



circuitcellar.com

circuit cellar

Inspiring the Evolution of Embedded Design

EDGE INTELLIGENCE ENHANCES ENERGY HARVESTING



▶ **Product Focus: Panel PCs** ▶ **Analog ICs in Industrial Systems** |

Designing Manufacturing Test Systems | **4-20 mA Current Loop Devices** |

Build a Multi-Key Electronic Flute | **Self-Organizing Wi-Fi Mesh Network** |

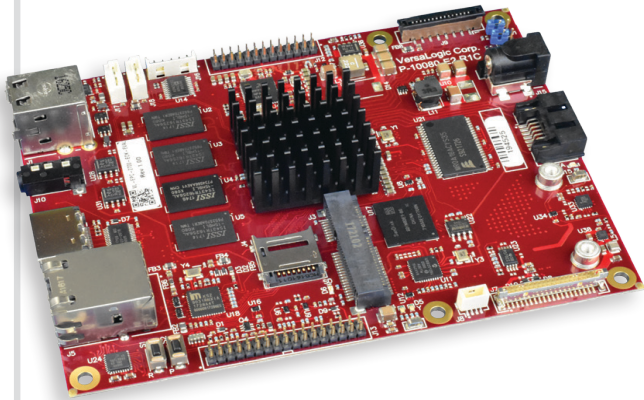
▶ **ECG Monitor Uses Cypress PSoC 6** | **Embedded System Security Live** |

Semiconductor Basics (Part 3) | **MQTT (Part 2)** ▶ **The Future of Smart Homes**



Enter to Win!

Android Demo/Eval Kit



Visit circuitcellar.com/versalogic

The kit includes:

- 7" 1024 x 600 HDMI Touch-screen flat panel display
- Tetra Single Board Computer (SBC) with Quad-core i.MX6
- Pre-loaded Android (Oreo 8.0) on MicroSD card
- Wall power adapter
- USB Hub
- Start-up guide
- Required cables

"Set-up was a breeze,
it was up and running
in less than 10 min."

– Electronics Technician



VERSALOGIC
CORPORATION

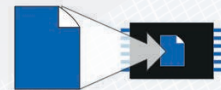


The Embedded Experts



emCompress-ToGo

Compress Data in Real-time on any Embedded System!



More Storage



More Bandwidth



Faster Updates

One Professional Compression Solution for All Applications



Data Loggers



Internet of Things



Space / Avionics



Networking



Medical Devices



Consumer Electronics

- Real-time compression
- Small footprint
- No static RAM required
- Compression of data stream
- High performance
- High compression ratio
- On-target compression & decompression

Worldwide: sales@segger.com

+49 2173 99312 0

U.S. East Coast: us-east@segger.com

+1 978 874 0299

U.S. West Coast: us-west@segger.com

+1 408 767 4068

segger.com

OUR NETWORK



VOICE COIL

LIS LOUDSPEAKER
INDUSTRY
SOURCEBOOK

SUPPORTING COMPANIES

Accutrace, Inc.	C3
All Electronics Corp.	77
CCS, Inc.	77
EzPCB	27
HuMANDATA	73
IAR Systems	33
Measurement Computing Corp.	49
Revenue Control Systems	77
SEGGER Microcontroller Systems	1
SlingShot Assembly	13
Siborg Systems, Inc.	29
Technologic Systems, Inc.	C4, 77
Texas Instruments	23
VersaLogic	C2

NOT A SUPPORTING COMPANY YET?

Contact Hugh Heinsohn

(hugh@circuitcellar.com, Phone: 757-525-3677, Fax: 888-980-1303)
to reserve space in the next issue of *Circuit Cellar*.

THE TEAM

PRESIDENT
KC Prescott

EDITOR-IN-CHIEF
Jeff Child

ADVERTISING COORDINATOR
Nathaniel Black

CONTROLLER
Chuck Fellows

SENIOR ASSOCIATE EDITOR
Shannon Becker

ADVERTISING SALES REP.
Hugh Heinsohn

FOUNDER
Steve Ciarcia

TECHNICAL COPY EDITOR
Carol Bower

PROJECT EDITORS
Chris Coulston
Ken Davidson
David Tweed

GRAPHICS
Grace Chen
Heather Rennae

COLUMNISTS

Jeff Bachiochi (From the Bench), Bob Japenga (Embedded in Thin Slices), Robert Lacoste (The Darker Side), Brian Millier (Picking Up Mixed Signals), George Novacek (The Consummate Engineer), and Colin O'Flynn (Embedded Systems Essentials)

Issue 352 November 2019 | ISSN 1528-0608

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

KCK Media Corp.
PO Box 417, Chase City, VA 23924

Periodical rates paid at Chase City, VA, 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.

SUBSCRIPTION MANAGEMENT

Online Account Management: circuitcellar.com/account
Renew | Change Address/E-mail | Check Status

CUSTOMER SERVICE

E-mail: customerservice@circuitcellar.com

Phone: 434.533.0246

Mail: Circuit Cellar, PO Box 417, Chase City, VA 23924

Postmaster: Send address changes to
Circuit Cellar, PO Box 417, Chase City, VA 23924

NEW SUBSCRIPTIONS

circuitcellar.com/subscription

ADVERTISING

Contact: Hugh Heinsohn

Phone: 757-525-3677

Fax: 888-980-1303

E-mail: hheinsohn@circuitcellar.com
Advertising rates and terms available on request.

NEW PRODUCTS

E-mail: editor@circuitcellar.com

HEAD OFFICE

KCK Media Corp.
PO Box 417
Chase City, VA 23924
Phone: 434-533-0246

COPYRIGHT NOTICE

Entire contents copyright © 2019 by KCK Media Corp. All rights reserved. Circuit Cellar is a registered trademark of KCK Media Corp. Reproduction of this publication in whole or in part without written consent from KCK Media Corp. is prohibited.

DISCLAIMER

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

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

© KCK Media Corp. 2019 Printed in the United States

INPUT Voltage

High Stakes Future for Commercial Drones

I've spoken before in this column about the commercial drone market, and how it differs greatly from both the military drone and consumer drone market segments. While the military and consumer drone realms have pretty well-established design requirements, many of the embedded electronics design decisions for commercial drones haven't been nailed down. Drones in this segment are performing all sorts of missions for construction, agriculture, security, delivery, media and many more. And commercial drones have to fly higher and longer than consumer drones. Meanwhile, a diverse industry of drone software and services has sprung up around the commercial drone space. Reliability is a major concern in commercial drones, and those service firms are very much aware that, if a drone stops working, so does their service.


At this year's InterDrone show in September, Tom Walker, founder and CEO of DroneUp, was among the keynote speakers and he provided an interesting perspective on the drone industry from a service provider point of view. DroneUp is a provider of end-to-end aerial data collection services for large, autonomous drone missions. The company delivers on-demand drone services to commercial, government and public safety organizations through its patent-pending verification platform, Mission Match.

Walker's InterDrone keynote offered a fresh perspective on the huge potential and high stakes of today's commercial drone market. A full transcript of his speech is posted on dronelife.com, but I'll share some of the highlights here. Walker started off stressing just how young and unique the drone and drone services industries are. "We started businesses not knowing what it was we would be selling to customers who didn't know what they were buying, in an industry that nobody has figured out how to regulate," said Walker. "You see, we aren't just start-up businesses and organizations in a young industry. We are a start-up industry."

Walker says he views this industry as all part of a team. This team is made up of small one- or two-person drone shops trying to build revenues, drone services providers managing global pilot fleets, brave early-adopters (users) fighting to integrate drone services into your organization, members of academia racing to design curricula to ensure we have a sustainable workforce and government bodies trying to regulate drone operations.

In contrast to other areas of embedded system design, drones have to factor in broader issues like the many safety and regulatory issues surrounding them. Drones have to operate within the same air space as manned aircraft. And the drone industry is relatively new with a regulatory landscape that's still evolving and with many safety issues still to be resolved.

From his perspective, Walker says he sees an industry hungry to evolve and get it right. "I am involved with dozens of industry boards, alliances, committees and organizations, all of which are in one way or another committed to promoting the commercial drone industry," he said, "So far this year, I have received more than 7,000 emails covering every imaginable topic and program: BVLOS (Beyond Visual Line of Sight), Large UAS (unmanned aerial systems), UPP (UAS Pilot Program), IPP (UAS Integration Pilot Program), and, as always, membership dues. The fonts and words have minor differences but the message is consistent: Help us influence policy to facilitate adoption of drone technology. Help us shape a regulatory environment that will pave the way for industry emergence and growth. The implication is subtle but clear. Only through continued efforts to guide policies and regulations will our industry ever really get off the ground."

In his keynote, Walker also stressed that technology innovation is playing a key role in the growth and sustainability of the commercial drone industry. "The best way to ensure our industry's stability is to remain steadfastly focused on operating responsibly while providing tangible value," said Walker, "BVLOS, asymmetric data protocoling, AI-enabled dispatch and the many other technologies on the roadmap will extend our capabilities and contribute to our industry's growth and sustainability." 



Jeff Child

COLUMNS

PRODUCT FOCUS

40 Panel PCs HMI Intelligence

By Jeff Child

45 Embedded System Essentials Embedded System Security Live Coverage of Two Security Events

By Colin O'Flynn

50 Picking Up Mixed Signals Bluetooth-Enabled ECG Monitor Using the Cypress PSoC 6 MCU

By Brian Millier

61 From the Bench MQ Telemetry Transport (Part 2) Bringing it All Back Home

By Jeff Bachiochi

70 The Consummate Engineer Semiconductor Fundamentals (Part 3) Transistor Topologies

By George Novacek

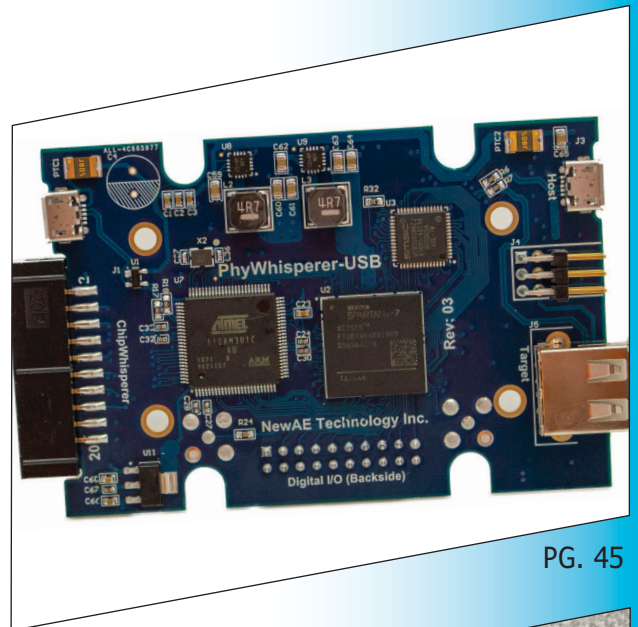
TECH THE FUTURE

79 The Future of Smart Homes The Essentials of Smart Home Security

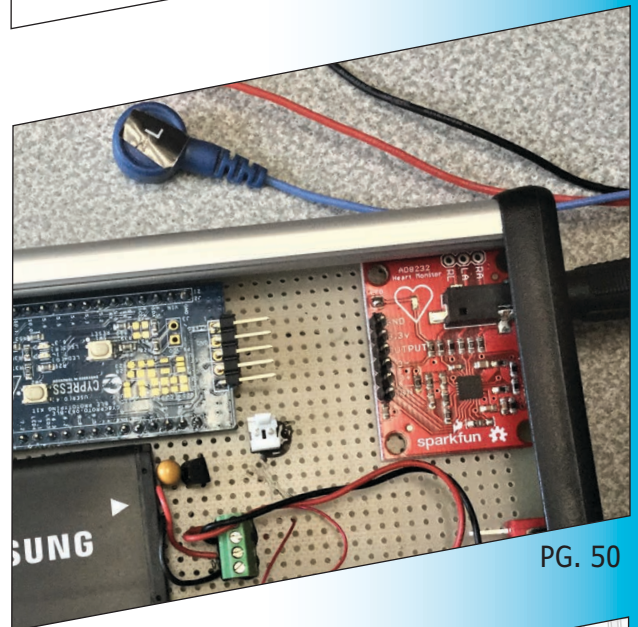
By Michelle Tate

74 : PRODUCT NEWS

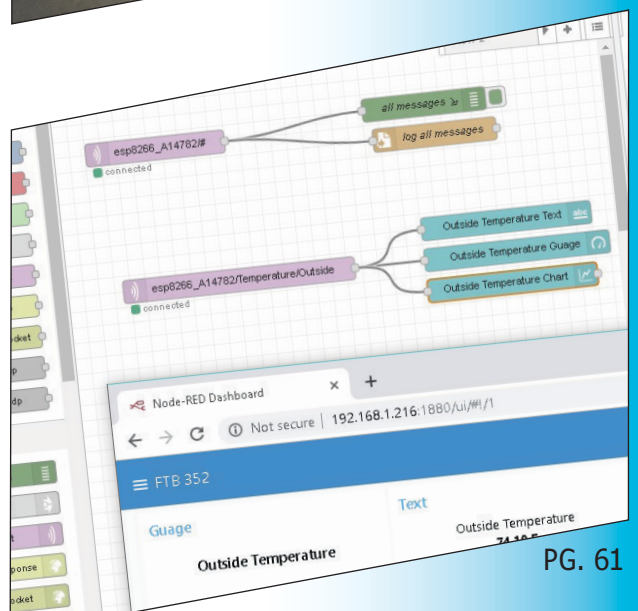
78 : TEST YOUR EQ



PG. 45

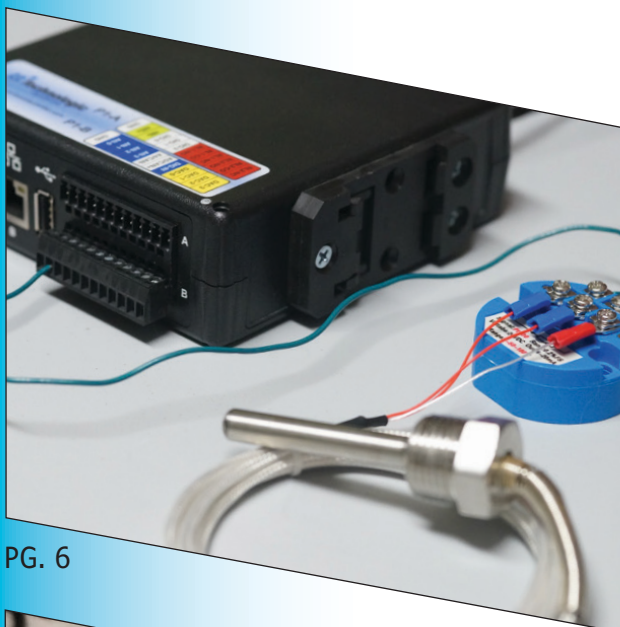


PG. 50



PG. 61

FEATURES



PG. 6

6 4-20 mA Current Loop Devices

An SBC-Based Project

By Derek Hildreth

14 Self-Organizing Wi-Fi Mesh Network

Using PIC32 MCUs

By Daniel Weber and Michaelangelo Rodriguez

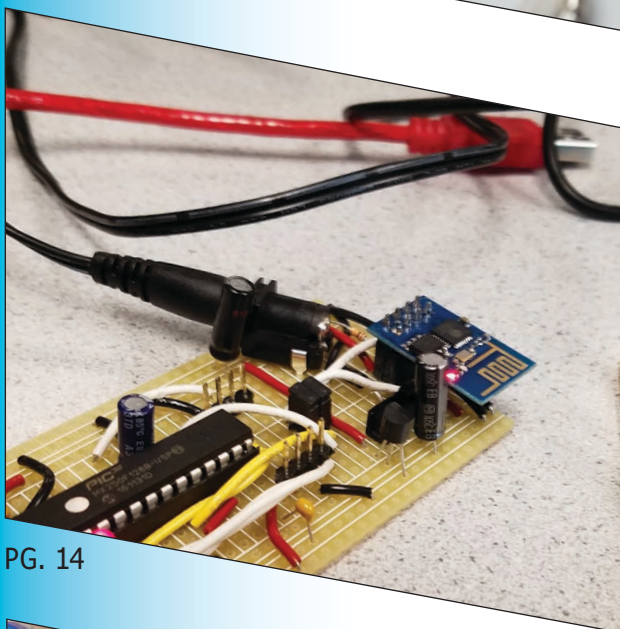
20 Designing Manufacturing Test Systems

Quality and Cost

By Nishant Mittal

24 Multi-Key Electronic Flute

Sensors and Synthesis

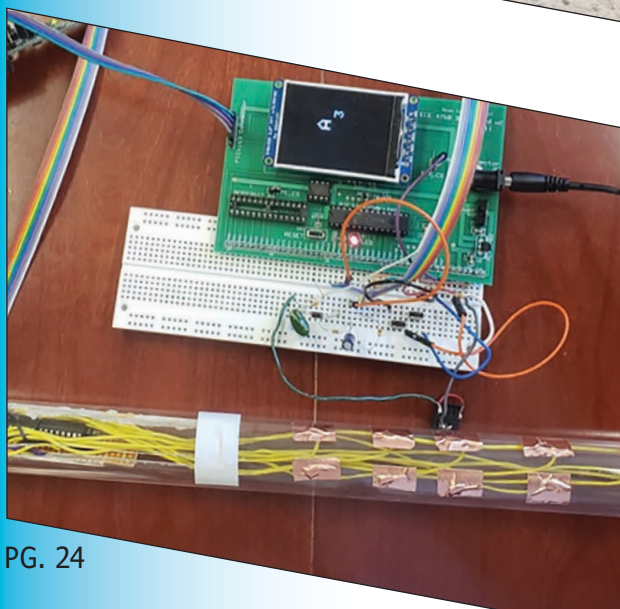
By Trisha Ray, Parth Bhatt and Qing Yu

PG. 14

SPECIAL FEATURE

30 Chip Solutions Tackle the Energy Harvesting Challenge

Self-Sufficiency at the IoT Edge

By Jeff Child

PG. 24

TECHNOLOGY SPOTLIGHT

36 Analog ICs Feed Needs of Industrial Systems

Advances for Automation

By Jeff Child

4-20 mA Current Loop Devices

An SBC-Based Project

FEATURES

By *Derek Hildreth*,
Technologic Systems

In this article, Derek helps you gain deeper understanding of 4-20 mA current loop devices and process control systems. He looks at some history, explains why things are the way they are, looks at simple example components of a process control system (sensor, transmitter, receiver) and works through a practical example with working code.

Anybody who has experience in process control knows that 4-20 mA current loop devices dominate the industry. These devices are built to transmit analog signal information through varying amounts of current, which is then read and interpreted by a receiver unit. The receiver can then be configured to display the sensor data in a human friendly format and/or perform some action. This is fundamental for automating processes in simple or complex systems.

Why 4-20 mA? The history on how 4-20 mA came about is quite interesting—going back to pre-electronic process control systems based on a 3-18 psi linear measurement scale. The low-end value of 4 mA was chosen because the necessary equipment consumes about 3 mA—so, a desire to add some wiggle drove the industry to choose 4 mA. And 0 mA wasn't chosen as the lower value so that line faults could be detected if it falls below 3.8 mA. The high end of 20 mA was chosen because anything over 30 mA is very dangerous to humans—and because 20 is a nice multiple of 4 [1].

Taking a look at an overly simplified example, let's imagine a water pump scenario where a hot water valve must be automatically opened or closed depending on water temperature in a pipe carrying mixed hot and cold water. A temperature sensor is installed in the mixed water pipe, transmitting an analog signal (most likely a signal measured in resistance). The wires from the sensor are then connected to a transmitter which converts the analog signal to a 4-20 mA current signal.

This transmitter is then connected to a single board computer (SBC) with on-board relays that has been programmed to read the 4-20 mA current signal of temperature, and convert it into units of degrees Celsius. The program continually reads the temperature value, and when it becomes greater than 41°C, the program will energize the relay connected to an electronic valve actuator, thus closing the hot water valve until the temperature has returned to nominal value. All this without any human interaction, using simple, inexpensive and low power hardware.

LET'S BUILD SOMETHING

Now with basic 4-20 mA current loop system principles out of the way, let's have some fun and deepen our understanding so we can actually begin to build something! We're going to further explore reading temperature data using a sensor and an SBC. So, grab your digital multimeter and let's get rolling!

Project Requirements:

- RTD PT100 temperature sensor [2]
- PT100 4-20 mA transmitter, -50 to 100°C, 24 VDC [3]
- SBC or microcontroller (MCU) with ADC port. The TS-7680 [4] is used in this article because it has built-in 4-20 mA support, but the principles can be applied globally. The SBC or MCH can have ADC ports with 4-20 mA loop support or not
- Digital multimeter with current measurement (mA); Optionally, temperature measurements to confirm temperature conversion
- Small Philips and flathead screwdrivers for screw terminals
- Hookup wire

The Sensor: We'll be using a three-wire resistance temperature detector (RTD) temperature sensor (**Figure 1**). More specifically, we'll be using a PT100, which means the resistance at 0°C is 100 Ω and measures from -200°C to 600°C. We've chosen a three-wire RTD sensor mainly because of its min and max range, accuracy, availability, packaging types and lead wire range. We can install this sensor in a pipe a long distance away from a facility without accuracy loss.

The third wire in an RTD is used to compensate for the resistance added by the length of the wires. This allows for maximum cable lengths up to 200 feet. If longer cable lengths are required, then it is best to use a temperature transmitter—like we talk about in a moment—which converts temperature into a current output or digital signal. Pro Tip: The main players in temperature sensors are thermocouples, thermistors, RTDs, and one-wire (DS18B20). Naturally, each have their advantages and disadvantages [5]. Think about your application requirements when choosing.

One advantage of using an RTD sensor is that it has a linear temperature vs resistance output [6]. That means that it's easy to calculate a slope intercept formula to calculate final, human-readable values. Adventurous folks who want to build their own circuit (in other words, voltage divider or Wheatstone bridge) instead of buying a transmitter will appreciate the linearity. As for the rest of us, we'll use a transmitter.

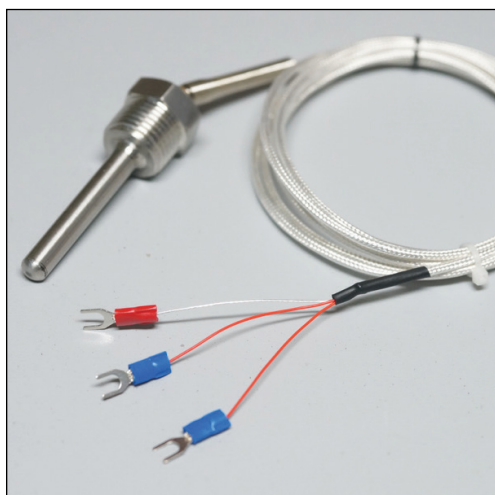


FIGURE 1

Three-wire RTD PT100 temperature sensor

The Transmitter: In our application so far, we've determined that we want to use a 4-20 mA current loop signal. We've also determined that an RTD PT100 sensor suits our application nicely. Now, we need to find a transmitter that will take an analog signal from the sensor and translate it into a nice 4-20 mA current loop signal. Let's consider our water temperatures. Water will be flowing through a pipe, so let's figure we won't be measuring anything below 0°C (freezing). Let's also figure we won't be sending anything more than 90°C through it (water boilers for home heating are typically 80°C). A quick trip to the store will yield a PT100, 4-20 mA transmitter with a -50 to 100°C range and three-wire input (**Figure 2**). It will need to be supplied with 24 VDC [7], which is very common in process control systems.

We're almost there. Now, we need to choose a receiver in order to read and convert the 4-20 mA signal into human friendly values and then do something with them.

The Receiver: The final piece of equipment in our application is the receiver. Its main job is to make sense of the 4-20 mA signal



FIGURE 2

RTD PT100 24 VDC transmitter



FIGURE 3
A TS-7680 single board computer with enclosure

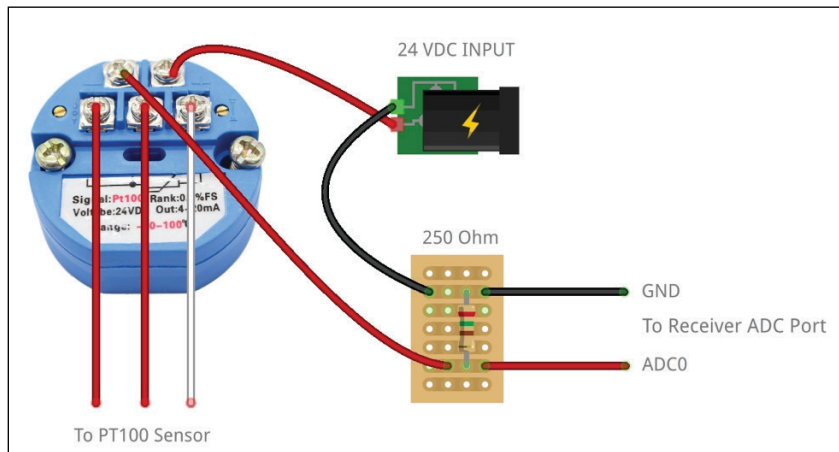


FIGURE 4
Wiring diagram for connecting transmitter to standard ADC port without 4-20 mA current loop support

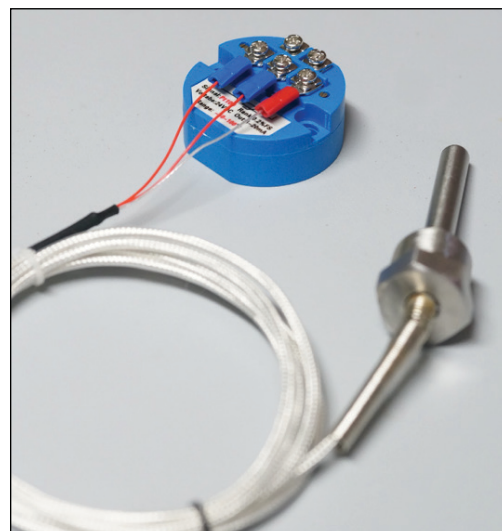


FIGURE 5
RTD temperature sensor probe connected to 4-20 mA transmitter

by digitizing it so that we can program it to perform some action based on the signal. Actions include things like displaying the temperature in °C or °F on a display, toggling DIO based on set thresholds, energizing relays for system control, serving up a web application for the world to see (think Internet of Things, IoT or IIoT) or some combination of all these things. The receiver could be a simple circuit (555 chip, transistors, voltage divider, relay), an MCU (Arduino) or an SBC.

It just so happens that the Technologic Systems' TS-7680 [8] falls into this application very nicely, with 4× ADC inputs supporting 4-20 mA current loop (Figure 3). You can conveniently power it with the same 24 VDC supply as the transmitter. It also has a lot of digital output channels available for controlling external relays or devices, along with a load of other interfaces like Modbus, CAN, RS-232, RS-485 and more. The feature that captures my attention is the networking abilities, including dual Ethernet, Wi-Fi and Bluetooth. The TS-7680 is well suitable for many interesting applications [9] and meets the requirements for this project article.

If you choose to use some other receiver device, just make sure it has an ADC port. If the ADC port does not support 4-20 mA current loops, don't fret. It's actually quite simple to add support using a single load resistor for measurement. By placing a 250 Ω resistor between GND and ADC, we can interpret the 4-20 mA output as a voltage drop across the resistor (Figure 4).

Why 250 Ω? Let's apply Ohm's Law: $V = IR$, and assume the max voltage we want to supply to our ADC channel is 5 VDC. We know the max current is 20 mA, so we can now apply the formula to come up with a resistance value: $R = V/I = 5/0.020 = 250 \Omega$. This works out nicely for the low end as well, since $V = IR = 0.004 \times 250 = 1 \text{ V}$. This equates to a very nice 1 VDC to 5 VDC range. However, going back to fault detection when current is below or above 4 and 20 mA, we may opt for a lower resistance value, like 225 Ω (0.9 VDC to 4.5 VDC) or, in the case of the TS-7680's built-in load resistors, 240 Ω (0.96 VDC to 4.8 VDC).

CONNECTING THE PIECES

Sensor and Transmitter: Connecting our temperature sensor to it is easy enough. The two red wires are common and the clear wire is what we'll call the sensor signal. The clear wire will be attached to the far right terminal while the two red wires will be attached to the remaining terminals in that row (Figure 5). Hooking the transmitter up to power might make your brain twitch a little. Instead of negative wire to negative terminal and positive to positive, we're going to be

connecting the negative terminal in line with a receiver in order to measure the current.

Let's start out simple and use a digital multimeter (DMM) to prove the concept and get our minds wrapped around it. So, the positive wire of your 24 VDC power supply will go to the positive terminal as usual, but the negative wire will be connected to the COM port of your DMM. Then, you'll connect the mA port of your DMM to the negative terminal (**Figure 6** and **Figure 7**).

This is the exciting part! Grasp the sensor fully with your hand and watch the mA measurement rise with the temperature of your hand. Dunk the sensor into a mug full of ice water and watch it drop. This validates everything we've been working on so far! Now that we have proof of concept out of the way and we've validated our sensor and transmitter, let's get the receiver, in our case the TS-7680 SBC, hooked up.

Transmitter and Receiver: Just as we connected our multimeter inline to the 4-20 mA current loop transmitter, we're going to do the same for the ADC channel of the receiver. Remember, it's here in the receiver that we're going to convert the analog signal into a digital one so that we can translate it into a meaningful number in °C and then do something with it.

Without going into analog signal theory too much, you can easily have the transmitter and receiver installed on opposite sides of several football fields and still connect them using a shielded wire (to prevent EMI and essentially turning the long wires into an antenna). So long as there is still voltage potential at the receiver for the 240 Ω resistor, the receiver can measure the voltage drop. For example, using 24 AWG wire [10] and 24 VDC, that turns out to be about 35,000 feet (19 VDC drop leaving 5 VDC potential at the 240 Ω resistor of the receiver).

Since we're working with the TS-7680 in this guide, we'll be taking advantage of the variable input voltage and supply it and the transmitter with 24 VDC from a single source. We only need to run a single wire to the AN-0 pin on the bottom connector (**Figure 9**). This simplifies wiring for the case of our TS 7680, but we might find that there are a lot of systems that do not have a variable input voltage as high as 24 VDC. More commonly, it'll be 5 VDC or 12 VDC. So, let's see what a separate power supply would look like in **Figure 10**.

Here's a pitfall to consider: Keep in mind, using separate supplies in this way requires them to be able to be grounded together. Using separate isolated supplies can cause a ground potential which can give incorrect readings at best, or damage equipment at



FIGURE 6
Multimeter measuring current from the transmitter before connecting to the receiver

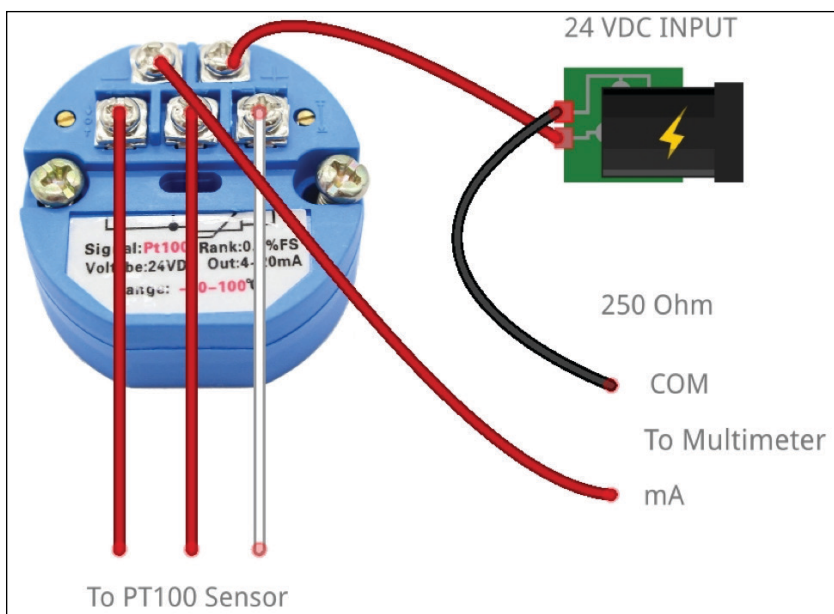


FIGURE 7
Wiring diagram for measuring current from the transmitter

ABOUT THE AUTHOR

A small-town Montana boy, born and raised, with a passion for programming and embedded systems. Derek Hildreth has been working for Technologic Systems (in various capacities) since 2010 starting as an intern embedded engineer. Off the clock, he's an avid skier and backpacker who slays double black diamonds, conquers mountain peaks, and just generally loves life at 10,000 feet. What he loves most about his career at Technologic Systems is the extended family cultural feel and the opportunity to wear many hats as responsibilities change.

worst. It would be best to use a supply that's able to offer multiple voltage taps with the same reference ground.

There. Now, we have two power supplies—one for the receiver and one for the transmitter. We could opt for a 12 VDC to 24 VDC converter [11] as well, but we'll abstract that away for now. At this point, we're ready to dive into programming the receiver, but while we're still on the topic of hooking everything up, **Figure 11** shows an example of how we'd hook up multiple transmitters. Okay! Now, we're ready to move on to even more exciting stuff: programming the receiver.

PROGRAMMING THE RECEIVER

If you're not following along with a TS-7680 or similar Technologic Systems' product, that's okay because this article is generic enough to apply to other boards. Just keep your product's manual handy! For those of you who are anxious to see the code, you may jump to the TS-7680 4-20 mA Current

Loop Example Code gist on GitHub [12]. I'll explain it in a little more detail below.

Read the ADC Value: Get your receiver into a state where you can program it. For the TS-7680 running Linux, this means powered on and connected to a serial console (see also TS-7680 getting started guide [13]). Chances are, your receiver came with some example code for how to read in values from the ADC channels. For the TS-7680, we'll be using the `mx28adcctl.c` example code [14] within the TS-7680 Utility Sources repository on GitHub [15] as a base.

Copy or download the example code onto your receiver and update it with the correct pin locations. Since the code will likely be reading voltages, you might also want to add a conversion back to milliamps or microamps based on the load resistor you used for measurement. The TS-7680 has a 240 Ω load resistor between AN-0 and GND, so the conversion from milliamps to microamps looks like:

```
uA = (((meas_mV)*1000)/240); //
Ohms law: I = V/R
```

Compile your program and test it! Since we have our measurements from our multimeter, we can do a sanity check that our ADC value is good. On the TS-7680, we've named this program `getadc` (`getadc.c`, `Makefile`) [12]. This is what the output looks like:

```
root@ts7680:~/getadc# ./getadc 0
mV: 2718
uA: 11325
```

This is very cool, because so much has been building up to this moment! We're very close to finishing up by converting these raw numbers into something meaningful. Tip: You could stop here if you don't care what the units are and want to use the 4-20 mA signal directly, but it does help to have meaningful, human-readable values represented.

Convert Signal to Meaningful Units: The final piece of code! Instinctively, I separated the `getadc.c` code [12] from the system control code called `rtdTemp.sh` [12]. You could tie these two into the same program if you wanted, but I find it easier to maintain modular code. Plus, you can use the other ADC ports for different sensors, each requiring their own conversions and set of instructions but all rely on the `getadc.c` code to get there.

Right to the point, the 4-20 mA to temperature conversion formula you'll want to use is:

```
tempC = (mA - 9.333) / .107
```

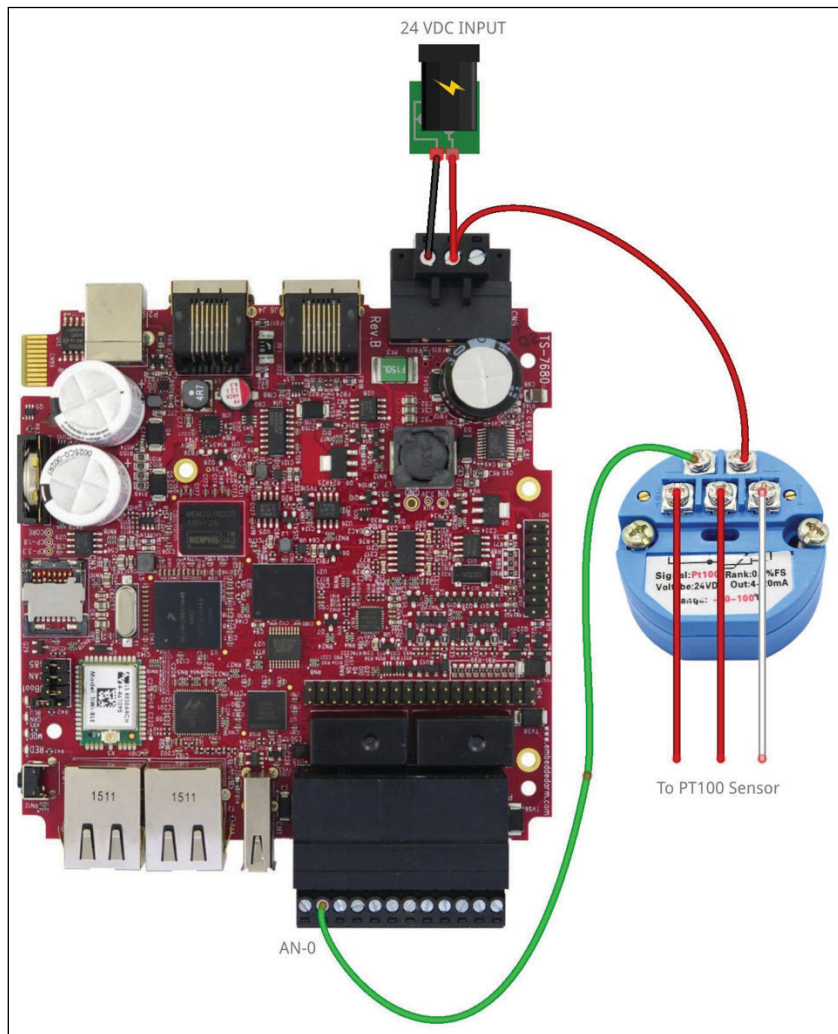


FIGURE 9

Wiring diagram showing single 24 VDC power supply and 4-20 mA signal connection.

Why? How? Think back to our high school days. We have a linear temperature vs current scale thanks to our sensor and transmitter choices. We want the slope intercept form of this line so we can solve for X (temperature) given Y (mA). Using a spreadsheet, we can calculate this using INTERCEPT() and SLOPE() functions knowing -50°C is 4 mA and 100°C is 20 mA. Go to [17] and take a look at the RTD PT100 4-20 mA Transmitter Temperature Conversion spreadsheet I used to get this formula if you're still curious or want to verify it. Make a copy of it and modify it to fit your needs.

We'll be applying that formula in our system control script, rtdTemp.sh, which will read the mA value from the output of getadc, convert it to °C (and °F), and display it. Here's what the conversion looks like in code:

```
#!/bin/bash

uA=$(getadc 0 | awk 'FNR == 2 {print $2}')
mA=$(echo "scale=3; $uA / 1000" | bc)

tempC=$(echo "scale=1; ($mA - 9.333) / .107" | bc)
tempF=$(echo "scale=1; ($tempC * 9/5) + 32" | bc)

echo "Temp (C): $tempC"
echo "Temp (F): $tempF"
```

We're using bc because it supports floating point math whereas eval does not.

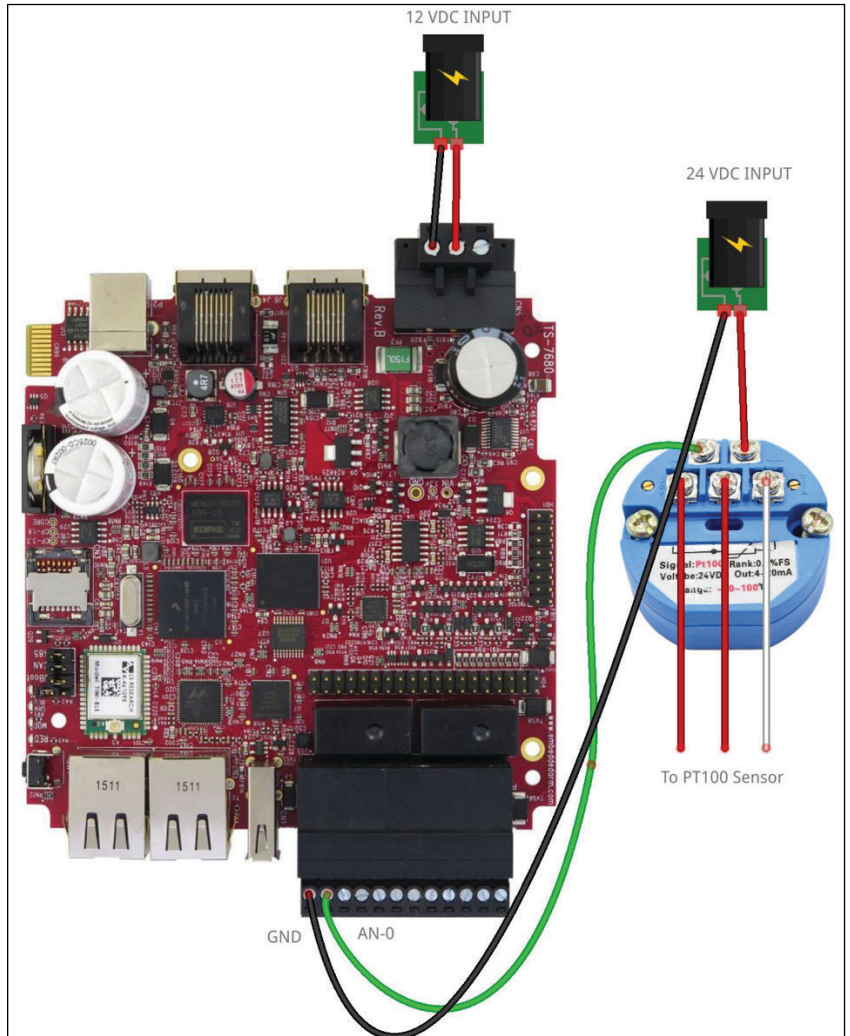


FIGURE 10 Wiring diagram showing multiple power supplies and 4-20 mA signal connection.

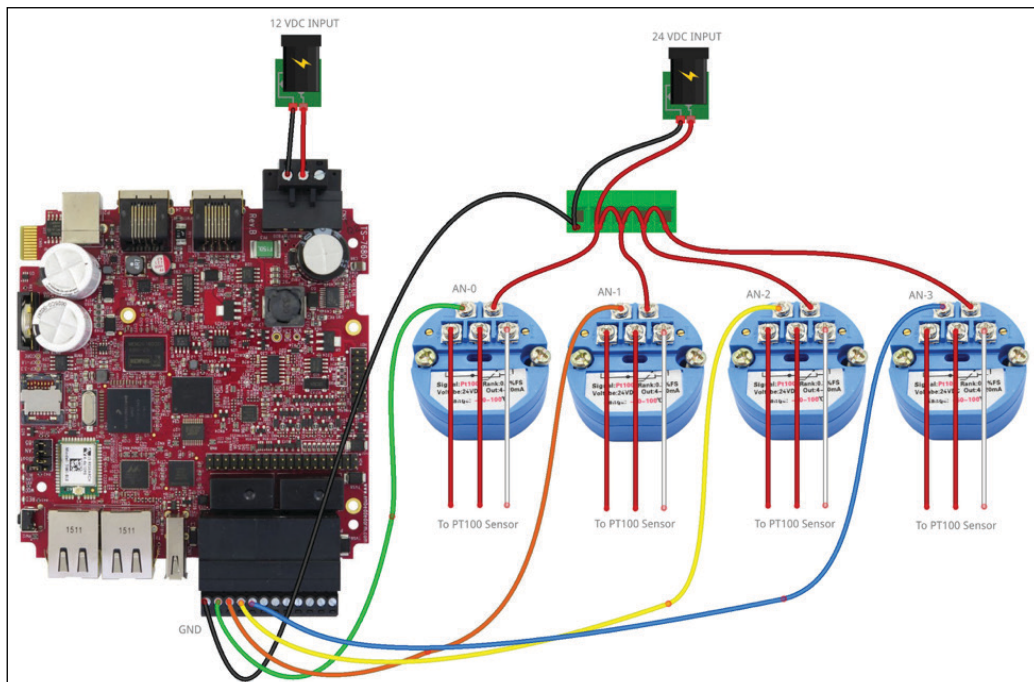
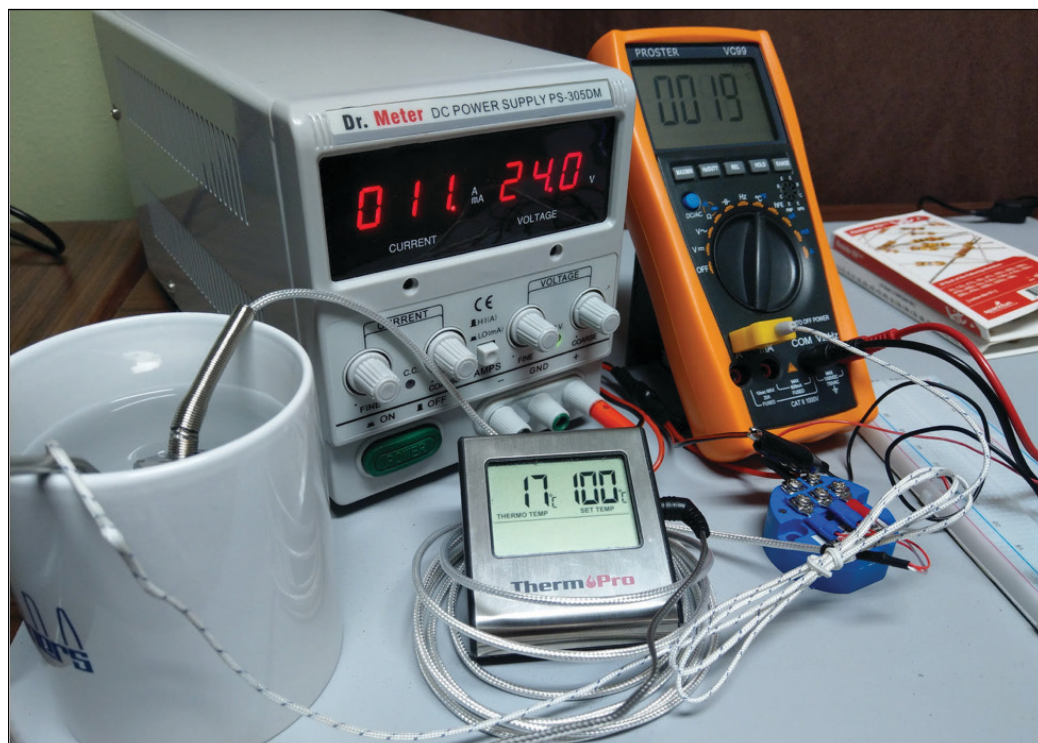


FIGURE 11 Wiring diagram showing multiple temperature sensors connected.

FIGURE 12

Testing the sensor accuracy and mA values using two different thermometers.



Running the script will yield the output:

```
root@ts7680:~/getadc# ./
rtdTemp.sh
Temp (C): 20.1
Temp (F): 68.1
```

Momentous! We're seeing the meaningful value for the first time! Congrats and great job so far!


Take some time and verify your output. Do the values make sense and are they accurate? Use other thermometers you might have to do sanity checks (**Figure 12**). For example, use the temperature measurement from your DMM. Grab cups of water (hot, cold and luke warm) and stick the PT100 sensor and DMM sensor in each of them. Are they getting the same values? I also grabbed a kitchen thermometer to get a third opinion. I found that the DMM value was about 2°C higher than what the kitchen thermometer and the PT100 sensor was reporting, so it's good to have another opinion handy. From here, we can continue smoothly sailing through the rest of our application, wherever it may take you!

Congratulations! We've now reached the end of our example application of working

with a 4-20 mA current loop sensor device to get temperature data. At this point, you should be ready to move onto the next component in your application, whether that be hooking the receiver up to a relay to open or close a valve or setting up a web application to display temperatures in JSON format for other systems to consume. Whatever it is, you're well on your way! Pick your favorite compiled (C/C++) or scripting language (Python, Node.js, Bash and such) and keep moving forward.

WRAP UP

This article should have left you with a deeper understanding of 4-20 mA current loop devices and process control systems. We took a look at some history, why things are the way they are, looked at simple example components of a process control system (sensor, transmitter, receiver) and even worked through a practical example with working code.

There's still more to learn, and if you're hungry for it, take a look at the excellently written and illustrated "Back to Basics: The Fundamentals of 4-20 mA Current Loops" series on predig.com [18]. Also, be sure and review Technologic Systems' product line, because most of its product is fit for industrial applications like process control systems with multiple ADCs, onboard relays, industry standard connectors, industrial temperature range and more. Now, go enjoy planning and building out the rest of your control system! 

For detailed article references and additional resources go to:

www.circuitcellar.com/article-materials

References [1] through [18] as marked in the article can be found there

RESOURCES

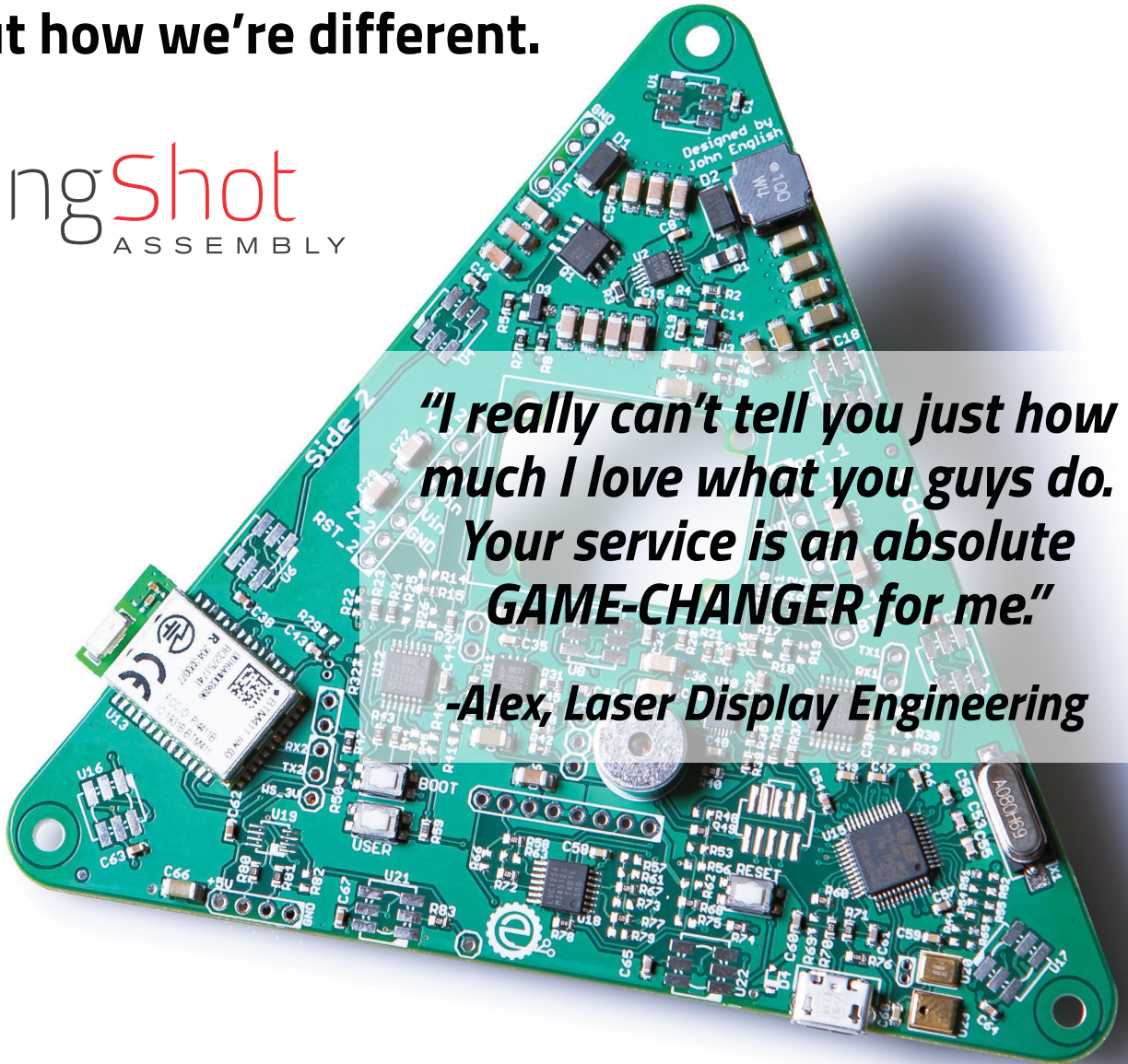
Technologic Systems | www.embeddedarm.com

\$1000 IN FREE LABOR ON YOUR FIRST ASSEMBLY

GET QUOTE NOW

circuitcellar.com/slingshot

Speed. Quality. Service.
Find out how we're different.



*"I really can't tell you just how much I love what you guys do. Your service is an absolute **GAME-CHANGER** for me."*

-Alex, Laser Display Engineering

THE QUICK-TURN PCB PROTOTYPE EXPERTS!

sales@sassembly.com | (720) 778-2400



Self-Organizing Wi-Fi Mesh Network

Using PIC32 MCUs

FEATURES

Gone are the days when networking embedded devices was a big deal. And today, such devices can be linked in powerful mesh networks over wireless protocols. In this article, learn how these two Cornell students used Microchip PIC32 MCUs and Espressif's ESP8266 Wi-Fi module to create a mesh network of wirelessly connected devices. The mesh network is able to configure itself, and requires no manual intervention to connect the nodes.

By
Daniel Weber and Michaelangelo Rodriguez

In this project, we created a mesh network of Microchip PIC32 microcontrollers (MCUs) that were connected to each other wirelessly through ESP8266 Wi-Fi modules (**Figure 1**). The primary objective for this project was to create a self-contained wireless mesh network of MCUs. The criteria were that the network should be able to add new nodes as they turn on, and should be robust to nodes disconnecting.

We considered several different wireless technologies when designing this system. To create a network of nodes, we needed multiple wireless devices that could be connected to one another simultaneously. We considered several different types of wireless technologies, including Bluetooth, packet radio and Wi-Fi. Preliminary investigations revealed that most hobbyist Bluetooth modules had relatively short ranges, and multiple Bluetooth modules couldn't be connected at once. Most of the packet radio modules that we found could only be configured as transmitters

or receivers, and multiplexing those nodes would have resulted in significant packet drop [1]. We settled on the ESP8266 Wi-Fi module from Espressif Systems, because it met the requirements for this project and has a relatively long range.

The hardware for our project was designed around the PIC32 MCU. We designed a schematic for connecting the PIC32 and the ESP8266 through their serial connections. Our software was designed as a layered architecture. This type of architecture is common in network stacks and allows independent implementation and optimization of individual layers. This approach can help simplify the design process and make the implementation easier.

HARDWARE DESIGN

The primary hardware components of a node were a PIC32 MCU and an ESP8266 Wi-Fi module. Given our previous experience with the PIC32, it proved to be an inexpensive,

FIGURE 1 (ABOVE)

Two complete nodes. The left one is turned on and is actively scanning for other nodes.

powerful chip that would allow us to meet the demands of the mesh network. The ESP8266 Wi-Fi-module is a device that is known for its versatility, cost-effectiveness and ease of use.

The PIC32 and the ESP8266 both require a 3.3V power supply. To meet these demands, we devised a section within our board dedicated to regulate any 4.2-12 V power supply to 3.3 V. As confirmed with a voltmeter, the output of the voltage regulator was a fixed 3.3 V. This output was connected to the VCC pin on the ESP8266 Wi-Fi module and then connected to the appropriate pins on the PIC32.

One of the peripherals added to our nodes was an LED. The LED was useful for testing our algorithms and visualizing the behavior of our system. The LED would constantly blink when the node was searching for a connection, and then stay lit up when it connected to another node. Therefore, if we expected a connection to occur or had an unexpected connection, the LED would be an easy visual to identify an issue. Likewise, if the LED acted according to the expected behavior, it would help confirm the functionality of our node.

As shown in **Figure 2**, each of our nodes had four sockets: the Microstick socket, the UART socket, the Wi-Fi socket and the PIC32 socket. These sockets were mainly composed of DIP (dual in-line package) sockets and male headers. Having sockets for our most important parts allowed us to easily swap out components. The integration of these sockets allowed us to replace faulty parts with relative ease.

Another key aspect of the hardware design was the inclusion of the Wi-Fi debug jumpers. As shown in the Figure 2 schematic, pin 1 of the Wi-Fi debugger is connected to RB7, while pin 2 is connected to RA2. This wiring is how we intended the node to be connected for normal use. Since RB7 was connected to TX on the ESP8266, and RA2 was connected to RX, the inclusion of Wi-Fi jumpers led to fairly easy debugging on the Wi-Fi module. However, we could also disconnect the Wi-Fi and PIC UART modules and then use a cable to communicate directly with the Wi-Fi module from a PC. As explained later, one of the times we had to use this direct communication with the Wi-Fi module was when we flashed the firmware.

FIRMWARE AND SOFTWARE

The firmware on an ESP8266 module determines the commands we can give to the module over UART. To get the most up-to-date commands working on the ESP8266 modules, we needed to flash the latest firmware from Espressif [2]. This required the ESP8266 to be put in flash mode, which was done by pulling the GPIO_0 pin low during reset. Then, we

used a serial connection to a computer with the firmware-flashing software to load the new firmware into the ESP8266. We used a USB-to-UART cable to connect the ESP8266's RX and TX pins to a computer, and used Espressif's esptool to flash the firmware.

The software design was largely influenced by the constraints of the ESP8266 Wi-Fi modules. Wi-Fi devices typically are configured either as stations or access points. A station is a device such as a computer, which can connect to one Wi-Fi network at a time. An access point is something like a router, which allows many stations to connect to it and acts as a hub for connecting Wi-Fi devices. The ESP8266 Wi-Fi modules can be put in a third mode, which is a hybrid of the two. A single chip can act as both a station and an access point. However, it limits the total number of connections to five.

A station can only connect to a single access point. This means that each Wi-Fi module can make only one connection to another module. A Wi-Fi module can have up to five other modules connect to it, but an individual module can only make one connection. If each Wi-Fi module is treated as a node in a graph and each connection is treated as an edge in that graph, this tells us that the number of

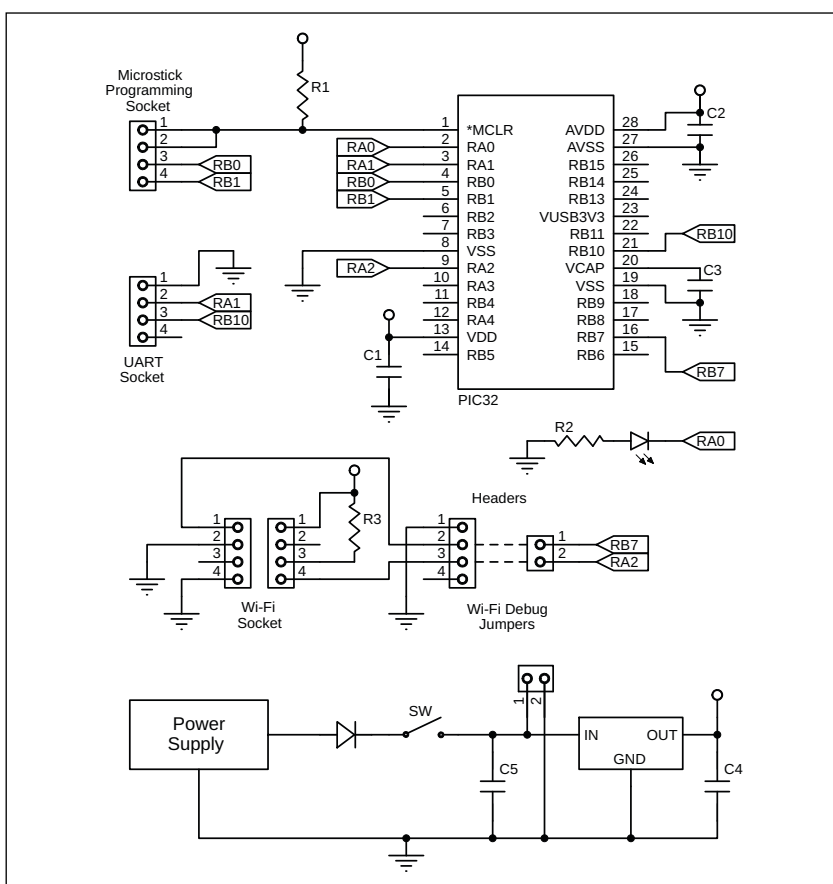


FIGURE 2
Mesh network node schematic

edges in our network is limited to the number of Wi-Fi modules in the network. This means we can't make a very fault-tolerant network, since we can have at most one loop in our network. For this reason, we decided to focus on creating software that tries to interconnect as many devices as possible. To accomplish this, we split the software into four logically separate layers: Serial, Wi-Fi, Routing and Application. Now let's discuss each of these four layers.

SERIAL LAYER

At the bottom of the software stack is the serial layer. This layer was responsible for communicating with the ESP8266 and exposing a simplified API for sending and receiving data from the Wi-Fi module. The UART hardware on the PIC32 has a buffer for up to eight characters, but if the buffer doesn't get read, subsequent characters will be dropped by the UART module. This becomes an issue, because the ESP8266 can sometimes send data over UART when we aren't expecting it—such as when it receives a message from another Wi-Fi module. To ensure that all characters that come in over the UART are stored, we used the PIC32's DMA controller.

We configured one of the DMA channels to move data from the UART RX queue into a large buffer statically allocated in the PIC32's main memory. The DMA controller automatically wraps back around to the beginning of the buffer once it has been filled. In this sense, the buffer is treated as a ring buffer. To keep track of the write head of the ring buffer, we set up an interrupt that incremented a write pointer, which fired whenever a cell/byte was transferred using DMA. When we wanted to read data from the buffer, we waited until the write pointer advanced past the read pointer, then marched the read pointer through the data of interest.

Subsequently, we abstracted this functionality into two functions that could be used by the layer above the serial layer to communicate with the ESP8266. The first function sent a string of characters to the ESP8266 over UART. The second used the

above method of waiting for the write pointer to advance past the read pointer to read data from the ring buffer and return the data to the layer above. This abstraction hid the complexity of DMA and UART, and allowed the next layer to concern itself only with the bidirectional communication stream between it and the ESP8266.

Wi-Fi LAYER

The layer above the serial layer is the Wi-Fi layer. This layer is mainly concerned with setting up the Wi-Fi module, handling connections and disconnections from stations and access points and receiving messages from other Wi-Fi modules. All communication with the ESP8266 is done by issuing AT commands to the device over UART and listening for a response. We were able to obtain a full list of the supported AT commands for the version of the firmware that we flashed onto the devices [3].

The Wi-Fi layer issues several AT commands when setting up the ESP8266. First, it sets the Wi-Fi module into the hybrid station+access point mode we discussed earlier by issuing the following command:

```
AT+CWMODE=3
```

Next, the Wi-Fi module gets its MAC address, which the rest of the software uses as a unique identifier for this node. It does this by executing the following command and listening for a response:

```
AT+CIPAPMAC_CUR?
```

During testing, we hard-coded an IP address for each access point. However, we discovered that there were issues connecting two Wi-Fi devices with the same IP addresses. To correct these problems, we gave each device an IP address based on its MAC address, where the %d is the lower 8 bits of the MAC address:

```
AT+CIPAP_CUR="192.168.%d.1","192.168.%d.1","255.255.255.0"
```

Next, we needed to allow multiple connections with the Wi-Fi module. We also found in the documentation that the multiple connections mode was required to start up a TCP server on the Wi-Fi module:

```
AT+CIPMUX=1
```

Then, we initialized the TCP server on port 80:

```
AT+CIPSERVER=1,80
```

For detailed article references and additional resources go to: www.circuitcellar.com/article-materials
References [1] through [4] as marked in the article can be found there.

RESOURCES

Espressif Systems | www.espressif.com

Microchip Technology | www.microchip.com

SparkFun | www.sparkfun.com

And finally, we set the SSID of the WiFi module so that other nodes could find it, where %d is again the lower 8 bits of the MAC address:

```
AT+CWSAP_CUR="ESP8266-Mesh-%d"
```

The primary reasons for using a TCP server instead of a UDP server were that we wanted reliable packet delivery between nodes, and we also wanted to have knowledge about the state of connections. TCP perfectly fits the bill for these requirements as a reliable, connection-oriented, message delivery protocol. This completed the setup portion of our code.

Next, we abstracted several AT commands into simple functions. The first was a function to scan for Wi-Fi modules to connect to. The ESP8266 has a command to return a list of all nearby Wi-Fi access points:

```
AT+CWLAP
```

We created a function that would invoke this command and filter the results to return only a list of Wi-Fi access points with SSIDs starting with "ESP8266-Mesh-". When a node has found a node to which to connect, it needs to do two things. First, it needs to connect to the node's access point, which is given by the SSID in the list returned by the scanning function:

```
AT+CWJAP_CUR="<access point SSID>"
```

Second, it needs to connect to the TCP server running on port 80 on that node:

```
AT+CIPSTART="TCP", "<other node's ip address>", 80
```

Last, we created a function that would send messages between two connected nodes. This function first invokes the command:

```
AT+CIPSENDERBUF=<connection id>,<data length>
```

This command tells the Wi-Fi module which connection it should send the data to, and how many bytes the data are. Then, the function sends each byte of the message to the Wi-Fi module.

When a Wi-Fi module receives data from another Wi-Fi module, or when another Wi-Fi device connects to it, the ESP8266 sends out messages over UART indicating the event. We set up a loop in our code to constantly listen for these events and invoke event handlers when the events were detected.

ROUTING LAYER

The layer above the Wi-Fi layer is the routing layer. It stores the network topology as a graph, and sends messages to the routing layer of other nodes to construct the graph. We considered an on-demand routing approach, in which each node only knows about its direct neighbors and then sends out special packets to discover paths to other nodes. However, we realized that mesh network applications would want to know about the topology of the network to optimize the connectivity of the network. Therefore, we decided to make a custom routing algorithm that used special messages to alert the network about the addition and removal of edges in the network. This is similar to the way some link-state routing protocols are implemented.

When a station "S" connects to an access point "A," it may be the case that the two nodes are on separate sides of a partitioned

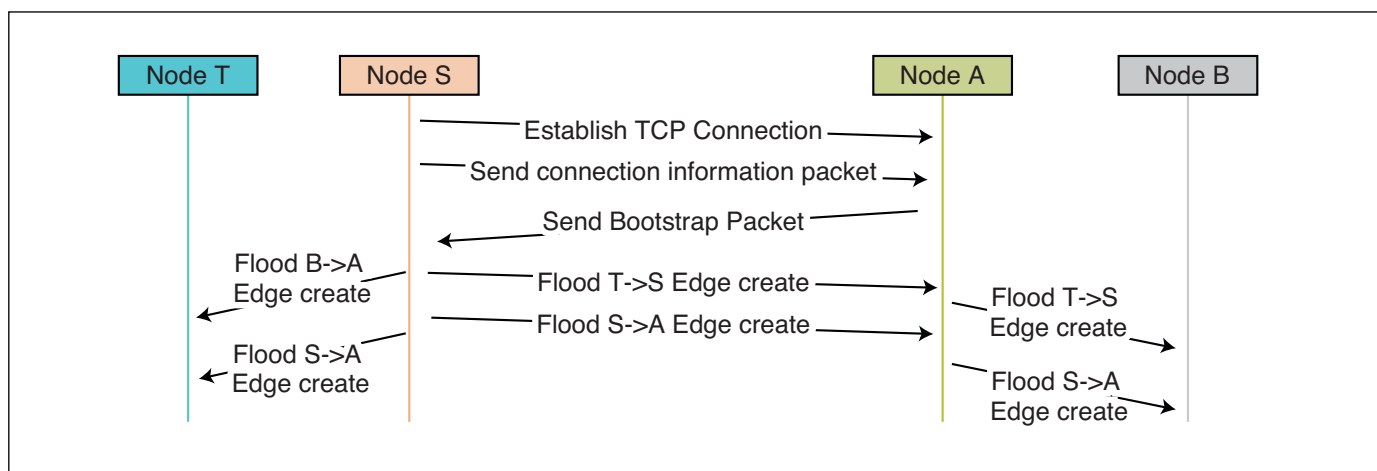


FIGURE 3 The messages created during a connection event. Time flows from top to bottom. All the edge-creation messages originate from Node S.

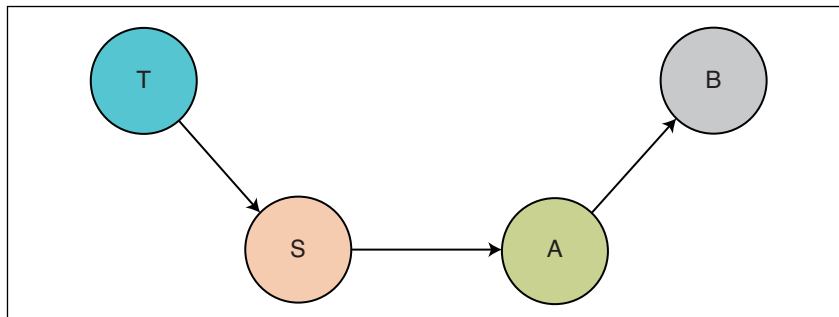


FIGURE 4

The network topology stored in every node after nodes A and S are connected. The edges between nodes point from a station to the access point to which the station is connected.

network. Therefore, they must exchange their current network topology graphs, to merge their two network graphs. Instead of having both nodes be responsible for this, only the station node S receives this so-called “bootstrap packet” from the access point (node A). This bootstrap packet contains node A’s current network topology graph. Node S will then figure out the differences between A’s graph and its own graph. Edges that are in A’s graph but not S’s graph will need to be flooded to S’s side of the network. Edges that are in S’s graph but not in A’s graph will need to be flooded to A’s side of the network. Finally, node S will flood the new S-A edge to the whole network.

As a concrete example, consider the following scenario. Node T is connected to node S and node A is connected to node B. The network graph of T and S thus consists of nodes S and T connected by an edge. The network graph of A and B consists of nodes A and B connected by an edge. Next, node S connects to node A. An overview of the messages sent between the four nodes is given in **Figure 3**.

As described above, the access point A sends the bootstrap packet back to node S, and then node S initiates the flood of messages to get every node in the network up to date on the new network topology. After the flood of messages has subsided, all four nodes in the network contain the same network topology graph: node B connected to node A, node A connected to node S, and node S connected to node T. The resulting topology graph for

this scenario is shown in **Figure 4**.

To stop messages from circulating infinitely throughout the network, we use sequence numbers. Each message that a node creates is given a unique sequence number, which allows other nodes to identify and drop duplicate messages. The routing layer also implements an algorithm for sending directed messages between two nodes. Since the routing layer stores a graph of the mesh network, it can use a shortest-path algorithm to route messages through the network from a source node to a destination node. The algorithm we used was a breadth-first search algorithm. Whenever a node receives a directed message, it first checks to see if it is the intended recipient. If it is, then it passes the message up to a higher layer. Otherwise, it finds the shortest path between it and the destination node, and sends it along that path. It also checks the sequence number in the directed message, to prevent a message from being sent in a cycle forever.

APPLICATIONS AND TESTING

The serial, Wi-Fi, and routing layers formed the core of our mesh network software. We decided to build a few simple applications on top of the core software to demonstrate its capabilities. The first application we built was a way to view the network graphically as nodes came online. To do this, we set up a simple loop that would constantly use the scanning function exposed by the Wi-Fi layer to scan for other modules. If a module was found, the application would then tell the routing layer to connect to that device. Because the nodes used Wi-Fi to communicate, we could connect to the mesh network using any Wi-Fi-enabled device. We connected a laptop to a node in the mesh network as if it were a regular access point. We implemented the same protocol that we created on the nodes for the laptop, essentially turning the laptop into another node. This allowed the laptop to receive the edge-creation messages and have its own graph of the network. We then added some code to display the network graph on the laptop’s screen.

We tested this code by first turning on a single node and connecting the laptop to the node. We then turned on two additional nodes, and gave them some time to find each other and establish a connection. We observed this self-connecting behavior as the graph displayed on the laptop **Figure 5**. The full code used for this project can be found on GitHub [4].

To test our network as a communication network, we used the same basic auto-connection functionality from the previous application. We added an LED to one of the

ABOUT THE AUTHORS

Daniel Weber is a Masters of Engineering in Computer Science student at Cornell University. He is interested in programming languages and distributed systems, and enjoys participating in CTF competitions with the Cornell Hacking Club.

Michaelangelo Rodriguez is a graduate of Cornell University’s electrical & computer engineering program. He is interested in working with embedded systems, and enjoys learning more about the practical applications of these embedded systems through projects with the Arduino and Raspberry Pi. In his spare time, he also enjoys film and playing soccer and chess.

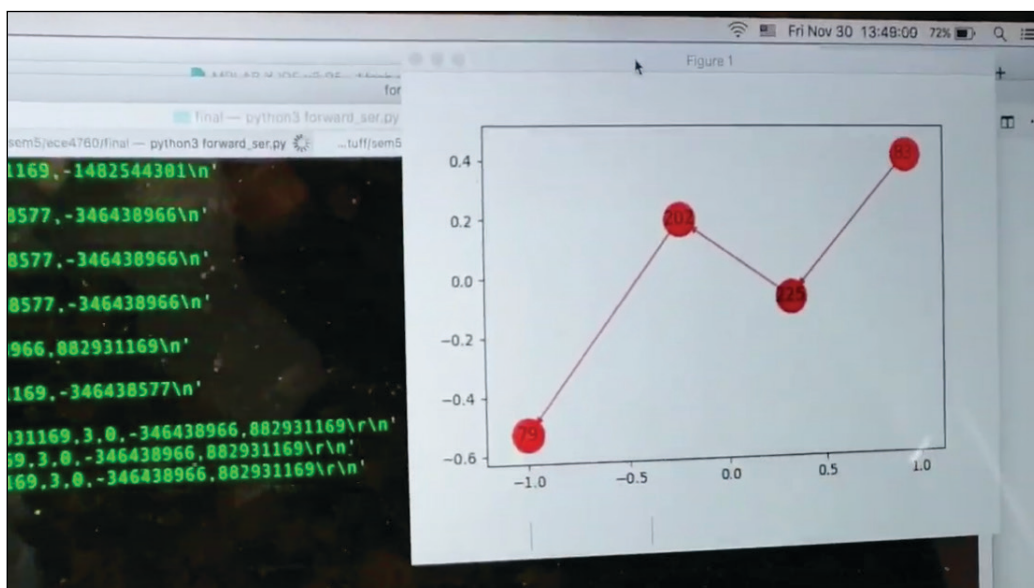


FIGURE 5

A screenshot from the demo video for connecting the three nodes and the laptop. The graph on the screen shows the laptop's current view of the network topology. The video is available on *Circuit Cellar's* article materials webpage.

nodes, along with some code to turn the LED on and off, depending on what message the node received. We then started up all the nodes as before and sent messages from the laptop to the node with the LED attached to it. We set up the network so that the message would have to pass through at least one node before it got to the node with the LED. This would confirm that our message-routing algorithm worked.

We observed that a few seconds after sending the command from the computer, the LED turned on or off. The primary reason for the lag was that we hard-coded a delay of 5 seconds between iterations of the main application loop. This was chiefly for debugging purposes and could have been removed. Removal would have made the delay less noticeable and ideally seem like the LED instantly reacted to the message being sent.

RESULTS


Overall, the hardware performed well. The boards we created had no issues and reliably connected the ESP8266 modules, the PIC32 and the UART-to-USB debugging cable. The main problem we had with the software was that when more than a few modules were present in the network, some modules would often disconnect. This may have been due to the 30 second timeout for the TCP connection and the large 5 second delay that was introduced to aid debugging. Although we would have liked to test our implementation without the 5 second delay, this would have required rewriting parts of our DMA buffer reading code in a non-trivial manner.

Additionally, sometimes the Wi-Fi modules were unable to see the access points of other Wi-Fi modules on their scans, even when they were very close. Furthermore, sometimes the

Wi-Fi modules failed to set up immediately on power up. A hard reset of the Wi-Fi module usually resolved these issues, though we were unable to identify the cause. Nevertheless, we successfully validated the self-organizing property of the mesh network and the ability of the routing layer to route a message from a source node to a destination node.

CONCLUSIONS

We were quite pleased with the outcome of our work. We met most of our initial goals and made some interesting software along the way. One consideration for future work on this project is to improve the functionality of our routing algorithm on a larger scale. We tested our routing algorithm with a relatively small number of nodes. It likely would not scale to a greater number of nodes, because each node needs to know about the existence of every other node. Furthermore, we would have liked to optimize the speed at which the network could propagate messages. However, this would have required rewriting some of the lower-level code and eliminating the 5 second debug delay. Unfortunately, we ran out of time while creating this project. We also had minimal support for handling disconnections and link failures. We had some ideas about how to solve this problem, but didn't get a chance to adequately implement them.

In future work on this project, we would like to implement and test some algorithms for keeping the network graph consistent for all nodes when edges are removed. Finally, we want to test how well our system performs as a long-range communication system, by having the network bootstrap itself into a multi-hop mesh network and try and get two computers at the endpoints to communicate with each other. 

Designing Manufacturing Test Systems

Quality and Cost

Manufacturing tests are vital to ensuring high-quality products. Quality is a factor that no company or individual wants to compromise because quality defines the product and ultimately is the main thing that retains a customer. In this article, Xilinx's Nishant Mittal discusses various techniques to manage quality, cost and "corner case catching" scenarios in the manufacturing test environment of a board fabrication house.

By
Nishant Mittal

Manufacturing tests are arguably the most important aspect in any kind of hardware design company, be it small or big. These tests are essential for ensuring quality. Apart from quality, cost is one of the major factors that are responsible for defining the profit margin of the hardware. For example, if a board is manufactured where let's say out of 1,000 units there are 200 with a defect. Or, let's say that the manufacturing test setup is so costly that it downsizes the profit. Or, here's the important one: What if the manufacturing test misses a defect that a customer finds? That could cost the company a lot.

There are a variety of ways to manage quality and cost. In this article, I'll discuss some these factors and also look at corner case catching scenarios in the context of a manufacturing test environment in a board fabrication house. I will also discuss architecture for crafting manual, semi-automatic and automatic manufacturing tests. For these purposes, in the article, I'll look at these issues as applied to FPGA- and processor-based board, but the same principles apply to less complex boards as well.

The manufacturing test design process runs parallel to the board design process. With

that in mind, the steps involved are similar, but involve more critical judgement. Manufacturing tests have to consider the cost of development, minutes per board to test, corner case reviews and so on. All these factors are necessary to optimize cost without compromising the quality of the product.

The first step toward designing a manufacturing test is to choose one of three approaches: manual, automatic or semiautomatic. This choice depends on the organization's budget, the complexity and quantity of boards as well as the use case. A manual approach has less development time while its test execution time is more per board. In contrast, an automatic approach has more development time, however the test execution time is much less, thereby increasing the productivity. Semi-automated systems are generally in between the two others, and are generally appropriate done for situations where some processes require human intervention.

FPGA EXAMPLE

Let's consider an example of a one-of-a-kind Xilinx Zynq Ultrascale plus FPGA Evaluation board. This board has the FPGA loaded on board with peripherals such as temperature sensors, infrared sensors, power supply, FTDI chip, IO header, SD card and DIP Switch.

In a system like this, we can think of different ways of testing this board. However, when we test a board that is going to thousands of



ABOUT THE AUTHOR

Nishant Mittal is a Systems Engineer at Xilinx in Hyderabad, India.

customers, many things need to be documented such as board test coverage, corner cases, time to test and production test cost. Let's focus on each of these points, and then complete the manufacturing test design.

Board test coverage doesn't necessarily mean the test should cover each and every component on the board. A standard rule in this kind of environment is divide and conquer. A standard board can be divided into its major sections. A board like this contains a power supply—which is given input either through a power jack/USB—an analog region, a digital region, filtering circuits, I/Os and communication blocks.

The first step is to create a block diagram of the system as shown in **Figure 1**. Based on this diagram, we should make a table of coverage showing the number of components, which are actually affected during test and the ones which are not affected. This gives us a fair idea about the percentage coverage and failure scenarios. This not only helps in getting an error-free board out of production, but also creates a "database," which is helpful in future to debug the board when same issue may occur. **Figure 2** shows a format of the table that could be used to create a clean database along these lines.

With the table in Figure 2 in mind, let us consider the design of a typical microcontroller board that contains lot of decoupling capacitors and RC networks which are required for proper decoupling of ground noise in the PCB. In a typical manufacturing test environment, it is very difficult to test the presence or absence of each and every decoupling capacitor, so they generally are considered to be in the "not covered category."

DFMEA

When we say not covered, that doesn't necessarily mean we are ignoring how critical

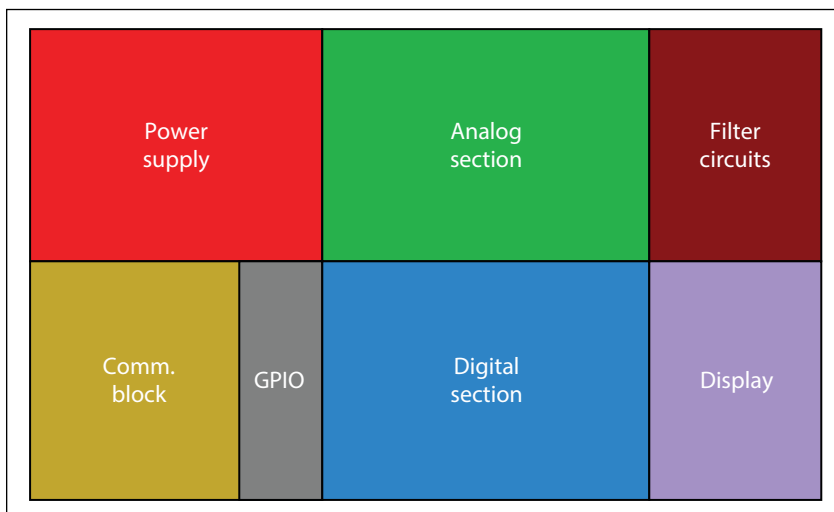


FIGURE 1
Block-based bifurcation of components

the presence or absence of that particular component is. To judge the criticality of failure, coverage goals and actions to be taken. For this, the team needs to perform DFMEA (design failure mode and effect analysis). For DFMEA, an Excel sheet is prepared that looks like the one shown in **Figure 3**. This is a standard format for DFMEA, with a few things here and there that may differ for different organizations.

In this analysis, the design team finds out the potential causes of failure, their impact on the design from a user and board safety point of view and the possible workaround. Based on this, the designer rates all these parameters and the average of all these parameters are then judged to determine that critical test coverages to be made. DFMEA not only makes the manufacturing test foolproof, but also identifies loopholes in the design and even helps you fine tune your design, if done in the early stage. Once the DFMEA is completed, the next step is to design the test system. The type of test system can be

S.No	Ref Des	Part number	Covered/Not covered	Region	Function	Impact
1	C2,C33		Covered	Digital	Filter	
2	C5,C34		Covered	Digital	Filter	
3	C25,C26,C27,C28		Not covered	Digital	Decoupling	Least
4	C30		Not covered	Digital	Decoupling	Least
5	C123		Covered	Comm Block		
6	C124		Covered	Display		
7	C125		Covered	Analog		
8	D2		Covered	Digital		
9	J9,J12,J13,J16,J17		Covered	IO		
12	LED2		Covered	Digital		
13	L1,L2,L3		Covered	Filter		
14	L10		Covered	Analog		
15	R13		Covered	Analog		

FIGURE 2
Format for planning the bill-of-materials (BOM) coverage in the manufacturing test

dependent upon the complexity of the board. A system could be manual test, automated or semi-automated.

Manual Tests: Manual tests are done for very low complexity, lesser volume boards which have fewer interfaces to be tested. A pure manual test involves extensive human

involvement, which can lead to human errors. Proper documentation is the key to a successful manual test system. That said, these tests require lot of time per board, putting both efficiency and cost at stake. Generally, manual tests are preferred when some kind of observation or calibration is required.

Automated Tests: The next method is the one that is mostly preferred throughout the industry: automated test. Automated tests are performed to test the board automatically—without human intervention. This is achieved both in the product's hardware and software.

Figure 4 shows what a typical automated test looks like for hardware. For the board picked as an example, there are metal beads running all around the I/Os, which will perform loopback tests between each other. If any of the I/O presents a short- or an open-circuit, the result is a fail status. For LEDs, we use light sensors on the test systems that detect the light intensity. There are actuators that press the buttons and report the operating status of the buttons. Sensors—such as light sensors, infra-red sensors and so on—can be tested by providing potential stimuli and then the results can be analyzed in the software using the ADC.

Software such as Mathwork's MATLAB, National Instruments' Labview, python and pearl scripts can be used to create UI-based interfaces to that display pass and fail status. The UI is basically used to monitor what's happening and to trigger the tests. Once the test is completed, the UI is supposed to report all the data in the log file, which may be exported to a pdf file.

Semi-Automated Tests: The next category of test systems is essentially the combination of manual and automated tests: semi-automated systems. Semi-automated systems are used in cases where human intervention becomes necessary. Human intervention doesn't have to mean triggering tests, putting the board into the proper location or even sitting in front of the system to monitor the events going on. Rather, it applies to whenever there's any human intervention impacting the result of the particular test—then it becomes a semi-automated system.

Let's look at an example of a board that has a microphone, a capacitive touch sensor and the rest of the interfaces I mentioned earlier. The tester is supposed to test the mic sensitivity by feeding it sound from different directions and at different volumes. Meanwhile, a capacitive touch sensor needs to be touched by a human hand to see if it's sensitive to human touch. These tests can be automated, but for optimum performance it has to have some human intervention. These types of use cases could force a designer to

Design FMEA Analysis									
Item and Function	Potential Failure Mode	Potential Effects of Failure	∇ S E V	Potential Cause(s) of Failure	O C C	Detection Method & Quality Controls	D E T	R P N	Recommended Actions
List Part Name, Number and Function	List the possible modes of failure	List the consequences of failure on part function and on the next higher assembly		List those such as: inadequate design, improper materials, etc.		List these measures available to detect failures before they reach the customer			List them for each of the failure modes identified as being significant by the RPN

= Critical characteristic which may effect safety, compliance with Gov. regulations, or require special controls.
 SEV = Severity rating (1 to 10)
 OCC = Occurrence frequency (1 to 10)
 DET = Detection Rating (1 to 10)
 RPN = Risk Priority Number (1 to 1000)

FIGURE 3

DFMEA format (Image courtesy of Superfactory)

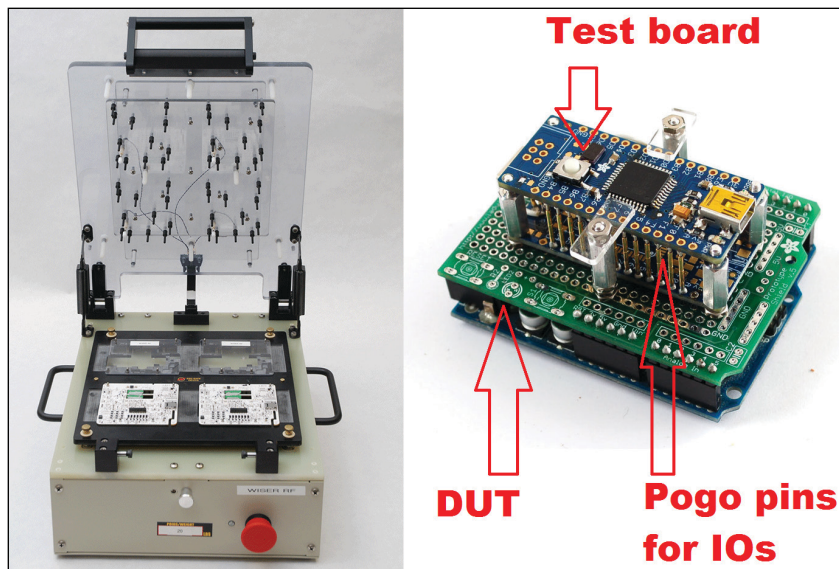


FIGURE 4

Complete automated manufacturing test rig (left); A tear down of a DUT and a test board (right), (Image left courtesy Brioconcept.com; image right courtesy Adafruit).

For detailed article references and additional resources go to:
www.circuitcellar.com/article-materials

RESOURCES

Mathworks | www.mathworks.com


Xilinx | www.xilinx.com

make the system semi-automated.

Once the test design is complete, the designer needs to validate whether the coverage really matches what's theoretically stated. To validate this, the tester would remove the major components on board to test and see if the manufacturing test it really is doing its job correctly. **Figure 5** shows the algorithm that displays the entire design flow.

There are other techniques such as JTAG scan chain. This uses system controllers on board that can equally perform the board testing as well as control the interfaces—either by themselves or along with traditional test techniques. It's open for debate whether the amount of cost reduction, visibility of the board and test coverage that such controllers can provide compared with the traditional approach of external test systems. I'll plan to discuss that question further in future articles.

CONCLUSION

In this article, we discussed the concepts of how manufacturing tests are developed and analyzed in order to cater to the requirements of cost, efficiency and accuracy. We also discussed how the test system designer would decide whether the board should be tested using a manual, automated or semi-automated approach. 

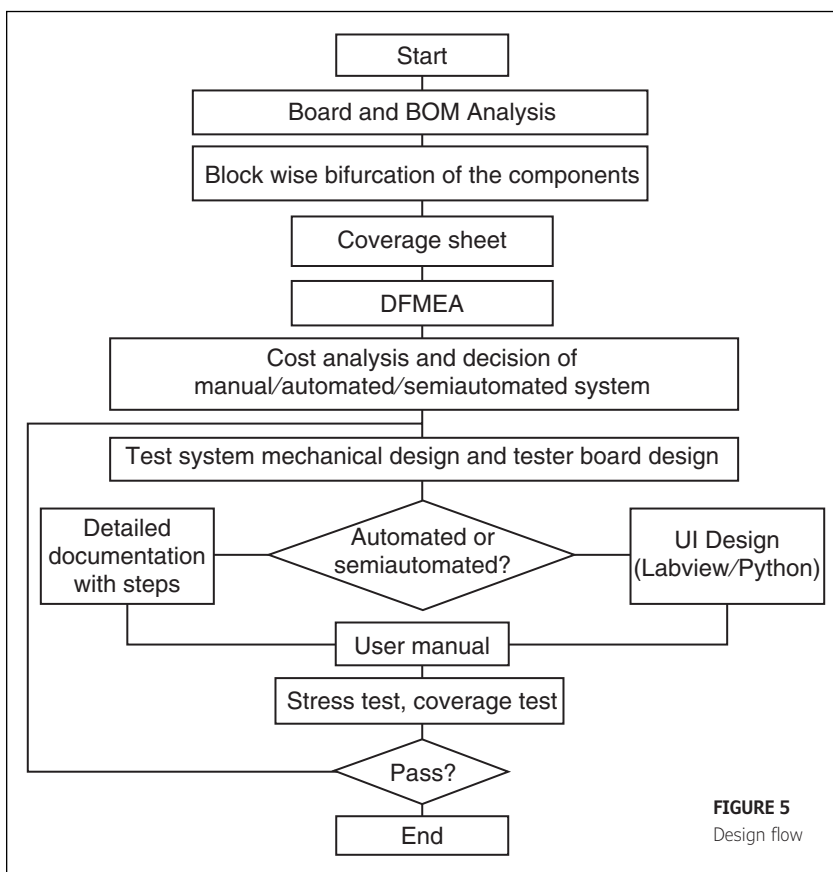


FIGURE 5
Design flow

FEATURES

Why choose one wireless protocol when you can have them all?



SimpleLink™ connectivity for **smart buildings**

TI.com/SimpleLink



Multi-Key Electronic Flute

Sensors and Synthesis

FEATURES

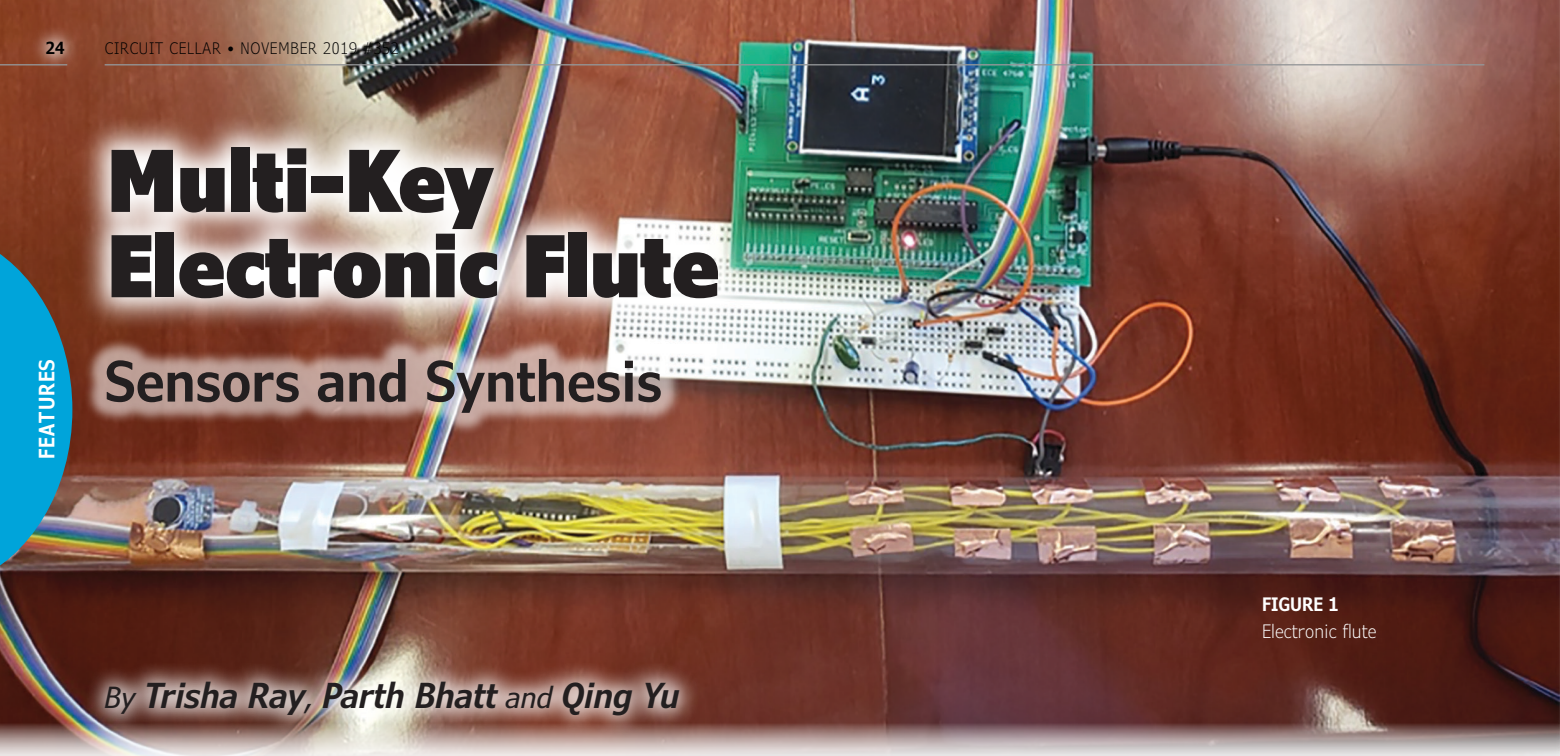


FIGURE 1
Electronic flute

By *Trisha Ray, Parth Bhatt and Qing Yu*

Musical instruments such as the piano allow musicians to play in different keys on a single instrument. In contrast, bamboo flutes are designed for only one key. This means flute players must own a different flute for every additional key in which they want to play in. Learn how these three Cornell students built an PIC32 MCU-based electronic flute that reduces the need for owning multiple flutes by incorporating two buttons that allow a flute player to change the key and octave.

Our goal for this project was to build an electronic flute that can play in any key. The first step in this project was understanding the design of a bamboo flute, which differs greatly from Western concert flutes. A typical bamboo flute can be played in only one key. It has a total of seven holes, six of which are used to play different notes. The seventh hole is for the inlet of wind (the player's breath). The strength of air blown into it determines the octave of the note.

If the strength of the air blown exceeds a certain threshold, the flute produces sound in a higher octave. Otherwise, the flute produces sound in a lower octave. The arrangement of the player's fingers over the six holes distinguishes the different notes. Whether a hole is open, half-covered or fully covered by a finger also differentiates the note being played. For example, if a fully covered hole generates a note C major key, the half-covered hole generates a note in C minor key.

PHYSICAL/HARDWARE DESIGN

Our electronic flute, shown in **Figure 1**, is built to be comparable in size, design and spectral dynamics to a typical bamboo flute. We simulated the six finger holes of a typical flute using capacitive touch sensors. A seventh hole holds the microphone and simulates a flute's blow hole.

Physically, these switches are pieces of copper tape connected to wires. We used a total of 13 capacitive touch sensors—two for each hole and one for the "chin sensor." The chin sensor determines when someone is playing the flute. It is positioned directly under the microphone hole, and needs to be touched when playing. The microphone detects if air is blown into the flute, indicating that a sound should be produced. Note that the words button, switch and sensor used throughout this article functionally refer the same general mechanism.

At the heart of our electronic flute is a PIC32 microcontroller (MCU) from Microchip Technology, which reads the inputs from the copper tape buttons and microphone to produce the correct notes and sound. The sound is outputted to a speaker after going through a digital-to-analog converter (DAC).

A detailed breakdown of the hardware components of our electronic flute is shown in the block diagram in **Figure 2**. The 13 copper-tape touch sensors, the microphone, the key control switch and the octave control switch are inputs into the PIC32 MCU. The outputs from the PIC32, after running direct digital synthesis, are sent through the DAC to the amplified speaker to produce the flute sounds. Two buttons are used to control the key and octave of the flute, which get displayed on the TFT Display.

One unique feature on the PIC32 is the Charge Time Measurement Unit (CTMU). Our first step when designing our electronic flute was determining how to use the CTMU to create the capacitive touch sensors. The CTMU peripheral is available for use on all the ADC pins of the MCU. It is essentially a settable current source that can measure resistance, capacitance and more. To understand how we used a settable current source to measure capacitance, recall that the formula for capacitance is $C = q/V$, where C is the capacitance, V is the voltage across the capacitor and q is the charge. Charge can also be denoted as the product of the current (I) and time (t). Hence, the formula can be rewritten as $C = (I \times t) / V$. Using the CTMU, if the pin is provided with a fixed current for a fixed amount of time, we get C as inversely proportional to V . This logic is used to measure the voltage on an ADC pin, which changes when the pin is touched and released. This is how CTMU can be used to create touch sensors.

TOUCH SENSORS

The PIC32 does not provide enough ADC channels for the 13 touch sensors in our design, so we chose to connect the touch sensors to two analog multiplexers (**Figure 2**). The multiplexers enable us to connect the touch sensors to only one ADC channel and five other GPIO pins, which saves plenty of pins on the PIC32. We used the CD4051xB analog 8x1 multiplexer from Texas Instruments. With this multiplexer, if the select lines are 000, for example, then the input connects to the output 0. And if the select lines are 111, then the input connects to the output 7. Its chip-select line has the ability to turn off the entire chip, so that none of the outputs are connected to the input. This feature was useful to our project, since the outputs from the two different multiplexers are connected to one ADC pin, meaning that one chip is always off.

The breath-detecting microphone we used is the Electret Microphone Amplifier

from Adafruit. In our circuit, the microphone is connected to a peak detector circuit to obtain the absolute value of the signal. The absolute value of the signal is needed to get a proper ADC reading. When we tested this microphone, the ADC readings obtained from its circuit were either in the range of the 500s (when no air was blown into it) or 900s (when air was blown into it). We later realized this observation was probably due to a calibration issue, since the ADC readings should increase linearly with the amount of air blown into the microphone. Because of the binary behavior of the microphone that we first observed, we used the microphone only to control the octave being played by the flute.

The uniqueness of our electronic flute design comes from the two buttons (switches) (SW1 and SW2 in **Figure 3**), which can be used to adjust the key and octave of the flute. When the user presses switch SW1, the key goes up—for instance from B to C. When the user presses button SW2, the octave goes up by number—for instance from C3 to C4. Both buttons can be used to circle back to the lowest key and octave. Then, the TFT display in our design lets users see the current key and octave being played.

The last main hardware component in our circuit is the DAC. As the name suggests, it converts the digital signal of the sound generated by the MCU into an analog signal, so it can be fed into the amplified speaker. The schematic in Figure 3 shows all the hardware components in our design. The schematic for the PIC32 development board was first created by Sean Carroll. A link to more information about Sean Carroll's development board can be found on *Circuit Cellar's* article materials webpage. The 13 touch sensors are indicated by the 13 circles on the left side of the schematic.

SOFTWARE DESIGN

The entire software design was coded in C and consists of four different threads and an interrupt service routine (ISR). The ISR uses additive synthesis and direct digital

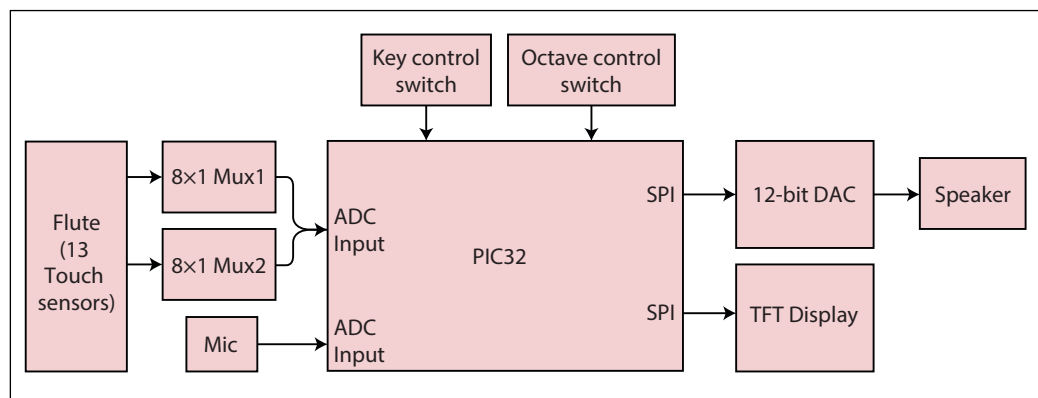


FIGURE 2
Block diagram of the electronic flute

synthesis to output the sound to the DAC. The Display Thread displays the current key and octave that the flute is playing. The ADC/CTMU Thread runs CTMU and reads the ADC value of the switches. The Frequency Thread determines what note to play. Last, the Debouncing Thread debounces the two buttons that change the octave and key of the flute. The Display Thread and the Debouncing Thread are straightforward to implement, so they will not be discussed in detail here.

ADC/CTMU Thread: The main purpose of the ADC/CTMU Thread is to read the ADC values of the 13 capacitance touch sensors, to determine which sensors are being pressed. As noted in the previous section, we used two multiplexers connected to the ADC channel AN11. The chip-select line of the multiplexers ensures that only one multiplexer is turned on at a time. A threshold of about 90% of the full ADC value is used to determine if a finger is touching any of the 12 buttons. The thread starts off by setting the ADC channel to AN11. Next, the CTMU is turned on, and a for-loop measures the voltage on each of the capacitive touch sensors, using CTMU. In this for-loop, the thread also determines which

multiplexer to turn on using chip-select. The first eight values in the for-loop correspond to the first multiplexer, and the rest correspond to the second multiplexer.

For the CTMU to work correctly, the following sequence of events must occur in the ADC/CTMU Thread:

- 1) The internal discharge switch is closed to drain the external circuit of any charge, by connecting the ADC channel to ground.
- 2) The internal discharge switch is opened, and the internal charge switch is closed for 2 μ s, to allow charge to build up.
- 3) During the charging period, the interrupts corresponding to the ISR are turned off to ensure that the program was not interrupted while charging. Since the interrupts are only turned off for 2 μ s, the direct digital synthesis (DDS) that takes place in the ISR isn't affected.
- 4) After the 2 μ s, the internal charge switch is opened and the ADC value is read.

This sequence of events is placed inside a for-loop in our code so that CTMU can run for all 13 capacitive touch sensors.

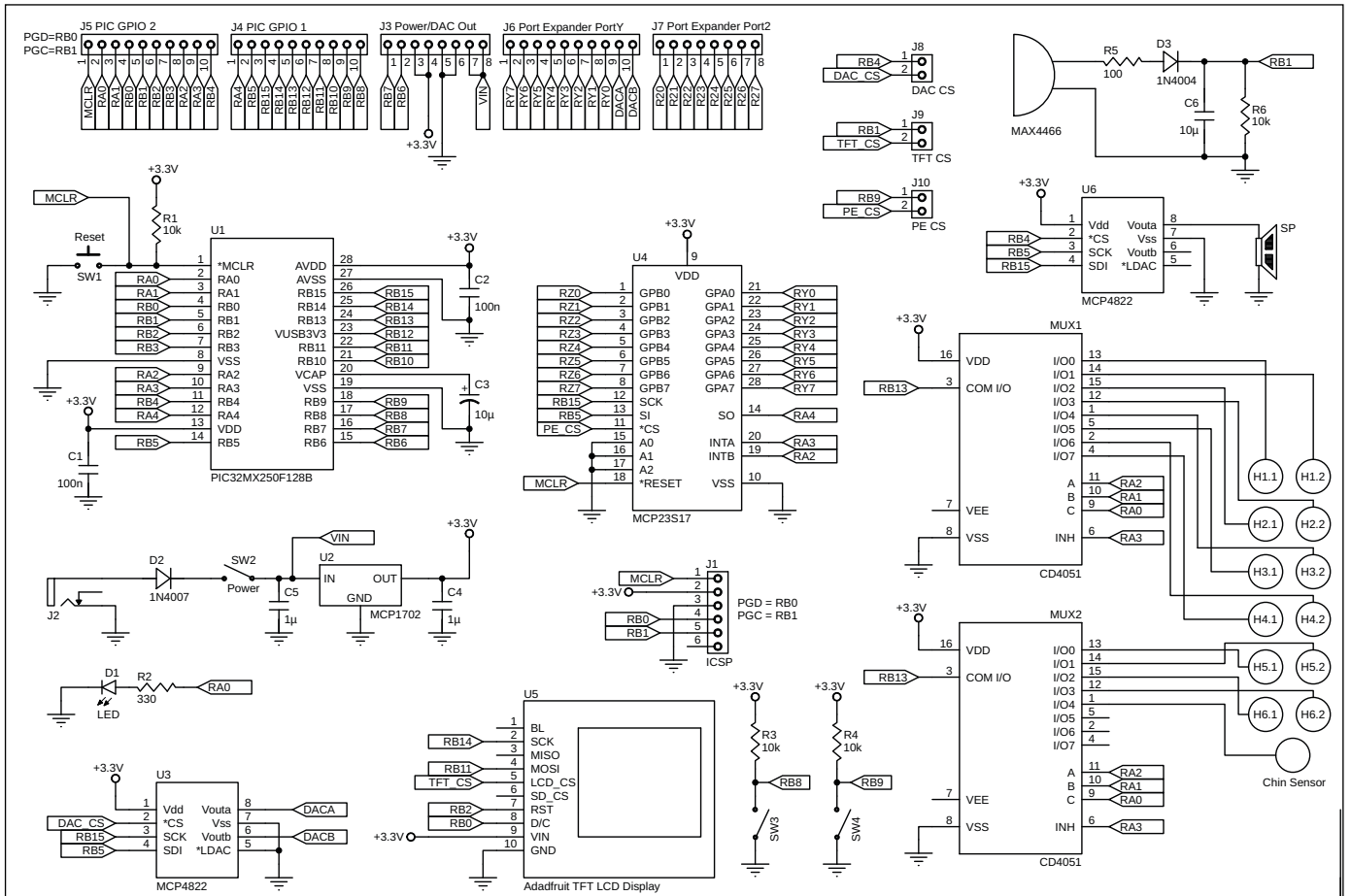


FIGURE 3
Schematic of two switches and TFT display

There's one more important aspect of the ADC/CTMU Thread. Outside the for-loop that runs CTMU, the thread sets the ADC channel to AN3, enabling the ADC value of the microphone to be read. The corresponding CTMU circuit is shown in **Figure 4**, where S1 is the internal charge switch and S2 is the internal discharge switch.

Frequency Thread: One of the main functions of the ISR is to run direct digital synthesis. We used DDS to produce sound waves from the PIC32. DDS works by creating a sine table of one sample frequency that contains the signal's amplitude values at evenly spaced phase values. Then, by moving through the sine table at different rates, different frequencies can be produced. Because every note has a different frequency, the following equation shows how to generate different notes using one DDS sample frequency:

$$(\text{inc}) \frac{F_s}{2^{32}} = F_{\text{out}}$$

In this equation, F_{out} is the frequency trying to be produced, and F_s is the sample frequency. Manipulating this equation, we can solve for the phase increment value (inc), which determines the rate of

movement through the sine table. In our ISR, every time an interrupt occurs, a phase accumulator variable is incremented by the phase increment value. The top byte of the phase accumulator variable is then used as the reference index for the sine table matrix, such that incrementing through one sine table using different-sized increments produces different frequencies.

The following method was used to determine how the Frequency Thread works and how we selected which note to play. Outside the Frequency Thread, each

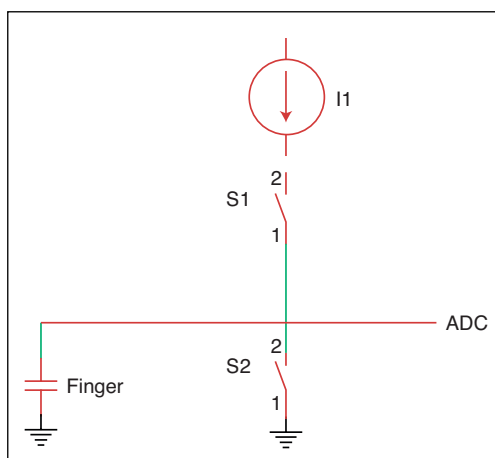


FIGURE 4
Charge Time Measurement Unit (CTMU) circuit

When it comes to robotics, the future is now!

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

With this book, you'll learn about:

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

ADVANCED CONTROL ROBOTICS
HANNO SANDER

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

Get it today, cc-webshop.com

Sales@ezpcb.com

Ez PCB

Welcome
www.ezpcb.com

One-Stop PCB & PCBA Turnkey Service

PCB Online Calculator
No Need to Register
Instant Quote & Pay

1 To 40 Layers
Prototype to Mass Production
Amateur to Professional

Prototype Start At \$5/PC
2L 4"x4" each
Free Shipping

In the past years, our PCB have been shipped to 40 countries

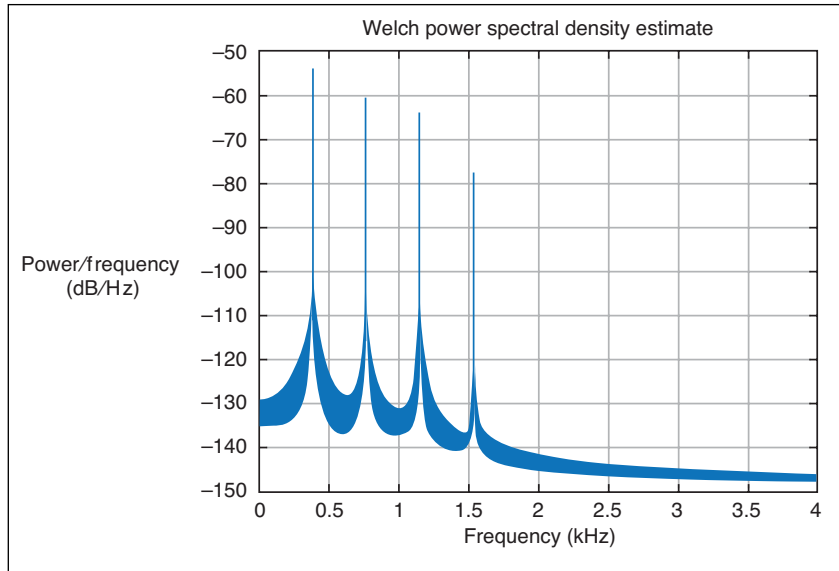


FIGURE 5
MATLAB flute spectrum

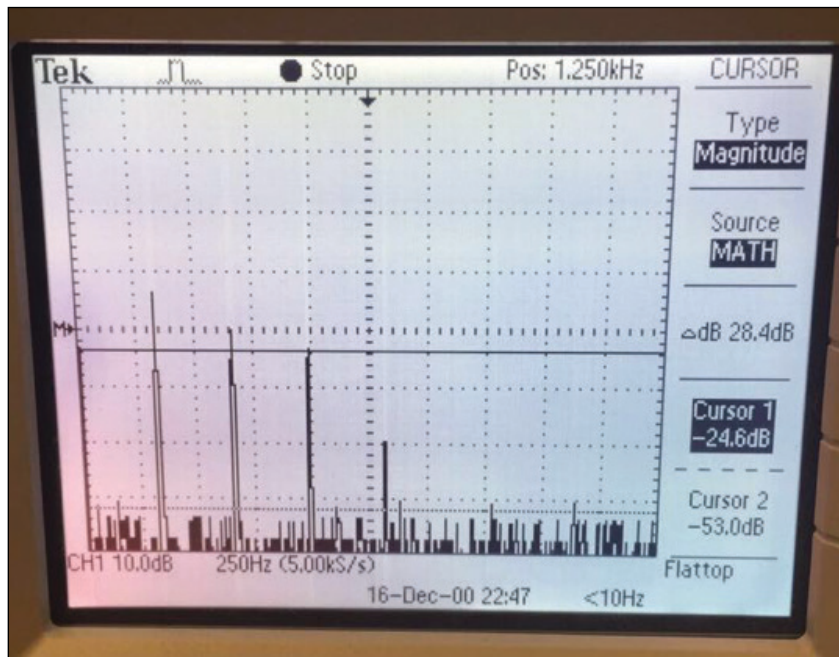


FIGURE 6
Sound spectrum from our flute playing in real time

frequency value from A2 to G4# is defined. A 12×7 matrix holds 12 different keys, from A to G#. Inside the Frequency Thread, the program first checks if the chin sensor (indicating whether someone is playing the flute) is being touched. This equates to an if-statement that checks if the 13th bit of the button integer is set to 1. The rest of the thread consists of a switch-statement that determines if we play the lower or higher octave of Do, Re, Mi, Fa, So, La, Ti, Do in any key. An exact combination of capacitive touch sensors must be pressed to play a certain note, such as Do. The exact combination is represented by 12 bits, since each simulated "hole" in the flute comprises two capacitive touch sensors. This combination is identical to the combination of holes that need to be covered on a bamboo flute.

SOUND SYNTHESIS

The most difficult part of the software design was sound synthesis. This was due to the fine tuning required for additive synthesis to create a flute-like sound. Additive synthesis is a technique that sums sine waves to mimic the natural sound spectrum of an instrument. Our main challenge with additive synthesis was determining how many harmonics to use, and how to adjust the amplitudes of those harmonics to create the most realistic sound. Another challenge was experimenting with FM modulation to create a vibrato effect.

The first step in our process was understanding the spectrum of harmonics that a flute makes. As a starting point, we used a sound spectrum of a flute, published by the University of New South Wales, which showed that a flute has a series of peaks at f , $2f$, $3f$, $4f$ and so on. The amplitudes of the peaks for $2f$, $3f$, and $4f$ are approximately -15 dB, -5 dB and -20 dB, respectively, from the fundamental f .

We first implemented the sound synthesis as stated above in Mathwork's MATLAB. We started with a fundamental and four harmonics, and noticed that the sound didn't improve much more when we added more harmonics to our additive synthesis. Moving the other way, the sound also didn't change much with three harmonics, but did sound less flute-like with two harmonics. So, we decided that a fundamental and three harmonics was optimal. We then ran a variety of tests changing the amplitude of the harmonics. **Figure 5**, shows one test that was close to the flute sound we wanted. The ratio of our amplitudes changed slightly from our initial test, to create what we considered a more flute-like sound.

After fine tuning the amplitudes of our three harmonics, we still were not satisfied

For detailed article references and additional resources go to:
www.circuitcellar.com/article-materials

RESOURCES

Adafruit | [Adafruit](http://Adafruit.com) | www.adafruit.com

Mathworks | www.mathworks.com

Microchip Technology | www.microchip.com

Texas Instruments | www.ti.com


with the sound being generated. For this reason, we used amplitude modulation (AM) sound synthesis to create a vibrato effect. AM sound synthesis works by multiplying a wave function with a very small frequency to the main wave function. Therefore, we multiplied a cosine function with a frequency of 1 Hz to the sum of the four harmonics as shown by the following equation:

$$f(t) = \cos(2\pi t) \times \sum_{n=1}^4 (a_n \cos(2n\omega_o \pi t))$$

After we finished testing in MATLAB, we implemented our additive synthesis into the ISR and PlaySound method. The PlaySound method first determines the correct note to play by adjusting the phase increment value for the fundamental frequency. The increment values for the three harmonics are 2, 3 and 4 times greater, since the harmonics are at 2f, 3f and 4f from the fundamental frequency, f. The phase increment for the frequency modulation (FM) wave never changes, because the frequency of the wave is always 1 Hz. The ISR then uses the constantly updating increment values from the PlaySound method to implement additive synthesis to produce the final sound wave. **Figure 6** shows the output of the sound spectrum produced by playing our flute after implementing this code.

RESULTS

Overall, we are satisfied with the performance of our electronic flute. With a total cost of \$38.64, this project was an inexpensive way to explore MCU design and sound synthesis. One of our authors, Parth, is a flute player, and he believes the sound is realistic. We encourage readers to listen to Parth play the Theme from Titanic on our electronic flute. You can watch and listen to this on the YouTube video of our project, which is posted on *Circuit Cellar's* article materials webpage. You can judge for yourself how our electronic flute sounds compared to a real flute!

We learned some lessons while working on this project. First, with additional time to improve the design of our electronic flute, we would enhance the flute's volume dynamics and sound quality. We didn't use the microphone's ADC readings to control the volume of the flute's sound. If implemented correctly, the ADC readings of the microphone should increase linearly with how much air is blown into it. We could use this relationship to linearly increase the flute's sound volume. When testing our sound synthesis design, we also tried to base our design on the Wind Instruments Synthesis Toolbox. Due to the complexity of the toolbox and our time constraints, we did not implement its algorithm. If we had more time, we believe this toolbox would have helped us create an even more realistic sound. 

ABOUT THE AUTHORS

Trisha Ray graduated from Cornell University with a Bachelors in Electrical and Computer Engineering in May 2019. Trisha is currently a Master's student in Electrical Engineering at the University of Washington. She hopes to pursue a career in the power industry. Outside of class, Trisha likes to sing and is learning how to play the piano and ukulele.

Parth Bhatt graduated from Cornell University with a Masters in Electrical and Computer Engineering in May 2019. Parth is currently working as a Firmware Engineer at Schlumberger. In his free time, he likes to read and play the flute. This is what motivated the project of a multi-scale flute.

Qing Yu is pursuing a Master degree in Electrical and Computer Engineering at Cornell University. She will graduate in December 2019. Her academic focus is on embedded control systems, firmware development and firmware validation. Her interests in music come from playing the violin.

LCR-Reader

Digital Multimeter



LCR-Reader-MPA

Ultimate PCB debugging tool

L-C-R, AC/DC Voltage/Current
LED/Diode/Continuity Test
Oscilloscope
Frequency, Period, Duty Cycle
Signal Generator
Super Cap Testing

Basic Accuracy 0.1%
Test Frequency: 100 Hz to 100 kHz
Test Signal Level: 0.1, 0.5, 1.0 Vrms

All-in-One
Digital Multimeter

SIBORG
SYSTEMS INC

LCR-Reader.com

Chip Solutions Tackle the Energy Harvesting Challenge

Self-Sufficiency at the IoT Edge

SPECIAL FEATURE

While many IoT edge devices often need to be extremely low power, having an ability to harvest their own power is an even better scenario. Chip solutions continue to emerge aimed at the energy harvesting challenge.

By *Jeff Child*,
Editor-in-Chief

Forecasts predict that there's likely to be a trillion IoT sensor nodes deployed in the world by 2025. Powering those devices is going to be a challenge because many of those will be low power modules residing in remote areas. Energy harvesting will be critical in those applications because it just won't be practical to replace trillions of batteries that only last a year or two.

To help you meet that challenge, there's a variety of chip and development platform solutions available that attack various parts of the energy harvesting puzzle. These include specialized microcontrollers (MCUs), power management chips, power regulator ICs as well as complete platform solutions and reference designs—all aimed at energy harvesting.

The three most popular types of energy harvesting are solar, piezoelectric (vibration/rotation) and thermoelectric. Of the three, solar is the most widely used today and it relies on photovoltaic cells to provide energy. It's the best fit for typical smart home, smart agriculture, smart industrial and similar applications. Piezoelectric energy harvesting leverages vibration/rotation types of energy, and is practical if you're monitoring motors, generators or turbines—anything that moves

or vibrates. Finally, thermoelectric energy harvesting is great for systems involving pipes—such as gas pipes or water pipes—where one side is hot and one side is cold, and energy can be harvested from heat transfer.

No matter what the power source, a module powered by energy harvesting relies on either a harvesting power supply or an alternative set of external components that converts input from a solar, piezoelectric or thermoelectric source into some voltage range and current. Some devices even accept multiple types of power source interfaces. But the key issue is that the system has to be efficient enough to be viable for the situation.

LOCAL EDGE PROCESSING

For its part, Eta Compute's approach to the energy harvesting challenge is to provide high performance local edge processing at low power levels. The idea is that it's the RF communication portion of an IoT edge device that's the most power hungry. If you can limit the amount of data communication needed, then you can more easily achieve a solution that can run off an energy harvesting power source.

To illustrate the point, Eta Compute's Chet Jewan cites an example of a low power image detection module designed to detect cars in your

driveway. A module without much intelligence would only be able to send a full image or video data of the driveway, consuming a lot of power by the RF transmitter. But with sophisticated local processing, the module could decide whether to send an image of a car, or just a simple text saying whether a car is present or not. Moreover, the local processing could even decide to take some local action itself, requiring no RF data transmission at all. "A high level of local processing can sense, infer and act locally," says Jewan.

Eta Compute's flagship product is its ECM3531 ASIC chip for machine learning algorithms based on the Arm Cortex-M3 and NXP Coolflux DSP processors. The SoC includes an analog-to-digital converter (ADC) sensor interface and highly efficient PMIC circuits. The chip also includes I²C, I²S, GPIOs, RTC, PWM, POR, BOD, SRAM and flash.

The ECM3551 has a dual-core architecture based on the M3 and Coolflux DSP that is designed for low power edge AI IoT applications (Figure 1). The device makes use of the company's patented delay insensitive asynchronous logic (DIAL), which enables dynamic voltage frequency scaling and near threshold voltage operation. The MCU uses an Arm Cortex-M3 processor and operates below 1 MHz to over 100 MHz with power consumption as low as 4.5 μ A/MHz. By using asynchronous processing of all digital logic, the architecture enables rapid interrupt response for low latency applications.

SOTB TECHNOLOGY

Renesas Electronics has approached the challenge of meeting extreme low power demands by applying innovations in semiconductor process development. A year ago, the company unveiled an innovative energy-harvesting embedded controller that can eliminate the need to use or replace batteries in a device. The R7F0E embedded controller—Renesas' first commercial product using SOTB (silicon on thin buried oxide) technology—is a 32-bit, Arm Cortex-based embedded controller. The device is capable of operating up to 64 MHz for rapid local processing of sensor data and execution of complex analysis and control functions. The R7F0E consumes just 20 μ A/MHz active current, and only 150 nA deep standby current, approximately one-tenth that of conventional low-power MCUs.

The extreme low current levels of the SOTB-based embedded controller enables

system designers to completely eliminate the need for batteries in some of their products through harvesting ambient energy sources such as light, vibration and flow (Figure 2). Although the solution was developed with IoT devices in mind, the controller is more broadly aimed at what they call the new market of maintenance-free, connected IoT sensing devices with endpoint intelligence. This includes health and fitness apparel, shoes, wearables, smart watches and drones.

In June of this year, Renesas followed up with the development of new low-power technology for use in embedded flash memory based on a 65 nm SOTB process. Available with 1.5 MB capacity, it is the first embedded 2T-MONOS (2 transistors-metal oxide nitride oxide silicon) flash memory based on 65 nm SOTB technology.

BLE SENSOR PLATFORM

As mentioned earlier, solar power is the most popular form energy harvesting used today. There are a growing number of IoT sensor applications where the duty

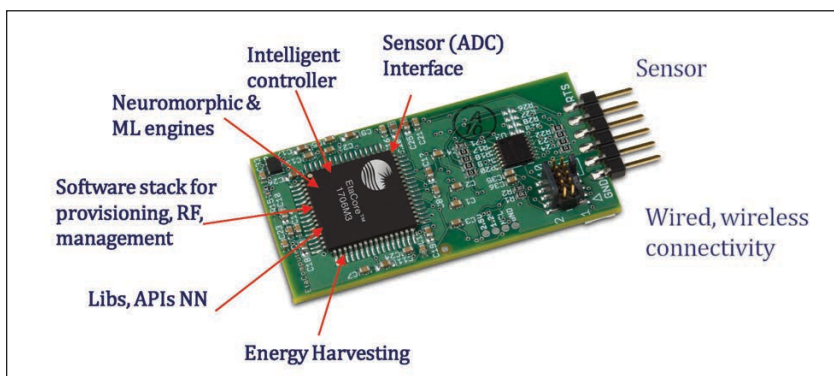


FIGURE 1

The ECM3551 chip has a dual-core architecture based on the M3 and Coolflux DSP that is designed for low power edge AI IoT applications. The device makes use of Eta Compute's patented delay insensitive asynchronous logic (DIAL), which enables dynamic voltage frequency scaling and near threshold voltage operation.

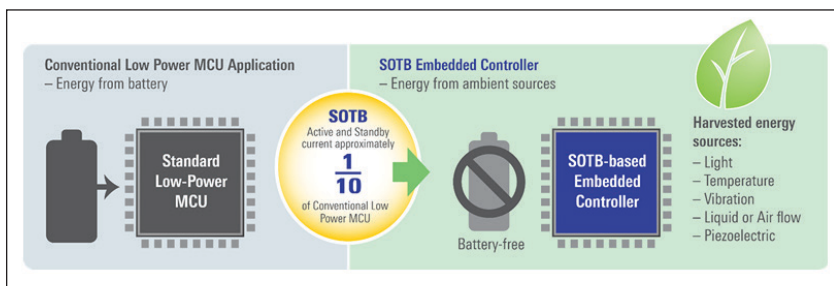
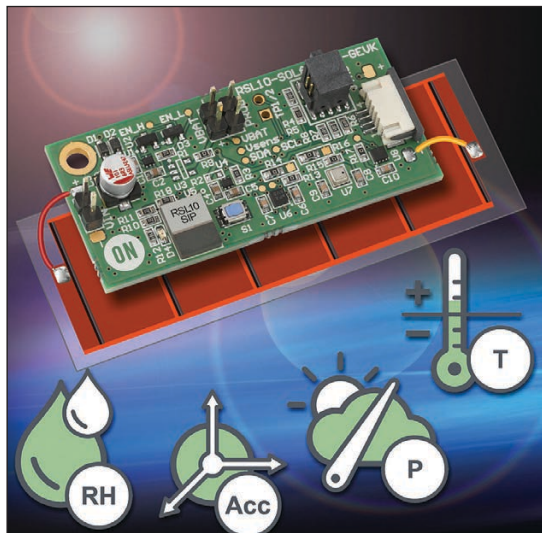


FIGURE 2

The extreme low current levels of the SOTB-based embedded controller enables system designers to completely eliminate the need for batteries in some of their products through harvesting ambient energy sources such as light, vibration and flow.

FIGURE 3

The RSL10 Solar Cell Multi-Sensor Platform includes the RSL10 SIP, a solar cell and a host of low power sensors from Bosch Sensortec, including the BME280 all-in-one environmental sensor (pressure, temperature, humidity) and the BMA400 ultra-low-power 3-axis accelerometer.



cycle is low enough to support intermittent communications, allowing the energy needed to support operation to be harvested using renewable sources such as solar. Applications are expected to include smart home and building automation such as HVAC control, window/door sensors and air quality monitoring. Asset tracking including package open/close detection, shock monitoring, and temperature and humidity data logging are also possible applications.

Offering a complete platform solution, in May, ON Semiconductor introduced its RSL10 Multi-Sensor Platform powered only with a solar cell. This complete solution supports the development of IoT sensors using continuous solar energy harvesting to gather and communicate data through Bluetooth Low Energy (BLE), without the need for batteries or other forms of non-renewable energy.

The combination of ultra-low-power wireless communications, small form-factor solar cell and low duty cycle sensing applications makes it possible to develop

and deploy totally maintenance-free IoT sensor nodes. The RSL10 Solar Cell Multi-Sensor Platform is enabled by the RSL10 SIP, a complete System-in-Package (SiP) solution featuring the RSL10 radio, integrated antenna and all passive components.

The platform combines the RSL10 SIP with a solar cell and a host of low power sensors from Bosch Sensortec, including the BME280 all-in-one environmental sensor (pressure, temperature, humidity) and the BMA400 ultra-low-power 3-axis accelerometer (**Figure 3**). Together, they enable developers and manufacturers to create complete IoT nodes that are entirely powered through renewable energy or energy harvested from the sensor's surroundings. For easy development, the platform is supplied with all design files (Gerber, schematic and BoM) and customizable source code as part of a CMSIS software package.

HIGH-EFFICIENCY BATTERY CHARGER

Energy efficiency can make or break an energy harvesting implementation. Offering a battery charging solution, STMicroelectronics provides its SPV1050 chip, an ultralow power and high-efficiency energy harvester and battery charger, which implements the MPPT (maximum power point tracking) function and integrates the switching elements of a buck-boost converter. MPPT is a common function used in solar electric charge controllers.

The SPV1050 device allows the charge of any battery, including the thin film batteries, by tightly monitoring the end-of-charge and the minimum battery voltage in order to avoid the over-discharge and to preserve the battery life (**Figure 4**). The power manager is suitable for both PV cells and TEG harvesting sources, because it covers the input voltage range from 75 mV up to 18 V and guarantees high efficiency in both buck-boost and boost configurations.

Meanwhile, the SPV1050 device boasts very high flexibility thanks also to the trimming capability of the end-of-charge and undervoltage protection voltages. That enables any source and battery to be matched. The MPPT is programmable by a resistor input divider and allows maximizing the source power under any temperature and irradiance condition.

An unregulated voltage output is available (for example, to supply an MCU), while two fully independent LDOs are embedded for powering sensors and RF transceivers. Both

Ultra low power energy harvester and battery charger

**FIGURE 4**

The SPV1050 device allows the charge of any battery, including the thin film batteries, by tightly monitoring the end-of-charge and the minimum battery voltage in order to avoid the over-discharge and to preserve the battery life. The power manager is suitable for both PV cells and TEG harvesting sources.

LDOs (1.8 V and 3.3 V) can be independently enabled through two dedicated pins.

THERMAL-BASED PMIC

Among the latest energy harvesting solutions from E-peas is its latest power management IC (PMIC) announced in February. The device is specifically optimized for energy harvesting from thermal sources in wireless sensor applications. Supplied in a space-saving 28-pin QFN package, the AEM20940 is a highly advanced device based on proprietary technology that is capable of extracting available input current up to levels of 110 mA.

Taking DC power from a connected thermal electric generator (TEG), it can supervise the storing of energy in a rechargeable element and simultaneously supply energy to the system via 2 different regulated voltages. This is done through its built-in low noise, high stability 1.2/1.8 V and 2.5/3.3 V LDO voltage regulators. The lower voltage can be employed for driving the system MCU, while the higher voltage is intended for the RF transceiver.

Through the AEM20940's deployment, it will be possible to extend the system battery life or, in many cases, eliminate the primary power source from the system completely. In this way any dependence on having to regularly replace batteries (which often has serious logistical challenges associated with it, as well as adding to the overall expense) can be removed.

In more recent news from E-peas, in April, the company confirmed that its AEM10941 devices for photovoltaic energy harvesting are being incorporated into



FIGURE 5

A key design requirement of the m0Ovement smart tracker project was that the size and weight of the unit had to be kept as low as possible, in order to minimize the impact on the animal. This placed severe restrictions on the surface of the solar panel that could be accommodated.

SPECIAL FEATURE

Make your code even faster, smaller, and smarter while ensuring robustness and high quality.

TOTAL TOOLS

Our customers build the technology for a new world.
We supply the tools to make it happen.
One toolbox, one view, one uninterrupted workflow.
As simple as that.



IAR Embedded Workbench



www.iar.com

Learn for free at
circuitcellar.com/iar



FIGURE 6

The TIDA-00242 reference design supports MPPT to provide optimal energy extraction from solar panels. It also has internal battery charging and protection circuits. It makes use of the buck and boost capabilities of the bq25570 chip.

tracking equipment employed in Australian cattle ranches. E-peas engineers worked in conjunction with the team at Dutch systems integrator SODAQ on the development and implementation of energy efficient livestock monitoring hardware for Brisbane-based client mOOvement.

Through use of mOOvement's smart tracker, valuable data on cattle herds can be acquired concerning their position and grazing patterns, with the ability to set alarms if individual animals are not moving or fenced boundaries have been breached (**Figure 5**). Attached to one of the cattle's ears, each tracker comprises an accelerometer, a LoRa communication module (with built-in MCU) and a GPS transceiver, as well as a passive NFC tag.

A key design requirement of the project was that the size and weight of the unit had to be kept as low as possible, in order to minimize the impact on the animal. This placed severe restrictions on the surface of the solar panel that could be accommodated (with it measuring slightly less than 19 mm x 43 mm in total and capable of generating

0.125 W). Consequently, the power system needed to be ultra-efficient.

COLD START-UP PMU

IoT devices relying on energy harvesting in low energy conditions often have to slowly accumulate enough energy to turn on, resulting in long delays before the device can start sensing, processing and transmitting. This can result in missed data collection, slow operation and poor user experience. With that in mind, Analog Devices provides its ADP509x power management unit (PMU) that's designed to solve these problems with a multiple-power-path design, which enables faster startups and smoother operation.

ADI says that a key barrier for energy harvesting is that in many applications energy from the environment is only available at very low levels (for example, low-light indoor solar harvesting), and periodically not at all. This requires power management solutions that can not only enable satisfactory system operation with very little energy, but also efficiently manage energy storage devices to satisfy energy demand at times when no energy is being harvested.

Due to its unique circuit design, ADI claims the ADP509x as among the most efficient energy harvesting PMUs on the market, converting harvested power down to the 16 μ W to 100 mW range with only sub- μ W operation losses. The ADP509x also delivers the fastest cold-startup time available, according to ADI.

BOOST/BUCK CONVERTER

Among the solutions for energy harvesting from Texas Instruments (TI) is its bq25570 chip, a nano power boost charger and buck converter for energy harvester powered applications.

The bq25570 device is specifically designed to efficiently extract microwatts (μ W) to milliwatts (mW) of power generated from a variety of high output impedance DC sources like photovoltaic (solar) or thermal electric generators (TEG) without collapsing those sources.

The battery management features ensure that a rechargeable battery is not overcharged by this extracted power, with voltage boosted, or depleted beyond safe limits by a system load. In addition to the highly efficient boosting charger, the bq25570 integrates a highly efficient, nano- power buck converter for providing a second power rail to systems such as wireless sensor networks (WSN),

For detailed article references and additional resources go to:

www.circuitcellar.com/article-materials

RESOURCES

Analog Devices | www.analog.com

E-peas | www.e-peas.com

Eta Compute | www.etacompute.com

Renesas Electronics America | www.renesas.com

ON Semiconductor | www.onsemi.com

STMicroelectronics | www.st.com

Texas Instruments | www.ti.com

Wiliot | www.wiliot.com

which have stringent power and operational demands. All the capabilities of bq25570 are packed into a small foot-print 20-lead 3.5 mm x 3.5-mm QFN package (RGR).

TI also offers a reference design based on the bq25570. The TIDA-00242 reference design is a solar charger and energy harvester, using a highly integrated power management solution that is well-suited for ultra-low power applications (**Figure 6**). The product is specifically designed to efficiently acquire and manage the microwatts to milliwatts needed to power your design. The storage method is a 47 nF super capacitor that is charged and maintained by 4 series low power solar elements using MPPT.

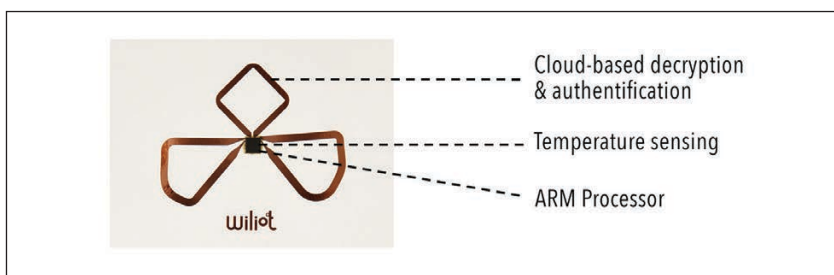
The TIDA-00242 reference design supports MPPT to provide optimal energy extraction from solar panels. It also has internal battery charging and protection circuits. It makes use of the buck and boost capabilities of the bq25570. Input voltage regulation prevents collapsing high impedance input sources (boost). And support is provided for programmable step-down regulated output (buck). Energy is stored in a super capacitor, for use in low power applications. The reference design is a complete solution, including the solar current source, charge management solution, super cap and a built-in LDO regulator.

RF ENERGY HARVESTING

While one viewpoint is that RF communication is major power problem for energy harvesting applications, start-up Wiliot takes an entirely different approach. Wiliot's technology seeks to harvest energy from the RF transmissions themselves. According to the company, there are two approaches to harvesting RF energy: RF scavenging and intentional RF energy transfer. The first mode of operation taps into existing sources of energy from devices being used without the intention of generating energy, the energy available over-the-air is intermittent and unpredictable. The resulting applications this mode can enable are stochastic in nature.

Wiliot says that in the latter approach, the source of energy is deterministic in terms of power levels and time, with a specific duty cycle pattern delivered from an infrastructure planned to provide it. As such the resulting energy output is also more predictable, and the transmission of packets from radio powered are transmitted at a predictable cadence.

Wiliot uses RF harvesting techniques to power



its chip, consisting of a Bluetooth radio, the Arm Cortex M0+ core, a set of sensors and a security element (**Figure 7**). It can work in both modes of operations, though the one it's designed for is the first. When considering the increase in the background interaction of products and consumer devices that are battery-powered like smartphones, the prospect of harvesting power without the need for infrastructure is attractive.

TECHNOLOGY RELEASE PLAN

In August, Wiliot announced an update on its release plan for its technology. So far this year, Wiliot has designed and built 5 prototype chips. With each version, the company has increased robustness, and also added encryption, multiple on-chip sensing capabilities, and harvesting from three radio bands simultaneously. Its most recent milestone was the completion of the first production chip design, a "release candidate," which should power the Version 1.0 Wiliot tag and move them from making small batches of product to volume production.

The rest of this year will be focused on taking this release candidate chip from wafer, through processing, testing, configuration, all the way to conversion into the final tag form factor, ready for the first field tests next year. By the end of 2019, Wiliot expects to have a good sense of the performance of the release candidate. In Q2 2020, it plans to roll out some of the existing Early Advantage Program projects it has been working on this year. During 2020, the company will continue a controlled release of that product.


Clearly, the stakes are high for future development of energy harvesting technology. As designers of IoT edge modules strive for lower power operation, energy harvesting solutions expand the conditions in which that can operate. The battery-free advantages of energy harvesting will open up new areas of IoT implementations that would otherwise not be practical. Chips developers will continue to address that challenge with a variety of energy harvesting solutions. 

FIGURE 7

Wiliot uses RF harvesting techniques to power its chip, consisting of a Bluetooth radio, the Arm Cortex M0+ core, a set of sensors and a security element.

Analog ICs Feed Needs of Industrial Systems

Advances for Automation

Industrial automation and process control applications rely heavily on a variety of analog ICs to ensure smooth, reliable system operations. Chip vendors are responding with new solutions across the spectrum of analog ICs, including amplifiers, data converters, motor drivers and more.



TECH SPOTLIGHT

By **Jeff Child**,
Editor-in-Chief

FIGURE 1

The ADuM4122 is a simple dual-drive strength output driver that efficiently toggles between two slew rates controlled by a digital signal. The device can control how fast or slow a MOSFET or IGBT turns on or off by user command, on the fly, thereby controlling motor currents.

As factories migrate to ever more automated and “smart” operations, system developers are hungry for new advances in a variety of analog IC product areas. Advances span everything from data converters to comparators to motor drivers. These devices must meet the particular performance levels for industrial designs while meeting the harsh environmental demands of the factory floor.

To keep pace with the needs of industrial system developers, over the past 12 months analog ICs vendors have continued to roll out new chips designed to meet a variety of industrial design needs, including factory robotics, instrumentation systems and control automation systems. Products include both ICs specifically designed for the industrial market and those for which industrial is one among a range of other applications targeted.

ENERGY-EFFICIENT DRIVER

Exemplifying these trends, in September, Analog Devices (ADI) announced the ADuM4122, an isolated, dual-drive strength output driver that uses iCoupler technology. It’s designed to empower designers to harness the benefits of higher efficiency power switch technologies. Electric motor-driven systems account for 40% of global electricity

consumption, according to the International Energy Agency, and improvements in motor efficiency can have wide-reaching economic and environmental benefits.

With the increased adoption of industrial automation and IoT within smart factories, there is a growing demand for intelligent technology and features within systems to ensure maximum efficiency, says ADI. The ADuM4122 is claimed to be the first simple solution that accomplishes this by controlling how fast or slow a MOSFET or IGBT turns on or off by user command, on the fly, thereby controlling motor currents (**Figure 1**).

The new ADuM4122 is a simple dual-drive strength output driver that efficiently toggles between two slew rates controlled by a digital signal. Smaller than existing discrete or complex integrated solutions that have 20 or more pins, the ADuM4122 features only eight pins and works in a variety of operating conditions. The ADuM4122 further improves system capabilities with high common-mode transient immunity and low propagation delay for high performance applications such as motion control, robotics and energy.

HIGH-ACCURACY ADCs

Analog-to-digital converter (ADC) technology continues to be critical in industrial applications, particularly for precision

instrumentation system designs. With that in mind, in June, Microchip Technology rolled out a new family of compact ADCs that offer high programmable data rates of up to 153.6 Ksps. According to the company, the 24-bit MCP356x and 16-bit MCP346x delta-sigma ADC families offer faster programmable data rates than similar devices on the market while providing high accuracy and lower noise performance (**Figure 2**). Available in a tiny 3 mm x 3 mm UQFN-20 package, these integrated ADCs are designed to meet the increasing demand for small packaging in space-constrained applications such as portable instrumentation devices.

Most high-resolution delta-sigma ADCs on the market have slower programmable data rates of a few Ksps, says Microchip. The MCP356x and MCP346x families offer a much faster data rate, making the devices ideal for a variety of precision applications that require different data speeds, including industrial process control, factory automation and sensor transducers and transmitters. The ADCs also offer integrated features to eliminate the need for external components and reduce the overall cost of a system, including an internal oscillator, temperature sensor and burnout current source.

The new families provide 24-bit or 16-bit resolution, two/four/eight single-ended or one/two/four differential channel options, allowing developers to choose the most suitable ADC for their designs. For development tools, Microchip provides the MCP3564 ADC evaluation kit (ADM00583). The kit includes a MCP3564 ADC Evaluation Board for PIC32 MCUs (ADM00583), a PIC32MX795F512L PIM (processor plug-in module) and a USB cable.

AMPLIFIERS AND COMPARATORS

Current-sense amplifiers and comparators are among the list of analog ICs important to many industrial electronic systems. Addressing those needs, in June, Texas Instruments (TI) introduced what it claims is the industry's smallest current-sense amplifier in a leaded package and the smallest, most accurate comparators with an internal 1.2-V or 0.2-V reference. Offered in industry-leading package options, the INA185 current-sense amplifier, and open-drain TLV4021 and push-pull TLV4041 comparators enable engineers to design smaller, simpler and more integrated systems while maintaining high performance. In addition, pairing the amplifier with one of the comparators produces the smallest, highest performing overcurrent detection solution in the industry, says TI. **Figure 3** shows the INA185 in a typical circuit.

These new devices are optimized for a variety of industrial and communications

applications and well as personal electronics. With a small-outline transistor (SOT)-563 package measuring 1.6 mm by 1.6 mm (2.5 mm²), the amplifier is 40% smaller than the closest competitive leaded packages. Featuring a 55- μ V input offset that enables higher precision measurements at low currents, the INA185 enables the use of lower-value shunt resistors to cut system power consumption. Additionally, its 350 kHz bandwidth and 2-V/ μ S slew rate enable phase-current reproduction to enhance motor efficiency and save system power.

The precisely matched resistive gain network in the amplifier enables a maximum gain error as low as 0.2%, which contributes to robust performance over temperature and process variations. The device's typical response time of 2 μ s enables fast fault detection to prevent system damage. System designers can add functionality in the same form factor and enable high-performance design with the TLV4021 and TLV4041 comparators. Available in an ultra-small die-size ball-grid array (DSBGA) 0.73 mm by 0.73 mm package, the comparators' integrated voltage reference saves board space while supporting precise voltage monitoring, which optimizes system performance.

The comparators can monitor voltages as low as the 0.2-V internal reference, and feature a high threshold accuracy of 1% across a full temperature range from -40°C to +125°C. Low 2.5- μ A quiescent current

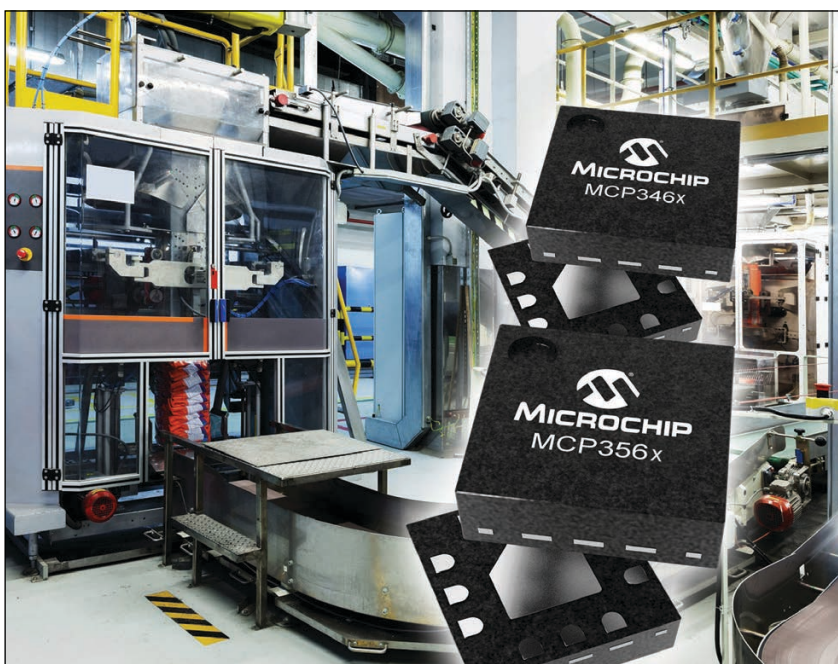


FIGURE 2

The 24-bit MCP356x and 16-bit MCP346x delta-sigma ADC families offer high programmable data rates of up to 153.6 Ksps. Available in a tiny 3 mm x 3 mm UQFN-20 package, these integrated ADCs are designed to meet the increasing demand for small packaging in space-constrained applications.

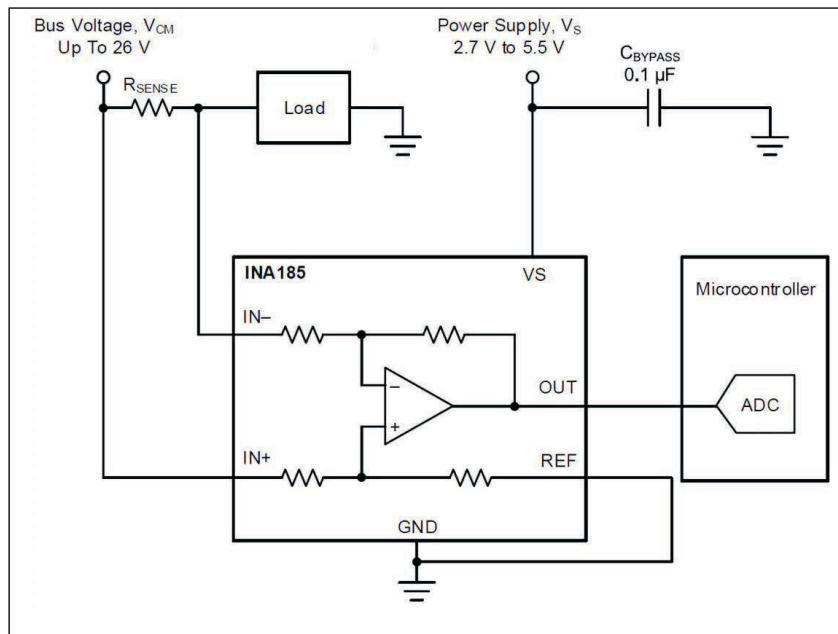


FIGURE 3

Shown here in a typical circuit, the INA185 current-sense amplifier features a 55- μV input offset that enables higher precision measurements at low currents. The INA185 enables the use of lower-value shunt resistors to cut system power consumption. Its 350-kHz bandwidth and 2-V/ μs slew rate enable phase-current reproduction to enhance motor efficiency and save system power.

delivers extended battery life for smart, connected devices. Fast propagation delay as low as 450 ns reduces latency, enabling power-conscious systems to monitor signals and respond quickly to fault conditions.

When using both the INA185 and the TLV4021 or TLV4041, engineers can shrink their total footprint to enable smaller systems. In combination, these devices produce the smallest, highest-performing overcurrent detection solution—15% smaller and 50 times faster than competitive devices, says TI. Pairing the amplifier with one of the comparators to support overcurrent detection on rails as high as 26 V delivers more headroom to better manage current spikes.

Production quantities of the INA185 are now available through the TI store and authorized distributors in a SOT-563 package, measuring 1.6 mm by 1.6 mm. Production quantities of the push-pull TLV4041 and preproduction samples of the open-drain TLV4021 comparators are now available through the TI store and authorized distributors in an ultra-small DSBGA package, measuring 0.73 mm by 0.73 mm.

SMART SHUT DOWN

The ability to reliably shut down factory equipment in industrial applications is important for safety as well as ensuring

smooth operations. Along those lines, in July, STMicroelectronics (ST) announced the STDRIVE601, a 3-phase gate driver for 600 V N-channel power MOSFETs and IGBTs. It was designed to provide state-of-the-art ruggedness against negative voltage spikes down to -100 V and responds to logic inputs in a class-leading 85 ns.

Featuring smart-shutdown circuitry for fast-acting protection, the STDRIVE601 turns off the gate-driver outputs immediately after detecting overload or short-circuit, for a period determined using an external capacitor and resistor. Designers can set the required duration, using large C-R values if needed, without affecting the shutdown reaction time. An active-low fault indicator pin is provided. The STDRIVE601 replaces three half-bridge drivers to ease PCB layout and optimize the performance of 3-phase motor drives for equipment such as home appliances, industrial sewing machines and industrial drives and fans.

All outputs can sink 350 mA and source 200 mA, with gate-driving voltage range of 9 V to 20 V, for driving N-channel power MOSFETs or IGBTs. Matched delays between the low-side and high-side sections eliminate cycle distortion and allow high-frequency operation, while interlocking and deadtime insertion are featured to prevent cross conduction.

Fabricated in ST's BCD6S offline process, the STDRIVE601 operates from a logic supply voltage up to 21 V and high-side bootstrap voltage up to 600 V. Bootstrap diodes are integrated, saving the bill of materials, and under-voltage lockout (UVLO) on each of the low-side and high-side driving sections prevents the power switches operating in low-efficiency or dangerous conditions. An evaluation board, EVALSTDRIVE601, is available to help users explore the features of the STDRIVE601 and quickly get first prototypes up and running.

INDUSTRIAL PHOTOCOUPLEDERS

While photocouplers are used in a variety of applications, they must meet special requirements to be used in the harsh environment of a factory setting. Offering a solution, in July, Renesas Electronics announced three new 15 Mbps photocouplers designed to withstand the harsh operating environments of industrial and factory automation equipment. The trend toward higher voltage, compact systems is driving stricter international safety standards and eco-friendly solutions that require smaller ICs with lower power consumption. The RV1S9x60A family meets this need with low threshold input current (IFHL) ratings:

RESOURCES

Analog Devices | www.analog.com

Microchip Technology | www.microchip.com

Renesas Electronics | www.renesas.com

STMicroelectronics | www.st.com

Texas Instruments | www.ti.com

the RV1S9160A (SO5) operates at 2.0 mA, the RV1S9060A (LSO5) at 2.2 mA, and the RV1S9960A (LSDIP8) at 3.8 mA (**Figure 4**).

Lower power consumption allows the RV1S9x60A photocouplers to meaningfully suppress power supply heat generation. And high temperature operation up to 125°C enables board space savings by mounting the photocoupler near the IGBT or MOSFET power device. The devices are targeted at DC to AC power inverters, AC servo motors, programmable logic controllers (PLCs), robotic arms, solar and wind input power conditioners, and battery management systems for energy storage and charging.

The RV1S9x60A photocouplers feature high common mode rejection (noise tolerance) up to 50 kV/ μ s (min) to protect MCUs and other I/O logic circuits from high voltage spikes while transferring high-speed signals. The RV1S9x60A family also offers a variety of packages with the smallest footprint for each reinforced isolation (up to 690 V_{RMS}), and minimum creepage distances of 4.2 mm to 14.5 mm to ensure safe operation.

The RV1S9160A, RV1S9060A and RV1S9960A photocouplers provide low voltage power supply operation of 2.7 V to 5.5 V. Isolation voltages for the devices are as follows: 3750 V_{RMS} (RV1S9160A), 5000 V_{RMS} (RV1S9060A) and 7500 V_{RMS} (RV1S9960A). The devices operate in high temperatures from -40°C to +125°C (RV1S9160A and RV1S9060A), and from -40°C to +110°C (RV1S9960A). Supply current of 2.0 mA maximum, while pulse width distortion at is a low 20 ns maximum. Propagation delay for the devices is of 60 ns max with propagation delay skew of 25 ns max.

POWER FACTOR CONTROLLER

For industrial equipment to operate efficiently, system designers need power-factor control suited today's digital power system configurations. With that in mind, in August, STMicroelectronics announced the STNRGPF12, a dual-channel interleaved boost-PFC controller that aims to blend the flexibility of digital power with the responsiveness of analog algorithms. The device can be easily configured and optimized using the ST's eDesignSuite software. Suited to applications over 600 W, the STNRGPF12 enhances efficiency and reliability in equipment as diverse as industrial motor controls, charging stations, uninterruptable power supplies, 4G and 5G base stations, welding machines, telecom switches, home appliances and data-center power supplies.

The STNRGPF12 operates in continuous-conduction mode (CCM) at fixed frequency with average-current-mode control (**Figure 5**). The best of both digital and analog worlds



FIGURE 4

These three 15 Mbps photocouplers are designed to withstand the harsh operating environments of industrial and factory automation equipment. The RV1S9x60A family offers low threshold input current (IFHL) ratings: the RV1S9160A (SO5) operates at 2.0 mA, the RV1S9060A (LSO5) at 2.2 mA, and the RV1S9960A (LSDIP8) at 3.8 mA.

meets in the STNRGPF12's inner and outer control loops. The inner current loop utilizes a hardware analog Proportional-Integral (PI) compensator, while the outer voltage loop is performed by a digital PI controller with fast dynamic response. This enables the STNRGPF12 to manage cascaded control of the voltage and current loops to regulate the output voltage by acting on the total average inductor current.


Integrated features include digital inrush-current limiting, which leverages silicon-controlled rectifiers (SCR) in the high-side switching circuitry to facilitate soft-start management and enhance system robustness. The STNRGPF12 also supports load feed-forward, current balancing, phase shedding, and fan control. An integrated UART allows access to non-volatile memory for user configuration of PFC parameters to meet specific application needs and permits monitoring of parameters in the field. In support of the STNRGPF12, ST provides an extensive ecosystem that includes the STEVAL-IPFC12V1 dual-channel 2 kW interleaved PFC reference design, as well as the configuration software. 



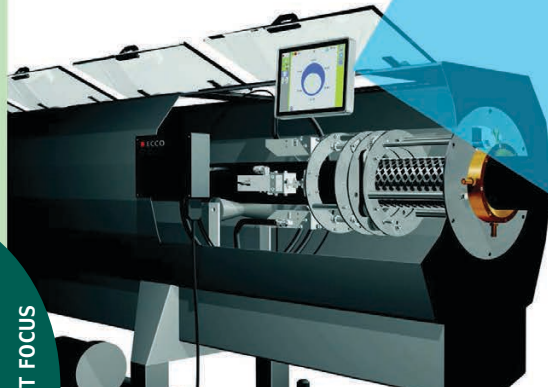
FIGURE 5

The STNRGPF12 is a dual-channel interleaved boost-PFC controller designed to blend the flexibility of digital power with the responsiveness of analog algorithms. Suited to applications over 600 W, the STNRGPF12 enhances efficiency and reliability in equipment such as industrial motor controls.

Product Focus: Panel PCs

HMI Intelligence

By *Jeff Child,*
Editor-in-Chief



IP66

FIGURE 1

iNOEX used a Panel PC for an ultrasonic measurement that they developed for pipe centering to achieve optimum wall thickness. They needed a Panel PC to serve as the systems' visualization client and HMI. Because of the environment the machine is used in, they required a 10" Panel PC with a robust, fanless and completely-sealed IP66-rated design.

Panel PCs are a category of display systems that are meant to be mounted on a factory wall or on the side of an industrial machine. They're also well suited for transportation systems like railway user interfaces. Rather than simply being a display, panel PCs embed complete single board computing functionality, providing a comprehensive embedded solution.

By providing a complete, all-in-one, embedded computing and HMI (human-machine interface), Panel PCs enable any embedded system to display information and enable user control. Most modern Panel PCs are touchscreen displays unless they are basically digital signage displays. In this article, the product album of representative Panel PCs shows a wide variety of products that vary in size and feature sets. Panel PCs embed full blown SBCs, often using the latest and greatest embedded microprocessors, including Intel Kaby Lake, AMD Embedded G-Series, NXP i.MX8M and others. Because Panel PCs tend to be installed in long-life cycle applications, using embedded processors makes sense.

Industrial systems such as factory automation and process control are among the most common uses for Panel PCs. And because industrial applications come in all types, today's crop of Panel PCs are available for a variety of environmental conditions, for example extreme heat environments, high-hygiene environments and more benign applications like retail systems. Panel PCs are also attractive for transportation applications like railway systems. As the product album shows, some Panel PCs are even designed with railway needs in mind. In recent years,

wireless communication support has become more universal in Panel PCs, providing Wi-Fi or Bluetooth technology for remote applications where cabling connections aren't practical.

Illustrating an example application, iNOEX needed a Panel PC for an ultrasonic measurement that they developed for pipe centering to achieve optimum pipe wall thickness. With this new measurement technology, the die head used to produce large-sized and thick-walled pipes could be quickly centered, reducing start-up scrap and saving considerable costs. Engineers at iNOEX needed a Panel PC to serve as the systems' visualization client and HMI. Because of the environment the machine is used in, they required a 10" Panel PC with a robust, fanless and completely-sealed IP66-rated design (**Figure 1**).

iNOEX chose the Kontron's Intel Atom processor-based Kontron Nano Client panel PC with stainless steel housing. At the time, the Atom processor was the optimum choice for a low power system that could be fully enclosed and cooled without the use of fans. The unit offers all required interfaces for HMI or terminal applications. Essential for pipe extruding equipment was its Compact Flash port for data storage. Unlike a traditional rotating hard drive, Compact Flash features no moving parts and excellent shock and vibration protection.



Rugged Touch Panel PC for Railway Systems

The DMI-1210 from ADLINK Technology is a 12.1" Driver Machine Interface (DMI) panel PC designed specifically for the railway industry, equipped with Intel Atom x5-E3930 processor (formerly Apollo Lake), resistive touch and MVB interface. It can be applied as an HMI unit for driver's desks, control panel for passenger information systems, surveillance system control/display unit or in railway diagnostics and communications applications.

- Intel Atom x5-E3930 processor, up to 1.8 GHz
- 12.1" color display: 4:3, 1024x768 pixels, 600 cd/m², 5-wire resistive touch
- Isolated 2x M12 GbE, 2x DB-9 serial and 1x M8 USB 1.1 ports
- MVB/CAN bus support by PC/104 or Mini PCI Express add-on module
- Built in GNSS and two Mini PCI Express card slots for cellular modem with USIM
- Nominal Voltage: 24 VDC, 36 VDC, 72 VDC and 110 VDC (EN50155 compliant)
- IP65 front and IP20 rear ingress ratings

ADLINK Technology
www.adlinktech.com



Industrial Panel PC Supports Wide Temp Range

Advantech's TPC-71W is a compact touch panel computer equipped with an NXP i.MX 6 Cortex-A9 dual/quad-core processor, 7" TFT LCD display, multi-touch glass sensor, 2 GB of DDR3L RAM and 8 GB of eMMC storage. TPC-71W supports a wide operating temperature range (-20°C to 60°C / -4°F to 140°F) and VESA mounting and features an IP66-rated front panel that protects against water and dust ingress.

- 7" 16:9 WSVGA LCD with multi-touch P-CAP control and true-flat IP66-rated front panel
- Up to 2 GB DDR3L RAM and 8 GB eMMC storage onboard
- 10/100/1000 Mbps LAN with IEEE 802.3at PoE-PD support
- 1 MB FRAM backup memory for unexpected power interruptions
- Serial port with 120 Ω termination resistor that supports the CAN 2.0B
- Embedded browser and VNC tool for rapid Web App development
- Compatible with VESA and panel mounting

Advantech
www.advantech.com



10.4" Fanless Panel PC Features Stainless Steel Design

The GOT810-845 from Axiomtek is a 10.4" stainless steel fanless touch panel computer. Its full IP66 and IP69K-rated enclosure and IP66-rated M12-type connectors are designed for harsh industrial and outdoor environments. The unit is powered by the Intel Celeron processor N3060 (codename: Braswell). It has a 10.4" XGA TFT LCD display with 350 nits of brightness.

- 10.4" XGA TFT LCD flat bezel projected capacitive (or resistive) touch
- IP66/IP69K-rated (NEMA 4X) water/dust/corrosion-proof design
- Full stainless-steel enclosure with type 316
- -10°C to +50°C wide operating temperature range
- Fanless system with Intel Celeron N3060
- Optional Wi-Fi/3G kit for wireless network connectivity
- 9 to 36 VDC wide range power input
- M12 type I/O connectors for harsh environments

Axiomtek
www.axiomtek.com

Panel PCs



10.1" WUXGA Panel PC Sports i.MX8M Processor

Estone Technology's PPC-4310 is an all-in-one industrial PoE (power over Ethernet) touch Panel PC with WUXGA 1920x1200 touch screen. The system equips with NXP's i.MX8M Quad Core Arm processor with a guaranteed ten year lifespan. Edge to edge glass front panel meets IP65 rating. The Gbit Ethernet PoE option makes it perfect for industrial control, building automation, HMI and more.

- NXP i.MX8M application processor with long life cycle support
- IEEE 802.3af PoE Gbit Ethernet port, second PCIe GbE option
- 10.1" IPS LCD panel with projected capacitive touch screen
- On-board Wi-Fi/BT, RS-232/485 and GPIO ports
- Smart codec with dual-core DSP for digital MICs and voice control
- Camera, digital MICs and light sensor options
- Support Android 8.1, 9.0, Yocto Embedded Linux, Ubuntu

Estone Technology
www.estonetech.com



21.5" Kaby Lake-Based Panel PCs for Smart Retail

The UPC-7210 from Ibase is a fanless 21.5" panel PC is created for smart retail applications. It features a full flat bezel design and a 1920x1080 IPS LCD with projected capacitive touch screen that allows multi touch and gesture touch functionalities. The unit has high reliability and sealed housing, with IP65 rated front panel for waterproof resistance and are powered by 7th Gen Intel Core Processors.

- 21.5" IPS LCD, 1920x1080
- Projected capacitive touch screen, supports multi touch
- IP65 front-panel waterproof protection
- Supports a variety of processor platforms, from performance to entry level
- Optional wireless solution

Ibase
www.ibase.com.tw



Light Industrial Interactive Panel PC has PoE

IEI Integration's AFL3-W10A-AL is panel PC based on Intel's Celeron J3455 (quad core, 1.5 GHz up to 2.3 GHz) processor. The unit supports PoE which allows electronic devices to receive both power and data through one Ethernet plug-in. The ability to receive both power and data through one cord means less cabling and less cost for the factory automation.

- 10.1" light industrial interactive panel PC
- 9 V to 30 V wide range DC input with lockable DC jack
- Selectable AT/ATX power mode
- Built-in speakers
- Support PoE PD IEEE803.2 af/at/bt
- IP64/IP65 compliant front panel
- Touch screen with anti-UV / anti-glare coating

IEI Integration
www.ieiworld.com



Panel PC with IP69K is Made for High Hygiene Systems

Designed according to EHEDG guidelines, the FlatClient HYG from Kontron features a maximum protection class with IP69K. These specifications qualify the new FlatClient HYG for use in sensitive hygienic scenarios, in the food and pharmaceutical industry as well as in clean rooms of semiconductor manufacturing plants, in optics and laser technology, life sciences and nanotechnology. The robust basic design enables operation next to dirt-generating manufacturing machines.

- Designed for high hygiene standards following EHEDG guidelines
- Stainless steel housing with IP69K protection (support arm version)
- Suited for washdown applications
- Scalable performance from Intel Atom to Intel Core i5
- Smooth, seamless display surface for perfect cleanability
- Water drop rejection, palm rejection

Kontron
www.kontron.com



Rugged 12.1" Panel PC Meets Railway HMI Needs

MEN Micro's DC17 is a rugged, fanless and maintenance-free human-machine interface (HMI) for the train driver cabin desk, for example as the operator display for CCTV control, diagnosis and maintenance, or fleet management.

- 12.1" display with LED backlight
- 1024x768 pixels resolution
- AMD Embedded G-Series
- Wireless communication 2G, 3G, 4G, WLAN, GNSS
- MVB interface (optional)
- All external interfaces on M12 connectors
- -40°C to +70°C (+85°C), fanless
- Maintenance-free design
- Compliant to IP65 (front) and EN 50155 (railway)
- Windows and Linux support

MEN Micro
www.menmicro.com



In-Vehicle Panel PC Supports Camera Expansion

The VMC 3021 from Nexcom is a 10.4" all-in-one robust vehicle mount computer designed for the warehouse, port, logistic and material handling markets. It implements the Intel Atom x7-E3950 processor (codename Apollo Lake) on both of Windows and Linux platforms, and offers complete IP65. VMC 3021 is able to support analog camera x3 for security purpose and takes less than 1 second to see video content.

- 10.4" XGA TFT LCD monitor with 5-wire resistive touch
- Built-in Intel Atom x7-E3950 processor, 1.6 GHz
- Aluminum die-casting and fanless design
- Analog camera x3 (CVBS)
- Complete IP65 housing
- Automatic/manual brightness control
- Isolated CAN bus 2.0 x2
- UPS Battery and PoE 802.3af/at (optional)

Nexcom
www.nexcom.com

Panel PCs



Panel PC Family Offers Choice of 64 Models

Taicenn's TPC-DCM industrial series of panel PCs lets you choose between 64 configurations, with 8x Intel processor choices and 8x screen sizes: 15.0", 2x 15.6", 17.0", 18.5", 19.0", 21.5" and 24.0" models ranging from 1024x768 to 1920x1080 pixels. For processors, you can choose between 6th or 7th Gen U-series Core, Apollo Lake or Bay Trail CPUs.

- Various dimensions: 15.0", 15.6", 17.0", 18.5", 19.0", 21.5" and 24.0"
- High quality LCD display with LED backlit
- Multi-level CPU options, including Intel Baytrail, Apollo Lake, Skylake, Kaby Lake
- Memory max. support up to 8 GB or 16 GB (DDR4L)
- Low power, compact and fanless design
- True flat, zero bezel front panel, front IP65 protection
- Self-developed anti-finger print industrial capacitive touch screen
- Panel mount and VESA mount

Taicenn
m.taicenn.com



7" Touch Panel PC Serves Up iMX6 Processor

Powered by the NXP i.MX6 Cortex-A9 Arm CPU, Technologic Systems' TS- TPC- 7990 Touch Panel Computer (TPC) features a 7" capacitive or resistive touch display, high performance CPU subsystem, wide variety of connectivity options and multimedia capabilities. The TS-TPC-7990 is well suited for applications that require a touch-based HMI, including: industrial automation, medical, automotive, self-service kiosks and retail point-of-sale terminals.

- HMI solution featuring capacitive or resistive touch high brightness LCDs
- Processor: NXP's 1 GHz i.MX6 Arm CPU
- Wireless connectivity for remote access and IoT applications
- Storage: on board eMMC, mSATA drive support, microSD card
- High speed industry standard connectors like Gigabit Ethernet and Mini-PCIe
- 2x USB host, 1x USB device, 1x Bluetooth, 1x Wi-Fi, 1x SPI, 1x I²C

Technologic Systems
www.embeddedarm.com



Fanless E3800-Based Panel PCs are IP65-Rated

WinSystems' PPC65B-1x Panel PC Series is IP65-certified and optimized for use in demanding market applications including industrial machinery, utilities, petroleum, transportation, pipeline and food processing, wherever there is a need for tight system integration in a minimal footprint.

- 1.9 GHz Quad-Core Intel Atom E3845 processor
- Up to 8 GB of RAM
- IP65-compliant fanless panel PC
- Front display sealed against water and dust
- -20°C to +70°C operating temperature range
- Wide input power: 12-24 VDC
- 2x Gbit Ethernet ports
- 1x USB 2.0 port (up to 3x with expansion)
- 1x USB 3.0 port
- Watchdog timer

WinSystems
www.winsystems.com

Embedded System Security Live

Coverage of Two Security Events



By
Colin O'Flynn

Colin summarizes some interesting presentations from the Black Hat conference in Las Vegas—along with an extra bonus event. This will help you keep up-to-date with some of the latest embedded attacks, including execute only memory attacks, fault injection on embedded devices, 4G cellular modems and FPGA bitstream hacking.

I know it's not easy to stay current with all the latest embedded security news and attacks. With that in mind, this month I want to bring together a few pieces of research I thought would be of interest to you. To do this, normally I'd focus only on topics from the Black Hat "hacker" conference in Las Vegas, NV. But this time, I'm also including some additional research from the USENIX WOOT (Workshop On Offensive Technology) conference. WOOT took place in Santa Clara, CA shortly after Back Hat.

EXECUTE ONLY MEMORY (XOM)

I'm going to start with a presentation from WOOT, because it's probably of the most importance for embedded developers. The paper in question is entitled "Taking a Look into Execute-Only Memory" by Marc Schink and Johannes Obermaier. The paper attacked the idea of Execute Only Memory (XOM), which is present in different forms on many devices. Generally, this means a memory space (typically flash or ROM) from which we can execute code, but can't actually read data from. This is almost always sold as a way to protect your sensitive code from being read out by an attacker.

To be effective, this is enforced in hardware. The enforcement happens because reads from the execute-only memory space must come from the instruction bus and not from the data bus. See **Figure 1** for an example of these different memory buses. The bottom of that example processor core in Figure 1 has several buses: "ICode" is the instruction bus

interface and "DCode" is a data bus interface. In theory that means that certain memory sections should only be able to send data over the ICode bus. All that said, here the authors found several implementation errors for detecting what counts as instruction

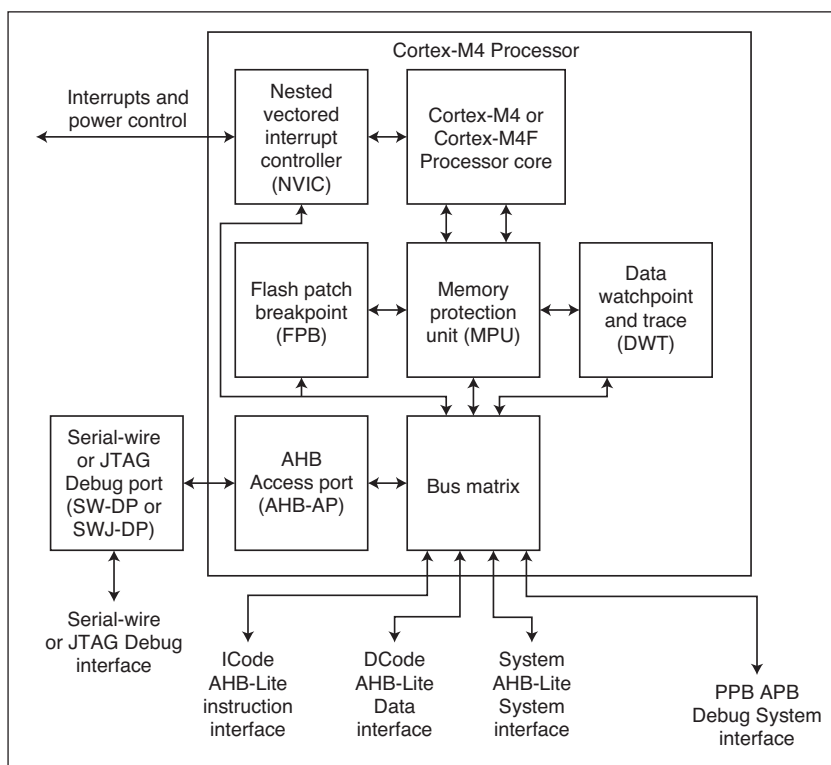


FIGURE 1
An example processor core. Note the bottom side shows different buses for Instruction (ICode) and Data (DCode) access.

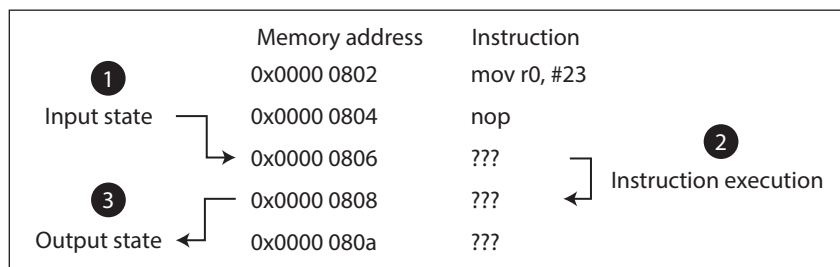


FIGURE 2

Observing the side-effect of instruction execution can reveal both the instruction and the arguments.

vs data bus access. Some devices—the STMicroelectronics STM32F7 in particular is called out in the paper—incorrectly classify certain accesses from the debug access port logic as instruction bus access, and allow reading out of the protected memory.

More fundamentally, the authors also attacked the very idea of XOM. It's instructive to observe the side effects of instruction execution. For example, we can see a single unknown instruction executed in **Figure 2**. We assume we know the input state ("1" in Figure 2), which is the values in all registers, processor flags and similar. We can also observe the state ("3" in Figure 2) after the instruction execution. You can observe this in two possible ways. The easiest is when devices still allow debugger connectivity during XOM execution. The debugger cannot see any of the instructions being executed, but can observe the registers/SRAM. Therefore, as you single-step through the instructions, you get new ideas of the instruction set. If debug access is

not possible, they also demonstrated a second way, which is using an interrupt after each instruction executed. The interrupt handler can then observe or download the system state in a similar manner to the debugger.

Certain instructions would have certain side-effects. A memory load for example would see the value in a register overwritten. But many other instructions would also change a register value—an addition or subtraction, for example, would also overwrite a register. But, if we had a known pattern loaded into memory and the registers, the load would be distinguishable from an addition or subtraction. Therefore, by observing side-effects in a more controlled environment, it becomes possible to discover both the instruction and the arguments. This is iteratively repeated to narrow down similar instructions that might require different starting states to distinguish them. For example, there are several conditional branch instructions, and you would need to distinguish from a "branch if not equal" and "branch if less than".

The authors of the paper extended this idea to demonstrate a full read-out attack on a device, and also worked to prove how this worked against devices that disable debug during XOM execution. The result is that a demonstration of how XOM could be "reversed" by a dedicated attacker.

FAULT INJECTION ATTACKS

Meanwhile, the Black Hat event had (at least) two talks on fault injection. One was my own, entitled "MINimum Failure." I presented the work that I wrote about in the May 2019 issue (*Circuit Cellar* 346). If you don't recall that article, the summary is that you can use fault injection to corrupt the processing of the `wLength` value of a USB packet. This allows an attacker to read back up to 65 KB of memory, which I demonstrated as recovering the private key from a Bitcoin wallet.

Since that article, I extended that work in a few ways. First, I realized that similar processing is present in almost all USB stacks. This includes many vendor-provided examples alongside most USB stacks that vendors provide. I also released my open-source hardware tool that I call PhyWhisperer-USB. It enables you to easily trigger on USB data—along with some basic sniffing of USB 2.0 LS/FS/HS data. You can see a photo of that in **Figure 3**. You can learn more about PhyWhisperer-USB from its github page. A link to it is provided on the *Circuit Cellar* article materials webpage. This includes all the hardware documentation, along with example Python scripts and documentation.

The second presentation was entitled

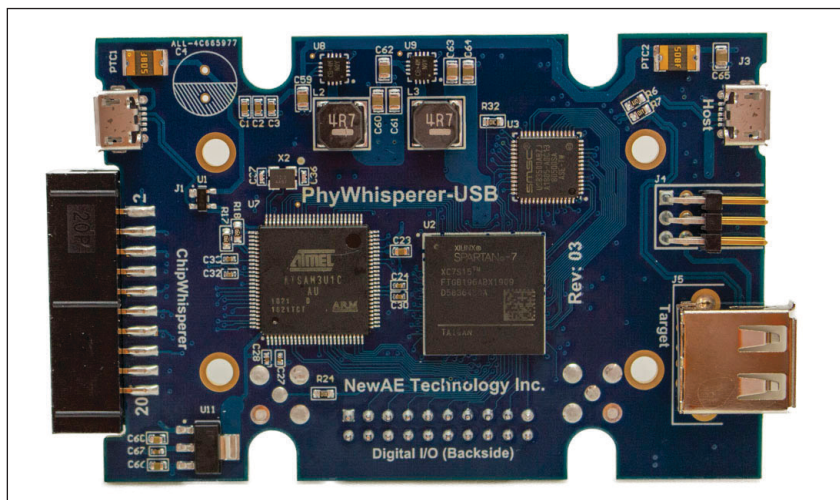


FIGURE 3

PhyWhisperer-USB is an open-source tool for USB 2.0 triggering and sniffing.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials

“Chip.Fail” by Thomas Roth, Josh Datko and Dmitry Nedospasov. This demonstrated basic voltage fault injection attacks on various devices, which helps demonstrate that you should consider fault-resistant coding techniques to help prevent some of these attacks. These fault-resistant techniques are something I plan on talking about in my next article, so keep an eye out for the January 2020 issue where you can start your new year out by working with fault-resistant designs.

4G MODULE ATTACKS

Another interesting presentation was one from the Baidu Security Lab, which presented a talk entitled “All the 4G Modules Could be Hacked.” This talk is of interest to anyone who uses cellular modules in their products, because these modules are a common method of adding remote connectivity for data logging and remote control. These modules are often found as part of a mini-PCIe card in an embedded Linux system, but they take other forms as well. The talk focused on both problems with the 4G modules, including common issues with the user configuration and fundamental problems in the baseband device itself. I’ll give you a few examples of both of these.

One of the most common problems was that the security implications of adding that module may not be understood. Some carriers, for example, would add multiple 4G devices to a network without isolating them from each other, allowing someone to scan (and connect to) other devices. As a user, you don’t necessarily know the configuration of the access point to which you are connecting, so you need to assume someone else on the network can find this device. A screenshot of the port scan test they performed is given in **Figure 4**.

Assuming someone does find the device, what can they do? Part of this presentation showed how services (such as TELNET or SSH) were running with hard-coded usernames and passwords. It was possible to recover these hard-coded passwords, giving someone remote access to the system. And, because these appeared to be reused across the deployment, the high effort involved in breaking one password now allowed a more widespread attack to apply.

In addition, there are some more fundamental security implications of adding these modules. Most of them will downgrade to older (GSM or “2G”) protocols if they cannot reach a 3G/4G base-station. Unfortunately, these older standards can be easily abused to force the device to connect to a malicious cellular base station, which was also demonstrated in this talk. Such attacks

```

~ masscan 10.78.252.226/22 -p 23,5555 --rate=50
Starting masscan 1.0.4 (http://bit.ly/14GZzCT) at 2019-07-18
-- forced options: -sS -Pn -n --randomize-hosts -v --send-etc
Initiating SYN Stealth Scan
Scanning 1024 hosts [2 ports/host]
Discovered open port 5555/tcp on 10.78.252.175
Discovered open port 5555/tcp on 10.78.253.35
Discovered open port 5555/tcp on 10.78.253.20
Discovered open port 5555/tcp on 10.78.252.73
Discovered open port 5555/tcp on 10.78.255.190
Discovered open port 23/tcp on 10.78.255.191
Discovered open port 23/tcp on 10.78.255.202
Discovered open port 5555/tcp on 10.78.255.203
Discovered open port 5555/tcp on 10.78.255.90
Discovered open port 5555/tcp on 10.78.253.184
Discovered open port 5555/tcp on 10.78.255.234
Discovered open port 23/tcp on 10.78.253.141
Discovered open port 5555/tcp on 10.78.253.210
Discovered open port 5555/tcp on 10.78.252.216
Discovered open port 5555/tcp on 10.78.252.48
Discovered open port 5555/tcp on 10.78.253.4
Discovered open port 5555/tcp on 10.78.252.159
Discovered open port 5555/tcp on 10.78.252.209
Discovered open port 5555/tcp on 10.78.255.84

```

FIGURE 4

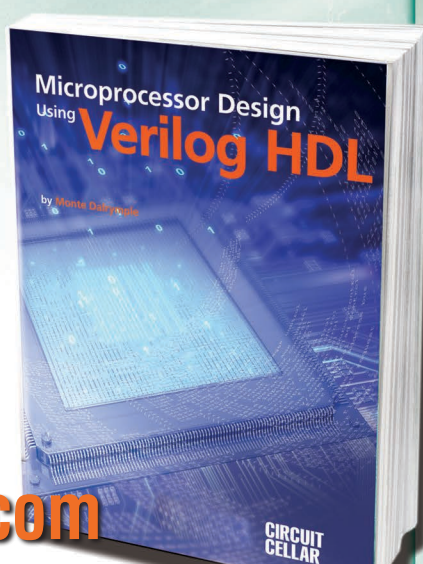
Port scan results can reveal open ports on 4G networks, where the utility has not isolated clients.

Verilog HDL

With the right tools designing a microprocessor can be easy.

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

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

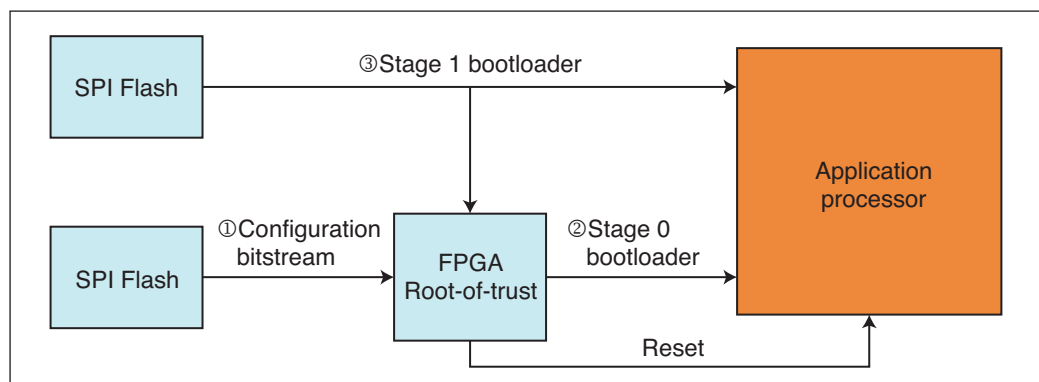


cc-webshop.com

CIRCUIT CELLAR

FIGURE 5

Cisco Root of Trust relies on a FPGA to perform initial device boot, along with resetting the processor during security violations.



are well-known, but again someone looking to simply add easy cellular connectivity to their product may not consider that an attacker could easily observe (or control) any network traffic.

Forcing use of higher-layer encryption is a requirement to survive in such an environment, for example, by only allowing encrypted traffic going over HTTPS or SSH. In addition, turning off 2G support may be wise to prevent these older standards from being used at all. Many carriers around the world have stopped supporting 2G in order to free up bandwidth for 4G, with most major carriers already fully turning off their 2G networks or announcing dates in the next few years to do so.

REWRITING FPGA BITSTREAMS

A final attack of interest was one with a very odd name. The name is depicted by three angry cat emojis, which we can't

represent on these magazine pages. So, instead you can go by the pronounced version of that attack that the paper's authors suggest: "Thrangycat." I'll get to the title of paper in moment—you'll see why. This attack invalidated a huge amount of security assumptions on Cisco gear, which used an FPGA as a hardware root of trust. A block diagram of the Cisco setup is shown in **Figure 5**.

In the Cisco setup, the FPGA bitstream is loaded from a SPI flash. The FPGA forms the "root-of-trust" because it then loads the basic bootloader into the application processor (main processor). The FPGA can then observe the loading of the Stage 1 bootloader, and during the loading of this Stage 1 bootloader the FPGA can validate that correct code is being loaded—in other words, that no attacker has modified the system.



FIGURE 6

This tool is part of the open-source Spartan 6 bitstream reverse engineering efforts.


If any security violation is found, the FPGA uses the reset line to kill the system. The idea here being that the FPGA is performing a hardware action to prevent a clever attacker from doing low-level modifications. The FPGA has a 100-second timeout before it resets the system. And that is the origin of the title of this talk presented at Black Hat by Jatin Kataria, Richard Housley, and Ang Cui: “100 Seconds of Solitude: Defeating Cisco Trust Anchor with FPGA Bitstream Shenanigans.”

But, as you know, the FPGA itself is not fixed hardware. It loads a bitstream from a SPI flash. Modifying the SPI flash allows you to modify the FPGA bitstream. Because the FPGA bitstream itself does other tasks (including loading the Stage 0 bootloader), the original bitstream does need to be loaded. But the Cisco assumption was that reverse-engineering the full bitstream to modify the design would be very difficult (which is true). Luckily a full reverse-engineering isn't actually needed. In this case, they only need to change the reset pin output such that it is no longer asserted. This minor modification is something that can be reverse-engineered, since it only modifies the output drive of the FPGA.

To assist with this work, they built a FPGA bitstream visualizer tool. Back in June 2014, I actually talked about the Spartan 6 bitstream partial reconfiguration, and discussed some of the bitstream format since it is partially documented. My article was called “Partial FPGA Configuration,” (*Circuit Cellar* 287, June 2014). But back then I didn't have such a nice visualization tool as you can see in **Figure 6!**

To round out the attack, they demonstrated how someone can remotely reload the SPI flash that holds the FPGA bitstream. The result of this means a remote attacker could also disable the ability to recover the SPI flash—the reprogramming of the SPI flash is done via a FPGA feature. The attacker can build a new bitstream that simply disables those I/O pins once the bitstream is booted. This requires a technician to physically reprogram the SPI flash on-board the affected Cisco product.

KEEPING UP ON MORE ATTACKS

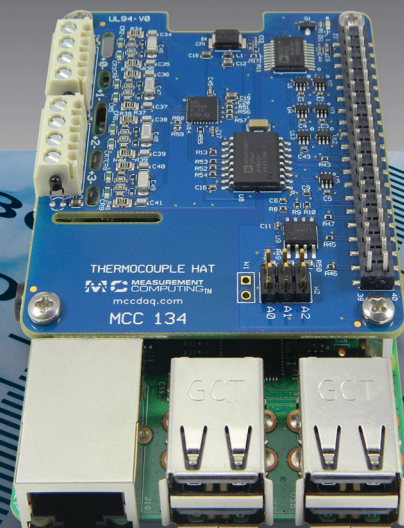
Hopefully, this summary has given you a look at a few new attacks from 2019. It's hard to keep up with everything that comes out (even for myself), and for space reasons I can't hope to fit every important attack into this article. But having an idea of these attacks is useful as you design your own products. 



ABOUT THE AUTHOR

Colin O'Flynn (colin@oflynn.com) has been building and breaking electronic devices for many years. He is an assistant professor at Dalhousie University, and also CTO of NewAE Technology both based in Halifax, NS, Canada. Some of his work is posted on his website at www.colinoflynn.com.

Bring Your Pi to Work



Thermocouple Measurements on a Pi

The MCC 134 thermocouple measurement HAT provides best-in-class, professional-grade accuracy. Up to eight MCC DAQ HATs can be stacked onto one Raspberry Pi® allowing users to create multifunction DAQ solutions based on this low-cost computer.

MCC 134 DAQ HAT

- Four thermocouple inputs
- 24-bit resolution
- 1 second update interval
- Supports most thermocouple types
- Complete SW library for easy programming
- Full set of examples in C® and Python™

www.mccdaq.com/DAQ-HAT

 MEASUREMENT COMPUTING™

©2019 Measurement Computing Corporation • info@mccdaq.com

Picking Up Mixed Signals

Bluetooth-Enabled ECG Monitor Using the Cypress PSoC 6 MCU

By
Brian Millier



Brian has written articles in the past about projects that use Cypress Semiconductor's PSoC MCUs, including his most recent piece about the variable frequency drive project he built using the SoC5LP MCU. This month, he explores the latest offerings from this MCU family, the PSoC6 5LP MCU. In this project article, Brian selects the Cypress CY8CPROTO-063-BLE to build a Bluetooth-enabled ECG monitor.

In my last column, I described a Variable Frequency Drive project I built, using Cypress Semiconductor's PSoC 5LP microcontroller (MCU). Although I had used PSoC MCUs in the past, I was quite impressed by the advances they had made in both the hardware and in the Creator 4.2 application that you use to develop code for the PSoC family. I decided to continue to explore this family of devices while things were still fresh in my mind.

I looked at the latest offerings in the PSoC family—the PSoC6. Earlier generations of the PSoC family were each based on a specific central processing unit, beginning with a very limited custom CPU used in the PSoC1 family, advancing to an 8051 derivative (PSoC3), and then finally an Arm Cortex M3 in the PSoC 5LP. What differentiates each PSoC device in a given generation are the number and scope of the

custom analog and digital blocks contained in that device.

The PSoC 6 is the first PSoC generation to contain any wireless connectivity. The higher-end PSoC 63/64 models have Bluetooth Low Energy (BLE) functionality built in. The PSoC 60,61 and 62 contain a similar CPU, but no BLE function. All devices in the PSoC6 line are dual core, containing both a Cortex M4 and a low-power Cortex M0+. If you check out the Cypress development board offerings for the PSoC6, it appears that Wi-Fi is also supported in some PSoC 6 devices. This is not true, however. The PSoC 6 development boards that feature Wi-Fi do so via a separate CYW4343W Wi-Fi/BT daughterboard. I wasn't interested in such a development board, because I generally use the Espressif Systems ESP8266 or ESP32 modules for anything I do that requires Wi-Fi.

In the past, I always liked the way that

Cypress designed its development boards. It has the usual assortment of larger boards containing a variety of peripherals, including displays and cap-sense buttons. However, Cypress also made small modules containing only the bare-essential components, in DIP-style packages on 0.1" centers. These suit my construction practices perfectly. As a bonus, such boards for the PSoC3,4 and 5LP families (which I had used in the past), were priced between \$5 and \$15. These are obviously sold below cost, since the parts on the snap-off programmer alone (included with these modules), would cost more.

I chose Cypress's low-cost CY8CPROTO-063-BLE for this project. The daughterboard containing the PSoC6 MCU/BLE antenna, (right), and the snap-off programmer (left) are shown in **Figure 1**.

BLUETOOTH LE AT FIRST GLANCE

I've created many Wi-Fi projects in the past, using the ESP8266 or ESP32 modules and compatible Arduino IDE/libraries. Most of the complexity of the Wi-Fi protocol is hidden by the Arduino Wi-Fi library, and it's not that hard to get a project up and running (at least at the security level imposed by a WPA2 connection to a home wireless router). Programming the MCU firmware is a lot more complicated in the case of Bluetooth.

The simplest approach to a Bluetooth-enabled peripheral uses a Bluetooth module containing a standard UART interface. I've used the Adafruit "Bluetooth LE UART Friend" module. It's easy to use because Adafruit supplies not only a sample Arduino sketch written for it, but also matching iOS and Android smartphone apps.

However, once you go beyond a simple BLE-to-UART bridge device, things become much more complicated. BLE is loosely based on a Server/Client concept, though what you might normally consider a server may not be how it is defined in the BLE environment. The basic concept is that one BLE device can pair up to another, completely unknown BLE device, determine which of the Bluetooth SIG profiles that BLE device emulates, and then communicate measurements/commands back and forth.

The BLE devices don't need advance knowledge of anything specific about each other, since all the relevant measurement/control parameters are defined in whatever Bluetooth profile(s) the device is emulating. For example, a smartphone BLE app that monitors a person's heart rate, should work properly regardless of what brand of heart rate monitor it pairs up with. While this works in theory, I suspect that manufacturers often tailor the BLE profiles enough so that their

smartphone apps only work with their own hardware devices.

While this capability is very useful, it results in a complicated communications protocol—much more complex than what you would come up with, if you were designing a custom device for a specific, dedicated purpose.

I generally peruse datasheets, specifications, and so on before getting too involved in a project. I examined the Bluetooth SIG documentation early on, but found it hard to understand. I'm quite familiar with Wi-Fi IP #s, MAC #s and SSIDs, but the many UUID numbers involved in BLE are much more elaborate. They're basically 128-bit values, but are expressed as long, hyphenated ASCII strings: not at all like the "standard" notations used for IP and MAC addresses. Furthermore, the common BLE SIG profiles use "shortened" 16-bit values, which are concatenated with a common base value to provide the full 128-bits. You will have to get used to typing these long UUIDs into your programs without errors, or nothing will work!

The PSoC Creator 4.2 program contains support, in the form of a component configuration "wizard," to help you write a BLE-enabled application. If you have used the PSoC Creator application with earlier PSoC devices, you'll be familiar with dragging the required hardware "components" onto your "schematic" workspace (the TopDesign tab). Then, when you invoke the Build -> Generate Application option, Creator will add the various ".h" and ".c" files needed to implement comprehensive APIs for each hardware component that you added to your design. To be clear, these hardware components are the built-in peripheral blocks contained in the PSoC device, itself. You don't have to figure out what driver files are needed, since Creator software adds them all for you.

Double-clicking on a "component" brings up a graphical configuration "wizard," used to define the initial configuration of the hardware component. This "wizard" is as simple or complex as needed to configure the hardware component that it serves.

To add the BLE function, the same process

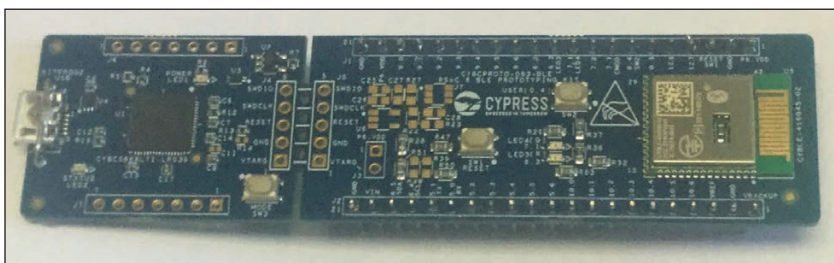


FIGURE 1

The Cypress CY8CPROTO-063-BLE prototype board that I used for this project. Note the snap-off programmer on the left.

is followed. You first add the BLE component from the Component Catalog window to the right of the Creator screen (in the Communications folder). When you double-click on the BLE component, a complex configuration “wizard” appears (**Figure 2**). In the “General” tab, you select whether your PSoC device will be a Peripheral, Central, Broadcaster or Observer. For this project, a Peripheral is the right choice.

Here you also decide how you want the BLE functionality to be implemented—either using both the CM0+ and CM4 Arm cores of the PSoC, or just one of them. I readily admit I don’t know all the pros and cons of this choice, apart from knowing that for low-power applications, it makes sense to use the CM0+ core to handle the low-level BLE functions. The CM4 core is powered up only to handle application-specific code, when needed. I found that the Cypress sample programs generally used both processors.

Most of the BLE configuration is done using the GATT tab. This is where you specify what BLE SIG profile you want the device to emulate. You can either pick one of the standard profiles (Health Thermometer, Heart Rate, Cycling Speed, and so on), or define your own custom one.

This is where I found things got complicated. It seems very difficult to come up with a custom profile, yourself, unless you are well-versed in BLE standards and other subjects. Speaking as a BLE novice, it’s my opinion that you stand virtually no chance of developing your own custom BLE device, if you try to design both the device hardware/software and the smartphone app by yourself, from scratch. Instead, I followed this procedure:

- 1) Program the PSoC 6 with one of the sample BLE programs that comes with the Creator software package. From the Creator “Start” page, this is done by clicking on “Find Code Example.”
- 2) Test for connectivity/functionality using Cypress’s CySmart application (iOS app for iPhone/iPad, Android app or PC application).
- 3) Examine the PSoC sample program closely, and modify it to perform the task you have in mind. At this stage, you may or may not be able to use the CySmart app for testing, if you are not using a custom profile or one of the standard profiles for which Cypress has provided a sample.
- 4) Write a custom smartphone app to handle the required function(s).

To be honest, I didn’t have a specific project in mind at the outset. I just wanted to learn how to use the PSoC 6 in a BLE application. Even starting with this “clean slate,” I didn’t find many of the Cypress BLE sample programs to be very relevant to me. I settled on the “PSoC 6 BLE Multi-Slave” sample program. This implements a BLE multi-slave functionality containing the following services:

- 1) Device Information Service
- 2) Health Thermometer Service
- 3) Custom service controlling an RGB LED
- 4) Custom service performing a 128-bit read/write
- 5) Custom notification service.

Although this sample program is named “Multi-Slave,” it actually implements a multi-master, multi-slave device, acting as a Peripheral and containing the above five GATT servers. Among those servers were one that sent out data (Health Thermometer) and one that received data (RGB LED control). Those were the ones I figured I could modify to fit my own tasks, regardless of whether those tasks were monitoring data or controlling something connected to the PSoC 6. I didn’t have any use for either the device information service or the custom notification service. While the custom 128-bit read/write service looked useful, I could only figure out how to write the 128-bit data, not read it, so I left it alone.

When I later decided that I wanted to build a BLE-enabled ECG (electrocardiogram) monitor, I knew I’d be sending the ECG data from the PSoC 6 device, and there would be no need to control anything on the PSoC 6 from the iPad app.

I chose to modify the Health Thermometer Service. Doing so would allow me to monitor the values sent from my PSoC 6 firmware using the CySmart iOS app provided by

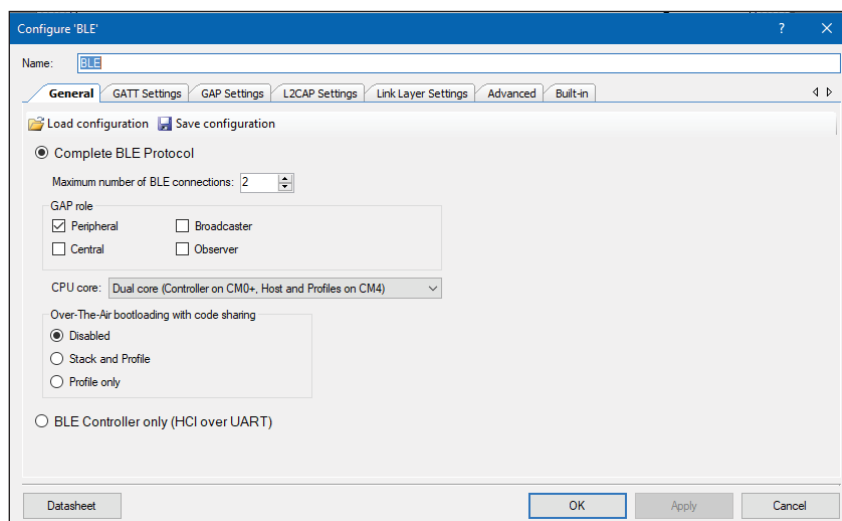


FIGURE 2

The main page of the BLE component’s configuration wizard, part of Cypress’ Creator IDE application. Many of the important settings are exposed in the GATT Settings tab.

Cypress. That is, I wouldn't need to have my own iOS app ready to go in order to test the PSoC6 firmware I was trying to write.

The “.c” and “.h” files from the Cypress “multi-slave” sample are shown in **Figure 3**. Many more files are needed for this program than those shown here. Most of the other files are generated automatically by the Creator application (for the specific hardware components that you have added to your “schematic”). These are the files I needed to work with to customize the “multi-slave” sample for my own purposes.

I added the “ECG.h” and “ECG.c” files, myself. Actually, the sample program originally contained “temperature.h” and “temperature.c” files. I renamed these two files, and modified them to replace the original code, which read a thermistor via the PSoC6 SAR ADC, with code that measured the voltages coming out of the ECG amplifier module. In both cases, the ADC used the 3.3 V V_{cc} as a reference, so no change was needed there. However, Cypress used what they call “Double Sample Correlation” to measure both the voltage across the thermistor and across a 10 k Ω 1% reference resistor, to determine the thermistor resistance. This value was then converted to temperature using the Steinhart-Hart equation.

I changed this code to take just one ADC reading of the voltage directly from the ECG amplifier. Note that the code in the “ECG.c” file merely takes an ECG reading. It must be called at a specific sample rate to be useful. That is done in the “ble_application.c” file. Specifically, in this file, the function, “ble_ProcessEvent” is a loop that constantly handles BLE events. When I say I take one ECG reading, the “ADC_GetResult16” function that I use takes 256 samples and averages them. The PSoC6's SAR ADC is very fast, so in the ADC setup wizard, the ADC is configured to take an average of 256 settings, which minimizes noise.

Within that loop, the original Health Thermometer readings were taken at a 4 Hz rate, triggered by the PSoC's watchdog timer. To be useful, ECG readings should be taken 30+ times per second. I added a PWM component to my “schematic,” set for a 40 Hz rate. In past PSoC projects, I would merely wire an “Interrupt” component to the PWM output. Creator would then generate all the needed code for this interrupt, including an “isr.c” file--where you would add the code you wanted executed when the interrupt occurred. This method no longer works with PSoC6 BLE applications. For PSoC6, Cypress now uses the Peripheral Driver Library, which is part of what they call “middleware,” because it includes drivers from other sources, including

RTOS and emWin. I couldn't quickly figure out the new way to handle a PWM interrupt, so I wired the PWM output to Port 10.5, and tied that to Port 10.4 (set as an input). I could poll the state of the P10.4 pin, giving me a way to pace the ECG readings at a 40 Hz rate. When I get more time, I'll try to figure out how the PSoC6 firmware handles interrupts.

The BLE standard Health Thermometer Service is defined to output the temperature as an IEEE-11073 format floating-point number. However, the PSoC's “C” compiler uses the standard IEEE-754 single-precision format. Therefore, a format conversion was done in the original sample program. For my purposes, I chose to take four, sequential 8-bit ECG readings and pack them into the 4-byte IEEE-753 floating point variable originally used for the temperature value. I did this so that only one-quarter as many BLE packets needed to be transmitted.

When I later wrote the iOS app, it was only possible to receive and graph the ECG data at a 40 Hz sample rate, by sending the data in this “packed” format. With only one ECG value per BLE packet, my iOS program routinely failed to collect all the data, when pushed up to the 40 Hz rate. In **Figure 4**, you can see the iOS app running. Here, the PSoC6 is sending out a triangle waveform via BLE. The data loss without data-packing, mentioned above, was clearly visible here, when it occurred. After implementing the 4 sample/BLE packet protocol, the data were received properly, as shown here.

In summary, the ADC reading of the ECG waveform takes place in the “ECG.C”

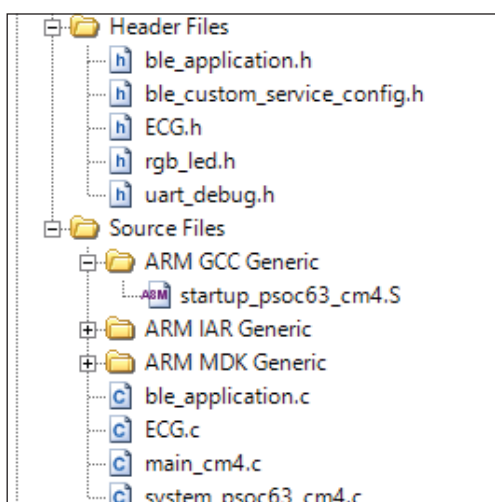


FIGURE 3

Many files are generated after you design the PSoC6 “hardware” aspects of your project and click on “Build Application.” The ones shown here are a subset of those, and contain the files where I made modifications/additions to the sample program to match my project.

ABOUT THE AUTHOR

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

file. The processing of BLE events, including the 40 Hz sampling of the ECG signal and its transmission, takes place in the "ble_application.c" file. The entire, complex, BLE communications protocol is handled

transparently by routines in other files, most of which were added automatically by the Creator application when the BLE component was added to the "schematic."

TESTING USING CySMART

When I first ordered my CY8CPROTO-063-BLE board, I considered also purchasing the Cypress CySmart USB dongle for testing the BLE link on my PC. However, Cypress also provides an iOS app that I could use on my iPad, so I didn't think I would need the dongle. It turns out that the CySmart iOS app can handle the various BLE profiles used by some, but not all, of their sample programs, nor can it handle a custom profile that you design yourself. You would need the CySmart USB dongle for this. I have since purchased the dongle, but unfortunately, I didn't have it while designing this project.

I first downloaded the Cypress multi-slave firmware to the PSoC6, and tested it using the CySmart iOS app on my iPad. All of the BLE Services included in this sample showed up in the CySmart app and worked properly. I didn't have the thermistor/feed resistor hooked up, as would have been the case on the Cypress development board to which this sample is targeted. As a result, I got random temperature readings that varied at the 250 ms watchdog period.

As mentioned earlier, I had decided to modify the standard Health Thermometer profile to handle the ECG data. When I started making changes to the sample program to read the ECG signal, I initially didn't pack 4 ECG samples into one BLE packet. I also did not trigger the ADC at the 40 Hz rate, but rather, at the much slower rate of 4 Hz used in the original sample. I also sent the ECG readings to the PSoC6's UART Tx port for debug purposes. Doing it this way allowed me to see the "Raw" ADC readings taken from the ECG amplifier, and to monitor them in the Health Thermometer section of the iOS CySmart app. Whatever the ADC value was, it showed up identically in both the UART output stream and in the CySmart app's Health Thermometer window.

The Cypress multi-slave sample program is written to send out many debug status messages during the various phases of a BLE link connection. By default, however, these messages are turned off. I would not have known how to enable them, had I not watched the "PSoC6 101" series of YouTube videos by Cypress's Alan Hawse (which I thoroughly recommend). You must edit the `uart_debug.h` file and change line 75 to read:

```
#define UART_DEBUG_ENABLE (true)
```

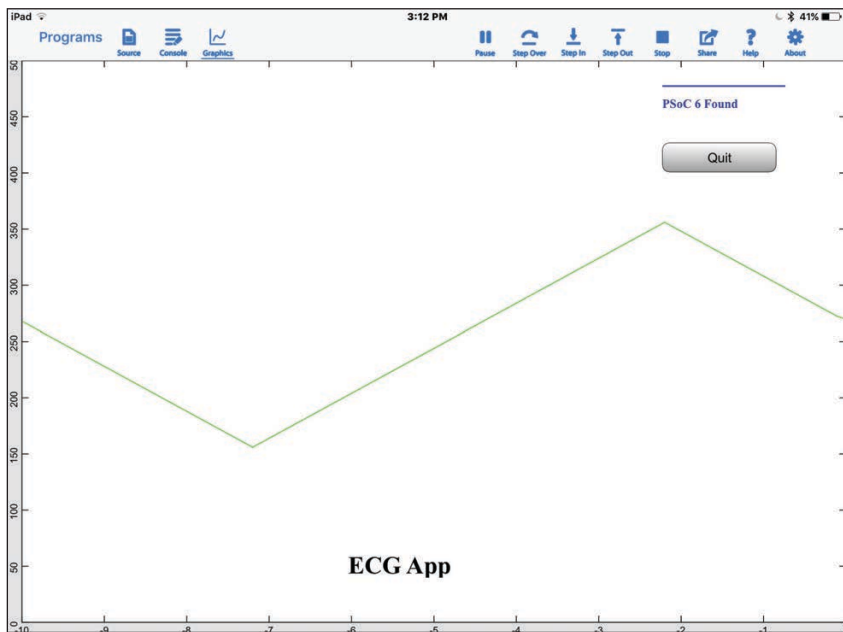


FIGURE 4

My techBASIC program running. At this stage, the PSoC6 was generating a triangle "test" wave, rather than an actual ECG signal.

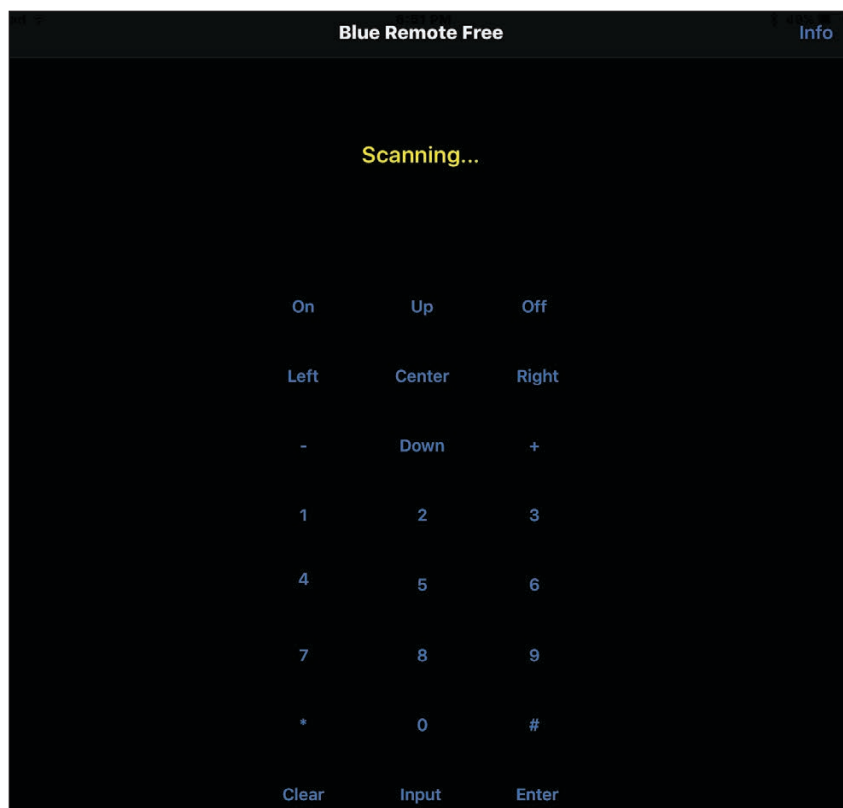


FIGURE 5

While not connected with this project, this is a screenshot of the Blue Remote iOS app. It can be used for certain BLE projects, without the need for you to write your own custom iOS app.

Once done, you can use statements, such as the following in your code for debugging:

```
DebugPrintf("S2= %d \r\n",temperature);
```

Once I had my program working properly with the CySmart app's Health Thermometer window, I was free to change my code to:

- 1) Pack 4 ECG samples per BLE packet.
- 2) Increase the ADC sample rate to 40 Hz.
- 3) Scale the ADC values down to 8-bits.

The next step was to write a custom iOS program for my iPad, to receive and graph the ECG data.

BLE APPS FOR iOS DEVICES

Writing native apps for iOS devices is not an easy job, unless you plan to do it regularly. Native apps must be written in Apple's Objective C or Swift languages, and the Xcode development platform for these languages will only run on a Mac computer. I have an older Mac Mini as a spare computer, but I believe its OS is too old to run current versions of Xcode. Also, you must register as an Apple developer to be able to download your app to the iOS device during testing. I had checked out Objective C several years back, and found it to be hard to work with. Granted, I wasn't as experienced with the C language then as I am now.

As previously noted, if your BLE application is not too specialized, an Adafruit BLE "Friend" module, which acts as a BLE-to-UART bridge, might be your best bet. Adafruit provides a "generic" iOS app called Bluefruit, which communicates with their modules. Another choice for Adafruit BLE "Friend" modules is an iOS app called Blue Remote (**Figure 5**). This app acts like a TV remote control, and sends out an ASCII string descriptor via BLE for each button pushed. The data communication is unidirectional—from the iOS device to your BLE "Friend" module.

Getting an iOS application to communicate with the PSoC 6 BLE module was more involved. I had ruled out writing a native app in either Objective C or Swift, for the reasons mentioned above. The Adafruit apps only work with their BLE "Friend" modules. Some software companies produce development software that allows you to write iOS apps without using Apple's development languages. I looked at them briefly, but felt they wouldn't be a good match for my PSoC 6 BLE device.

I had earlier installed an iOS app called techBASIC, by Byte Works, on my iPad. It consists of a BASIC interpreter that runs on the iOS device. You write your application in BASIC, and run this application when you need to use it. techBASIC contains all the expected

BASIC functions, and provides high-level APIs to access most of the I/O devices found in iPhones and iPads. You can access an iOS device's WiFi connection, Bluetooth LE, the various gyro, accelerometer and magnetometer sensors, and the display and touchscreen features, to name a few. Due to the many high-level functions available, it's possible to write powerful graphical programs with modest amounts of coding. techBASIC runs on iOS version 8 and above. I am still using iOS 9, so I haven't tried it on newer iOS versions.

There are a few caveats. When you open this app, you are presented with a list of sample programs provided with the app plus those that you have written (**Figure 6**). You must choose the applicable app to start it running. If you're both the developer and end-user, this is straightforward, but other users wouldn't find it quite as transparent as a "dedicated" app. Also, there doesn't appear to be a way either to protect your code from being modified by the end-user, or to prevent end users from viewing the source code.

To overcome these limitations, Byte Works sells a techBASIC App Builder program that converts your techBASIC source code into a form that Apple's Xcode development platform can compile. This application—just like the Xcode development software—must be run on a Mac computer, and you must be registered as an Apple developer. I don't have any experience with this particular program.

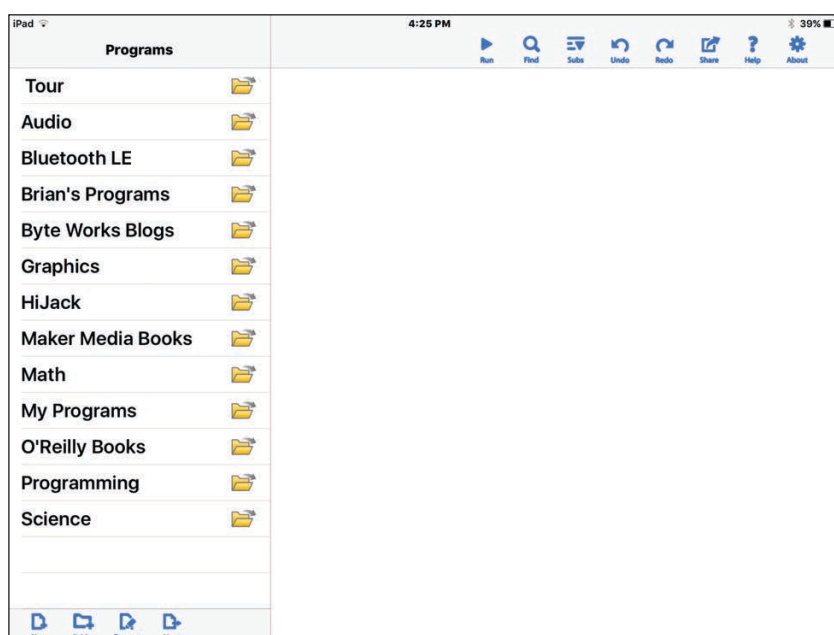


FIGURE 6

The opening screen for the techBASIC iOS app. The user must choose a program from the folders listed at the left. Once selected, the program will run automatically.

THE techBASIC APP

I followed the same procedure for the iOS app that I used in writing the PSoC 6 BLE firmware. That is, I used one of techBASIC's sample programs (designed for the Texas Instruments SensorTag BLE modules), and incrementally began modifying it to match the UUIDs and data structure present in my PSoC 6 project's firmware. I must admit that even though the documentation that comes with techBASIC is very good, the BLE functionality is quite complex. I doubt I could have written a working app, had I been forced to write it from scratch.

Since techBASIC runs on iOS devices, even when using an iPad there is a limited amount of screen space to run a development IDE. Byte Works gets around this by providing three discrete "views":

- 1) Source View—where you compose/edit your source code.
- 2) Console View—basically for debugging. Anything that your program outputs using

the PRINT command is shown in the console view.

- 3) Graphics View—the graphic user interface that the user interacts with during program execution.

During program execution, you can switch between the Graphics and Console views. Thus, if you sprinkle your program with PRINT statements at strategic spots, you can debug your program easily by switching to the Console view to see how things are going. When I was modifying one of the sample programs to suit this PSoC6 project, I placed a lot of PRINT statements in the source code, to see how the BLE discovery/pairing process was proceeding.

Before I describe my source code, I'll mention that techBASIC's Bluetooth API makes heavy use of "callback" functions. These functions are not a part of your program's "main" loop, but instead are invoked automatically by techBASIC when specific BLE messages are received by the iOS device. Since such messages can arrive at random times, it makes your program is much easier to write if you don't have to check for all the different types of BLE messages from within your program's main loop. Instead, you add your application-specific code to the body of the various callback functions, and techBASIC acts as a "traffic cop" by sending the numerous BLE messages to the correct callback function.

In simple terms, a BLE program must:

- 1) Perform a "scan" to discover any BLE devices within range.
- 2) Determine what BLE profile(s) this device supports.
- 3) Tell this device to start sending the desired data (if it is not broadcasting that data by default)
- 4) Add your own code to the appropriate callback function(s), to receive and process that data.

techBASIC supports breakpoints. Even if you are not using breakpoints, you should be aware of them, because it's easy to place them by mistake. In the Source View, there is a narrow pale blue region to the left of your code window. If you touch this area of the screen, a blue right-arrow will appear, and a breakpoint will be placed at that line. Program execution will halt when this line is reached. You can remove this breakpoint by tapping on the blue arrow.

techBASIC's program structure is somewhat different from what you might expect if you program in C. There you would expect a "main" function that generally

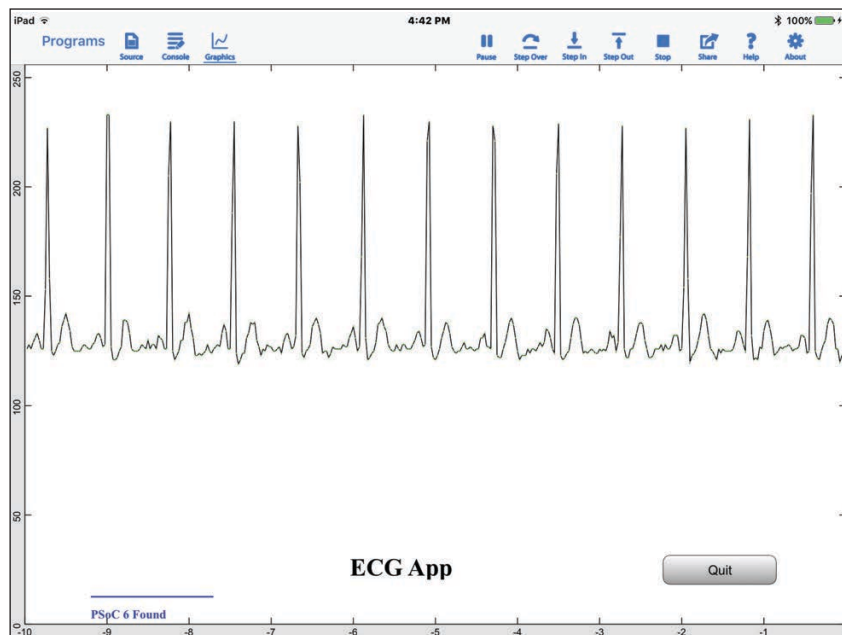


FIGURE 7

The techBASIC program running with actual ECG data (compare with Figure 4 test run).

For detailed article references and additional resources go to:
www.circuitcellar.com/article-materials

RESOURCES

Analog Devices | www.analog.com

Cypress Semiconductor | www.cypress.com

Espressif Systems | www.espressif.com

Infineon Technologies | www.infineon.com

Microchip Technology | www.microchip.com

contains an endless loop. That loop will either invoke actions or monitor something continuously. Alternately, it could contain an empty loop, leaving callback functions to do the actual work. Similarly, in the Arduino environment, you would have a “setup” function to initialize peripherals and such, and a “loop” function to handle ongoing events.

In techBASIC, you define all your variables and objects at the top of the program, and place your initialization code next. There is no “loop” function as such. Your program’s various functions immediately follow. They are generally callback-type functions or functions that are themselves called by those callback functions.

The initialization portion of my program consists of only a few lines:

```
services(1) = "1809": ! Health
thermometer
serverFound=0
BLE.startBLE
PRINT "Program Start"
BLE.startScan(uuid)
SetupGUI
```

First, I define the profile name that I’m expecting the PSoC 6 project to support. I use the standard BLE “Health Thermometer” profile used in a PSoC 6 BLE sample program, modified to handle my ECG data stream. The “serverFound” flag variable is initialized to zero. Once a BLE connection is made to a device with the name “PSoC 6,” this will be set to 1, and other BLE callback functions will be enabled.

BLE is the class supporting BLE operations, and the “startBLE” function enables it. The “startScan” function scans for BLE devices within range, and the UUID of the discovered device is stored in the “uuid” string variable.

The SetupGUI function defines and places the various buttons, labels and other graphics elements on the graphics screen. The GUI for this project is pretty basic—a graphics area in which to plot the ECG data, a QUIT button to exit the program, and a “progress bar” indicating the status of the various stages of the BLE scan/discovery process.

Beyond the initialization functions described above, all other program activity is handled by callback functions, which are triggered by the arrival of BLE data from the PSoC 6 BLE device.

The Cypress BLE sample program that I modified for this project contains five services (listed earlier). Although I only used the “Health Thermometer” profile, modified for the project, the PSoC 6 still “advertises” the other profiles, and the diagnostic PRINT statements in my techBASIC app displays

these other profiles, even though they are not used in any way.

After the program starts the BLE scan, the callback function “BLEDiscoveredPeripheral” is triggered for each BLE device that the iPad finds within range. I check for a “peripheral.BLEname” equaling “PSoC 6” (the name I assigned in the PSoC 6 BLE component’s name variable). When

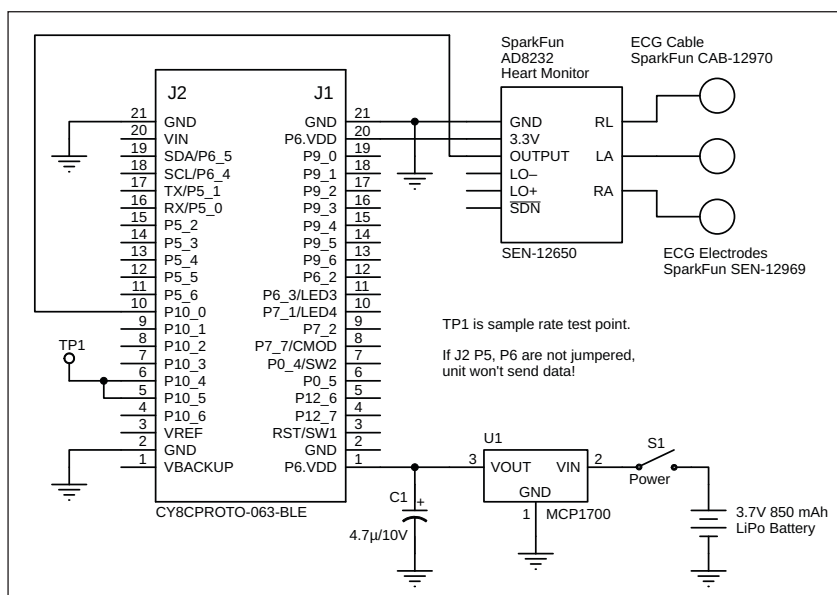


FIGURE 8 Schematic diagram of the project. The ECG electrodes are labeled as RL (reference lead), RA (right arm) and LA (left arm).

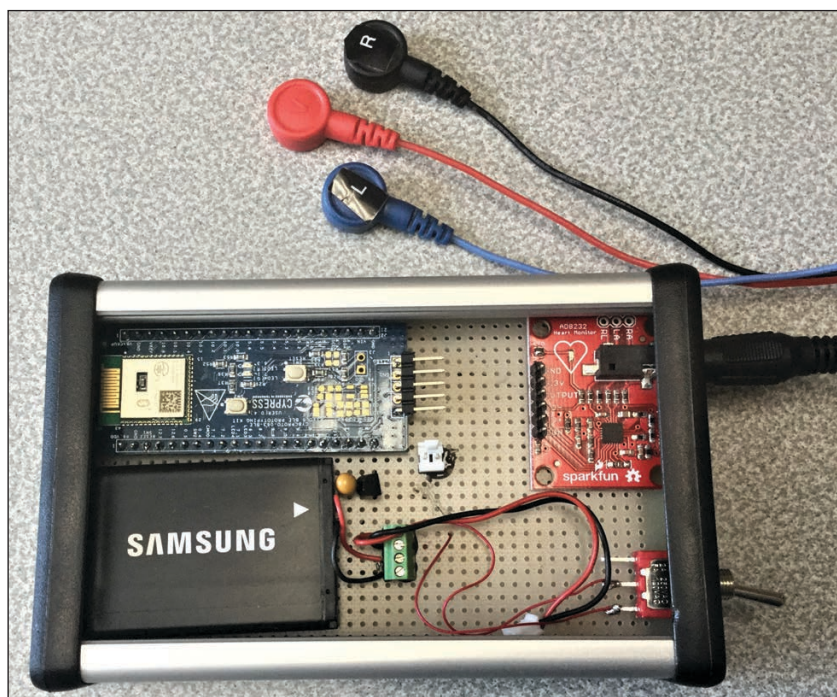


FIGURE 9 The unit just before sliding the top panel into place. While not visible, the GND bus of the circuit is wired up to the aluminum case for shielding.

found, for diagnostic purposes I PRINT out the UUID of the device and its name (PSoC 6). I set the “serverFound” flag variable to 1 and perform a BLE.stopScan.

The next callback function invoked is “BLEPeripheralInfo.” It provides an INTEGER variable, “kind.” If “kind” equals 1, the program calls the “discoverServices” function, which requests the available services that the paired BLE device provides. If “kind” =4, that indicates that the device is responding with the services it provides. For diagnostic purposes, the program then PRINTs out the UUIDs of any services that the device has indicated it provides. It then calls the “discoverCharacteristics” function (for each discovered service), which queries the BLE device for the UUIDs of the characteristic(s) of the various service(s) it has found.

The callback function, “BLEServiceInfo” gets invoked by the above sequence of events. It returns the INTEGER variable “kind.” If “kind” equals 1 then the BLE device is returning characteristic information. For diagnostic purposes, the program PRINTs out the various characteristic UUIDs that have been reported.

All the UUIDs that are PRINTed out by the above routines can be checked against the UUIDs defined in the Creator IDE’s BLE component’s setup wizard for the project’s PSoC 6 firmware. This setup wizard was described earlier (Figure 2).

While the “BLEServiceInfo” function is returning characteristic information, the program checks for the following condition:

```
IF service.uuid = "1809" AND
characteristic(i).uuid = "2A1C"
```

When both of these conditions are met, we know that the “Health Thermometer” profile and its “temperature” characteristic have been found. The program then invokes the “peripheral.setNotify” function, which tells the PSoC6 to start transmitting ECG data.

As noted above, my PSoC6 program will be sending out ECG data as if it were temperature. I pack four, 8-bit ECG readings into each 4-byte packet (originally defined as the floating-point Temperature variable).

The last piece of the puzzle is handling the data that the BLE device is sending. This is done by the “BLECharacteristicInfo” function. This returns an INTEGER variable, “kind.”

- If “kind” equals 1, then we are receiving a description of the characteristic. For the Health Thermometer profile, the characteristic’s description would be “temperature.” I didn’t bother to PRINT this to the console.
- If “kind” equals 2, then we are receiving the actual data, in the 4-byte format described above. This is obtained using the “characteristic.value” function. I extract the four, 8-bit ECG values from the 4-byte “value” array. The Health Thermometer BLE packet is actually 5 bytes, but the first byte is the “Centigrade/Fahrenheit” flag, which I don’t use.

These are all the BLE-related functions. All that remains is to plot the ECG data to the graphics screen. The ECG array (ECGArray) comprises 400 ECG data points. When this array is full, I plot the data. Plotting is done by a high-level plotting routine:

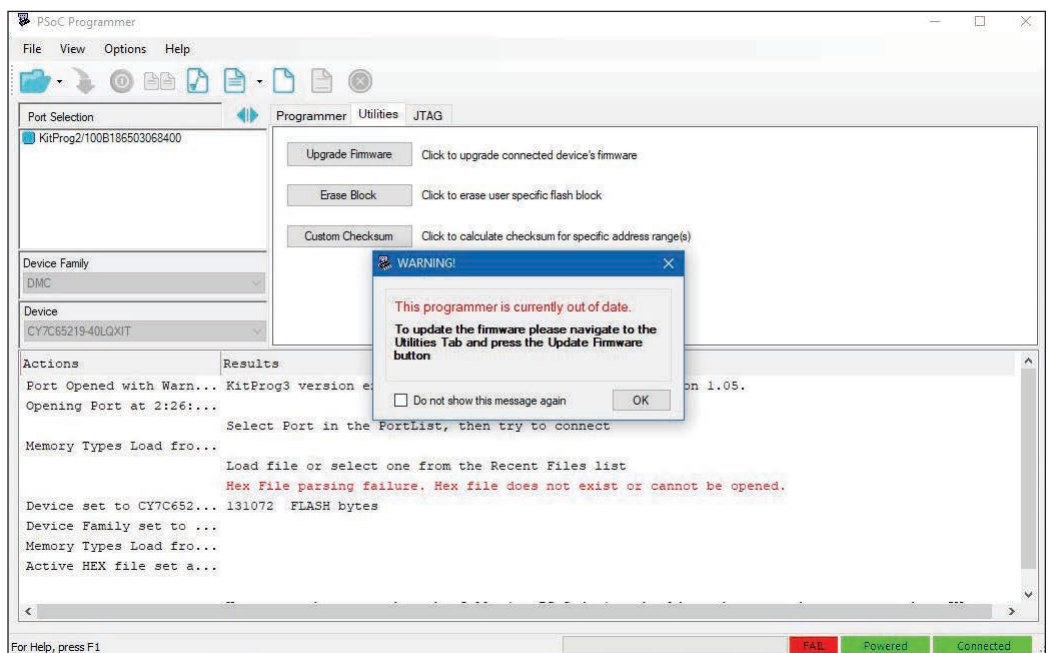


FIGURE 10

My CY8CPROTO-063-BLE arrived with the programmer containing KitProg2 firmware. This was out of date, as shown in the PSoC Programmer application (part of the Creator IDE install).


```
Px.setPoints(ECGArray)      REM Px
is Graphics.newPlot object
Graphics.repaint
```

To display the progress of the BLE discovery/pairing process at startup, I have placed a “Progress Bar” on the screen. As various BLE callback functions are executed, I indicate this by incrementally increasing the value of the Progress Bar, using the following function:

```
p.setValue()      REM “p” is the
Progress Bar’s name in the program
```

I have an Exit button on the screen. The callback function “touchUpInside” returns a variable, “ctrl” whenever a button (or other user-activated object) is pressed. If “ctrl” equals Button5 (the name I gave the Exit button), then the program performs a BLE.disconnect(PSoC6) to disconnect from the PSoC6. It then waits 5 s and issues a STOP to end the program. The 5 second wait allows the necessary BLE messages to be exchanged between the iPad and the PSoC6, before I end program execution with the STOP command.

Figure 4 shows my techBASIC program running. While developing the techBASIC software, I was initially unsure how quickly it could plot ECG data and concurrently handle all the BLE message traffic. What you see in Figure 4 is the resulting plot when I modified the PSoC6’s firmware to output a triangle wave—not an actual ECG wave. When I initially sent only one ECG data point per BLE packet, I observed significant data loss at this sample rate. After switching to 4 data points per BLE packet, things looked OK (Figure 4). **Figure 7** shows the app running with actual ECG data.

THE CIRCUITRY

Most of my recent projects have been powered by either Arm MCUs or Expressif’s ESP8266/32 Wi-Fi SoCs. In either case, the MCU comes in a small-footprint SMD package. Since I can’t mount these small devices on PCBs of my own design, I have been saved by the abundance of small, inexpensive MCU development boards offered by various manufacturers, and break-out boards for various peripheral chips. For this project, Cypress’ CY8CPROTO-063-BLE, containing the PSoC6/BLE RF sub-system and programmer, was ideal. Its \$20 price is much less than that of the individual components I would need to build it myself.

The ECG analog front end is not easy to design from scratch. Luckily, Sparkfun sells a small PCB containing an Analog Devices AD8232 device. This chip is specially designed for ECG and similar low-level biological signals. Sparkfun also sells the stick-on ECG electrodes in packs of 10, and a matching cable (Part numbers are shown on the **Figure 8** schematic).

Besides a LiPo battery and a few other components, these two modules are all that is required to implement the project. Figure 8 shows the schematic diagram of the project, and **Figure 9** is a photo of the unit in its case.

Note that I removed the “snap-off” programmer in the finished unit to save space. Without it, programming can still be accomplished by connecting a 5-wire cable between J5 on the PSoC6 board and J4 on the programmer. Note, however, that when you snap off the programmer board, the Rx and Tx signals from the PSoC6 module are no longer connected to the applicable pins on the programmer board. This means that you can’t

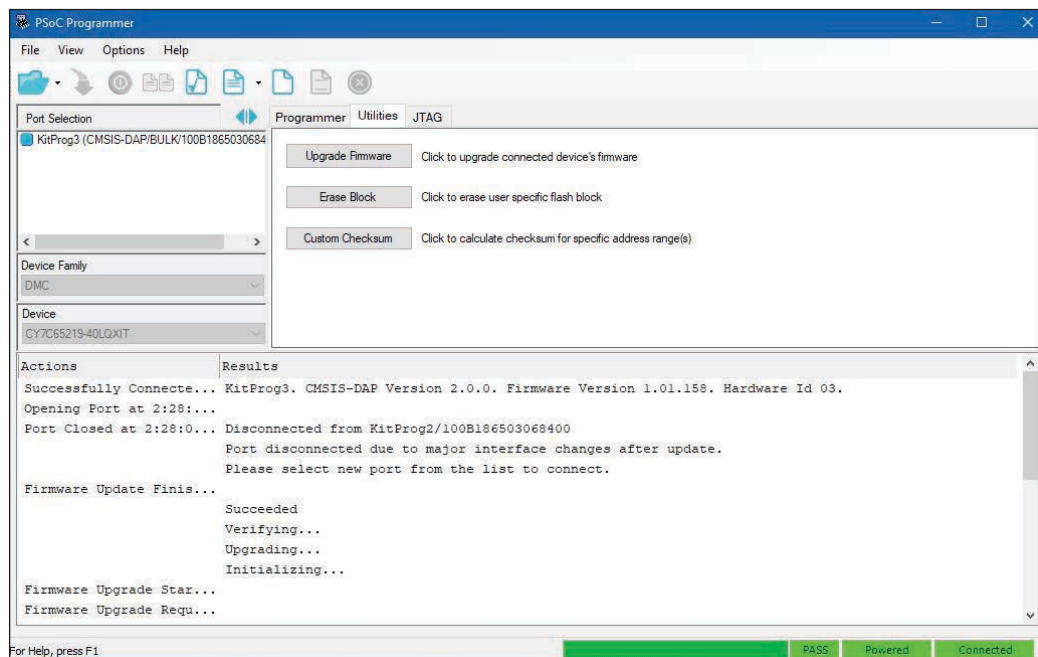


FIGURE 11

The PSoC Programmer screen looked like this when connected to the CY8CPROTO-063-BLE, after the KitProg2 firmware had been updated to KitProg3.

FIGURE 12

The CY5677 CySmart BLE 4.2 USB dongle. I didn't buy it until after developing the firmware for this project. I expected that the iOS CySmart app would handle debugging, but it turns out that the hardware USB dongle is much more versatile in this regard.



use the USB-serial port to send out Debug messages. However, you can run jumper wires between PSoC6's port 5.1 (Tx) and 5.0 (Rx) to the programmer's J6 pins 6 and 7, respectively. This restores the USB-serial connection.

I did all my program development with the KitProg2 programmer still attached to the PSoC6 board. In this case, power for the PSoC6 BLE board is provided by the KitProg2 programmer, which contains a 3.3 V regulator for both the programmer's PSoC 5LP and the PSoC6 on the target board.

I expected to be able to connect the 3.7 V LiPo battery to the V_{in} pin on the PSoC 6 target board. Nothing worked when I did this, and for a few minutes I panicked, thinking that the V_{in} pin required a regulated 3.3 V. This would have differed from the CY8CKIT-059 PSoC 5LP development board that I used for my last project, which contained its own 3.3 V regulator. In the CY8CPROTO-063-BLE user's guide, figure A-2 shows regulators on both the programmer board and the PSoC 6 target board. However, once I magnified the schematic, I noticed that the words "No load" became legible next to the U6 regulator. This explained why nothing was working. As a remedy, I installed an external Microchip Technology MCP1700-3302E low-dropout regulator to power both the PSoC6 board and the AD8232 Heart Monitor board. One could populate the PSoC 6 target board with a surface-mount regulator (U6 and associated parts) that it calls for, but I did not have a compatible regulator on hand.

When I first received the CY8CPROTO-063-BLE, it showed up immediately as a valid BLE device when I ran the CySmart app on my iPad. It ran the demo properly. However, it did not show up in the Creator IDE when I tried to edit sample code and download it to the board. This was unexpected, given that I had experienced no such problems with the CY8CKIT-059 PSoC 5LP development module during my last project.


I discovered that the Kitprog2 firmware on the CY8CPROTO-063-BLE development board (the same as what is contained on the CY8CKIT-059 PSoC 5LP board) requires upgrading to Kitprog3 firmware to work with PSoC6 devices. You must use the stand-alone Cypress PSoC Programmer v3.28 PC application to update the Kitprog2 firmware. The PSoC Programmer screen before and after I ran the update procedure is shown in **Figure 10** and **Figure 11**, respectively.

CONCLUSIONS

Even having finished this project, I still find BLE to be a complex protocol to handle. I'm certain that if 6 months passed before I used it again, I'd likely follow the same procedure I did here—modify a sample PSoC 6 program to serve my needs and then modify my techBASIC program to serve the new function. The only difference would be that I now have the CY5677 CySmart BLE 4.2 USB dongle on hand (**Figure 12**). This debugging tool is more versatile than the CySmart iOS app that I used for this project.

That said, I must give kudos to Cypress for the Creator IDE application. It does a lot to automate the process of incorporating a BLE function into your application—particularly if you can use one of their sample programs as a template for your firmware.

If you need to write a custom BLE app for an iPhone or iPad, and don't do this for a living, I think the techBASIC app is a smart option to explore. If you are accustomed to programming in Visual Basic, it doesn't take too long to get used to it. It also contains many library routines to handle most of the internal peripheral functions found on iOS devices (apart from those to which Apple doesn't permit access).

The PSoC 6 source code is available on the *Circuit Cellar* website, in what Cypress calls the "archive" format. It contains all the files needed to replicate the project. The techBASIC code is also provided as a text file. Because Apple restricts the loading of program code directly to an iOS device, you need a "trick" to do it. The techBASIC manual covers this in detail, but basically, you must email the source code to yourself, copy it to the clipboard, and then paste it into techBASIC. 

From the Bench

MQ Telemetry Transport (Part 2)

Bringing it All Back Home

In Part 1, Jeff described the MQTT protocol and how it is used by an MQTT server to keep all your IoT projects tied together and managed from a centralized server running a program such as Mosquitto on a local PC. He presented a simple project connecting two IoT nodes together via communication with the server. In Part 2, Jeff looks at modifying systems he uses to monitor his neighborhood well system and his weather station, for integration into the MQTT server.



By
Jeff Bachiochi

At a time when many companies are moving their storage off site, I am a believer in keeping it close. By that I mean under my own roof. As computing is becoming faster and more efficient, storage media is also growing in capacity. Today, gigabyte thumb drives and terabyte hard drives are becoming the norm. When I lose data because I made a mistake or a bad decision, I find it much easier to live with than if all my stuff were to suddenly disappear from the cloud. When things are out of my control, I feel I have no recourse. That's why I am attempting to bring all my needs back under my control.

Last month I discussed the MQTT (Message Queuing Telemetry Transport) protocol and introduced a project with two IoT clients. Client 1 monitors a digital input and "Publishes" the new state as a "topic" message to the server whenever its input changes state. Client 2 controls a light switch on its digital output and "Subscribes" to Client 1's topic on the server. Whenever a new topic is Published to the server, all nodes that have Subscribed to that topic get sent the updated message. Client 2 monitors the messages on this topic and sets/clears the

digital output based on the message. In this case, I defined Client 1's topic as "ESP_11E9B0\DIN\IO2\" and the message as an ASC "1" (ON) or "0" (OFF).

While not necessary, I also included the topic "ESP_A3D0DF\DOUT\IO2\" so that Client 2 could Publish the state of its output, and Client 1 could Subscribe to it, to see that the operation was actually completed. While this required two nodes to establish the remote control of a light, the real intention was to show that by shuffling communication through a server—as opposed to just having these two nodes talk to each other directly—the information could be recorded and stored for posterity. You may not care when a light was switched ON and OFF, but what if these were bank transactions or an alarm of some kind? Let's not forget the potential "bigger picture."

In the past, I've taken a number of different approaches on user interfaces (UIs). For example, I monitor my neighborhood's well system using ThingSpeak. I have also described a graphics server using HTML5 for my weather station. This month, we'll look into modifying these two projects for integration into the MQTT server.

WELL ENOUGH ALONE

My neighborhood's well supplies water for seven families. A deep well pump supplies water to a 360 gal. holding tank. We're fortunate that the water is potable from the pump, despite a high concentration of iron. Once the suspended iron is exposed to air, it oxidizes and gives the water a rusty color. This is visually unpleasant to drink, and creates rust stains on washed clothing. However, a pair of water softeners remove most of the iron before it gets to the main line that serves the neighborhood.

The softeners periodically require salt as a rinsing agent to rejuvenate (clean) the resin media that collect iron particles. Salt pellets must be added to an external tank, which contains a brine of water mixed with the salt. The brine is drawn by the softeners during their cleaning cycle, and the brine tank is refilled with water. A portion of the salt pellets dissolve during each cycle, until the water becomes saturated. Therefore, the salt lasts several cleaning cycles before it is used up. If the salt is not refilled, the resin is not cleaned and cannot remove any additional iron, giving everyone tinted water. Letting the salt disappear is a no-no, and should be avoided—hence the need for monitoring.

The well monitoring system has seven inputs. The first three are temperature sensors. I take the outside temperature and two inside temperatures—of the upper and lower well house. The well house is an 8' cube of concrete blocks below ground level. A 3' wall and roof above ground give access via ladder to the equipment. The upper temperature is in the roof area, and the lower temperature is below ground at floor level. It is interesting to see how wildly the outside and upper temperature vary, whereas the lower temperature hangs around 50°F until winter. With some winter days below zero, temperature becomes an issue. A frozen pipe means no water and the potential of a burst pipe, which has already happened once. Should the temperature fall too low at the lower sensor, a heater can be turned on. Currently, an incandescent light bulb throws enough heat to raise the temperature slightly.

The fourth and fifth sensors are attached to the water softeners. They are paddle-wheel flow sensors used by the softener's electronics to calculate—based on water usage—when the resin media should be cleaned. They produce a tick for each 6.4 oz. (20/gal.) of fluid. Monitoring the usage can bring to light a number of issues. For instance, when cleaning cycles are activated after midnight—so salt pellets can be added, or the relationship of water usage between softeners—monitoring can help to determine if other maintenance is required.

The sixth sensor measures the pressure in the storage tank. A regulator on the tank turns the pump on when the pressure falls below 40 psi, and turns the pump off when the pressure reaches 60 psi. This can help determine the pump's efficiency, since it does eventually get clogged with iron deposits.

The seventh sensor is a current probe around the pump lead that indicates when the pump cycles. For longest life, you want the pump to cycle as little as possible. Accordingly, you want to know if the pump starts running too long and isn't refilling the storage tank. The storage tank is actually pressurized with air above the water, which pushes the water out of the tank when a faucet is turned on. A storage tank with no bladder, or separator between the water in the lower part of the tank and the air in the upper part of the tank, will eventually absorb the air above the water. When this happens the air's volume at 60 psi will be less than before. This creates more room for water, but also decreases the volume of water that is available to leave the tank before the tank pressure drops to 40 psi. The pump will operate more often for shorter lengths of time—a bad thing. With that in mind, it's important to maximize draw-down by keeping adequate air in the tank. Most tanks have a bladder to prevent this from happening, but our tank has long since lost its bladder to old age.

MQTT MOSQUITTO

Last month, I began running Eclipse Mosquitto, an open source message broker, on my PC. It turns my PC into a MQTT server that runs in the background. I can reach it through my local LAN/WAN. It will accept MQTT Publish and Subscribe messages sent by any device on my network. All MQTT messages contain a topic and a payload. Messages are categorized by topics such as "Temperature," with an associated "payload" of some value. Publishers create the data, whereas subscribers consume the data. The MQTT server keeps track of the published topics and sends them to any device that has subscribed to that topic. To learn more about this, please refer to last month's Part 1 article (*Circuit Cellar* 351, October 2019).

Getting my well-house device to send MQTT data is pretty straightforward. This application has been using ThingSpeak since 2006. The Arduino library uses TCP over HTTP to communicate with the ThingSpeak server, whereas MQTT uses UDP over HTTP. TCP is connection-based and remains open until closed. UDP is connection-less—that is, each communication is complete in and of itself. In my application, I need to substitute

include WiFiUDP.h for include ThingSpeak.h. The Wi-Fi connection remains the same. In last month's article, we found that the MQTT communications begin with making contact with the MQTT server (at its IPAddress:1883) to establish a link between the two, but not an actual connection. This was discussed and the code was shown.

ROUND ROBIN

In this application, the sensors are read round-robin style—one every second. This keeps their values updated with the latest information. You can choose how often you want this information to be sent. Currently, I send it all once per minute. As noted in Part 1, when we want to share information with others, we Publish it to the MQTT server. The following line of code accomplishes this, once we've defined the topic and msg.

```
client.publish(topic, msg);
```

This next chunk of code is a special case

message I use to indicate that the device has just come out of reset. The variable reboot, which starts as true, gets cleared to false after the message has been sent once.

```
// reboot
if(reboot)
{
    t=" reboot";
    t.toCharArray(msg, msgSize);
    t = ID;
    t.toCharArray(topic, msgSize);
    //Serial.print(t);
    sendMQTT();
    reboot = false;
}
```

Note that the msg is set to the string reboot and the topic to the "ID" of the device—in this case ID = "esp8266_A14782", which is its friendly name (including the last 6 characters of the MAC).

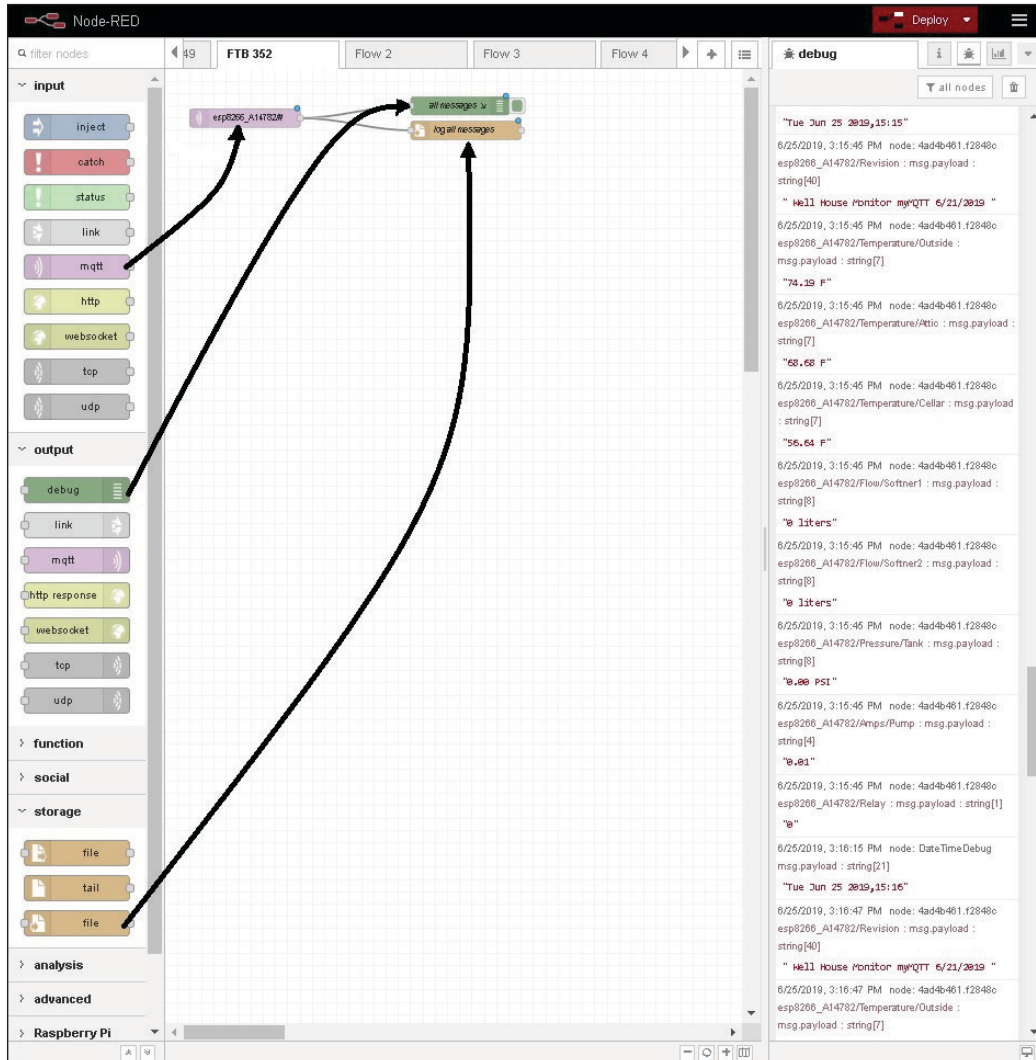


FIGURE 1 Three nodes are used here: "mqtt," to subscribe to all messages from esp-A14762; "debug," to view the messages in the debug window; and "file" to log the messages.

Here's are the messages this device sends:

Topic, Message

ID + "/Revision", "Well House Monitor myMQTT 6/21/2019 "

ID + "/Temperature/Outside", 78

ID + "/Temperature/Attic", 72

ID + "/Temperature/Cellar", 56

ID + "/Flow/Softner1", 27

ID + "/Flow/Softner2", 20

ID + "/Pressure/Tank", 45

ID + "/Amps/Pump", 0

ID + "/Relay", 0

I always include the ID as the beginning of the topic, so that I can identify its origin. Last month, the project used two devices. The 1st published a switch's status, and the 2nd subscribed to that topic and controlled a light based on switch status. Mosquitto handles all of this without intervention.

Node-RED is a companion application that allows you visually to connect devices. In Part 1, I detailed how it can be used to subscribe to an MQTT topic and do something with the messages received. In that case, all messages were sent to a log file. Other than viewing the file's data, there was no visual indication of any device activity. Now let's see how Node-RED can be used to display the data from the well house.

Node-RED EDITOR

Node-RED provides a browser-based editor for wiring together hardware devices.

The node-RED editor is available through your browser at IPAddress:1880. The editor window consists of four components: palette (left), sidebar (right), header (top) and workspace (middle). The palette contains the nodes (objects with possible inputs and/or outputs) available for placement on the workspace. The sidebar contains information about objects to be displayed. The header has a button for deploying a flow (interconnection of objects) and additional menu items. The workspace is where you drag and drop objects, and interconnect them using wires to tie inputs to outputs. **Figure 1** shows my browser with three nodes placed, wired and deployed. The sidebar shows the debug output from the debug node as messages are received.

Once you become familiar with all the nodes in the palette, you'll see that subscribing to an MQTT topic (MQTT input) is just one way of using Node-RED. This project will create a "dashboard" consisting of mainly graphs of the sensor data over time. Let's look at creating text, gauge and chart displays for one temperature message. Refer to **Figure 2** for this discussion.

Our tab FTB 352 shows four nodes added to the workspace. The first node (MQTT input) passes only the message payloads with a topic containing "esp8266_A14782/Temperature/Outside" as defined by editing the node. The next three nodes are from the dashboard in the node's palette. They are used to display the message payload in text, as a value on a gauge and as a charted

The screenshot shows the Node-RED editor interface. The workspace contains a flow with the following nodes: an MQTT input node (topic: esp8266_A14782/Temperature/Outside), a debug node, and three dashboard nodes: Text, Gauge, and Chart. The dashboard is titled 'FTB 352' and displays the following information:

- Gauge:** Outside Temperature, 74.19 (Degrees)
- Text:** Outside Temperature 74.19 F
- Chart:** Outside Temperature (Line graph showing temperature over time from 16:04:00 to 16:17:00)

FIGURE 2

Four nodes are added for this example, "mqtt", to subscribe to only outside temperatures messages from esp-A14762; "text," to display the message payload as text; "gauge," to display the message payload as a meter; and "chart," to display the message payload as a graph over time. The actual dashboard screen is shown as an inset, using another browser.

value over time. Because the payloads are all strings, the test display includes the payload just as it comes. You may have noticed an “F” (Fahrenheit) was part of this. In the gauge and chart displays, convert the string to a value first, then use the value for displaying.

Some parameters—such as max and min—can be edited in these nodes. Since I have three temperatures, all of them can be wired to the same chart display. With a small bit of massaging we can remove (change node) part of the topic that is used in the chart’s key for identifying the different data lines (**Figure 3**). The change node is used to set, change, delete or move specific parts of a message, flow or global object. Here specific message topics are truncated. In other words, `esp8266_A14782/Temperature/Outside` becomes `Outside`.

The water softeners measure water throughput by ticks (counts) of their paddle wheels. Total counts are sent as messages to be used as data for the chart. Before sending the count data to the chart, I use a function node to alter this. All nodes pass their message to their output, which is then received by the next “wired” node’s input. The function node allows some JavaScript to be written that can alter this message in some way. Here, I want to convert counts to gallons. The water softener documentation states that the paddle wheel creates 20 counts for each gallon of water that passes through it.

```
msg.payload = (parseInt(msg.payload) / 20);
return msg;
```

In a perfect world, the two softeners’ counts would share the flow equally, and their charts would be identical. If displayed on a single chart, the first softener’s data would always be obscured by the second. I chose to display these as separate charts. However, I also want to keep a running daily tally of the total gallons per day. These data aren’t given by the well-house node. Node-RED allows me to total the data and reset the value to zero at the end of each day. Take a look at the following JavaScript code:

```
var myFlow = flow.get('Flow1') || 0;
myFlow = myFlow + msg.payload;
flow.set('Flow1',myFlow);
msg.payload = myFlow.toFixed(3);
return msg;
```

I create a local variable `myFlow` (available to just this node), and assign it the value of a flow variable `Flow1` (available from any node within this flow or page on my workspace). The saved value is added to the present message’s value. The total is stored away for next time. Finally, the message sent out is replaced by this new value—fixed at three decimal places. This value updates a text box with Today’s total, but we still need a way to zero out this total every night and chart the daily totals.

This is created with three nodes: “inject”, “function” and “chart.” The “inject” node allows something to happen at a specific time or periodically. I’ve chosen once every day at 11:59 p.m.

```
var myFlow = flow.get('Flow1') || 0;
var msg1 = {payload:myFlow.toFixed(3)};
msg1.topic = "DailyFlow1";
```

```
flow.set('Flow1',0);
return [msg1];
```

Another “function” node is used to do this. I create a local variable `myFlow` (available to this node only) and assign it the value of a flow variable `Flow1` (available thanks to the node described above). Now I create a whole new message with a payload of `myFlow` and a topic of `DailyFlow1`. The `Flow1` variable is then cleared so it can begin with a new total, Tomorrow, which begins in 1 minute. The new message is sent on to the chart node. See the final display “Flow” in **Figure 4** and “Dashboard” in **Figure 5**.

WEATHER REPORT

I’ve had a weather station in various forms prior to the smartphone era, informing me of tomorrow’s weather today. While I’m not a HTML guy, one of the most interesting articles for me was my weather server project that handled its own HTML display. The two part article is “Serving Up HTML (Parts 1 and 2) in June and July 2016 (*Circuit Cellar* 311 and 312). In keeping with the MQTT theme here, I’ve re-coded that project and have Node-RED code for a new weather display. One of the advantages of using Node-RED for a project is its extensive palette of the available nodes. For instance, the “inject” node lets you simulate the reception of messages as if your external device had Published data.

Here, I use the “switch” node to separate each topic into its own path, which is wired directly to a gauge or text output. Each gauge has three ranges associated with it. These ranges can be color coded, and will display that color depending on the color range settings and the actual payload being

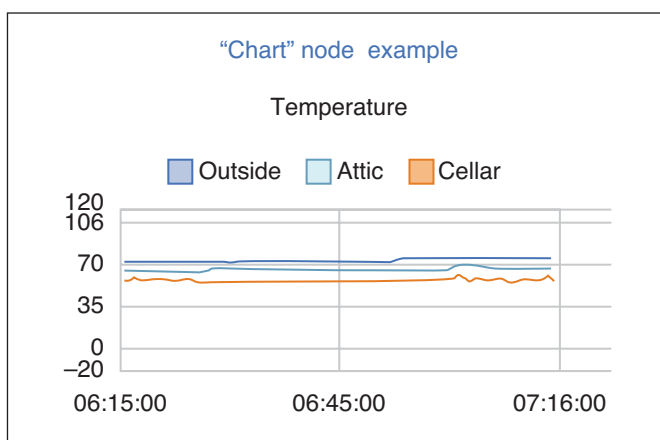


FIGURE 3

The “chart” node can display multiple data graphs with or without a key (labels).

Additional materials from the author are available at:
www.circuitcellar.com/article-materials

RESOURCES

Espressif Systems | www.espressif.com

Mosquitto | www.mosquitto.org

MQTT | www.mqtt.org

Node-RED | www.nodered.org

openHAB | www.openhab.org

ThingSpeak | www.thingspeak.com

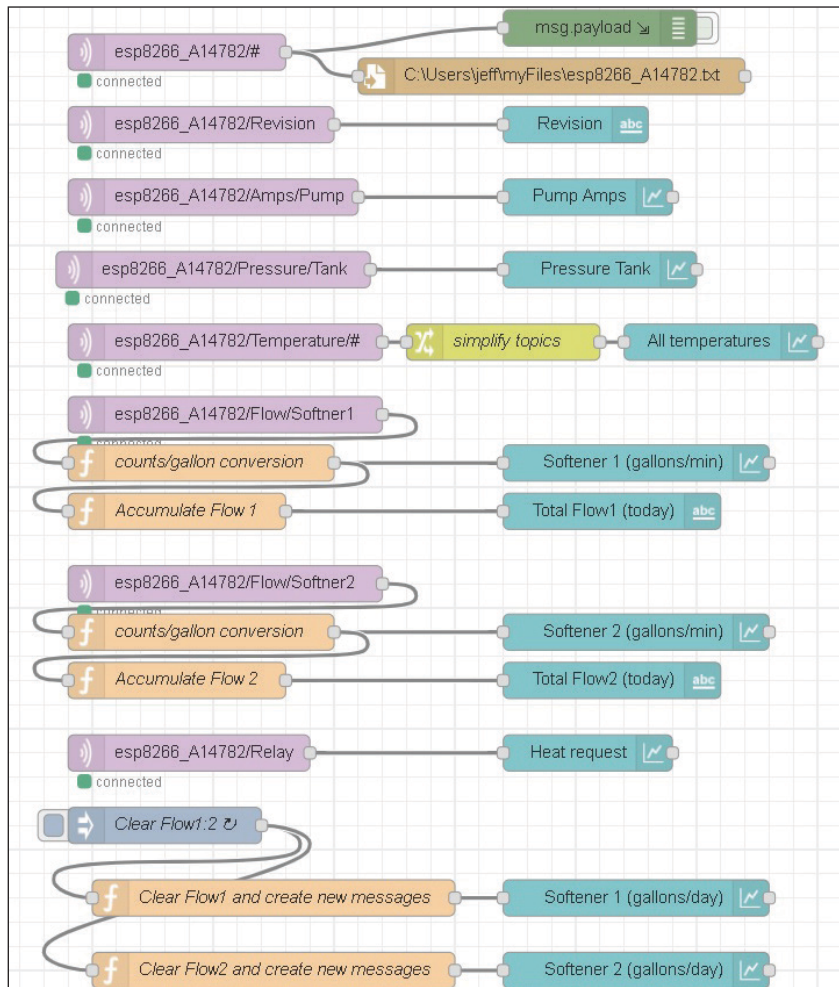


FIGURE 4
The complete flow for logging and displaying the dashboard of the well-house monitor.

displayed. If the range colors are defined but range settings are not, then the color displayed will be a blend of the defined colors proportional to the actual payload. If max and min values are not defined, then the gauge will auto-range based on the payload values.

You might want to separate real-time data (gauges), as in **Figure 6**, from statistical data (text) on separate flows. Each flow will be offered as tabs on the display page of Node-RED. Clicking on the menu icon on the left side of the blue bar at the top of the UI display page reveals additional “flows” that you have created (**Figure 7**). You can reach every flow from this one page! The weather station provides plenty of statistical data. It has hourly, weekly, monthly and/or yearly averages for almost every sensor. These might all be displayed from an alternate flow.

SUNNY DAYS

At this time, I have neither solar panels covering my roof, nor a huge solar matrix in the yard. I have a few small panels affixed to my shed roof so I can dabble. But so far I’m not overwhelmed by their energy production. They receive no direct sun in the mornings and late afternoons, so their production is limited. This produces just about enough energy to keep the weather station and solar reporting nodes alive 24/7. The MQTT server also handles the solar node.

The solar charge controller from Epsolar Technology has an RS-485 (ModBus) interface to access its more than 100 registers. These are separated into logical sections: Rated Datum, Real-time Datum, Real-time Status,

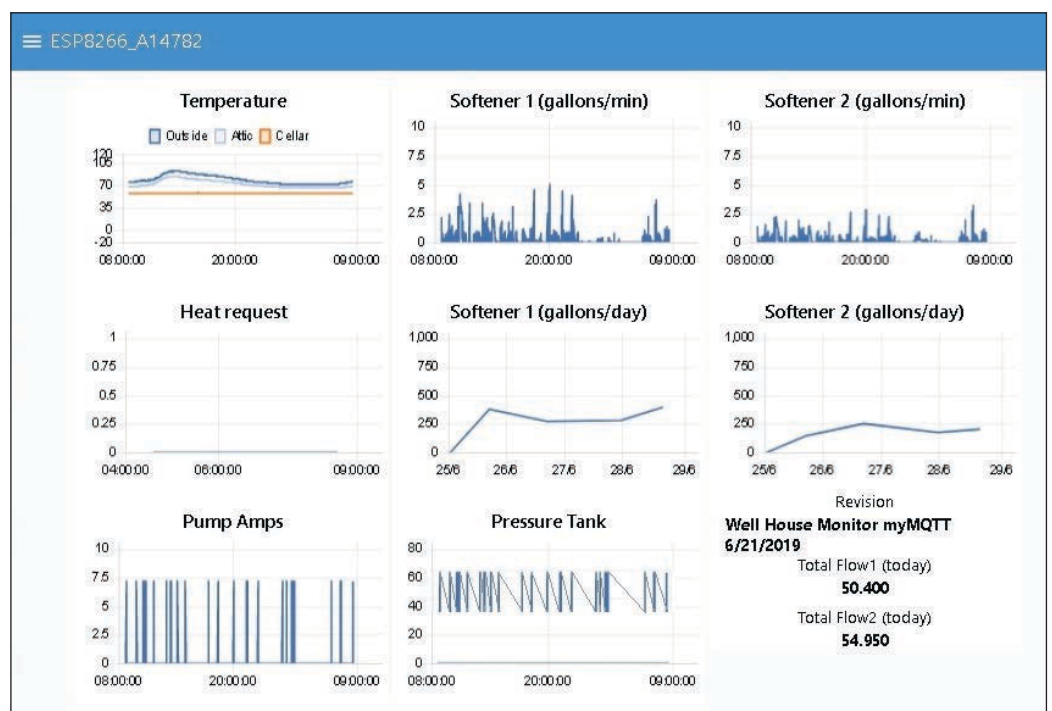


FIGURE 5
The dashboard of the well house monitor is available through a browser directed to the MQTT server at IPAddress:1880/ui. {8 JEFF ,SHOULD THIS BE 1880 OR 1883?}

Statistical Parameters, Holding Registers, Switch Value and Discrete Value. My solar monitor circuitry must poll the charge controller to gather copies of all the registers and publish them to the MQTT server. While the server collects all the registers, I only use a fraction of these to give a real-time display of energy produced, stored and consumed.

The upper-level folder name for all topics from the solar controller node use `esp_82952A` to identify the device. This topic is appended with the section name and register name as additional folders to give a hierarchy to the topics published. In this Node-RED flow, I subscribe to all `esp_82952A#` topics. A switch node is used to filter all eight topics by section name. The flow is therefore divided into eight sub-flows. Don't confuse my term "sub-flow" with Node-RED's sub-flow. More on this later. Each of these could be sub divided again with eight additional switch nodes, to produce a separate sub-flow for each register in the charge controller. To illustrate this, I am using three of these eight sections, and dividing the three sections into their respective individual registers with switch nodes.

The Real-time Datum section has sub-flows for 10 of the 15 registers. The Statistical Parameters section has sub-flows for seven of the 21 registers. The Discrete Inputs section has sub-flows for both of its two registers. As you can see from the Node-RED editor's



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.

workspace in **Figure 8**, the diagram is getting quite complicated. For ease of viewing, you can designate portions of the flow to sub-flows, and move them onto separate pages (workspaces), which can then be identified on the flow's workspace as separate sub-flows, thus simplifying the workspace. However, by doing that you lose your ability to debug each sub-flow because they no longer produce output in the debug window. For that reason, I have not used sub-flows in any of my flows.

My use of the term "sub-flow" has to do with dividing each `esp_82952A#` topic (they all come in through the same input wire) to the switch node labeled `esp_82952A#` and are separated into section topic outputs (sub-flows). Each of these is then input into

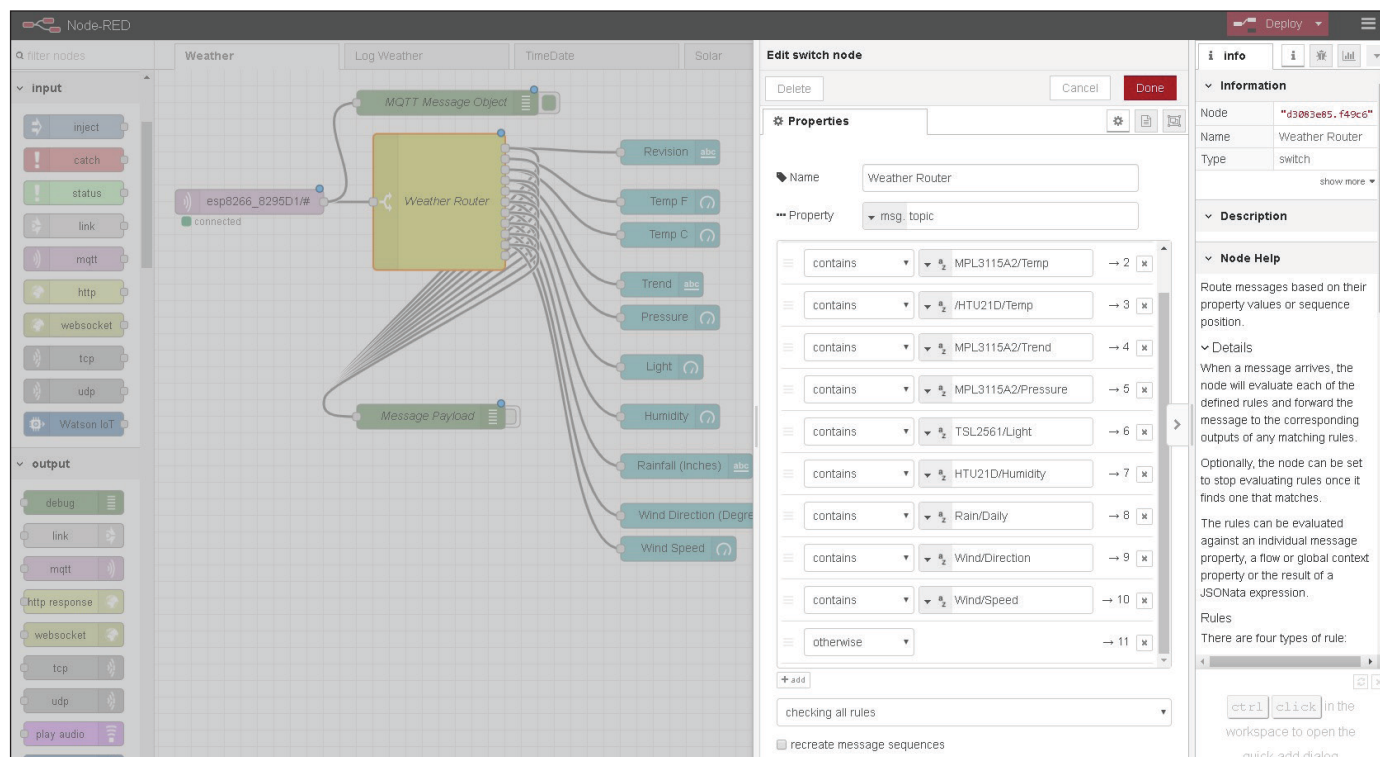


FIGURE 6 The "switch" node can be used to separate a stream of messages into specific topics, appropriate for displaying each payload in its own user-selected way. Most of the weather data are real-time and displayed as meters (gauge). The revision, barometer trend and accumulated rainfall will be displayed as text.

COLUMNS

additional switch nodes to further separate the topic into individual outputs (sub-flows) containing just one register

The display of data here is a combination of text, charts and LEDs. You'll notice there is no LED node in the palette dashboard section. Another great aspect of Node-RED is the palette manager, which allows you to

import nodes designed by third parties or create your own. In this case, I imported the node-red-contrib-ui-led node to use in this flow. Although an LED is considered a binary device, its color can be set based on different payloads, binary, other numerical value, string, JSON or buffer. This is similar to most other nodes. For instance, I use yellow

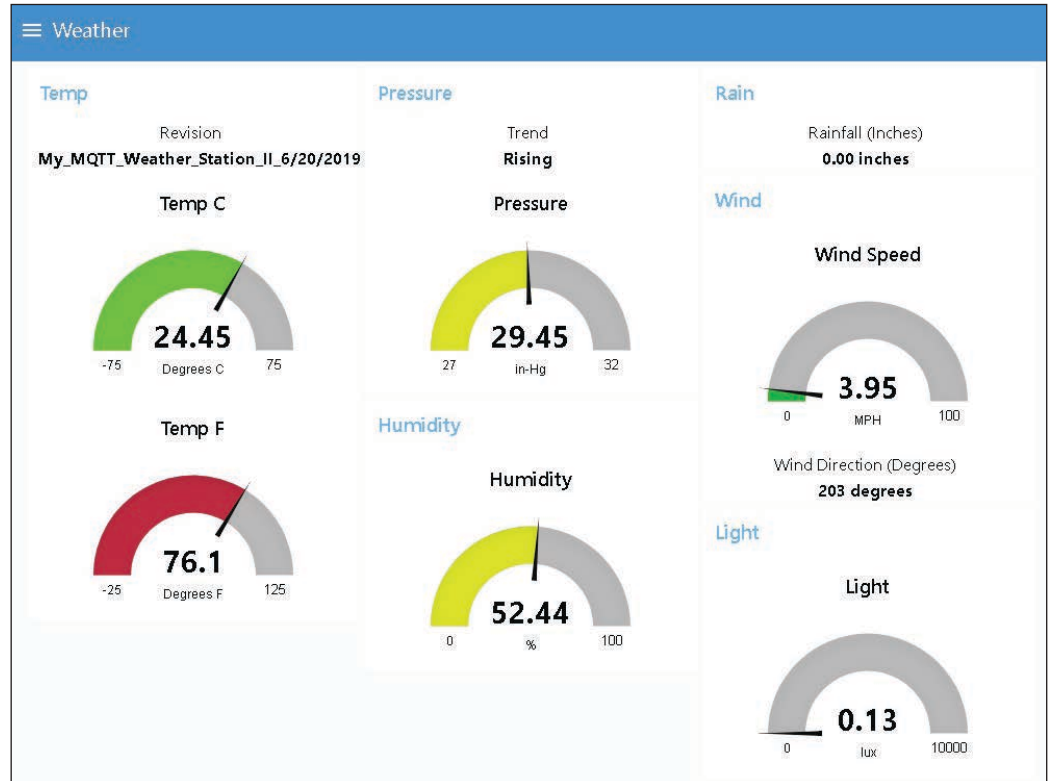


FIGURE 7 The weather dashboard is a real-time display of weather station data. When I want to alter the program of an IoT device, it's always a struggle to locate the latest copy. I found that displaying the program name (revision) helps to steer me to the right one.

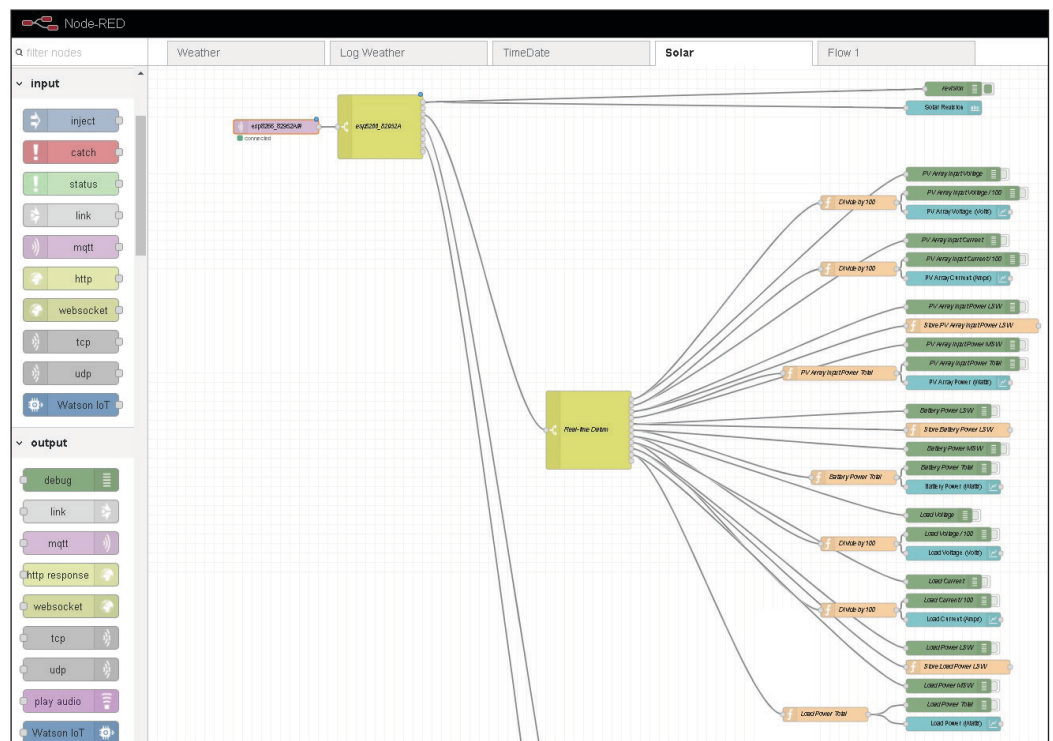


FIGURE 8 The "switch" node is helpful for steering groups of messages. Multiple switch nodes are used in this Solar flow, to break groups down to the individual message topics.

COLUMNS




FIGURE 9
The Solar dashboard is one of many that are needed to show all the registers available, but this might be the most informative.

for daytime and black for night time. In this case, day/night is a Boolean value, but it could have been the strings yellow or black.

I was caught by surprise by the data I was seeing from the Battery Current register, shown in **Figure 9**. If I had thought about this a bit more it would have been obvious! The PV Array produces energy. The load consumes energy. The battery both produces and consumes energy. When it consumes energy, it is charging (current is positive). However, it also produces energy (current is negative) once the PV array is no longer producing. Therefore, the Battery chart must show current in both directions. I wanted this to be obvious, by using a green trace when zero or positive and red when negative. I did this by dividing this sub-flow (esp8266_82952A/Statistical Parameters/Battery Current LSW) into two sub-flows. Sub-flow topic positive, when the payload is ≥ 0 , else sub-flow topic negative. Both of these sub flows go to the same chart input. The chart plots both as separate topics, positive as green and negative as red. The chart key is disabled, so the topics are not defined in the chart.

HOSTING MQTT

Right from the get-go, I used my PC as the MQTT host. Just recently, I installed openHAB 2 on a Raspberry Pi. This is an open source, home-automation platform, which runs as the center of a Smart Home. My thinking is that openHAB 2 supports MQTT (and Node-RED) which I could use as a permanent location for my MQTT server. The weather and solar nodes are operating on the Pi (running Linux OS). Although I have not played with openHAB 2 at all, the MQTT link allows me to find a path that supports my work in the present, while lighting my path into the future. I'll be changing the subject next month, but if you are interested in this, please let me know so I can make plans for a future article.

Oh yeah, one more note on using Node-RED and MQTT that was installed with openHAB 2 on the Raspberry Pi: The Pi palette has some additional nodes listed (**Figure 10**). There are two addition palette sections: "Raspberry Pi" and "Home Automation." One gives access to the Pi's I/Os, and the other opens up communications to openHAB 2 giving my sensors life in the future. Too little time, too much to do! 

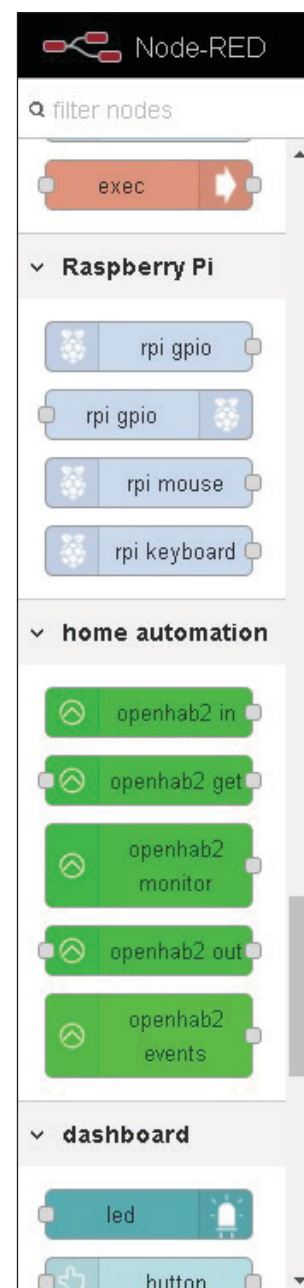


FIGURE 10
I'm excited about these new nodes available in Node-RED, when using the Raspberry Pi as the MQTT server. I plan to use some of the Pi's I/O for displaying some LED status of the operational activity. Adding my well, weather and solar data to a Home Control System like openHAB will really tie everything together quite nicely.

The Consummate Engineer

Semiconductor Fundamentals (Part 3) Transistor Topologies

In Part 2, George discussed devices built with one P-N junction, appropriately named diodes. In this article, he considers devices with more junctions. He starts with two and looks at the ubiquitous, three-terminal bipolar junction transistor (BJT). George looks at the math, science and circuitry of these devices.



By
George Novacek

Last month we dived deeper into diodes. Now it's time to consider devices with more junctions. Let's start with the three-terminal bipolar junction transistor (BJT). Its terminals are called collector (C), emitter (E) and base (B). **Figure 1** explains its construction, which is essentially a combination of two junction diodes. Based on their connections, we have two basic structures: PNP (Figure 1a) and NPN (Figure 1b), with the transistors designated accordingly. PNP and NPN devices with similar characteristics are called complementary pairs. In the early days, germanium (Ge) was used as the base semiconductor material and although germanium NPN transistors existed, the PNP variety ruled. The first NPN transistor I got to use was the silicon (Si) type.

The first point-contact transistor was the result of the work of American physicists John Bardeen, Walter Brattain and William Shockley in 1947—an achievement for which they received the Nobel Prize in Physics. As it usually happens with many inventions, the transistor was developed independently in Europe in 1948 by Germans Herbert Mataré and Heinrich Welker. The bipolar junction transistor was then developed and patented by William Shockley in 1950 at Bell Labs.

Germanium transistors manufactured in the '50s by the diffusion method had many growing pains. One was a poor frequency response due to large capacitance of the diffused electrodes. But the manufacturing technology improved and by the '60s we had ultra-high frequency (UHF) devices, for example the amazing (for their time) AF139 and AF239 PNP transistors. Germanium devices suffered from high leakage, relatively low operating voltage and, above all, significant temperature dependency. But all that changed with the arrival of the silicon planar NPN transistor.

One issue you have to keep in mind is that transistors are current amplifiers. They're not voltage amplifiers like vacuum tubes before them and field effect transistors (FET) today. BJTs operate in distinct modes. The first is linear, where the collector current $I_C = \beta \times I_B$. β stands for the transistor's current gain—generally greater than 100 in modern devices.

The second mode of operation is saturation, which is used in digital circuitry. The transistor becomes a switch. It is fully turned on with the saturated collector current determined by its type and collector-emitter voltage V_{CE} close to zero. Or it is turned off (cut off) with $I_B = 0$. The collector current I_C would ideally be zero too, but there is always some leakage.

THREE AMPLIFIER CONFIGURATIONS

Let's consider the three fundamental transistor amplifier circuit configurations as seen in **Figure 2**. Because of the present-day prevalence of NPN transistors, I shall use them in my examples whenever possible. We begin with the common base topology (Figure 2a). The base is grounded and, therefore, common to both the input and output. To amplify an AC signal, you'll need to bias the base to overcome the base-emitter diode's forward voltage—around 0.65 V for silicon transistors—and cause a base current I_B to flow. The current gain of the common base topology is less than 1 because the collector current I_C flows through the emitter as well. The input impedance of the common base topology is very low. The voltage gain, however, is high—provided the load resistance R_L is also high. It is defined by Equation 1:

$$A_v = \frac{V_{out}}{V_{in}} = \frac{I_C \times R_L}{I_E \times R_{IN}} \quad [1]$$

Common base configuration is used primarily in radio frequency (RF) circuits

because it minimizes frequency-limiting collector-base capacitance. Common (grounded) emitter configuration is commonly used in amplifiers as well as switching circuits because it has the highest power gain. Similar to common base, the input impedance is somewhat low, but can be increased, at the cost of gain, by a small resistor between the emitter and ground. Here, the emitter current $I_E = I_C + I_B$. The ratio I_C/I_E is called α and is always less than one. The relationship of the transistor currents can be expressed mathematically as:

$$\alpha = \frac{I_C}{I_E} \quad \beta = \frac{I_C}{I_B} \quad \dots \quad I_C = \alpha \times I_E = \beta \times I_B \quad [2]$$

$$\alpha = \frac{\beta}{\beta + 1} \quad \beta = \frac{\alpha}{\alpha - 1} \quad \dots \quad I_E = I_C + I_B \quad [3]$$

Common collector topology is better known as emitter follower. It is frequently used for transformation of high impedance input signals to low impedance output. The current gain equals approximately β of the

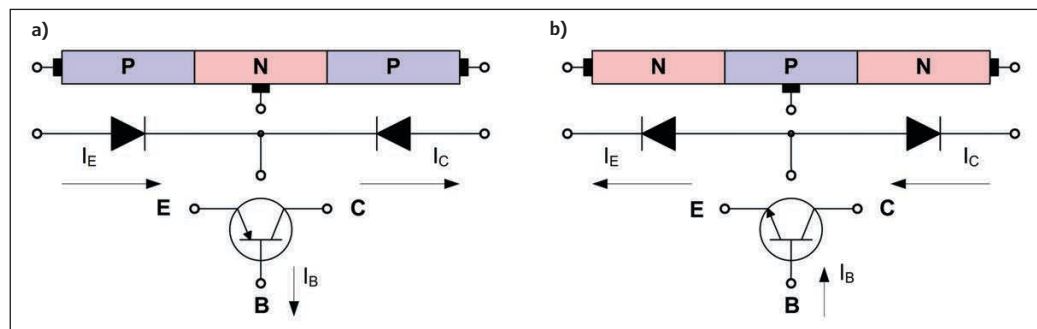


FIGURE 1 Principle of the bipolar junction transistor. PNP (a) and NPN (b)

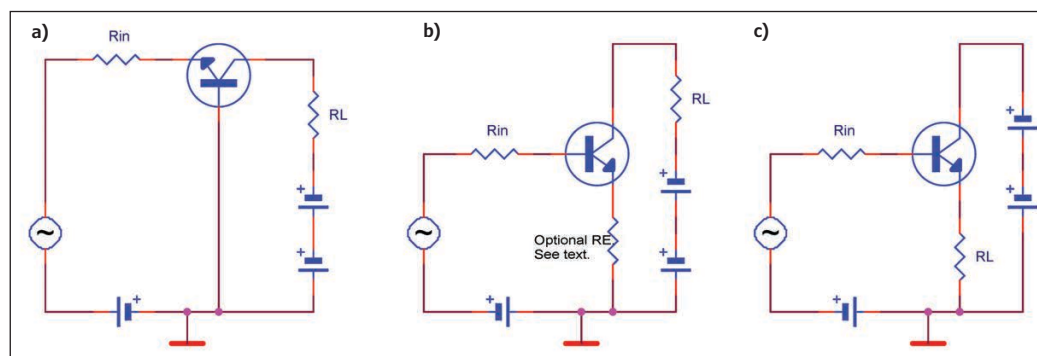


FIGURE 2 Three fundamental configurations of a transistor amplifier: Common base (a), common emitter (b) and common collector (c)

	Common Base	Common Emitter	Common Collector
Input Impedance	Low	Medium	High
Output Impedance	Very High	High	Low
Phase Shift	0 degrees	180 degrees	0 degrees
Voltage Gain	High	Medium	Low
Current Gain	Low	Medium	High
Power Gain	Low	Very High	Medium

TABLE 1 Summary of characteristics of transistor amplifier topologies

COLUMNS

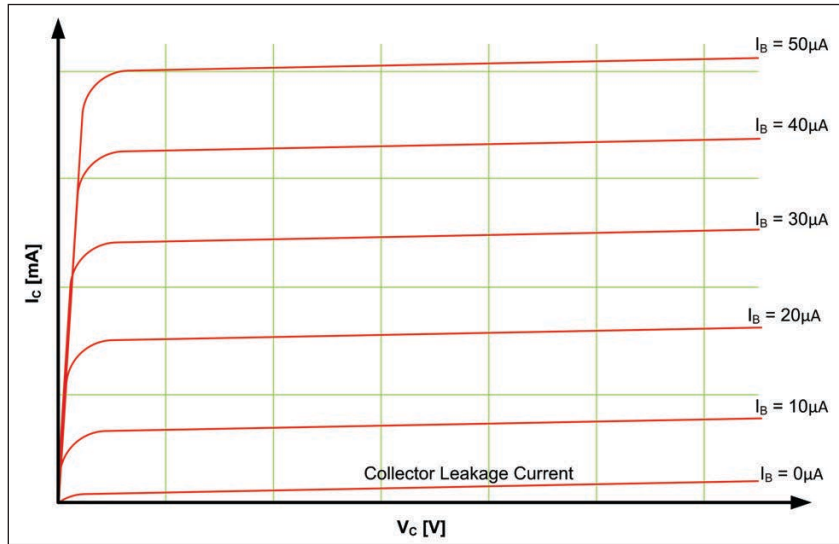


FIGURE 3
Collector I-V characteristic of an NPN transistor. The graph is not to scale.

transistor and the input resistance, as a rule of thumb, $\beta \times R_L$. The gains are expressed mathematically as:

$$I_E = I_C + I_B \dots A_T = \frac{I_E}{I_B} = \frac{I_C + I_B}{I_B} = \frac{I_C}{I_B} + 1 = \beta + 1 \quad [4]$$

Characteristics of the three transistor topologies are summarized in **Table 1**. Sometime in the past, the common emitter DC gain symbol β was replaced with h_{FE} . This is the parameter you will find nowadays in transistor specification sheets. It is an abbreviation that stands for “hybrid parameter forward current gain, common emitter.” **Figure 3** is an example of a common emitter, NPN transistor collector’s I-V (current-voltage) characteristic where the base current I_B is a parameter. The I-V characteristic of the base current versus base voltage is that of a diode as presented in Part 2 of this article series (*Circuit Cellar* 351, October 2019).

Notice that the collector current I_C dependency on the collector voltage V_C is quite large for small collector voltages. Once the collector current saturation is reached the current changes very little. You can analyze transistor amplifiers’ low frequency response by utilizing the transistor’s equivalent circuit. More often than not parasitic characteristics are considered negligible for the given application and, therefore, ignored. **Figure 4a** is the common emitter equivalent circuit. For practical reasons it is often converted into a so-called T-circuit equivalent **Figure 4b**.

For the purpose of electrical analysis, you can consider the transistor to be a black box—a four terminal linear network as is shown in **Figure 5**. A transistor is a three-terminal device, but one terminal, the emitter for the common emitter configuration is, obviously, common. You can analyze this network under different conditions, each rendering a different set of parameters. With an open circuit, for example, impedance z-parameters will result. Short circuit conditions will produce admittance y-parameters. But, because transistors in common emitter connections have low input and high output impedances, it is advantageous to use hybrid parameters, called h-parameters. But this is not the end of it! There are also parallel-series m-parameters, cascade-forward a-parameters and cascade-backwards b-parameters.

SELECTING PARAMETERS

So which parameters do you select and what can you do with them? For once, knowing one set of the parameters, you can convert them mathematically to any other set with just a small error. Then, inserting them into a matrix as shown by Equation (5), the performance data of the black box can be calculated. Equation (5) uses the h-parameters:

$$\begin{bmatrix} V_{in} \\ i_{out} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \times \begin{bmatrix} i_{in} \\ V_{out} \end{bmatrix} \quad [5]$$

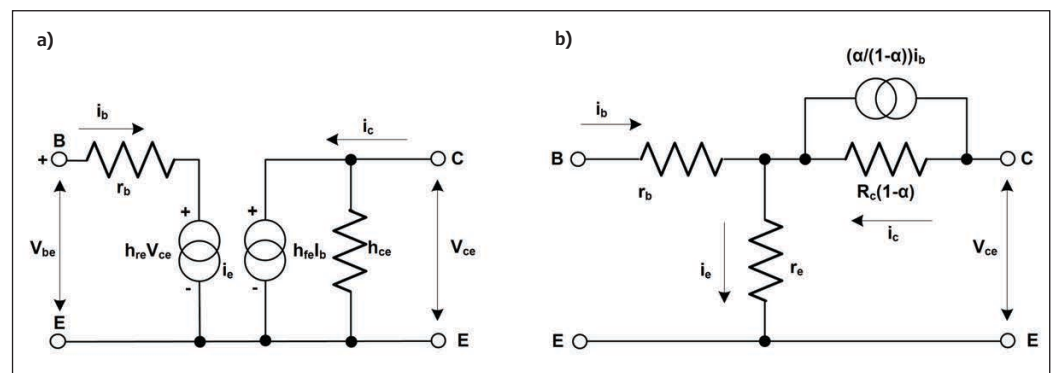


FIGURE 4
Transistor equivalent circuit in common-emitter configuration

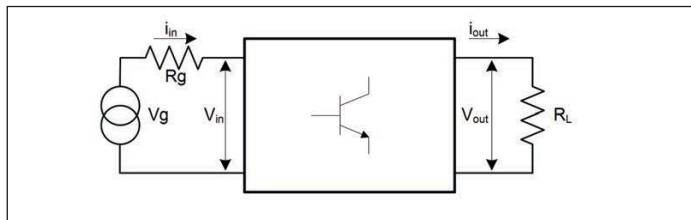



FIGURE 5
Transistor amplifier four terminal linear network

All this theory is very interesting, but unless you have access to a curve tracer and measure those parameters yourself, you're out of luck. I've not found the black box parameters in any specification published by a transistor manufacturer.

But that's not a showstopper. Present day manufacturers provide a number of graphs with their flagship devices, allowing you to design any transistor circuit you can imagine. But you can also find many inexpensive transistors—on e-Bay for instance—whose specifications, if you're lucky to get any, may provide you with perhaps only the following (and nothing else): maximum voltage and current ratings, perhaps the pin-out and maybe the h_{FE} . And yet, even that isn't a showstopper either. You can still build a circuit satisfying simple requirements, especially for low frequency operation.

Relying on feedback, you can set the DC operating point even without knowing the accurate h_{FE} . Quite often it is safe to assume the h_{FE} will be greater than 100. In such a case, a single stage, common emitter amplifier with a small emitter resistor R_E will provide voltage gain of approximately R_L/R_E —where R_L is the load resistance, comprising the collector resistor in parallel with whatever the additional load may be. For relatively slow switching digital circuits, the design is even simpler.

Next month, we continue this article series. In Part 4, I'll show you some useful discrete transistor circuits and then we'll zero in on the field effect transistors: Junction FETs and MOS. 



ABOUT THE AUTHOR

George Novacek was a retired president of an aerospace company. He was a professional engineer with degrees in Automation and Cybernetics. George's dissertation project was a design of a portable ECG (electrocardiograph) with wireless interface. George has contributed articles to *Circuit Cellar* since 1999, penning more than 120 articles over the years.

For detailed article references and additional resources go to: www.circuitcellar.com/article-materials

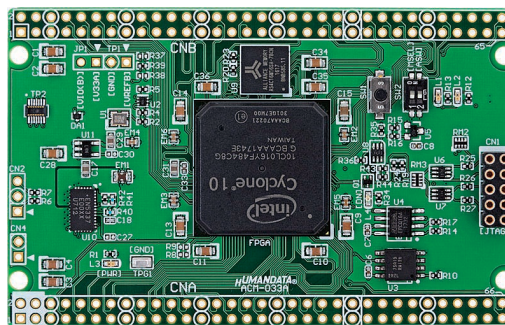
FPGA Boards from HUMANDATA®

SAVING COST=TIME with readily available FPGA boards

- Basic and simple features, single power supply operation
- Free download technical documents before purchasing

INTEL ACM-033

Intel Cyclone 10 LP F484 FPGA board



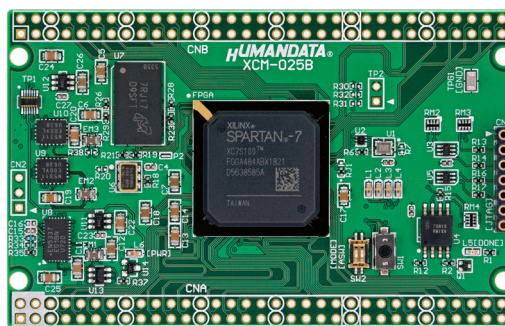
Cyclone 10 LP SDRAM SPI-Flash RoHS

SIZE : 3.386" x 2.126" (86 x 54 mm)

ACM-033 is an FPGA board with Intel high-performance FPGA Cyclone 10 LP. It's compact and very simple. 3.3V single power supply operation.

XILINX XCM-025

Xilinx Spartan-7 FGA484 FPGA board



Spartan-7 DDR3 RoHS

SIZE : 3.386" x 2.126" (86 x 54 mm)

XCM-025 is an FPGA board with Xilinx high-performance FPGA Spartan-7. It's compact and very simple. 3.3V single power supply operation.

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

www2.hdl.co.jp/CC19B

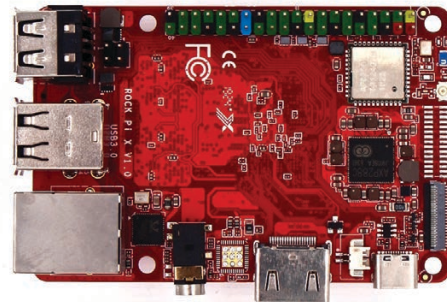


PRODUCT NEWS

Raspberry Pi Clone Sports 1.84 GHz Intel Cherry Trail Processor

Radxa has posted specs for a new member of its community backed “Rock Pi” Raspberry Pi lookalike SBC family, this time with an Intel Cherry Trail Atom x5-Z8300, USB 3.0, microSD, HDMI, eDP/MIPI, and GbE, plus optional WiFi and Bluetooth 4.2 LE. In June, Radxa unveiled its Rock Pi S SBC that runs Linux on a RK3308 and updated its RK3399-based Rock Pi 4 with extra memory. Now, Radxa is preparing to add to that family of Raspberry Pi pseudo clones with an SBC called Rock Pi X, based on the Intel “Cherry Trail” Atom x5-Z8300.

While this is Radxa’s first Intel Atom SBC, several open spec boards are based on the Atom x5-Z8300, including the Atomic Pi from Team IoT (DLI) and the UP board and UP Core board from Aaeon UP. Intel’s “Cherry Trail” Atom x5 Z8350 SoC can be clocked at up to 1.84 GHz and has a 500 MHz Intel Gen 8 HD 400 GPU featuring 12 Execution Units. Aside from having different processors,

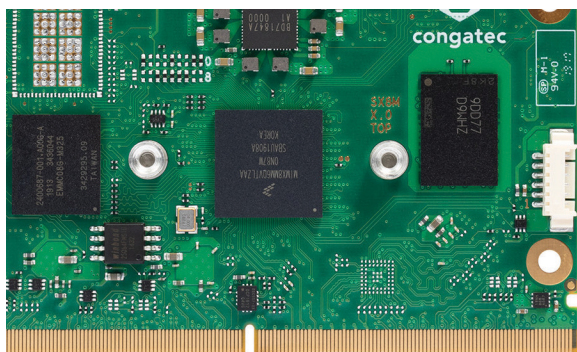


spec-for-spec, the 85 mm x 51 mm Rock Pi X is most similar to Radxa’s 85 mm x 54 mm Rock Pi 4. Both provide 4 GB of RAM, microSD, HDMI and a Gbit Ethernet port.

Radxa | wiki.radxa.com

SMARC 2.0 Module Runs Linux on i.MX8M Mini

Congatec’s “Conga-SMX8-Mini” SMARC 2.0 module runs Linux on NXP’s i.MX8M Mini with up to 4 GB LPDDR4 and 128 GB eMMC and optional Wi-Fi and -40 to 85°C. There’s also a new carrier and coolers for Congatec’s Epyc 3000 based conga-B7E3 module.



Congatec touts the module for its MIPI-CSI-2 interface and support for an upcoming SMARC MIPI-CSI-2 starter kit to be released in cooperation with industrial camera manufacturer Basler. No more details were available about this “highly integrated embedded vision platform” that support the “development of cost-efficient vision devices for sparse modeling-based AI.” Congatec offers a similar Conga-CAM-KIT/MIPI kit for its Intel Apollo Lake based Conga-PA5 Pico-ITX SBC that uses a Leopard Imaging sensor instead of a Basler camera.

The 82 mm x 50 mm Conga-SMX8-Mini offers Linux, Yocto Linux, or Android BSPs with “ready-to-go boot loader implementation” for the single, dual, and quad-core versions of the i.MX8M Mini. The Cortex-A53 cores are clocked at 1.8 GHz on the standard 0 to 60°C models and 1.6 GHz for the industrial -40°C to 85°C SKUs.

Congatec | www.congatec.com

Functional Safety Tools Support STMicro’s 8-bit STM8 MCUs

IAR Systems has further extended its tools offering for safety-related software development by launching a certified version of its development tools for STM8 MCUs.

STMicroelectronics’ 8-bit STM8 MCUs are used for automotive and other industrial applications where reliability and cost effectiveness are important. The functional safety edition of IAR Embedded Workbench for STM8 is certified by TÜV SÜD according to the requirements of IEC 61508, the international umbrella standard for functional safety, as well as ISO 26262, which is used for automotive safety-related systems.

In addition, the certification covers the international standard IEC 62304, which specifies life cycle requirements for the development of medical

software and software within medical devices, and the European railway standards EN 50128 and EN 50657.

The functional safety edition of IAR Embedded Workbench for STM8 includes a functional safety certificate, a safety report from TÜV SÜD and a Safety Manual. With the certified tools, IAR Systems provides a Functional Safety Support and Update Agreement with guaranteed support for the sold version for the longevity of the contract. Along with prioritized technical support, the agreement includes access to validated service packs and regular reports of known deviations and problems.



 IAR Embedded Workbench

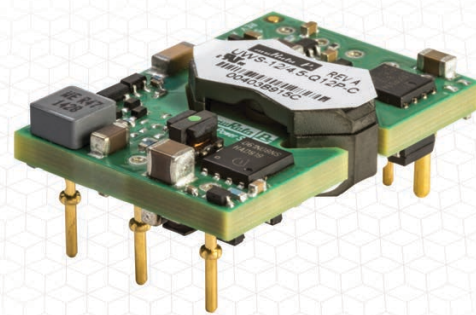
IAR Systems
www.iar.com

PRODUCT NEWS

1/16th Brick DC-DC Converter Provides 9-36 VDC Voltage Input

Murata has announced the introduction of its UWS-Q12 series, the latest in a line of 50 W, 9-36 V_{in} range DC-DC converters. This series was developed for a wide range of applications including, network equipment, industrial, railway, power grid and transportation. In an industry standard 1/16th brick pinout, the solution provides a basic I/O insulation system rated at 2,250 VDC isolation with a fully regulated DC output. The series also offers single output modules with outputs of 3.3 V, 5 V, 12 V, 15 V and 24 V DC all rated at 50 W. Further, with a universal V_{in} range, requirements for Intermediate Bus Converter (IBC) architecture are readily met.

The UWS-Q12 series provides numerous standard features including positive or negative ON/Off control, output over current protection, over temperature protection, Input under Voltage lock-out, short circuit protection, Pre-Bias protection, V_{out} trim, and V_{out} sense function. The line has a galvanic isolation barrier between the input and output of the module with a basic insulation system rated at 2,250 VDC. Each converter



is designed to deliver 50 W of power with efficiencies reaching 91 percent on the 5 V, 12 V, 15 V and 24 V_{out} versions and 89.5% on the 3.3 V_{out} model.

Murata Power Solutions
www.murata-ps.com

Single-Chip Motor Driver Enables Virtually Silent Motor Operation

Trinamic Motion Control has unveiled an ultra-small single-chip motor driver that uses StealthChop technology to enable virtually silent operation. The device is designed to drive two-phase stepper motors up to 1.2 A_{RMS} and with a voltage range of 1.8 VDC to 11 VDC. With a standby current draw of < 50 nA, it can provide a solution that requires only one or two Li-Ion cells or two AA batteries.

The TMC2300 incorporates three exclusive Trinamic technologies: StealthChop2: A high-precision algorithm that produces drive waveforms which enable motors to be inaudible—both in motion and at standstill; StallGuard: Sensorless

motor load measurement, a combination of on-chip circuitry and firmware that enables the driver to perform sensorless homing and detect mechanical obstacles; and CoolStep

Sensorless: Load-dependent current control that optimizes the motor's energy consumption on the fly, enabling energy savings of up to 80% over conventional motor drives, according to Trinamic.



Trinamic Motion Control
www.trinamic.com

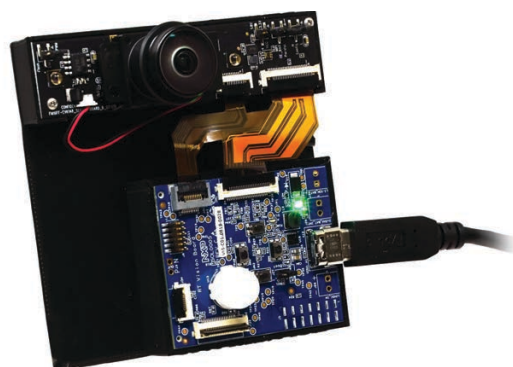
MCU-based Solution Enables Offline Facial Recognition

NXP Semiconductors has unveiled what it claims is world's first MCU-based solution for adding offline face and expression recognition capabilities to smart home, commercial and industrial devices. Built on NXP's latest crossover MCU, the i.MX RT106F, running FreeRTOS, the new MCU-based face recognition solution enables original equipment manufacturers (OEMs) to quickly, easily and inexpensively incorporate face, expression and emotion recognition into a diverse range of IoT products.

The i.MX RT106F leverages NXP's OASIS face processing engine and uses a neural network to perform face detection, recognition and anti-spoofing, without the need for cloud connectivity. OEMs can take advantage of NXP's hardware and software-based platform to offer advanced human machine interface (HMI) capabilities that can anticipate and personalize the end user's experience with smart edge devices such as smart appliances, thermostats, lighting, alarms and power tools.

NXP is now engaging with OEMs to provide early access to the evaluation and development kit for this solution, and broad market availability is expected to begin in Q1 2020.

NXP Semiconductors | www.nxp.com



PRODUCT NEWS

Two Power Delivery Chips Provide USB Type-C Charging Solutions

Microchip Technology has announced two new solutions that simplify USB Type-C PD (Power Delivery) for a range of applications. The company claims it as one of the industry's first USB-IF-certified USB 3.1 SmartHub devices with integrated support for Power Delivery (TID1212). The USB705x family enables fast device charging and introduces unique PD implementations called HostFlexing and PDBalancing. The second device, the UPD301A, is a standalone USB Type-C PD controller that significantly simplifies the implementation of basic USB Type-C PD charging functionality, making it well suited for applications from rear seat charging in vehicles to portable equipment to public charging stations.

The USB705x family includes two unique features that simplify USB Type-C PD implementations – HostFlexing and PDBalancing. HostFlexing simplifies the user's docking station experience by allowing all USB Type-C ports to function as the "notebook" port, eliminating the need for cryptic labels that try and explain overall functionality of each USB Type-C port.

The UPD301A is available today starting at \$1.50

in 10,000-unit quantities. The USB705x family is available today with options and pricing for 10,000-unit quantities, with price ranging from \$4.82 to \$5.35, depending on configuration.

Microchip Technology | www.microchip.com



Reusable Solderless Robotics Kit Features SimpleLink MCU

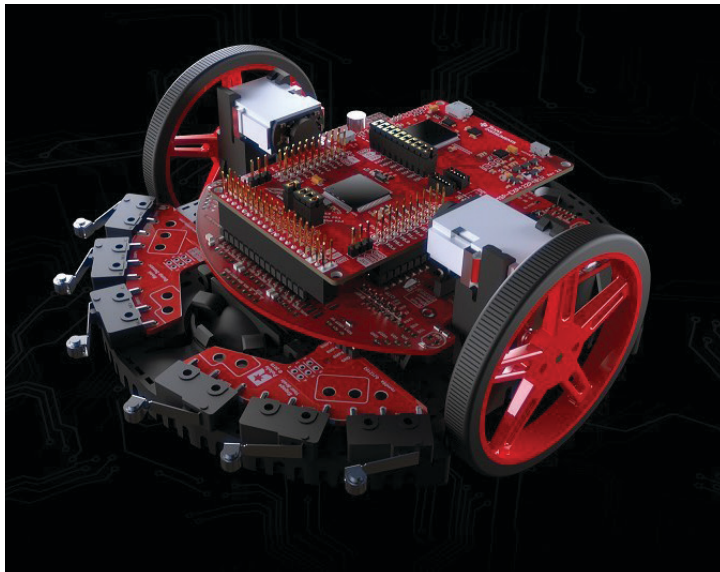
Texas Instruments (TI) has introduced the newest addition to the TI Robotics System Learning Kit (TI-RSLK) family, the TI-RSLK MAX, a low-cost robotics kit and curriculum that is

simple to build, code and test. Designed for the university classroom, the solderless assembly allows students to have their own fully functioning embedded system built in under 15 minutes. Classrooms that may not have access to soldering equipment benefit from the solderless, hands-on kit and curriculum that can be reused year after year.

Designed for the university classroom, the TI-RSLK MAX is a low-cost robotics kit and curriculum that is simple to build, code and test. The new kit includes TI's SimpleLink MSP432P401R microcontroller (MCU) LaunchPad Development Kit, easy-to-connect sensors, and a versatile chassis board that turns the robot into a mobile learning platform. Through accompanying core and supplemental curriculum, students learn how to integrate their hardware and software knowledge to build and test a system.

The TI-RSLK MAX is available for purchase for US\$109 from the TI Store and includes the SimpleLink MSP432P401R MCU LaunchPad Development Kit, as well as all additional components required for assembly. To expand kit functionality and learning paths, optional accessories are available for purchase.

Texas Instruments | www.ti.com



STATEMENT REQUIRED BY THE ACT OF AUGUST 12, 1970, TITLE 39, UNITED STATES CODE SHOWING THE OWNERSHIP, MANAGEMENT AND COPY CIRCULATION OF CIRCUIT CELLAR. Published monthly at 650 Dairy Farm Rd, Red Oak, VA 23964. Annual subscription price is \$50.00. Publisher: KC Prescott. The owner is KCK Media Corp., 650 Dairy Farm Rd, Red Oak, VA 23964. The names and addresses of stockholders holding one percent or more of the total amount of stock are: KC Prescott, 650 Dairy Farm Rd., Red Oak VA 23964. EXTENT AND NATURE OF CIRCULATION: Average number of printed copies of each issue published during the preceding twelve months; (A) total number of copies printed, 5,608; (B.1) paid/requested mail print subscriptions, 1,835; (B.3) sales through dealers and carriers, street vendors and counter sales, 2,040; (B.4) paid/requested copies distributed by other mail classes, 29; (C) total paid/requested print circulation, 3,898; (D.4) Nonrequested copies distributed outside the mail, 1,092; (E) total nonrequested distribution (sum of D.1 & D.4), 1,092; (F) total distribution (sum of C & E), 4,990 (G) copies not distributed (office use, leftover, unaccounted, spoiled after printing, returns from news agents), 618 (H) total (sum of F & G), 5,718. Percent Paid Requested: 78.11% (A) total number of paid Electronic Copies, 2,377; (B) Total Paid Print & Paid Electronic Copies 6,275 (C) Total Print Distributions and Paid Electronic Copies 7,367 Percent Paid: 85.17% Actual number of copies of a single issue published nearest to filing date: (A) total number of copies printed, 5,398; (B.1) paid/requested mail print subscriptions, 1,815; (B.3) sales through dealers and carriers, street vendors and counter sales, 1,793; (B.4) paid/requested copies distributed by other mail classes, 24; (C) total paid/requested circulation, 3,633 (D.4) Nonrequested copies distributed outside the mail, 1,250; (E) total nonrequested distribution (sum of D.1 & D.4), 2,017; (F) total distribution (sum of C & E), 4,883 (G) copies not distributed (office use, leftover, unaccounted, spoiled after printing, returns from news agents), 515; (H) total (sum of F & G), 5,398. Percent Paid Requested 74.4%. (A) total number of paid Electronic Copies, 2,235; (B) Total Paid Print & Paid Electronic Copies 5,986 (C) Total Print Distributions and Paid Electronic Copies 7,236 Percent Paid: 82.72% I certify that the statements made by me above are correct and complete. KC Prescott Publisher.

IDEA BOX

The Directory of PRODUCTS & SERVICES

AD FORMAT:

Advertisers must furnish digital files that meet our specifications (circuitcellar.com/mediakit).

All text and other elements MUST fit within a 2" x 3" format.
E-mail adcopy@circuitcellar.com with your file.

For current rates, deadlines, and more information contact Hugh Heinsohn at 757-525-3677 or Hugh@circuitcellar.com.

Revenue Control Systems **PnP**

- Pick & Place Machines starting @ \$6,250
- Direct U.S. Sales, Support, Training, Parts, Accessories, Warranty
- PCB Fabrication Equipment
- Makerspace Specials
- Reflow Ovens

757-258-0910

RCSPnP.com



RAPID18 CCS

PIC 18 Universal Development Kit



ONLY \$99!

- Real time Clock/Calendar with Supercap
- USB port for text communication to the running program
- Built-in bootloader for loading code
- Good for AC/DC Data Logging

Kit includes everything you need to develop with a PIC® MCU:

- Rapid 18 Prototyping Board
- Single-Chip IDE C Compiler
- Exercise Book
- Power Adapters and Cables

sales@ccsinfo.com (262) 522-6500 x35
www.ccsinfo.com/cc1119

ALL ELECTRONICS

Surplus & New Parts & Supplies Since 1967

LEDS · CONNECTORS · RELAYS
SOLENOIDS · FANS · ENCLOSURES
MOTORS · WHEELS · MAGNETS
PC BOARDS · POWER SUPPLIES
SWITCHES · LIGHTS · BATTERIES
and many more items...




We have what you need for your next project.

Discount Prices
Fast Shipping

www.allelectronics.com

Technologic Systems

Single Board Computer



TS-7250-V2

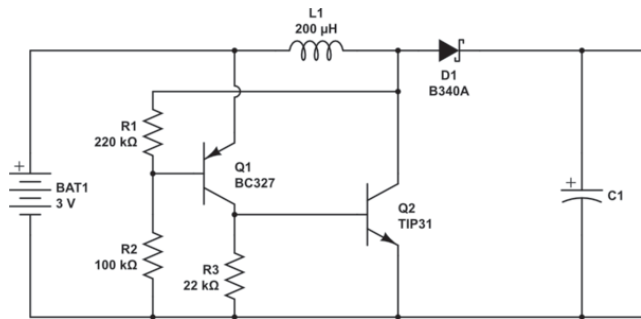
1GHz ARM Computer with Customizable FPGA-Driven PC/104 Connector and Several Interfaces at Industrial Temp

www.embeddedARM.com

TEST YOUR EQ

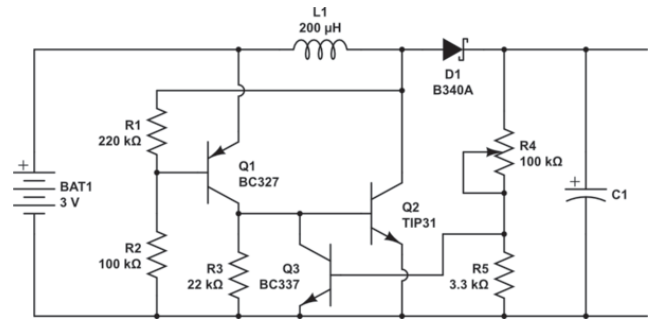
Contributed by David Tweed

Problem 1— This circuit (Figure 1) is used to boost the output of a 3V battery to levels high enough to light a string of LEDs. Explain how this circuit oscillates.



Problem 2— What limits the amount of power that this circuit transfers from input to output?

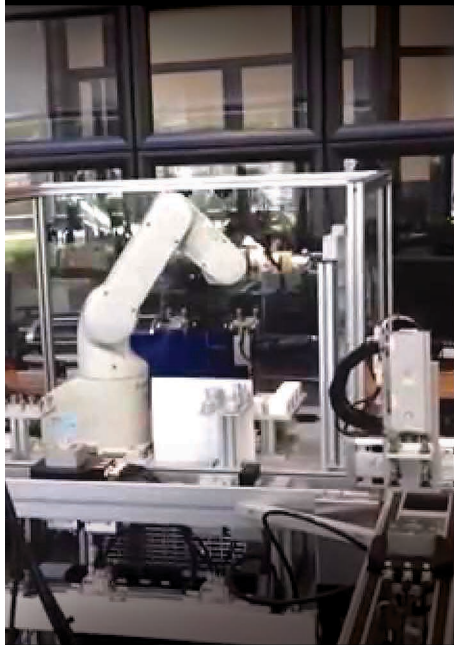
Problem 3— Figure 2 shows a modified version of the Figure 1 circuit. Explain what Q3 does.



Problem 4— Suppose the Figure 2 circuit is used to drive an LED assembly that requires 200 mA at 12 V. How much current must pass through L1 and Q2?

You can take it almost anywhere.

Where will it take you?



Now you can have the complete Circuit Cellar issue archive and article code stored on a durable and portable USB flash drive.

Includes PDFs of all issues in print through date of purchase.

Visit cc-webshop.com to purchase

The Future of Smart Homes

The Essentials of Smart Home Security

According to *Forbes*, it is estimated that total spending on Internet of Things (IoT) devices and infrastructure will reach \$1.2 trillion by 2022—up from an estimated \$151 billion last year. Such rapid growth indicates that the IoT is penetrating deeply into many markets, from first adopters a decade ago to today, where 90% of business executives in technology, media and telecom consider IoT technology to be central to their business strategy.

When you consider that one of the major bottlenecks to widespread IoT adoption and development has been the rapidly evolving radio frequency (RF) technology landscape, it's clear that companies should start investing in this ever-growing space with scalable and flexible microcontroller (MCU) platforms.

One of the fastest growing IoT spaces is in the home network. The “home network” includes products that work together seamlessly to provide both a smart and secure home experience. Inside today's smart home products and talking home assistants, however, is a much more sophisticated and intricate story. The smart home market is fragmented at several levels, making device interoperability a challenge.

At one level, the smart home consists of several sub-categories such as building security, heating, ventilation, air conditioning (HVAC) and fire safety systems. Anyone who even casually follows the smart home market has seen the rapid growth of the building security sub-category with the influx of out-of-the-box security systems available for homeowners to purchase. These security systems ship directly to the front door and are immediately ready to install by the homeowner.

When designing a smart home security system like this, flexibility and scalability are paramount. Meanwhile, interoperability, given the need for multiple peripheral devices, can be a major challenge. To overcome these design challenges, security service providers and security system companies are working to make products interoperable out of the box by integrating the essential components illustrated in **Figure 1**. These components can be divided into these three categories:

- Sensing, including door and window sensors, motion detectors and glass break detectors.
- Monitoring, using security cameras and video doorbells.
- Control (both local and remote) using gateways, access panels, electronic smart locks, cloud-based dashboards and smartphone apps.

The requirements for implementing sensing, monitoring and control into a security system differ, making it increasingly challenging for security system companies to keep up with and stay ahead of the needs of all three functions while minimizing additional design time and effort.



By
Michelle Tate,
Product Marketing Engineer,
Texas Instruments

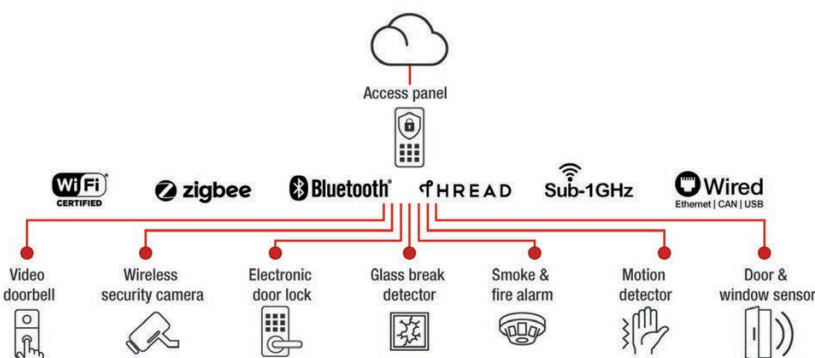


FIGURE 1
Home security system setup example



EXAMINING PRIORITIES

Let's review what security providers should prioritize when developing a smart home security system.

Sensing priorities: long range and low power: We all know a person's homestead is typically the largest asset on their balance sheet. At the same time, it also provides storage for a plethora of personal valuables. The interest of homeowners to easily protect their assets with low hassle is driving the need for security companies to provide solutions that can both secure all of the entry points of a home as well as enable the least amount of upkeep maintenance as possible.

Long range enables homeowners to place a sensor in remote and hard-to-reach locations, such as a window on the third floor of a home, therefore extending coverage to areas previously unreachable by wired systems. Enabling greater coverage through extended range in turn increases the number of connected sensors in a home as well as the burden for homeowners to monitor battery life and battery replacement cycles. By creating lower power sensors, battery life can be elongated to multiple years, therefore reducing the maintenance burden on the homeowner.

Sub-1 GHz is currently the leading technology for sensors due to its extremely long range, ability to penetrate walls and low-power capabilities. Because there are no standards bodies currently overseeing the Sub-1 GHz bandwidth, many developers must create their own proprietary protocols from scratch, requiring significant R&D investment, time and RF expertise. Some silicon providers will make this investment for security system developers and provide an out-of-the-box toolkit to help them get started in Sub-1 GHz design.

Zigbee and Thread are 2.4 GHz mesh standards. Zigbee enables ultra-low power through Zigbee Green Power, which supports battery-less devices by enabling sensors that can harvest mechanical energy from movement, such as opening a door or window sensor. Thread is designed specifically for home networks and is based on Internet Protocol ver. 6, which enables Thread devices to have an easy connection to existing networks.

Monitoring priorities: high throughput and security: If a picture is worth a thousand words, video is worth a million — when it comes to home security, that could not be more true. Wi-Fi is typically used to achieve the throughput required to stream video from monitoring home entry points. Choosing a Wi-Fi MCU that can support 4 Mbps or more is important to enable 1080p video streaming.

In addition, monitoring undergoes extra security scrutiny from home/building installers, as the information being stored and/or sent over the air is more sensitive. Selecting an MCU that has comprehensive end-to-end security, from storage to run time to transfer, can further help secure a monitoring design.

Control priorities: multiprotocol concurrency and remote control: Multiple wired and wireless connectivity standards enable connections to wireless sensing and monitoring functions, and for large buildings, wired connections back to a central server.

A combination of wireless technologies, such as Wi-Fi, Bluetooth low energy, Zigbee, Thread and Sub-1 GHz, and a wired connection, such as Ethernet, are commonly used for control. Combining multiple wireless technologies in a single control component requires both multiprotocol concurrency and coexistence.

Enabling multiprotocol concurrency on a single MCU can be done by developing software that switches between protocol stacks in real time based on protocol priority tables. Developing low latency multiprotocol managers helps enable the control unit to interact in multiple networks at the same time while also ensuring successful packet transmissions due to the fast switching time.

Supporting coexistence on two 2.4 GHz MCUs, such as a Bluetooth MCU and Wi-Fi MCU, is another important design aspect to consider. By using time division multiplexing, the two MCUs can share the same antenna and reduce the bill of materials (BOM) of a design. In addition, by designing with Bluetooth and Wi-Fi, homeowners have the ability to remotely access and control their smart security system, with Bluetooth providing shorter-range remote access through the phone and Wi-Fi providing remote access through the cloud.

SCALABILITY AND FLEXIBILITY

There are many dominant and newly emerging connectivity solutions that aim to meet sensing, monitoring and control requirements. However, with the increasing number of connectivity solutions and system requirements, it has become increasingly difficult for security system companies to stay up to date with market demands, such as longer range, lower power, faster networks and greater security, without having to divert additional resources toward system redesigns.

Selecting an MCU platform that supports both wired and wireless connectivity protocols enables code reuse through common software development kits and application programming interfaces. Security system companies benefit from more flexible and scalable designs, further enabling them to stay ahead of market needs. 

For detailed article references and additional resources go to: www.circuitcellar.com/article-materials

RESOURCE

Texas Instruments | www.ti.com

At Texas Instruments, Michelle Tate serves as a product marketing engineer for the SimpleLink connected MCU team, specializing in building security systems. She received her bachelor's degree in electrical engineering from The University of Texas at Austin.

STRONG FOUNDATION

Whether you are an
EMS, CM or OEM,
let our bare boards be the foundation
you build your reputation upon!

Technology:

Up to 50 Layers
Any Layer HDI
Sequential Lamination
Blind / Buried Vias
Laser Drilling / Routing
Heavy Copper

Materials:

Fr4
Metal Core
Isola
Rogers
Polyimide - Flex
Magtron

**We will make only what is needed,
when it's needed,
and in the amount needed.**

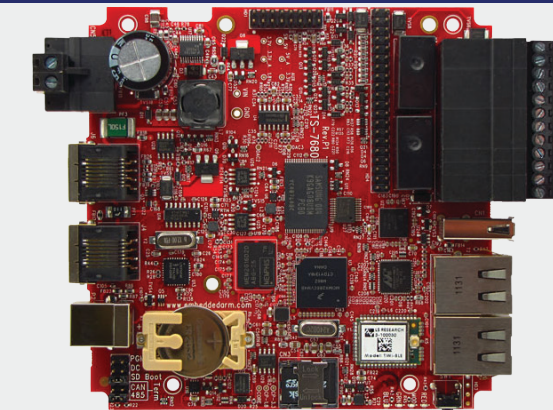
You no longer have to worry about long shelf life
or tie your capital in bare board inventory.

Accutrace[®] inc.

www.PCB4u.com sales@PCB4u.com

SAM & ITAR Registered UL E333047 ISO 9001 - 2008

FROM THE DEEP BLUE SEA TO THE WILD BLUE YONDER



TS-7680

Low Power Industrial
Single Board Computer with
WiFi and Bluetooth

\$159
Qty 100

The TS-7680 is designed to provide extreme performance for applications demanding high reliability, fast boot-up/startup, and connectivity at low cost and low power. Because there are so many features packed on to one single board computer you will see a reduction in payload weight since there is no need for additional boards, micro-controllers, or peripherals.

Rated for industrial temperature range of -40°C to $+85^{\circ}\text{C}$ the TS-7680 is deployed in fleet management, pipeline monitoring, and industrial controls and is working in some of the most demanding places on Earth.

The TS-7680 will help you perform at your very best in a variety of critical missions.



Made in USA
with Global Parts

 **Technologic**
Systems