correctly, take a deep breath, cross your fingers, and restart Eclipse,

This time, no error messages should be displayed. There are still a couple of potential pot holes that are nice to avoid. Select Eclipse's menu item Windows | Preferences and verify that the SDK directory's location is C:\eclipse (see Figure 4d). If you plan to put your WAT_AN_APP on a phone with an older version of Android (before 5.0), watch out for compatibility issues. For example, for an Android phone running KitKat, you also need to install all the packages included in the folder Android 4.4.2 (API 19).The Quick Help Guide posted on the *Circuit Cellar* FTP site will help you determine if this step is necessary. Once the Android Developer Tool and the Android Software Development Kit are finally installed and configured, you can start creating your first Android project. Poltergeists beware!

WAT_AN_APP PROJECT

We wanted our first WAT_AN_APP application to start by displaying a congratulations message on a screen. There has to be a button to press to start "A something." This something is the empty AudioRecord activity in **Figures 5a** and Figure 5b. This is essentially an Android Hello-World equivalent to allow us to get familiar with the developing environment, learn an easy way to display text and a button using a screen layout file, and to implement a response to the action of clicking on the button. All of this done without a teenager in sight!

To start building the application, we selected File | New | Project from the Eclipse toolbar to create a new Android Application Project (see **Figure 6a**). Fill in the blanks for the application name, WAT_AN_APP, the project name, WAT_AN_APP, and the package name com.wat_an_app (see Figure 6b). This is the time to take another look at the Quick Help Guide posted on the *Circuit Cellar* FTP site to determine the version of Android running on your device and be able to answer the question "Target SDK"?

If your phone has Android version 5.0.1 installed, respond to the questions "Target SDK" and "Compile with" by selecting the current highest API (API 21). If your device runs the previous version of Android, KitKat (4.4.x), select API 19. If you want the adventure of hunting haunting noisy ghosts using an Android wearable smartwatch, you need to select API 20. As you can see in Figure 6b, the "Minimum Required SDK" should not be set to an API lower than 9 to avoid introducing incompatibility problems

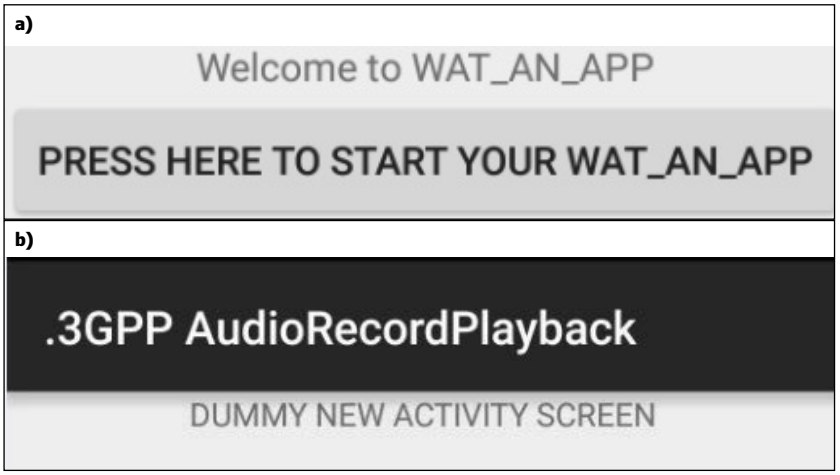


FIGURE 5
The WAT_AN_APP has a button waiting to be pressed. This is the "A Something" activity screen **(b)**.

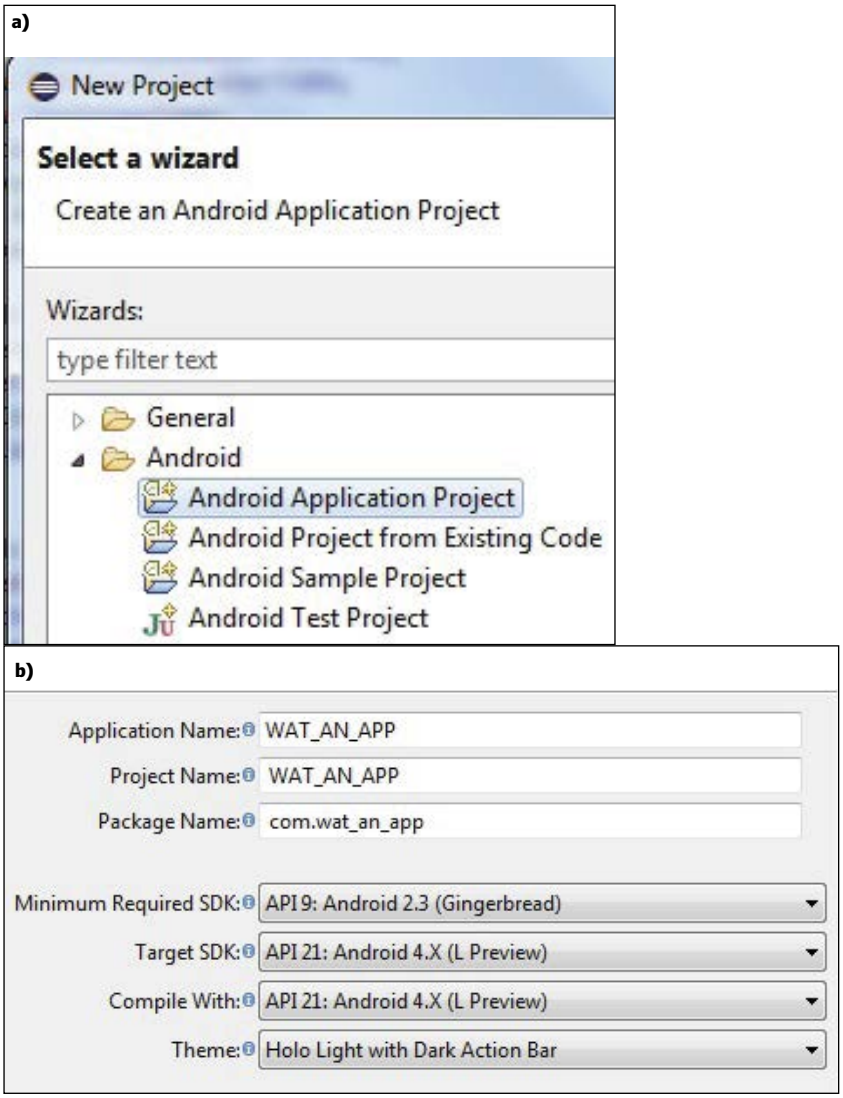
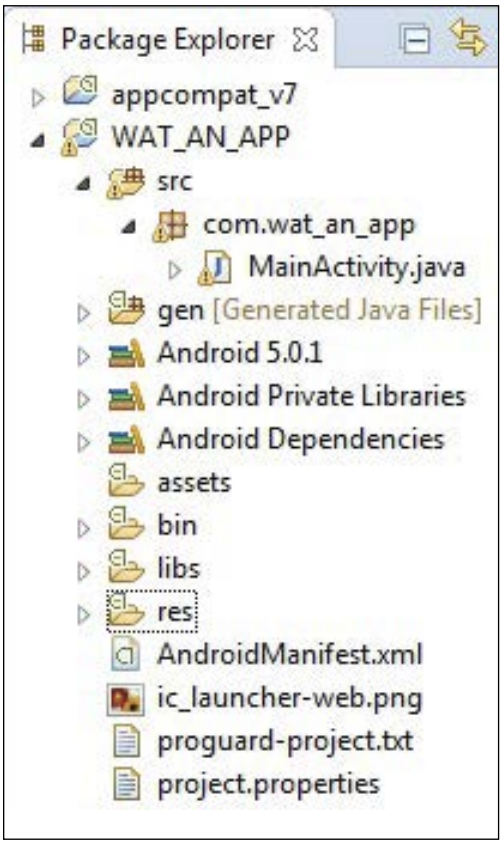


FIGURE 6
Here we are **(a)** creating a new Android application project and **(b)** creating the WAT_AN_APP project.

FIGURE 7
Project's package explorer window




```
1. package com.wat_an_app;
2. import android.app.Activity;
3. import android.content.Intent;
4. import android.os.Bundle;
5. import android.view.View;

  // Cause display of MainActivity screen layout (Listing 2)
10. public class MainActivity extends Activity {
11. @Override
12.   protected void onCreate(Bundle savedInstanceState){
13.      super.onCreate(savedInstanceState);
14.      setContentView(R.layout.activity_main);
15.  }

      //NEW ACTIVITY
      //Call the AudioRecordPlayback activity when the user
       //presses the button on the main screen
20. public void AudioRecordPlayback(View v) {
21.   Intent beginAudioRecordPlayback
22.      = new Intent(this, AudioRecordPlayback.class);
22.   startActivity(beginAudioRecordPlayback);
23.  }
24. }
```

LISTING 1
The modified MainActivity.java (WAT_AN_APP\src\ folder) generates a welcome message and uses a button to activate the AudioRecordPlayback activity.

with our code. The "Theme" specifies your app's user interface (UI) style. Try "Holo Light with Dark Action Bar" before clicking Next.

The window that appears configures the project. Make sure that the boxes "Create custom launcher icon", "Create activity," and "Create Project in Workspace" are checked before clicking on Next.

Each Android application requires an icon for the user to press and start the application from the phone's main menu. The next window is used to configure the attributes of this icon. Simply click Next to go environmentally friendly, as in using the green Android Bugdroid for the application icon.

It is now time to tell to Eclipse that we want to create an activity that forms your app. Each application is composed of a main activity launched as soon as the application's icon is pressed. The main activity sets the first main menu screen to appear, and then starts other activities, such as recording audio. Check the Create Activity box and then select a template. We recommend starting with an EmptyActivity, given all the issues we have had with the other templates not generating the activity code the way we were expecting. Click Next and get an opportunity to give names to your activity and its layout. Keep the default activity name, MainActivity, and layout name, activity_main, to remain consistent with our project pictures.

After so much activity, you can finally press Finish and relax: your WAT_AN_APP Android new application's project containing a main activity is being created! Once created, there is a Package Explorer window in Eclipse that contains the project with the main activity in the WAT_AN_APP\src\com.wat_an_app folder (see **Figure 7**). A library supporting the user interface implementation is in the appcompat_v7 folder. To learn more about the support features in the different Android libraries, go to https://developer.android.com/tools/support-library/features.html.

OUR FIRST WAT_AN_APP

We now need to update the default project code to implement a GUI so we can interact with the application via visual indicators. The default code in the MainActivity.java file in the WAT_AN_APP\src\ folder needs to be replaced with the Lines 1 to 15 from **Listing 1**. Lines 20 to 24 describe a new activity, `AudioRecordPlayback`, which was not automatically generated when the project was built. Line 21 discusses an Android `Intent`, which is an object used to

start another activity. The intent object makes use of the `AudioRecordPlayback` class when starting the new AudioRecordPlayback activity.

The appearance of the MainActivity screen is controlled by the activity_main.xml layout file automatically added into the WAT_AN_APP\res\layout folder. The default layout file contains a hello_world `View` object (TextView widget) source code to provide an immediate runnable test of any new Android project. That default code needs to be replaced by the **Listing 2** layout code.

You can build an Android activity screen using a linear or relative layout tag description. A linear layout would automatically place the text and the button horizontally or vertically next to each other. A relative layout (see Listing 2) is a `ViewGroup` that displays the widgets using `layout_CenterVertical` and `layout_CenterHorizontal` parameters to specify exactly where message `TextView` (Lines 110 to 117) and `Button` (Lines 120 to 129) should be displayed on the screen. The IDs are needed to describe the `TextView` (Line 111) and `Button` (Line 121) tags to be called from our Android Activities.

Details of the text strings to be displayed in the main screen message (Line 116), and on the button (Line 127) need to be stored with other preset string values in the strings.xml file in the WAT_AN_APP\res\values folder (see **Listing 3**, lines 206 to 208). The `layout_width` and `layout_height` parameters control how these strings are displayed.

Last, but definitely not least, the `onClick` information (Line 128) allows the Button tag to start the AudioRecordPlayBack

```
100. <RelativeLayout
101. xmlns:android="http://schemas.android.com/apk/res/android"
102. xmlns:tools="http://schemas.android.com/tools"
103. android:layout_width="match_parent"
104. android:layout_height="match_parent"
105. tools:context="${relativePackage}.${activityClass}"   >

        <!-- Main Screen Text -->
110.    <TextView
111.      android:id="@+id/main_text"
112.      android:layout_width="wrap_content"
113.      android:layout_height="wrap_content"
114.      android:layout_centerVertical="true"
115.      android:layout_centerHorizontal="true"
116.      android:text="@string/introduction_main_text"
117.    />

        <!--Button to activate AudioRecordPlayback -->
120.    <Button
121.      android:id="@+id/start_WAT_AN_APP"
122.      android:layout_width="wrap_content"
123.      android:layout_height="wrap_content"
124.      android:layout_centerVertical="true"
125.      android:layout_centerHorizontal="true"
126.      android:layout_below="@id/main_text"
127.      android:text="@string/press_to_start_WAT_AN_APP"
128.      android:onClick="AudioRecordPlayback"
129.    />
          <!--More code from other articles -->
199. </RelativeLayout>
```

LISTING 2

The modified activity_main.xml layout file (WAT_AN_APP\res\layout folder) details `<Button />` and message `<TextView />` tags. " To reduce the typing, you can always cut and paste code from the electronic edition of this article if you remember to remove the line numbers. In the next article we will give the code for a JAVA applet to remove the numbers and extra lines automatically.

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.

Nicholas Lepine (nnlepine@ucalgary.ca) is an undergraduate student at the University of Calgary studying Electrical and Computer Engineering. In May 2015, Nicholas interned at the telecommunications company NTT. He intends to work in the embedded systems industry after graduation.

LISTING 3
Preset string values must be set in strings.xml file (WAT_AN_APP\res\values folder).

```
200. <?xml version="1.0" encoding="utf-8"?>
201. <resources>
205.     <!-- Strings required main screen message and button -->
206.     <string name="app_name">WAT_AN_APP</string>
207.     <string name= "introduction_main_text">
               Welcome to WAT_AN_APP</string>
208.     <string name="press_to_start_WAT_AN_APP">
               Press here to start your WAT_AN_APP</string>

210.      <!-- Strings for second screen layout - display text
           and button to start empty activity -->
211.      <string name="introduction_audio_record_playback_text">
212.          DUMMY NEW ACTIVITY SCREEN </string>
213.      <string name="title_activity_audio_record_playback">
214.          .3GPP AudioRecordPlayback </string>
          <!-- Other code to come from future articles -->
249. </resources>
```

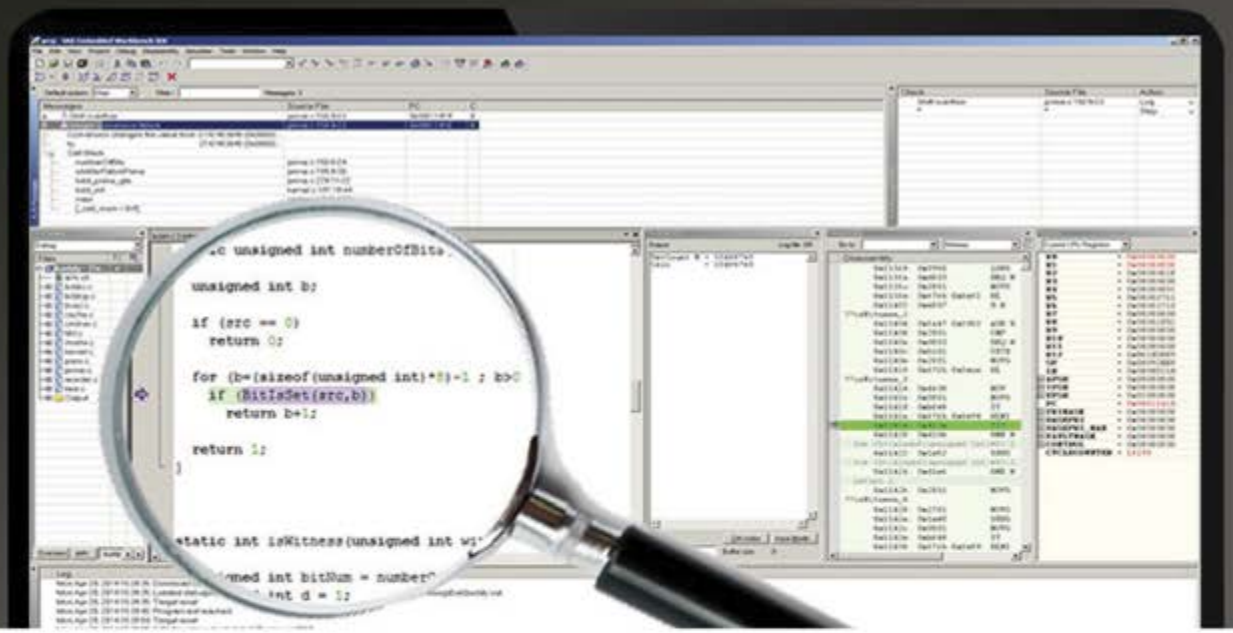
LISTING 4
The new activity AudioRecordPlayback layout setup coded in AudioRecordPlayback.java (WAT_AN_APP\src\ folder

```
300. package com.wat_an_app;
301. import android.os.Bundle;
302. import android.support.v7.app.ActionBarActivity;

305. public class AudioRecordPlayback extends ActionBarActivity {
306.  @Override
307.  protected void onCreate(Bundle savedInstanceState) {
308.   super.onCreate(savedInstanceState);
309.   setContentView(R.layout.activity_audio_record_playback);
310.   }
311. }
```

LISTING 5
Layout details used in the activity_audio_record_playback.xml file (WAT_AN_APP\res\layout folder) to display a basic message

```
     <!-- Used by AudioRecordPlayback -->
400. <RelativeLayout
401.     xmlns:android="http://schemas.android.com/apk/res/android"
402.     xmlns:tools="http://schemas.android.com/tools"
403.     android:layout_width="match_parent"
404.     android:layout_height="match_parent"
405.     tools:context="${relativePackage}.${activityClass}" >

     <!-- Dummy New Activity Screen Text -->
410. <TextView
411.     android:id="@+id/audio_recordplayback_text"
412.     android:layout_width="wrap_content"
413.     android:layout_height="wrap_content"
414.     android:layout_centerVertical="true"
415.     android:layout_centerHorizontal="true"
416.     android:text="@string/introduction audio
           _record_playback_text"
417. />

499. </RelativeLayout>
```

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

IAR SYSTEMS

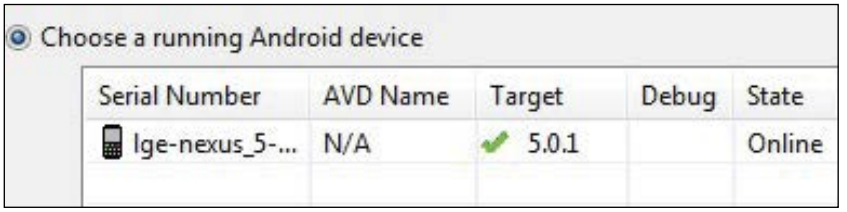
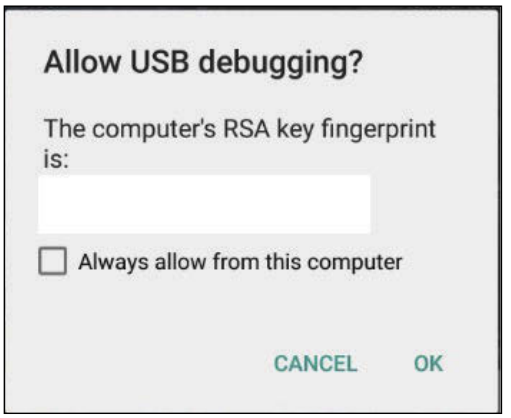


FIGURE 8
Selecting the device to run the application

FIGURE 9
Allow USB debugging from your computer to your phone



activity. For more layout info, visit developer.android.com/guide/topics/ui/declaring-layout.html.

AUDIORECORDPLAYBACK

If you download the code from Listings 1 to 3 to the phone, you would crash as there is no activity to be activated when you press the main screen button. This needs to be fixed by creating a new activity. Press New in Eclipse's toolbar, open the Android folder, and select Android Activity. As previously done for the main activity generation, select the EmptyActivity and give your activity a name, AudioRecordPlayback.

The default AudioRecordPlayback.java file in the WAT_AN_APP\src folder currently contains the minimum code to display the content described in its layout file (see **Listing 4**, Lines 300 to 311). Update the default layout activity_audio_record_playback.xml template file (WAT_AN_APP\res\layout folder) with the code in **Listing 5**, lines 400 to 417.

RUNNING WAT_AN_APP

Some issues might appear while trying to run the application for the first time. If your Android device has never been connected to the computer before, you might need to download its drivers. You also need to enable USB debugging on your device. Please refer to the Quick Help document on the *Circuit Cellar* FTP site for more information.

To run the application, select the project in Eclipse and click on the Run button in the toolbar. Run the WAT_AN_APP project as an Android Application. The Choose a Running Android Device window will pop up (see **Figure 8**). Pressing Start enables the app to run on the phone. You are given the opportunity to activate the LogCat. This useful tool collects and displays the system debug output in the unlikely situation you make a mistake. For the moment, click on: Yes, monitor logcat and view the logcat window to see if there are messages with priority higher than warning. We will detail the advantages of using the LogCat in the next part of this article series.

You might receive a message asking if you want to allow USB debugging from your computer (see **Figure 9**). Press OK to have the application starting to run on your device.

Finally, the MainActivity screen pops up. Pressing the button starts the dummy AudioRecordPlayback activity. It only contains a text at the moment. We will fix that in the next article.

VIRTUAL DEVICE EMULATION

If all of this just generates too much excitement and your pulse is high, then there is a slower running choice that emulates a device using an Android system image. Warning: We have experienced the emulator taking a considerable amount of time to start, so prepare to makes some tea or get healthy with a little bit of in-your-chair yoga.

To install a system image, open the Android SDK manager (see Figure 4c). For emulating a device running on Android Lollipop (API 21), unroll Android 5.0.1 (API 21) and make sure that the ARM EABI v7a System Image system images for the examples here is installed. You then need to create and set up an Android Virtual Device by clicking on the Android Virtual Device Manager panel located in Eclipse's menu bar. Press Create to have a configuration window appear. For this article, set this window as shown in **Figure 10a**. This enables the emulation of a Nexus 5 phone running on Lollipop (API 21). Choose an ARM (armeabi-v7a) for CPU. Set the RAM setting to 500M. Using 768 MB of RAM also works, but anything bigger will cause the Windows emulator to fail when launching. Make sure that the VM Heap is set to 64, as 32 crashes the emulator. Don't be greedy: 200 MB of internal storage is enough for our simulation. With parameters set, press OK and the virtual device MyVirtualDevice will, like magic, appear in the AVD Name row.

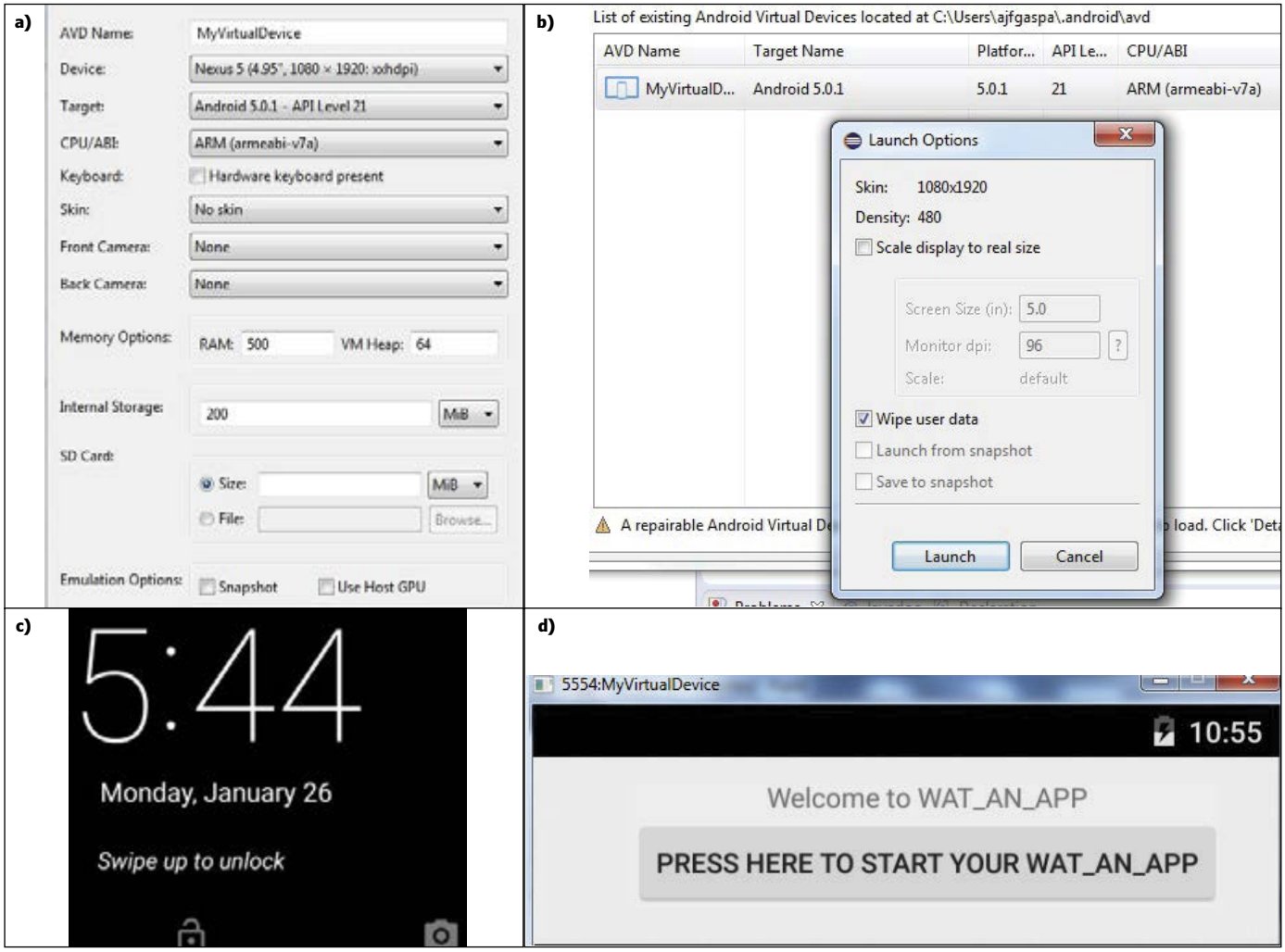


FIGURE 10
Here you see: **(a)** how setting up the emulator, **(b)** launching the emulator, **(C)** the emulator lock screen, and **(d)** emulation of the WAT_AN_APP application.

It is very important to avoid repetitive strain injury during the rest of the virtual device configuration. So, use your left thumb to close this window. Now right click on the WAT_AN_APP project with your right index finger. Using your ring finger, click on Run As and then Android Application.

In the next window, use your right thumb to select Launch a New Android Virtual Device. If this window does not pop up with the application running on your physical device by default, please use the following approach. Using alternate fingers, right click the WAT_AN_APP project and then Run As followed by Run Configurations.

In the new Target window, make sure that Always Prompt to Pick Device is selected. Go back to the Choose a Running Android Device window, click on MyVirtualDevice, and then click on Start.

As you are now more familiar with Eclipse, it goes without saying that you won't be surprised to discover that a new window pops up. You have to set the launch options (see Figure 10b). Check Wipe User Data and press Launch.

When the emulator finishes booting (see Figure 10c), click on the emulated window, swipe up to unlock the screen, and wait for a bit: you should see your application appearing (see Figure 10d). To learn more about using the emulator, go to developer.android.com/tools/devices/emulator.html.

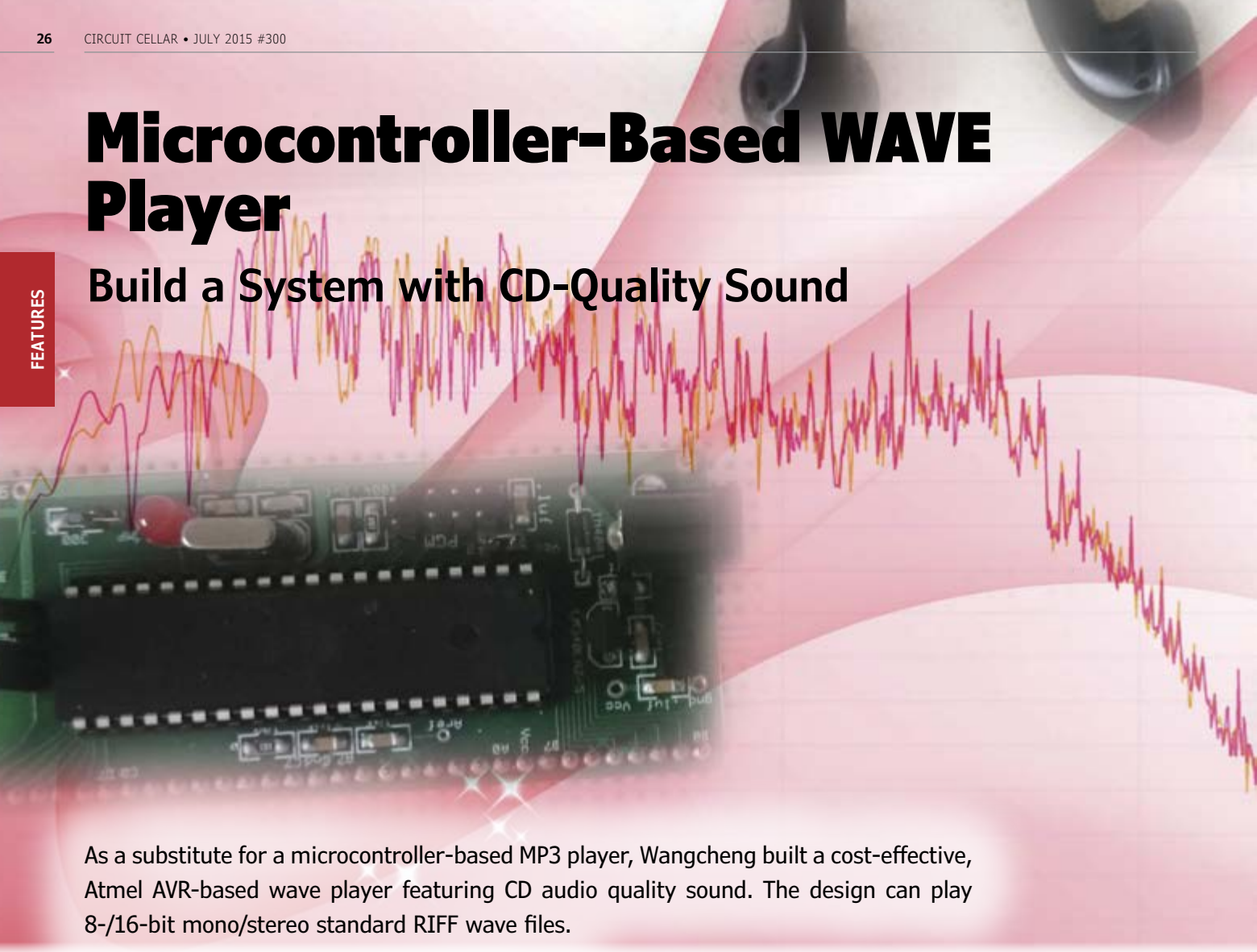
LOOKING AHEAD

In this article, we explained how we developed a basic Android application. In our future articles, we will add the features to record and play-back .3GPP files, analyze recorded signals, and display their spectral analyses. All of these steps are necessary for identifying community noise sources that excite resonances in your home, affecting your audio health and general wellbeing.

FEATURES

Microcontroller-Based WAVE Player

Build a System with CD-Quality Sound



As a substitute for a microcontroller-based MP3 player, Wangcheng built a cost-effective, Atmel AVR-based wave player featuring CD audio quality sound. The design can play 8-/16-bit mono/stereo standard RIFF wave files.

By Wangcheng Zhou (United States)

I built an Atmel ATmega1284P-based wave player with CD audio quality that can play 8-/16-bit mono/stereo standard Resource Interchange File Format (RIFF) wave files (see **Photo 1**). You could incorporate this cost-effective design in a variety of applications, such as bus/subway auto-announcement system or an elevator voice indication system. Most announcing systems are limited by one-time programmable (OTP) voice chips with small capacity (normally using EPROM as storage media), not to mention the relatively expensive price tags. (An OTP voice IC's price is determined by its capacity—voice recording time, normally 10 to 200 s.) For instance, APlus's AP89010 OTP chip costs $0.48 (10-s OTP) and its AP89341 is $2.43 (150-s OTP). One announcing system may use multiple voice chips (two to 10 pcs), which also results in a complex hardware design. On the contrary, a standard SD card is more affordable with the rapid development of storage technology. A 512-MB Kingston SD card is $3.65. Compared to an OTP chip, the SD card's 512 MB of storage can hold as many as 128 songs (MP3 format) and 10 lossless songs (WAV format). The point is that you can easily format an SD card. You can modify songs and recorded files at will, as long as you have a PC supports a standard FAT file system.

As I mentioned above, an announcing system using an OTP chip solution has a complicated circuit. Different audio announcements need to be played from different chips. The control logic is quite involved, which definitely increases the circuit's complexity, including the PCB. In contrast, my system requires only six main wires—SPI bus (four wires) and PWM output channel (two or four wires)—to play the songs or any recorded wave files. Some Atmel AVR chips cost $4 (e.g., ATmega328). So, the entire system can be very cost-effective.

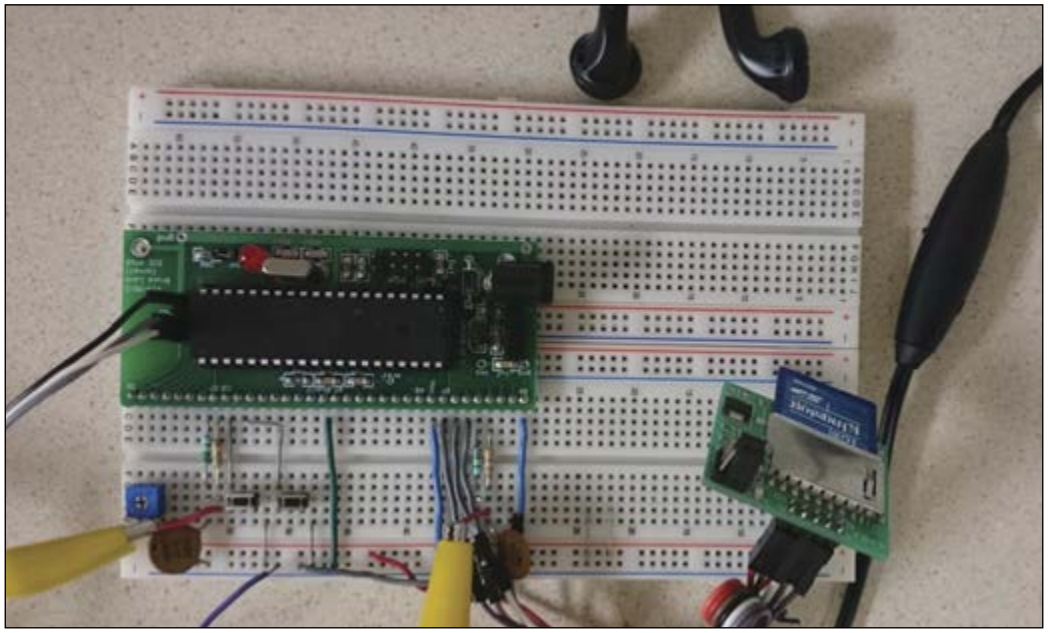


PHOTO 1
The WAVE Player comprises a minimum AVR system board, an SD card module (with level conversion circuit) to store WAVE files, two buttons for Play/Pause control, and several resistors and capacitors for a combination circuit (combine four 8-bit PWM outputs into two analog outputs with 16-bit resolution).

HIGH-LEVEL DESIGN

Most microcontroller-based MP3 player designs require an extra hardware decoder and DAC chip. The result is a complicated circuit. So, how you can to design chip-based music player with acceptable audio quality? Substitute the MP3 decoder using a WAVE file, which has a format simpler than MP3 and a microcontroller can easily decode. To substitute the DAC chip, you can use PWM to replace it with simple RC filter, such as the Cricket Call Generator. Ideas like these led me to design my AVR-based WAVE player.

Other than SDIO, the only way to communicate with the SD card is via a serial peripheral interface (SPI). The ATmega1284P has a hardware SPI interface with four working modes (see **Figure 1**). However, to operate with SD card with SPI mode, you must determine one of the four working modes. The official SD card physical layer specification doesn't contain a detailed timing diagram of the supported SPI, but additional details about SPI timing are revealed in SanDisk's 2004 product manual, "SanDisk SD Card" (Version 2.2).

As you can see in **Figure 2**, you should configure SPI working mode to "Mode 0" (Positive Pulse, Latch Data then Shift) to correctly operate the SD card. Meanwhile, instead of reading RAW data from the SD card, the FAT16/32 file system is implemented so WAVE files can be easily stored in the SD card via any PC with an SD card reader socket. The FAT file system is open-source Petit FatFs, which is a simpler version of the popular FatFs developed by ChaN (see **Figure 3**).

Note that the WAVE file format is actually a subset of RIFF format developed by Microsoft. I'll detail the RIFF audio format later in this article.

SYSTEM STRUCTURE & HARDWARE

At a high level, the system structure is pretty simple. It contains a Storage Module

FIGURE 1
The Atmel AVR's SPI working modes

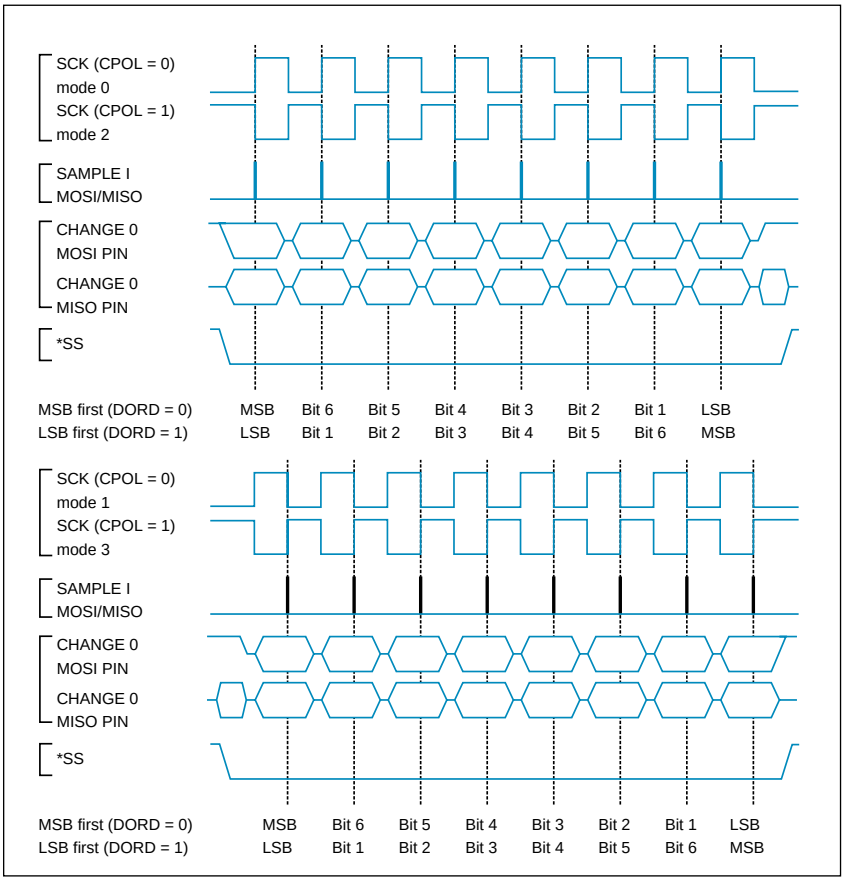


FIGURE 2
SD Card SPI timing

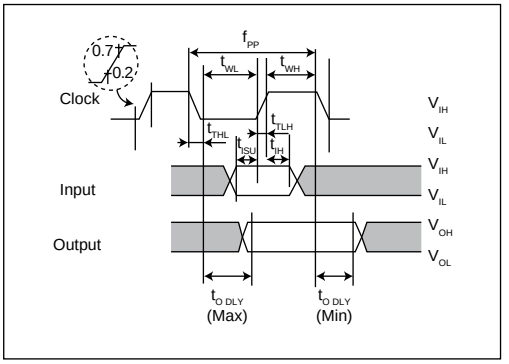
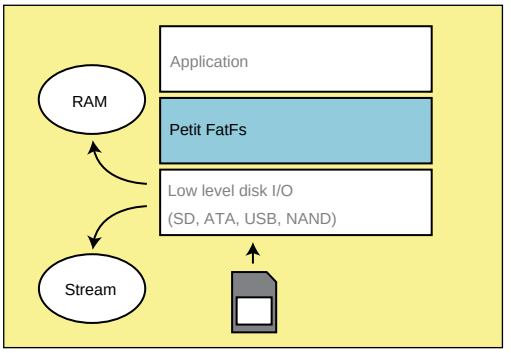
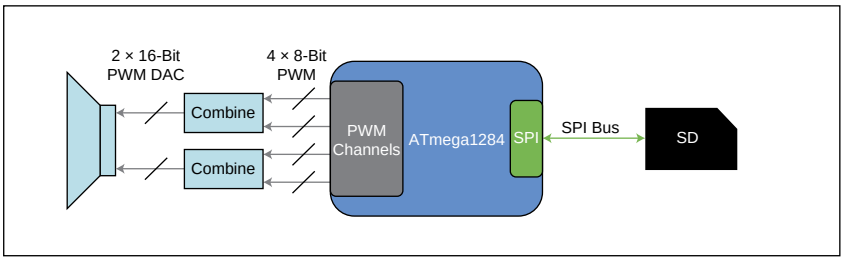


FIGURE 3
Petit FatFs open-source file system module





and a PWM DAC Module. WAVE files stored in the SD card can be retrieved with embedded hardware (SPI) and software (Petit FatFs module). Then the ATmega1284 parses the WAVE file and obtains the pure audio data, which is used as the PWM output (four channels). Four 8-bit PWMs are divided into two groups (corresponding to audio left channel and right channel), and combined separately into two 16-bit PWM DAC outputs. These two output channels are finally connected to the loudspeaker (see **Figure 4**).

The system hardware is very simple (see **Figure 5**). Note that the SD card can only handle a 3.3-V operating voltage level (see **Figure 6**). Even though ATmega1284p can also work at 3.3 V, it may not be able to work at 16 MHz.[1] To ensure system stability, the ATmega1284P is powered by 5 V, while the SD card is powered by 3.3 V with an inexpensive, linear regulator LT1117-3V3 (not shown in schematic).

To interact with SD card correctly, resistor dividers (3.3 and 1.8 kΩ) are used to guarantee the AVR's SPI signal level won't exceed 3.3 V. For he MISO signal, the microcontroller accepts 3.3 V, so it does not need an extra resistor divider.

A single PWM output can use one output pin to generate analog signals with 8-bit resolution. If you have multiple PWM outputs, you can add the outputs with appropriate "summing ratio" to get higher resolution. This summing ratio is implemented with a couple of resistors. For instance, mixing PWM outputs with the same resistor value will increase resolution by 1 bit. For the same reason, if

FIGURE 4
The system's overall structure

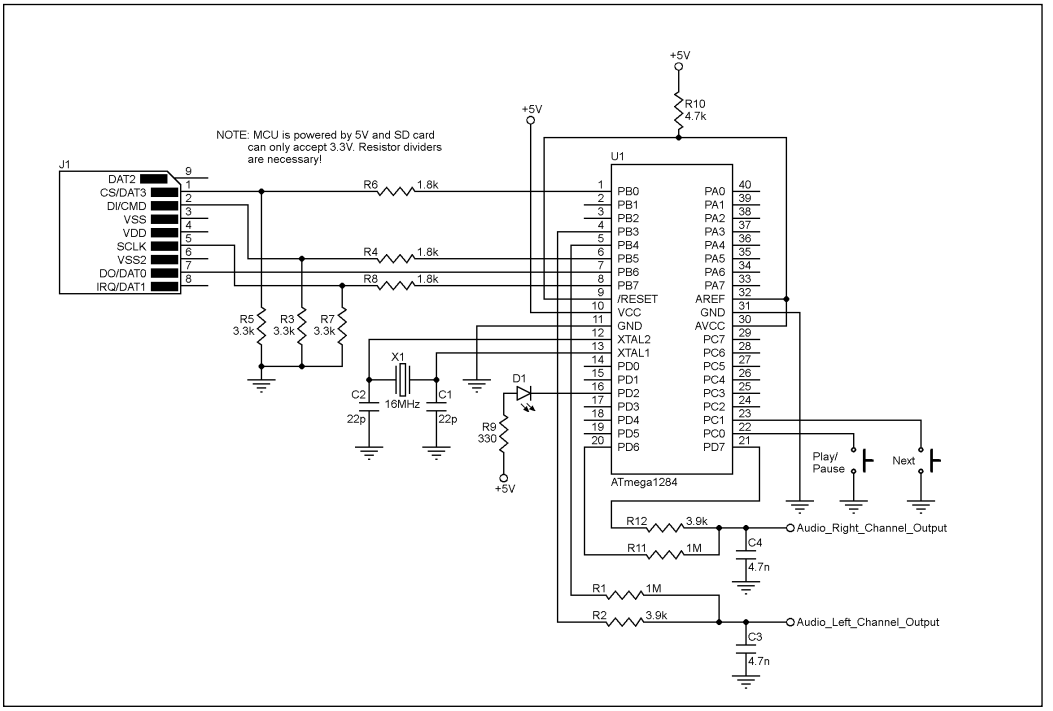
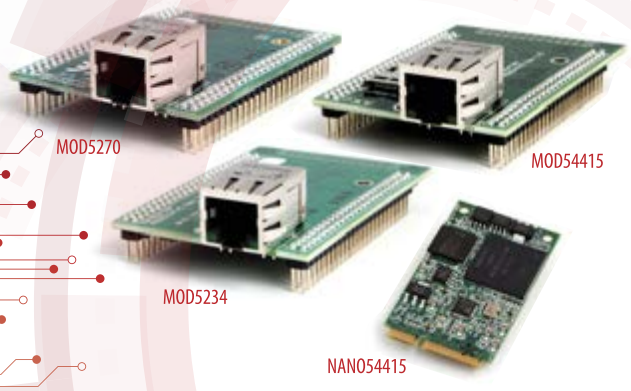


FIGURE 5
The system's circuitry

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

NetBurner
Networking in One Day!

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

freescale™
Alliance Member

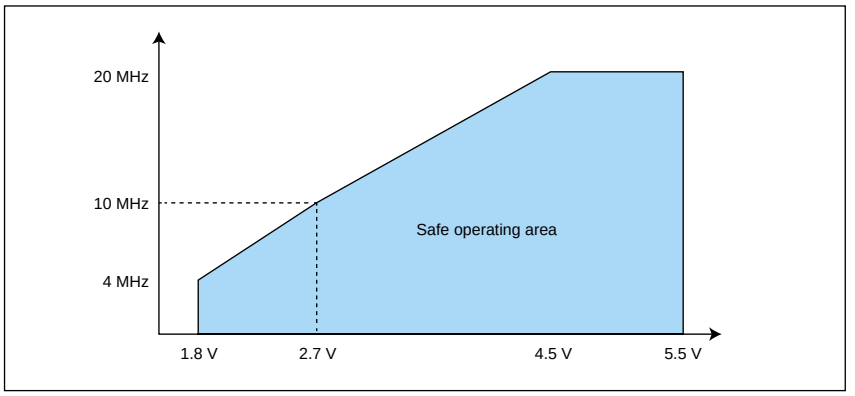


FIGURE 6
ATmega1284p Frequency vs Power. This figure shows that with 16-MHz operation frequency the safe power supply voltage should be larger than 3.6 V. So 3.3 V is not safe for the AVR to run at 16 MHz.

the resistor ratio is 1:256, the resolution will be increased by $\log_2 256 = 8$ bits.

In this project, the PWM combination circuit comprises two resistors: 1 MΩ and 3.9 kΩ (resistor ratio: 256.410). The larger resistor is connected to the LSB PWM output channel, while the smaller resistor is connected to the MSB PWM channel. The voltage resolution of the MSB PWM is about 0.0195 V. If you assume resistor accuracy (tolerance) is 1%, then the actual output should be: 0.0191 V < V < 0.0199 V. The deviation range is only 800 µV. If the resistor accuracy can be improved to 0.1%, it is possible to achieve full 16-bit resolution (76 µV).

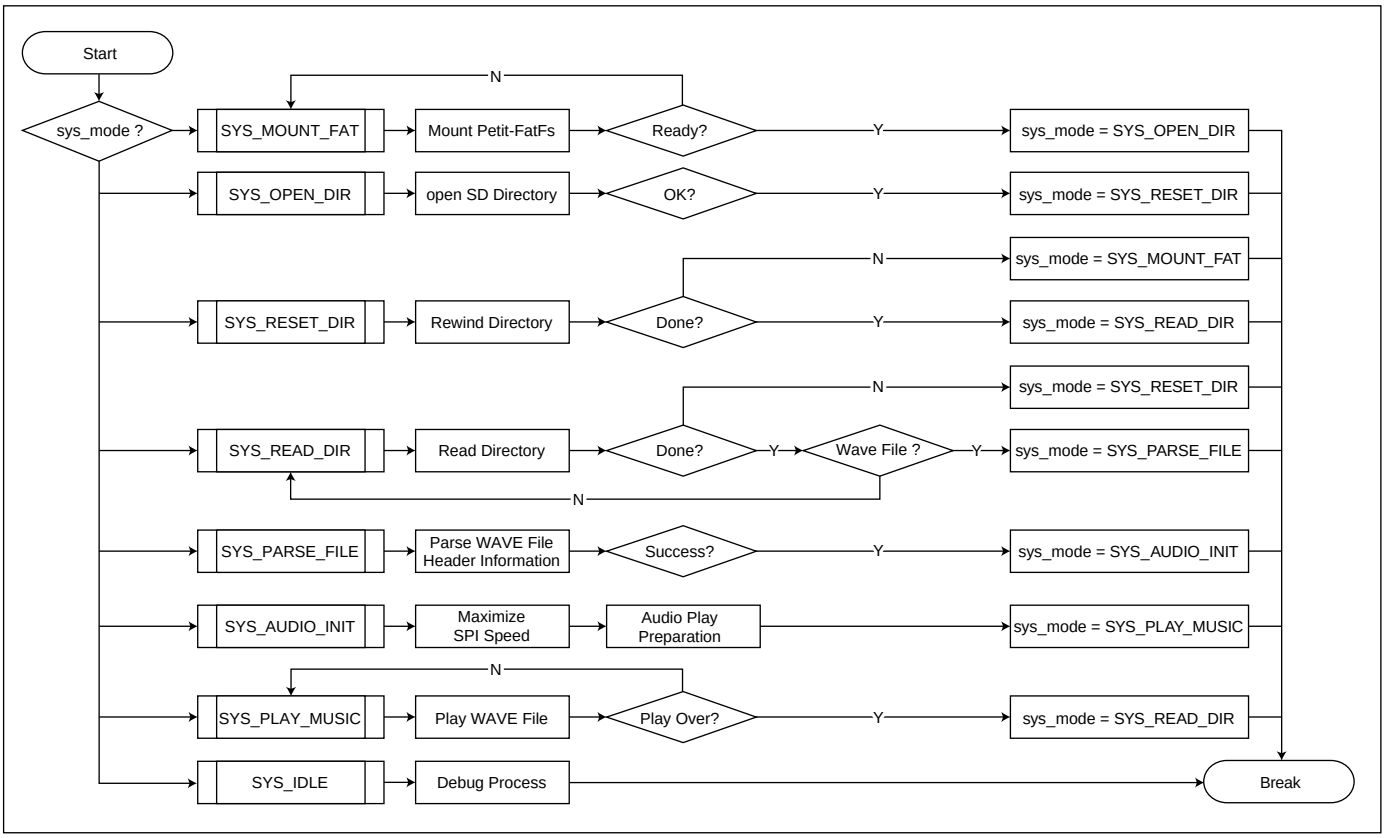
SOFTWARE DESIGN

From the perspective of top level, the whole system can be divided into seven states, and a simple state machine can be created to guarantee the system functionality (see **Figure 7**). All seven states, plus one state for debugging, is clearly organized; but it is still required to change the low-layer Petit FatFs interface to cooperate with this state machine. Specifically, we must change the low-layer data retrieving function in order to implement special data processing functionalities.

The low-layer data retrieve function of Petit FatFs is named `disk_readp`. File System layer calls this function to initialize FAT, read the directory, and read specified files, etc. This function first checks the type of the SD card to see if it is required to convert the address (Logical Block Address to Block Address). Then the CMD17 (`READ SINGLE BLOCK`) command is sent to the SD card. Once a valid response is received (0xFE), a data stream is created to forward the specified data into an assigned buffer. Finally, it skips the "un-aligned" data stream (the required data volume may not be an integer multiple of 512 bytes). All of the SD card's supported SPI commands are noted in the published specifications.

Since the `disk_readp` function needs a pointer to the target buffer, I can manipulate the pointer to embed my own function. For example, if the pointer is valid, it will execute the required read operation. On the contrary,

FIGURE 7
Top-level software flow chart



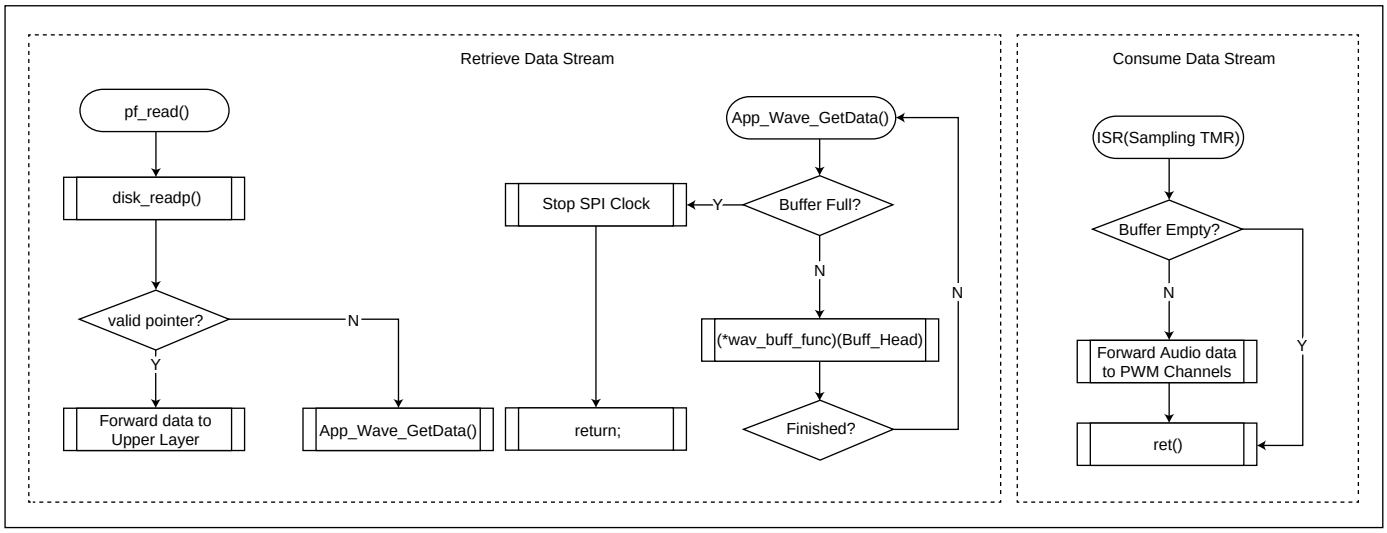


FIGURE 8
Diagram of data retrieving and consuming

if it is invalid—for instance, if it equals 0 (NULL pointer)—then it will execute my function. In this way, I can manipulate the data flow without hurting the normal operation of Petit FatFs. With this modification, the core functionality turns into buffer data flow control.

Since I can know all the information by parsing the wave file's header, I can set hardware parameters based on the parsing result, such as the "Sampling TIMER" configuration, which is used to forward audio data to PWM channels in its interrupt service routine (ISR). In this way, the function of the file system layer is to retrieve data from the SD card to a specified audio buffer while the "Sampling TIMER" consumes data from the audio buffer. A ring buffer is implemented to balance the Retrieve/Consume relationship (see **Figure 8**).

To guarantee an accurate interrupt interval, the code in the ISR is very short and simple. I used a function pointer to execute different buffer allocation functions, and it is initialized as soon as the wave file's format is identified.

PARSE WAVE FILE

The WAVE file format is pretty simple to parse. As a subset of RIFF, it has exactly the same structure defined in Microsoft's 1994 standards update titled "New Multimedia Data Types and Data Techniques (version 3.0)." According to this specification, the RIFF WAVE file format is shown in **Table 1**. Based on the format, I can design appropriate software to parse the wave file and get essential information I actually need. Note that the total bytes of `Chunk ID` and `Chunk Size` field is always 8. And considering that the upper layer function `pf_read` will internally increment the data pointer, I can actually fetch 8 bytes every time and do further processing based on `Chunk ID` information. (This coding style is Four Character Coding, or FCC.) The information is stored in Big Endian format, so I can either convert the original data or `convert` normalized ID for comparison. For instance, I can load 4 bytes directly since it is efficient and compare this word to reversed Chunk ID. This file parsing information is implemented in the `App_Wave_ParseHeader` function.

Please note that, the judgments of `LIST` (play list) chunk, DISP (display) chunk, and `fact` chunk are also added to guarantee compatibility, and we just need to skip these chunks if it happens. Meanwhile, for both debugging purposes and software robustness,

RIFF WAVE File Format

Endian	File Offset	Field Name	Field Size	Description
big	0	Chunk ID	4	**"RIFF" Chunk Descriptor**
little	4	Chunk Size	4	The format field here is "WAVE", and it requires
big	8	Format	4	two sub-chunks: "fmt" chunk and "data" chunk.
big	12	SubChunk1 ID	4	
little	16	SubChunk1 Size	4	
little	20	Audio Format	2	**"fmt" Sub-Chunk**
little	22	Channel Number	2	This region contains all essential information we
little	24	Sample Rate	4	need to know about a WAVE file, such as sample
little	28	Byte Rate	4	rate, channel number etc.
little	32	Block Align	2	
little	34	Bits Per Sample	2	
big	36	SubChunk2 ID	4	**"data" Sub-Chunk**
little	40	SubChunk2 Size	4	The field "SubChunk2 Size" indicates the size of
little	44	Data	SubChunk2 Size	RAW audio data.

TABLE 1
WAVE file format

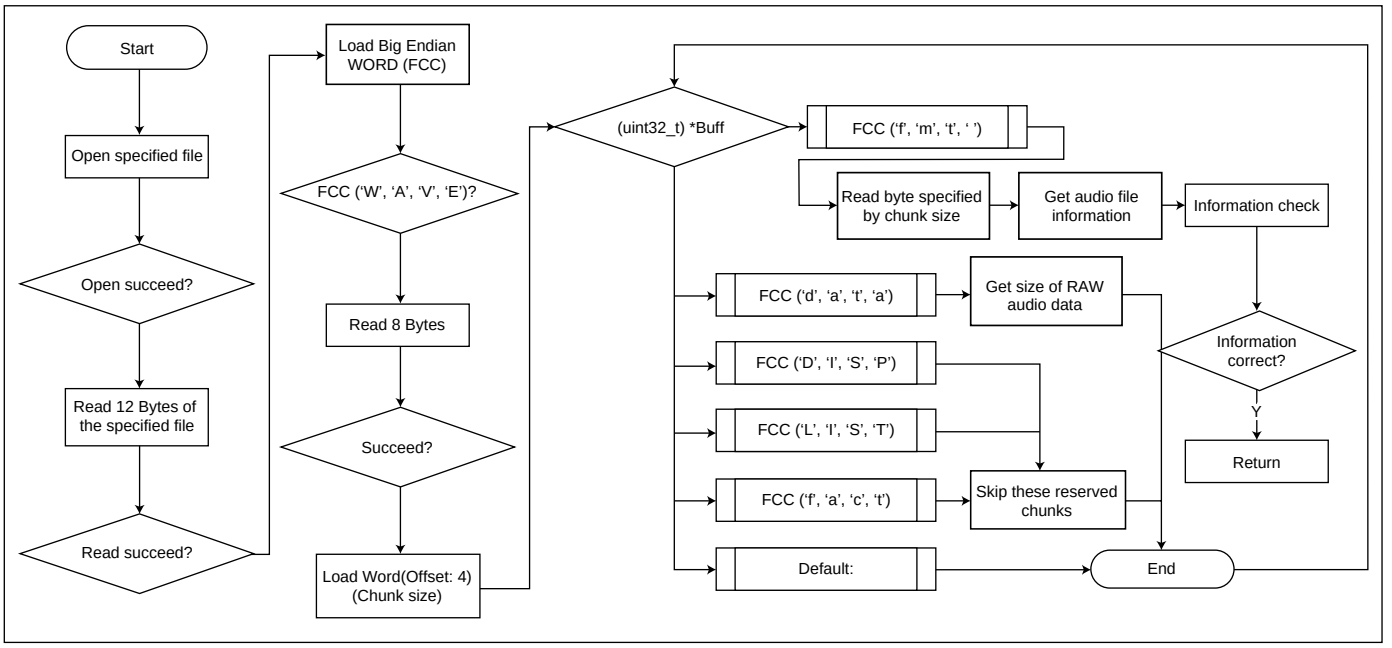


FIGURE 9
Parsing function diagram

different return values are defined. Based on the audio information I get from the audio file, I can check whether or not the file is corrupted and return corresponding error information. The function diagram is shown in **Figure 9**.

Once I knew the format of the header file, I still had to know the audio data arrangement of different WAVE files. There are four kinds of WAVE files: 8-bit Mono, 8-bit Stereo, 16-bit Mono, and 16-bit Stereo. Understanding that, I was able to design efficient buffer structure and data retrieving functions. Simple code in the ISR means extra work is needed outside the ISR. Refer to **Table 2** for the data arrangement based on RIFF file format of WAVE file.

AUDIO BUFFER DESIGN

Based on the audio data distribution of different wave files, four independent buffers are created according to the format of 16-bit Stereo Wave File Format, since it needs 4 bytes per sample. All buffers are of the same size, 256, and equipped with the same `Head` and `Tail` 8-bit pointer. With 8-bit pointers, ring buffer operations becomes simpler: simply increment head or tail pointer without checking buffer size limitations, since it will turn into zero once its value exceeds 255. Another reason is that 8-bit addition is faster that 16-bit addition. It is better to design four independent buffers rather than creating a really large buffer.

Meanwhile, since pointer variables are used frequently, I defined these variables in register for fast access. Information on implementing this is available in Atmel's online document, "AVR Libc Reference Manual." You need to use keyword `register` to declare the variable you want to put in register. For instance, to declare variable `buff_head` in register `r3`, simple use `register uint8_t buff_head asm("r3")` to declare it. According to official explanation, it should be safe to use r2 through r7. Registers r8 through r15 can be used for argument passing by the compiler in case many or long arguments are being passed to callees. If this is not the case throughout the entire application, these registers could be used for register variables as well. Extreme care should be taken that the entire application is compiled with a consistent set of register-allocated variables, including possibly used library functions.[2]

Before playing a specified wave file, I needed to configure the sample TIMER correctly and assign appropriate variables to make the data process efficiently. Note

8-bit Mono	Sample 1	Sample 2	Sample 3	Sample 4
8-bit Stereo	*Left Channel*	*Right Channel*	*Left Channel*	*Right Channel*
	Sample 1	Sample 1	Sample 2	Sample 2
16-bit Mono	*LSB*	*MSB*	*LSB*	*MSB*
	Sample 1		Sample 2	
16-bit Stereo	*Left Channel*		*Right Channel*	
	LSB	*MSB*	*LSB*	*MSB*
	Sample 1		Sample 1	

TABLE 2
Audio data allocation

Verilog HDL

With the right tools

such as this book,

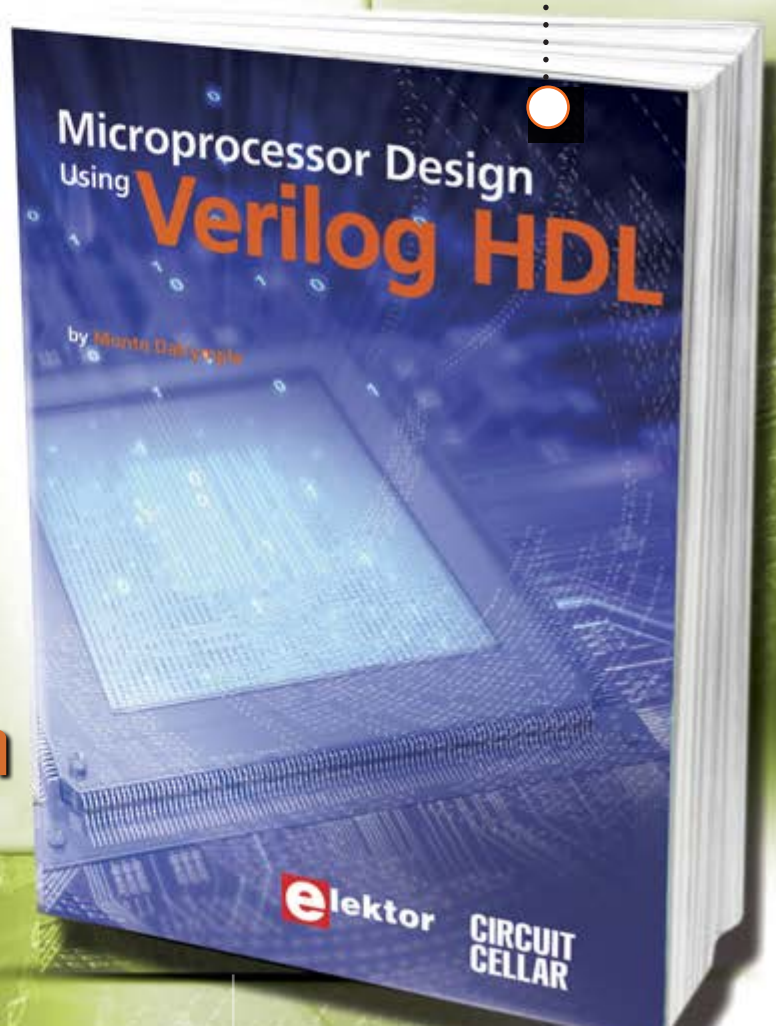
designing a microprocessor can be easy.

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.

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!

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

Wave Format	Buffers	Value
8-bit Mono	Buff_Lch_Lo[Buffer_Head]	0
	Buff_Lch_Hi[Buffer_Head]	Bsp_SPI_Receive()
	Buff_Rch_Lo[Buffer_Head]	0
	Buff_Rch_Hi[Buffer_Head]	Buff_Lch_Hi[Buffer_Head]
8-bit Stereo	Buff_Lch_Lo[Buffer_Head]	0
	Buff_Lch_Hi[Buffer_Head]	Bsp_SPI_Receive()
	Buff_Rch_Lo[Buffer_Head]	0
	Buff_Rch_Hi[Buffer_Head]	Bsp_SPI_Receive()
16-bit Mono	Buff_Lch_Lo[Buffer_Head]	Bsp_SPI_Receive()
	Buff_Lch_Hi[Buffer_Head]	Bsp_SPI_Receive() - 0x80
	Buff_Rch_Lo[Buffer_Head]	Buff_Lch_Lo[Buffer_Head]
	Buff_Rch_Hi[Buffer_Head]	Buff_Lch_Hi[Buffer_Head]
16-bit Stereo	Buff_Lch_Lo[Buffer_Head]	Bsp_SPI_Receive()
	Buff_Lch_Hi[Buffer_Head]	Bsp_SPI_Receive() - 0x80
	Buff_Rch_Lo[Buffer_Head]	Bsp_SPI_Receive()
	Buff_Rch_Hi[Buffer_Head]	Bsp_SPI_Receive() - 0x80

TABLE 3
Audio buffer allocation function

that in the Software Design section of this article, I used a function pointer to call specified function to forward data stream into corresponding buffers. Based on the data arrangement of different wave files, all I need to do is to make sure that each data is put into exactly the same location as the data arrangement. As for mono wave files, I simply put the same value to buffers for left/right channels. The function pointer prototype is `typedef void (*memProcFunc)(uint8_t)`, and a variable of `memProcFunc` type is defined. This function pointer is initialized when the audio information is correctly parsed from the header. Four independent functions are implemented for different data buffer process. **Table 3** shows the audio buffer arrangement of these four functions.

PLAY WAVE FILE & SCAN KEY

The Wave Play function is pretty simple (see **Figure 10**). First, it processes the audio data after the header since the size of the data volume is less than 512 bytes. After that, this function processes the remaining audio data in the unit of 1,024 bytes until the size of remaining data volume is less than 1,024 bytes.

The key scan subroutine is embedded in the Wave Play function. To remove key glitches, a scan subroutine must be implemented now that 1,024 bytes are read every time (except the data following the header). Based on the SPI's speed, I can roughly calculate the time interval between read operations. In this way, I can take advantage of this processing interval to execute key scan routine.

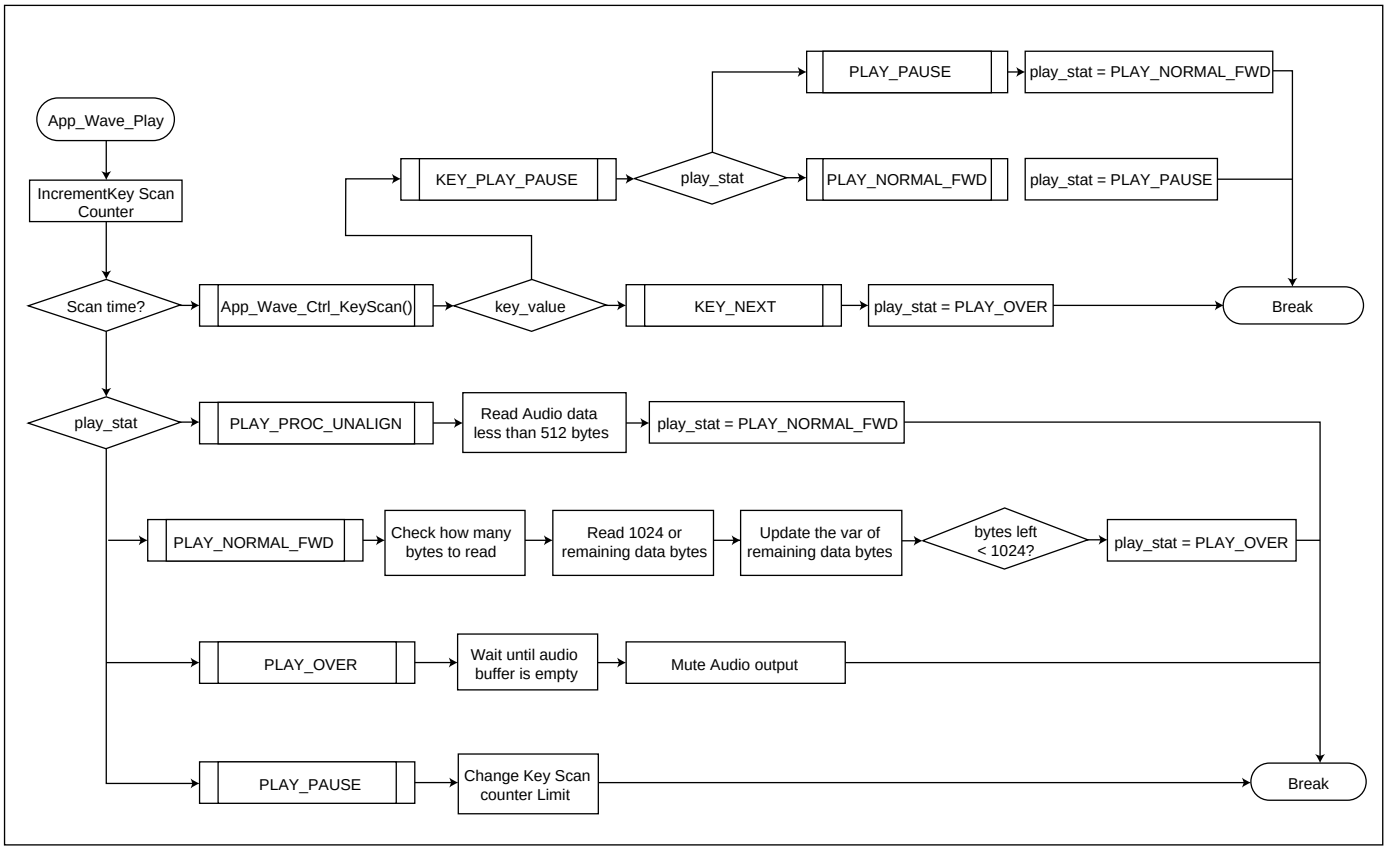


FIGURE 10
WAVE play function flowchart

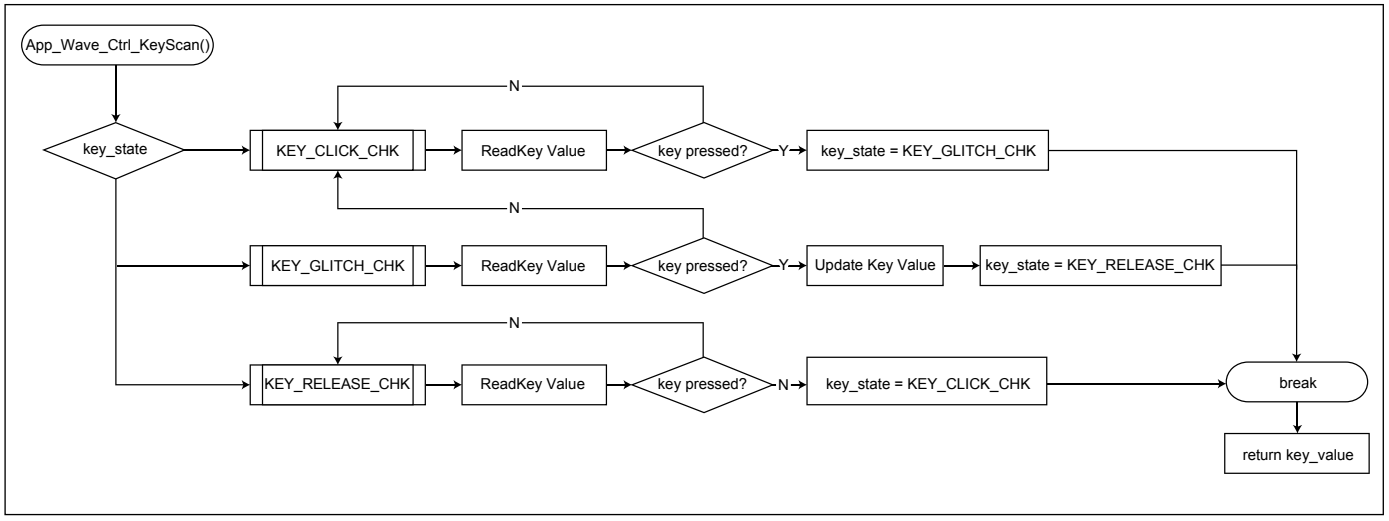


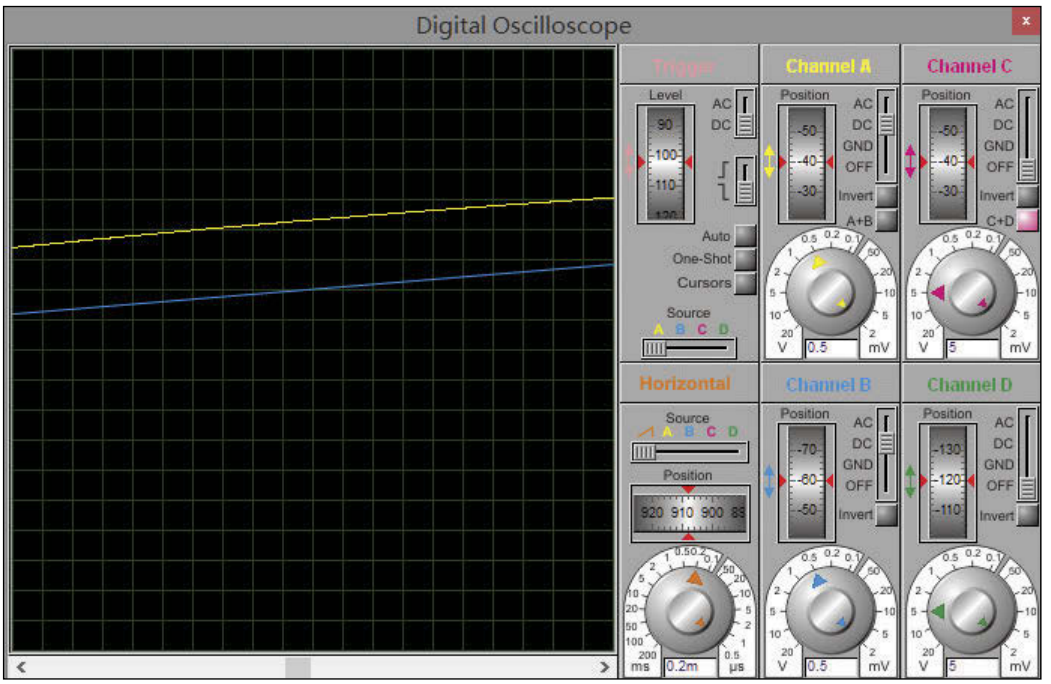
FIGURE 11
Key scan flowchart

$$f_{SPI} = \frac{f_{CPU}}{\text{SPI Clock Prescaler}} \times (\text{SPI2X ? (2:1)}) = \frac{16,000,000}{2} = 8 \text{ MHz}$$

$$t_{SD\_BlockRead} = \frac{1}{f_{SPI}} \times 8 \times 1,024 = 1.024 \text{ ms}$$

With the block read time interval calculated above, the `App_Ctrl_KeyScan` function is called in `App_Wave_Play` to execute the key scanning routine to avoid key glitches when pressed or released. **Figure 11** is the key scan flowchart.

PHOTO 2
Proteus Simulation



TESTS & RESULTS

The purpose of the hardware test was to determine the accuracy of the 16-bit combination composed of high-precision resistors. It is difficult to measure its functionality. A high-precision multimeter was required to achieve rigorous validation. To logically verify this test, I created test firmware to evaluate the hardware. The purpose of the test firmware was to generate PWM in Fast PWM mode, and the PWM value was incremented every 200 ms. With Proteus (Version 8.1 sp1) and the schematic mentioned in the "System Structure & Hardware" section of this article, I generated the oscilloscope output shown in **Photo 2**. Note that the horizontal scale is adjusted until the "Climbing Steps" can be clearly observed.

Testing the audio quality is problematic. Results are subjective. What's more, the human ear is not as accurate as a machine. There are actually two amateurish ways to evaluate audio quality. One way is to record the audio data output (via LINE-IN audio port) and save the audio data to the same wave format as the original wave files. Then, you can use Adobe Audition software to compare the two audio files. The other way is to capture the play time and compare it with original wave file.

SAMPLE RATE

For each wave file format, I chose a different song with same sample rate (44,100 Hz) to implement this simple test. A simple MFC program is created to capture the song

circuitcellar.com/ccmaterials

REFERENCES

[1] Atmel Corp., "8-bit Atmel Microcontroller with 16/32/64/128K Bytes In-System Programmable Flash," 8272G-AVR-01/2015, 2015.

[2] Atmel Corp., "AVR Libc Reference Manual," 2014, www.atmel.com/webdoc/AVRLibcReferenceManual/.

RESOURCES

Elm-Chan, "Petit FAT File System Module," http://elm-chan.org/fsw/ff/00index_p.html.

Free Software Foundation, Inc., "GCC Manual," Version 4.9.2.

Microsoft Corp., "Microsoft Extensible Firmware Initiative FAT32 File System Specification," Version 1.03, 2000.

———, "New Multimedia Data Types and Data Techniques," Revision 3.0, 1994.

SanDisk, "SanDisk SD Card," Version 2.2, 80-13-00169, 2004.

———, "SanDisk Secure Digital Card," Version 1.9, 80-13-00169, 2003.

SD Group, "SD Specifications Part 1 Physical Layer Simplified Specification," Version 4.10, 2013.

SOURCES

ATmega1284p Microcontroller
Atmel | www.atmel.com

SD Card
KingSton | www.kingston.com

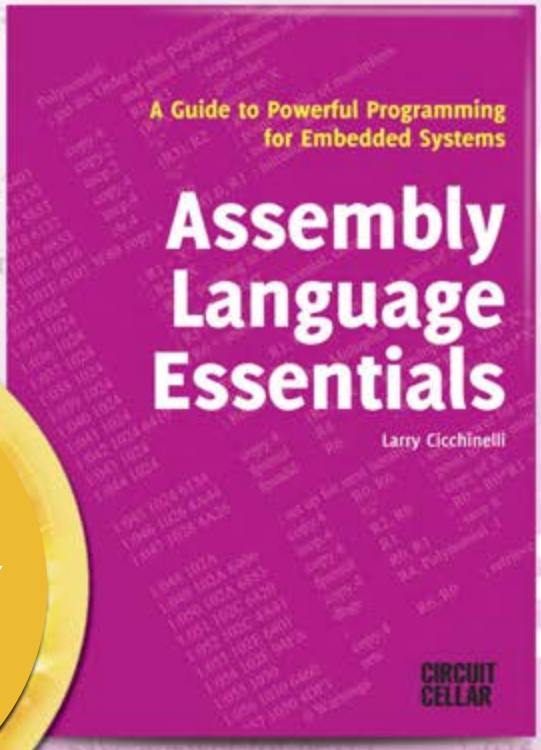
ASSEMBLY LANGUAGE ESSENTIALS

Assembly Language Essentials is a matter-of-fact guide that will introduce you to the most fundamental programming language of a processor.

Larry Cicchinelli provides readers with:

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

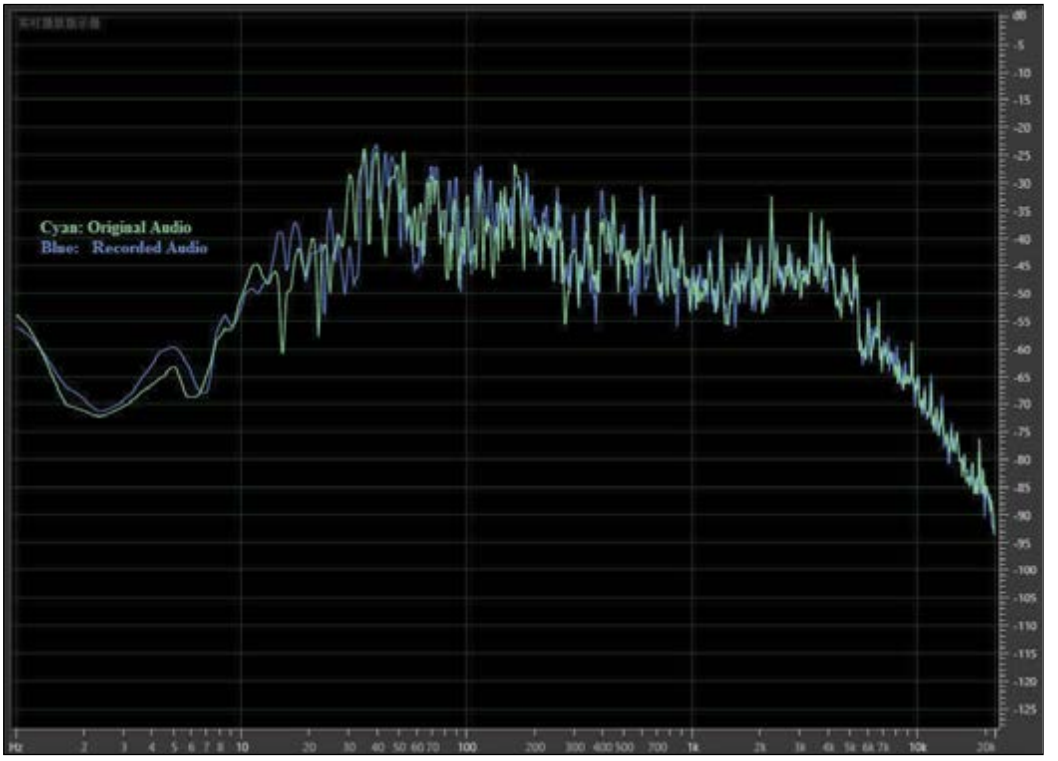
circuit cellar



ORDER YOUR COPY TODAY!

www.cc-webshop.com

PHOTO 3
FFT Analysis based on Adobe Audition



start information (a simple "play!" string) output from hardware USART interface.Once the string is captured, a TIMER is started to record elapsed time until the end information ("over!" string) is captured. The results are available on the *Circuit Cellar* FTP site. Note that 16-bit TIMER1 is used as the sampling TIMER, and the value for output compare register is: 16,000,000/44,100 + 0.5 – 1 = 362). The "0.5" is there for correct rounding. However, the sample frequency in this way is 44,077.135 Hz. That's why the deviation exists.

As the bytes per sample increase, the deviation value becomes smaller. It is because the audio data is consumed faster with 16-bit stereo audio format. I can imagine that, if the sample rate of the wave file goes a bit higher, it is very possible that current system cannot play it normally because of the limitation of SPI speed (8 MHz). CPU overclock may be essential for higher audio quality.

AUDIO QUALITY ANALYSIS

It is difficult to compare two different audio files intuitively, even with Adobe Audition. However, I can merge the recorded wave file with original wave file to test mono wave files. For instance, with Adobe Audition, I can put the original wave file to the left channel and copy the recorded wave file to the right channel. Adobe Audition can then analyze it.

For 8-bit Stereo wave files, we actually need to extract audio data from each channel and repeat the aforementioned procedure. For 16-bit Stereo wave files, I can simply use the high-order byte as the data source for analysis. It is safe because Adobe Audition only accumulates the audio samples at different frequency points. It is not relevant with the amplitude. For this project, I focused on 16-bit Stereo analysis. However, I was not able to analyze all four kinds of wave files due to time limitation. But this test method can help to perform cross analysis to get more information.

Photo 3 shows the FFT diagram generated by Adobe Audition. All the data of FFT analysis can be exported for further analysis. Then the analysis result is imported to Excel to calculate the deviation. Part of the result is posted on the *Circuit Cellar* FTP site. From the FFT waveform I can still see some noise which causes the deviation with original data. However, if a good low-pass filter is implemented, the result should be better. Even though, the average deviation is 0.46%, which proves that the audio quality is still good.

ABOUT THE AUTHOR

Wancheng Zhou (leo2010@msn.com) earned a Master's degree at Cornell University. He is interested in embedded system design based on 8/32-bit MCUs.

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

contest cds

back issues

archive cds

CIRCUIT CELLAR
CC-WEBSHOP
ARCHIVE • STORE

www.cc-webshop.com

FEATURES

'Net-Connected Security Network

PHOTO 1
Bottom and top view of the first hardware version

This innovative security system comprises several nodes that collect and process information independently, generate alarms, and communicate with the others in order to examine the threats from multiple angles. Each node has a WIZnet W5500 network chip, passive Power over Ethernet, PIR, servo motor, storage, video camera, and image-processing capabilities.

By Claudiu Chiculita & Liviu Ene (Romania)

We started this project because we wanted an inexpensive module that we could replicate for various purposes in a home area network. In addition, we wanted a system with a surveillance camera connected to a wired Ethernet network (see **Photo 1**). We had a few requirements for the system: it had to be connected via wired Ethernet; it has to be powered via passive power over Ethernet (PoE); and it had to be flexible enough so it can be repurposed for other applications that require Ethernet connectivity.

Initially, we made plans to build the system around a Microchip Technology PIC microcontroller with an Ethernet module (from the PIC18 or PIC32 series). One important disadvantage was that the TCP/IP stack compiled with the non-optimized free compilers from Microchip takes most of the program memory, leaving little room for anything else. Thus, due to our need for Ethernet connection, we decided to implement a WIZnet W5500 chip. In addition, because the components required for this project were inexpensive, it was feasible to build a network comprising multiple devices (each one of them being a node in the network) that could interact in a surveillance/security application (see **Figure 1**).

Each node taken independently behaves as a surveillance camera that can continuously acquire images and process them in order to detect movement. When movement is detected, it rotates to keep the target in view using its internal servomotor; meanwhile, images are sent to a PC or saved in the internal storage. All the nodes in the network know their position and orientation in space, and if one of them detects movement, it sends an alert to all the others with the approximate coordinates of the event. If the other nodes are able, they will rotate to capture the event taking place at the noted position. This way, the event is examined from all available angles. A PC application initially configures the devices with the network map. Optional features include a user interface, remote storage, and e-mail messaging.

In this first part of this article series, we'll detail the system's hardware and present a node's basic firmware. In the second article, we'll explain how the network functions and

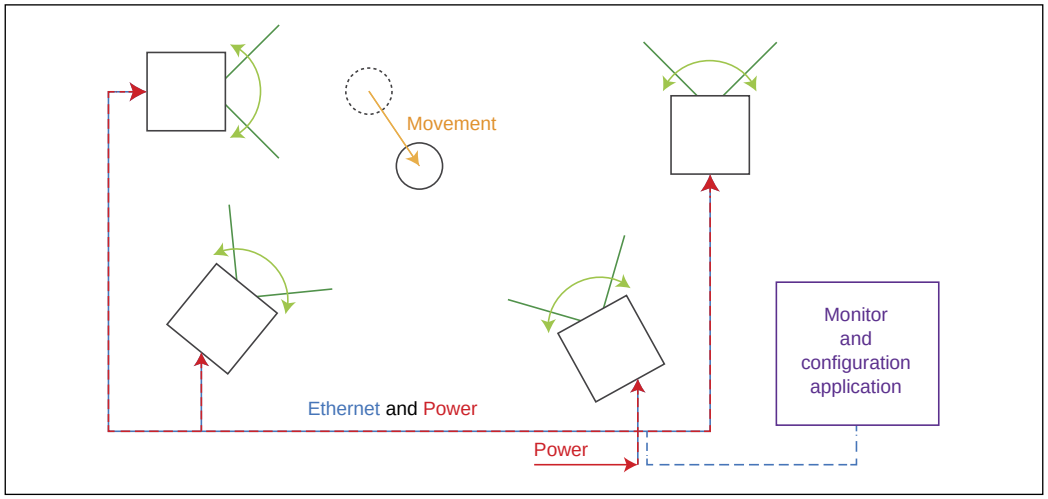


FIGURE 1
An overview of my security network

cover the topics of image processing inside each node, movement detection, messaging, and monitoring.

NODE STRUCTURE

Figure 2 depicts a node's structure. The core components are the microcontroller and W5500 chip. The former interacts with all the peripherals. The latter provides the interface to Ethernet and includes the network protocol stack.

We use a COMedia serial C328 camera module to capture images that are transmitted through UART to the microcontroller. Storage is provided via EEPROM or an SD card connected to the same SPI bus as the W5500 chip. A miniature servomotor rotates the module and a PIR sensor can detect movement detection during night.

The network nodes can be placed far apart in locations where a power supply is not readily available. The Ethernet cable will always be present, so it makes perfect sense to have the power running through the same cable. In conclusion, all the devices are designed to be powered using passive PoE. Power is transmitted using the unused pairs in the Ethernet cable (DC positive is applied on pins 4 and 5 and ground on the RJ-45 connector's pins 7 and 8). Note that this passive method is not compatible with the IEEE 802.3af PoE standard that requires additional circuitry.

Because you can locate the devices far from the supply source, the cables' resistance could create an important voltage drop. Thus, the power source has to be at a significant higher voltage than the working voltage of the device. The voltage drop is dependent on the length and the type of cable (e.g., for a AWG24 cable) having a resistance of 85 mΩ/m, on a distance of 100 m, two wires in parallel, back and forth, we get 8.5 Ω (i.e., 2 × 100 m × 0.085/2). If a device reaches a maximum peak consumption of 1 A, then there could be a voltage drop of 8.5 V. This means there is a large variation in the input voltage

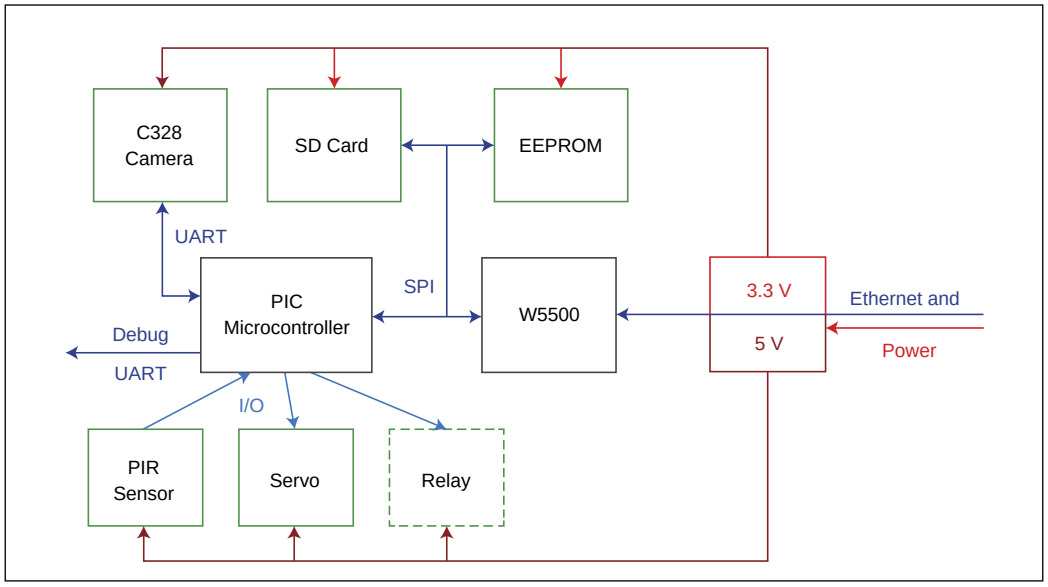


FIGURE 2
Interconnections between the blocks inside a node

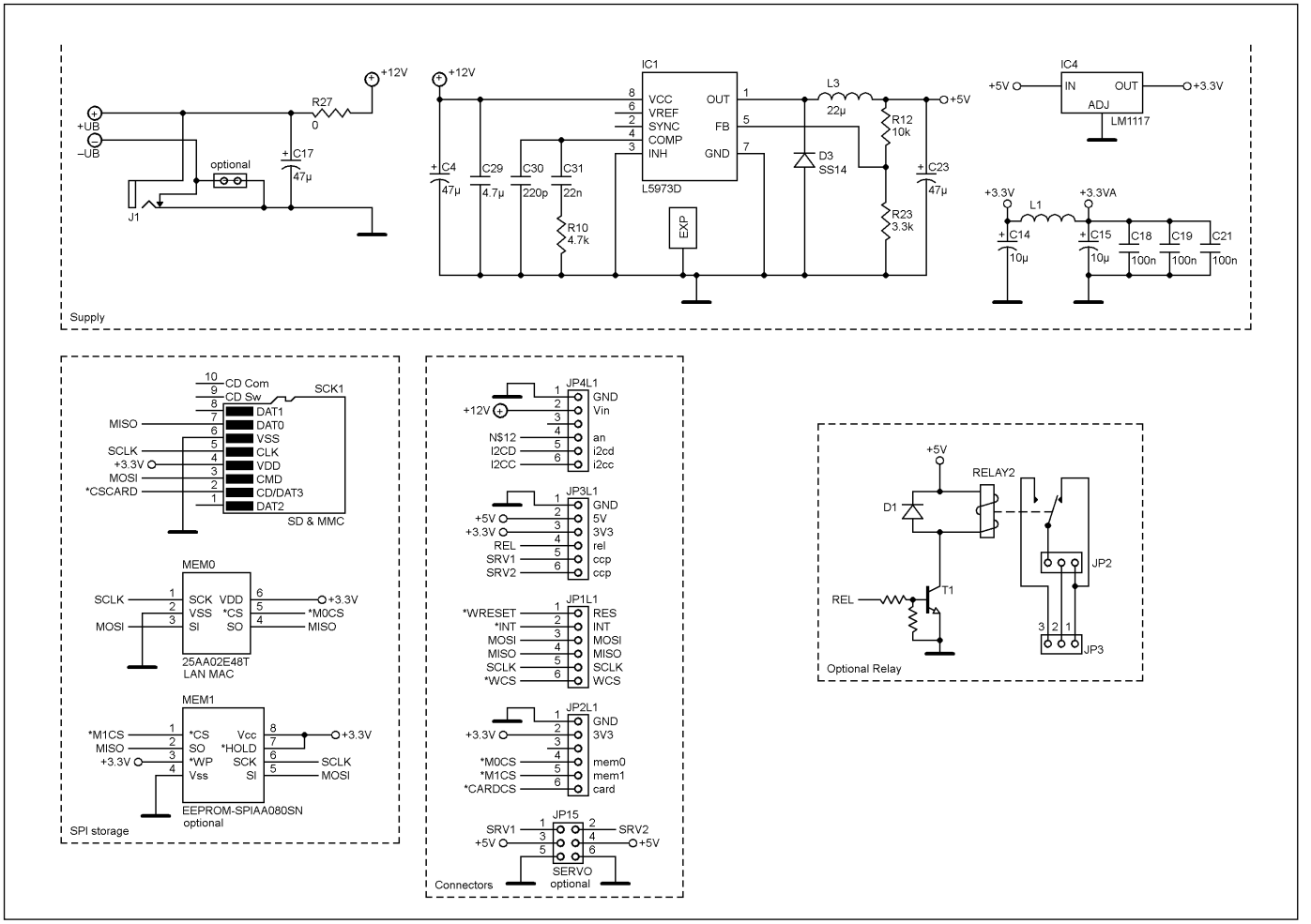


FIGURE 3
Schematics for the supply and connectors located on the base board

(depending on the current drawn and length of the cable), and the use of a DC/DC voltage converter is justified. For the supply, we used an STMicroelectronics L5973D step-down power switching regulator (see **Figure 3**). This allows the device to be powered with voltages up to 35 V and a maximum consumption of 2 A. The output voltage is selected at 5 V, which is required by servomotor, PIR sensor, and relay. The 3.3-V supply for the rest of the components is obtained using a LM1117 0.8-A linear regulator.

During tests, the power to all the nodes was delivered through a 24-V power brick that can supply up to 3 A. Power was injected into the Ethernet cables using simple passive PoE injectors (see **Photo 2**).

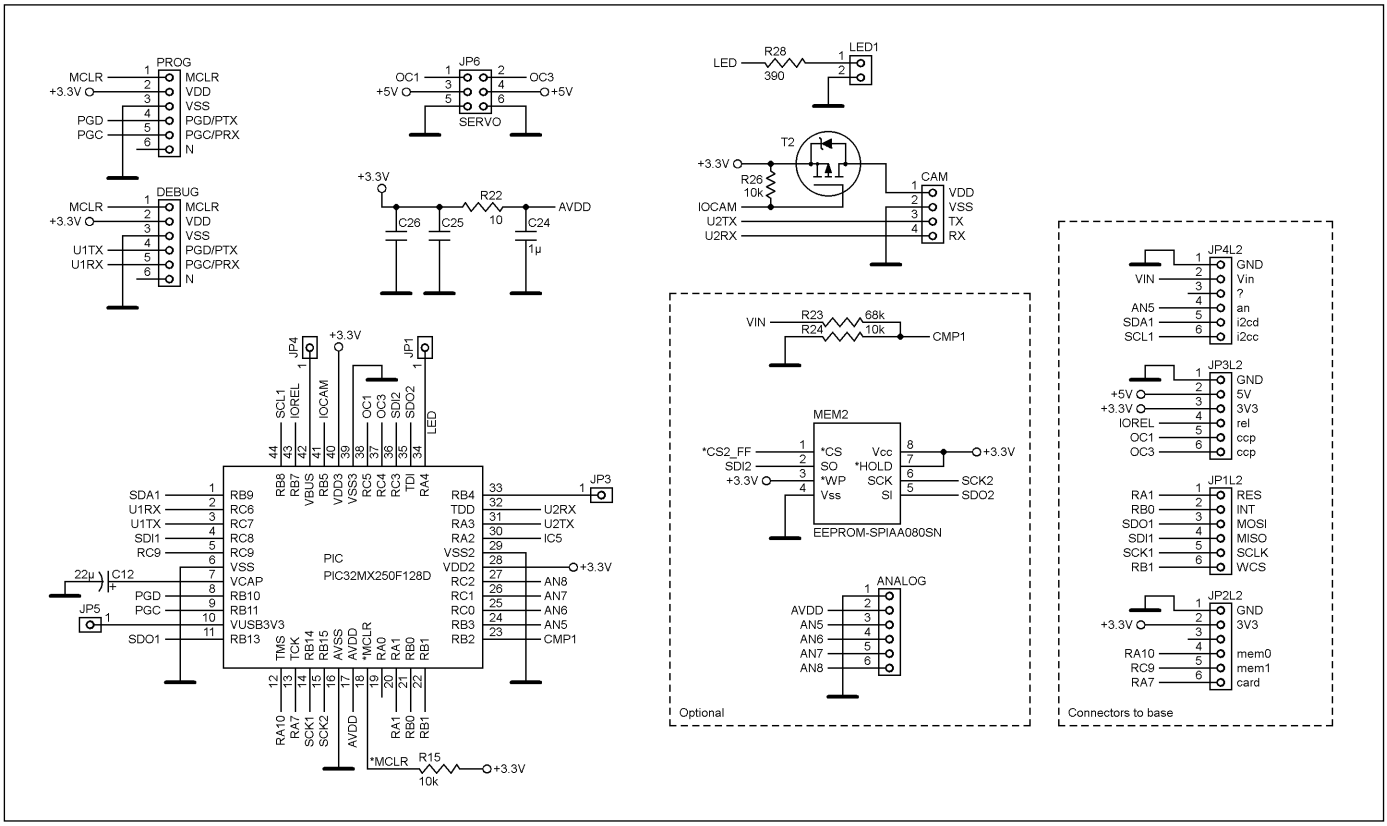
PHOTO 2
Powering the nodes using is done using a simple passive PoE injector

W5500 CONFIGURATION

All the complexity of managing the network protocols is contained in the W5500 chip, which has sufficient RAM buffers and a fast SPI. The most notable disadvantages are the lack of an internal unique MAC address and the inability to perform IP fragmentation.

For the transport protocol, we chose UDP instead of TCP for a few reasons: we had to be able to broadcast messages to all the nodes; the central PC application might not always be available to maintain a connection; and it is better suited for real-time image transmission. A single UDP socket is initialized with a buffer of 8 KB for all communication purposes inside the network.

The w5500 does not have its own MAC



address, so we used Microchip Technology 25AA02E48T external SPI EEPROM (in a small SOT23-6 package) containing a preprogrammed unique 48-bit node address. The MAC is read from EEPROM and written into the W5500, thus giving it a proper unique address on the LAN.

Because there was enough memory available in W5500 buffers, two functions were implemented to use it as an external serial RAM in order to store a raw image taken from the camera in case internal RAM was not enough. A small raw image is 4,800 bytes, so an 8 KB buffer is enough. (In the end, we didn't use this feature because there was still RAM available in the PIC microcontroller.)

FIGURE 4
Schematics for the microcontroller and connectors located on the top board

MICROCONTROLLER

Image processing is performed inside the node, so the first criteria in choosing a microcontroller (to perform all the acquisition, processing, and communication) was the amount of available RAM. The second requirement was of available interfaces:

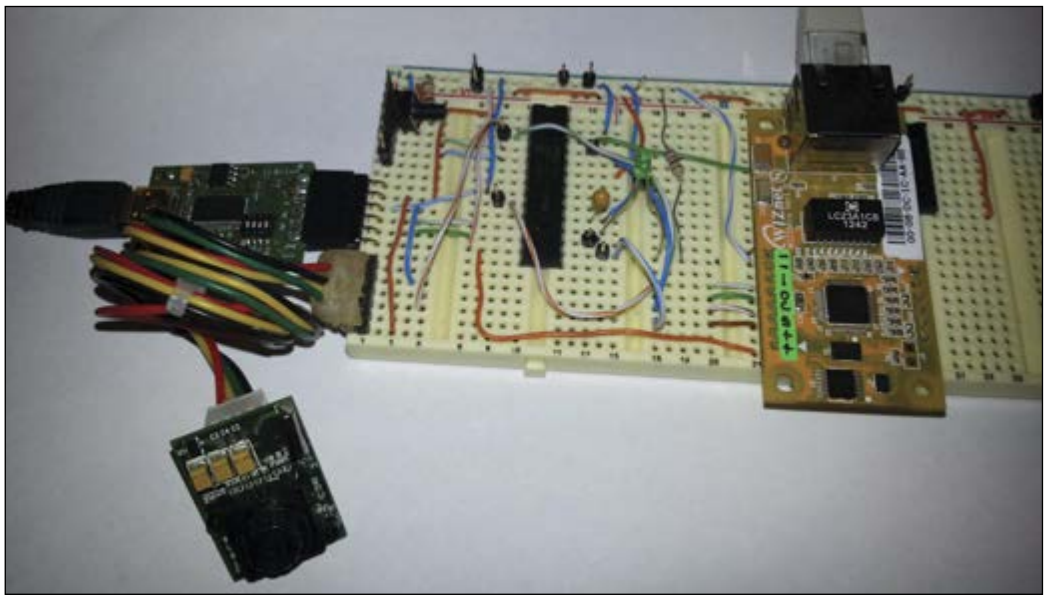
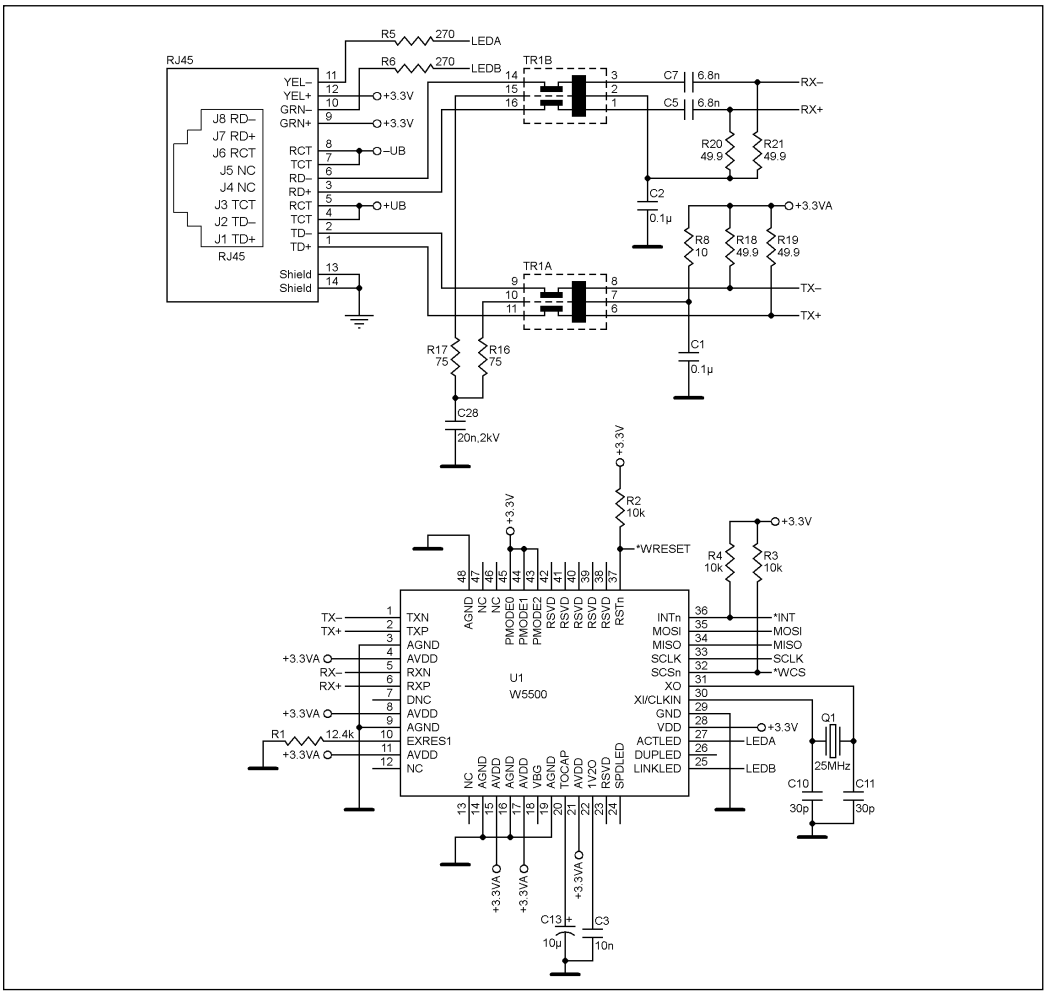


PHOTO 3
First tests and development with DIP components on the breadboard

FIGURE 5
Schematics for the network interface



a minimum of two UARTs (one for camera interface and the other for monitoring and debugging), at least one SPI for interfacing with the W5500 chip, but a second SPI was desired for interfacing with other peripherals or a possible SPI camera.

Image processing is computing intensive. But because in this case the communication interface is the bottleneck, processing power was not a priority. For this task, we selected the PIC32MX250F128D microcontroller for its 32 KB of RAM, the largest memory available at the time on PIC with a package of maximum 44 pins. It has the desired two UARTs, two SPIs, and can run up to 50 MHz, which is enough for the processing tasks (see **Figure 4**). The 128 KB of program memory offers nearly enough space for the firmware and for future developments, due to the large code generated by the unoptimized XC32 compiler.

CONSTRUCTION

The WIZnet WIZ550io module was easy to implement because it is auto-configurable and simple to breadboard (as opposed to older WIZnet boards with double rows of pins spaced 2.0 mm apart). The PIC32MX250F128B microcontroller we first used has the same core and peripherals as the "D" model, but it comes in a 28-pin DIP package. The setup also included a USB-to-UART FTDI adapter, a camera, and an SD card. This was the development platform until the first PCBs arrived (see **Photo 3**).

Before the PCB design, we selected a small case (to suspend it as a camera) that was tall enough for the Ethernet controller (the tallest essential component in the design). The case we chose was relatively short (at 6 cm × 6 cm) in length, but with more spacious in height (2.5 cm). As a result, we stacked two boards. We also knew this configuration would give us the flexibility in the future to replace the microcontroller and peripherals.

The baseboard contains the W5500 chip, an Ethernet jack, magnetics, and a DC/DC power supply for both 5 and 3.3 V (see **Figure 5**). It also includes a micro SD card and the EEPROM because the SPI bus was available. In addition, we positioned the servomotor (or optional relay) and power connectors near the tall Ethernet jack. The top board contains the microcontroller and all the connections. As

The #1 Flash Memory Conference!

Featuring Flash in Enterprise Storage

Keynotes

Micron • NetApp
Oracle • PMC
Samsung • SanDisk
Seagate • SK-Hynix
Toshiba • Toyota
Kaminario • Tegile

"Solid state drives will be the biggest change in storage, a total game-changer, and flash will be the dominant type of SSD for the foreseeable future."

Joe Tucci, EMC

Features

Major Exhibitors
Top Industry Keynotes
Leading Experts
SSDs and Controllers
New Technologies
Enterprise Storage
Pre-Conference Seminars
VC Forum
Market Research Session

Are you struggling with crucial Solid State Drive (SSD) decisions? Can SSDs resolve your application bottlenecks? How can you maximize SSD performance? Flash Memory Summit will explore new frontiers in enterprise storage and help you make the right choices.

Highlights

Three days packed full of seminars, forums, keynotes, and sessions:

- Plenary on 3-D Flash
- Forums on NVMe, Architectures, Enterprise SSDs, Enterprise Storage Design, PCIe SSDs, and Application Performance
- Sessions on Enterprise SSD Buying Trends and Flash Storage Performance Measurements
- New Technologies Sessions: RRAM, MRAM, and Life Beyond Flash
- Annual Update: Flash Technology, Enterprise Flash Storage, Interfaces, and New Technologies
- Beer, Pizza and Chat with the Experts Session

Don't miss the 10th Annual Flash Memory Summit and Exhibition! Hear from industry leaders, learn about the latest technologies, and talk to key vendors. Nothing else comes close!

Register online today

www.FlashMemorySummit.com

Circuit Cellar readers: Enter priority code SPGP for $100 discount!

FREE PARKING

10TH ANNIVERSARY
Flash Memory
SUMMIT

August 11-13 2015

Santa Clara Convention Center

produced by Conference ConCepts, Inc

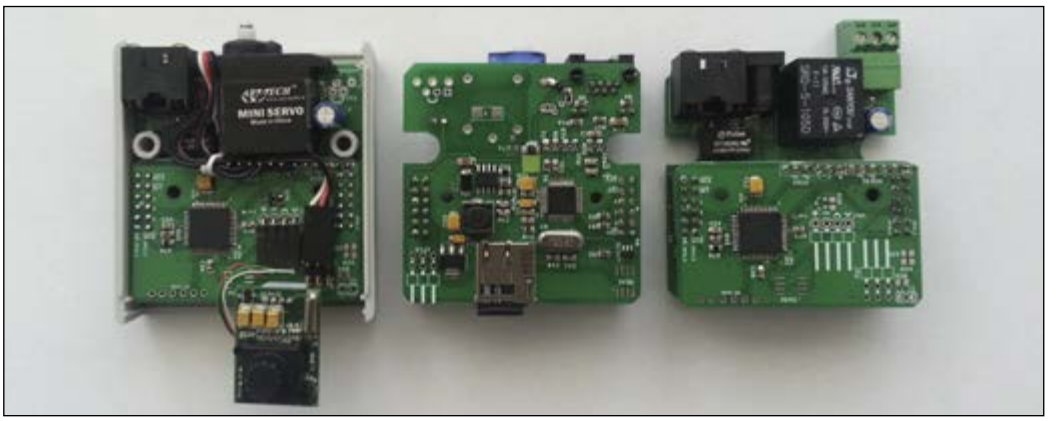


PHOTO 4
Three opened nodes: two with a camera (view from top and bottom) and a third with a relay instead of servo

we already noted, we can upgrade it without affecting the base.

Because the case had quite high mounting pads (4 mm in height), it was decided to use that space (under the board) and to mount all the components on the bottom, thus allowing the board that will come on top to be mounted very close to the base board if necessary. This indeed proved to be the case in order to fit the camera and PIR sensor.

We used a separate RJ-45 jack and magnetics because of the need to implement the passive PoE design. Normal Ethernet jacks with included magnetics do not provide access to pins 4, 5, 7, and 8, and the jacks specially designed for PoE are expensive.

In our rush to get the boards to the factory, we made some minor mistakes on the first version of the baseboard, but we fixed those using wires. One of the errors was to use a slightly different configuration near magnetics, which worked well with other Ethernet chips, but did not work here until the design was similar to that in the reference schematics from WIZnet.

The top board includes the microcontroller, the connection to baseboard, and other peripherals. Because of the time constraints, it was necessary to send it to the factory before we decided on all the peripheral connections. (The microcontroller contains re-mappable peripherals to different pins; but unlike other models, only a limited subset of options is available.) So the top PCB was laid out in more of a "live-bug" design (as opposed to "dead-bug"). Only the essential traces were connected to allow the microcontroller to start up properly. We added pads and connectors for programming, UART, servo and optional components remaining to make the connections to them with wires when the boards arrived (see **Photo 1**). The two PCBs were mounted on top of each other, with a spacing of 2 mm, which we implemented in case more interventions to the broad are necessary. The PIR sensor and the camera were mounted through holes on the top of the case. All the tall (vertical mounted) components from the PIR sensor module had to be resoldered, lying flat, in order to reduce the overall height of the sensor and get it to fit in the enclosure.

IMAGE ACQUISITION

The video camera used for image acquisition is a C328 module that can offer images in both compressed JPEG and raw format. This model was chosen because two of them were available on hand.

ABOUT THE AUTHORS

Claudiu Chiculita has an MS in Computer Science, a MS in Telecommunications and a PhD in Control Systems. His interests are in researching, teaching and development of embedded systems and software. You may reach him at claudiu.chiculita@gmail.com.

Liviu Ene has a BE in Applied Electronics and he is currently an MTech student in Advanced Informatics Technologies. His main area of interest is embedded systems. You may reach him at ene.liviu09@yahoo.com.

circuitcellar.com/ccmaterials

SOURCES

C328 JPEG Camera Module
COMedia Ltd. | www.comedia.com.hk

25AA02E48T Serial EEPROM and
PIC32MX250F128 microcontroller
Microchip Technology | www.microchip.com

L5973D 2.5-A Regulator
STMicroelectronics | www.st.com

W5500 Network chip
WIZnet | www.wiznet.co.kr

Divider(1)	Divider(2)	C328 baud	PIC32 baud	brgh=1 BRG(divider)
15	1	115200	117647	16
15	0	230400	125000	15
14	0	245760	133333	14
13	0	263314	142857	13
12	0	283569	153846	12
11	0	307200	166667	11
10	0	335127	181818	10
9	0	368640	200000	9
8	0	409600	222222	8
7	0	460800	250000	7
6	0	526629	285714	6
5	0	614400	333333	5
4	0	737280	400000	4
3	0	921600	500000	3
2	0	1228800	666667	2
1	0	1843200	1000000	1
0	0	3686400	2000000	0

TABLE 1
Fast data rates obtainable from camera and microcontroller

The feature that makes this camera popular, the communication through UART is also its biggest disadvantage, because of the slow speeds available, taking a long time to transfer an image. The maximum advertised baud rate between C328 camera and a master is 115,200 bps that approximates to roughly 10 KB per second. At this speed, a full JPEG image can take around 2 s to arrive and a raw image around 0.5 s, which is a long time for surveillance purposes.

In the quest to obtain a faster image transfer rate, we tried the camera with different data rates outside the range given in datasheet. Interfaced with a USB-to-UART FTDI adapter to the PC, the camera could be tested with almost any data rate (up to a limit). The camera will synchronize (autobaud) just fine when using data rates around 230,400 bps, but not when using significantly higher data rates. In order to obtain higher transfer speeds, the camera was first synchronized using 115,200 bps, and then a "Set baud rate" command was issued to change the data rate at a much higher speed by adjusting the two divisors (D1 and D2) using the following formula:

$$\text{Baud} = \frac{14.7456 \times 10^6}{2 \times (D2+1) \times 2 \times (D1+1)}$$

You can achieve faster transfer rates with this method. When the camera is connected to a microcontroller, a data rate can't be obtained in the PIC (as was the case with the USB-to-UART FTDI adapter).

Table 1 lists different data rates that can be obtained using the divisors and internal clock for both devices used (C328 and PIC32). For example, when a camera uses 115,200 bps (dividers 15 and 1), using a divider of 16 in microcontroller gives a data rate within 2% error, which is acceptable. An acceptable baud match at higher speeds is achieved only at around 400,000 bps (with a 2.4% error) using dividers 8 and 0 for camera and 4 for the microcontroller. The data rate difference could be higher due to the error of the internal clock of the PIC that is advertised to be less than 1%. Using this higher data rate configuration, images from camera can be transmitted about three times faster. Because problems can appear during the synchronization phase with the camera, the power to the camera can be switched on or off under software control from using a MOSFET.

FLEXIBLE DESIGN

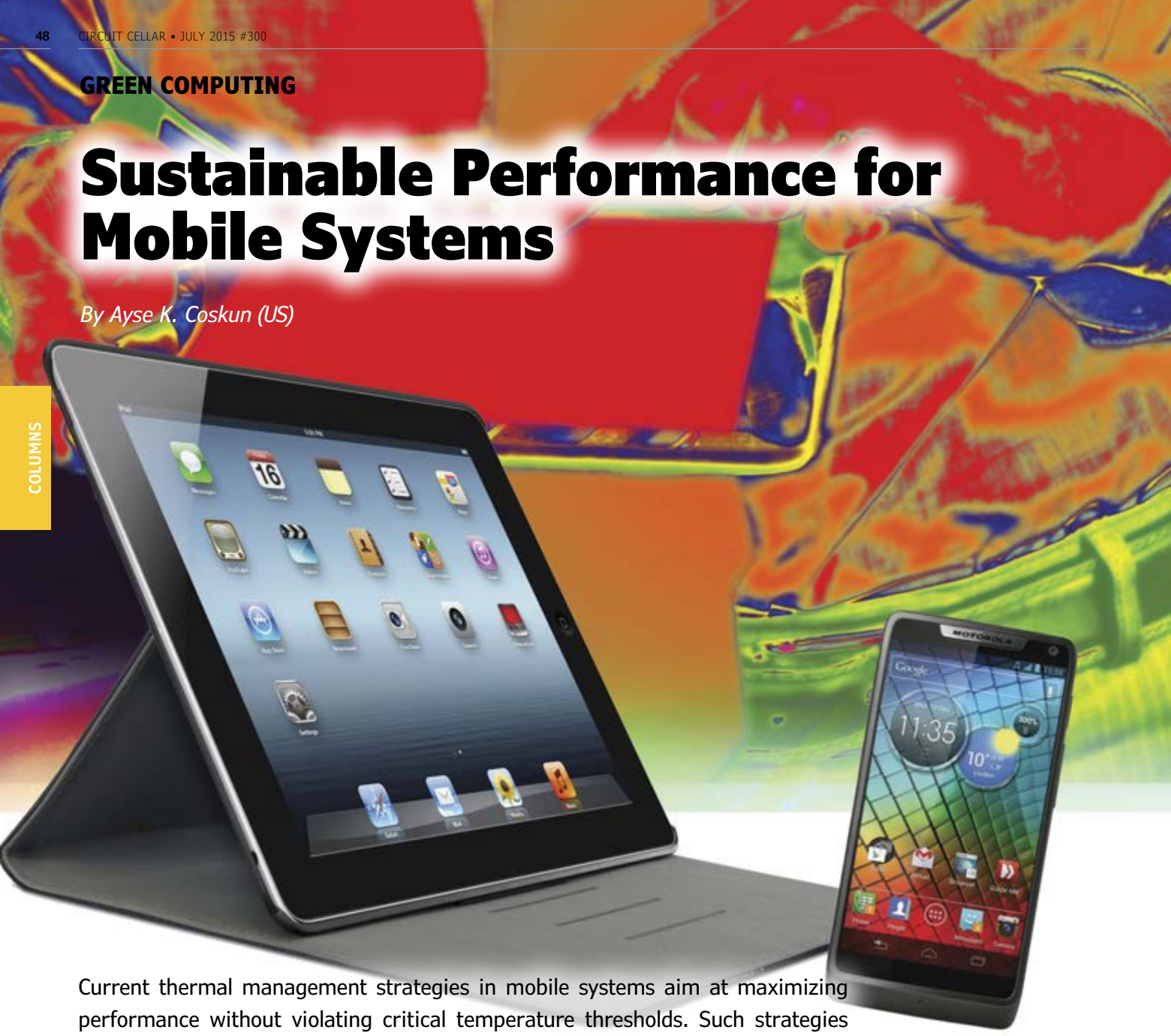
In this article, we presented the hardware and construction process associated with a compact device that connects to the Ethernet via a W5500 chip and is powered by passive PoE. Our design is flexible enough to be used in a variety of home area network systems (see **Photo 4**). The nodes' main functions are to enable a video camera to detect movement and record images and permit a servomotor to position the system toward an area of interest. They can communicate with each other and a central application. We'll detail the reset of the project in the second part of this article series.

GREEN COMPUTING

Sustainable Performance for Mobile Systems

By Ayse K. Coskun (US)

COLUMNS



Current thermal management strategies in mobile systems aim at maximizing performance without violating critical temperature thresholds. Such strategies may lead to significant losses of efficiency during longer periods of device use. This article discusses new approaches to provide "sufficient performance" to users for extended periods of time instead of maximizing instant performance.

User performance demands from mobile systems such as smartphones and tablet computers continue to grow with each generation of these systems. Many popular software applications already perform computationally heavy multimedia, artificial intelligence, or data processing tasks. In tandem, many mobile systems today include high-performance multicore CPUs, GPUs, accelerators, and other components to cater to the increasing performance demands.

This trend results in higher power densities and, in turn, higher chip and skin temperatures. As mobile systems are severely limited by cost and size constraints, it is generally not possible to include advanced cooling schemes in these devices (i.e., efficient heat sinks, fans, etc.). The system, however, still needs to operate under strict thermal limits because of reliability, leakage power, and usability constraints. Thus, efficient thermal management is an essential component of the hardware-software stack so as to maintain safe temperatures while providing the desired performance to users.

Operating system runtime managers (such as the ondemand governor in Linux[1]) help with power and energy management by setting processor frequency to lower levels when the processor is under-utilized.

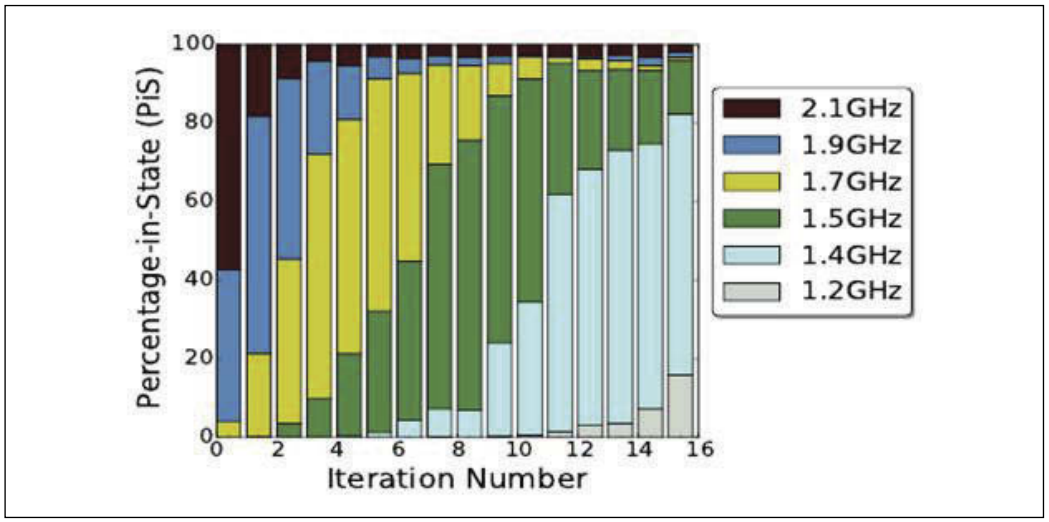


FIGURE 1
Frequency residencies over time on the MSM8974 smartphone during continuous use. In this experiment, the same Fast Fourier Transform (FFT) application is run repeatedly on the phone. The x-axis is the iteration number of each of the repeated runs. In the first few iterations, the default thermald policy allows the application to utilize the highest three frequencies to boost performance while meeting the thermal constraints. Over time (right side of the figure), there is a clear shift towards using lower frequencies, which significantly reduces performance. For instance, in the last iteration, more than 80% of the running time is spent at 1.4 GHz and 1.2 GHz.

On the other hand, when the system is actively used by computationally intensive applications, temperature of the CPU (or other components) can rise rather quickly. Upon reaching a threshold temperature value determined by the manufacturer, the runtime manager then starts "throttling" the CPU by enforcing lower dynamic voltage and frequency scaling (DVFS) settings to reduce the power, and in consequence, temperature. When temperature is at a safe level, the DVFS setting is increased to a higher value again.

The end result of such an approach is that during high-power, performance-demanding phases of operation, the CPU operates close to (i.e., right below or at) the thermal threshold. This outcome seems like a good indicator of getting the most performance out of the system; that is, a higher DVFS setting generally means faster operation and better performance, so operating right around the thermal threshold may seem to show the system is using the highest DVFS setting possible.

Figure 1 shows that the above intuition is not always true, especially when the mobile system is used for an extended period of time. The figure indicates that as one keeps running the same application over and over, the CPU is throttled more and more: this effect is easily visible as later iterations of the application on the right-side of the figure uses lower DVFS settings for a much larger portion of the time.

Why do different iterations of the same application, which have exactly the same dynamic power profile, result in different dynamic voltage and frequency scaling (DVFS) residencies? This is because the CPU and the system as a whole heat up over time, and the runtime manager has to apply more throttling

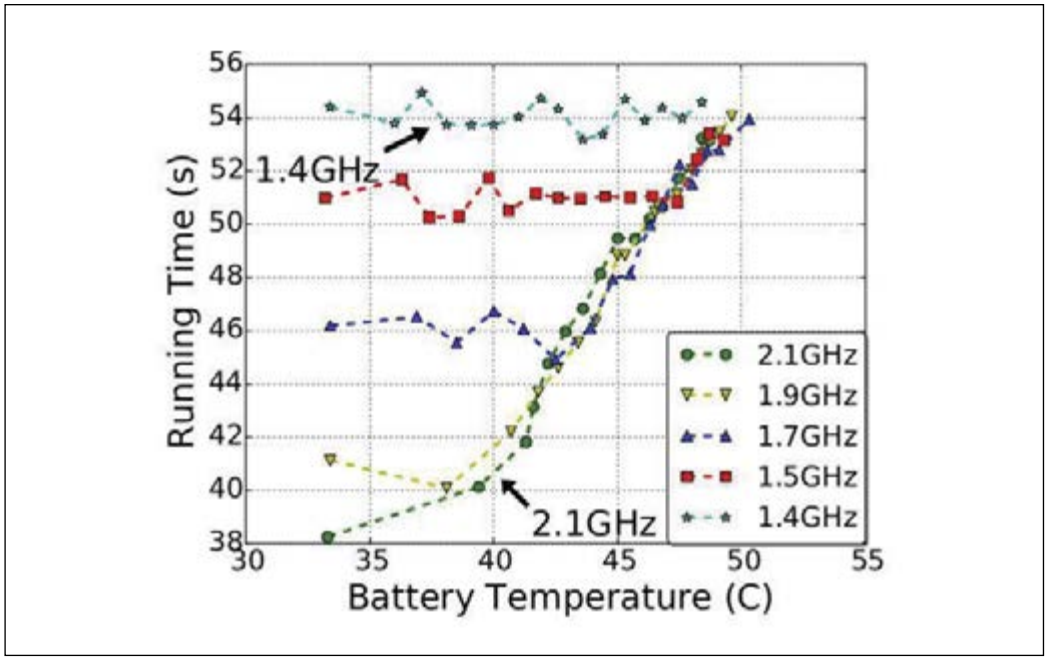


FIGURE 2
This plot shows how the running time changes as the battery temperature increases over time for the default PID controller. Each data point represents the initial battery temperature of that iteration and the resulting running time. Assume that 46 s (80% of maximum performance) is an acceptable running time for the user. Using 2.1 GHz as the maximum frequency limit greedily aims to maximize performance at every run but causes a rapid increase in battery temperature. Increased battery temperature causes more aggressive CPU throttling and running time increases sharply. When the maximum frequency is limited to 1.7 GHz, the application can run for a longer time before reaching 42.5°C, sustaining a running time of 46 s for six iterations (for 270 s).

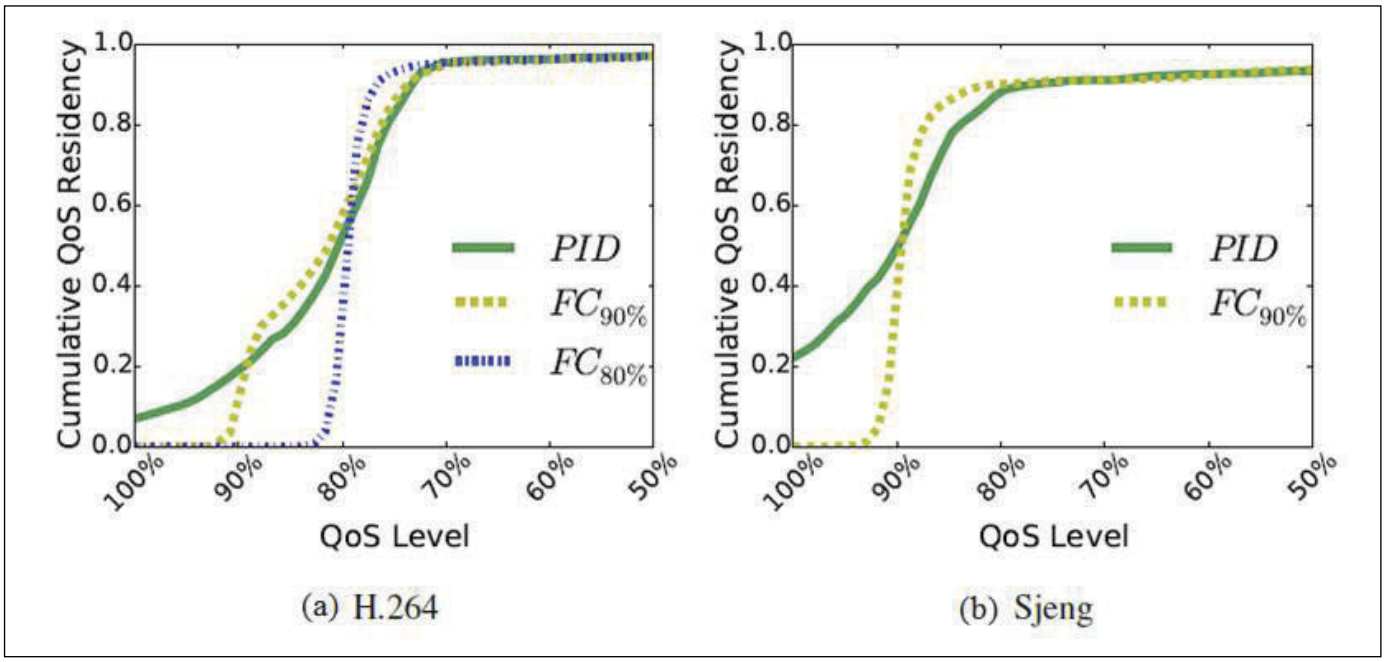


FIGURE 3
This plot shows the cumulative QoS residencies for two applications. A data point indicates the fraction of running time spent (y-axis) above the corresponding QoS level (shown on x-axis). In an ideal scenario, when the target QoS is 80%, 100% of the running time would be spent above the 80% QoS level. Our frequency capping policy, FC, achieves a much larger percentage of time spent above the target compared to the default PID controller.

to make sure the temperature safety thresholds are not exceeded. It is not only the CPU that heats up, but other components such as the GPU and the battery may heat up substantially. All these components that are located within the small form factor of the mobile system share their heat with each other to some extent. For example, **Figure 2** shows that as one runs the same application repeatedly, the battery temperature rises, the system throttles more, and the running time of the application increases as a consequence.

Why is the performance degradation over time a potential problem? Especially for interactive applications, the users typically expect consistent performance over time. For example, a user would play a game if the game is able to proceed with a reasonable speed or watch a video only if the frame rate is above an acceptable quality level. For such applications, if the user is provided with a few minutes of high performance initially and the performance drops to an unacceptable level afterwards, the user would most likely need to stop using the application.

Is it possible to do better in such scenarios? This article argues that, if one proactively manages the temperature by providing "just enough" performance instead of greedily maximizing performance under the thermal threshold, then heating slows down, allowing for sustaining sufficient performance levels for a much longer time. Going back to the application examples above, providing just enough performance means the user would be watching a video with an acceptable (but not the highest possible) frame rate. But this time, a much longer time period is elapsed before throttling starts to cause disruptive performance loss.

QoS TUNING FOR SUSTAINABLE PERFORMANCE

In my research lab, my students and I designed a proactive thermal management technique to address the sustainable performance issue.[2] The purpose of the technique is to provide sufficient quality-of-service (QoS) levels for software applications for as long as possible. QoS can be observed and controlled using a number of different metrics: instructions completed per second (which can be measured via performance counters in the hardware) or application-specific metrics (frames per second, bytes processed per second, etc., which can mostly be measured through instrumented application software or through the OS) are some example metrics.

Runtime managers in the OS often have a "frequency cap" that corresponds to the

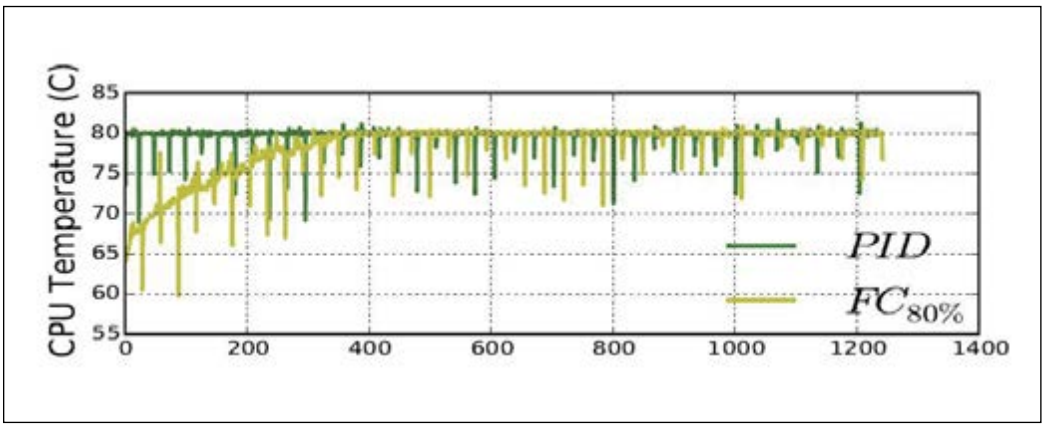


FIGURE 4
This is the CPU temperature for an SOR application with a target 80% QoS constraint. The baseline PID controller allows the CPU to greedily operate at maximum thermal limits, which induces throttling early on and degrades the performance over time. As a result, after 270 s, our frequency capping policy, FC, starts to provide better running time (not shown in the figure).

maximum DVFS setting allowed for that policy at a given time. Instead of directly modifying the DVFS setting of the CPU, our policy adjusts this frequency cap as a proxy. In this way, we let the ondemand governor (or other runtime policies) optimize the overall efficiency of the system and only interfere with the thermal limits.

The first proactive management policy we implemented is a rather straightforward one.

```
Perform the following steps to adjust fcap every period t:
  1. Measure QoS (e.g., instructions per second)
  2. if (QoS < QoStarget) then fcap++
  3. if (QoS > QoStarget) then fcap--
  4. Clip fcap such that fmin <= fcap <= fmax
```

In the pseudo-code above, `fcap` represents the frequency cap, `QoS` is the measured performance metric of choice, `QoStarget` is the desired performance set by the user or the system administrator, and `{fmin,fmax}` are the minimum/maximum DVFS settings available on the system.

It is possible to provide a "knob" to the user to adjust the desired QoS level up and down as needed rather than setting a fixed value throughout. Imagine adjusting the performance level manually up and down to reach an acceptable minimum performance level, in a similar fashion to adjusting the screen brightness or volume.

IMPLEMENTATION

In my research lab, we performed experiments and tested our technique on a Qualcomm Snapdragon MSM8974 smartphone platform. The chipset contains a Quad Core Krait 400 CPU (a common CPU in mainstream smartphones such as Nexus 5 and Samsung S4) along with an Adreno 330 GPU, 2GB LPDDR3 RAM and it is powered by a 1,600-mAh Li-ion battery. The phone runs Android Jelly Bean version 4.1.2 and Linux kernel 3.4.0. The Krait 400 CPU supports 12 operating frequencies ranging from 300 MHz to 2.1 GHz.

The phone has a set of temperature sensors, including a sensor on each core and a sensor on the battery. These sensors can be read through the virtual file system provided by the Linux kernel (i.e., /sys/class/thermal) with ±1°C accuracy. We used the perf_event API for accessing hardware performance counters to measure performance. We also monitored the overall power consumption of the phone using the internal sensors. To isolate CPU power consumption as much as possible, we disabled cellular activities and Wi-Fi by switching the phone into airplane mode and turning-off the LCD.

Our phone uses the ondemand governor as default, which is also the default policy in many state-of-the-art mobile devices. Thermal management in modern smartphones is typically carried out by a thermal-daemon (thermald)[3] that uses a PID controller to keep the temperature below a given temperature value by adjusting the maximum CPU frequency limits of the governor. As thermald is not open-source, we implemented our own DVFS-based PID controller as a user-level program, and tuned the controller's parameters empirically to mimic thermald.

circuitcellar.com/ccmaterials

REFERENCES

[1] V. Pallipadi and A. Starikovskiy, "The ondemand Governor," in Proceedings of the Linux Symposium, Vol. 2, 2006.

[2] O. Sahin and A. K. Coskun, "On the Impacts of Greedy Thermal Management in Mobile Devices," to appear in IEEE Embedded Systems Letters, 2015.

[3] Intel Open Source Technology Center, 2014, https://01.org/linux-thermaldaemon/documentation/introduction-thermal-daemon.

Recall that while the default PID controller aims to keep temperature below a given value, our frequency capping (FC) policy aims to keep QoS at a given target (instead of temperature). In both cases, the ondemand governor is still in use but operates adhering to the frequency caps set by the thermal management policy (PID or FC).

We configured our frequency capping policy such that the algorithm steps above are repeated every 100ms (period t = 100 ms). This granularity resulted in less than 1% performance overhead on the system and allowed for effectively managing potential thermal emergencies. t can be adjusted for other systems depending on the system thermal time constants.

ABOUT THE AUTHOR

Ayse K. Coskun (acoskun@bu.edu) is an associate professor in the Electrical and Computer Engineering Department at Boston University. She received MS and PhD degrees in Computer Science and Engineering from the University of California, San Diego. Coskun's research interests include temperature and energy management, 3-D stack architectures, computer architecture, and embedded systems. She worked at Sun Microsystems (now Oracle) in San Diego, CA, prior to her current position at BU. Coskun serves as an associate editor of the IEEE Embedded Systems Letters.

RESULTS

Figure 3 shows a comparison of our QoS-aware frequency capping policy against the PID controller (which represents the default thermald policy) for a video encoding application, H.264, and an artificial intelligence application, Sjeng. Let us assume the user performance demand is such that the application should run with at least 80% of the maximum QoS achievable by that application on the given phone. In this case, ideally, 100% of the cumulative QoS residency would be at the target 80% level or higher. The PID controller, however, is able to hold the QoS above the 80% target only about 60% of the time. Our frequency capping policy extends this period by over 30%. From the user's perspective, this would correspond to a significantly longer time a certain video quality could be sustained without causing disruptive performance drops. Using the frequency capping policy with a QoS target of 90% extends the duration of time spent above this QoS level from 57% to 72% for the Sjeng artificial intelligence application.

Figure 4 shows the reasoning behind how our frequency capping policy achieves sustainable performance. By letting the application run slower but still above the required QoS level initially, our policy delays throttling substantially (by 400 s in this case, where a scientific computing kernel, Jacobi Successive Over-relaxation (or SOR), is run repeatedly). Even after throttling starts, our policy achieves lower application running time compared to the default PID controller (thus, higher performance) as it achieves lower battery temperature and results in less aggressive throttling.

FUTURE DIRECTIONS

In my lab, we recently started investigating more advanced policies after the highly promising results of the policy I described above. In addition to testing and enhancing the policy for a wider range of software applications (e.g., GPU applications) and thermal constraints (e.g., skin temperature constraints in addition to CPU constraints), we are also working on DVFS scheduling methods to further improve thermal efficiency. For example, instead of iteratively increasing or decreasing the frequency cap, it is possible to set average frequency for the next interval through a controller, and then schedule DVFS states intelligently during the interval to reach that average frequency while minimizing the temperature rise.

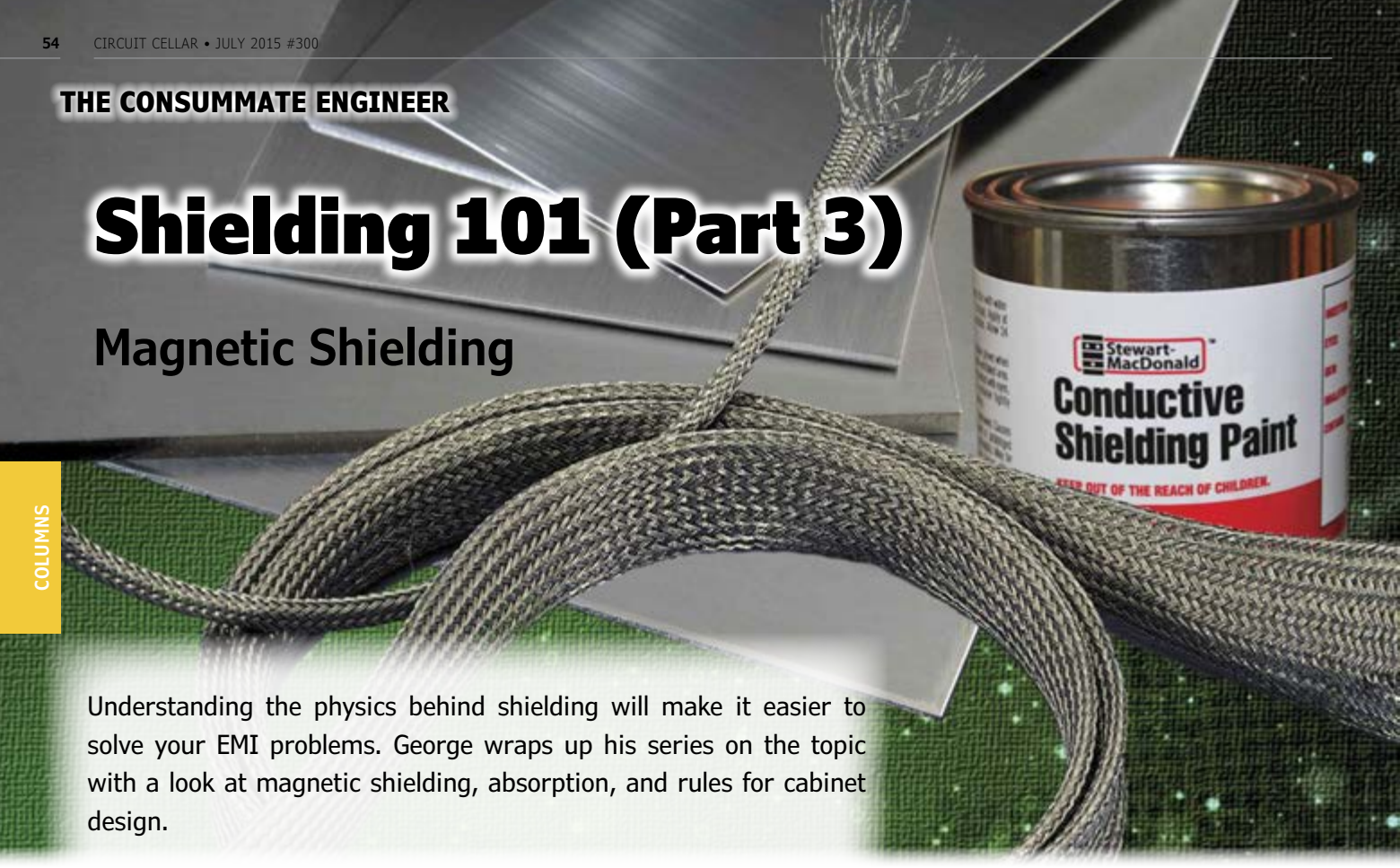
Open problems in modern smartphone thermal management also include taking into account the performance and thermal coupling between the CPU, GPU, and other components. Performance coupling refers to the fact that DVFS actions taken on the CPU impact the GPU performance as well (and vice versa); thus, for applications with heavy GPU use, this coupling effect needs to be considered by the runtime management policy. In addition, thermal coupling occurs as CPU and GPU are often in close proximity, and therefore, share some of their heat with each other. A hotter GPU may cause the CPU to heat up faster.

While there are quite a few benchmark suites available online (especially for use during performance evaluation), creating a collection of well-instrumented, open-source, automated benchmark applications representing real-life scenarios is still an open problem. This is particularly the case as runtime optimization of mobile systems is highly application dependent and maintaining the quality of the user experience is essential. Future energy and thermal management schemes will need to consider not only the hardware features and limits, but also the application behavior and user constraints.

THE CONSUMMATE ENGINEER

Shielding 101 (Part 3)

Magnetic Shielding



COLUMNS

Understanding the physics behind shielding will make it easier to solve your EMI problems. George wraps up his series on the topic with a look at magnetic shielding, absorption, and rules for cabinet design.

By George Novacek (Canada)

In this last part of the series on shielding, we'll take a closer look at magnetic shielding. When low-frequency magnetic field sources—such as power cables, transformers, motors and relays—are close to equipment sensitive to magnetic fields—the likes of CRT tubes, compass, electron microscopes, and so forth—magnetic shielding may become unavoidable. As we have seen in the previous parts of this series, absorption is the best approach to tame the interfering AC magnetic field.

Absorption is a function of permeability. Thus permeable materials, such as iron, nickel and their alloys have been instrumental in building magnetic shields. Besides absorption, these materials also feature low magnetic reluctance (a characteristic of magnetic circuits analogous to resistance in electrical circuits), so that the lines of magnetic flux flow within the shield rather than through the air within the shielded volume. A typical shielding could be a conduit made of steel or another permeable material.

SHIELD EFFECTIVENESS

Considering a conduit pipe that is a cylindrical shield, the magnetic field inside it is as follows:

$$H_i = H_0 \times \frac{R_S}{R_S + R_0} \quad [1]$$

H_i is the magnetic field inside the conduit. H_0 is the magnetic field outside the conduit. R_s is magnetic reluctance of the conduit shield. R_0 is magnetic reluctance of air. In addition:

$$R = \frac{s}{\mu \times A} \quad [2]$$

s is the length of the magnetic path.

$$\mu = \mu_0 \times \mu_r \quad [3]$$

μ_r is relative permeability of the shield material. A is the area through which the flux flows. As a result, the reluctance of a conduit pipe is:

$$R_S = \pi \times \frac{b}{\mu 2ts} \quad [4]$$

s is the length of the conduit pipe. b is its outer diameter. t is the thickness of the wall. If, for simplicity, we assume a square cross-section of the conduit and the $R_s << R_0$, the shielding effectiveness SE is:

$$SE = 2\mu_r \times \frac{t}{\pi b} \quad [5]$$

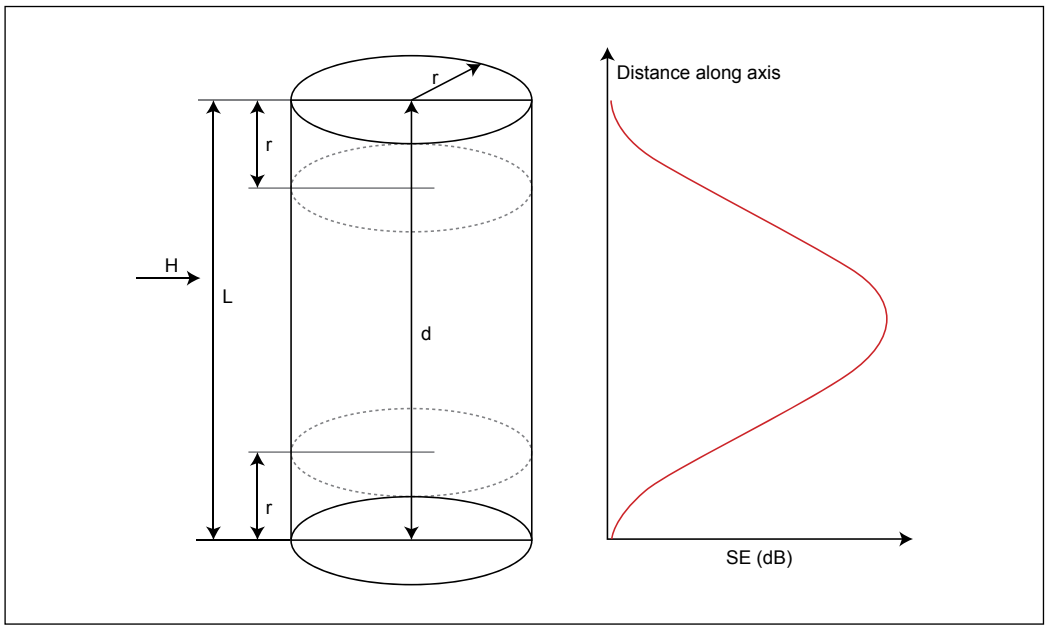


FIGURE 1
Open-ended magnetic shield

It is readily apparent that the greater the conduit thickness and its material permeability, the greater its shielding effectiveness. But it can also be seen that as the protected volume decreases—that is, the outer diameter b is reduced—SE increases as well. It means that the shield should be as close as possible to the protected component for its volume must be minimized. We have seen it, for example, in old oscilloscopes with

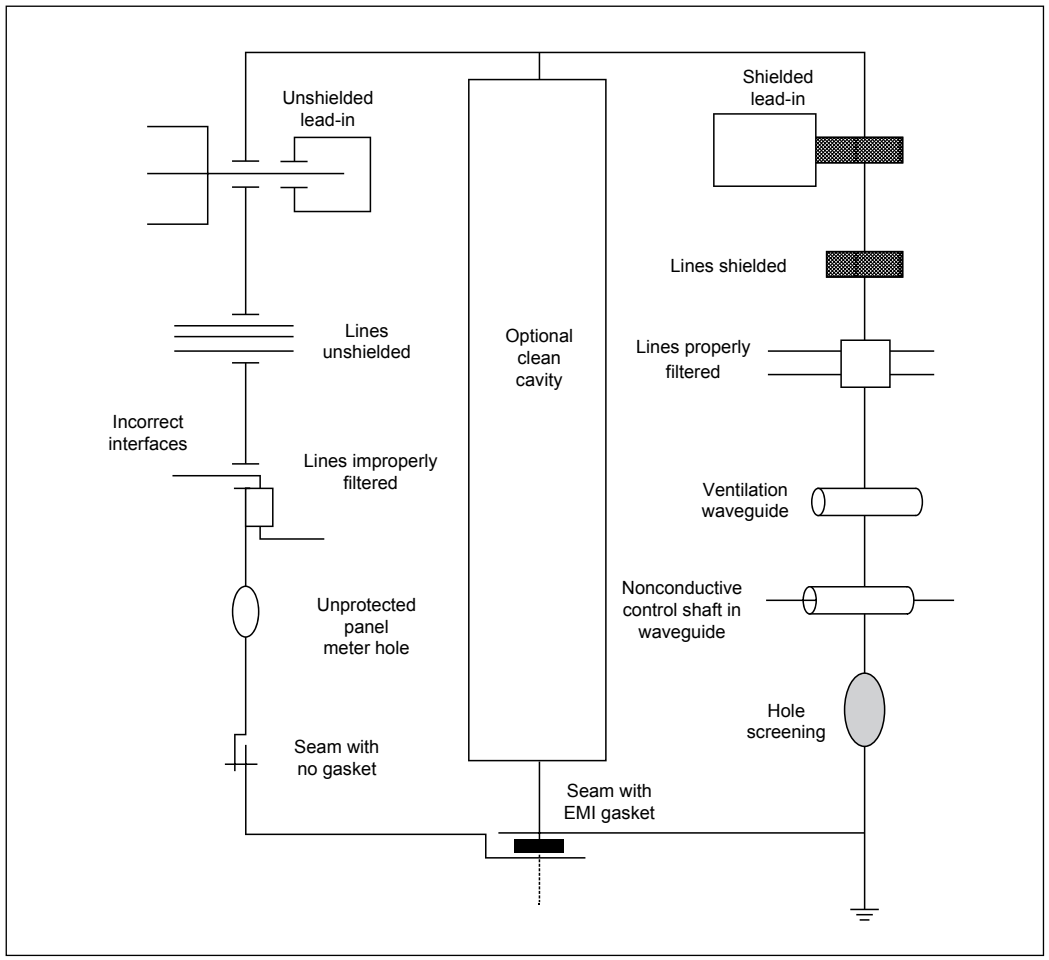


FIGURE 2
General rules for cabinet design

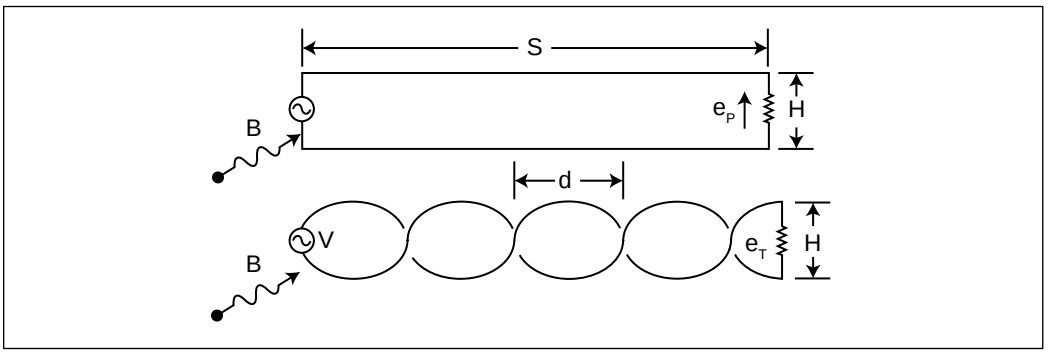


FIGURE 3
Magnetic field induction reduced by wire twisting

cathode ray tubes (CRT) where the metal shield fits very closely around the tube.

The magnetic shields are made of high-permeability materials, typically alloys of iron and nickel. To select a material, you must make a trade-off based on the saturation flux, cost, and the needed shielding effectiveness. For example, an alloy consisting of 80% nickel and 20% iron saturates at 8,000 Gauss (Gs). Its permeability reaches 400,000 and the shielding effectiveness of a 0.02" (0.5 mm) layer at DC is –100 dB. Carbon steel of the same thickness, on the other hand, saturates at 22,000 Gs, but its permeability is merely 1,000 and the shielding effectiveness –58 dB (i.e., 126× less attenuation). You need to remember, however, that SE at DC does not include any absorption because the skin depth at DC is infinite. As the frequency increases, the absorption comes into play.

The permeable shielding efficiency increases with frequency, but this trend ends around 100 kHz, after which the SE begins to decrease rapidly. Therefore, permeable shields should not be used for frequencies above 100 kHz as they become essentially useless.

As you can see in Equation 5, shielding efficiency increases with the thickness of the shield. This presents another engineering trade-off. A thick shield can be replaced with a number of thin shields nested within each other with an insulating material in between. This may be less expensive and weigh less than a single thick shield.

A dilemma occurs when we need to magnetically shield a device which cannot be completely enclosed, such as a CRT. Obviously, some loss of shielding efficiency will occur. A compromise can be attained when the length to width ratio of the shield is at least 4:1 as is shown in **Figure 1**. The shielded device should be placed at least one radius r inside the tube. The trace in Figure 1 shows the shielding efficiency relative to the location within the length d of the tubular shield.

ENCLOSURE DESIGN

Let us turn our attention to the design and building of electronics enclosures in general. There are several rules one needs to follow to be successful. They are best explained by Figure 2. On the left side of the imaginary enclosure are several examples of what not to do. Simply put, you must take care not to destroy the integrity of the shielding provided by the cabinet. Needless to say, the cabinet must also be bonded to the chassis ground. Any discontinuity between the cable shields and the cabinet or the loss of shielding integrity of the cabinet itself due to seams, unshielded wires, holes and so forth greatly reduce if not completely destroy the shielding effect up to the point as if there was no shielding at all.

The right side of **Figure 2** shows how to do it. Seams, including lids, need to have gaskets, shielding must be bonded to the case, holes screened, or designed as waveguides.

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.

circuitcellar.com/ccmaterials

RESOURCES

O. Hartal, *Electromagnetic Compatibility by Design*, R&B Enterprises, 1995.

G. Novacek, "Essential Electromagnetic Compliance," *Circuit Cellar* 293–296, 2014–2015.

———, "Impedance Matching," *Circuit Cellar* 281, 2013.

There is also glass available with conductive properties to cover holes needed to view an internal display, for example. In extreme environments, this may not be sufficient. In those cases an electrically clean cavity, bonded to the box, containing the sensitive electronics is provided within the box. Inputs and outputs to or from it are conditioned in the "dirty" cavity of the cabinet and thus cleaned signals pass to the clean cavity via feed-through filters.

WIRE TWISTING

To conclude this series on shielding, I would like to mention an effective technique for the suppression of signals induced by low frequency magnetic fields, even though it is not related to shielding. As I mentioned in Part 2 of this series and other articles, the technique involves twisting the wires.

As is well known, a field emitted from a wire loop or voltage induced in a loop from a magnetic field is proportional to the area of the loop. The same consideration applies to suppression of ground loops or reduction of emission or susceptibility of circuits on printed circuit boards.

You can use the wire twisting method for balanced as well as unbalanced interfaces, as long as the latter uses a wire pair for the signal and its return. For S and H much lower than λ, the condition is illustrated by **Figure 3**.

$$\text{If } S \text{ and } H << \lambda,$$
$$\text{then } e = A \times \frac{dB}{dT} \qquad [6]$$

In Equation 6, A is the loop area in meters squared. B is magnetic flux density in Teslas (T). e_P stands for induced voltage in the parallel wires and e_T in the twisted wires. It can be concluded that induced voltages in parallel e_P lines and twisted e_T lines are proportional to SH.

In a twisted pair, only one loop is active. All loop pairs' voltages cancel each other out as they are 180° apart. Suppression of the magnetically induced signals by twisting the wire pair as compared to no twisting would be mathematically the total number of twists in the connection. The practical attenuation limit up to about 100 kHz is roughly 1/1,000, or –60 dB.

This concludes the series on shielding. Understanding the physics behind it will make your life easier when you're faced with solving EMI problems.

ABOVE THE GROUND PLANE

A Touch-Screen LCD Control Panel

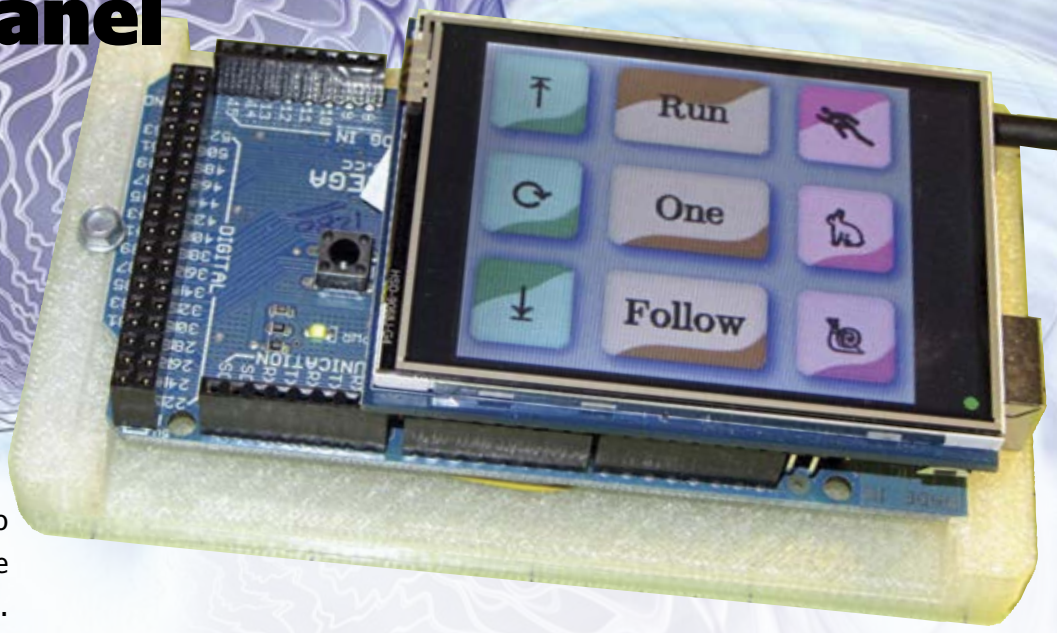


PHOTO 1
The Kenmore 158 sewing machine UI runs on an Arduino Mega with a Adafruit TFT LCD and a resistive touch-screen overlay. The cable connects a serial port to the Arduino Pro Mini controlling the motor and brings preregulated +7 VDC power from the power supply. These skeuomorphic buttons didn't look nearly as attractive as I expected!

Turning a user interface into a simple matter of software requires writing the software. Ed takes us behind the screen of his sewing machine controller.

By Ed Nisley (US)

Most of the software I write hides inside microcontroller projects, with a "user interface" only an engineer could love: configuration jumpers, a few LEDs, and perhaps a button or two. As you've seen in the last several columns, the Arduino Pro Mini controlling the Kenmore 158 sewing machine's motor has exactly that style of UI, but I knew from the start this project required an interface better suited for an actual human: my wife!

In this column, I'll describe the logic behind the sewing machine's UI, explain how I produced the images for the graphic LCD, then show how it all fits together.

SIMPLIFYING THE UX

My Kenmore 158 retrofit project started when Mary asked me to improve its graphite-rheostat foot pedal to stabilize low speeds and eliminate the hotfoot caused by power dissipation in the pedal. She also wanted to control whether the machine stopped with the needle fully down (to allow pivoting the quilt around the needle) or fully up (to allow moving from one part of the quilt to another). She did *not* want a contemporary plastic sewing machine with fancy features, because she doesn't do that type of sewing. She prefers a familiar setup and definitely didn't want to fight a modern UI to accomplish simple tasks.

I considered building a button-and-LED interface, but quickly realized that a fat control cable attached to an ugly box wouldn't be appropriate, particularly because we might add, delete, or rearrange buttons and indicators as we tweaked the controller's functions. Rather than machining holes for physical switches and indicators, I turned the entire UI into a program controlled by a touch-screen graphic LCD. The buttons became images activated by a finger press, with the compelling advantage that the UI's layout, appearance, and function became a simple matter of software.

Photo 1 shows my first pass at the controller UI, with three columns of buttons selecting the operating modes. Pressing a button activates that mode, changes the images to indicate the new state, and sends the corresponding command to the motor controller.

The LCD board, an Adafruit 2.8 inch TFT Touch Shield for Arduino with Resistive Touch Screen, includes a MicroSD card slot that can hold image files, which turns out to be essential for Arduino applications.

Although it can plug into Arduino boards based on ATmega328 microcontrollers, I used an Arduino Mega board from my collection to get more Flash and RAM storage, plus additional serial port hardware.

Although Adafruit also offers a capacitive touch screen, the sewing machine controller requires a resistive sensor, because Mary often wears cotton quilting gloves with non-slip silicone rubber dots on the fingers. As many phone users in wintery climates have discovered, capacitive screens do not respond well to gloved fingers. Adafruit's resistive and capacitive touch sensors have single-point response, which works perfectly to simulate buttons.

Now, if you think those button images look both ugly and cryptic, so did Mary. **Photo 2** shows the second version, with the same functions in the same places, but sporting different colors in a simpler graphic design. We're continuing to refine the UI without hardware modifications and she's finished several quilts in the process.

MICROCONTROLLER CONSTRAINTS

Even the largest members of the Atmel ATmega microcontroller family have severely restricted program and data memory capacities, because they're based on an eight-bit architecture predating the start of this millennium. Although you can write intricate C programs without touching assembly language, the microcontrollers simply don't allow access to the vast expanses of memory we take for granted in contemporary hardware: a laptop PC may have a billion times more memory and even phones contain a million times more RAM.

The LCD screen has 320 × 240 pixels, each one capable of displaying 262,144 colors from 6 bits of red, green, and blue. The LCD controller handles screen refreshing from an internal 172,800 byte graphics RAM that's 21 times larger than the Arduino Mega's 8 KB and roughly the same size as the Flash memory holding the program. Fairly obviously, the microcontroller cannot store even a single screen in its internal memory.

In fact, the microcontroller can't store a single button image. The smaller, square button images, just 80 pixels on a side, would require twice the amount of RAM available in the Mega. That's why the LCD shield includes a MicroSD card slot.

Adafruit provides Arduino libraries that handle file I/O from a FAT filesystem on the MicroSD card, read graphic files in uncompressed BMP format, convert BMP images to the LCD controller's 5-6-5 binary format (thus limiting the display to 64 K colors), and send them to the controller. Indeed, their code includes a `bmpDraw()` function that produces a rectangular image on the LCD at a specified location, given the name of the BMP file on the MicroSD card.

The inner loop of that function reads 20 pixels from the BMP file into RAM, converts each pixel from three bytes of BMP data to two bytes of LCD data, then sends the results to the LCD controller. Each iteration requires 60 to 70 µs, which means displaying a single 80x80 pixel button takes nearly half a second and transferring a full-screen BMP image requires four seconds.

It's fairly obvious that an 8-bit microcontroller can't deliver the performance we've come to expect from 32- and 64-bit microprocessors equipped with wide peripheral buses, huge memories, and specialized hardware. The fault doesn't lie with Adafruit's software, it's inherent in the nature of the hardware.

That notwithstanding, the Arduino-driven LCD panel works well enough for a control panel replacing an array of buttons and LEDs. It certainly wouldn't be suitable for a more complex control panel requiring rapid full-screen updates, but that's not what I needed.

CHARACTERS AND GRAPHICS

Because the buttons act as both switches and indicators, each button requires two images: "enabled" and "pressed." I also included a "disabled" button image, even though none of the controls shown here require that state. The prospect of drawing 27 separate images for each UI iteration convinced me to find an easier way.

The first nine lines of the `mkAll.sh` Bash script in **Listing 1** define the file name and

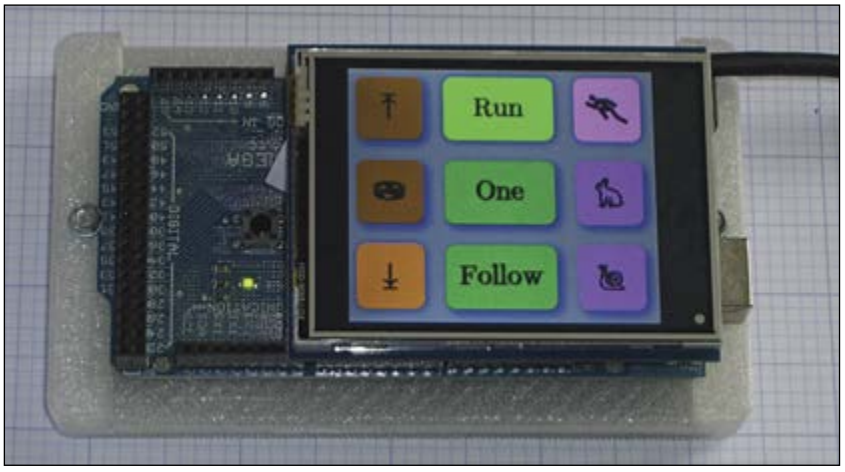


PHOTO 2
Simplifying the buttons made them more usable. A MicroSD card on the TFT display board holds the button images as BMP files, so copying the new files to the SD card updated the entire display with no program changes.

```
./mkBFam.sh NdDn  9 ⍊
./mkBFam.sh NdUp  9 ⍑
./mkBFam.sh NdAny 9 ⛀ 80 80 40
./mkBFam.sh PdOne 33 One 120 80
./mkBFam.sh PdFol 33 Follow 120 80
./mkBFam.sh PdRun 33 Run 120 80
./mkBFam.sh SpMax 83  80 80 40
./mkBFam.sh SpMed 83  80 80 40
./mkBFam.sh SpLow 83
montage *bmp -tile 3x -geometry +2+2 Buttons.png
display Buttons.png
```

LISTING 1
This Bash script calls the code in Listing 2 to produce the button image files. Unicode characters produce the button glyphs without requiring any drawing ability, but you must have a font with extensive Unicode support.

color of each button, the text appearing on it, and its size and position. The `mkBFam` script, which I'll describe below, produces a family of three image files for each button. The `montage` program, part of the ImageMagick suite, then collects all 27 button files into the image shown in **Photo 3**.

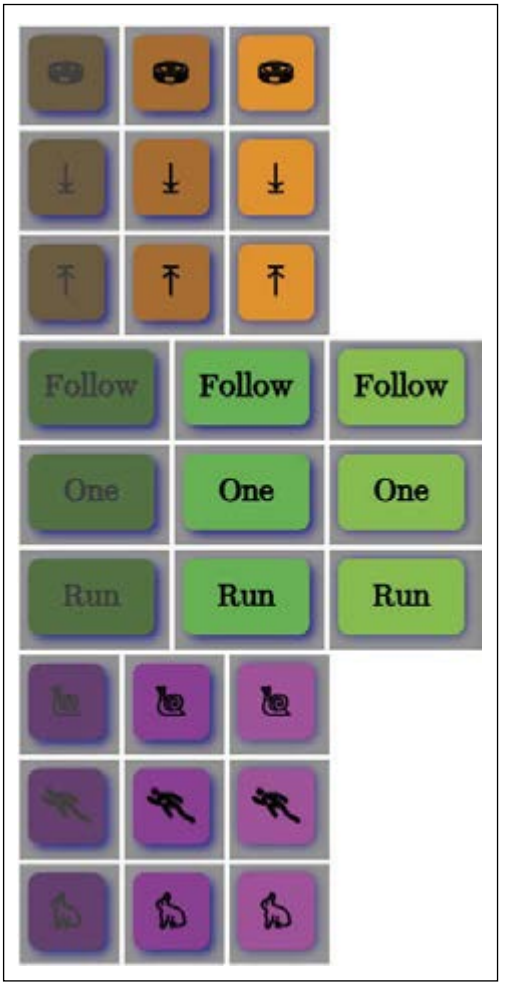


PHOTO 3
Each button can be Disabled, Enabled, or Pressed, with the UI presenting a different image file for each state.

The color number specifies the button's hue as a percentage of the usual 0° to 360° range: a hue of 33% = 120° represents pure green. I used the HSB (Hue, Saturation, Brightness) color model, rather than the more familiar RGB (Red, Green, Blue) model, so that the mkBFam script could vary the Saturation and Brightness to produce three visually distinct buttons based on the same color.

ImageMagic can process Unicode characters entered on the command line, but it took a while to find a font supporting all those characters. The Symbola font (see Sources and Resources) probably includes everything you want, although I was surprised to discover that Unicode lacks any sewing-related glyphs: the disk in the first line that represents a bobbin is actually a "White Draughts Man," which we in the USA would call a "White Checkers Piece."

The `mkBFam.sh` Bash script in **Listing 2** calls the ImageMagick `convert` program twice to create each button image, adding a numeric suffix to the file name stem to identify the images:

stem0 = disabled
stem1 = enabled, not pressed, "up"
stem2 = pressed, "down"

The file names must obey the FAT file system's 8.3 length restriction, which means the name stem cannot exceed seven characters.

The HUE parameter controls the button color, with each of the three images having different Saturation, Brightness, and text color settings. The final `for` loop converts the PNG files into the uncompressed BMP format required by the Adafruit code.

Even in uncompressed form, the image files occupy only 648 kB, leaving the 2 GB MicroSD card 99.96% empty. Microcontrollers simply aren't equipped to handle the volume of data we take for granted these days!

OPERATING THE BUTTONS

The Arduino code in **Listing 3** creates the `Buttons` array of structures containing all the information required to read the button image files, position them on the display, and control their operation.

The `NameStem` field contains the file name stems, which must match those used to generate the image files in Listing 1. The image size, given by the `szX` and `szY` fields, must also match the corresponding button. In principle, the program could extract those values by reading the BMP file header for each image, but I decided to leave that for a later enhancement. You can rearrange the

```
# create family of button images
# Ed Nisley - KE4ZNU
# March 2015

[ -z $1 ] && STEM=Test || STEM=$1
[ -z $2 ] && HUE=30    || HUE=$2
[ -z $3 ] && TXT=x     || TXT=$3
[ -z $4 ] && SX=80     || SX=$4
[ -z $5 ] && SY=80     || SY=$5
[ -z $6 ] && PT=25     || PT=$6
[ -z $7 ] && BDR=10    || BDR=$7

echo STEM=$STEM hue=$HUE txt=$TXT sx=$SX sy=$SY pt=$PT bdr=$BDR

echo Build Buttons
echo  .. Disabled
convert -size ${SX}x${SY} xc:none \
  -fill hsb\(${HUE}%,50%,40%\) -draw "roundrectangle $BDR,$BDR $((SX-BDR)),$((SY-BDR)) $((BDR-2)),$((BDR-2))" \
  ${STEM}_s.png
convert ${STEM}_s.png \
  -font /usr/share/fonts/custom/Symbola.ttf  -pointsize ${PT}  -fill gray20  -stroke gray20 \
  -gravity Center  -annotate 0 "${TXT}"  -trim -repage 0x0+7+7 \
  \( +clone -background navy -shadow 80x4+4+4 \) +swap \
  -background snow4  -flatten \
  ${STEM}0.png

echo  .. Enabled
convert -size ${SX}x${SY} xc:none \
  -fill hsb\(${HUE}%,100%,70%\) -draw "roundrectangle $BDR,$BDR $((SX-BDR)),$((SY-BDR)) $((BDR-2)),$((BDR-2))" \
  ${STEM}_s.png
convert ${STEM}_s.png \
  -font /usr/share/fonts/custom/Symbola.ttf  -pointsize $PT  -fill black  -stroke black \
  -gravity Center  -annotate 0 "${TXT}"  -trim -repage 0x0+7+7 \
  \( +clone -background navy -shadow 80x4+4+4 \) +swap \
  -background snow4  -flatten \
  ${STEM}1.png

echo  .. Pressed
convert -size ${SX}x${SY} xc:none \
  -fill hsb\(${HUE}%,100%,100%\) -draw "roundrectangle $BDR,$BDR $((SX-BDR)),$((SY-BDR)) $((BDR-2)),$((BDR-2))" \
  ${STEM}_s.png
convert ${STEM}_s.png \
  -font /usr/share/fonts/custom/Symbola.ttf  -pointsize ${PT}  -fill black  -stroke black \
  -gravity Center  -annotate 0 "${TXT}"  -trim -repage 0x0+7+7 \
  \( +clone -background navy -shadow 80x4+4+4 -flip -flop \) +swap \
  -background snow4  -flatten \
  ${STEM}2.png

echo Create BMPs
for ((i=0 ; i <= 2 ; i++))
do
  convert ${STEM}${i}.png -type truecolor ${STEM}${i}.bmp
# display -resize 300% ${STEM}${i}.bmp
done

rm ${STEM}_s.png ${STEM}?.png

echo Done!
```

buttons by changing the `ulX` and `ulY` fields that define the location of the upper-left image corner.

The numeric value of the `Status` field determines which of the three images should appear on the LCD, with convenient labels defined by the `bstatus_t` enumeration.

The `pAction` field contains a pointer to the routine that performs whatever action the button represents. All of the buttons use the default handler shown in **Listing 4**, which updates the button image and sends the `Cmd` string to the motor controller.

Because only the `Status` value changes during the program's execution, the other fields could be stored in the microcontroller's EEPROM memory to reduce the RAM requirement. The program could read those values from a configuration file on the MicroSD card and update the EEPROM if they differ. That would remove all the file-related values from the source code, so that the entire UI could be reconfigured without recompiling and reloading the program.

HOMEWORK: MAKE IT SO!

Each column of buttons in Photos 1 and 2 controls a separate function: needle stop position, pedal function, and maximum speed. Only one button can be active in each column, so they operate in what's called "radio button" mode, in homage to the old car radios where pressing one station selection button made another pop out with a satisfying mechanical *clunk*.

Buttons with the same nonzero value in the Group field of Listing 3 act as a set of radio buttons, implemented by the `if (Group)` conditional in Listing 4. Given `BID`, the button ID of the just-touched button image, the code determines if it's already pressed, in which case no other button image

LISTING 2
This Bash script uses Imagemagick's convert program to produce three image files for each button, creating the shape with the given color, then adding the text and a shadow. The three variations share the same Hue, with different Saturation, Brightness, and text color. The for loop converts the PNG files to the BMP format required by the Arduino library routine.

LISTING 3
The array entry for each button position contains everything the UI needs to read the corresponding image file and place it on the LCD screen, then match touch-screen coordinates to buttons.

```
#define MAXCMDLEN 2

enum bstatus_t {BT_DISABLED,BT_UP,BT_DOWN};

typedef void (*pBtnFn)(byte BID);  // button action function called when hit

struct button_t {
    byte ID;                       // button identifier, 0 unused
    byte Group;                    // radio button group, 0 for none
    byte Status;                   // button status
    word ulX,ulY;                  // origin: upper left
    word szX,szY;                  // button image size
    pBtnFn pAction;                // button function
    char Cmd[MAXCMDLEN + 1];       // command string
    char NameStem[9];              // button BMP file name - stem only
};

struct button_t Buttons[] = {
    { 1,    1, BT_UP,      0,0,      80,80, DefaultAction, "Nu", "NdUp"},
    { 2,    1, BT_UP,      0,80,     80,80, DefaultAction, "Na", "NdAny"},
    { 3,    1, BT_DOWN,    0,160,    80,80, DefaultAction, "Nd", "NdDn"},

    { 4,    2, BT_DOWN,   80,0,     120,80, DefaultAction, "Pr", "PdRun"},
    { 5,    2, BT_UP,     80,80,    120,80, DefaultAction, "P1", "PdOne"},
    { 6,    2, BT_UP,     80,160,   120,80, DefaultAction, "Pf", "PdFol"},

    { 7,    3, BT_DOWN,  200,0,      80,80, DefaultAction, "Sh", "SpMax"},
    { 8,    3, BT_UP,    200,80,     80,80, DefaultAction, "Sm", "SpMed"},
    { 9,    3, BT_UP,    200,160,    80,80, DefaultAction, "Sl", "SpLow"},
};

byte NumButtons = sizeof(Buttons) / sizeof(struct button_t);
```

will change. However, if the button was up (and will now be down), the code scans all of the other buttons to find members of this radio button group, change their state, redraw them from the corresponding image file, and then change this button's state to show that it's down. Although all of the buttons currently belong to groups, the code also handles a non-grouped button by toggling its state between up and down.

Because drawing the button images requires so much time, the computation required for the UI simply vanishes in comparison. In fact, because the redrawing

LISTING 4
Given the ID of the button matching the screen touch, this routine updates the other images in the button group, then sends the button's command to the motor controller.

```
void DefaultAction(byte BID) {

byte i,BX;
byte Group;

    if (!BID) {                                 // not a valid ID
        return;
    }

    BX = FindButtonIndex(BID);
    if (BX == NumButtons) {                     // no button for that ID
        return;
    }

    if (Buttons[BX].Status == BT_DISABLED) {    // cannot do anything to disabled buttons
        return;
    }

    Group = Buttons[BX].Group;

    if (Group) {                                // member of group?
        if (Buttons[BX].Status == BT_DOWN) {    //  if down, remain that way
        }
        else {                                  //  is up
            for (i=0; i<NumButtons; i++) {      //   so unpush other buttons in group
                if ((Buttons[i].Group == Group) && (Buttons[i].Status == BT_DOWN) && (i != BX)) {
                    Buttons[i].Status = BT_UP;
                    DrawButton(Buttons[i].ID,Buttons[i].Status);
                }
            }
            Buttons[BX].Status = BT_DOWN;       //   and push this button down
        }
    }
    else {                                      // not a group, so just toggle
        Buttons[BX].Status = (Buttons[BX].Status == BT_DOWN) ? BT_UP : BT_DOWN;
    }

    DrawButton(BID,Buttons[BX].Status);

    if (Buttons[BX].Status == BT_DOWN) {        // is this button now (or still) pressed?
        SendCmd(Buttons[BX].Cmd);
    }
}
```

HOT CHIPS 27

ADVANCE PROGRAM

August 23-25, 2014

A Symposium on High-Performance Chips

Flint Center for the Performing Arts-Cupertino,CA

http://www.hotchips.org

HOTCHIPS brings together designers and architects of high-performance chips, software, and systems. The tutorial and presentation sessions focus on up-to-the-minute developments in leading-edge industrial designs and research projects.

Register Now: https://www.123signup.com/register?id=ytdtb

Sunday August 23

Tutorial 1: Deep Learning — **University of Montreal**
- Architectures, Algorithms and Applications
- Common SoftwareTools, Research Questions, Outlook

Tutorial 2: Makers from Hobbyists to Professionals — **UC Davis**
- Maker Trends: The Path of Least Resistance — **SparkFun**
- IoT Device Development Challenges and Solutions — **Redpine Signals**
- Implementing Software Defined Radio on the Parallella — **Adapteva**
- Current Trends for Hardware & Software Developers — **ARM**

Monday August 24

Internet of Things
- PULP: A Parallel Ultra-Low-Power Platform for Next Generation IoT Applications — **DEI, ETH,STMicro**
- Design of an Ultra-low Power SoC Testchip for Wearables & IOT — **Intel**
- Ultra-Low Power Wireless SoCs Enabling a Batteryless IoT — **PsiKick**

Keynote 1 — ***Convolutional Neural Networks*** — ***Yann LeCun*** — ***Facebook***

Multimedia and Signal Processing
- Architecture of the V6x Hexagon DSP for Mobile Imaging and Always-On Applications — **Qualcomm**
- A Scalable Heterogeneous Multicore Architecture for the Advanced Driver Assistance System — **TI**
- Energy-Efficient Graphics and Multi-media in 28nm Carrizo APU — **AMD**
- Revisiting DSP Acceleration with the Kalray MPPA Manycore Processor — **Kalray**

High Performance and Cloud Computing
- Scale-Out Computing and the X-Gene Processor Family — **Applied Micro**
- A 64-Core ARM v8 Processor for HPC — **Phytium**
- Oracle's Sonoma Processor: Advanced Low-Cost SPARC Processor for Enterprise Workloads — **Oracle**
- I/O Virtualization and System Acceleration in Power8 — **IBM**

Tuesday August 25

FPGAs
- Xilinx 16nm UltraScale+ MPSoC and FPGA Families — **Xilinx**
- Stratix 10: Altera's 14nm FPGA Targeting 1GHz Performance — **Altera**
- Toward Accelerating Deep Learning at Scale Using Specialized Logic — **Microsoft Research**

GPUs
- MIAOW – An Open Source GPGPU — **University of Wisconsin**
- AMD's next Generation GPU and Memory Architecture — **AMD**
- The ARM Mali-T880 Mobile GPU — **ARM**

Keynote 2 — ***The Road to 5G*** — ***Matt Grob, CTO*** — ***Qualcomm***

Applications
- Professional H.265/HEVC Encoder LSI Toward High-Quality 4K/8K Broadcast Infrastructure — **NTT**
- Ultra-Low-Light CMOS Biosensor Helps Tackle Infectious Diseases — **Anitoa**
- Five-Speed PHY Enables 2.5Gbps and 5Gbps Ethernet Rates Over Legacy Copper Cables — **Aquantia**

Processors
- Knights Landing: 2nd Generation Intel "Xeon Phi" Processor, — **Intel**
- Intel "Xeon" Processor D: The first Xeon Processor Optimized for Dense Solutions — **Intel**
- Raven: A 28nm RISC-V Vector Processor with Integrated Switched-Capacitor DC-DC Converters and Adaptive Clocking — **UC Berkeley**
- Intel Atom-X5/X7-8000 Series Processors, Codenamed Cherry Trail — **Intel**

A Symposium of the Technical Committee on Microprocessors and Microcomputers of the IEEE Computer Society and the Solid-State Circuits Society

LISTING 5
Given a screen touch coordinate, this function returns the ID of the button surrounding that point or zero if the touch falls outside of all buttons. Excluding the MARGIN around each button's border ensures valid touches hit near the middle of the button image. Superimposing the white rectangle on the button helps debug the button handlers.

```
byte FindHit(TS_Point hit) {

byte i;
TS_Point ul,lr;

#define MARGIN 12

    for (i=0; i<NumButtons ; i++) {
        ul.x = Buttons[i].ulX + Buttons[i].szX/MARGIN;
        ul.y = Buttons[i].ulY + Buttons[i].szY/MARGIN;
        lr.x = Buttons[i].ulX + ((MARGIN - 1)*Buttons[i].szX)/MARGIN;
        lr.y = Buttons[i].ulY + ((MARGIN - 1)*Buttons[i].szY)/MARGIN;
        if ((hit.x >= ul.x && hit.x <= lr.x) &&
            (hit.y >= ul.y && hit.y <= lr.y)) {
            break;
        }
    }

    if (i < NumButtons) {
        tft.drawRect(ul.x,ul.y,lr.x-ul.x,lr.y-ul.y,ILI9341_WHITE);
        return Buttons[i].ID;
    }
    else {
        return 0;
    }

}
```

happens immediately after the UI detects a screen touch, your fingertip hides most of the action.

Now that you see how the buttons operate, perhaps you can see how the shading in Photo 1 represented light glinting off buttons bumped up or down from the LCD surface. You can't? Neither could Mary, which explains why the buttons in Photo 2 have simple, flat colors with no textures or overlays.

After all that, the code draws the button in its new state and sends its command string to the motor controller. A more complex UI might need the ability to send commands for just-released buttons, but that's not needed here.

The commands consist of two-character strings sent to the motor controller through the Mega's `Serial1` hardware serial port. The controller returns status and debugging information through that port, which the UI code copies to the default USB-serial port used for console output and program transfer. I set the USB port to 57600 bps and the motor controller port to 9600 bps in order to prevent overruns.

On the motor controller side, a function called in the main loop returns a complete UI command string. If the format looks correct, a simple parser identifies the command and updates the appropriate flags and values accordingly. It doesn't return a confirmation to the UI: the serial link has been completely reliable.

Before all that can happen, of course, the UI must determine which button image lies underneath the fingertip on the screen.

FINDING THE HITS

The Adafruit library includes functions that return the coordinate of the most recent touch in the screen's native coordinate system. I added a function to scale, rotate, and map those touch-screen points into the LCD's coordinate system, returning a `TS_Point` triplet containing the X, Y, and Z values of the touch. The Z "coordinate" increases with the pressure applied during the touch, but the UI code ignores that value.

circuitcellar.com/ccmaterials

RESOURCES

E. Nisley, "Linearizing the Speed Control," SoftSolder.com, 2015, http://softsolder.com/2015/02/18/kenmore-158-linearized-speed-control/.

———, "Pulsed Motor Drive," *Circuit Cellar* 298, 2015.

E. Nisley, "Arduino Mega PCB Holder," SoftSolder.com, 2015, http://softsolder.com/2015/02/02/arduino-mega-pcb-holder/

Symbola Unicode font (and many others), http://users.teilar.gr/~g1951d/.

Unicode Character Tables, http://unicode-table.com.

SOURCES

2.8" TFT Capacitive Touch Shield and 2.8" TFT Resistive Touch Shield
Adafruit Industries | www.adafruit.com/products/1651, www.adafruit.com/products/1947

Arduino Mega Microcontroller Board
Arduino | www.arduino.cc/en/Main/ArduinoBoardMega2560

ImageMagick Software Suite
ImageMagick | www.imagemagick.org

The `FindHit()` function in **Listing 5** scans through the `Buttons` array to determine which button image contains the touched point. Although the touch point represents a single pixel, my fingertip on the resistive touch screen can return any of several neighboring points; the accuracy seems to be about ±3 pixels in all directions. Touching a button image off-center could activate the adjacent button, even though I made the images as large as possible.

Therefore, I added the `MARGIN` constant to put a small inactive border around the central part of the button and have the code ignore a touch in the border. Although touching the screen and having nothing happen seems odd, triggering the wrong action based on an errant coordinate turned out to be even worse.

When the code identifies the button, it draws a thin white outline around the active region. That changes so few pixels it's almost instantaneous, although it's hidden below the fingertip.

The UI checks for a press on each pass through the main loop. All of the processing you've seen earlier happens immediately after detecting a touch and identifying the button, so it's generally finished while the fingertip is still touching the screen. As a result, I added some code to wait until the touch vanishes and drain the queue inside the touch screen's interface chip before continuing. That wouldn't be appropriate for a drawing program, but this UI requires only discrete activations of separate buttons.

RUNNING THE MACHINE

The UI starts up with Mary's preferred settings selected, as shown by the highlighted buttons in Photo 2:

- Stop with needle down
- Normal pedal mode
- Full speed

The machine's speed in the normal "Run" mode varies directly with pedal pressure. We tested several different transfer curves and finally settled on one that linearizes the pedal-to-speed relationship, with smooth transitions on each end of the range due to the pedal's Hall effect sensor response. Having floating point arithmetic available in a microcontroller dramatically simplified those curves: measure and plot the data, fit a curve, then transfer the coefficients directly to the motor controller.

The "One" mode acts as a single-shot trigger: pressing the pedal produces one stitch, then the machine waits for the pedal to release. "Follow" mode causes the needle to track the pedal: push to lower the needle, release to raise it. "One" mode turned out to be superfluous, as Mary can get the same result by tapping-and-releasing the pedal. She hasn't yet encountered an occasion that requires "Follow" mode, but it may still be useful for something.

She expected that a maximum speed limit would be helpful, so I implemented three levels:

- "Sprinter" = full speed
- "Rabbit" = 75% of maximum
- "Snail" = 50% of maximum

However, as we progressively improved the pedal-to-speed transfer curve, the need for the speed limits gradually went away. The original rheostat tended to start abruptly at a fairly high speed, giving her very little control. The Hall effect sensor and linear speed mapping provide such smooth control that she can maintain whatever speed she needs for as long as it's needed.

CONTACT RELEASE

Mary now has what she's wanted for years: a simple sewing machine that does exactly what's needed and nothing more. I didn't expect the project to require two microcontrollers, a huge bipolar transistor, and pulse-mode motor control, but that's how it worked out.

After using the buttons shown in Photo 3 for a while, however, Mary decided that subdued colors were in order. I substituted HSB hues 50%, 14%, and 44% in the code in Listing 1, ran the script, copied the files to the MicroSD card, and reskinned the UI with barely five minutes of downtime.

As you can tell from the photos, I must also wrap a box around that Mega and LCD panel before it's ready for actual use. Given a bit of luck, a 3D printer, and some spray paint, perhaps it won't be quite so ugly as the buttons.

ABOUT THE AUTHOR

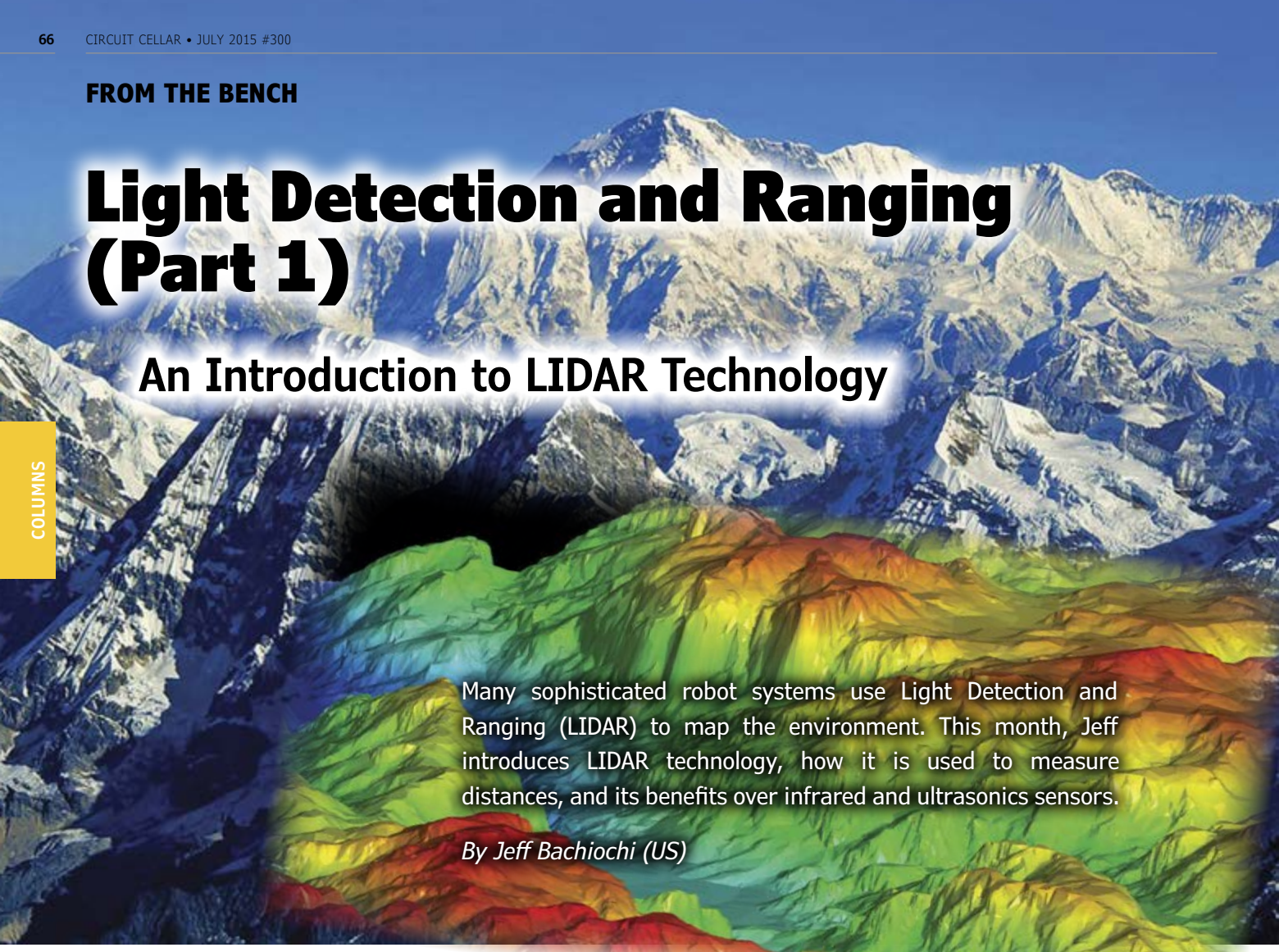
Ed Nisley is an EE and author in Poughkeepsie, NY. Contact him at ed.nisley@pobox.com with "Circuit Cellar" in the subject line to avoid spam filters.

FROM THE BENCH

Light Detection and Ranging (Part 1)

An Introduction to LIDAR Technology

COLUMNS



Many sophisticated robot systems use Light Detection and Ranging (LIDAR) to map the environment. This month, Jeff introduces LIDAR technology, how it is used to measure distances, and its benefits over infrared and ultrasonics sensors.

By Jeff Bachiochi (US)

You may be familiar with Light Detection and Ranging (LIDAR) from the DARPA Grand Challenge or other sophisticated robotic application that requires mapping of the environment. LIDAR is similar to infrared and ultrasonic rangers in that they send out light or sound and look for reflections returning from objects out in front of the sensor. The distance is determined by measuring the time of flight of the reflection to return. This is calculated at the speed of light or the speed of sound (for ultrasonics).

The speed of sound is most friendly to calculations because it travels at a rate of 1,126 feet per second, which takes milliseconds and can be handled with most microcontrollers. The speed of light, however, travels at a rate of 186,282 miles per second. To measure feet, we need to measure parts of a nanosecond—a bit tricky. Infrared distance sensors aren't actually measuring the speed of light, but the angle of returned signal and are limited to inches.

Probably the biggest difference between ultrasonic and LIDAR is the width of transmission field. Ultrasonic transducers are like most speakers in their radiation pattern—quite wide. LIDAR, however, uses an LED/laser or optics to keep the transmission focused. Since an ultrasonic ranger has such a wide lobed dispersion pattern, a pole anywhere in its field will look the same as a wall! You may not be able to determine there is actually a doorway until you get close enough that none of the beam is reflected by the doorjambs. With LIDAR and other optical sensors, the transmission is concentrated. And as such the target area is very small and more energy reaches its localized target and is reflected back, allowing for targets further away. Directional accuracy becomes more important or you can miss the intended target. This becomes a trade-off between field of view and maximum distance. Both of these devices are useful in their own right.

Since LIDAR devices are more selective, they can be more representative of the surroundings. They can be used to map an area by moving the sensor and collecting data from many points. When this data is displayed appropriately, it can provide a map of the area it covers. While this is a far cry

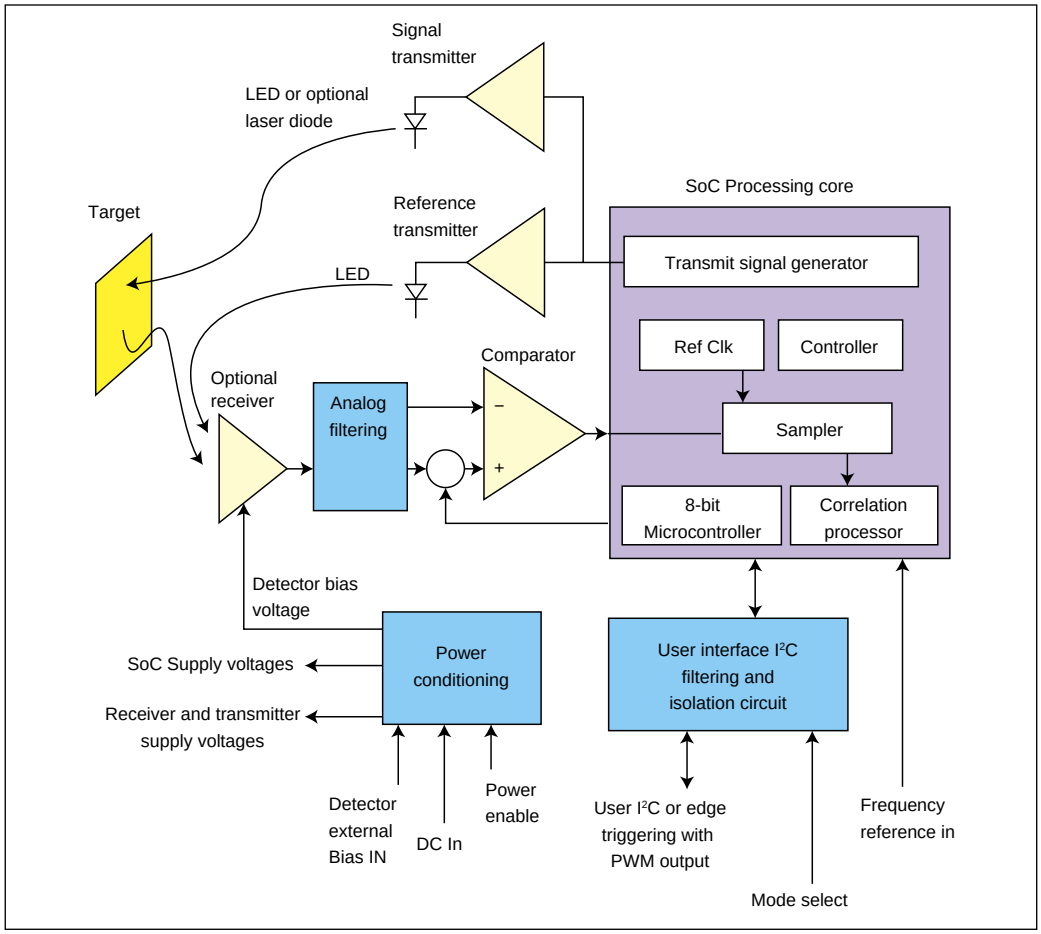


FIGURE 1
A PulsedLight SoC is the heart of the LIDAR-Lite optical distance measurement sensor. This signal-processing core handles all of the functions required to support the company's proprietary range finding system architecture.

from recognition of what's actually there, it can provide information on free space versus objects in its field of view. Many LIDAR units are spun for a 360° field of view. One of Roomba's competitors, from Neato Robotics, uses a LIDAR for room mapping in its Neato XV-11 robot vacuum cleaner. While still at hundreds of dollars, it represents one of the first cost breakthroughs in LIDAR technology. PulsedLight now brings LIDAR within reach for all designers with its LIDAR-Lite, which costs less than $100.

MEASUREMENT SENSOR

In March 2014, PulsedLight successfully completed a crowd-sourcing campaign with a low-cost measurement sensor technology that was small, low-power, and high performance. The LIDAR-Lite demonstrated that you can have all of this in a sensor capable of measuring out to 40 m using inexpensive, off-the-shelf, electro-optical components.

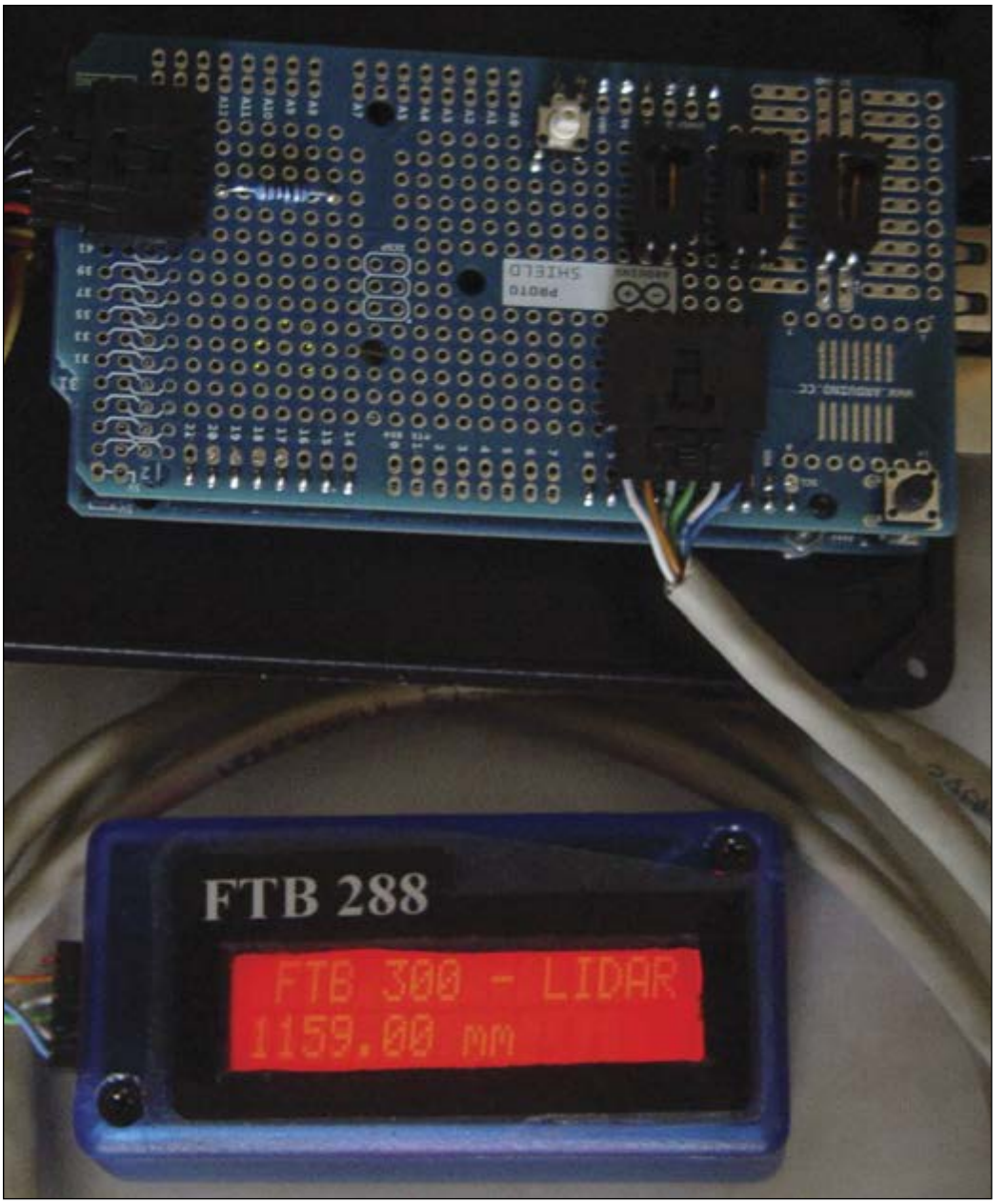
Up to this point if you wanted to measure distance in bright light, or against non-reflective targets, you needed a powerful or collimated source, a very sensitive detector for the return signal, and precise clock capable of measuring the time-of-flight in picoseconds. Available solutions at the time cost hundreds and even thousands of dollars. A new approach would be necessary if lower costs were to be achieved.

PulsedLight uses edge-based signal reconstruction methods, similar to that used in radar, along with signal correlation processing to perform a "signature match" between a captured transmit reference record and the received signal record. A specialized digital signal processor (DSP) would enable a single-chip processing implementation suitable for low cost, ultra-small optical distance measurement sensor applications. While the DSP is the key, what better way to showcase this new technology than by producing a picture that would be worth more than a 1,000-word datasheet? Enter LIDAR-Lite.

THE THEORY

Let's start by looking at what is required to use the System on a Chip (SoC) in this application. The LIDAR-Lite user interface has six connections: 5 V, Ground, I2C Clock, Data, Power Enable, and Trigger In/PWM Out. These are similarly labeled at the bottom of **Figure 1**, and they are the only signals with which you need to be concerned. The binocular-shaped enclosure has a pair of parallel optical tubes protruding off of the electronics PCB. These

PHOTO 1
The Arduino shield atop an Arduino MEGA is the transition between the Arduino's mapped I/O and the circuitry that you might need to add for your project. When the project uses modules that are complete unto themselves, you need only provide connections to those modules as in this project using an external LCD and the LIDAR-Lite sensor.



keep the transmitter and receiver optically isolated from one another. They form the transmit and receive path for the ranging signal.

Note that there are actually two transmitters. A reference transmitter (RT) is placed at the optical receiver (OR) so the receiver can record the actual signal that is being transmitted, hence the reference. The signal transmitter (ST) sends its burst out the transmitter tube. If this signal is reflected back in through the receiver tube, the delayed signal is recorded by the OR.

The on-board supply takes care of creating various supplies used by the various components. A 5-V supply at 100 mA is all that is required for this module (even though internal and intermittent transmitting current is significantly higher). The Power Enable input is internally pulled up so the module is automatically enabled. You can disregard the pin, unless you want to place the device in a

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.

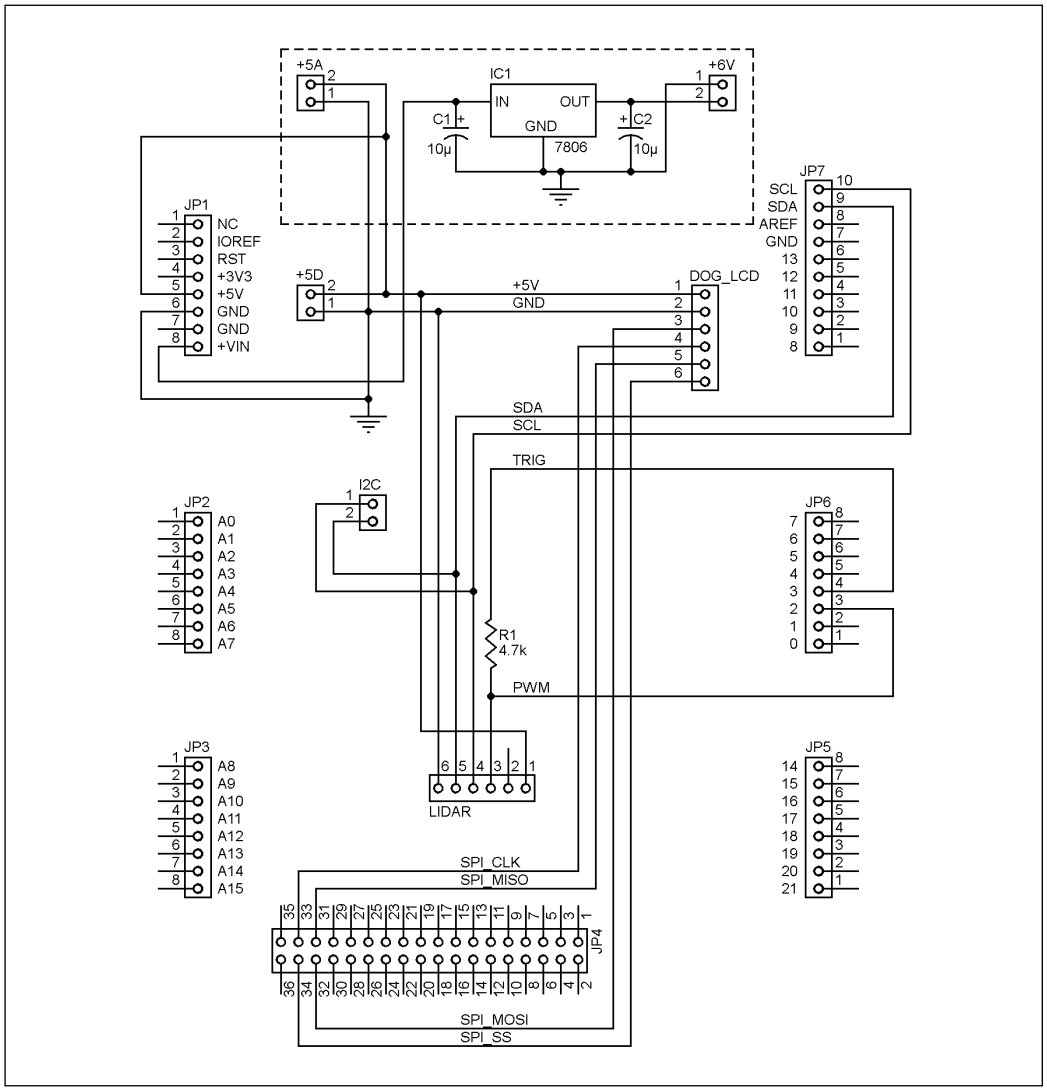


FIGURE 2
This schematic shows the SPI serial interface to the DOG LCD and the LIDAR-Lite ranging module. The interface was built on an Arduino shield to speed up the prototyping process. An optional 6-V regulator adds support for next month.

low-power mode (by grounding the input). I won't discuss the I2C pins in this column; they can be left unconnected as well. We only need be concerned with the Trigger/PWM pin.

We will be using the external trigger mode for this column. Everything we need (besides the power connections) is on this single input/output pin! If we place a 4.7 kΩ resistor from this pin to ground, the internal application will recognize that we wish it to perform an acquisition. Once the internal circuitry sees this Trigger, has woken up, performs its ranging function, and enables its output driver, it will force this output high for a period of time equal to the calculated distance. This pulse width follows a 10 µs/cm relationship to the measured distance, or 1 ms per meter. We can then measure the duration of the high period of this pulse to determine the reported distance.

SAMPLING

I want to have time for a quick application using LIDAR-Lite in this mode, so I'll save some of the mystery (including using the I²C interface) for my next article. However, a few technical specifications may answer some immediate questions. An acquisition begins with a bit of internal initialization after which the transmitter sends a burst directly to the

```
// the DOG LCD Display:
const int slaveSelectPin = 53;
const int RSSelectPin = 50;
const char SetUpArray[9]={0x39, 0x1C, 0x69, 0x52, 0x70, 0x38, 0x06, 0x0F, 0x01};
char Row1Array[]={" FTB 300 - LIDAR"};
char Row2Array[]={" jeff bachiochi "};
```

LISTING 1
I set up these definitions to use the DOG display.

```
const int LIDARTriggerPin = 18;
const int LIDARPWMPin = 19;
unsigned long pulse_width;
char ModeString[] = (" mm cm m  ft in");
const int ModePin = 17;
int Mode = 0;
float val;
int pointer;
int i;
```

LISTING 2
When the LIDAR is disabled, it will go into a low power mode. The pins are defined for the LIDAR as well as a number of variables for the application.

```
void setup()
{
  Serial.begin(9600);
  // set the slaveSelectPin and RSSelectPin as an outputs:
  pinMode (slaveSelectPin, OUTPUT);
  pinMode (RSSelectPin, OUTPUT);
  // initialize SPI:
  SPI.begin();
  // initialize the DOG LCD
  DOGInit();

  // set the LIDARTriggerPin as an output:
  pinMode (LIDARTriggerPin, OUTPUT);
  // set the LIDARPWMPin as an input:
  pinMode (LIDARPWMPin, INPUT);
  // set the LIDARTriggerPin LOW for continuous reads:
  digitalWrite(LIDARTriggerPin, LOW);

  // set the ModePin as an input:
  pinMode (ModePin,  INPUT_PULLUP);
}
```

LISTING 3
The `setup()` function handles all of the initialization for the LCD, the LIDAR, and the mode switch

circuitcellar.com/ccmaterials

SOURCES

Arduino Mega
Arduino | www.arduino.cc

LIDAR-Lite Laser Module
PulsedLight | www.pulsedlight3d.com

receiver which records the burst as a reference signal. The burst is what I would call a 50-MHz Manchester encoded Barker sequence, which lasts 500 ns. Barker data is an easily correlated sequence of data. Being able to identify the data is key to determining the time of flight.

The receiving circuitry is interested in the zero crossing transitions of the received signal. This is done by providing a low-pass and high-pass path to a comparator's inputs. An offset controller is summed with the high-pass input to allow for DC offset control. Sampling of the comparator is done at a 500-MHz rate building a nibble of serial data to pass to the processor at a 125 MHz rate. These samples are not actually stored, however, the processor looks for rising and falling edges amongst the data and keeps a running tally of accumulated edges for each time slot and that is stored in memory. The edge totals are stored in memory in real time from start until memory is filled. For the reference sample, 64 samples represent the 500-ns burst. Each location in the record is therefore an accumulation of modulation activity for that time slot. Multiple bursts may be used to add additional data to the previous accumulations until a maximum allowable value has been reached by any time slot. This builds up a reference of the pattern we want to look for in the real signal return.

Next, the bursts are now emitted from the device. If there is an object within the flight path, a reflection is returned to the receiver. Like the reference we just discussed, the processor will build a record out of edge accumulations, but this time a much larger record. This record must encompass all of the time that it would take a signal to reach some maximum distance and come back. Since we don't know if this will be 1′ or 100′, we have to record it all. The further away an object is, the further into the record we go until the receiver 'sees' the reflection. Further objects also return smaller signals. This means that multiple acquisitions must be performed until a maximum allowable value has been reached (just like with the reference). Obviously, the smaller the received signal the closer it is to the noise level, so noise becomes a larger part of the distant signals.

This is where the Barker sequence helps because we will not just be looking for some modulation, we will be looking for a particular modulation. And this magic happens during the correlation process.

CORRELATION

Correlation is a statistical technique that can show whether and how strongly two things are related. For example, if you hold two identical hair combs horizontally up to the light and move one back and forth in front of

Experience extraordinary training for embedded systems professionals at ESC Conference Series 2015!

The **Embedded Systems Conference (ESC)** is the industry's largest, most comprehensive technical conference for embedded systems professionals in the U.S. ESC is excited to announce an expanded 2015 USA Conference Series, taking place in **Boston (May 6-7), Silicon Valley (July 20-22)**, and **Minneapolis (November 4-5)**.

The ESC series continues in Silicon Valley, a leading hub for high-tech innovation and development; its home to many of the world's largest high tech corporations, as well as thousands of high-tech startup companies. Silicon Valley boasts more engineers per square foot than anywhere else on earth.

Come join us at the Santa Clara Marriot and check out some of our sponsors for the event:

Rohde & Schwarz | **All Quallity & Service, Inc. (AQS)** | **AdaCore** | **Altium, Inc.** **AMP Display** | **Azul Systems, Inc.** | **Chefree Technology Corp** | **Code** | **Data Image Corporation** | **EBS Net Inc.** | **EMA Design Automation** | **STMicroelectronics** | **Teledyne LeCroy** | **Toradex Inc.** | **WolfSSL** | **Embedded Works** | **Green Hills Software, Inc.** | **IEEE** **Ironwood Electronics** | **Lauterbach** | **Micro Computer Control Corp.** | **Pico Technology** **Rigol Technologies** | **Gemalto** | **Silex Technology America** | **Symmetry Electronics** **The QT Company** | **Trusted Computing Group**

Register today at **www.embeddedconf.com/silicon_valley** with promo code **CircuitCellar15SV** and get 15% off an All Access Pass. The offer ends July 15, 2015.

LISTING 4
For this application, the Arduino's `PulseIn(pin, state)` command is used to translate LIDAR-Lite's pulse output to a distance measurement. At a rate of 1 µs/cm, the pulse width time is easily converted into any unit of measurement. A single push button allows the user to select from one of five units for a continuous display of distance on the backlit DOG LCD.

```
void loop()
{
  // if Mode Switch is pushed then increment Mode
  if (digitalRead(ModePin)==0)
  {
    Mode++;
    if (Mode==5)
    {
      Mode=0;
    }
  }
  // count how long the pulse is high in microseconds
  pulse_width = pulseIn(LIDARPWMPin,HIGH);
  // if a reading isn't zero, display it
    Serial.print("pulse width=");
    Serial.println(pulse_width);
  if (pulse_width !=0)
  {
    // 1us = 1mm distance for the LIDAR-Lite PWM
    int millimeters=pulse_width;
    float centimeters=pulse_width/10.0;
    float meters=pulse_width/1000.0;
    float inches=pulse_width/25.4;
    float feet=pulse_width/304.8;
    switch (Mode)
    {
      case 0:
      {
        val=millimeters;
        break;
      }
      case 1:
      {
        val=centimeters;
        break;
      }
      case 2:
      {
        val=meters;
        break;
      }
      case 3:
      {
        val=feet;
        break;
      }
      default:
      {
        val=inches;
        break;
      }
    }
    dtostrf(val, 4, 2, Row2Array);  //4 is mininum width, 2 is
precision
    pointer=0;
    do
    {
      pointer++;
    }
    while (Row2Array[pointer]!=0);
```

Listing 4 continues on page 73

LISTING 4 CONTINUED

```
    for (i=0;i<=2;i++)
    {
      Row2Array[i+pointer]=ModeString[Mode*3+i];
    }
    pointer=pointer+i;
    for (i=pointer;i<=15;i++)
    {
      Row2Array[i]=' ';
    }
    DisplayRow2();
    delay(500);
  }
}
```

the other, you will see light between the teeth when they are in perfect alignment and no light when the teeth are staggered. The more light you see, the closer the correlation. This is normally indicated by a positive number, while a negative number would mean poor correlation. In this case the two combs would correlate positively (and negatively) in a large number of places.

Now, block off all of the teeth on one comb except for one single slot. This will be the reference. Again, slide the two combs back and forth. Hmm, we are seeing the same thing with reference to this single reference slot. The correlation is high, then low, then high, and so on for the length of the second comb. This is just as it should be. The second comb, our sample match the reference comb at every slot.

Now let's block off all the teeth except one on the sample comb. When the reference comb passes in front of the sample comb, all will be dark until the two open slots align. If we were to count the number of teeth we had to go by until we got to the open slot, this might be used to indicate a measurement. We found a positive correlation at some point in the sample record. With the LIDAR this would indicate a point in time from the beginning of the record period where we found a positive correlation between the record and the reference.

If we were looking for simply a sign of modulation we've got it. There are a few problems with this. First, we can't really rely on catching the first edge of the modulation. This can throw off the distance calculation. Second, when the returned signal is small as compared to the noise, the first edge is even harder to detect if at all possible. Enter the modulation pattern. Suppose we were to break off some pattern of teeth on both combs—say, break 1, skip 1, break 2. When the reference comb is passed over the sample comb, there will be a few places where there is a positive correlation, but only one position where all the gaps align to give a perfect match. By selecting the position with the highest correlation, we have found a point of reference.

It has been found that certain patterns are best for determining high correlation. These Barker codes or specific patterns of 1s and 0s not only have a high correlation when the match is close. The correlation falls off quickly on either side as well. Picking the pattern out, especially when in the midst of random noise, becomes easier.

LIDAR-Lite uses this technique. In fact, PulsedLight has given users access to many of the system's registers by adding an I^2C interface. Next month, I'll go into this a bit more; but for now, let's just use the default register settings and play with the device using the simple Trig/PWM I/O.

AND THE ANSWER IS...

One of the great design points for this device was to make it easy to use. The Trig/PWM pin gives a user the option of getting distance information without ever having to deal with any of the internal registers. Pulling the Trig pin low with a 4.7-kΩ resistor will tell LIDAR-Lite to acquire a distance measurement and then drive the pin high for 10 µs for every 1 cm of distance measured. The user is required to measure this period to determine the distance. To demonstrate how this is done, let's use an Arduino.

The Arduino is popular for its low cost, free IDE, and plethora of sample applications and library functions. I chose an Arduino MEGA because I like to have the additional I/O it provides. Prototype shields for the MEGA are relatively inexpensive and allow projects to be customized and removed from the Arduino at the end of the project (see **Photo 1**). This allows the Arduino to be reused in another project (keeping project costs down). Many times the project will interface to modules and this means that the proto shield may contain only connectors. Again, this allows all of the components (minus the shield) to be reused in another project. This first application uses

PHOTO 2
While you can eliminate the LCD, if you wish to connect using the console serial (USB) port, I chose to add the LCD for measurement results. Providing batteries for the Arduino makes this a mobile platform.

two modules, the LIDAR and an LCD to present the distance data. A past column introduced DOG LCDs. I'll use this display with its six-wire modified SPI. **Figure 2** is simply a wiring diagram of the shield between the Arduino MEGA and the two external modules.

The Arduino "sketch" or application program requires three basic sections: definitions, the `setup()` function, and the `loop()` function. Definitions are used to describe variables, I/O designations, or external libraries. The `setup()` function is used to initialize I/O and any libraries used in the application. The `loop()` function contains the meat of the application and will run continuously. You may also have a number of additional functions. These might include user-written or library functions. The DOG LCD uses the SPI library function to handle the display. To use the DOG display, I set up the definitions shown in **Listing 1**.

While the SPI SCK, MOSI, are MISO are predefined for the MEGA as pins 52, 51, and 50, I redefined pin 50 as the `RSSelectPin` (an output) and defined 53 as the `slaveSelectPin` (an output). The display only takes data (has no data output) so the MISO pin now determines whether input data is command or display data for the display. The `SetUpArray` holds the initialization data for the display and the two `Row_Arrays` hold sign-on messages.

Since the LIDAR-Lite module is used in its pin mode, we don't need to initialize the I²C clock and data lines. We need only assign the Trig/PWM I/O pin to two I/O lines. Actually, if this pin is pulled to (hard) ground with a 4.7-kΩ resistor, we need only attach the pin to one input to read the duration of the pulse. Here I use a separate output pin (soft ground) to enable (pull the resistor to ground) or disable (pull the resistor high) the LIDAR. When the LIDAR is disabled, it will go into a low power mode (see **Listing 2**). The pins are defined for the LIDAR, as well as a number of variables I'll use for the application. Note the `ModeString` array. This array holds three character suffixes that will be added to the distance measurements to indicate the distance mode. The distance mode is set by a switch connected to the `ModePin` input. I added this switch to allow the user a choice of how the distance is displayed. The distance can be displayed as millimeters, centimeters, meters, feet, or inches. While the LIDAR output is based on centimeters, simple calculations can transform output value to reflect other units.

The console serial port can be used as a display. While I wanted this application to be standalone using the DOG LCD, it is handy to

begin using the console for debugging, so it is included here but those lines can be eliminated in the final version. It is important that after being defined all of the I/O is properly configured. The `setup()` function handles all of the initialization for the LCD, the LIDAR, and the mode switch (see **Listing 3**).

The `loop()` function has 5 tasks, check the mode switch, read a pulse width, convert the distance in centimeters to the proper unit for display, format the floating-point value, and display the results. If the mode switch is being pressed, increment mode and confine it to (0–4). Since we've set the LIDARTriggerPin low, the LIDAR will continue to automatically take distance acquisitions. The next time the LIDARPWMPin goes high the Arduino will measure how long it stays high and assign pulse_width that value. While we only need to calculate a new value when any unit other than cm is chosen, I calculate all of these anyway. Then I pick the right one to display and format the value with two decimal places to be displayed along with the unit of choice. The distance is displayed on the second line of the DOG LCD (see **Photo 2**).

As you can see in **Listing 4**, there are four user functions that deal with the DOG LCD: `DOGInit()`, `DOGWrite()`, `DisplayRow1()`, and `DisplayRow2()`. These are contained in the complete sketch file posted on the *Circuit Cellar* FTP site.

SEEING IS BELIEVING

While there is nothing that compares to our eyesight, great strides have been made in using inexpensive cameras for object detection. With that comes a lot of baggage, or software that can't even run on a small microcontroller. For now that equates to a separate expensive module dedicated to a particular function. While LIDAR-Lite can't give you object recognition, it can give you a pretty good indication of any barriers it sees within its narrow cone of transmission. This means you can easily differentiate between a chair leg and a wall. This is something that is not easily done using ultrasonics. LIDAR handles all the hard stuff and gives you a 1-bit interface (Trig/PWM) that requires little of your microcontroller.

When we "take in" our surroundings our body generally stays still while our eyes and/or head turns and scans the area. This scanning process allows us to piece together a bunch of information into a bigger picture. Next time, we'll look into the I^2C interface, our window into its world, but also an advanced project using this new and accessible technology.

CC SHOP

1

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

2 CC 2014 CD

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

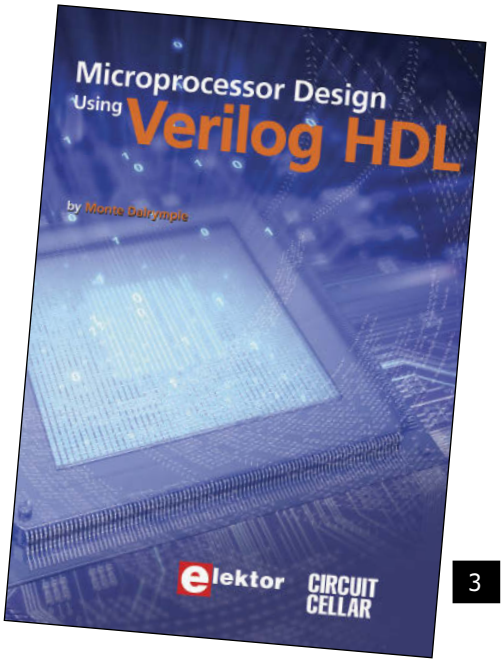
Item #: CD-018-CC2014

Previous Years Also Available

3 MICROPROCESSOR DESIGN USING VERILOG HDL

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

Author: Monte Dalrymple
Item #: CC-BK-9780963013354



3

4

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

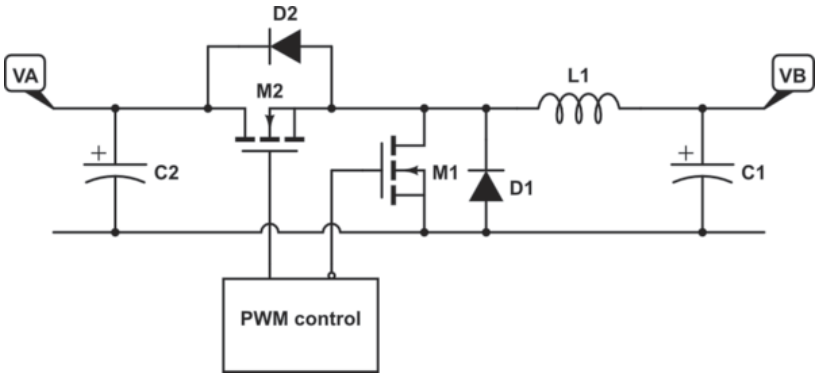
PROBLEM 1—The diagram below is a simplified illustration of a switchmode "buck" DC-DC converter with synchronous (active) rectification. The switching elements are shown as MOSFETs, with the associated body diodes drawn explicitly. The details associated with driving the MOSFET gates are ignored, other than to say that when one is on, the other is off, and the duty cycle is variable.

This is, by definition, a continuous conduction mode (CCM) converter. What does this tell us about the relationship between VA, VB, and the duty cycle of the switching?

PROBLEM 2—Can the output of such a converter sink as well as source current? If so, where does the current go?

PROBLEM 3—Draw a similar diagram for a switchmode "boost" DC-DC converter with synchronous rectification. What interesting thing can you say about the two diagrams?

PROBLEM 4—Based on the answers to the previous questions, what can you say about the direction of power flow through this type of converter?

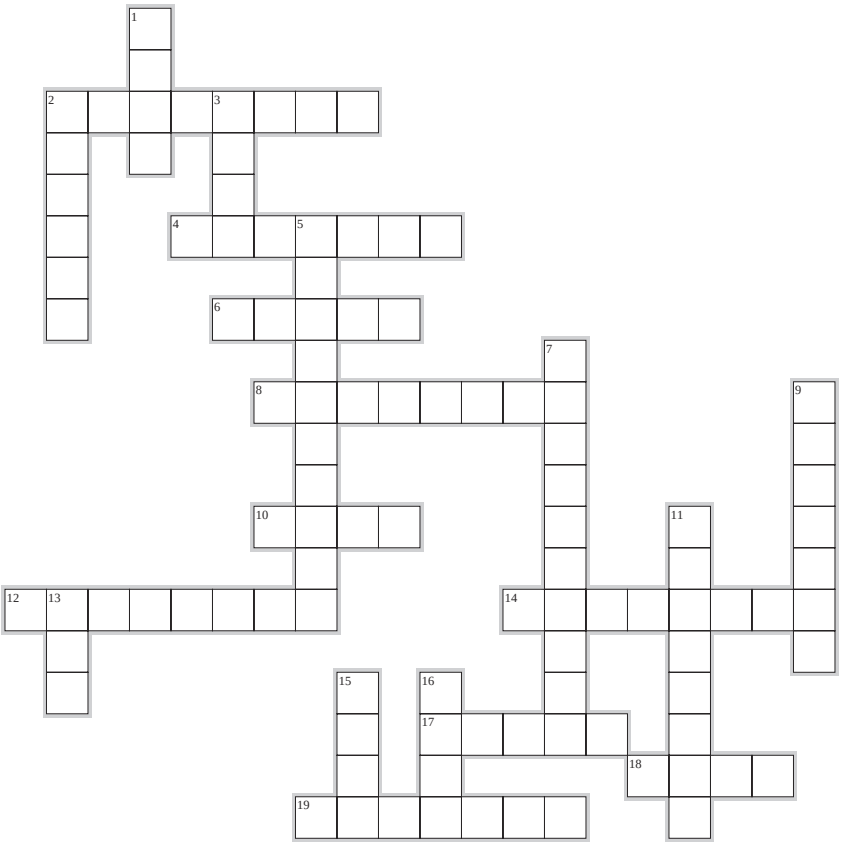


What's your EQ? The answers are posted at www.circuitcellar.com/category/test-your-eq/. You can contact the quizmasters at eq@circuitcellar.com.

CROSSWORD

JULY 2015

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



ACROSS

2. abA; Biot
4. 6.28 × 10^{18} electrons
6. 19th-century physicist credited with inventing the first battery
8. Anion charge
10. False value
12. 1,000 kBd
14. Significand
17. *e*, Leonhard
18. Sensor node
19. "Excited dimer", laser

DOWN

1. Robert Dennard, IBM, 1967
2. "Guidance Computer" invented in 1966 at The Charles Start Draper Laboratory for spacecraft navigation and control
3. $1{,}000^{-4}$
5. Patented the FET in 1925
7. V/I
9. Scientific programming language developed in the 1950s by John Backus and IBM
11. Two stable states; latch
13. 10^{18}
15. Unijunction can be used to measure this
16. $1{,}000^{-1/3}$

IDEA BOX the directory of PRODUCTS & SERVICES

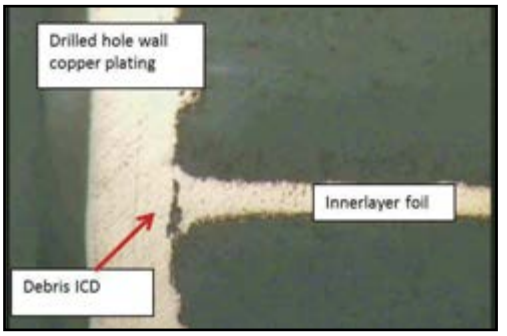
For current rates, deadlines, and more information contact Peter Wostrel at 978.281.7708 or circuitcellar@smmarketing.us.

Interconnect Defects (ICDs) Explained

By Doug Trobough

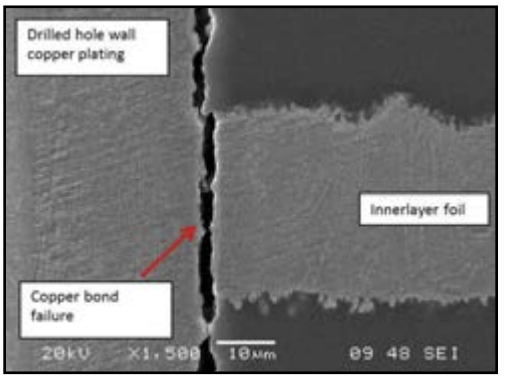
What is an Interconnect Defect (ICD)? An ICD is a condition that can interfere with the internal circuit connections in a printed circuit board (PCB). These internal connections occur where the innerlayer circuit has a drilled hole put through it. PCB processing adds additional copper into the drilled hole to connect the innerlayer circuits together and bring the circuit to the PCB board surface where connectors or components are placed to provide final function.

If there is a defect at or near this interconnect or plating and innerlayer copper, it could lead to failure of a specific circuit (or net). This defect typically causes open circuits, but could be intermittent at high temperatures. Of significant concern is that the functionality may be fine as the PCB is built, but will fail in assembly or usage, becoming a reliability risk. This latency for the defect has put ICDs on the serious defect list in the industry. Another item is that ICDs have increased in frequency over the past five to seven years, making this a higher priority issue.



Debris-based ICD

The majority of ICDs fall into two categories: debris-based ICDs and copper bond failure ICDs. Debris-based ICDs are caused by material left behind by the hole drilling process. This material is supposed to be removed from the holes, but is not when ICDs are found. Some causes are drill debris residues, drill smear and particles (glass and inorganic fillers) embedded into the innerlayer copper surface. The increases in this ICD type seems to be related to the increased usage of low Dk/low Df materials that use inorganic filler types. These materials generate more drilling debris and are often more chemically resistant materials, compared to standard FR-4 epoxy materials. This combination of effects makes the drilled holes much more difficult to clean out completely.



Copper bond failure ICD

Copper bond failure ICDs occur when the copper connection is physically broken. This can be due to high stress during assembly or use, or the copper bond being weak (or a combination). This failure mode is also design related, in particular, increased PCB thickness, increased hole size and wave soldering all tend to increase the risk of copper bond ICDs. It seems that there has been an increase in the rate of this ICD type, which is related to increased lead-free soldering temperatures and increased board thickness over the past 10 years. Note: This condition also occurs on HDI microvias. The causes are similar but the processing is different.

Reliability testing has been run on both types of ICDs. Copper bond type ICDs are a significant reliability issue. They show up as assembly failures and product with weakness may have increased tendency for field failures. Drill debris type ICDs have not been shown to be a significant reliability issue in several studies, but they are an industry specification failure, so they affect product yield and costs. Well run IST testing, using a valid coupon structure, has been a very valuable testing method for determining risk due to ICDs.

ICDs can be prevented by good PCB design and improved PCB processing methods. Debris type ICDs are a function of drilling parameters and desmearing. Many of the newer materials with fillers do not drill like standard FR-4. Instead of forming a chip during drilling, they break apart into small particles. These particles then tend to coat the drilled hole walls. One factor associated with debris ICDs is drill bit heating. Factors that result in hotter drill bits cause more debris formation and residues.

Desmearing, which is done to remove drilling residues, often needs to be more aggressive when using these material types. This has been effective at reducing or eliminating debris ICDs

Copper bond failures are a little more complex. In PCB processing, the key factors are cleaning the innerlayer copper surface so that a strong bond can form. In addition, the electroless copper deposit needs to be in good control, having the correct thickness and grain structure, to have the required strength. Testing and experience show a good processing focus, along with appropriate reliability testing can result in consistently robust product.

Design factors also play a big role. As noted above, board thickness and hole size are key factors. These relate to the amount of stress placed upon the interconnect during thermal exposure. Eliminating soldered through-hole connectors is one of the major ways to reduce this issue, as these often contain most of the larger holes. If you need to have thick boards, look into the z-axis CTE and Tg of your material. Lower z-axis CTE values and higher Tg values will result in reduced stress.

With PCB performance requirements constantly on the rise, ICDs will remain an issue. A better understanding of ICDs will help designers reduce the impact that they have on the performance of the board. Better PCB processing practices in drilling and desmear and selecting electroless copper will improve quality. Implementing best practices will reduce opportunities for ICDs, particularly changing connector approaches. Finally, this issue is taken seriously by the PCB suppliers, many of which are working to combat the sources behind ICD failures.

Doug Trobough is the Corporate Director of Application Engineering at Isola Corp. Doug has worked introducing new material introduction and PCB processing enhancement with Isola for five years. Prior to Isola, Doug had almost 30 years of experience building a wide variety of PCB types and interconnections systems, for Tektronix and Merix Corp., in a variety of technical positions, including CTO for Merix Corp.

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


R.P.A.S
Rapid Prototype Assembly Service
Imagineering's proprietary RPAS (Rapid Prototype Assembly Service) utilizes a combination of state of the art hardware and software to deliver assembled boards to you in as little as 1 DAY!
Our state of the art stencil-less jet printer coupled with our Proprietary Online Turnkey Quoting Engine will help you realize your design in no time!
Industry leader in Printed Circuit Board Manufacturing and Assembly Services since 1985.
imagineering,inc.
www.PCBnet.com
847-806-0003 sales@PCBnet.com
Certified Woman-Owned Small Business