

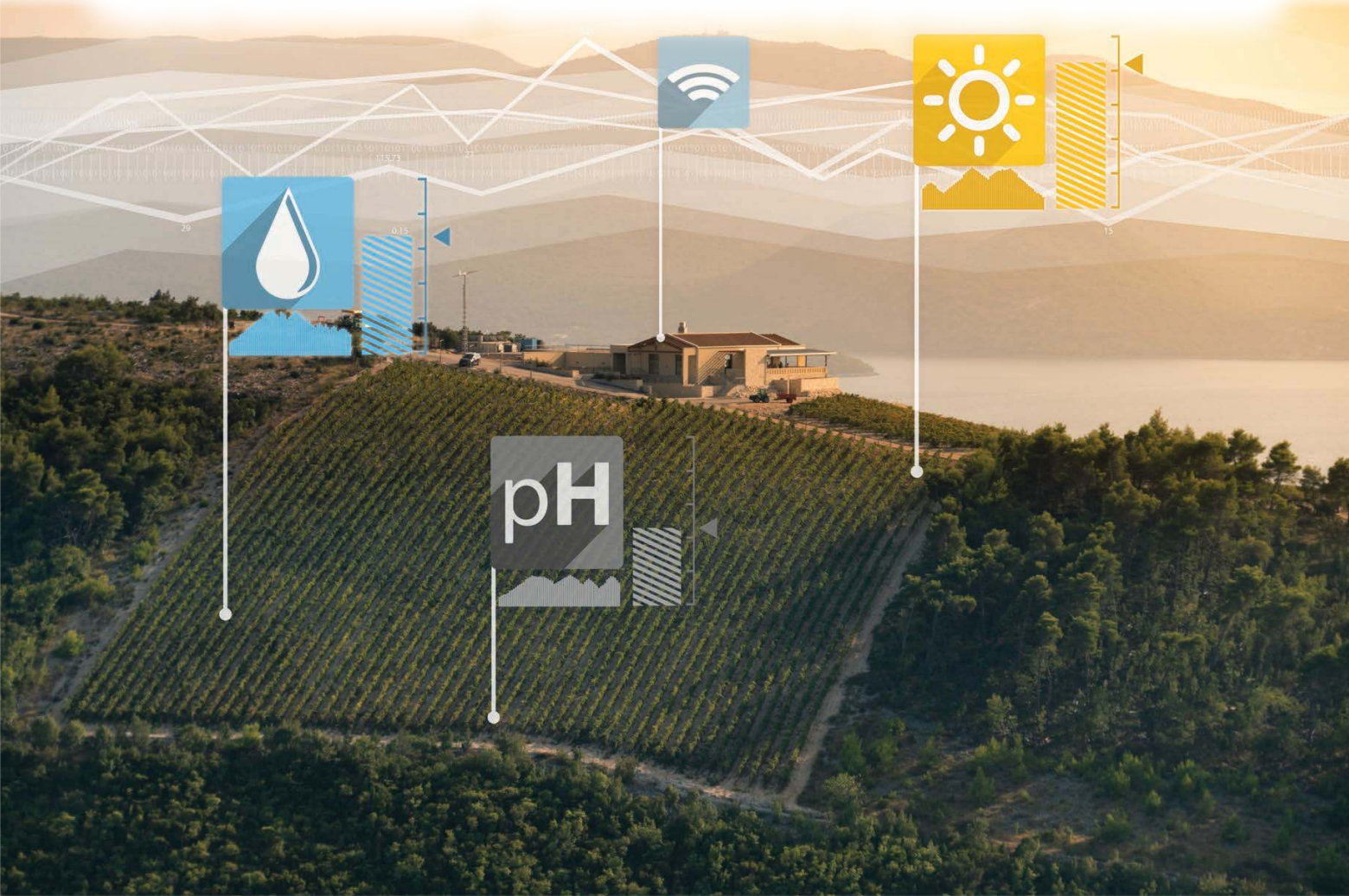


circuitcellar.com

circuit cellar

Inspiring the Evolution of Embedded Design

IoT TECHNOLOGIES FEED SMART AGRICULTURE



- ▶ Datasheet: Mini-ITX and Pico-ITX SBCs
- ▶ Embedded Software Tool Security | Food Delivery Notifier Uses Bluetooth | Intro to Ardupilot and PX4 (Part 2) | Creative Mechanical Ideas for Embedded | Modernizing Antique Clocks | Pest Control System
- ▶ Broad Market Secure MCUs | Weather Tree Upgrade | Build a SoundFont MIDI Synthesizer (Part 1)
- ▶ The Future of Linux Security



You can take it almost anywhere.

Where will it
take you?



CC VULT

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

Complete archive includes PDFs of all issues in print through date of purchase.



Visit cc-webshop.com to purchase

\$29
value



ADVANCED
ASSEMBLY

PCB ASSEMBLY

AT THE SPEED OF IMAGINATION

DESIGN-FOR-ASSEMBLY FUNDAMENTALS

for getting your PCBs back fast



bit.ly/fastPCBs | (800) 838-5650

OUR NETWORK



SUPPORTING COMPANIES

Advanced Assembly	1
All Electronics Corp.	77
CCS, Inc.	77
easyOEM	49
HuMANDATA, Ltd.	13
IAR Systems, Inc.	C3
Newhaven Display International, Inc.	11
Pico Technology	21
Technologic Systems, Inc.	C4, 77
University of Cincinnati	19
Wind River	41

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

COLUMNISTS

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

Issue 358 May 2020 | 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 © 2020 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. 2020 Printed in the United States

INPUT Voltage

Tech Community Steps Up to Battle COVID-19

On behalf of the *Circuit Cellar* staff, we hope that you and your loved ones remain safe and healthy during this challenging time. Our team all work remotely and we're committed to keep assembling the quality magazine you've come to expect each month. Hopefully we can keep inspiring you, provide some distraction and share some interesting project stories during this uncertain time.

Shifting gears, I've been incredibly inspired by the roles the embedded community—from technology companies to individuals—have played as they've stepped up in their own unique ways to battle the COVID-19 pandemic. These roles include large, generous efforts in resources and equipment, but also intriguing cases where embedded technologies have been crucial in enabling solutions for dealing with COVID-19 at many levels.


As I write this (in early April), Intel has just announced it's pledging \$50 million in a pandemic technology initiative to combat the coronavirus through accelerating access to technology at the point of patient care, speeding scientific research and ensuring access to online learning for students. Around \$40 million will fund the Intel COVID-19 Response and Readiness and Online Learning initiative. Intel says that initiative will provide funding to accelerate customer and partner advances in diagnosis, treatment and vaccine development, leveraging technologies such as artificial intelligence (AI), high-performance computing and edge-to-cloud service delivery. Through the initiative, Intel will help healthcare and life sciences manufacturers increase the availability of technology and solutions used by hospitals to diagnose and treat COVID-19.

You may have heard of Kinsa Health in the news recently. Kinsa makes smart thermometers that connect via Bluetooth to the Kinsa App. The app can not only keep a log of readings, but also provide users with a range of benefits including guidance on when to seek further medical advice, provide medication reminders and so on. For several years, anonymous data from the Kinsa App has enabled Kinsa Health to produce a temperature heat map of the US (that Kinsa calls its US Health Weather Map) that could be used to identify potential COVID-19 hotspots much more quickly and help government agencies and healthcare organizations in their on-going battle against the virus.

Kinsa's thermometers are based on Nordic Semiconductor's nRF52810 SoC. Because Kinsa's smart devices are battery-powered, they require an ultra low-power Bluetooth Low Energy solution, but one that also has enough on-board processing power and memory to essentially run the entire smart medical application from a single chip.

The DIY community has also been engaging in ways to contribute their expertise to the COVID-19 battle. Embedded processor vendor Espressif Systems reports a story of an Indian engineer, Abhijit Mukherjee, that has come up with an Espressif ESP8266-based solution for safe measurements of body temperature during the COVID-19 crisis. Abhijit says he "felt the urge to do something which could help" during the current global pandemic. By posting his project on hackster.io, he also wants to invite suggestions from other makers on how to improve his solution.

Baffled by the lack of reasonably-priced contactless thermometers on the market, and prompted by the necessity to safely check from a distance the temperature of people he had to deal with in his personal and professional, daily life, Abhijit says he built a completely autonomous and contactless, IR temperature-measuring device which can be mounted anywhere—an office door, an apartment entrance, or the on front gate of an apartment block. Abhijit's gadget can track people's temperature and post the results to any cloud or incoming webhooks (a simple way to post messages from apps into the Slack messaging app). We'll post a link to Abhijit's ESP8266-based, contactless, IR thermometer on *Circuit Cellar's* article materials webpage.

All of you please stay healthy and safe, but also as connected and engaged as you can. 



Jeff Child

COLUMNS

50 TECHNOLOGY SPOTLIGHT Embedded Software Tools Bulk Up on Security

Connected System Concerns

By Jeff Child

54 DATASHEET Mini-ITX and Pico-ITX SBCs

Performance Platforms

By Jeff Child

58 Picking Up Mixed Signals Build a SoundFont MIDI Synthesizer

(Part 1) Using Teensy 4

By Brian Millier

64 Embedded System Essentials Broad Market Secure MCUs

Spotlight on the MAX32520

By Colin O'Flynn

68 From the Bench Upgrading the Weather Tree

With I²C Interfacing

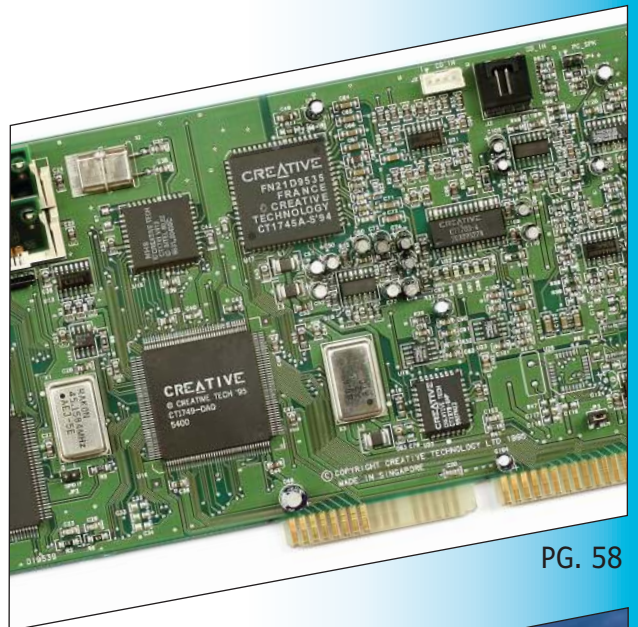
By Jeff Bachiochi

79 TECH THE FUTURE The Future of Linux Security Securing Linux-Based Systems in 4 Steps

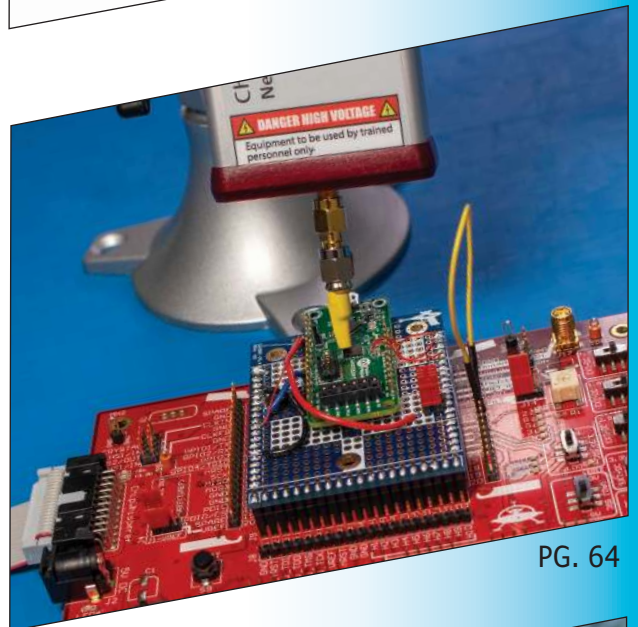
By Glenn Seiler

76 : PRODUCT NEWS

78 : TEST YOUR EQ



PG. 58

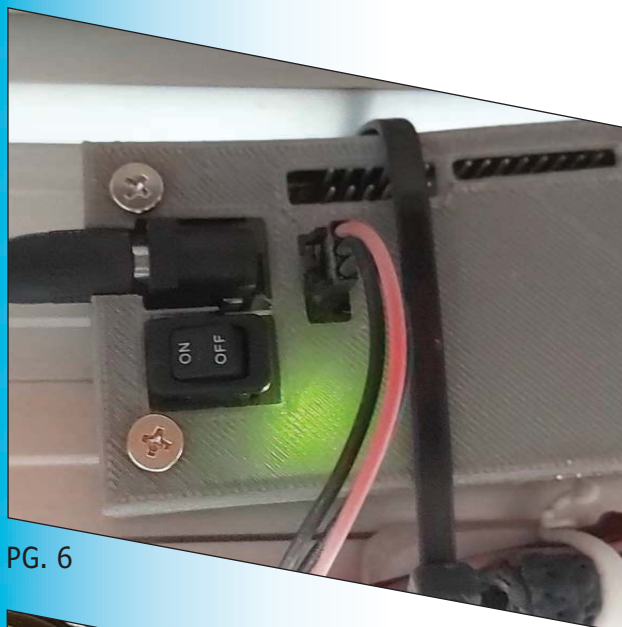


PG. 64

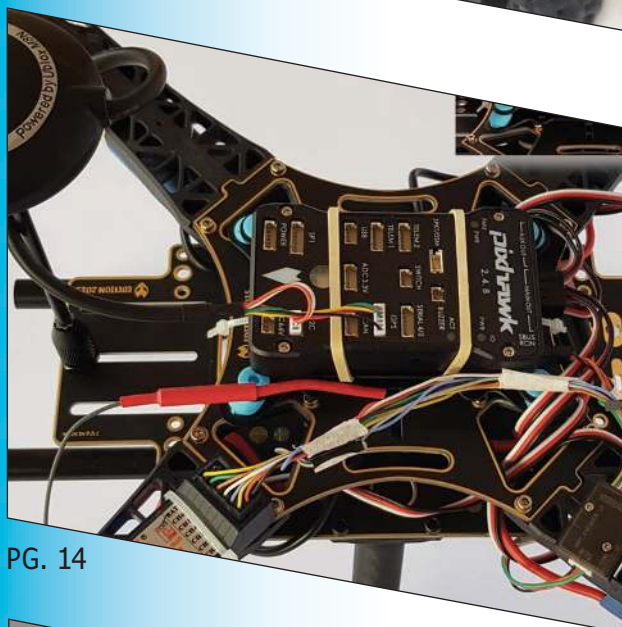


PG. 68

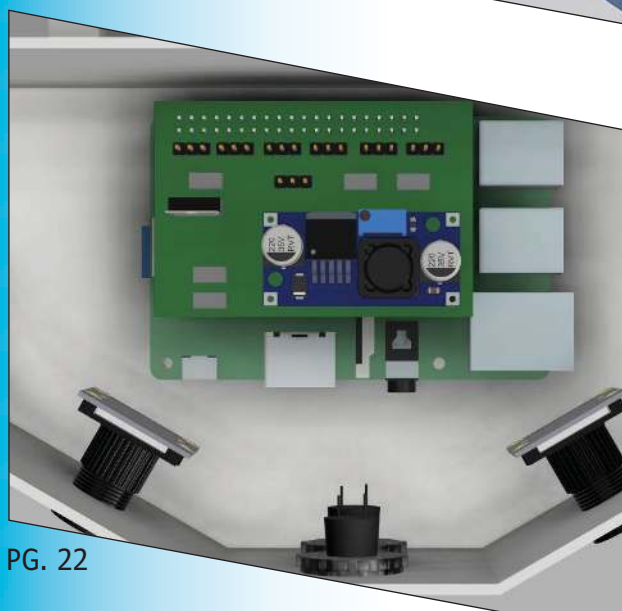
FEATURES



PG. 6



PG. 14



PG. 22

6 Proximity Food Delivery Notifier

Bluetooth-Based Design

By Kenichi Kato

14 Intro to Ardupilot and PX4 (Part 2) Building the Drone

By Raul Alvarez-Torrico

22 Build an Automated Pest Deterrent System

Using Raspberry Pi

By Cole Gamborski, Cameron Phillips and Simon Fowler

28 Creative Mechanical Ideas for Embedded Systems

Professional Style Projects

By Wolfgang Matthes

36 Modernizing the Accuracy of an Antique Clock

Using a PSoC4 MCU

By Aubrey Kagan

SPECIAL FEATURE

42 Smart Agriculture Designs Tap IoT Technologies

The Internet of Growing Things

By Jeff Child

Proximity Food Delivery Notifier

Bluetooth-Based Design

Home delivery of meals is very popular these days, and technology is only fueling that trend. But nobody wants food that's gone cold because the delivery guy forgot to ring the doorbell. Like any good engineer, Kenichi built a sensor-based solution designed to wirelessly alert him via Bluetooth that the food has arrived on his doorstep.

By
Kenichi Kato

It's well known that Singapore is a food paradise. I couldn't agree more. But, it's really not healthy to go out to eat every day. Homecooked food is always the best, but it can be extremely time consuming and not so economical for a household of two like ours. With that in mind, so, we order "Tingkat"—home delivered food services—for our dinner

during the weekdays. It's very economical and convenient and best of all, you can get food made for health-conscious customers.

For us, the delivery time varies between 3 PM and 7 PM. The problem is that delivery guy—due to his tight delivery schedule—often forgets to press the doorbell to alert us that the food is delivered. Sometimes, the food would be at the gate for more than a few hours. If not kept at a certain temperature, some foods can spoil easily—or a mischievous neighbor might just take it and have a free meal. I'm not saying that from our experience. We have nice neighbors! So, as any engineer would do, all this inspired me to create a solution. This article describes my Proximity Food Delivery Notifier system for solving this problem.

FOOD ALERT

The basic idea is to have a system to alert us that our food has been delivered at our doorstep. The overall design (**Figure 1**) shows a fixed location with a hook for placing the food where a sensor will also be planted, a controller that will read and wirelessly relay that information to a receiver. In this case, a smartphone would work just fine. So, for the specific location, a sensor detects the existence of the food based on the proximity of an object. In this project, I have chosen an ultrasonic sensor as the proximity sensor because it provides some accuracy on the

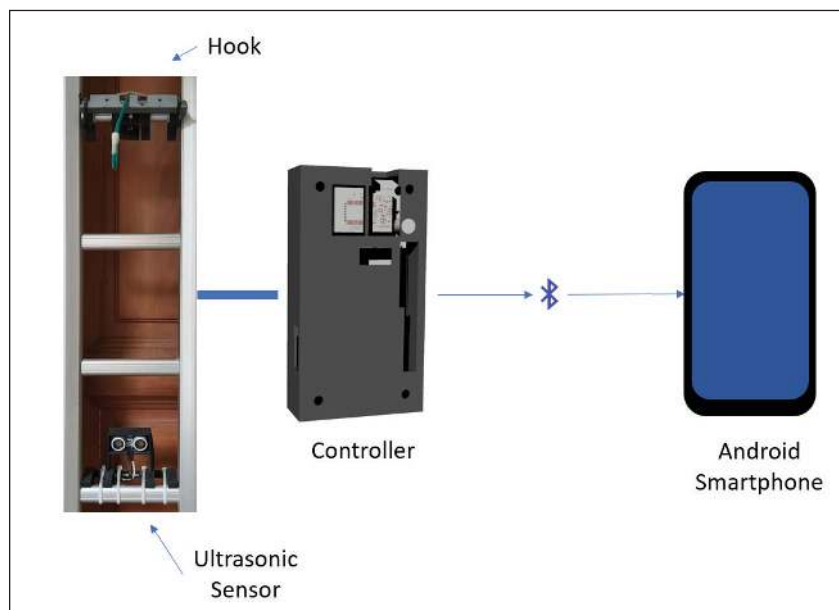


FIGURE 1
This shows the overall design with the major sub-systems.



distance from the object. I found the Parallax PING sensor (**Figure 2**) to be particularly easy to implement because it provides information on distance in both centimeters and inches. Since I am using the Parallax Propeller for this project, the code that comes with the PING sensor is particularly helpful—providing distance readings in centimeters and inches.

I selected Bluetooth as the wireless protocol for this project. That's mainly because the Bluetooth module I had on hand is a Class 2 and the distance between the delivered food and the receiver is also less than 10 meters. I used an HC-05 module (**Figure 3**) because I have quite a stock of these modules from my previous projects that I did back in 2013. Moreover, this module is very easy to implement because the PCB has castellation pads. These pads make it easy to either mount it as a SMT device on a PCB or insert to breadboard after soldering in the headers. Besides all that, this module is also very versatile in that configuration to it are all via the AT commands in a UART terminal. The module is also very widely used, so there are many articles and websites providing useful information about its configuration. And, the best part is you can get it for less than \$5 (US).

As for the controller, the location for holding the delivered Tingkat food is very small and narrow, so I designed a small PCB that houses all the necessary components and the Parallax Propeller chip. This is the Propeller version 1 since the Propeller version 2 was just being released as of this writing.



FIGURE 2
Parallax's PING ultrasonic sensor.

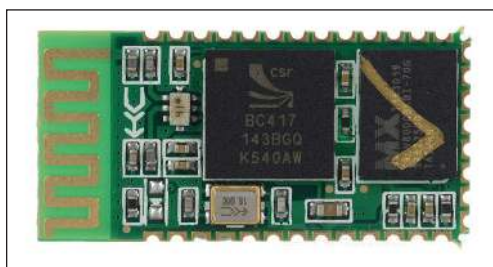


FIGURE 3
Wavesin's Bluetooth Class 2 module.

The **Figure 4a** schematic shows the basic components for the microcontroller, which includes the EEPROM and 5MHz crystal plus connectors, while the **Figure 4b** schematic shows the HC-05 Bluetooth module and an optional tactile switch for the microcontroller as user input. Finally, schematic **Figure 4c** is the power circuit with LDOs for 5V and 3.3V. **Figure 5** shows the top and bottom image of the PCB. **Figure 6** shows 3D model of the top and bottom enclosure for the PCB. I used a 3D printer to print out the enclosures and hold

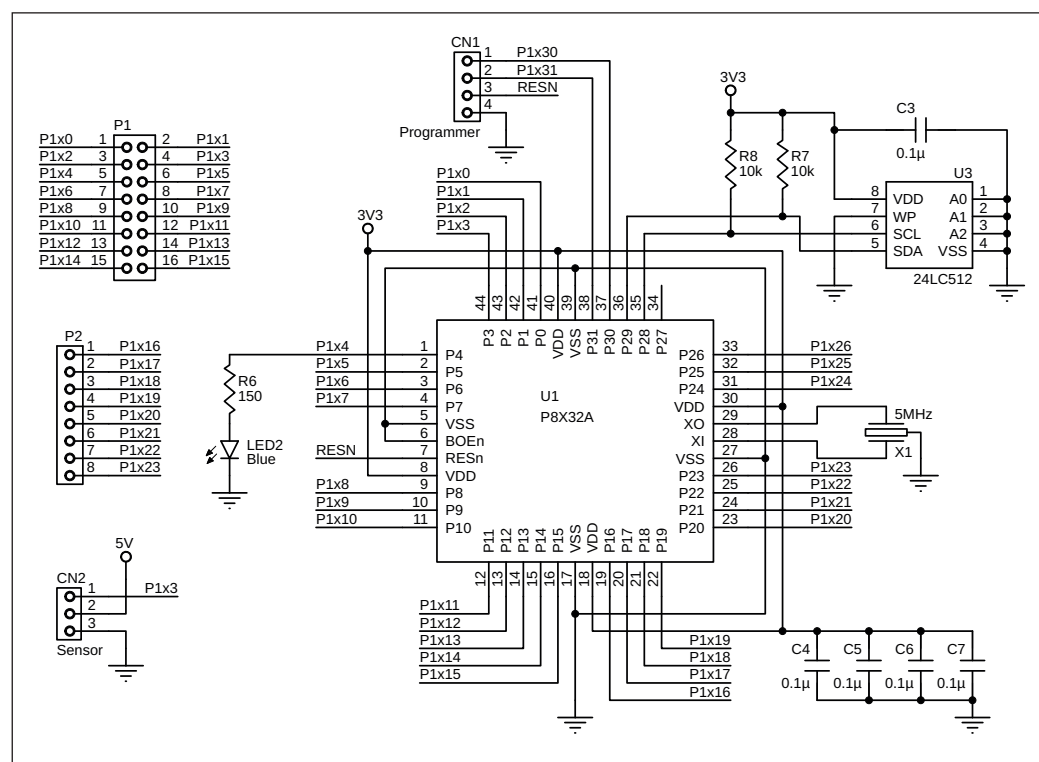


FIGURE 4A
Shown here are the basic components for the microcontroller which includes the EEPROM and 5MHz crystal plus connectors.

them together with 4 bolts and nuts. **Figure 7** shows the final working unit that has been in use since July 2016. It hasn't failed since.

The power requirements are 5V for the PING sensor and 3.3V for the other onboard components and peripherals. With that in

mind, there is a simple power circuit to step-down the input voltage of 9V. Because the input voltage is rather low, LDOs (low dropout regulators) will work just fine providing low inefficiencies. However, you can use any Propeller-based boards available on the Parallax website, for example the Propeller QuickStart or the Propeller Project Board USB. I personally prefer the latter because it provides ample prototyping space for mounting other components.

On the receiver part, I used an Android phone. Using the MIT App Inventor 2 tool [1], which is free, I could rapidly develop an app using the code blocks. **Figure 8** shows the Designer view of the MIT App Inventor 2 and **Figure 9** shows the Blocks Editor View. If you are new to MIT App Inventor, I encourage you to look at some of the tutorials on their resource page. This will give you a sense of what tools will be available for you as you start coding using their blocks. Coming back to my app, as I like to keep it simple, I only used less than 10 components. So, that's all on the main building blocks for this system. Now, let me go deeper into each of the sub-system.

CONTROLLER

The programming language I used for the controller was SPIN by Parallax for the Propeller 1 microcontroller. My code for this

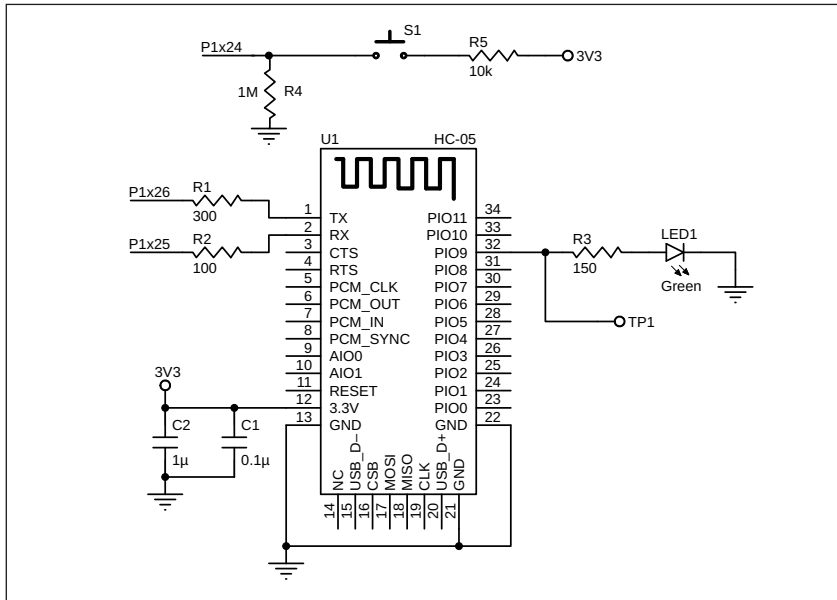


FIGURE 4B

Shown here are the HC-05 Bluetooth module and an optional tactile switch for the microcontroller as user input.

FIGURE 4C

This is the power circuit with LDOs for 5V and 3.3V.

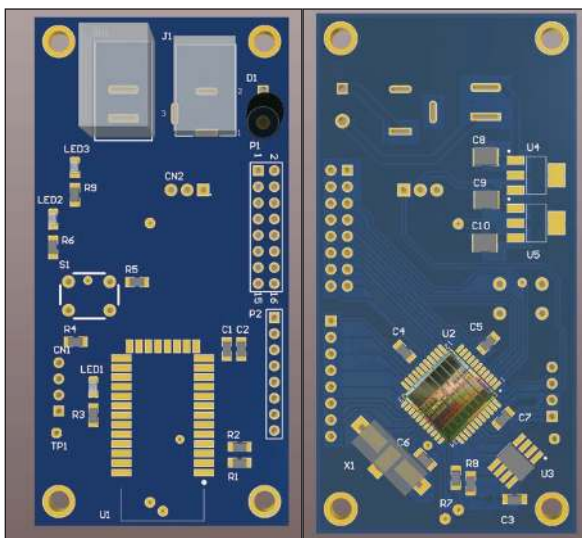
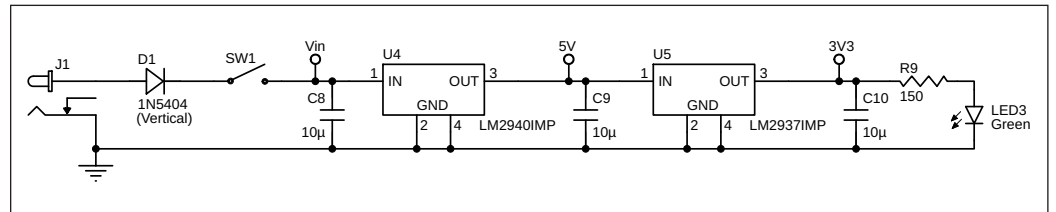


FIGURE 5

This is the top (left) and bottom (right) image of the PCB from my CAD.

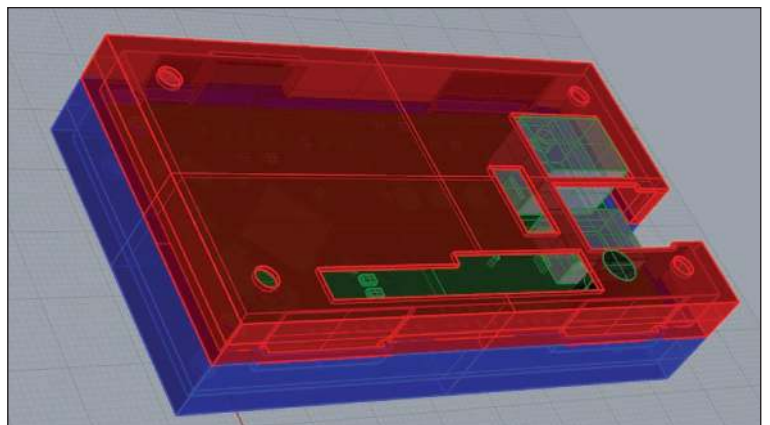


FIGURE 6

This is the 3D model of the top (red) and bottom (blue) enclosures for the PCB. I used a 3D printer to print out the enclosures and hold them with 4 bolts and nuts.

project available on the *Circuit Cellar* article code and files download webpage. Even though the “.spin” files are in a plain-text format, you will need a compiler to compile into bytecode. I usually use the Propeller Tools for Windows by Parallax. You can download and install the Propeller Tools software for Windows from Parallax’s website [2]. If you are not using a Microsoft Windows machine, you can also download PropellerIDE, which runs on Linux/Ubuntu, OS X and Windows.

The main file is #TingkatR1v1.spin. In a

normal .spin file, there are five major sections: CON (Constant blocks), OBJ (Object blocks), VAR (Variable blocks), PUB (Public method blocks) and PRI (Private method blocks). For this project, I don’t need the VAR because I only needed the local variables within the PUB Main section. In the CON section, I declared the required _clkmode which is the application-defined clock mode and since the board uses a standard 5MHz crystal, the _xinfreq which is the application-defined external clock frequency was declared as 5_000_000.



FIGURE 7
This is the final working unit that has been in use since July 2016. It has not failed since.

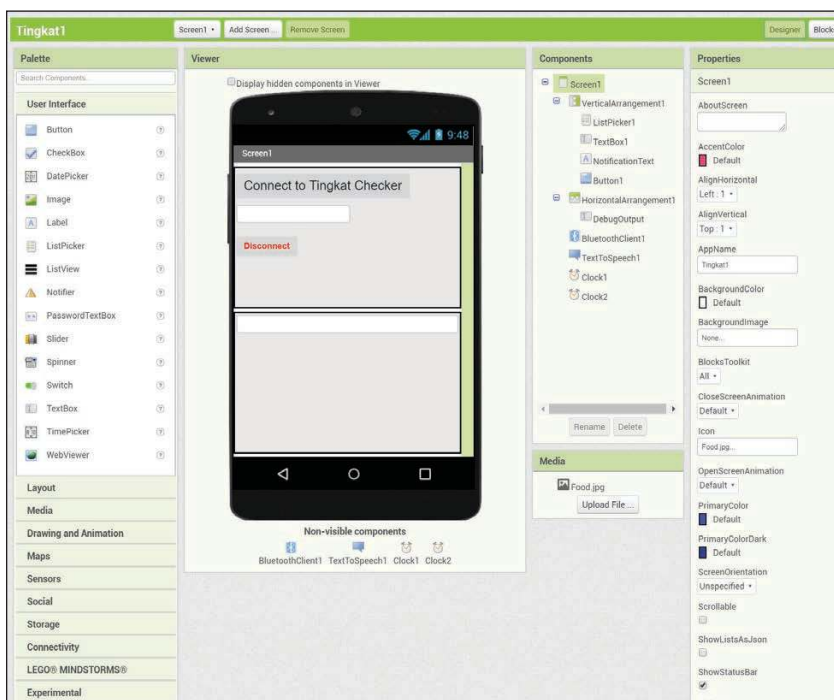


FIGURE 8
Designer view of MIT App Inventor 2

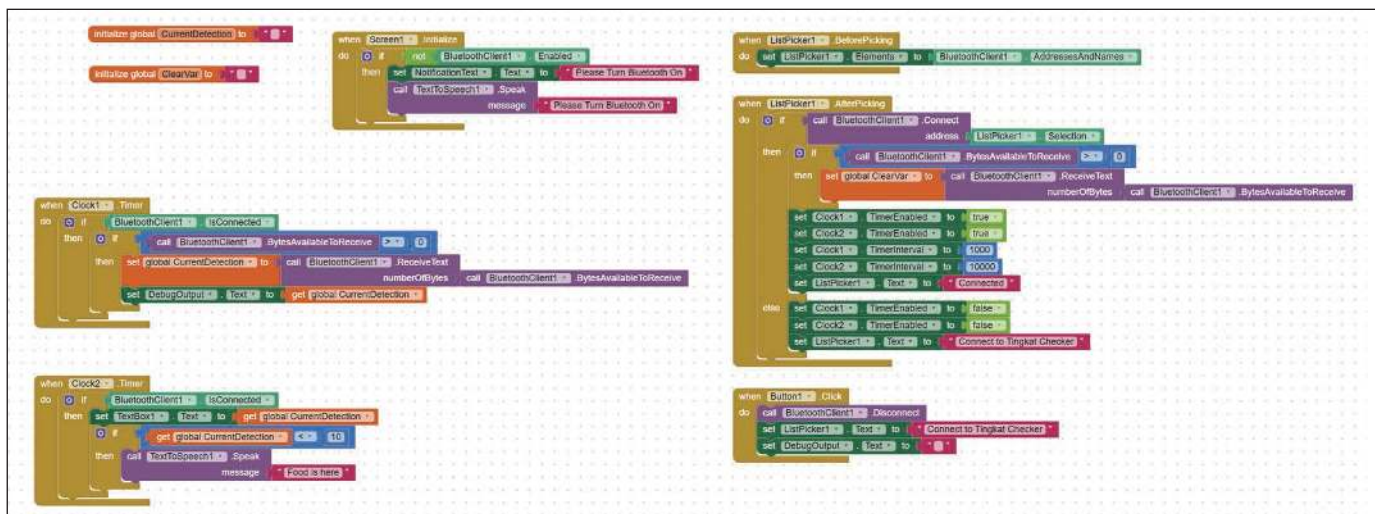


FIGURE 9
Blocks Editor view of MIT App Inventor 2

```
CON
    _clkmode = xtall + pll16x
    _xinfreq = 5_000_000
```

Next, we declare the I/O pin from the ultrasonic sensor's connection to the Propeller. The connection is on pin 3.

```
_ping = 3
```

The final declaration would be for the Bluetooth module's connection to the Propeller.

```
BTComm_Tx = 26                '(Transmit from Bluetooth Module)
BTComm_Rx = 25                '(Receive from Bluetooth Module)
BTComm_Baud = 19_200         ''Baudrate
```

The next section, OBJ, we will need two object files. These are also downloadable from Parallax's OBEX website [3]. Most of these objects were written by members of the Parallax community. I, myself, have contributed a few of them.

```
OBJ
    Ultra      : "Ping.spin"                ''Connects to ultrasonic sensor
    BTM        : "FullDuplexSerialExt.spin" ''Using UART to communicate with HC-05
```

The PUB will hold the Main section which are the main codes. Notice the "| scannedObj"? That is the local 4-bytes variable within the Main section.

```
PUB Main | scannedObj
    BTM.Start(BTComm_Tx, BTComm_Rx, 0, BTComm_Baud) ''Initialises the Bluetooth
    Pause(500) ''Pause for 500 ms
    Repeat
        scannedObj := Ultra.Centimeters(_ping) ''Get the measurement
        BTM.Dec(scannedObj) ''Sends out to the Bluetooth module to receiver
        Pause(2200) ''Pause for 2.2 secs
```

Finally, for the PRI section I usually like to add this function to most my projects because it converts the system clock into milliseconds which becomes very convenient to use throughout the code. It was initially written by a member of the Parallax community, Jon McPhalen.

Depending on your personal needs, you can make the necessary changes or add additional sensors for other locations. Say if you have 2 ultrasonic sensors, you will then declare two objects like this:

```
CON
    _ping1      = 3                ''Assigning sensor 1's I/O pin to pin 3 of MCU
    _ping2      = 4                ''Assigning sensor 2's I/O pin to pin 4 of MCU

OBJ
    Ultra[2]    : "Ping.spin"
```

And in your PUB Main section, use them like this:

```
scannedObj1 := Ultra[0].Centimeters(_ping1)
scannedObj2 := Ultra[1].Centimeters(_ping2)
```

ABOUT THE AUTHOR

Kenichi (ken@kenichikato.com) currently works as the Head of Research & Development (Software) in a Singapore local SME. Previously, he worked as an educator and engineer in Singapore Institute of Technology. His passion has been in software development since the late 80s with designing embedded systems and robotics (humanoids) from the mid-2000s.

RECEIVER

Now onto the receiver unit. I used Google Chrome as my default browser. Navigating to appinventor.mit.edu [1], you will see the home page of MIT App Inventor 2. Click on the "Create App!" button and it will direct you to the login and account creation page. For me, I logged in using my Google account. You will need to create one if you do not have a Google account. Next, it will direct you to your project listings. If you are new to MIT App Inventor, you won't see any project listings yet. So, you could either create a new project or load my ".aia" project file and modify from there. Just to give you a brief introduction to MIT App Inventor, I will show you how to add components from the Palette.

In the Designer interface, you will be able to drag and drop the types of user interface, layout, sensors and so forth onto the main screen called

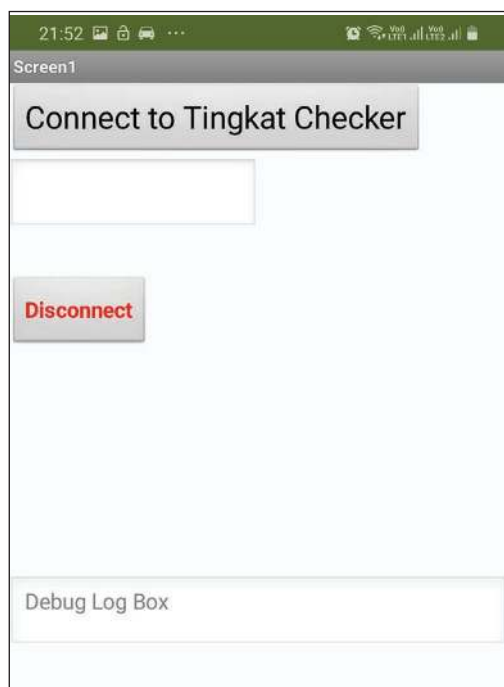


FIGURE 10
Once you launch the app as here, the first thing is to connect the app to your Bluetooth device. There is a button “Connect to Tingkat Checker” that allows the app to bring up the list of Bluetooth devices already paired with your phone.

Screen1 by default. This is what you will see when you first launch the app on your phone. You will be able to add multiple screens depending on your needs. First, I selected the Bluetooth client component from the

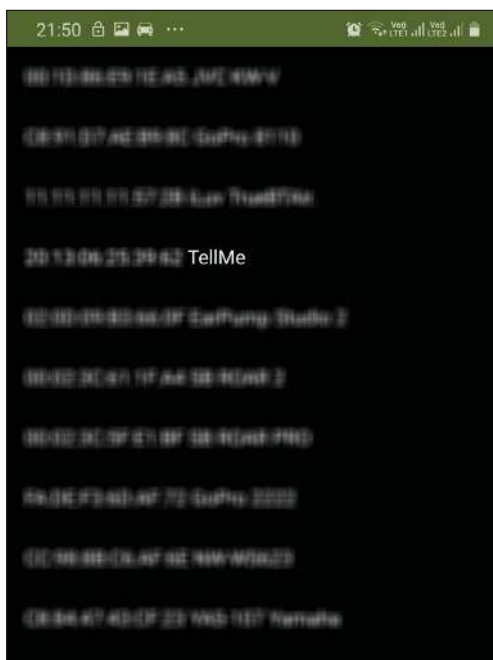


FIGURE 11
If you already have your device paired, upon clicking the button, the app will bring up the list of devices as shown in Figure 11. Since I was using a Bluetooth module from my previous project, it was renamed as “TellMe” instead of the default HC-05. (Other devices have been blurred out here for privacy sake).



N INTERNATIONAL

Designing with Passion. Engineering with Precision.



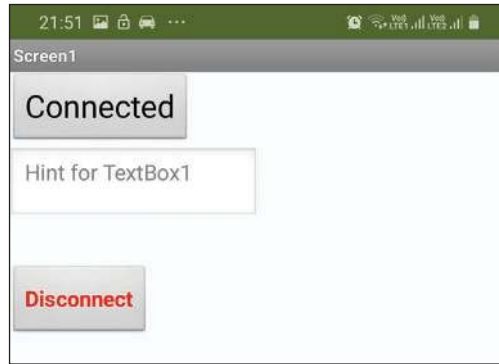
Working together to design the ideal display for your application. Let's get started on your next project.



nhdisplay.com/cc5m

FIGURE 12

When I tap on my Bluetooth device, the button will be changed to "Connected" as shown here. Notice the text box at the bottom that initially showed "Debug Log Box" has changed to "373". This "373" is the reading that was received from the controller which is the 373 cm between the ultrasonic sensor and the object.

**FIGURE 13**

The "373" reading that was received from the controller means that there is 373cm. The datasheet for the ultrasonic sensor states that the non-contact distance measurements are from about 2cm (0.8") to 3 meters (3.3 yards). That means there is nothing being placed on the hook shown here.

**FIGURE 14**

When food is placed on the hook as shown here, the value in the app shows 5cm (see Figure 15).

Connectivity Palette and dropped it onto the Viewer. This is needed to connect to our HC-05 Bluetooth module on the controller.

Next is the Text-to-Speech component. This will inform us that the "food is here" via the text-to-speech feature. I have added two Clock components which are timers. Clock1 is used to check if there are any data received in the Bluetooth client in the app. Clock2 is used to define the duration to announce "Food is here" when food was placed in the assigned location. Next, I'll show you some screen captures as the app is launched and explain the process flow. Hopefully this will help you understand the code in the Blocks Editor.


USING THE APP

Once you launch the app as shown in **Figure 10**, the first thing is to connect the app to your Bluetooth device. There is a button that is labelled "Connect to Tingkat Checker." This allows the app to bring up the list of Bluetooth devices already paired with your phone. Therefore, you will need to ensure that the Bluetooth module (HC-05) is already paired with your phone before it can be listed. If you already have paired, upon clicking the button, it will bring up the list of devices as shown in **Figure 11**. Since I was using a Bluetooth module from my previous project, it was renamed as TellMe instead of the default HC-05.

If you are going to use the HC-05 module, please look at its datasheet under "Set/inquire device's name" section for this feature. Back to the list of Bluetooth devices, once I tapped on my Bluetooth device, the button will be changed to "Connected" as shown in **Figure 12**. You will notice the text box at the bottom that initially showed "Debug Log Box" has changed to "373." This "373" is the reading that was received from the controller which means there is 373cm between the ultrasonic sensor and the object.

Since the datasheet for the ultrasonic sensor states that the non-contact distance measurements are from about 2cm (0.8") to 3 meters (3.3 yards), that means there is nothing being placed on the hook (**Figure 13**). However,

the debug textbox was just to show the actual readings coming from the controller since the retrieval was timed at every 1,000ms (1 second) by Clock1. Clock2 is assigned to activate every 10,000 ms (10 seconds). It will first transfer the value received from the controller and place it into the TextBox1 then perform a verification to see if it is below the value 10 (meaning 10cm). However, when food is placed on the hook as shown in **Figure 14**, the value shows 5cm as shown in **Figure 15**. This will then trigger the Text-to-Speech component to say "Food is here" every 10 seconds until someone removes the food from the hook!

On the *Circuit Cellar* article code and files webpage, I have provided the code, 3D models for the enclosure in .STL format and the Gerber files with the bills of materials if you wish to produce the same PCB as mine. My device has been in operation since July 2016 and it is still in good working condition today. By using this simple concept, you can also create your own proximity sensors around your house as home surveillance or a door response system to do something when someone is at your door. The possibilities are endless. So, create your very own mobile notification system today! 

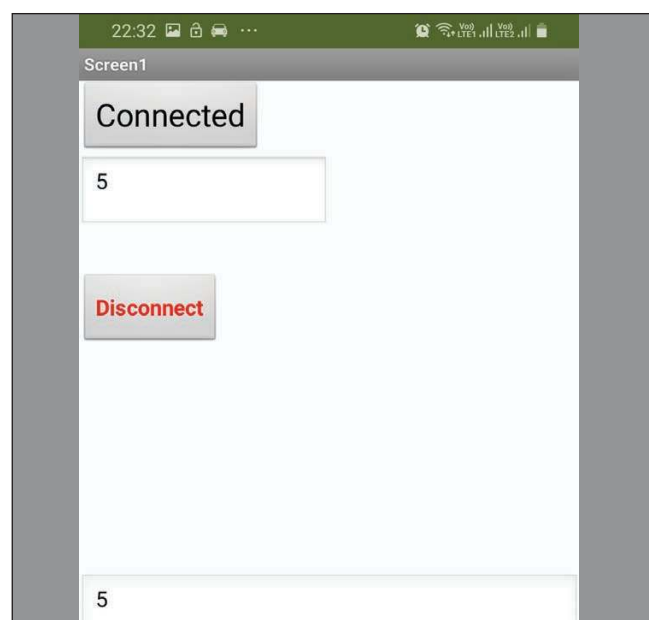


FIGURE 15

The "5" appearing on the app means that the food has arrived because it is 5 cm from the sensor. This triggers the Text-to-Speech component to say "Food is here" every 10 seconds until someone removes the food from the hook!

For detailed article references and additional resources go to:

www.circuitcellar.com/article-materials

References [1] and [3] as marked in the article can be found there.

RESOURCES

Parallax Inc. | www.parallax.com

Guangzhou HC Information Technology | www.wavesen.com

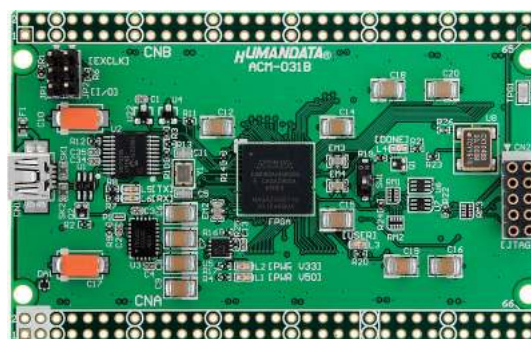
FPGA Boards HUMANDATA from JAPAN

SAVING COST=TIME with readily available FPGA boards

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

INTEL ACM-031 MAX 10 U169 FPGA board

5V I/O support



MAX 10 5V I/O CDC UART RoHS 10

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

ACM-031 is an FPGA board with Intel high-performance FPGA MAX 10. It's compact and very simple. 5V single power supply operation.

XILINX XCM-026 Spartan-7 FGGA484 FPGA board

5V I/O support



Spartan-7 5V I/O RoHS 10

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

XCM-026 is an FPGA board with Xilinx high-performance FPGA Spartan-7. It's compact and very simple. 5V 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/CC20C



Intro to Ardupilot and PX4 (Part 2)

Building the Drone

FEATURES

In Part 1 of this article series, Raul gave an overview of a drone design using open-source autopilot platforms. Here in Part 2, he goes step by step and describes how to build, configure and test a DIY quadrotor drone with the PX4 platform.

By
Raul Alvarez-Torrico

In Part 1 of this article series last month, I discussed the general architecture of a DIY multirotor drone and its main hardware and software components. I also gave a general introduction to the Ardupilot and PX4 platforms, discussing some examples of supported flight controller hardware and vehicle types, as well as available ground control software from both platforms. Additionally, I provided a general introduction to the MAVLink protocol used to communicate vehicles with ground control stations in both platforms. In this second part, I will discuss the main steps involved in the building and configuration of a DIY quadrotor with the PX4 platform. It will not be a complete tutorial, but a review of the most important steps involved. I will also give some tips that address potential problematic situations in the build process that are not always immediately obvious for beginners.

There are a lot of well documented tutorials out on the Internet. Consider this article a summary of steps and important tips from my own experience, for beginners who want to build their first quadrotor. At the end, I will also share additional tips on how to get started with autonomous flight by using the MAVSDK library and MAVROS package. Let's see if, with this article, I can encourage you to build your first quadcopter, and begin to experiment with autonomous flight in the near future.

Figure 1 shows an amazing wiring chart made by Jethro Hazelhurst for the Ardupilot website [1]. To see greater detail, you can check the original version by using the link in the figure caption. Analyze the picture in detail. It will be instructive if you have never built a quadcopter before.

The build process for a typical quadrotor—like the one I will show as an example in this article—is practically the same for both the PX4 and Ardupilot platforms. For this summary, I will use the PX4 platform to explain the process. So, what steps are generally involved in building, configuring and flying a quadcopter? I divide the process in four sections: 1) the hardware build; 2) flight controller firmware programming and configuration; 3) Electronic Speed Controller (ESC) calibration and pre-testing; and 4) flight test(s). Within each of those sections are several steps. The following development assumes you are already familiar with the main parts used to build a multirotor. Please see Part 1 of this article series (*Circuit Cellar* 357, April 2019) for more details.

So, here is one of the first tips I would give to anyone doing their first quadrotor build.

Tip #1: Never rush with any steps involved in the process, always double-check everything you do, and when in doubt, it is always better to do further research or ask anyone who's more knowledgeable before proceeding.



THE HARDWARE BUILD

Step 1: The Frame. You'll generally begin by building the frame. Frames come with a build instructions chart, or you can easily find one on the Internet. Just don't tighten the screws hard the first time. I highly recommend that you build the frame first, just to "present" it.

Tip #2: For the first builds, it is better not to tighten the screws tightly at first, or to use threadlocker glue. Leave that for the final steps, after you are sure everything is correct and in place.

Step 2: ESCs and brushless motors. For connecting brushless motors, ESCs and battery power supply lines, "banana" type connectors are generally used. Although the S500 frame I used for my example build (Figure 2) comes with a power distribution board "embedded" in the bottom plate of the frame (which is just a double-sided PCB used also as part of the frame structure), for beginners I would recommend using a separate power distribution board (PDB).

The reason for that is because it's always possible to mess up with the soldering needed in the PDB. If this happens, it is cheaper and easier to replace the separate PDB than to replace the frame's bottom plate. A typical quadrotor of this size will consume around 15A in total while hovering in soft wind and without payload. But the total current can easily peak up to the double (if not more), when carrying a payload, doing aggressive maneuvers or flying in severe wind conditions.

Tip #3: All soldering related to the propulsion system must be done carefully, tidily and robustly. Those connection points will carry high currents, so they must have good electrical continuity. There must also be the best isolation possible between the positive and negative terminals.

As a safety measure, always check the absence of electrical continuity between every pair of "+" and "-" pads on the PDB. You want to be sure there isn't a short circuit in it, due to the soldering process. Some ESCs, brushless motors and power distribution boards come with bullet connectors already soldered. If not, you must solder them by yourself. Figure 2 shows the ESCs, motors and PDB embedded in my S500 frame, with all the necessary connections. ESCs and cables will later be secured to the frame by using plastic zip ties.

Step 3: Flight controller, RC receiver and GPS module. The spinning motors and propellers in the quadrotor generate a certain amount of mechanical vibration

that is detected by the accelerometer and gyroscope sensors in the flight controller. This vibration is noise to the sensors, and can potentially affect the quadcopter's stability. To minimize it, the flight controller can be installed on top of an anti-vibration mount, as shown in Figure 3. I initially used rubber bands to fix the flight controller to the anti-vibration mount. In the final steps, it will be fixed with double-sided tape. Figure 4 shows the motor layout for a "Quad X" configuration. Each motor has a given number, position and rotation direction, so it is critical to connect their corresponding ESC signal cables to the flight controller's outputs in the given order.

Tip #4: The correct layout of all motors in the frame and connection of ESC input signal cables to correct output pins in the flight controller are of utmost importance. Failing

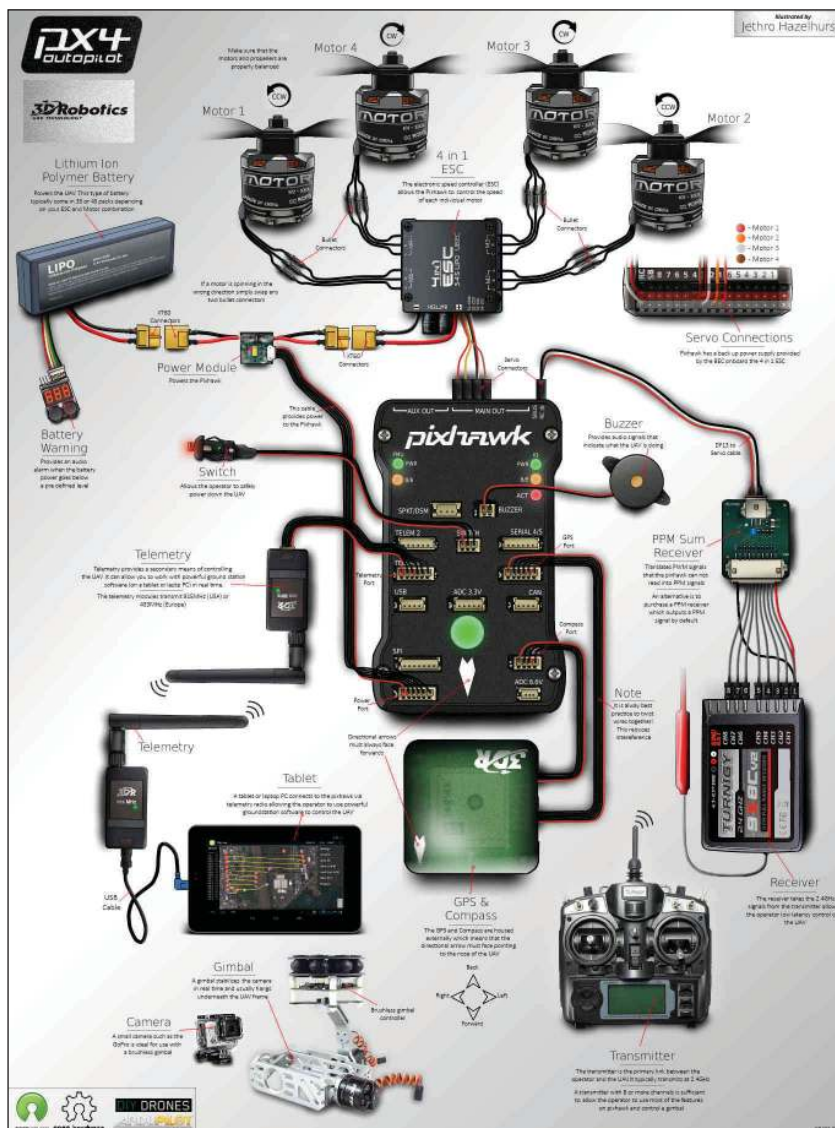


FIGURE 1 Advanced Pixhawk quadcopter wiring chart [1]



FIGURE 2
S500 frame with motors, ESCs and power distribution board

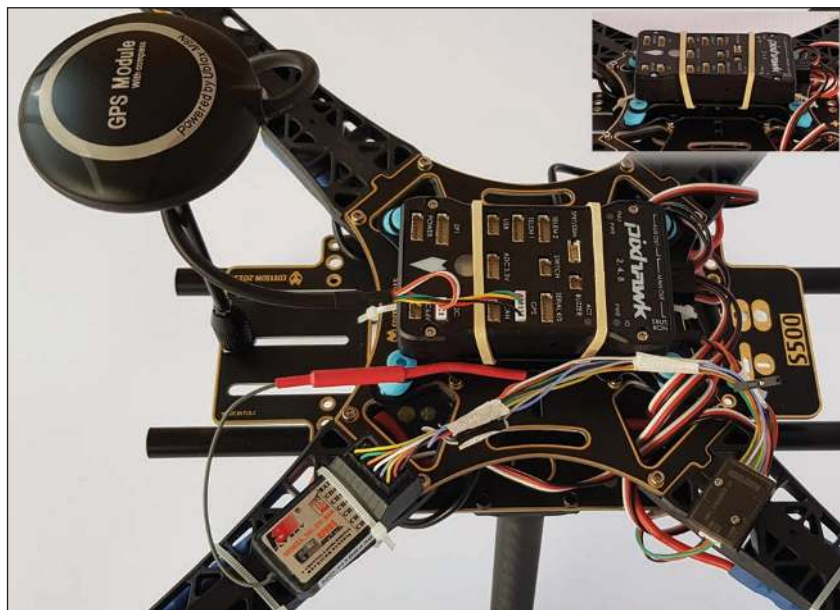


FIGURE 3
Flight controller, RC receiver and GPS module

ABOUT THE AUTHOR

Raul Alvarez-Torrico has a BEng in electronics and is the founder of TecBolivia, a company offering services in physical computing and educational robotics in Bolivia. In his spare time, he likes to experiment with wireless sensor networks, robotics and artificial intelligence; he is also committed to publishing articles and video tutorials about embedded systems and programming in his native language (Spanish), at their company's website www.TecBolivia.com. You may contact him at raul@tecbolivia.com.

to do this can make the quadcopter behave erratically and ultimately crash.

For this build, I used a conventional 6-channel remote control (RC) system with parallel analog PWM outputs. The Pixhawk controller, however, just accepts serial PPM-Sum or S.Bus input signals, which provide all RC channel signals via a single serial digital output. You can purchase an RC system with a serial output receiver suitable for connecting directly to the Pixhawk. Nevertheless, if you have a system with parallel analog PWM channels, low-cost parallel analog PWM-to-digital serial converters, such as the one I used for this build, are available.

The connection of the GPS and external compass module is straightforward. The module has two connectors. The wider one is for the GPS receiver, and the narrow one for the digital compass. The digital compass is highly sensitive to magnetic interference. With that in mind, a stand for the GPS module is generally recommended to avoid interference from the rest of the electronics, especially the power system. Figure 3 shows the flight controller, RC receiver and GPS module with its stand.

Step 4. Power module, battery and alarm. The power module connected to the flight controller and the PDB, as well as the LiPo battery in place with its low voltage alarm, are shown in **Figure 5**. The connection between the flight controller, power module, battery and propulsion system are illustrated clearly in Figure 1, so go back to this at any time to get a more detailed view of other connections.

To fly this quadcopter, I'm using a 5,000mAhour, 3S, 8C battery.

Tip #5: Improper use of LiPo batteries can be extremely dangerous. Before charging, connecting and using them, take the time to thoroughly research their use and care. Many resources on the Internet cover this topic in detail.

Step 5. Telemetry module, gimbal and FPV (first person view) transmitter. The telemetry module must be connected to the TELEM1 port in the Pixhawk (Figure 1). The gimbal receives power from the power distribution board. Mine can work with LiPo batteries between 2S and 6S. Gimbals generally have a signal input to manually control the camera pitch angle. This input can be connected to any unused output in the RC receiver, preferably one from a channel associated with a knob in the RC transmitter.

Once the gimbal is powered on, the camera should be automatically stabilized, and the pitch angle controlled with the associated knob in the RC transmitter. Drone cameras

have a video signal output to be connected to the FPV video transmitter (powered also from the PDB). Usually, the FPV system is separated from the drone system itself (Figure 1).

Step 6. Finishing the hardware build. I usually connect the battery once at the end of the hardware build, to be sure everything is okay and nothing is getting hot due to a short circuit, especially in the propulsion system. Then, I proceed to tighten all screws with threadlocker glue. I take them out one by one from the frame and motors, apply threadlocker and tighten them well in place again. Threadlocker needs at least 24 hours to properly cure. I use Loctite Threadlocker Blue, which is recommended for multirotor builds.

Tip #6: After your first flight and before flying again, always check that the motor and frame screws are still tight. Sometimes they can loosen, because parts tend to settle further with mechanical vibration, and the screws would need to be tightened again. A loose motor or frame part can make the quadcopter vibrate too much and make it crash.

THE FIRMWARE

Although I'm using PX4 firmware for this build example, the same quadrotor setup can also be flashed with Ardupilot firmware by following a similar set of steps, such as the ones outlined as follows:

Step 1. Firmware flashing. It is really easy to flash the firmware to the flight controller by using ground control software. With QGroundControl, you can do that by going to the "Firmware" submenu located in the "Vehicle Setup" menu (**Figure 6**) and following the instructions given there. After flashing the PX4 firmware, the frame type must be configured in the "Airframe" submenu, under the "Quadrotor x" option. I chose the "S500" for this build (**Figure 7**).

Step 2. Sensor calibration. Next, from the "Sensors" submenu, we calibrate the compass, gyroscope and accelerometer, and also level the horizon (**Figure 8**). Plenty of guidelines are given in each step by QGroundControl, itself.

Step 3. Radio setup. The radio calibration, done in the "Radio" submenu (**Figure 9**), ensures that the minimum and maximum values for each RC channel are correctly set in the flight controller. These values generally vary from one RC transmitter to another.

Step 4. Flight modes configuration. In the "Flight Modes" submenu (**Figure 10**), we can configure up to six different flight modes. With a 6-channel RC, I generally assign channel 5 for controlling the flight modes and channel 6 for the "Kill switch" function. The three most

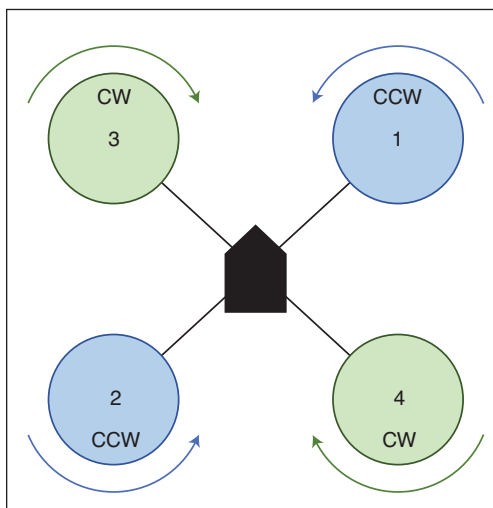


FIGURE 4
"Quad X" configuration layout

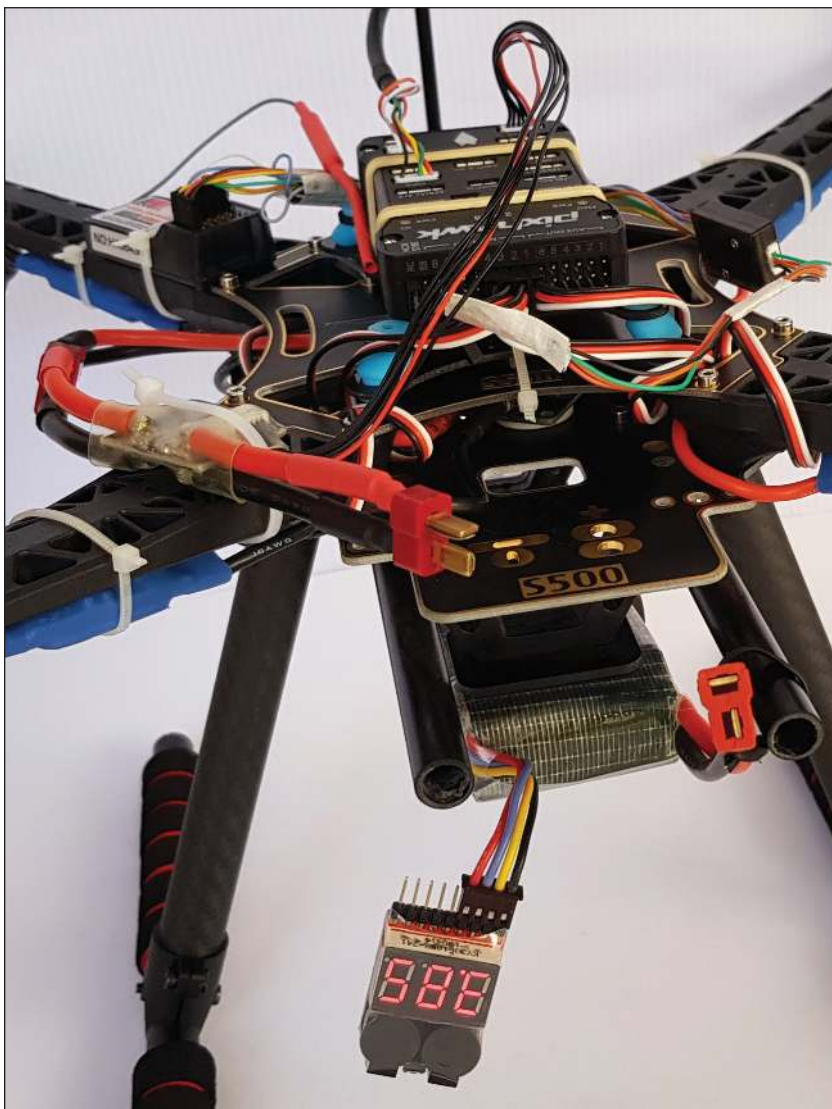


FIGURE 5
Power module, battery and low voltage alarm

useful flight modes I can recommend for beginners are Altitude, Position and Return.

Altitude automatically stabilizes the quadcopter's altitude by using the barometric pressure sensor (altimeter), but it doesn't stabilize position in the horizontal plane. This is because this flight mode does not use the GPS receiver. It requires you to stabilize the quadcopter in the horizontal plane manually, which can be tricky for beginners, especially with strong winds. In this mode, the quadcopter can be armed without the need for GPS fix (useful for indoor motor testing).

Tip #7: Never fly a quadcopter indoors if you are a beginner, or if there are people around. For your flight tests, always use big, open fields without people, animals or obstacles around.

Position is the preferred flight mode I recommend for beginners, especially for flying in windy conditions, because it uses the barometric pressure sensor and also the GPS receiver for stabilizing the drone vertically and horizontally. But it doesn't allow arming the motors and drone take-off without a good GPS fix, which you won't get unless the GPS receiver has good sky visibility.

Return flight mode automatically returns the drone to the home position, and is useful if, for some reason, you lose control, and just want the flight controller to take over and land the drone by itself at the original take-off spot.

Finally, it is also important to configure the "Kill switch" function to turn off the motors, especially when something goes awfully wrong. For example, if the drone falls down to the ground and the motors are still spinning, even with the throttle stick at zero—or when the drone is out of control and it is better to switch it off (and crash it) before anything worse happens.

Step 5. Power setup. In the "Power" submenu, the flight controller's battery power monitoring function can be configured, so it can accurately estimate the remaining power and flight time. I generally just configure the "Number of cells" parameter ("3S" for a 3S battery), and leave the rest at their default values. At first, it's better not to change the remaining parameters, until you understand well how these parameters affect the way the flight controller does the aforementioned estimations.

Step 6. Failsafe actions. A "failsafe action" is a predefined safety measure the quadcopter will take when some types of failures occur. Examples of failures include when the battery is at its minimum, when the RC signal is lost, or when the data link (telemetry) is lost. Available options for safety measure configuration range from doing nothing or giving a warning, to landing the drone, returning it to home or hovering it in place. Failsafe actions can be configured in the "Safety" submenu. The default options are also a good starting point.

ESC CALIBRATION & PRE-TESTING

One last important step to take before the first flight is ESC calibration (available from the "Power" submenu). ESC calibration ensures that all motors respond equally to the available throttle range from the RC transmitter. This is accomplished by teaching the ESCs to recognize the minimum and maximum PWM values for zero and full throttle.

Tip #8: Before attempting to take off for the first time, check the rotation direction

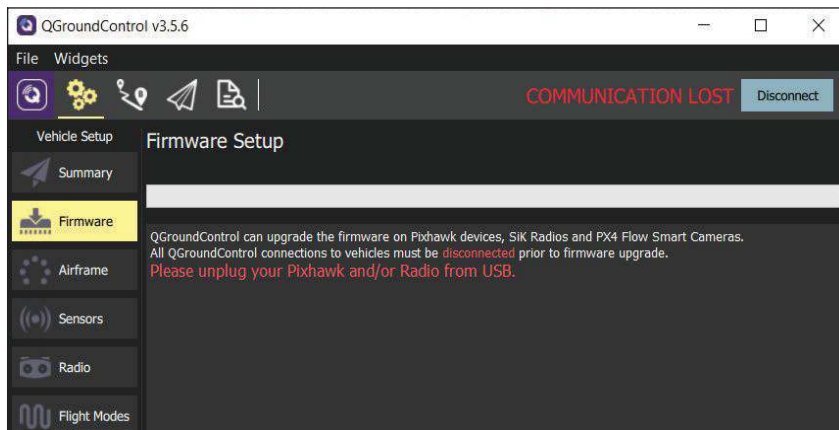


FIGURE 6

Vehicle setup menu and firmware flashing submenu

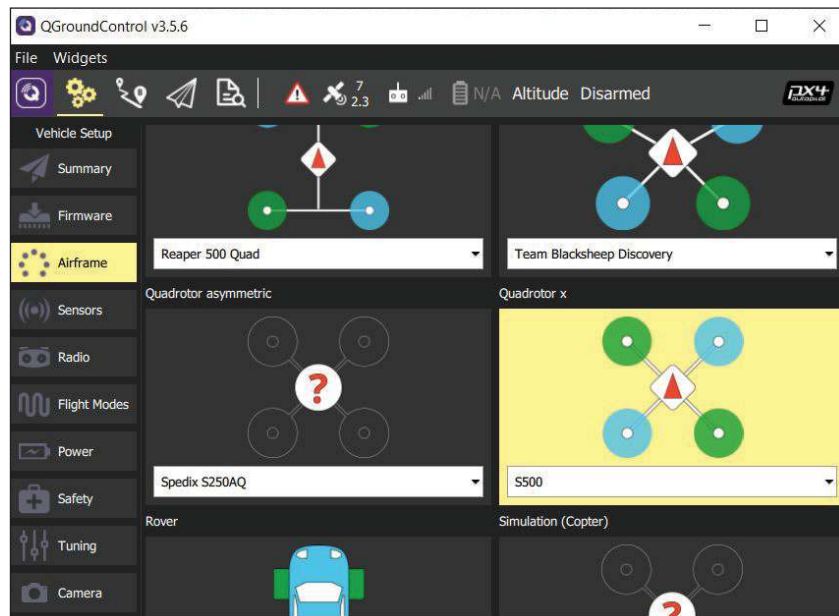


FIGURE 7

Airframe setup

of all motors according to what's specified in the layout for the selected configuration. Also check the correct propeller mounting. Motors spinning in the wrong direction, or propellers incorrectly mounted will cause erratic behavior, ranging from not being able to take off at all, to taking off and spinning uncontrollably in the horizontal plane, or taking off and turning upside down and ultimately crash.

If a motor spins in the wrong direction, swap two of the three motor cables connected to the ESC (any two of the three available) to change the rotation direction. Also verify the proper propeller mounting, with the upper and lower cambers correctly positioned, and the leading edge "cutting" right into the air, in the right rotation direction.

FLIGHT TESTS

Until this point, for safety reasons, always do all motor pre-tests without propellers, and put the propellers on the motors just before the first flight. Pick an open field with no people, animals or obstacles around. If there are natural or man-made obstacles (such as tall trees, hills, buildings or towers), the GPS receiver will take longer to get a fix, or worse yet, it will lose it intermittently during flight.

Tip #9: Always try to do your flight tests in the morning, when the wind is generally calmer than in the afternoon. Avoid flying in relatively strong winds until you develop good piloting skills. You can carry a pocket anemometer to measure wind speed before taking off.

The maximum wind speed in which a quadrotor can be safely flown depends on the

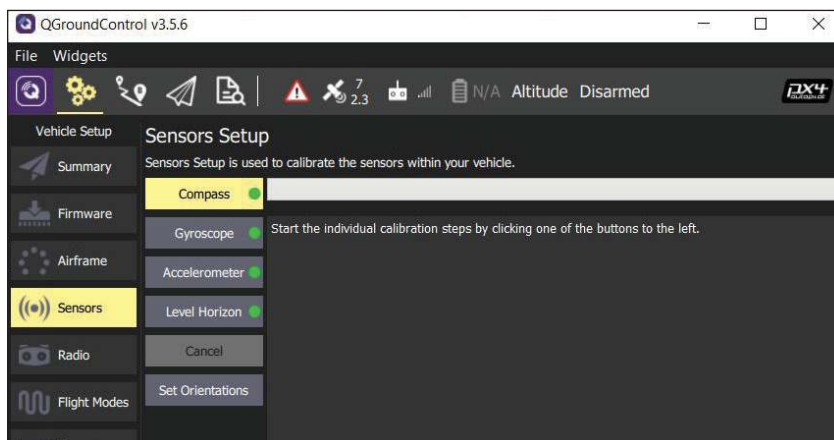


FIGURE 8
Sensor calibration

LOOKING TO
**ADVANCE
YOUR
CAREER?**

Be a part of one of the
**top Electrical Engineering
programs in country**
and experience the
Bearcat Promise!

University of
CINCINNATI | ONLINE

ucengineer.online

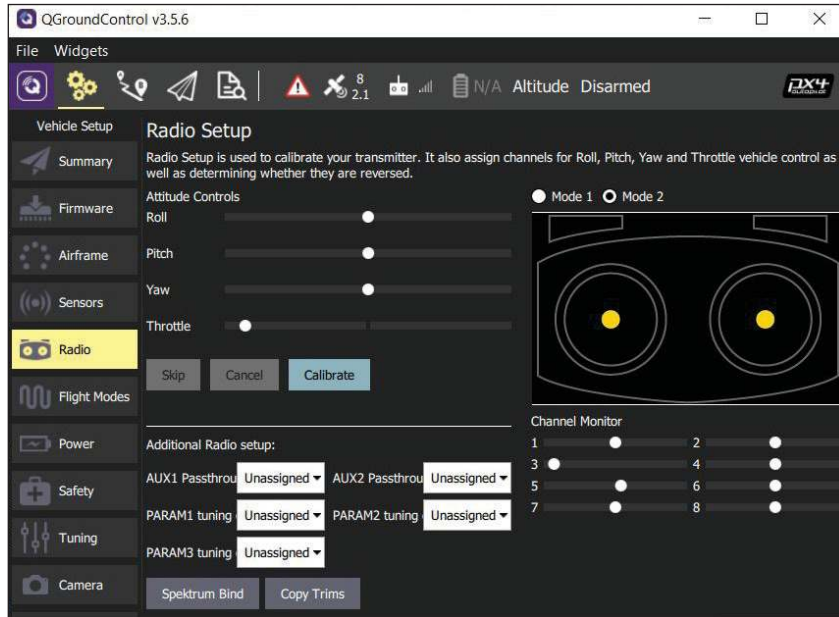


FIGURE 9
Radio setup

quadrotor's maximum speed. A good rule of thumb is to set the maximum wind speed at two-thirds or less of the quadrotor's maximum speed. For example, if the quadrotor's top speed is 80mph, two-thirds of that is about 53mph. This, according to the rule, will be the maximum recommended wind speed to fly the quadcopter safely. Last but not least, don't forget to comply with all regulations in place for flying drones in your town, to avoid legal problems.

From a functional standpoint, PX4 and Ardupilot are not very different. The processes of flashing firmware and configuring parameters are almost the same in both platforms. Besides, both are MAVLink-compatible and generally work well with the same quadcopter setup, as long as the flight controller is compatible with both platforms.

After you have flashed and configured your quadcopter with, say, PX4, and flown your drone for some time, you can try Ardupilot if you wish. Every step described in the Firmware Flashing and Configuration section of this article will also apply, with minimum differences. You can even try to use Mission

Planner or APM Planner this time. These are two ground-control software applications from the Ardupilot ecosystem, and are similar to QGroundControl.

MAVSDK AND MAVROS

MAVSDK is a MAVLink library from the PX4 ecosystem that's written in C++ and has bindings to Python, Swift and Java. It also has bindings to JavaScript, C# and Rust, though still in "proof-of-concept" stage. MAVSDK allows programmatic interaction with any MAVLink-compatible vehicle, to get general information from it, get telemetry data, send action commands like arming, take off, land or return to home, send commands to calibrate sensors and send "low level" commands to directly control vehicle movement [2]. For instance, you can control position, velocity and acceleration in three dimensions, to make the drone change its states in response to decisions made by some type of navigation algorithm.

In contrast, the MAVROS package (which is an ROS-to-MAVLink bridge, also from the PX4 ecosystem), allows the use of MAVLink communication, to make it possible for any computer running the Robot Operating System (ROS) middleware to communicate with any vehicle or ground-control station software that uses the MAVLink protocol.

In summary, MAVSDK will allow you to program a robotic drone, and MAVROS will allow you to do it straight away in an ROS environment, for a more modular, distributed, scalable and professional approach. Ardupilot has also its own MAVLink library, called Dronekit, with available Python and Android APIs. But apparently, not much work has been done lately to provide support for other programming languages.

If you want to get started with autonomous flight application development within the PX4 ecosystem, I suggest you try MAVSDK and/or MAVROS in three stages.

First, generally the easiest way to try MAVSDK is to install the Python wrapper and try some included examples with a "Software-in-the-Loop" (SITL) simulator. The C++ version installation can be a little more involved, in some cases requiring building the library from source. MAVROS is useful only if you already know how to use ROS. If you are new to ROS, perhaps you should first learn ROS, before trying MAVROS. But you still can make your drone fly autonomously by using MAVSDK. Both MAVSDK and MAVROS can connect to vehicles via serial port, UDP and TCP connections. When working with a simulation, TCP/UDP connections will usually be the way to connect our software application with the simulator.

Second, once your code runs well on simulation, you can try it on a real drone by

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

References [1] through [2] as marked in the article can be found there

RESOURCES

Ardupilot | <https://ardupilot.org>

Dronecode | www.dronecode.org

mRobotics | <https://mrobotics.io>

PX4 Autopilot | <https://px4.io>

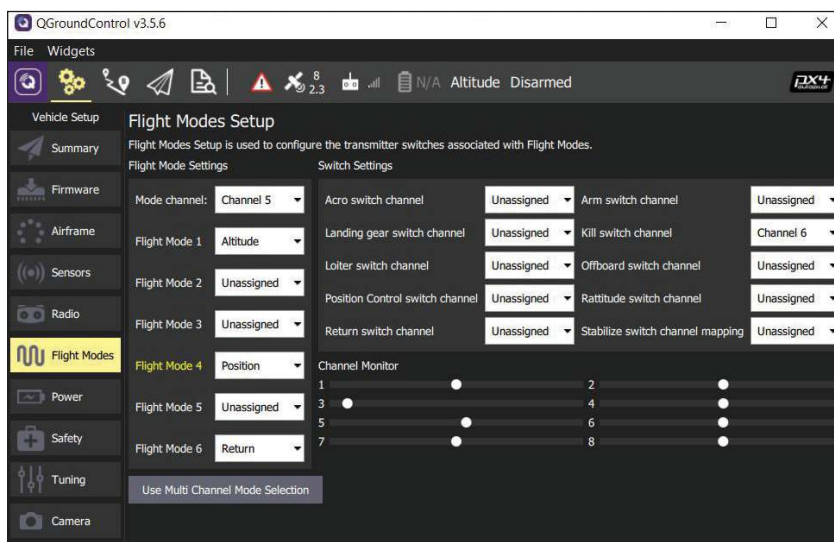
opening the telemetry module serial port from the application running on your development computer or mobile device. In this case, MAVSDK and MAVROS will communicate with your drone by using the MAVLink connection established via the telemetry modules.

Third and last, to make your drone truly autonomous, you can upgrade your system by adding a companion computer (Raspberry Pi, Odroid, Jetson Nano or others) to your quadcopter setup. The companion computer can be connected directly to the TELEM2 serial port of the flight controller to run the MAVSDK/MAVROS code that controls the vehicle, along with some other code interfacing with additional sensors and actuators especially suited for your application.

CONCLUSION

I hope the information presented in this two-part article series gave you a general perspective about the PX4 and Ardupilot platforms, and what is involved in building a quadcopter for aerial photography, or perhaps more interesting, for experimenting with autonomous flight. If you have never built a quadcopter before, I really encourage you to do it and experiment.

I'm not going to lie, you'll crash many times while learning to pilot it and experimenting with



it, but that's the normal learning process for anyone. Still, it is a lot of fun! Most excitingly, once you get a good grasp of what's involved in building and configuring a quadcopter properly, you can begin to experiment with autonomous drone applications.

In a future follow-up to this article, I will cover in more detail—and with a concrete example—how to use the MAVSDK library to develop code for autonomous drones using SITL simulation. Until the next time! [E](#)

FIGURE 10
Flight modes configuration



The NEW PicoScope®
6000E Series



A smarter scope for faster debug

- 8-bit to 12-bit FlexRes® ADCs
- 8 x 500 MHz analog channels
- Up to 16 x 1 Gb/s digital channels
- Dual 5 GS/s ADCs
- 4 GS capture memory (up to 2 GS per trace)
- 50 MHz 200 MS/s 14-bit AWG
- 300 000 waveforms per second update rate
- Free PicoScope 6 and PicoSDK software
- Serial decoding and mask limit testing
- High-resolution timestamping of waveforms
- Over ten million DeepMeasure™ results per acquisition
- Advanced triggers: pulse width, runt pulse, windowed, logic and dropout

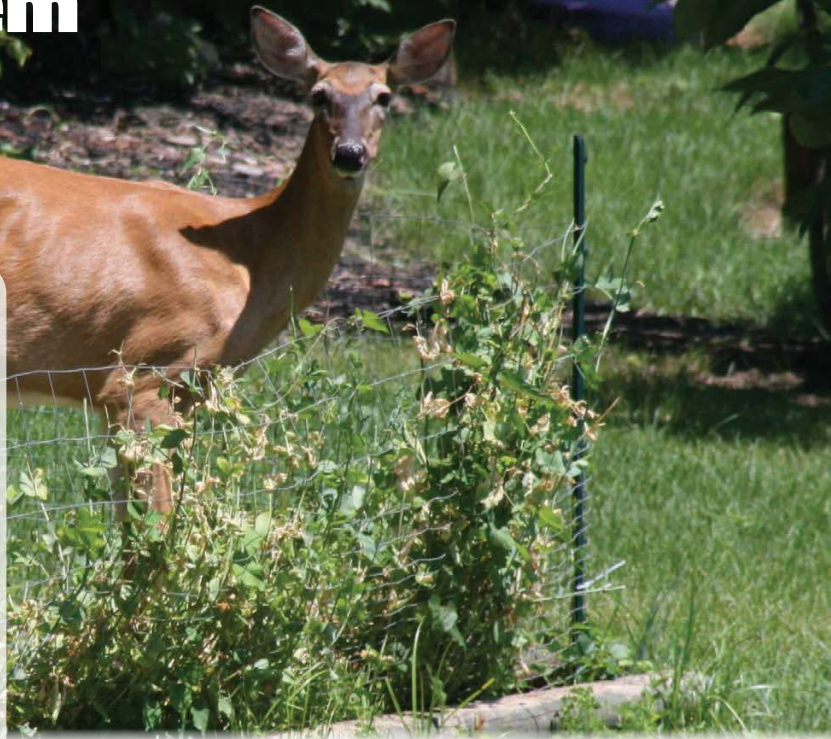
Visit www.picotech.com/A540 to discover MORE

Email: sales@picotech.com. Errors and omissions excepted. Please contact Pico Technology for the latest prices before ordering.

Build an Automated Pest Deterrent System Using Raspberry Pi

FEATURES

Garden pests are a threat to your flowers, vegetables and fruit. Learn how these Camosun College students designed an automated pest deterrent system that uses an ML-trained Raspberry Pi to detect and identify pests, and Bluetooth to communicate to its user. Their prototype base unit displays relevant information gathered by two field units.



By
Cole Gamborski, Cameron Phillips and Simon Fowler

What inspired us to choose this project? It was a conversation Simon had with his father last summer, while sitting and having lunch. His father told him how the deer had come again in the night and eaten the flowers off all 28 of his tomato plants. His father said, “Son, could you invent something to keep these pesky deer out of my garden?” Simon knew that his dad was not alone with this issue. Being a farmer himself, he had experienced just how much damage deer can do.

When considering crop security and pest deterrence, the current market for the average homeowner is oversaturated with inefficient, ineffective products. By combining machine learning (ML) with high-precision, passive infrared sensors (PIRs) and Bluetooth Low-Energy (BLE), we created a more sophisticated deterrent system that touched on each group member’s technological area of interest.

Based on our project scope and requirements, we divided our product into a four-phase system: Detection, Distinguishing, Deterrence and Disclosure.

DETECTION

Reliable and continuous scouting of your property is critical for garden and home

security. Knowing that your target is 40+ feet away and approaching allows the system plenty of time to exit sleep mode and enact its security measures. After researching the passive infrared sensor (PIR) market, we found the best choice to get this job done was the Panasonic EKMB1306112K. This high-precision PIR, made of top-of-the-line materials from a trusted manufacturer, boasts one of the densest detection zones available.

It’s able to achieve a 17-meter detection range, with as little as a 4°C temperature difference between the detected target and ambient temperature. This means that a single node could cover the average garden space of a home, or, if used close to the perimeter of the property, the node could see and begin deterring the target before it reaches your garden.

There is one downside to using PIRs—false triggering. We have gone to great lengths in the design to eliminate any possibility of false triggering. To achieve this, we started with stacked PIRs. These PIRs have a 60-degree horizontal viewing angle. We used two PIRs for each angle of approach—one above the other. To explain how this helps prevent false triggering, let’s take a common example: An outdoor security system.

You've got your motion-triggered sensor set up outside, and have left it overnight to watch over things. Then, unbeknownst to you, a moth or another photoactive bug lands on your sensor, causing your sensor to trigger constantly. You wake up the next day to find a night-long recording of nothing! What a waste of time and energy! By stacking two PIRs and requiring both to be triggered at once, we effectively reduce the likelihood of false triggering. We wondered if maybe this proactive design were not enough? This led us to the next phase of our system.

DISTINGUISHING

Let's return to the moth example. You now have moths on both of your PIRs, causing your alarm to run all night. How do you prevent false triggering in this situation? Machine learning comes to the rescue by enabling the system to detect, recognize and act judiciously instead of blindly.

Software: We had two initial contenders for our product's "brain": OpenCV and TensorFlow. We opted to use TensorFlow, because of its larger documentation base, its readily available tutorials, and—most importantly—because it is optimized for IoT and mobile applications. The latter makes it great for the Raspberry Pi family, which was important because we planned on using the Raspberry Pi 3B. However, TensorFlow had its own challenges. During install, some of its dependencies required updating. And since we were using Raspbian Lite OS, we had to install these dependencies one at a time—at times having to revert applications to older versions.

Training: Once we were able to run the basic tutorial example, the next step was to check the trained processor's accuracy. Unfortunately, Google's model was too global for our needs—it identified a raccoon as a red panda. We completed the next tutorial of retraining the processor for our own needs. This meant creating categorized folders (cougar, deer, raccoon, cat, dog, human, tractor and tree), and telling the processor where to locate them.

After our first retraining, we realized that the Raspberry Pi 3B (Pi) was not ideal for the processor-intensive task. We decided to do all future training on a desktop computer, and then copy the image over to our Pi. After retraining, we tested our model's accuracy and found it to be lacking. Results were between 60% and 76% accurate, using roughly 100 photos per category. We learned that you need at least 300 photos for consistent accuracy.

In addition to that, you need your photos to portray different perspectives of the object. Consider the appearance of your favorite dog.

If you had seen this dog only from the front at its eye level, is there any way you could say with confidence what it might look like from the side? Or from behind? For the AI to be truly effective at recognition, it needs images in different resolutions, lighting, perspectives and views.

On our third training attempt, we used more than 300 images per category, making certain to use many different elements to help flex our AI's brain muscle. However, gathering and manipulating 2,500 pictures is not easy, and it consumes time. We used websites such as Unsplash and Pexels as sources of free images. After finishing the retraining, we resumed testing and saw accuracies greater than 90% in some categories. It was a good day at the office.

DETERRENCE

We felt confident at this point to move forward with the Deterrence phase of the system. Because we intended to use sound as a deterrent, we researched how sound is perceived by different species. The average human hearing range is 20Hz to 20kHz, and most middle-aged humans can only hear in the 15kHz range. While human hearing is limited to sounds below 20kHz, other species are capable of hearing in the ultrasonic range (frequencies greater than 20kHz). For example, dogs, deer and raccoons can hear up to 40kHz, and cats can hear up to 80kHz.

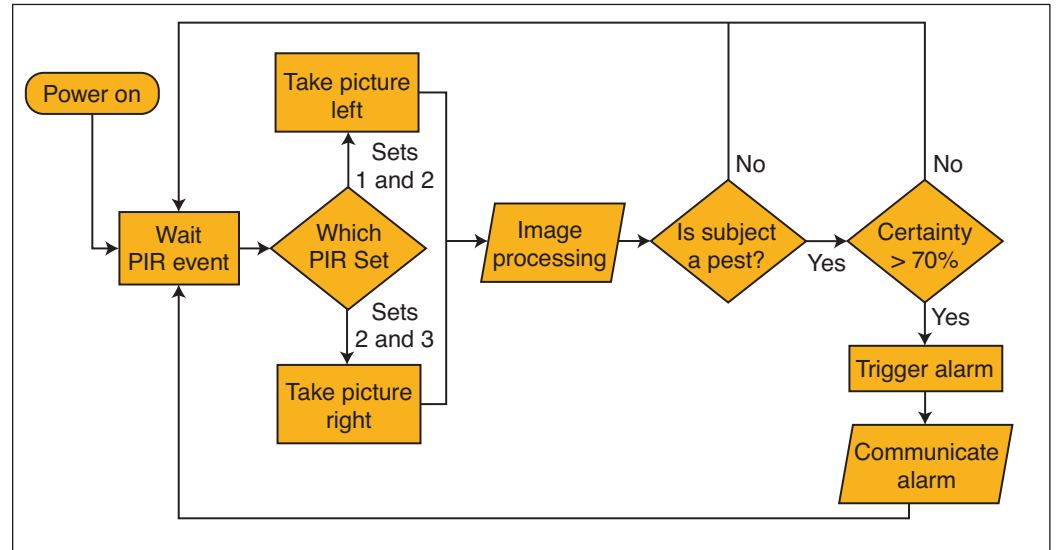
This meant that we needed a large, 10kHz audible range that would not wake you or your neighbor if an alarm went off at night. The simplest way to guarantee a sound will remain at a specified frequency is to use pure tones—waveforms such as sine or square waves. If you don't know what a pure tone is there are several free pure-tone generators available on the Internet. Simply put, they are waveforms such as sine or square waves.

Pure Tones and Man's Best Friend: While pure tones make an effective deterrence option for urban settings, we thought that rural options could use a more creative touch. After reading several articles on predatory vocalizations and how they are used to deter urban pests, we learned that the pests had one definitive predator in common: Dogs. Because there are so few cougars and bears in urban areas, pests have few, if any, encounters with them. However, the pests are more than likely to have several run-ins with dogs.

One of us (Simon) has 50-150 chickens on his farm at any given time. If they are not in their coop at night, raccoons will come to hunt them. Generally, the raccoons aren't afraid when he tries to chase them off. But if his Labrador retriever gets out there and starts barking, they immediately scatter and

FIGURE 1

This transmission flow chart begins as the system powers on. The system waits for a PIR to sense a target, and when it has detected something, it photographs the target using the appropriate directional camera. If, through image processing, the subject is determined to be a pest (with a certainty greater than 70%), the system triggers its alarm and communicates the event to the user. Otherwise, the system returns to waiting for the next PIR event.



run for the property line. Based on research, we decided to incorporate plenty of variety in the barking sounds used, and we added some other predator vocalizations for good measure.

DISCLOSURE

We were moving forward smoothly in all three of the initial phases for our system. It was time to discuss the final phase of the system, Disclosure. In the Disclosure phase, we convey important information about the alarm to the user via our mesh BLE network. BLE is on the same 2.4GHz frequency band as regular Bluetooth. However, BLE uses a different modulation system from that of regular Bluetooth. Here is what this difference meant to us: BLE has a greater potential range, but a lower over-the-air data rate.

While regular Bluetooth is limited to seven slaves, BLE can have more than seven active slaves. That means we can have a multitude of potential field nodes on the same network, covering a wide area of land (the perimeter of a farming acreage). Most importantly, BLE offers a lower power consumption (often half that of regular Bluetooth), making it ideal,

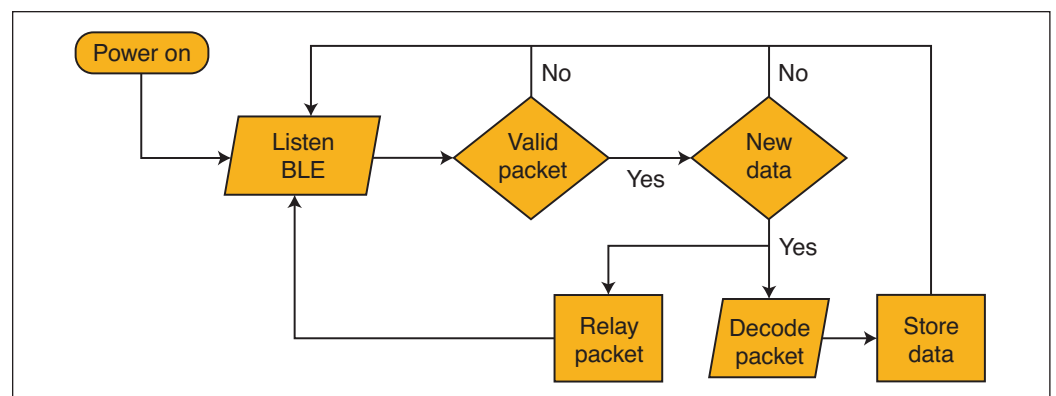
because we wanted our design to be optionally battery powered.

We had considered using Long Range (LoRa), but it was more range than we needed and had a smaller packet size. We also considered Wi-Fi, but the power required was too great to provide a long-term battery-powered option. Also, it required nodes to be within range of a Wi-Fi router, which might not be the case on larger properties.

Bluetooth Low-Energy 4.2: Once we had landed on BLE, we could begin writing code to transmit (**Figure 1**) and receive the desired data. Unfortunately, writing our own code based on the BLE protocols was too involved for our given time frame and required intensive documentation review. After some failed attempts, we opted to use a Python-based program called PyBeacon. This program is designed to advertise and scan for Eddystone-URLs. An Eddystone-URL is a 17-byte packet in the form of an HTTPS URL. Because we are not trying to send a URL, we needed to alter the provided code to fit our needs. The process of testing PyBeacon's code to discern what it deemed necessary for a valid packet took a while. But once we knew what the packet was expected to look like, we could begin

FIGURE 2

This receive flow chart begins as the system powers on. The system constantly listens for other broadcasting sources. When the system receives a broadcasted message, the codec determines if the packet is valid, and compares the message to ensure it is new and relevant data. If the data is new, the system decodes and relays the packet, and stores the data. After the data is stored, or if the packet is invalid, the system returns to listening for broadcasts.



modifying the code. We were able to remove the unwanted packet data, and now had nearly 17 bytes of data length to pack as much useful information as possible into our message.

It's important to note that we originally planned on sending images taken by the field node to the base node, but it was clear that this would not be possible with only 17 bytes. Because we were no longer concerned with sending large amounts of data, we decided on four important details to send.

WHAT, WHEN, WHERE?

First is the opcode, or operation code, which tells our program what type of signal is received: Alarm, status, time sync or settings update. Second is the node of origin—where the alarm was triggered. This is important because we are using a mesh network topology. In this arrangement, all nodes must be in constant communication with all other nodes within range, and the program can identify which of the nodes on the property is detecting pests. Third is the date of the event. This information is paramount to relay, because it lets the user know what time of day the pests are coming to feed, and if any patterns emerge. Finally, we send the machine learning outcomes. The outcomes include which category rated highest, and what percentage of accuracy the AI found for category certainty.

After in-house testing, we had reliable transmission and receiving (**Figure 2**) on our two field nodes and base node. We then tested the connection at the required 40', and could both transmit and receive without issue. However, to encode and decode this packet, an appropriate codec was required. We designed the codec to take all necessary information from the transmitting field node and encode it into our 17-byte packet. While on the receiving end, the codec would neatly decode the packet, interpret the string of characters and display the four aforementioned pieces of data.

DESIGN DECISIONS

PCB Design: With the network up and running, it was time to design our PCB. Because we are using a Pi as the center for our design, we already had most of our functionality covered. All the PIRs could be connected via the Pi's GPIOs, and the two cameras would be connected via USB and the 15-pin FFC header. The purpose of the PCB (**Figure 3**) was to allow us to accept a variety of input voltages and supply types. To accomplish this task, a power rectification and distribution system was needed.

We used an LM2596 DC buck regulator module by Lanpu to do this job. Based on Texas Instruments' (TI's) LM2596 chip, the

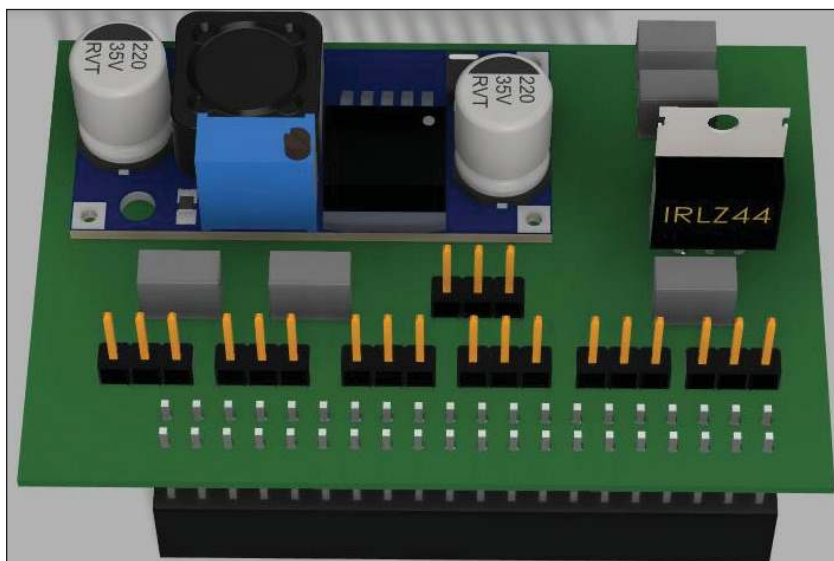


FIGURE 3

A 3D rendering of our custom-designed Pi-hat PCB (green). On the board is one Lanpu LM2596 Buck Power Supply Module (blue), one P-channel MOSFET, 4 Bulk Capacitors, 7 Jumpers (3-pin) and access for the 40-pin Pi header.

regulator module met all our requirements for power distribution: a 3A current output to power the board in its awake state, the PIRs and the camera modules with current to spare. It also provides a 3V to 40V input voltage range, allowing us to accept a wide variety of battery compositions (lead-acid or lithium-ion) and setups (large series battery banks). Finally, the module can be powered by a common AC adapter.

Enclosure Design: Once we had chosen and designed all our components, we had a good understanding of the dimensions we needed to work with for the field and base node enclosures. The requirements for the field node enclosure design (**Figure 4**) included a small overhang from the lid to cover the

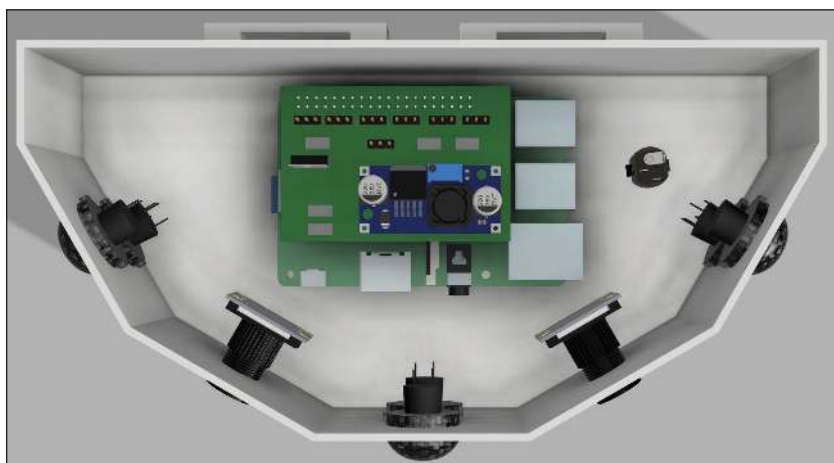


FIGURE 4

A top-down 3D rendering of our node enclosure. Custom holes for both the cameras and PIRs were placed with the intention for maximum coverage and consistency. To the right of our stand-off mounted Pi is a DC barrel jack that connects the battery back. At the back is the mounting bracket for the enclosure.



FIGURE 5

A 3D rendering of our battery pack. This pack is attached to the bottom of the node enclosure. We designed this feature to add versatility to our design, making it optional to expand to off-grid solutions.



FIGURE 6

Our 3D-printed node enclosure fully assembled with the battery pack. We opted for a classic camouflage paint job for the field node, since it seemed appropriate for the market. Using stencils, it took several days to paint it.

camera lenses from direct rainfall and evenly spaced mounting holes for the cameras and PIRs to achieve 180 degrees of horizontal view coverage. Also needed was an optional battery compartment (**Figure 5**) for off-grid applications, and the ability to be mounted a variety of ways for a simple installation process.

To achieve these requirements, we decided to 3D print our enclosures out of polylactic acid (PLA) using an Ultimaker 3. This allowed us to paint, shape and design our enclosures with relative ease, and helped to create a professional representation that portrayed our desired final product (**Figure 6**). Another benefit of 3D printing the enclosures was our ability to reprint an entirely new enclosure in less than 24 hours. If we didn't like the look, or if we found there wasn't enough space for the Raspberry Pi inside, we could make the necessary adjustments and be ready by the next day. The mounting bracket on the back of the enclosure allows it to be placed on a stake in the ground, or tied with a strap to a fence or house.

We designed the base node (**Figure 7**) to be placed on a countertop, take up minimal space and display useful information to the user. With those objectives in mind—and because we used a 7" LCD Raspberry Pi touchscreen—we designed the enclosure to frame the screen and slant it backward slightly for good readability and a stylish, welcoming appearance.

GUI Design: Once we had finished the enclosures, we began creating the GUI for the base node. The program we used to design the GUI was Node-RED, a web browser-based development tool for wiring hardware, APIs and other IoT devices. We chose Node-RED (**Figure 8**) because it is designed for Raspberry Pi applications and offers a simple and familiar flowchart-style IDE. We achieved an easy-to-use GUI that offers the user relevant data such as field node battery life (**Figure 9**) and alarm status (**Figure 10**) when interacted with, and otherwise displays the current time and date.

DISCUSSION

We originally set out to design an automated pest deterrent solution for use in gardens and on farms. In the end, we created a powerful and sophisticated deterrent system with the potential for growth and adaptation to the circumstances of any property owner. We believe this achievement gives our design great value and merit, particularly if we decide to produce it commercially.

In future iterations of the design, however, we would make hardware changes and quality-of-life improvements. First,

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

RESOURCES

Panasonic Electronic Components | na.industrial.panasonic.com


PyBeacon | www.pyipi.org

Node-RED | www.nodered.org

TensorFlow | www.tensorflow.org

Texas Instruments | www.ti.com

we would use a metal enclosure rated IP64 (dust-tight and protected from water splashes from all directions), to allow the field nodes to operate in a farmer’s field. Second, we would replace the Raspberry Pi 3B and all its excess features with a processor with a lower power consumption. This processor would be embedded into a custom PCB design encompassing only the necessary functionality. And finally, we would switch from BLE to Bluetooth 5.0, to achieve a larger packet size. This would allow images to be sent from the field node to the base node.

We owe much of our success to proper project management. We took the time early in the development process to address the faults and failures of the market for pest deterrence. As a result, we were able to overcome them by using new technologies and our current knowledge base, developed through Camosun College’s Electronics & Computer Engineering Technology – Renewable Energy program. 

Author’s Note: Cole, Cameron, and Simon are graduates of Camosun College’s Electronics & Computer Engineering Technology – Renewable Energy program, and they plan on continuing part-time development of their project to manufacture an affordable and effective pest deterrent product.

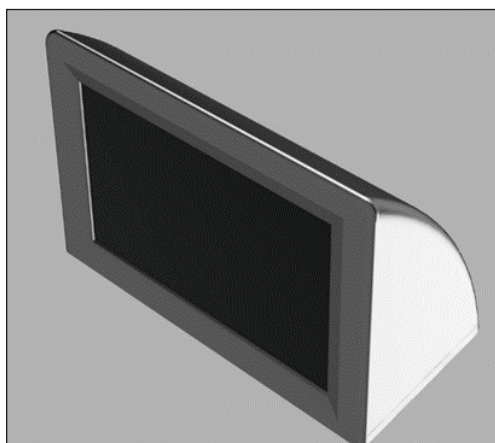


FIGURE 7

The 3D design for our base node. We had two thoughts in mind when designing this node—sleek and simple. The base node will sit tilted back, and will only require power to operate.

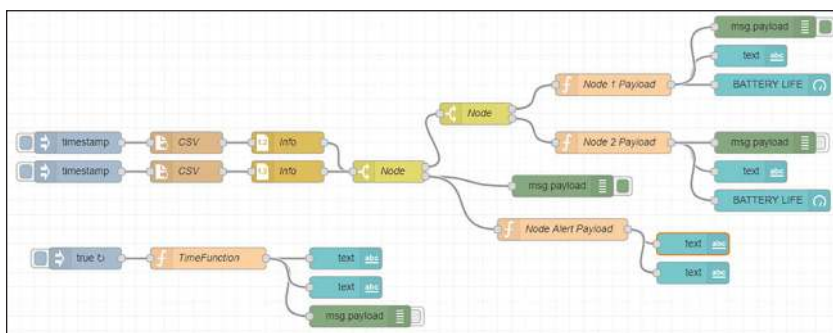


FIGURE 8

Node-RED flow chart, showing the operating functions of our base node. The operations are battery life for nodes 1 and 2, node alarm status, or date and time. The flow lines starting with a “timestamp” begin with received data from the field nodes.

ABOUT THE AUTHORS

Cole Gamborski is a Camosun College graduate with many years of experience in the field of property security. He has an aptitude for programming and is continuing at Camosun in its Engineering Bridge program in order to earn an Electrical Engineering degree from UVIC.

Cameron Phillips is a Camosun College graduate with many years of mechanical experience. He is well versed in machine learning and 3D design, and he will be taking on a co-op with FTS Forest Tech before continuing his education.

Simon Fowler is a Camosun College graduate with experiences designing solar PV installations. He hopes to find work in the Solar PV field before continuing his education in the Camosun Engineering Bridge program.

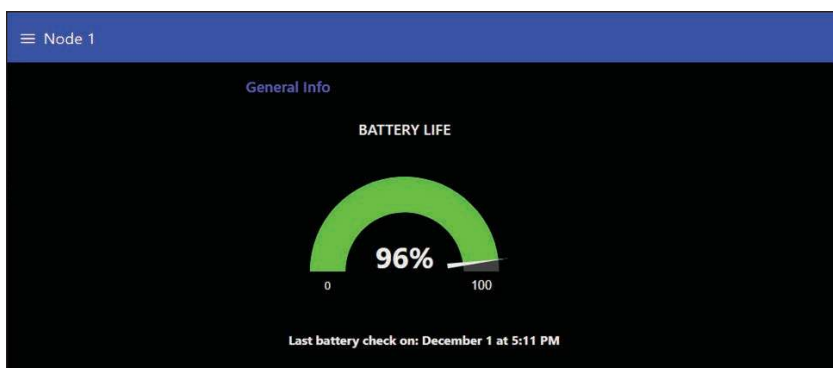


FIGURE 9

Screen capture of our base node’s display. The node is displaying the battery life of field node 1 in percentage, and the most recent check time. This information is under the “Node 1” tab.



FIGURE 10

Screen capture of our base node’s display of the most recent alarm status. The information we considered most important to the user includes the sending node, the detected animal, and the time of alarm. This information is under the “Warning” tab.

Creative Mechanical Ideas for Embedded Systems

Professional Style Projects

FEATURES

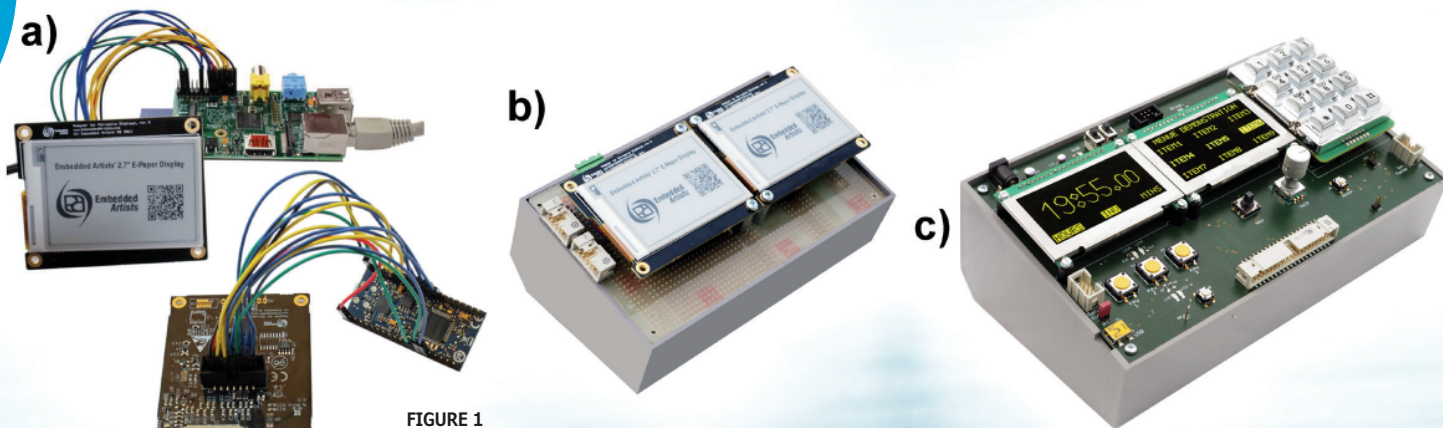


FIGURE 1

A display module is to be attached to an MCU module. (a) shows a hi-tech solution that the vendor recommends. Now, try to imagine implementing a dual-screen display. (b) is an improvised wire-wrapped module, quick and dirty, but rock-solid. A purpose-designed PCB (c) is more presentable. Free design software and affordable manufacturing services allow even the hobbyist to do it this way.

If anyone doubts that electronics tinkering as a hobby is on the decline, Wolfgang reminds us that there's never been more rich resources available for such pursuits. With that in mind, in this article Wolfgang presents several mechanical ideas for embedded systems using small- to medium-sized PCBs and common 19" hardware such as 3U front panels and subracks.

By
Wolfgang Matthes

Every now and then you hear someone decry the decay of electronics as a hobby ([1] to [5]). But the reality is that there is a plethora of microcontroller (MCU) platforms, shields, HATs and the like available these days. Everywhere you look, there are Internet forums and vendors serving the community of the so-called makers. There's also a multitude of fairs catering to makers—most postponed now during the COVID-19 pandemic of course, but sure to return to full strength in the future.

When you compare the today's projects shown at fairs or on the Internet to what ambitious hobbyists of the past have been accomplished, you see a conspicuous difference. More often than not, the maker's work manifests only in a contraption of modules, breadboards and jumper wires, but

not in a piece of impressive equipment like in **Figure 1c**, for example.

In some of my previous *Circuit Cellar* articles ([12] to [15]), I've already dealt with projects aimed at sturdy and perhaps even presentable devices. Here, we will concentrate on mechanical design, depicting some ideas that are not that common in today's realm of tinkering, experimenting and prototyping. References [6] to [11] are a small selection of relevant articles and web content.

We want to stay within affordable limits. With that in mind, the proposals here are centered around small- to medium-sized PCBs and ubiquitous 19" hardware—especially 3U front panels and subracks. These PCBs and front panels are not overly complicated or large. They could be developed using free design automation software and manufactured by any appropriate service provider. With

modest equipment and effort, they could be turned into presentable showpieces—even at the proverbial kitchen table.

The basic ideas in this article occurred over my years of project work. Instead of relying on the ubiquitous small modules, I preferred medium-sized PCBs with interface connectors located somewhat thoughtfully, so as to ease assembling more complex devices. To build them sturdy and presentable, I make good use of 19" hardware, above all of the subracks, chassis and front panels.

Becoming familiar with the 19" system, I revived the venerable technology of small PCBs to be plugged into a backplane. Occasionally, it turned out that PCBs make sufficiently rigid 3U front panels, at least for educational modules with small connectors. The concluding idea to be discussed here is to employ ZIF sockets as general-purpose component adapters, being a somewhat expensive, but much more reliable alternative to the white breadboards.

MORE THAN ONE PCB

A PCB typically carries one module—a functional unit. Typical hobbyist or educational modules should be neither too expensive nor too complicated. But most projects won't get by with only one module. As a result, problems arise on how to arrange and fasten the modules and how to interconnect them.

It is important to select the principal mechanical design early because each one has its particular requirements concerning

the selection, placement and pinout of the connectors. **Figure 2** introduces some of the well-proven approaches to how machines could be built from more than one module or PCB. Our preferred solutions will be illustrated in some photos. Additional photos can be found on *Circuit Cellar's* article materials web page. Still more photos and drawings are available on my own webpages (see "About the Author" box for links). On those webpages, the different approaches shown in Figure 2 will be illustrated and discussed more closely.

Figure 3 shows a generic block diagram of a typical project. Imagine, for example, a programmable logic controller (PLC) built for educational purposes and experimenting. In this respect, it makes sense not to strive for compactness and miniaturization, but to make all functional units easily accessible. The machine is centered around an MCU module as its central processing unit, connected to human-machine interface (HMI) modules, input/output modules and upstream and downstream interfaces. Furthermore, Figure 3 hints to additional modules to emulate or substitute the real-world process environment, here dubbed the diagnostic front-end.

Trying to implement such a project by selecting suitable modules in an ecosystem like Arduino or Raspberry Pi is an instructive exercise. Appropriate shields, HATs and the like are easy to find. But what's the best way to arrange, fasten and interconnect them to implement the complete machine? Usually,

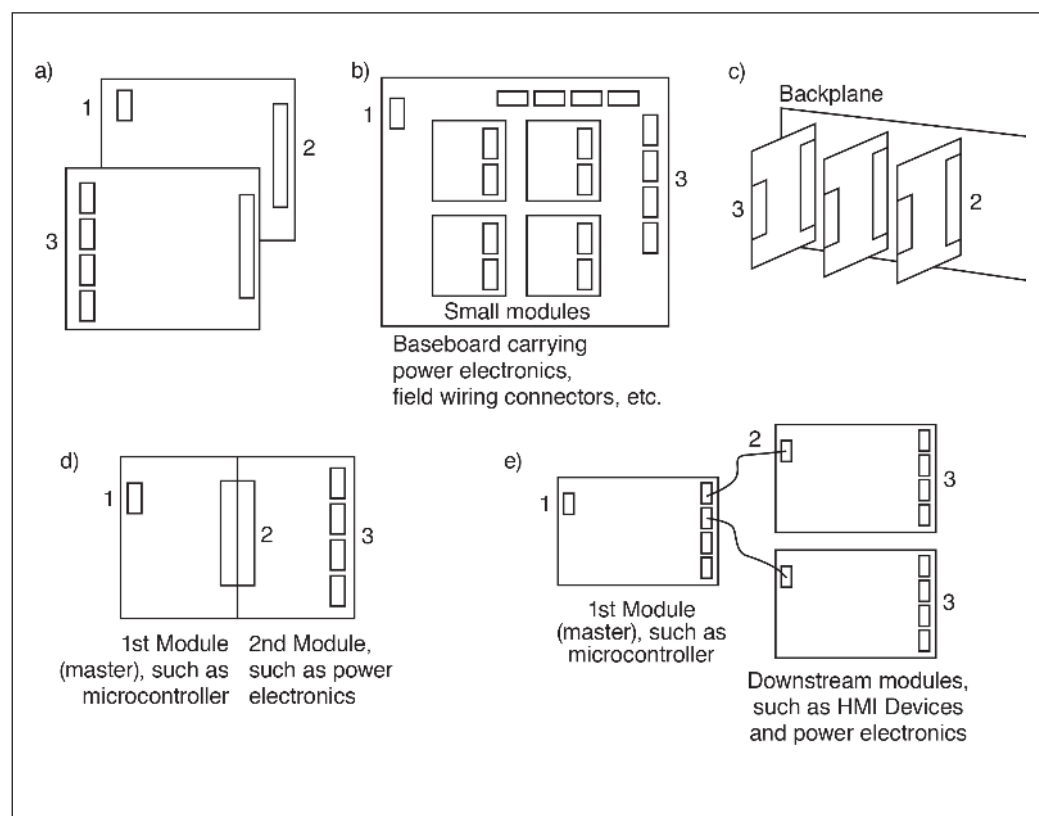


FIGURE 2

Shown here are some well-proven mechanical design principles to build machines from more than one module or PCB. (a) stack one upon another, (b) plug them into a baseboard, (c) plug them into a backplane, (d) plug them against each other, (e) set them up individually and connect them by cables. 1 - upstream interface; 2 - module-to-module connections; 3 - I/O connectors.

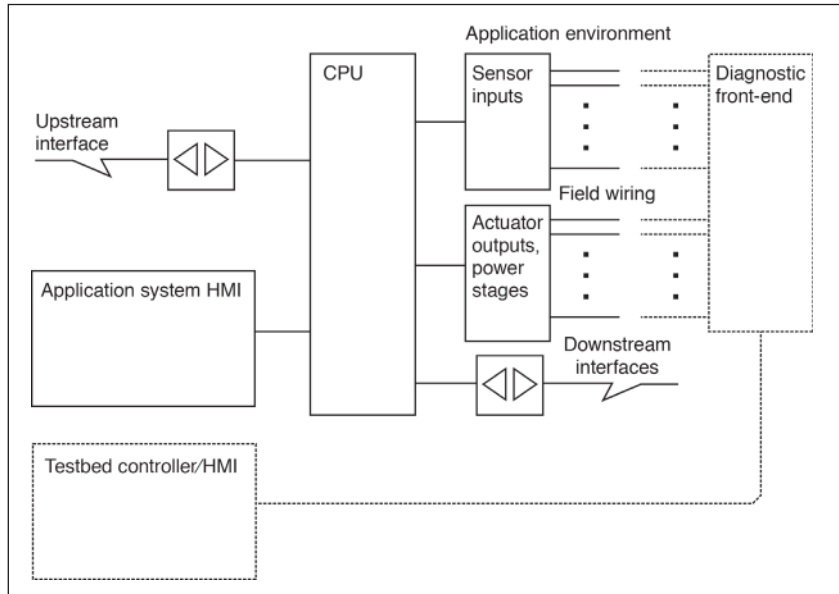


FIGURE 3

A typical project depicted as a generic block diagram.

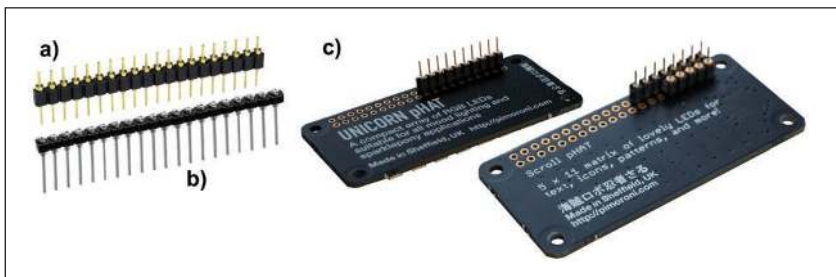


FIGURE 4

The header (a) is the type of connector to be soldered into the modules. The baseboard is to be populated with strip-line sockets (b). Modules with headers (c) are inserted into the sockets (b).

only one peripheral module can be stacked on top of the MCU module—remember, it is not an industrial-grade form factor, like PC/104. Three, four or more modules could be connected as shown in Figure 1a. That approach, however, is not very inviting.

An obvious solution is to arrange all the modules or PCBs next to each other, for example, by plugging them onto a baseboard. The modules, shields, HATs and the like must be equipped with appropriate pin headers, as shown in **Figure 4**. A quick-and-dirty approach could be wire-wrapping with a sufficiently large prototyping board or perfboard as the baseboard [14]. The ultimate solution would be, of course, the purpose-designed PCB. By the way, a baseboard with thoughtfully placed wire-wrapping sockets can be altered and even re-used multiple times, provided you employ an appropriate unwrapping tool and work somewhat cautiously. In principle, however, the shields, HATs and so on are not marketed to be used in projects outside their particular ecosystem. They are deliberately not meant as OEM components.

MORE COMPLEXITY

Alternatively, you can base your projects on modules of medium size and complexity [12], or even on industrial-grade hardware. **Figure 5** shows how the block diagram of Figure 3 can be implemented with such modules.

The modules are expressly designed for building systems of two or more modules.

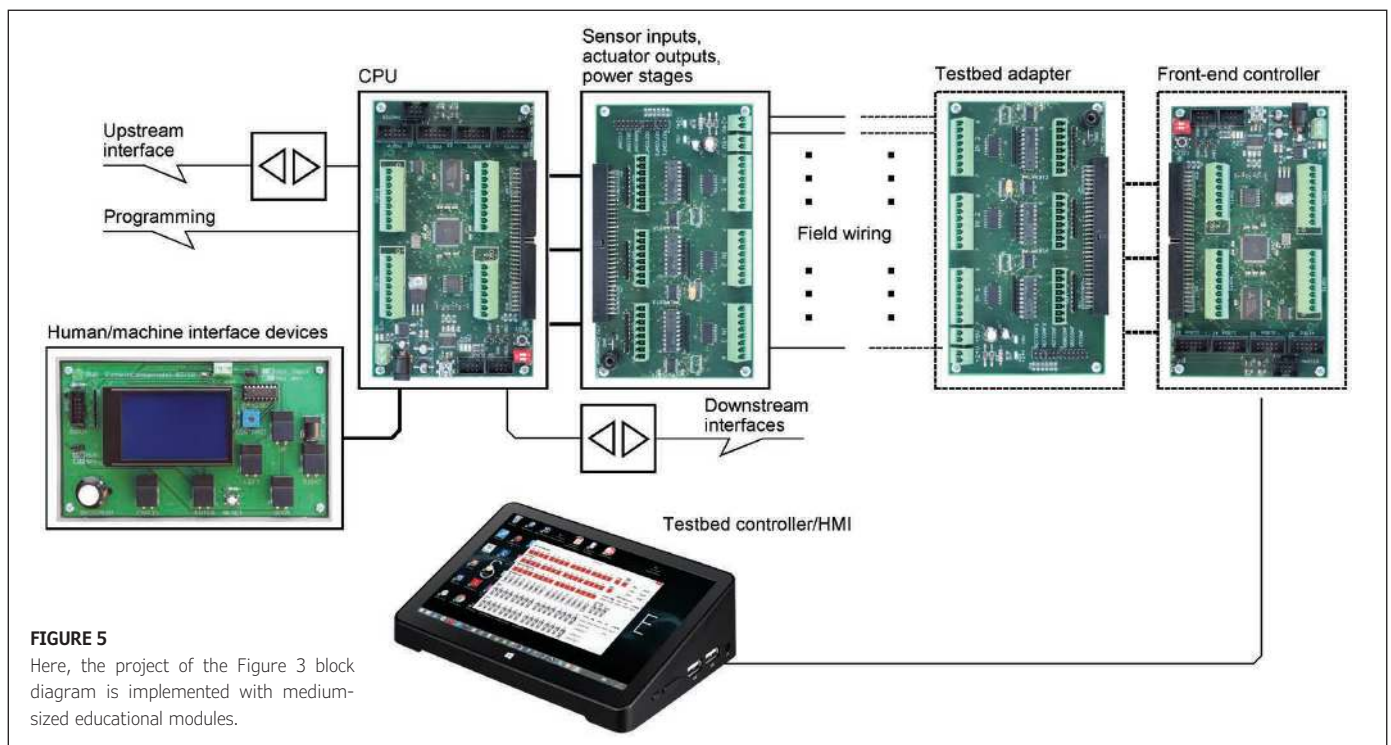


FIGURE 5

Here, the project of the Figure 3 block diagram is implemented with medium-sized educational modules.

That intention drove the selection of the form factor as well as the type and arrangement of the I/O connectors. Connections should be inexpensive and reliable. Besides, they should allow for maximum flexibility. Therefore, we prefer shrouded pin headers, ribbon cables and terminal strips. The modules can be mounted on top of separate enclosures (Figure 6), inserted into housings to be clipped onto DIN rails (Figure 7) or fastened on a chassis (Figure 8).

In most of the applications, the modules will be placed next to each other in order to make all components and interfaces easily accessible. Figure 9 shows a preferred arrangement. It serves as a good template to place the components on the boards, especially the connectors.

If the MCU board is in front of the user, the upstream devices are on the left and the downstream devices are on the right. An upstream device can be, for example, a personal computer (PC) or an MCU module acting as a master or hub. Downstream devices are MCU modules acting as I/O or slave processors, display modules, modules with power stages and so on.

The MCUs aboard the modules are connected via serial interfaces. The upstream device is the master. A downstream device is a slave. This results in a star topology with a hub in the center. It's the same principle as USB, just a lot more straightforward.

Figure 10 depicts how the connectors are to be placed. The connectors to the left and to the right are preferably shrouded pin headers. Serial and programmer interfaces have 6 pins, I/O ports 10 pins. Some modules support serial communication via an RS-232 interface or the USB. Accordingly, they carry D-sub or USB connectors. The connectors at the rear edge are terminal blocks or shrouded pin headers.

At the front edge, some modules have a so-called multi-purpose connector to attach additional PCBs. Figure 11 shows an

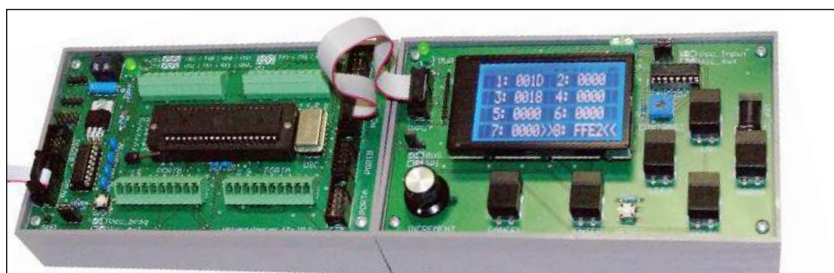


FIGURE 6
Here, two of our modules are shown, as described in [12].

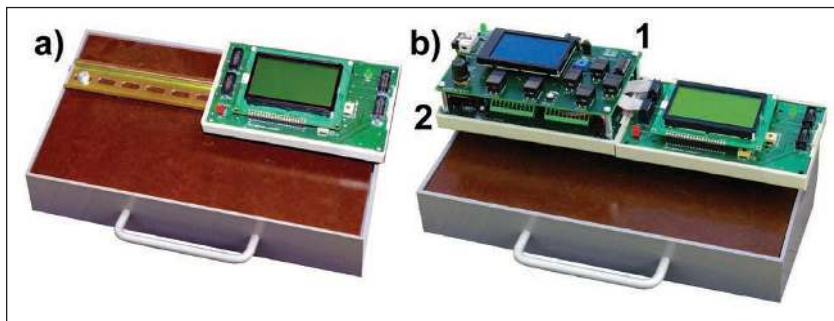


FIGURE 7
Modules clipped on a DIN rail. (a) illustrates the mounting principle, (b) shows an educational PLC consisting of three modules. The human interface module 1 is stacked atop the MCU module 2. All modules are connected by ribbon cables.

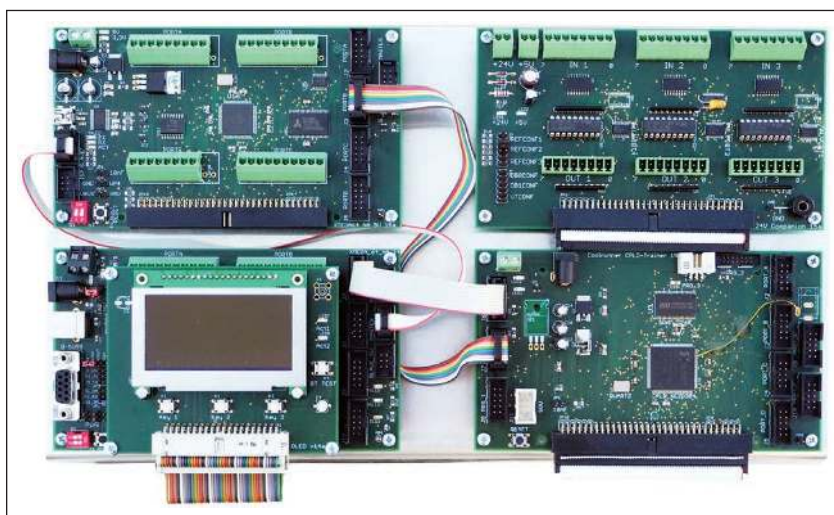


FIGURE 8
Four modules mounted on a chassis

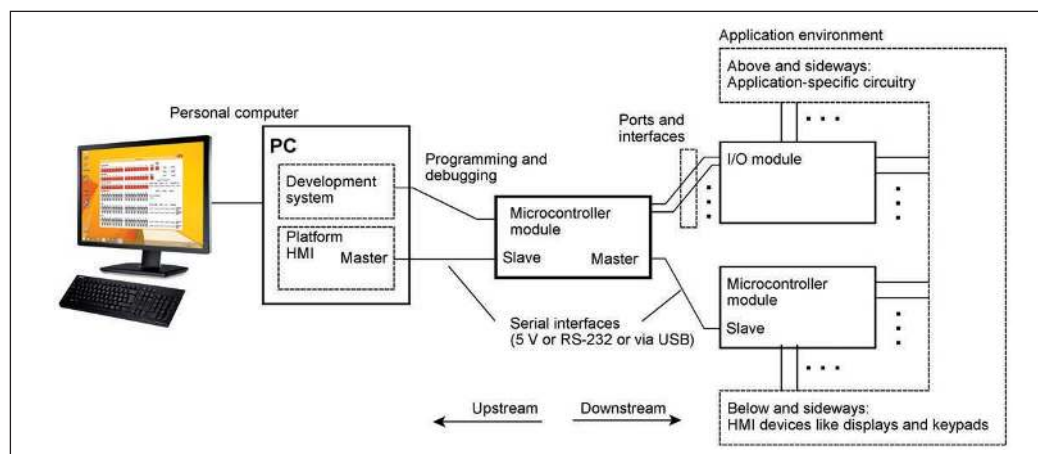
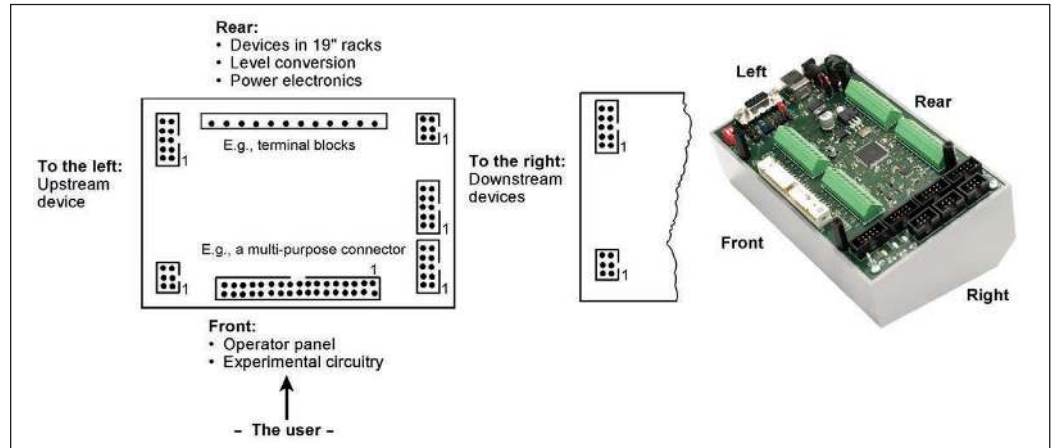


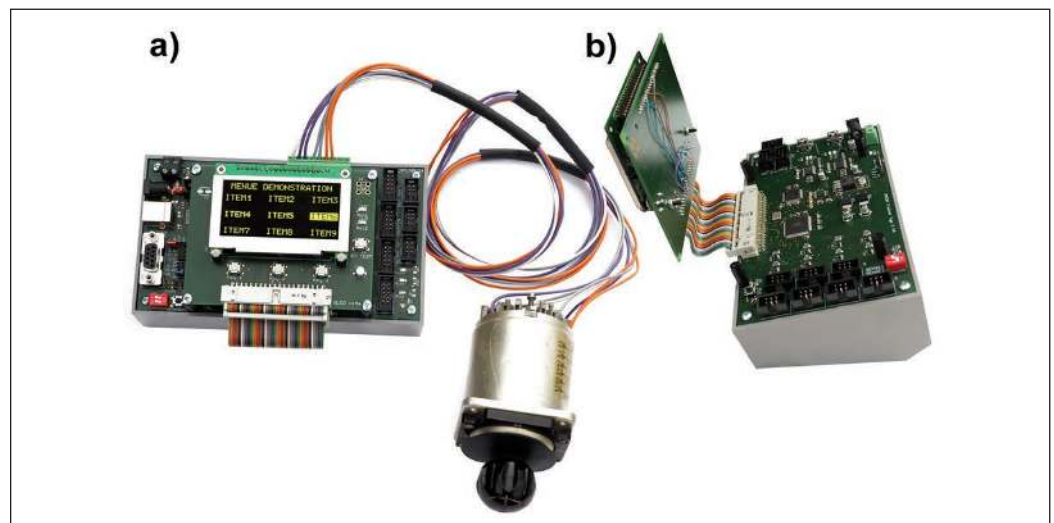
FIGURE 9
An MCU module in a programming and application environment

FIGURE 10

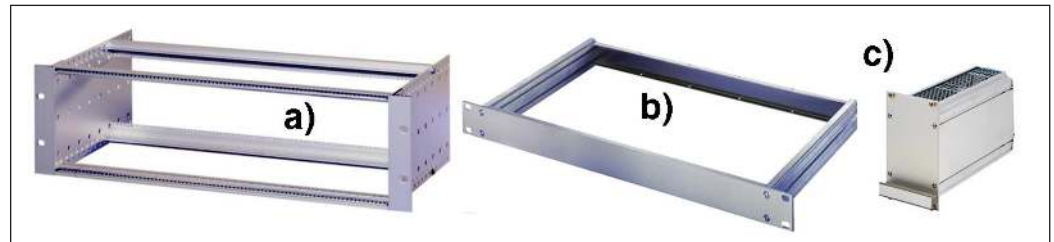
Connectors are placed according to the principle depicted here.

**FIGURE 11**

(a) One of the more advanced modules connected to a historical joystick. The operating and display panel on top of the module is attached via the multi-purpose connector. (b) shows the 40-pole pin header.

**FIGURE 12**

Some examples of ubiquitous 19" hardware components

**FIGURE 13**

Two historical examples of how devices are built from modules plugged into a backplane (Digital Equipment Corp.).



ABOUT THE AUTHOR

Wolfgang Matthes has developed peripheral subsystems for mainframe computers and conducted research related to special-purpose and universal computer architectures for more than 20 years. He has also taught MCU Design, Computer Architecture and Electronics (both digital and analog) at the University of Applied Sciences in Dortmund, Germany, since 1992. Wolfgang's research interests include advanced computer architecture and embedded systems design. He has filed more than 50 patent applications and written seven books. (www.realcomputerprojects.dev and www.controllersandpcs.de/projects).

example. The ribbon-cable connection can be opened like a book, thus allowing access to all components on both boards. The PCB layout of the multi-purpose connector can be thought of as some kind of standard, providing a maximum of 64 holes (in two rows of 32) to be populated with different pin headers.

SOME MORE HARDWARE OPTIONS

Using 19" hardware: 19" hardware has its origins in the realms of telecom and measuring equipment. **Figure 12** shows a few examples of ubiquitous 19" hardware components. Chiefly, we will rely on 3U subracks (Figure 12a) and the chassis (Figure 12b). Humble educational and hobbyist projects allow for a simplified mechanical design. Frame-type plug-in units (Figure 12c) will not be required. The front panel alone yields a sufficient mechanical platform.

Small pluggable modules: Decades ago, electronic devices were built from small modules plugged into a backplane (**Figure 13**). Why not revive this well-proven principle? Inspired by the small boards of IBM's SMS and SLT technologies, of DEC's modules, of the Control Data (CDC) computers and the like, we have chosen half a Euro-board (100mm x 80mm) with a DIN 41612 connector as the basic form factor. The boards shown in **Figure 14** carry CPLDs, MCUs (AVR, Arm and 8051), SRAMs and dual-port RAMs.

The modules are plugged into a backplane that is to be wire-wrapped. The mechanical platform could be provided by a 3U subrack including card guides and ejector handles (**Figure 15a**). A straightforward solution could be based on a perfboard with soldered-in connectors (**Figure 15b**). If the device is not mechanically stressed, card guides and handles are not necessary, because the cards will be held firmly in place by the connector's friction alone.

Front panels as chassis: Industrial-grade 19" modules are typically designed as a front panel with an attached PCB or as a frame-type plug-in unit. The mechanical design of

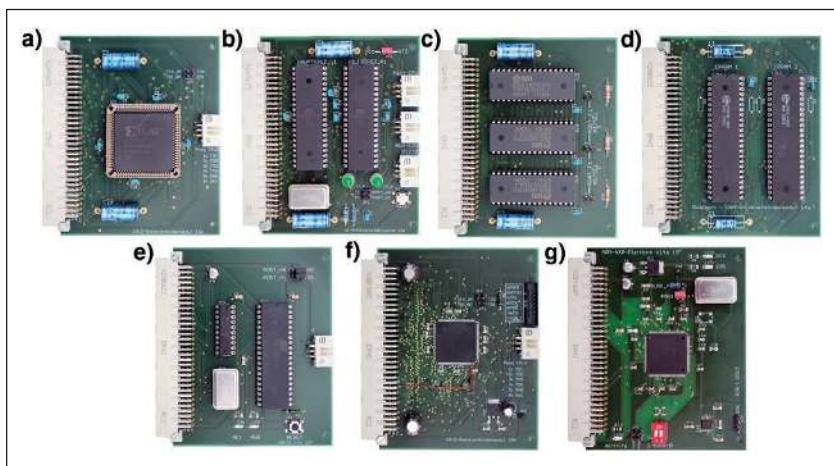


FIGURE 14 Examples of today's modules thought of for experimental, educational and ambitious hobbyist projects. (a) CPLD Xilinx 95108, (b) two ATmegs 1284, (c) three SRAMs 128kx8, (d) two dual-port RAMs, (e) MCU 80C51RD2, (f) CPLD Xilinx CoolRunner XC2C384, (g) MCU ARM NXP LPC2220.

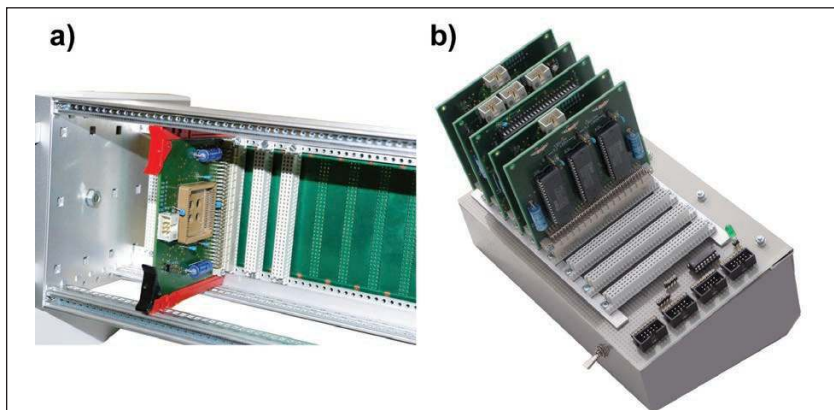


FIGURE 15 Modules in a 3U subrack and on a jerry-built experimental platform

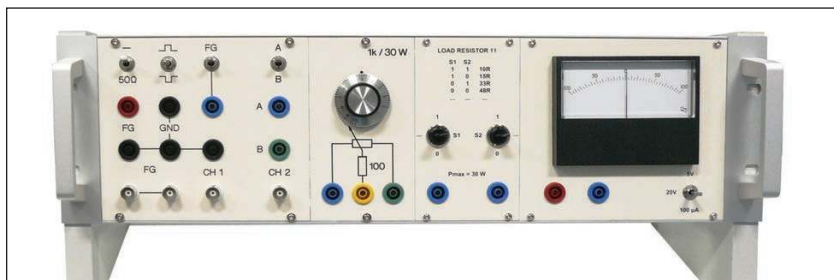


FIGURE 16 The modules in this rack are built this way. The front panels have been manufactured by a service provider, the mechanical design and wiring are homemade.

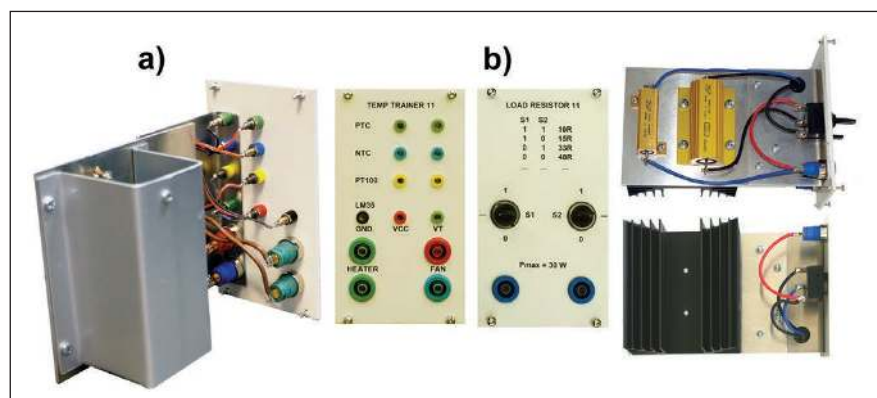


FIGURE 17 Two examples of modules. (a) shows a temperature trainer containing different temperature sensors that can be heated up or cooled down. (b) is a switchable load resistor, providing four different resistance values.

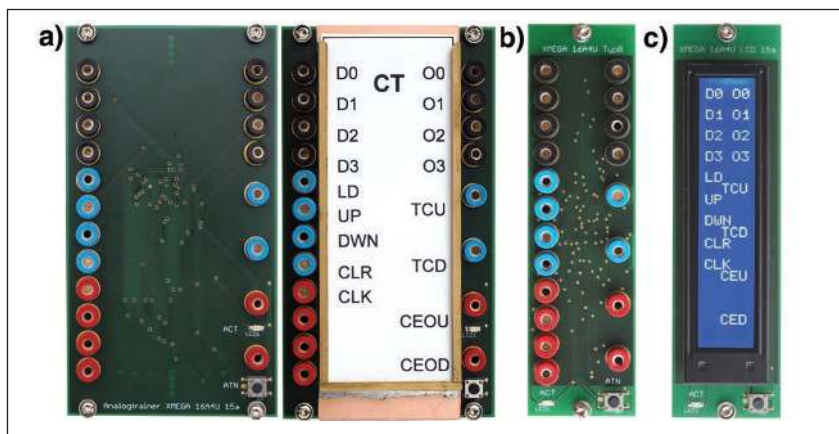


FIGURE 18

These MCU modules can be mounted in 3U subracks like front panels. The modules (a) and (b) have an AVR Xmega MCU. They are just different enough in their width, that one is 14 HP and one is 7 HP. Module (a) is that wide to allow for mounting a frame to insert a strip of paper. All the jacks are connected to freely programmable I/Os. Module (c) is a graphic LCD display of 32 x 180 pixels, controlled by an ATmega MCU.

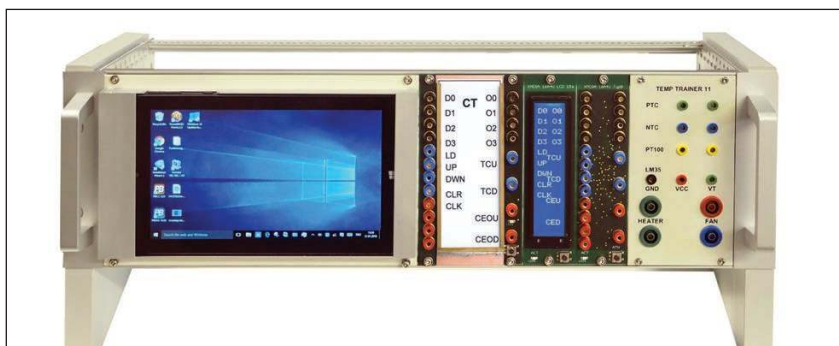


FIGURE 19

Some of our modules, accompanied by a 7" Windows tablet

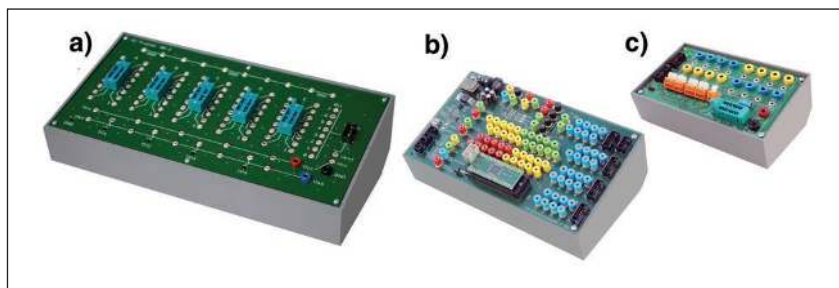


FIGURE 20

ZIF sockets as component adapters. (a) IC trainer, (b) DIL-40 baseboard and (c) general-purpose adapter

small, humble modules, however, may be centered around the front panel alone. All components are attached to the front panel. **Figure 16** shows a 3U rack with modules built this way. The modules presented in **Figure 17** are examples of mechanical designs that are somewhat more demanding.

PCBs as front panels: Epoxy PBCs (FR4, 1.6mm thick) make sufficiently rigid 3U front panels—at least for educational modules with small connectors. The modules shown in **Figure 18** and **Figure 19** have 2mm jacks, so insertion and withdrawal forces are not that high. All modules carry an AVR MCU. They are intended to be programmed as digital or analog simulators. In this respect, the modules are quite similar to the modules shown in the left picture of **Figure 13**. In the example of **Figure 18**, the modules are programmed to simulate an up/down counter. The jacks may be labeled by a strip of paper or by an LCD display (the luxury variant).

ZIF sockets as component adapters: ZIF (zero insertion force) sockets can accommodate arbitrary components, provided they are to be attached via pins or wire. This fact has led to devices intended as an alternative to the ubiquitous white breadboards. The devices are essentially PCBs with ZIF sockets and jacks.

The IC trainer shown in **Figure 20a** provides five ZIF sockets with 16 pins each. The DIL-40 baseboard (**Figure 20b**) has a ZIF dual-in-line (DIL) socket with 40 pins, chiefly to accommodate an MCU or a CPLD. It provides somewhat like an infrastructure, comprising a crystal oscillator, RS-232 and USB attachments, and pin headers to connect it to other modules. SMD components are inserted via interposer boards. The general-purpose adapter shown in **Figure 20c** carries a 16-pin ZIF socket connected to different jacks and terminals. Up to two 8-bit ports from other modules or starter kits can be attached and forwarded to 2mm or 4mm banana plugs, stripped wire and arbitrary components fitting into the ZIF socket.

For detailed article references and additional resources go to: www.circuitcellar.com/article-materials
References [1] through [15] as marked in the article can be found there.

RESOURCES

BusBoard Prototype Systems | www.busboard.com

Fischer Elektronik | www.fischerelektronik.de

Hammond Manufacturing | www.hammondmfg.com

NVent/Schroff | www.nvent.com

SchmartBoard | www.schmartboard.com

Rittal | www.rittal.us

Vector Electronics & Technology | www.vectorelect.com

Vero Technologies | www.verotl.com

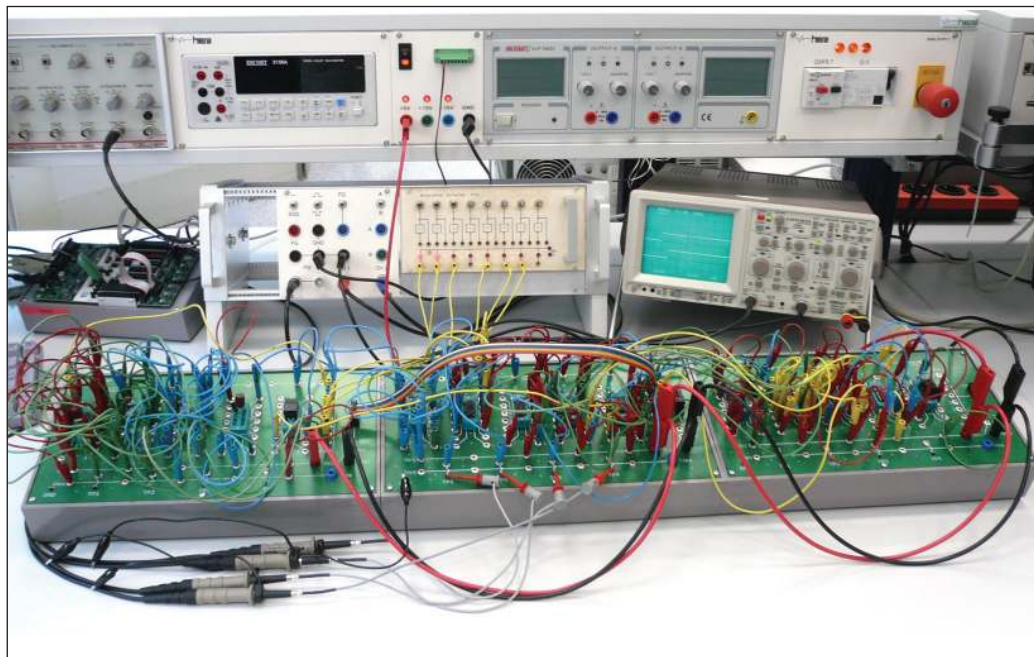


FIGURE 21
A somewhat more sophisticated analog circuitry on three IC trainers

Figure 21 and **Figure 22** illustrate how these devices are used in the lab.

SUMMARY AND SUGGESTIONS

Even today it is still possible to pursue projects that are sturdy, somewhat more ambitious and sometimes even presentable. The preeminent approach is to seek components showing an appropriate level of handiness. They should not cost too much. You should be able to handle them in your workshop or even at your kitchen table. It all depends on your inventiveness. Some technological challenges can be met by taking advantage of the vast range of accessories the market offers. Typical examples are interposer boards to accommodate SMT components.

Furthermore, you should not hesitate to make good use of components or modules outside their native ecosystem. In that respect, even complete tablet PCs could

be employed as components, for example, to serve as HMI devices, thus substituting homemade front panels carrying LEDs, keys and switches [15]. A basic tenet is to reduce complexity by building somewhat larger.

For example, it may be tempting to build your own computer from scratch. It goes without saying that FPGAs are around that could easily accommodate the complete project. But by pursuing such an endeavor, you will depend on a development environment that will cost months alone to become familiar with. Therefore, maybe a more adequate hobbyist or educational computer should be built from small modules, each containing a functional unit implemented in a tiny FPGA or even CPLD. This way, you will obtain not just a virtual, but rather a real testbed and playground—a machine you can bring up and troubleshoot hands-on with the oscilloscope or the logic analyzer. **E**

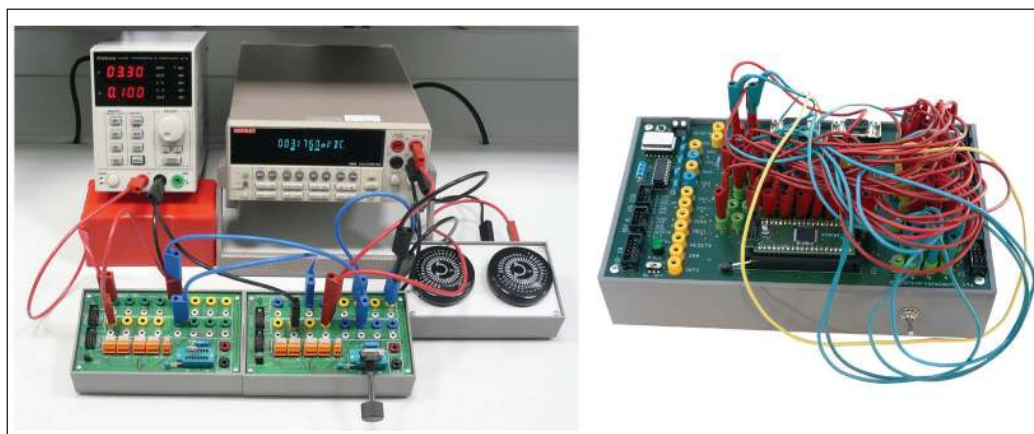


FIGURE 22
Two general-purpose adapters and a DIL-40 baseboard used intensively

Modernizing the Accuracy of an Antique Clock

Using a PSoC4 MCU

Antique electric clocks are both beautiful and elegant, but they really can't provide accurate timekeeping. In this project article, Aubrey makes use of modern Cypress PSoC4 MCU technology to convert an antique clock into a device that can continuously keep accurate time.

By
Aubrey Kagan



FIGURE 1

Eureka clock. The rotating pendulum is blurred, but you can see the balancing weights on its circumference.

My friend Nigel is a clock collector. Amongst his collection are two types of antique electric clocks (from the turn of the 20th century), which appear to have different mechanisms but work on a similar principle. One is an American-designed Eureka clock (**Figure 1**) and the other is a French designed Bulle clock (**Figure 2**).

The Eureka “pendulum” consists of a balanced disk that rotates in one direction—tensioning a coil spring—and then returns in a mesmerizing display. A video of this [1] is included on the *Circuit Cellar* article materials webpage. An electromagnetic coil is mounted in the center of the disk, and a voltage is connected to the coil via a momentary mechanical contact. Two fixed magnets are on the circumference of the frame around the pendulum. When the coil and the magnets align (as determined by an electrical contact), a current is passed through the electromagnet, creating repulsion against the magnets, and energy is imparted to the pendulum.

The Bulle mechanism may boggle your mind [2]. Although the electromagnet is visible at the end of the pendulum, the arc at the bottom is a 3-pole bar magnet, north at the ends and south in the middle. Apparently, this creates a strong magnetic field in the center,

but is also an unstable configuration. So, the magnet sometimes needs reconditioning—something not for the faint of heart [3].

As with the Eureka clock, a voltage is presented to the electromagnetic coil with a switching contact when the electromagnet is at the midpoint, creating repulsion, and the pendulum gains energy. The pendulum on each unit drives an escapement mechanism that is geared to drive the clock hands.

There are several explanations of both clocks on that impeccable source: The Internet. Both devices are intended to be operated from a 1.5V battery, and, surprisingly, the coils of each one measure about 1,200Ω, despite some claims to the contrary. See the *Circuit Cellar* article materials webpage for links to descriptions of the Eureka [4] and Bulle [5].

There are several other sources out there—including an article that describes some electronics to regulate the timing of the Eureka. That article is not included in my References because it is no longer available. It is suggested in several places to parallel two batteries to extend battery life. I trust that *Circuit Cellar* readers will understand why that is a bad idea. I can find no discussion why the Bulle needs a 3-pole magnet, and the picture [3] clearly shows only half an arc for the bar magnet. Any suggestions from readers will be welcomed.

Neither clock is great at timekeeping, though the Eureka is worse. The primary cause is thought to be the decaying battery voltage over time, but there are many other possibilities. Temperature, humidity and the cleanliness of the switching contact (affecting the current through the electromagnet) are some of the suspects. Nigel had read the description of the aforementioned electronic regulator, and thought that together, we might come up with a suitable means of regulating the time. These clocks have value as antiques, so we needed to control the timing by external means only.

SOME PHYSICS

According to the sages (Galileo included), the frequency of operation of harmonic oscillators is independent of the range of motion in the first approximation—so we will have to operate beyond that approximation. Our experiments will change the amount of energy imparted to a pendulum, in the hope that it controls the time. We think the further the travel, the more likely it becomes that we will exceed the first approximations of the mathematical analyses.

The energy imparted to the electromagnet can be controlled by the applied voltage. In the video in Reference [2], the contact is just visible behind the hour hand. It's also clearly visible at around 56 seconds in the video in Reference [3]. That contact closed for several tens of milliseconds, and ordinarily the energy is supplied for the full period of the contact. If we can control the period of the voltage for less time than that of the contact, this could create a second method of controlling the energy imparted to the pendulum.

The initial thought was simply to use a voltage regulator to ensure a constant voltage. That quickly evolved into switching that voltage through a relay. We then headed toward some forms of control that would measure the period of the pendulum and adjust the energy supplied. And, though this pointed in the direction of PID (proportional, integral, differential) control, there was clearly an awful lot we did not know.

We decided to use a microcontroller (MCU) and some electronics to allow us to check different modes of operation. At this point, we were reluctant to commit to a dedicated PCB, so the best choice was a development board. We also put the battery-operated requirement on hold, pending the results of the development.

It is no secret that I am a Cypress Semiconductor PSoC aficionado, and so I opted for the CY8CKIT-042 PSoC4 Pioneer kit (Figure 3). The PSoC4 is extremely versatile in configuring its internal peripherals, and the kit is compatible with the Arduino Uno. I invested in an OSEPP PROTO-01 prototype shield for the custom electronics, and an OSEPP 16X2SHD-01 16x2 LCD Display and Keypad Shield. Nigel has always accused me of overkill, and this kind of proves his point.

To test each stage, we created a series of “modes”—some of which had associated parameters. These could be selected using the keyboard and LCD display (Figure 4). I am ashamed to admit that despite several of my *Circuit Cellar* articles on creating menu hierarchies (*Circuit Cellar* 160, November 2003 [6] and *Circuit Cellar* 342, January 2019 [7]) I approached this with brute force. For the sake of brevity, though, I will not spend any time describing its operation.

The schematic in Figure 5 shows the internal configuration of the PSoC4 MCU. This should be read with the schematic in Figure 6, which shows the external electronics. Different

portions of both pertain to the different modes, and so I will need to refer to different subsets as we go through the modes.

TEST MODES

Test Mode M0: The first mode (M0) is only a vague improvement on the original operation of the clocks. It merely provides a regulated power supply through the Texas



FIGURE 2

Bulle clock. The pendulum here is stationary, and you can see the arc of the 3-pole magnet.

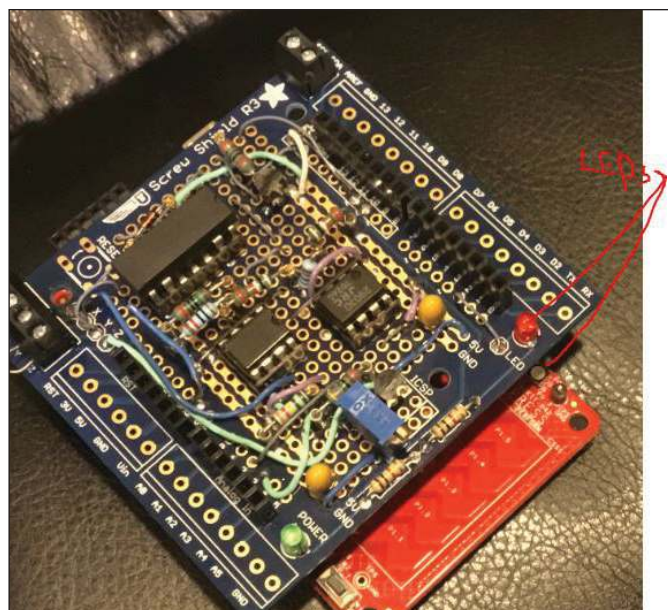


FIGURE 3

Pioneer Kit (red board) with prototype shield mounted on top.



FIGURE 4
The full assembly of three boards, with the display on top showing one of the modes.

Instruments (TI) step-down regulator LM317 (Figure 6, U1). The voltage can be adjusted using the R2 trimpot. The electromagnet coil is connected between the terminals Z and SCL. Do not confuse this with an I²C signal—it is merely the silk-screen name given to a screw terminal.

M1: If we are to control the clock by means of voltage, then we need a controlled voltage output. I called this mode 1 (M1), but with two sub-modes. The PSoC4 only has a current output DAC. I configured this as an 8-bit device, and passed it out through an analog multiplexer (DAC and AMuxHw, Figure 5). The multiplexer will be used later to generate a voltage pulse, but let's ignore that for the moment. Initially it is configured to pass the current.

On the protoboard (Figure 6) the current is converted to a voltage using R4, and is buffered by a LM358 configured as a voltage follower. The LM358 op amp from TI is capable

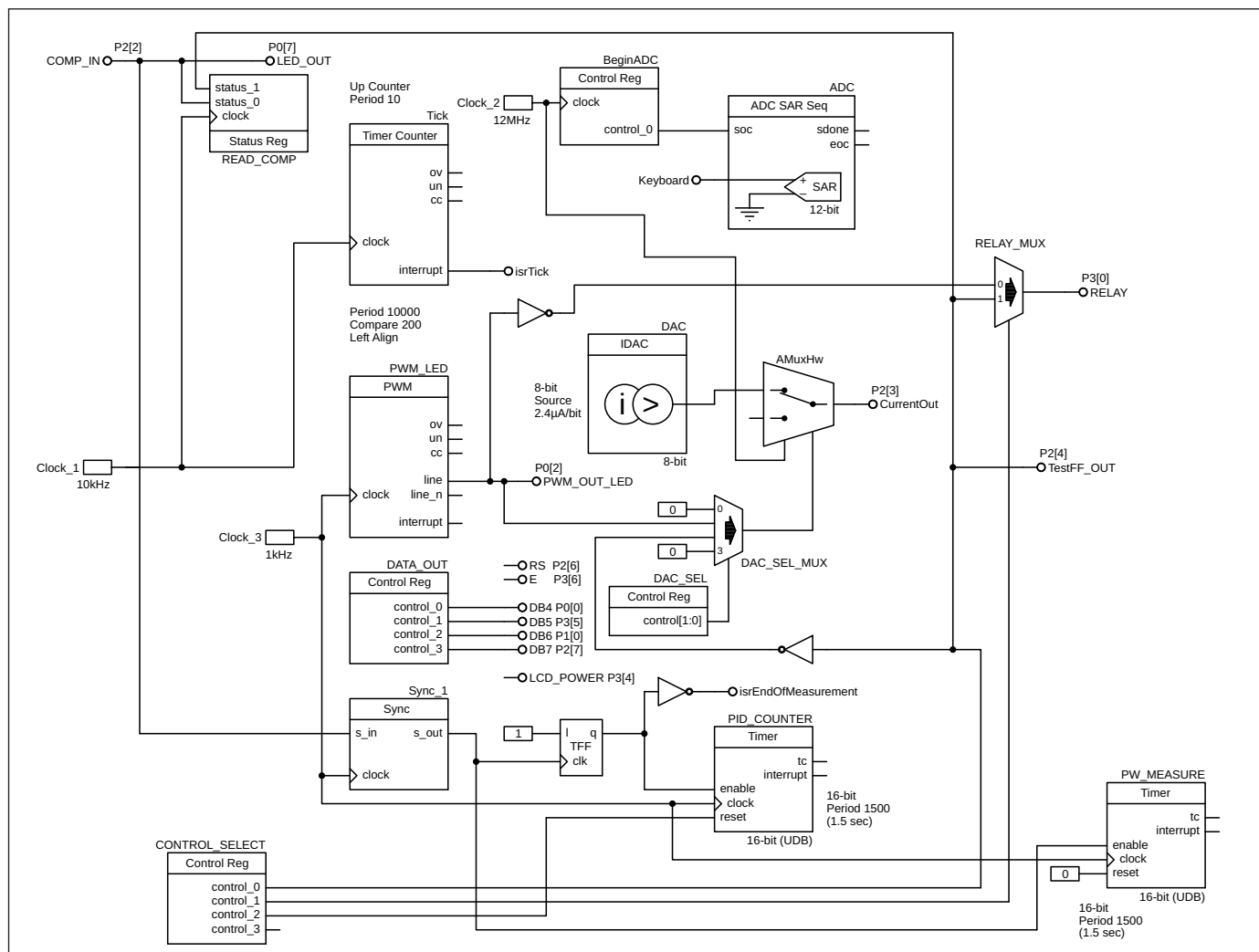


FIGURE 5
Schematic of the internal configuration of the peripherals of the PSoC4. Different functions are enabled/disabled by changing the bits on the CONTROL_SELECT output register.

of supplying up to 20mA, and since the coil is said to be 1,200Ω at 1.5V, it should be capable of supplying the nominal 1.25mA. The coil is connected across terminals X and SCL. The voltage can be set as a parameter using the LCD-based user interface, and the value is stored in EEPROM. This is the first sub-mode useful in proving that the DAC voltage driver works.

Now for the second sub-mode of M1. In later modes we will need to measure the current flow as the contact to the electromagnet closes. We will need to know the time the contact is closed (let's call that P_W) as well as the period of a single cycle of operation (which we will call P_D). To measure these two parameters (which we will also need to do later), I added a series resistor to the current path. In other words, the coil is connected between X and SDA.

Although this would reduce the overall current, it can be compensated by elevating the source voltage. I chose a low value for R5 (Figure 6), and then amplified the voltage so that it drives an LED, D2 and also a comparator (U3, LM393). The LED indicates when current is flowing. The comparator is used as a signal to the PSoC4 that the current is flowing. This signal has been wired to the COMP_IN terminal (Figure 5). The MCU can then measure P_W and P_D , which it shows on the display, and can be noted down. With the addition of this piece of software, no oscilloscope is necessary to take the reading.

M2: In the timing regulator article (mentioned earlier) the principle of operation was that the controller would

generate a regular voltage pulse, and the oscillations of the pendulum would gradually synchronize with them. To try this mode in all its simplicity, I added a reed relay to the design (K1, Figure 6). The regulated supply is connected to one terminal of the relay contact (Y to Z), and the coil across terminals W and SCL. Through the parameters, it is possible to adjust the period the relay is on, as well as the cycle time. But for the experiment, I just wanted the selected period to be greater than P_W and less than P_D . Of course, the LM317 voltage may be adjusted independently using the pot. There is an LED on the Pioneer Kit board, and I used this to indicate when the relay was energized.

Although this is driving a relay (with activation/deactivation and bounce times), I opted to use a hardware timer to drive the relay. The RELAY_MUX multiplexer (Figure 5) is configured via bit CONTROL_2 on the CONTROL_SELECT register to feed the output from the PWM_LED counter to the output. The two parameters are programmed onto the counter.

M3: Mode 3 (M3) is essentially the same as M2, except that it uses the DAC-controlled voltage source, so the DAC output is a third parameter. The relay is fed from the DAC (connect X to Y, Figure 6), and then we can go in one of the two sub-modes, as in M1, for energizing the coil. D2 will flash when the relay is active AND there is current flowing through the coil.

MORE TEST MODES

Before I move on to the next modes, I have to explain why we need to know P_D

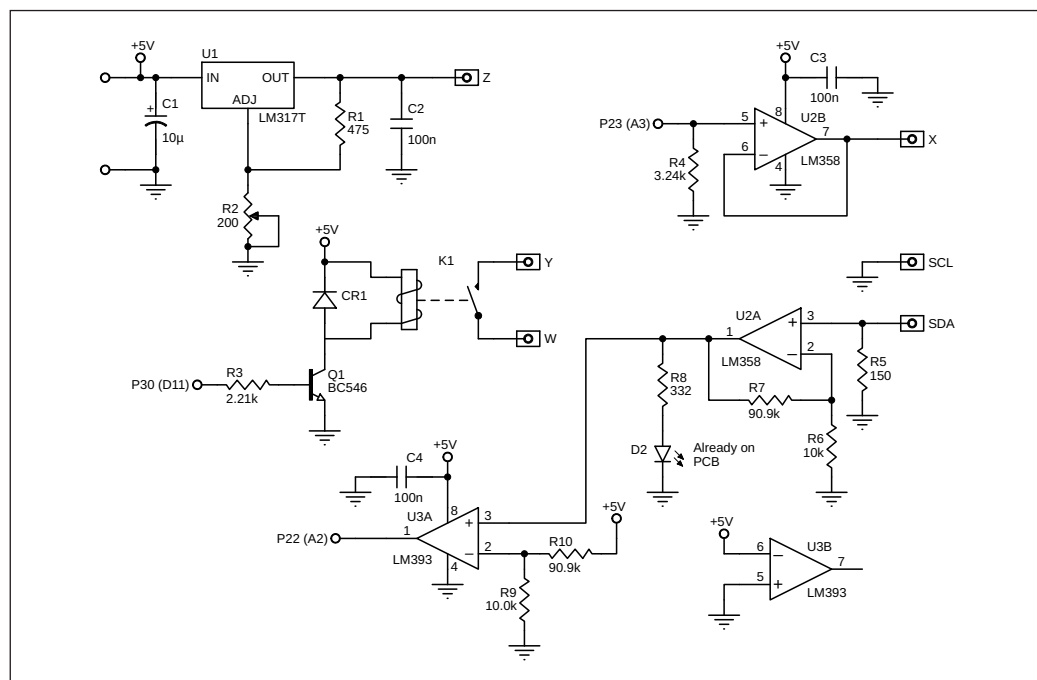
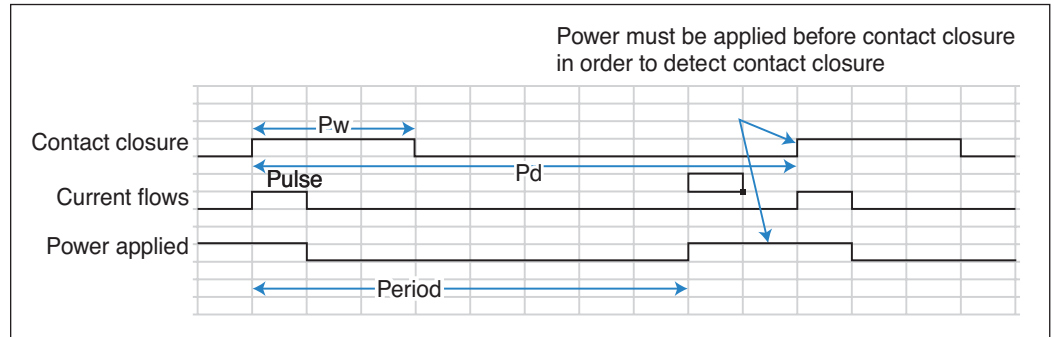


FIGURE 6 Schematic of the external hardware mounted on the prototype shield. The symbols X, Y, W, Z, SCK and SDA correspond to the silk screen on the prototype board associated with some screw terminals. You will notice some connections, such as P30(D11). The first three digits refer to the I/O pin on the PSoC4, and the digits in parentheses refer to the pin name used for the Arduino system.

FIGURE 7

Timing interrelationship



and P_W when we are measuring the current flow through R5. Refer to the timing interrelationship shown in **Figure 7**.

Current can only flow through the clock's contact when the contact is closed and there is voltage. If we are switching the voltage off and on, we need to think about the relative timing. The pulse voltage must be high before the contact closes, or the current through the coil will not reflect the initial closure of the contact. By definition, the pulse turns on and off. If the pulse width (coincident with the start of the current flow) is longer than P_W , it will be limited to P_W . If it is shorter than P_W then it can only be re-energized after P_W , but before P_D , to ensure synchronization with the contact closure.

The same timing concept as in M2 is set up on the PWM_LED timer, but the PWM_LED output is channeled to the AMuxHw by the setting on the DAC_SEL control register (Figure 5), thereby toggling the output current on and off.

M4: This mode (Mode 4) is an extension of M2, along with the current detection connection. The relay pulse is synchronized with the contact closure. The energy imparted to the pendulum is related to the regulated voltage and the pulse time ($\leq P_W$). The pulse time and the voltage restoration time ($>P_W$, but $<P_D$) are configurable variables.

The program waits for the COMP_IN signal, which it reads through the status_0 pin READ_COMP input register (Figure 5). It then initiates the counter sequence, as described in M2.

M5: Mode 5 (M5) mimics M4, except it

drives the coil with the DAC voltage. This adds an additional parameter. The output current is toggled when the COMP_IN signal is seen.

M6: This is the ultimate goal—PID control. We would like to maintain the clock time using a PID technique that modifies the pulse width or voltage. To begin, I just wrote the software for pulse width control. After our initial tests, I think the voltage would be the better control variable. I have not opted to implement a version of this using the on-board relay. I am just switching the output DAC voltage on and off.

SOME CONTROL THEORY

Reiterating some control theory, the calculated output is based on the measured error, as in Equation (1). The overall control function can be expressed mathematically as:

$$u(t) = K_p e(t) + K_I \int_0^t e(t') dt' + K_D \frac{de(t)}{dt'} \quad (1)$$

where K_p , K_I , and K_D are all non-negative and denote the coefficients for the proportional, integral, and derivative terms, respectively. In this case, the parameters are 1,024 times greater than the resulting coefficient. For example, if K_I is 0.046, the parameter would be $0.046 \times 1,024 = 47$. The binary number will make the division much quicker. There are six parameters to this approach. The additional three are scaled coefficients of the PID equation.

The period time is measured on the PW_MEASURE timer (Figure 5), and the time of the output pulse is generated on the PID_COUNTER timer. The timers are controlled in hardware from the COMP_IN signal. Unfortunately, there were not enough resources on the PSoC4 to implement the full PID loop and all the other settings, so the activation of the current output is implemented in software.

NEXT STEPS

From early measurements, I think future development of the PID loop would have to use the output voltage as the control instead

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

References [1] through [7] as marked in the article can be found there

RESOURCES

Cypress Semiconductor | www.cypress.com


Texas Instruments | www.ti.com

of the pulse time, but a lot more development is needed before getting to that.

In addition, we have presumed that in all the modes, the period time would be approximated, and nudged close to the correct period. In truth though, if the gearing ratio from the escapement to the hands were known, this would be a fixed number. The gearing ratio may also vary from Eureka to Bulle, and even different Bulle models. To my knowledge, the ratio has not been published, so the only options left are to count the teeth or, more tediously, to count the number of oscillations for a given time period.

When creating a PCB, we would like to include a boost converter from 1.5V, so that the unit will operate from a battery but maintain a regulated voltage. In turn, that poses a problem choosing the user interface, since the LCD display does draw current. We thought that we could sell a few of these, but didn't know the level of expertise that would be required, so the simplest possible setup was desirable. Some of the literature, however, suggests that temperature and humidity affect the output. It would be possible to monitor these variables over time, and create an historical record with a more sophisticated user interface. Finally, the MCU would be crystal controlled, making the time measurements and pulse

generation much more accurate and tolerant of temperature variation.

Unfortunately for this project, Nigel lives in England and I live in Canada, so this report is inconclusive to date. As an impoverished engineer, I don't have the wherewithal to buy an \$800+ Bulle, much less the \$3,000+ for a Eureka. And then we have to service the unit. Although we have done some preliminary testing, it is clear that this project is going to take a long time to complete, especially because time gains or losses for a particular setting must be measured over several days, at the very least. I hope to check back with you when I have more conclusive results. 

ABOUT THE AUTHOR

Aubrey Kagan has worked in electronics for more years than he cares to remember. He is currently Engineering Manager at Emphatec, an industrial electronics design house in Markham, Ontario. He has written many articles for *Circuit Cellar* over the past 25 years as well as a book *Excel by Example* based on three of those articles. Aubrey was one of the "notable contributors" interviewed in *Circuit Cellar's* 25th Anniversary issue. He has also published several design ideas as well as numerous blogs covering many aspects of electronic design. You can find a list and links to more of his publications on the *Circuit Cellar* article materials webpage. He can be contacted at akagan@emphatec.com.



WIND RIVER™

WIND RIVER SUPPORTS THE BATTLE AGAINST COVID-19

Wind River is donating software and services to those making a difference during the pandemic to accelerate their innovation of mission-critical systems.

JOIN THE BATTLE

windriver.com/covid19

Smart Agriculture Designs Tap IoT Technologies

The Internet of Growing Things

By *Jeff Child,*
Editor-in-Chief

Hungry to get the most productivity out of their operations, farmers large and small are turning to Smart Agriculture solutions. The technologies critical to such systems turn out to intersect sharply with those developed for the Internet-of-Things.

If you look at today's Smart Agriculture landscape, many of the challenges in that space beg for Internet-of-Thing (IoT) solutions. Advanced communication

and sensing technologies are being married to advanced, cloud-based data analytics in order to ensure agricultural production that is both sustainable and resource-efficient.

Like most segments of today's embedded market, Smart Agriculture system developers are leveraging technologies like IoT-centric wireless comms, artificial intelligence (AI), machine learning and computer vision. While some of these solutions are coming from specialist agriculture technology companies, included in the mix are vendors of microcontrollers (MCUs), sensors and wireless interfaces that are feeding the needs of the Smart Agriculture market. Over the past 12 months, a diverse array of products have been rolled out, including solutions at the system, board and chip level.

MCU + IoT SOLUTIONS

Several MCU vendors are seeing their products and technologies meet the needs of Smart Agriculture applications. Not surprisingly, many of those MCU-based solutions are IoT focused. An example is Microchip Technologies' PIC-IoT WG development board (**Figure 1**). It combines PIC24FJ128GA705 MCU, an ATECC608A CryptoAuthentication secure element IC and the fully-certified ATWINC1510 Wi-Fi network controller. The controller provides a simple and effective way to connect an embedded application to the Google Cloud IoT Core,



FIGURE 1

The PIC-IoT WG development board combines a PIC24FJ128GA705 MCU, an ATECC608A CryptoAuthentication secure element IC and the fully-certified ATWINC1510 Wi-Fi network controller.

says Microchip. The board also includes an on-board debugger, and requires no external hardware to program and debug the MCU.

Microchip recently produced a video discussing how IoT technologies are transforming agriculture and vertical farming [1]. The video also includes a discussion of tools—like the PIC-IoT WG—and techniques for successfully implementing a cloud-connected sensor network in your own designs. The video is available on *Circuit Cellar's* article materials webpage. Microchip also penned an article for *AgriTech Tomorrow* about using the PIC-IoT WG boards for Smart Agriculture [2].

On the low power side, Microchip says its SAM IoT board, based on its low power SAM MCUs, is suited for applications like farm sprinkler systems. Microchip's new SAM-IoT WG board connects the Google Cloud IoT Core with Microchip's 32-bit SAM-D21 Arm Cortex M0+ range of MCUs. A requirement in sprinkler systems is to have a low power MCU that can be used to control water flow, manage timings, direction of water flow and so forth. Such systems not only need an MCU to manage the system, but also components like analog sensors and power regulators. And the data from these sprinklers can then be collated and transferred to the cloud using Wi-Fi to provide the required information to manage water efficiently for a given area or farm land.

SMART FLOOD IRRIGATION

In another example of combining the connectivity and sensor input themes of IoT, Prescott Farm Innovations makes a product called WET Stake. WET Stake is a device that notifies flood irrigation farmers when the water reaches the end of the section that they're watering (**Figure 2**). As one of oldest methods of crop irrigation, flood irrigation is on the low end of the irrigation technology spectrum. Among the issues with flood irrigation is the man-hours needed. The farmer typically sends water to a main ditch at the top of the field, he opens gates or starts siphon tubes in one section of the field. He then waits for the water to get down to the bottom of the field and then stops that section and starts the next one.

Most of the man-hours wasted is because of the need to keep checking to see if the watering is done in each section—times can vary from between 20 minutes to up to 5 hours. Addressing this problem, a farmer can simply place WET Stake at the end of the

section he is watering and when the water reaches the right level, WET Stake notifies the farmer by phone call or text. Over watering can reduce the efficiency of fertilizer. WET Stake saves water, prevents over watering crop damage and saves the farmer's time.

According to the company, the IoT technology revolution was perfectly suited to facilitate the development of WET Stake. The design combines an IoT chip along with GPS, and simple sensors inside a custom-built housing. The mechanical design is a simple, durable and familiar shape for farmers—almost like a shovel. It is also solar powered to eliminate any charging downtime.

The WET Stake interface is also very simple, and requires no extra "Smart" technology overhead or processing. The user simply texts the device code to a phone number which gets paired their WET Stake device. All settings can be changed through texting as well. When the WET Stake detects water, the user is notified via call or text and a map link is sent if they want to see the location on a smart device



FIGURE 2

WET Stake is a device that notifies flood irrigation farmers when the water reaches the end of the section that they're watering. The design combines an IoT chip along with GPS. The mechanical design is a simple, durable, and familiar shape for farmers—almost like a shovel. It is also solar powered to eliminate any charging downtime.

map. Other features include a supervisor option for larger operations. This lets one person monitor the watering progress of fields by their hired hands.

ENERGY-HARVESTING SOLUTION

At several conferences over the past 12 months, Renesas Electronics has demonstrated its Silicon-on-Thin-Buried-Oxide (SOTB) technology using a Smart Agriculture example. The demo showed a SOTB agriculture soil monitoring proof of concept that showcases Renesas' SOTB technology at the MCU level by leveraging ambient energy sources such as wind, light, thermal, vibration and flow. Featuring the ultra-low power SOTB R7F0E embedded controller product, the solution allows energy-harvesting technologies to drive sensor networks in environments that require battery-free sensors or sensors that function for long periods without battery replacements.

Last November, Renesas introduced its RE Family that encompassed the company's current and future lineup of energy harvesting embedded controllers. Following the mass production of the RE01 Group (formerly known as the R7F0E embedded controllers), the first of the RE Family, the new RE01 Group Evaluation Kit was launched. The kit enables users working with the RE01 Group of devices to jump start system evaluations for energy harvesting applications (**Figure 3**).

The RE01 Evaluation Kit includes an evaluation board that features an RE01 embedded controller, an interface for the

energy-harvesting device and a rechargeable battery interface. The kit also includes an Arduino-compatible interface for easy expansion and evaluation of sensor boards and a Pmod connector to expand and evaluate wireless functionality. In addition, there is an ultra-low power MIP LCD expansion board so that users can evaluate display functions faster.

The kit also contains sample code and application notes that serve as references for power management designs which eliminate the need for battery maintenance, and driver software that supports CMSIS, Arm's Cortex Microcontroller Software Interface Standard. Sample code for ultra-low power A/D converters, digital filter and FFT (fast Fourier transform) routines, 2D graphics MIP LCD displays, and secure boot and secure firmware update functions for improved security are all available.

The SOTB process technology allows users to simultaneously achieve low active current, low standby current and high-speed operation at low voltage. The RE01's 32-bit CPU core enables users to implement intelligent functions in equipment powered by low levels of harvested energy through ambient energy such as light, vibration or fluid flow.

AI IN SMART AGRICULTURE

Just as they are in most all embedded applications these days, AI and machine learning are having an impact in Smart Agriculture. An example is IntelinAir, an analytics company that provides crop intelligence to farmers through aerial imagery, computer vision, machine learning, agronomic science and intelligent user interfaces. In 2020, IntelinAir plans to document images from close to 5 million acres of farmland across nearly 50,000 fields, collecting over 1 petabyte of raw data. Using computer vision and deep learning approaches, IntelinAir analyzes data to deliver near real-time Smart Alerts to farmers through its flagship product AGMRI.

AGMRI is a field health monitoring and early-warning system that enables farmers to proactively manage their operations (**Figure 4**). AGMRI uses proprietary, patented technology to collect and analyze data from numerous sources. IntelinAir uses AGMRI to gather high-resolution aerial images, temperature readings, humidity measurements, rainfall, soil samples, terrain type, equipment utilized, planting rates, applications and more. They then harness the

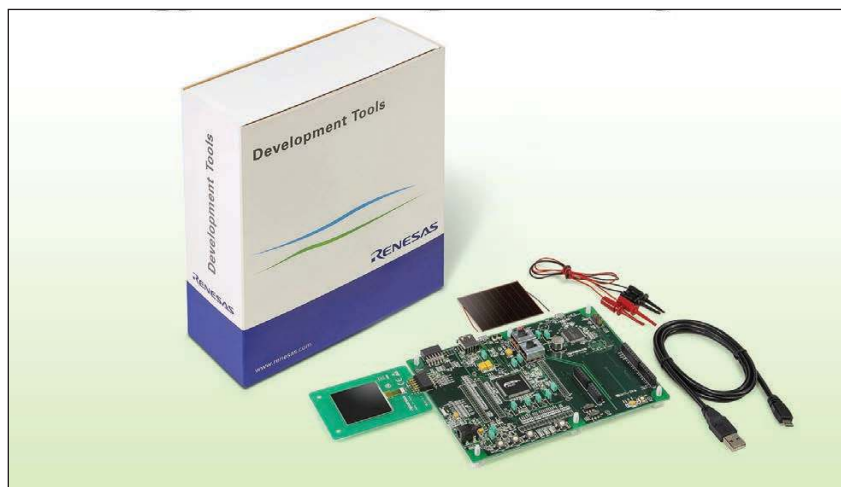


FIGURE 3

The RE01 Group Evaluation Kit includes an evaluation board that features an RE01 embedded controller, an interface for the energy harvesting device and a rechargeable battery interface. The kit also includes an Arduino-compatible interface.

power of hyperspectral analysis, computer vision and deep learning in order to identify patterns and build a complete and precise situational representation of every monitored field for the entire growing season.

Information is continuously aggregated, correlated and strengthened by remembering, relating and connecting past and present situations. As the system trains on new data, it becomes stronger, smarter and more effective every day. AGMRI identifies abnormal crop conditions long before the human eye can detect them and tracks their progress from week to week.

The system's AI cognitive decision-making engine has already processed hundreds of terabytes of crop images across multiple seasons. The company uses Big Data and AI to enable what it is says is a previously unattainable class of computation for agriculture. AGMRI uses self-learning algorithms to perform precise predictive analytics, which remove the sampling errors typically associated with relying on scouting efforts alone. Computer vision processes the imagery captured via the aerial sensors to extract meaningful environmental features such as bare ground, biomass, reflection, chlorophyll content and plant heights while removing the noise and clutter.

ANDROID-BASED SYSTEM

System level solutions for Smart Agriculture are rapidly evolving and, like component solutions, are leveraging the latest communications technology. Exemplifying these trends, in November, Trimble Agriculture introduced its GFX-350 display and NAV-500 guidance controller (**Figure 5**). The solutions were designed to provide a cost-effective option for farmers seeking to adopt the latest precision agriculture technology for their daily operations.

Continuing a tradition of Android-based high-definition touchscreen displays, the GFX-350 display is a cost-effective way to introduce auto-steering and application control to the farm. The 7" (18cm) screen is easy to read and can be used to control most field operations with just a few taps. The display is compatible with both the new NAV-500 and the NAV-900 guidance controllers, satisfying different user accuracy needs. The simple and intuitive Precision-IQ operating system speeds up field work and makes equipment configuration easy, says Trimble. Once vehicles, fields, implements and materials are set up during



FIGURE 4

AGMRI is a field health monitoring and early-warning system that enables farmers to proactively manage their operations. It uses patented technology to collect and analyze data and sources including high-resolution aerial images, temperature readings, humidity measurements, rainfall, soil samples, terrain type, equipment utilized, planting rates and more.

the first use, they are saved and can be reused with a couple of clicks.

In addition, the GFX-350 display is fully ISOBUS compatible, offering plug-and-play capability for ISO-enabled implements with native task controller and universal terminal functionality. The display also features onboard Wi-Fi and Bluetooth connectivity, allowing seamless sharing of data between the office and the field via optional Trimble Connected Farm solutions.

The NAV-500 guidance controller features a low-profile rugged housing capable of receiving signals from five different GNSS satellite constellations—GPS, Galileo, GLONASS, BeiDou and QZSS. This precision solution offers sub-meter repeatable accuracy and full-farm coverage ideal for tillage, broad-acre seeding, spraying and harvest



FIGURE 5

The GFX-350 display and NAV-500 guidance controller were designed to provide a cost-effective option for farmers seeking to adopt the latest precision agriculture technology for their daily operations.



FIGURE 6

The Healtag is small and light enough to be attached to an animal's earflap like a traditional livestock ear tag. Using the nRF9160 SiP meant they don't have to worry about having separate application processor, antenna, GPS and NB-IoT circuits.

operations. By using Trimble's ViewPoint RTX satellite-delivered correction service with the NAV-500, operators can consistently achieve 15cm pass-to-pass accuracy. Paired with either the new GFX-350 display or larger 10" (25.4cm) GFX-750 display, the NAV-500 can provide roll-corrected manual guidance or can automatically control steering with the EZ Steer assisted steering system and EZ Pilot Pro steering system.

SMART EAR TAG EXAMPLE

In another example of IoT technology being used for Smart Agriculture, Nordic Semiconductor says its nRF9160 System-in-Package (SiP) LTE-M/NB-IoT cellular IoT module is being employed in an IoT-enabled herding livestock management solution developed by Finnish startup, Anicare, called the Anicare Healtag. The Healtag became commercially

available in September last year. Healtag is designed to ensure farmers of commercially bred reindeer and other herding animals against the financial loss and livestock welfare issues associated with undetected illness or injury of herding animals that spend most of the year roaming in the wild.

Anicare says Healtag is a significant improvement over existing herding animal trackers that are so large they have to be hung from the animal's neck, and consume so much power that their batteries have to be replaced every year, which is not only expensive and time consuming for the farmer, but also highly stressful for the animal.

In contrast, by employing a highly miniaturized, low-power Nordic nRF9160 SiP, the 25g, 35mm x 22mm x 23-mm Anicare Healtag is small and light enough to be attached to an animal's earflap like a traditional livestock ear tag (**Figure 6**). It offers a maintenance-free battery lifetime of up to five years, which means the tag only needs to be attached once to the animal during its lifetime. The all-in-one module integration of the nRF9160 SiP means they don't have to worry about having separate application processor, antenna, GPS and NB-IoT circuits that would require a lot of supporting components and thus more PCB space.

In operation, the Anicare Healtag autonomously measures a herding animal's activity (using an accelerometer) and heat (using a thermal sensor) once every hour, and uses the latest NB-IoT cellular wireless technology to report of any significant changes that would tend to indicate either illness, injury or predator attack. This includes using the Nordic nRF9160 SiP's built-in GPS functionality to immediately send the exact location of a distressed animal to its owner, enabling rapid rescue and treatment. And Anicare says that in terms of coverage, the latest NB-IoT cellular wireless technology deployed throughout Northern Europe ensures cellular data communication remains possible even in areas with no 2G cellphone coverage.

GNSS AND RF FOR TRACKING

An important component of Smart Agriculture includes the ability to precisely track the position of crops and livestock. Here, access to the GNSS (Global Navigation Satellite System) provides a powerful solution to meet such needs. Along those lines, in December U-blox announced that Taoglas developed a centimeter-level GNSS positioning solution. The system comprises a high-precision L1/

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

References [1] and [2] as marked in the article can be found there.

RESOURCES

ACEINNA | www.aceinna.com

IntelinAir | www.intelinair.com

Microchip Technology | www.microchip.com

Nordic Semiconductor | www.nordicsemi.com

Prescott Farm Innovations | www.wetstake.com

Renesas Electronics | www.renesas.com

Semtech | www.semtech.com

STMicroelectronics | www.st.com

Trimble Agriculture | <https://agriculture.trimble.com>

U-blox | www.u-blox.com

L2/E5 GNSS receiver, the U-blox ZED-F9P, all the required RF electronics and antennas in a single package (**Figure 7**). Called Taoglas Edge Locate, this positioning module simplifies the development and deployments of IoT solutions that depend on high-precision positioning information.

Taoglas Edge Locate addresses the growing demand for highly accurate centimeter-level positioning performance, which, until recently, was reserved for high-value use cases such as guidance systems for precision agriculture and heavy machinery, says U-blox. This changed with the release of additional satellite signals and the announcement of U-blox F9 high-precision positioning platform, which lowered the cost of ownership of the technology, extending its benefits to mass market applications for the first time, according to U-blox.

Featuring the U-blox ZED-F9P high-precision GNSS module with concurrent reception of GPS, GLONASS, Galileo and BeiDou on multiple frequency bands, the Taoglas Edge Locate module can also use real-time kinematic (RTK) algorithms to help



FIGURE 7

Taoglas Edge Locate is a positioning module that comprises a high-precision L1/L2/E5 GNSS receiver, the U-blox ZED-F9P, and all the required RF electronics and antennas in a single package.

achieve even faster convergence times and reliable performance, even in highly dynamic applications. The integrated smart antenna is specifically designed and optimized for multi-band GNSS applications.

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

Verilog HDL

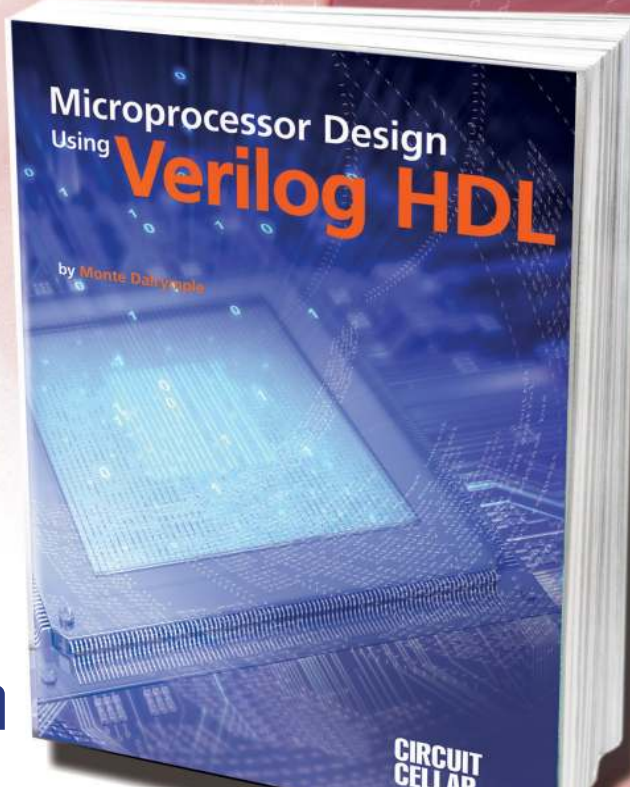




FIGURE 8

Well suited for precision agriculture, the OpenRTK330L device is a triple-band RTK/GNSS receiver with built-in triple redundant inertial sensors. It's designed to replace the expensive and bulky precision RTK/INS systems used in today's autonomous systems.

High-precision positioning enables a range of use cases like precision agriculture, but also emergency response, smart infrastructure, drone delivery and micro-mobility. Edge Locate's RTK positioning capabilities let end users benefit from centimeter-level positioning without subscribing to GNSS correction services, relying instead on a local RTK network that Taoglas can also help customers design and set up.

3x REDUNDANT SENSORS

In another example of a GNSS solution suited to precision agriculture, ACEINNA in January announced the availability of its OpenRTK330L device, a triple-band RTK/

GNSS receiver with built-in triple redundant inertial sensors (**Figure 8**). Designed to replace the expensive and bulky precision RTK/INS systems used in today's autonomous systems, this compact navigation solution meets the performance, reliability and cost requirements of Smart Agriculture systems, as well automotive, robot, drone and construction systems.

ACEINNA's OpenRTK330L includes a triple-band RTK/GNSS receiver coupled with redundant inertial sensor arrays to provide centimeter-level accuracy, enhanced reliability and superior performance during GNSS outages. The OpenRTK330L integrates a very precise 2 degree/hour inertial measurement unit (IMU) to offer 10 to 30 seconds of high-accuracy localization during full GNSS denial. This enables autonomous system developers to safely deliver highly accurate localization and position capabilities in their vehicles at prices that meet their budgets. OpenRTK330L's embedded Ethernet interface allows easy and direct connection to GNSS correction networks around the world. OpenRTK330L's CAN bus interface allows simple integration into existing vehicle architectures.

The multi-band GNSS receiver can monitor all global constellations (GPS, GLONASS, BeiDou, Galileo, QZSS, NAVIC, SBAS) and simultaneously track up to 80 channels. The module has RF and baseband support for the L1, L2 and L5 GPS bands and their international constellation signal equivalents.

The IMU and dead reckoning function contains a total of 9 accelerometer and 9 rate gyro channels based on ACEINNA's unique triple redundant 6-Axis IMU array. By integrating a triple-redundant IMU array, the OpenRTK330L is able to recognize and utilize only valid sensor data, ensuring high-accuracy protection limits and certifiability under ISO26262 standards. The OpenRTK330L is supported by ACEINNA's Open Navigation Platform allowing

CATTLE HEALTH MONITORING

Beyond just position tracking of livestock, IoT technologies are also being used to monitor the health of animals. In an example along those lines, last Fall Semtech announced that ITK, a French supplier of IoT-based Smart Agriculture applications, developed a new cattle health-monitoring solution based on Semtech's LoRa devices. The FarmLife Smart Agriculture service and its LoRa-enabled



FIGURE 9

The FarmLife Smart Agriculture service and its LoRa-enabled sensors detect cattle estrus, drive improved nutrition and predict the onset of disease to help ranchers better monitor their herds.

sensors detect cattle estrus, drive improved nutrition and predict the onset of disease to help ranchers better monitor their herds (Figure 9).

According to ITK, LoRa devices' flexibility in deployment makes a key difference for connecting animals and offers the potential for a significant return on investment (ROI). ITK's solution provides ranchers with tangible, actionable data on the health of their herds to remove variables from ranching and create productive, efficient and profitable ranches.

Offering a flexible solution for the Smart Agriculture vertical, Semtech's LoRa devices create applications that help minimize waste, maximize yield, reduce expenses and offer farms and ranches an opportunity to operate as efficiently as possible. ITK's LoRa-based sensors deploy simply through a collar equipped to each animal. The collar is non-invasive and immediately begins reporting data on the cow's health upon deployment.

Ranchers monitor their herds from this LoRa-based device, which provides the benefits of four value-added services: Heat'Live for heat detection, Feed'Live for nutrition optimization,

Time'Live for animal welfare and Vel'Live for calving detection. These services are available through ITK's FarmLife Cloud platform. In total, deploying ITK's solution costs less than 30 Euros per animal annually.

In addition to the 300,000 cows already monitored in Europe, approximately 20 ranches have deployed ITK's FarmLife platform in North America, connecting cows to the cloud through network connectivity from X-TELIA, a leading Canadian network provider. Following this initial deployment, farmers claimed they received an ROI in less than a year through an increase in ranch productivity and efficiency. X-TELIA and ITK plan to continue the rollout of this Smart Agriculture solution in the Canadian market, adding ITK's San'Phone cattle health monitoring service and its Thermo-bolus sensor to connect up to 2.4 million cows in the future.

Clearly there's a lot of interesting activity happening in the Smart Agriculture space. And its overlap with IoT will only grow as system developers seek out highly integrated, wirelessly connected solutions linked together with sophisticated cloud-based oversight and monitoring. 

easyOEM

Here Your Design Comes True!

www.easyOEM.com
email: sales@easyoem.com
Toll-free: +1(866)66-EZOEM

One-stop Electronics Manufacturing Service

- PCB Fabrication
- BOM Kitting
- Assembly
- Testing

New Product 50% OFF!
(for first order up to \$1,000)

We do much more...

- Design Checking -PCB Coating -MCU/CPLD/FLASH/EEPROM Programming
- Wiring Harness Assembly -Injection molding
- Printing -Labeling -Packaging -Certification
- Stocking -Distribution -Shipping -Customs Declaration

"Just focus on your design. We do the rest for you!"

Embedded Software Tools Bulk Up on Security

Connected System Concerns

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

The evolution of embedded systems into complex, connected systems continues to provide new challenges for embedded software tool vendors. But they are rising to the moment, as security becomes more of a priority than ever.

Gone are the days when most embedded systems worked in isolation, not linked to any networks. In order to reap the opportunities of a hyper-connected IoT era, today's embedded systems are routinely linked for purposes of monitoring, data collection, software updates and more. As a result, security has moved front and center for embedded software developers.

To keep pace, embedded software tool vendors continue to bulk up their security capabilities. They've been doing this both organically, and by adding expertise via key acquisitions and partnerships. Even though all these vendors are addressing a similar need, the major embedded tool vendors each have their own approaches when it comes to providing security capabilities.

SOFTWARE IP IN THE IoT AGE

According to Anders Holmberg, Chief Strategy Officer at IAR Systems, the need for security in today's embedded software goes hand in hand with the emergence of highly connected systems. In today's environment, complexity is no longer the only challenge. "When IoT entered the collective mind as a key enabler for new opportunities, that led to increased connectivity and talk about how to protect these connected devices," says Holmberg. "In recent years, the trend we've seen is about the importance of protecting software IP. Today, most of the business value of embedded devices is due to the software, so better protecting embedded devices and the software they run is a key activity to stay competitive."

Holmberg says that security becomes more and more relevant as more data is collected from both machines and humans, raising such questions about how to manage and store sensitive and functional data. The design of automatic update processes is also an issue.

"A side effect of implementing IP security based on cryptography and strong hardware-based roots of trust is that it enables more fine-grained control of what features are available to a particular user at a given point in time—which in turn enables new business models," says Holmberg.

For its part, IAR Systems has made a number of advances in recent years focused on security. First, it offers C-Trust, an extension to IAR's development toolchain IAR Embedded Workbench, which enables developers to add security into the normal development flow and easily protect the application and deliver secure, encrypted code. **Figure 1** shows how to enable C-Trust in IAR Embedded Workbench by simply clicking a checkbox.

IAR Systems also offers its Security from Inception Suite. It's aimed at companies looking for a solution to implement and customize security in their applications and to learn more about both how to deal with security in the development teams as well as take advantage of the coming possibilities on a company level. The suite enables developers to build a platform, which will extend with evolving security needs as threats appear, and as legislation impacts the business. It's available in different editions and also includes extensive security training resources, and if needed, custom design reviews.

IAR's toolchain is available in editions certified for functional safety. These editions are certified by the certification organization TÜV SÜD, according to the requirements put forth in the industry standards ISO26262 (automotive), IEC61508 (industrial control), EN50128/ EN50657 (rail transportation) and IEC62304 (medical devices). Along with the toolchain, there is special support for these functional safety editions that allow access to frozen versions of the toolchain for the longevity of the customer's contract, as well

as prioritized technical support and validated services packs.

OPTIMIZED AGILE DEVELOPMENT

For its latest security-related advancement, LDRA in February teamed up with Atlassian, integrating that company's Jira software to optimize agile development and verification of critical embedded applications. Embedded developers working in safety- and security-critical organizations must demonstrate compliance with industry functional safety and security standards, and to do this they are making the shift toward agile development methods, says LDRA. The new integration gives development organizations an agile solution that optimizes workflows with requirements traceability and automates software quality analysis and verification as well as documentation production.

The LDRA TBmanager Integration Package for Jira delivers bidirectional end-to-end traceability from Jira issues and test cases to requirements, design, code and testing activities and artifacts (Figure 2). This integration supports and enables both Scrum and Kanban agile workflows to address the requirements of critical software safety standards such as DO-178B/C (aerospace and defense), IEC62304, ISO26262, EN50128, IEC60880 (nuclear energy) and IEC61508 applications.

Bidirectional interface and exchange of requirements capabilities, along with test case and test execution results, enable users to see the status and verification of requirements reflected in Jira. Furthermore, developers can verify traceability through Jira's traceability matrix report and thereby ensure all documented issues in Jira and imported requirements have been addressed. The LDRA TBmanager Integration Package for Jira is available from version 9.8.1 (and newer) of the LDRA tool suite. Users can download a free 30-day trial of the LDRA tool suite with the TBmanager Integration Package for Jira.

SECURITY TESTING

Tools that rigorously test embedded software for security form an important part of today's development tool chains. For its part, GrammaTech announced the availability of its CodeSonar version 5.2. in December. The features in that latest version of CodeSonar provide software development organizations the capability to use a single tool to perform Static Application Security Testing (SAST) to further increase code security, quality and safety covering both embedded and enterprise applications (Figure 3).

CodeSonar now supports AUTOSAR C++14, the latest C++ coding guidelines

from AUTOSAR. With MISRA compliance included in previous releases, the addition of AUTOSAR support now sets CodeSonar at the forefront of the MISRA/AUTOSAR merging of standards. The release of CodeSonar 5.2 also includes improved compiler support and open standards, with support for new versions of the IAR, GNU C, and CLANG compilers. Updates to C, C++-17 and C++-20 standards have also been incorporated, providing customers with the confidence that CodeSonar support spans from old to new language features. GrammaTech continues its work on open standards, including contributing to and supporting SARIF version 2.1. This support also means that CodeSonar can work with the latest versions of IDEs such as Microsoft VS Code.

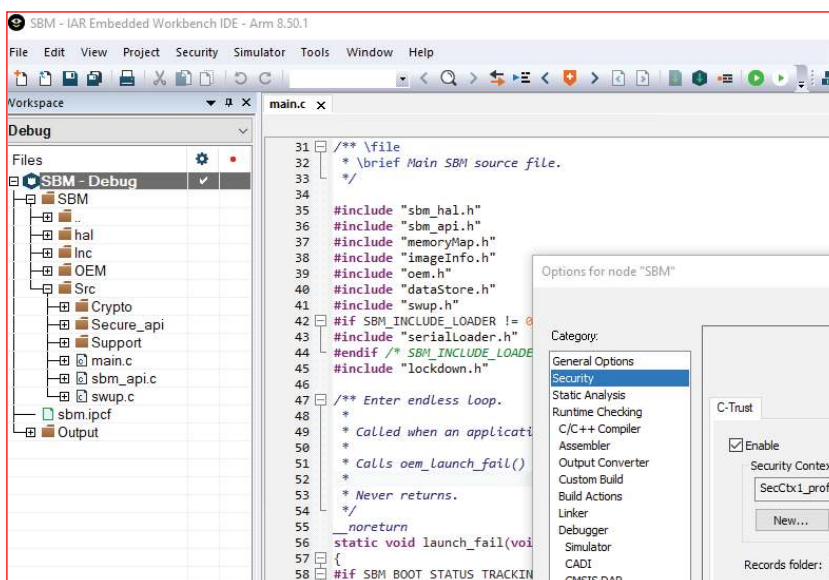


FIGURE 1 C-Trust is an extension to the development toolchain IAR Embedded Workbench, which enables developers to add security into the normal development flow and easily protect the application and deliver secure, encrypted code. As shown, users can enable C-Trust in IAR Embedded Workbench by simply clicking a checkbox (lower right).

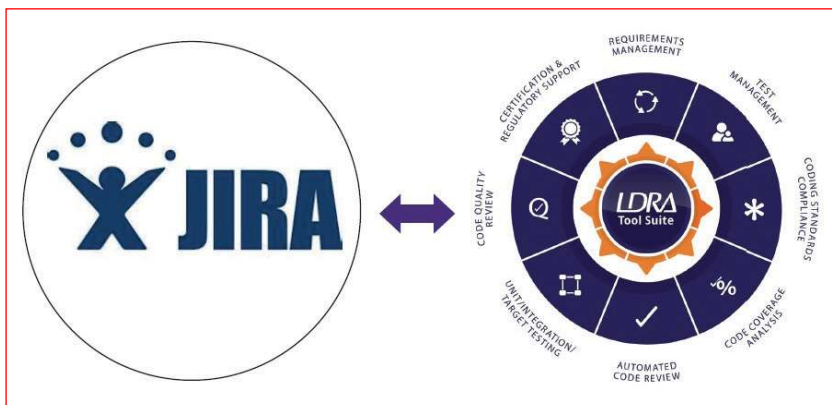


FIGURE 2 The LDRA TBmanager Integration Package for Jira delivers bidirectional end-to-end traceability from Jira issues and test cases to requirements, design, code and testing activities and artifacts.

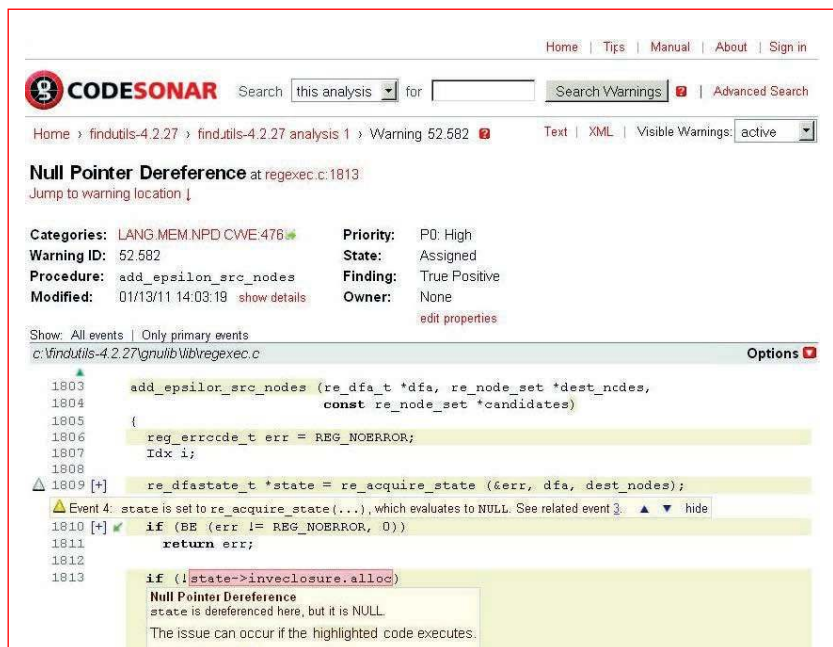


FIGURE 3

Enabling more secure code, CodeSonar lets you see the path to each flaw and how it can occur.

CodeSonar 5.2 continues its tight integration with JuliaSoft by supporting the latest release of the Julia engine, which provides high recall, high-precision detection of security vulnerabilities in Java and C#. In addition, GrammaTech is expanding support for CodeSonar for Binaries to include support for the Power architecture (PPC) in addition to the existing support for x86 and Arm architectures. The addition of the Power architecture support for CodeSonar for Binaries widens the scope of the product to another key processor family used in embedded and server-based systems, such as devices from NXP and IBM. The update is available as a free upgrade to eligible customers under active support and maintenance contracts. A 30-day trial of CodeSonar 5.2 is also available.

AUTOMATED TEST APPROACH

AdaCore has likewise beefed up its security testing capabilities. In June 2019, AdaCore announced a partnership with Code Dx, a provider of an application security management solution that automates and accelerates the discovery, prioritization and risk management of software vulnerabilities. Through this partnership, Code Dx Enterprise

now supports AdaCore's CodePeer advanced static analysis tool, an automatic Ada code reviewer and validator.

The solution provides developers with one central location from which to view the results of multiple application security testing (AST) tools and allows them to easily prioritize vulnerabilities for remediation. Developers can automatically pull results from AdaCore's CodePeer into Code Dx Enterprise, without downloading and then uploading scan results each time. Users simply open Code Dx Enterprise and the latest results are there.

Code Dx Enterprise supports and integrates with more than 70 commercial AST tools and techniques, including static, dynamic, and interactive tools; third-party component analyzers; and manual reviews, to provide total software application vulnerability correlation and management. The tool enables AdaCore users to more easily collaborate on testing and remediation processes, and to track findings over time.

For CodePeer users who are developing multi-language software within the same application, Code Dx Enterprise provides a single repository to manage all of their AST activities. The CWE-Compatible CodePeer advanced static analysis tool is an automatic Ada code reviewer and validator that can detect and eliminate errors both during development and retrospectively on existing software (**Figure 4**). CodePeer can detect a number of the "Top 25 Most Dangerous Software Errors" in the MITRE Corp.'s Common Weakness Enumeration (CWE).

SECURE AUTOMOTIVE SOLUTION

Automotive applications are an area where security and safety concerns intersect. In February, Green Hills Software and automotive technology specialist Tata Elxsi announced their partnership to develop software-driven, highly integrated automotive cockpit solutions. At Embedded World earlier this year, the companies showcased the first result of their cooperation: Tata Elxsi's eCockpit solution running on Green Hills Software's safe and secure INTEGRITY real-time operating system (RTOS) and INTEGRITY Multivisor secure virtualization.

The Tata Elxsi eCockpit solution addresses the requirements of a full feature vehicle cockpit, supporting infotainment, instrument cluster, HUD and ADAS functionalities on a single SoC while maintaining the highest levels of safety, security and performance (**Figure 5**). The demonstration at the show paired Tata's eCockpit with the Green Hills ASIL-certified INTEGRITY RTOS and its Multivisor secure virtualization architecture to safely and securely consolidate mixed-

RESOURCES

AdaCore | www.adacore.com

GrammaTech | www.grammatech.com

Green Hills Software | www.ghs.com

IAR Systems | www.iar.com

LDRA | www.ldra.com

Wind River | www.windriver.com

criticality applications on a single, automotive-grade Renesas R-Car H3 processor.

INTEGRITY Multivisor runs Linux and Android in independent, secure virtualized partitions. Tata Elxsi Infotainment is based on Automotive Android and the instrument cluster is running on Linux. Infotainment features are shown through a 2D/3D custom HMI on Automotive Android. V2X features are also integrated and displayed on the instrument cluster as warning messages. Linux guest OS is partitioned using Linux Containers to accommodate sub domains like ADAS. A separate Linux Container runs Tata Elxsi's Sensor Fusion ADAS IP over Tata Elxsi's own Adaptive AUTOSAR. Complete vehicle interface functionality is based on Tata Elxsi's own classic AUTOSAR 4.3.

The INTEGRITY RTOS microkernel architecture is designed for critical embedded systems demanding proven separation, security and real-time determinism. Its separation architecture helps software teams to safely and securely partition software running at different levels of criticality on the Renesas R-Car H3 processor while guaranteeing applications have the system resources required for their proper execution. This enables safe and secure execution of applications running graphics and multimedia while at the same time ensuring the safe operation of critical functions, such as the tell-tale status and warning lights.


CYBER SECURITY EXPERTISE

Another way that embedded software vendors have been bolstering their security capabilities over the past several months has been by integrating capabilities through acquisitions. In an example along those lines, in January Wind River announced its acquisition of Star Lab, a specialist in cybersecurity for embedded systems. According to Wind River, the acquisition broadens the Wind River software portfolio with a system protection and anti-tamper toolset for Linux, an open source-based hypervisor and a secure boot solution. Star Lab is now a wholly owned subsidiary of Wind River.

With the emergence of ubiquitous connectivity paradigms such as IoT and remotely monitored/autonomously controlled industrial and transportation systems, today's cyber threat landscape is rapidly evolving, says the company. Central to this evolution is the ease with which a focused and resourced adversary can acquire and reverse engineer deployed embedded systems. In addition to modification or subversion of a single specific device, hands-on physical access also aids an attacker in discovery of remotely-triggerable software vulnerabilities.



Specializing in cyber and anti-tamper security software for Linux, Star Lab provides embedded security for the most mission-critical systems, infrastructure and equipment in the world. Star Lab's products are founded on a secure-by-design engineering philosophy, leveraging design patterns that reduce attack surface, isolate critical functionality and contain or mitigate even successful attacks.

Star Lab's products, which are conformant with NIST 800-53 technical controls for federal information systems and consistently pass independent verification/validation testing, include the following: Security Suite: The suite offers robust Linux cybersecurity and anti-tamper capabilities for operationally deployed Linux systems and distributions; Embedded Hypervisor: Designed specifically for use in open, hostile computing environments, the Xen-based hypervisor offers a secure open source virtualization solution for embedded mission systems; and Secure Boot: A measured-boot solution ensures that a device's firmware and boot code is legitimate and has not been maliciously modified or manipulated. 

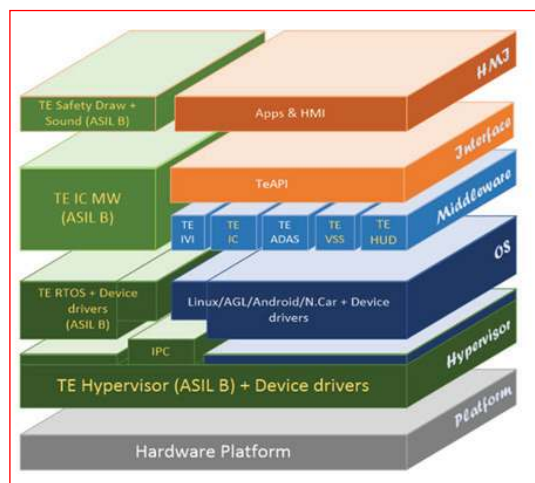


FIGURE 4 The CWE-Compatible CodePeer advanced static analysis tool is an automatic Ada code reviewer and validator that can detect and eliminate errors both during development and retrospectively on existing software.

FIGURE 5 Tata Elxsi's eCockpit solution runs on Green Hills Software's safe and secure INTEGRITY RTOS and INTEGRITY Multivisor secure virtualization. The Tata Elxsi eCockpit solution addresses the requirements of a full feature vehicle cockpit, supporting infotainment, instrument cluster, HUD and ADAS functionalities on a single SoC.

Datasheet: Mini-ITX and Pico-ITX SBCs

Performance Platforms

DATASHEET

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



FIGURE 1

Pico-ITX technology was embraced by at least one of the teams in Audi's 2017 annual Audi Autonomous Driving Cup (AADC). Participants develop fully automatic driving capabilities and the necessary software architectures. These are then put to the test in 1/8th-scale model cars. (Source: Audi).

Based on the small-sized versions of the ITX motherboard form factor, Mini-ITX and Pico-ITX keep growing in popularity and in embedded market share. These SBCs provide system developers with complete PC-functionality and advanced graphics.

There was a time when large slot-card based form factors were the only choices for embedded systems. Those days are gone, now that a complete computing solution can be designed into a small form factor embedded motherboard. Among these so-called bus-less embedded form factors are the various versions of the ITX. They offer a more complete SBC approach, integrating most or all of the typical desktop PC kinds of functions. Applications where graphics are a priority are particularly suited to these types of board-level products.

While the ITX form factor is based on the ATX PC motherboard standard, what's more popular in recent years are its spinoffs Mini-ITX and Pico-ITX. Mini-ITX is a 170 mm x 170 mm (or 6.7" x 6.7") low-power motherboard form factor developed by VIA Technologies in 2001. They are commonly used in small form factor computer systems. A more recent variant is the Thin Mini ITX, a version of Mini-ITX that is only 22mm in height, with a thinner port cluster and horizontally stacked SO-DIMM memory slots. Meanwhile, Pico ITX is a PC motherboard form factor released by VIA

Technologies in January 2007. The form factor was transferred over to SFF-SIG in 2008. The Pico-ITX form factor specifications call for the board to be 100mm x 72mm (3.9" x 2.8"), 75% smaller than the Mini-ITX form factor.

Pico-ITX technology was embraced by at least one of the teams in Audi's 2017 annual Audi Autonomous Driving Cup (AADC), a competition for engineering students. Participants develop fully automatic driving capabilities and the necessary software architectures. These are then put to the test in 1/8th-scale model cars. Specifically built for the competition by Audi, these serve as hardware platforms (Figure 1). The team named "FASzination - Autonom" from Hochschule Kempten, which was supported by Kontron, built their model car with 10 ultrasound sensors and 2D/3D cameras (RGB and deep image). A wheel speed sensor and six axis motion sensors for angular velocity and acceleration submit their data to the model car's control unit. For that control unit, the team chose Kontron's Pico-ITX pITX-E3845 SBC with 4-core 1.9 GHz Atom, 8GB RAM and 60GB SSD. [E](#)

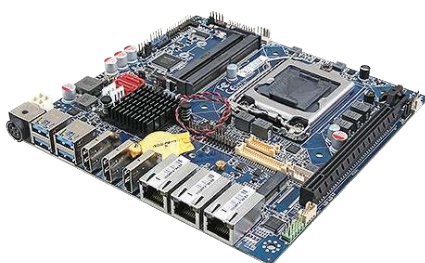


Whiskey Lake-U Processor Rides Mini-ITX

The IMB-1216 from ASRock Industrial Computer was among early Thin Mini-ITX boards to sport a Whiskey Lake processor. The board offers a 170mm x 170mm Mini-ITX footprint with a low profile. It supports industrial applications with a semi-extended 0 to 60°C range and offers a choice of 12V or 19V-28V DC inputs with an AT/ATX switch.

- Intel 8th Gen (Whiskey lake-U) Core MCP Processors
- Up to 32GB Channel DDR4
- 1x DisplayPort or VGA, 1x HDMI, 1x LVDS or eDP
- 5xUSB 3.1, 3xUSB 2.0, 2x SATA3, 6x COM
- 1x M.2 Key E, 1x M.2 Key B, 1x M.2 Key M
- 1x Intel LAN, 1x Realtek LAN
- 1x TPM Header (IMB-1216V), 1x TPM 2.0 IC onboard (IMB-1216M/P)
- +12V to 28V DC-in (DC Jack / 4-pin ATX PWR Con)

ASRock Industrial Computer
www.asrockind.com



Mini-ITX Card with 8/9th-Gen iCore or Celeron Processors

Avalue Technology's EMX-H310DP is a Thin Mini ITX board with 8/9th Gen Intel Core i3/i5/i7 and Celeron SoC Processor. It is a Triple Gigabit LAN industrial motherboard suitable application for network devices, NAS Server, media center, Industrial control systems and other embedded applications.

- Intel LGA1151 Socket Supports 8/9th Gen Xeon, Core i7/ i5/ i3, Pentium and Celeron Processors
- Up to 64GB in dual DDR4 SO-DIMM sockets
- Supports Max
- 3x HDMI, 1x dual-channel LVDS
- 2x Intel i210AT & 1x Intel i219LM Gbit Ethernet
- 3x USB 3.1 Gen 2, 1x USB 3.1 Gen 1, 3x USB 2.0
- 3x RS-232, 1x RS-232/422/485, 16-bit GPIO
- 1x PCI-ex16, 1x M.2 Key B, 1x M.2 Key E
- DC in +12V-28V 4-pin DC-In

Avalue Technology
www.avalu.com.tw



Mini-ITX Motherboard Features Advanced Graphics

The MANO521 from Axiomtek is a Thin Mini-ITX motherboard powered by the LGA1151 socket 9th/8th generation Intel Core i7/i5/i3 (code name: Coffee Lake) with Intel H310 chipset or optional Intel Q370 chipset. This board provides rapid video acceleration advantage, multiple expansion interfaces and triple-view capability.

- LGA1151 9th/8th gen Intel Core i7/ i5/ i3 processor
- Intel H310 chipset (Q370 optional)
- 2x DDR4 SO-DIMM up to 32GB of memory
- 4x USB 3.0 and 4x USB 2.0 ports
- 2x COM ports
- 2x SATA-600, mSATA and M.2 Key M (NVMe)
- PCIe x4, M.2 Key E, PCI Express Mini Card slot

Axiomtek
www.axiomtek.com

DATASHEET URLS:

ASRock Industrial Computer <https://download.asrock.com/Download/e-catalog/IMB-1216.pdf>

Avalue Technology www.avalu.com.tw/products/Industrial-Embedded-Motherboard/Thin-Mini-ITX/Thin-Mini-ITX/EMX-H310DP_2887

Axiomtek <https://us.axiomtek.com/Download/Spec/en-US/mano521.pdf>

Mini-ITX and Pico ITX SBCs



Whiskey Lake Min-ITX Board Can Drive 3 Displays

The Conga-IC370 from Congatec supports the Whiskey Lake-U with capability of driving up to 3x independent 60Hz UHD displays, each with up to 4096 x 2304 resolution. It has up to 64GB DDR4 support, 2x Gbit Ethernet ports and USB 3.1 Gen.2 host ports and an operating temperature range of 0 to 60°C.

- 8th Gen Intel Core SoC processors with up to 4 cores
- Intel UHD-Graphics 610/620, up to 24 Execution Units
- Up to 64GB dual channel DDR4 2400MT/s
- Flexible internal and external video interfaces
- 2.5Gbit Ethernet with TSN Support
- Wide Range Power Input 12V to 24V
- Congatec embedded Board Controller Features

Congatec
www.congatec.com



Pico-ITX Board Offers DSP Plus Optional AI

Estone Technology's EMB-2237-AI is a Pico-ITX (100mm x 72mm) PoE edge AI embedded board based on NXP i.MX8M Mini Arm application processor. The board features a PoE Ethernet port, on-board dual-core DSP that runs algorithms for voice control, noise suppression, and echo cancellation technology, a full set of I/Os including RS-232/485, and m.2 PCIe slot for Edge TPU AI based solutions.

- NXP i.MX8M Mini with up to four 1.8GHz Cortex-A53 processors
- One Cortex-M4 for real time requirements
- Fast Ethernet with build-in PoE
- Smart codec with dual-core DSP for digital MICs and voice control
- MIPI DSI, LVDS, RGB, I²C connectors for LCD and touch panel support
- m.2 PCIe slot supports Google's Edge TPU for high-performance ML
- Rich I/O with RS-232/485, I²C, GPIOs, USB 2.0 ports

Estone Technology
www.estonetech.com



Pico-ITX SBC Does Depth Sensing and Deep Learning

The 6560 in a Pico-ITX SBC from InForce Computing (now named SMART Wireless Computing) taps Qualcomm's octa-core Snapdragon 660 SoC for applications including stereoscopic depth sensing and deep learning. It supports stereoscopic depth sensing with the help of dual MIPI-CSI interfaces.

- Qualcomm Snapdragon 660 (SDA660 SoC) processor
- 3GB on-board LPDDR4 RAM
- 32GB eMMC ROM
- 1x MicroSD card v3.0 interface
- USB-C on USB 3.1/gen1 + USB-HS
- UltraHD (4K) display on USB-C
- H.265 (HEVC)/H.264 (AVC)/VP9 playback & capture at 4K30
- Dual MIPI-CSI cameras up to 16MP
- Dimensions: 100mm x 72mm
- Operating Temp: 0 to +70°C (Commercial)

Inforce Computing
www.inforcecomputing.com

DATASHEET URLS:

Congatec www.congatec.com/fileadmin/user_upload/Documents/Datasheets/conga-IC370.pdf

Estone Technology www.estonetech.com/wp-content/uploads/2020/01/EMB-2237-AI_DataSheet.pdf

Inforce Computing www.inforcecomputing.com/public_docs/Inforce_6560_Datasheet_003300_Rev%20A.pdf



Compact Pico-ITX Board Features Low Power Consumption

Kontron's embedded motherboard pITX-APL features a small 2.5" form factor and the latest generation Intel ATOM processor (formerly codenamed Apollo Lake). It offers improved graphics and computing performance and at the low power consumption of only 6W to 12W at 12VDC input voltage.

- High performance CPU, graphics, and media performance supporting up to 3 independent displays
- TPM2.0 and optional Kontron Approtect Security Solution
- mPCIe half size, MicroSD/MicroSIM Card Combo
- SO-DIMM Sockets DDR3L-1866 (up to 8GB)
- LVDS 24Bits dual channel and Display Port 1.2
- Extended temp. range of -40°C to +85°C (non-operating mode)
- -25°C to +75°C (operating mode)

Kontron
www.kontron.com



Skylake-Based Mini-ITX Targets IoT Gateways

WIN Enterprises' WIN MB-65040 Mini-ITX is motherboard for IoT gateways, robotics, industrial control, and casino gaming applications. MB-65040 supports the Intel Skylake-S CPU and Intel H110 chipset. The device features 6 COM ports and other robust I/O, making it an especially good fit in IoT gateways.

- 6th Gen Intel Core Skylake-S processor
- Intel H110 express chipset
- DDR4 / 2133MHz up to 16GB
- HDMI 1.4b, DP++ and 24-bit LVDS
- 2x Intel GbE LAN, 1x Mini-PCIe
- 6x COM, 4x USB3.0, 4x USB 2.0, LPC, SMBus
- 4x SATA w/ RAID, HD Audio
- PCIe X16 and optional PCIe X1 slot
- Optional TPM via LPC pin header
- DC 12V input

WIN Enterprises
www.win-ent.com



Pico-ITX SBC Serves Up NXP i.MX8M

The ITX-P-C444 from Winsystems is an industrial Pico-ITX SBC based upon NXP's i.MX8M application processor and packed with dual Ethernet, industrial I/O and expansion options. The processor supports industry-leading video processing along with M4 microcontroller for real-time subsystems.

- NXP. i.MX8M Industrial Processor at 1.3GHz
- Up to 4GB LPDDR4 RAM
- -40°C to +85°C operating temperature range
- Pico-ITX form factor (102mm x 73mm)
- Wide range power input (9V to 36V DC)
- 2x GbE, 1x USB 3.1 Gen 1, 3x USB 2.0
- 2x RS-232/422/485 serial ports
- 6x GPIO, 1x MIPI-CSI (4-Lanes), 1x SPI bus, 1x I²C
- HD Audio Interface
- HDMI output with 4K UltraHD

Winsystems
www.winsystems.com

DATASHEET URLS:

Kontron www.kontron.com/downloads/datasheets/p/pitx-apl-v2.0-datasheet-rev.1.0.pdf?product=146603

Win Enterprises www.win-ent.com/images/stories/download/datasheets/MB-65040.pdf

Winsystems https://resources.winsystems.com/datasheets/itx-p-c444_ds-1.3.pdf

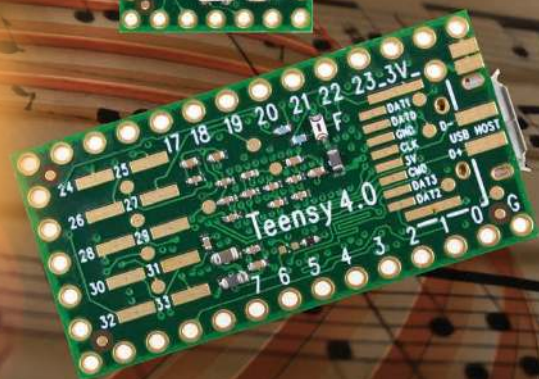
Picking Up Mixed Signals

Build a SoundFont MIDI Synthesizer (Part 1)

Using Teensy 4

In this two-part article series, Brian discusses several ways to emulate musical instruments with an electronic synthesizer, and the technologies they require. He then focuses on the SoundFont standard, setting up the groundwork for Part 2 where he programs and builds a MIDI wavetable synthesizer using Teensy 4.

By
Brian Millier



Back in *Circuit Cellar* issue 328 (November 2017), I described a Hammond tonewheel organ emulator using a Teensy 3.6 module. The Hammond organs used mechanical tonewheels to generate 91 sine waves at the required “musical” frequencies, and the organ mixed these to produce a wealth of different “voices.” It was based on the mathematical principle that you can generate virtually any desired waveform by combining sine waves consisting of the fundamental tone and various proportions of higher harmonic frequencies.

Although these organs produced a rich variety of voices, many sonic subtleties are present in conventional musical instruments that were not present in the Hammond organ’s sound—or any electronic organ, for that matter. The most notable difference is that when you play a conventional musical instrument, each note has an amplitude envelope. That is, its amplitude rises quickly from silence, stays at some relatively constant value while the note is being held and then decays (usually exponentially) back to silence after the musician stops playing that note.

There are actually more “phases” to this amplitude envelope, but you get the idea. Conversely, the Hammond organ’s sound goes from complete silence to some fixed amplitude immediately, and stays constant until the key is released, at which time it returns immediately to silence. I am ignoring the “percussive” voicing on the Hammond organ in this comparison, but it is a limited form of the envelope concept.

To emulate conventional musical instruments with an electronic synthesizer, many different approaches have been taken over the last 70 years. To a large extent, the forms of emulation that were commercially developed depended heavily on the available electronic technology of the time. I won’t delve into the various

approaches taken in the past, but will concentrate on the modern wavetable synthesis method that is in common use today. This method requires fast microcontrollers (MCUs) and lots of cheap RAM memory—both of which are commonplace today.

WAVETABLE SYNTHESIS

The concept here is to forget about using a complex algorithm to generate the required waveform, but instead to “record” a conventional musical instrument to obtain the actual waveform of the instrument’s sound. Alternately, a wavetable synthesizer can use algorithmically-derived waveforms, instead of acoustic musical instrument samples, to generate waveforms that are musically pleasing, but not derived from any actual musical instrument.

These sound “samples” are then stored in some form of non-volatile storage, such as a memory card or other high-capacity ROM memory chip(s). When you play the synthesizer, that waveform is read out at a fixed sample rate. A method known as Direct Digital Synthesis (DDS) is used to achieve all the necessary note frequencies.

The process I just described, while accurate, only describes a small part of the overall wavetable synthesizer functionality that is necessary to produce musically accurate recreations of acoustic instruments. If you did nothing beyond implementing the simple procedure described, the results would be musically boring and quite unrealistic in the emulation of many acoustic musical instruments. There are three reasons why this would be the case.

First, the whole concept of the amplitude envelope, mentioned above, is not captured in the recorded waveform. You could record the note being played for several seconds, thus including the amplitude

envelope. However, this would take up a lot of memory. And how would you handle notes of varying durations?

Second, conventional musical instruments, being physical objects, have acoustical properties, such as resonances, that vary as you go from the lowest note that they can produce up to the highest note. Therefore, you can't assume that a waveform recorded for a low note will bear any resemblance to that of higher notes within the instrument's range. In practice, you must record several sample waveforms spread over the instrument's range, and store each of them in memory.

Third, most musical instruments do not produce a perfectly stable frequency while a note is being played. Often a small frequency modulation, called "vibrato," occurs. Sometimes this vibrato isn't present at the very beginning of the note, but gets introduced slowly as the note is held. This vibrato may or may not be present for each note played, and is part of the musician's style of playing. Therefore, the synthesizer should allow for this possible variation, by responding to "modulation" commands coming from the attached keyboard.

Those three reasons are not a complete list of the parameters that must be taken into consideration to produce a realistic wavetable synthesizer. However, they are the most important ones.

LOOPING

Another required concept that's important in wavetable synthesis is "looping." The idea here is that the electronic waveform sample that has been recorded contains two main sections. They are: 1) the initial "attack" section, where the sound goes from silence to a reasonably stable frequency and maximum amplitude; and 2) the "loop" section, where the sound is reasonably stable in terms of frequency and amplitude.

Therefore, when you are "playing" a sample, you first read out the initial attack part of the sample. Then you continuously repeat the loop section of the waveform, for as long as the note is being held. Once the key is released, you can continue reproducing that loop section of the waveform, but you use another part of the wavetable routine that is handling the amplitude envelope to exponentially reduce the amplitude of the note down to silence.

You might question why you even need the initial attack portion of the waveform, when you have a separate section of the program handling the amplitude envelope. Conventional musical instruments often have quite complex attack waveforms, in terms of both amplitude and frequency/harmonic content. Therefore,

is would not be sufficiently accurate to depend solely on the amplitude envelope function in the program to emulate the attack section of the sound. The amplitude envelope is handled by a function called ADSR (from the Attack, Decay, Sustain and Release phases). An ADSR envelope is shown in **Figure 1**. Note that in acoustic musical instruments, the attack, decay and release phases are generally exponential in nature.

SoundFont is a file format and associated technology that uses sample-based synthesis to play MIDI files. In short, SoundFont files contain recorded audio samples of musical instruments. Looking at SoundFont files, the decay time is often specified as zero, because this isn't a big consideration in the overall replication of the note. (The Decay section of Figure 1 is somewhat exaggerated for illustration's sake.) Also, due to the relatively short amount of computing time that is available with a sample rate of 44,100Hz, the exponential curves found in the attack/decay/release phases are often simulated with a simpler linear ramp, instead of an exponential curve.

When wavetable synthesis was first developed commercially, the synthesizer companies basically generated their own sample waveforms, stored them in ROM memory devices in the instrument and developed proprietary methods to handle the envelope requirement and various other modulation requirements. These synthesizers were called "samplers." There were ways to extend the available voices on such instruments using plug-in memory cartridges, but nothing was standardized among the various synthesizer companies. In time, this led to the development of the SoundFont standard, which, in its own way, advanced the state of the art in wavetable synthesizers in much the same way that the earlier MIDI standard advanced electronic musical instruments in general.

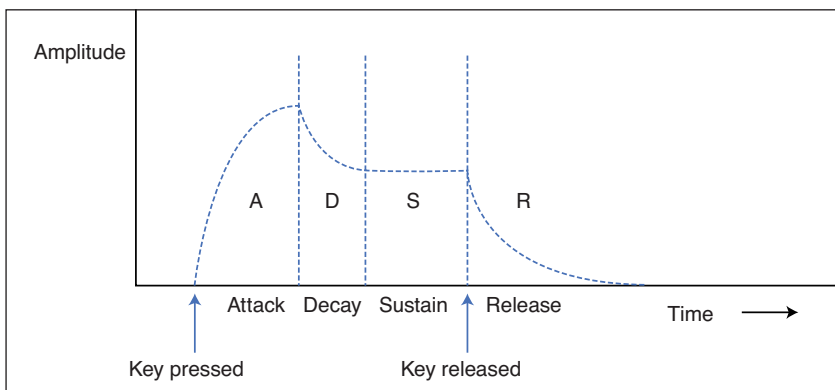


FIGURE 1

Any musical instrument's sound has an amplitude envelope comprising four basic sections: Attack, Decay, Sustain and Release. The richer the instrument's sound, the more complex this envelope will be.

FIGURE 2

Back in the '80s, a good-quality sound card for the IBM PC computers looked like this. Creative Labs, which produced this sound card, was the originator of the SoundFont standard.



THE SOUNDFONT STANDARD

You could write a whole book about this standard. On the *Circuit Cellar* article materials webpage, you can find all the references for this article—including a link [1] to just such a book that describes in detail the SoundFont specifications. The initial version 1.0 of the SoundFont standard was introduced by Creative Labs for its Sound Blaster AWE32 product. This wasn't a conventional synthesizer, but rather a sound board designed to be mounted in a PC computer—plugged into the internal ISA bus. **Figure 2** shows the AWE32 sound board for the PC computer. The SoundFont format is now also used by some stand-alone synthesizers, and in instrument plug-ins used by DAW software on PCs and Macs.

The SoundFont file structure is, by necessity, somewhat non-rigid. Depending on the acoustic “richness” of the instrument being emulated, the size of the waveform sample(s) required can vary dramatically. As mentioned earlier, to replicate the sound of any acoustic instrument accurately, you must provide different samples over the range of notes that the instrument can produce. A piano, for example, might require 20 or more individual samples to cover the its 88 keys—one for every group of three or four keys. Some deluxe piano wavetable voices use one waveform sample per note!

These individual samples are assigned to “regions,” and in addition to the region's basic waveform, you also have other required data—such as the ADSR envelope, vibrato parameters and numerous other parameters that affect the nuances of the instrument's voice. I refer to these other parameters in this article as “metadata,” although that isn't the term that is used in the SoundFont reference. **Listing 1** shows the data structure used in my program to define what I call the metadata. Note that there is one such structure for every region of the voice.

The SoundFont file structure was adapted to fit into the pre-existing Microsoft RIFF-wave file structure. This structure uses various

“chunks” and “sub-chunks” to store data. In the case of SoundFonts, the various sample waveforms required for each keyboard region are stored in one type of chunk, and that region's metadata (ADSR envelope settings, vibrato, sample rate, loop length and so on) are stored in other chunks. Another chunk stores identification data, such as the voice name and who engineered the voice samples. I provide a link to the RIFF file format [2], although that is very generic. The SoundFont reference manual describes the way in which the RIFF-file format is used for that purpose in greater detail.

If the SoundFont file structure had been defined differently—for example, as one in which there were a certain number of well-defined data fields with known lengths and readily-discernable terminators/separators—I might have been tempted to write a routine in the C language to parse that file into a form that this project could use. This could have been handled by the Teensy 4, itself, since it is certainly powerful enough, and there is plenty of program memory space available to easily handle a complex routine such as this.

However, seeing how complex these SoundFonts were in the RIFF format, I decided that it was more practical to use the work that had already been done by the group of students who had written the Teensy Wavetable library object itself.

CONVERTING A FILE USING PYTHON

I touched upon the complexity of the SoundFont file format in the last section. The people who wrote the Wavetable object for the Teensy Audio library decided to write a Python program that would allow you to:

- 1) Browse your filesystem for a desired SoundFont file.
- 2) Observe and choose which of the keyboard regions you wanted to import into the wavetable object library.
- 3) Format the SoundFont waveforms and

the metadata into several blocks of information, which are then stored as a “.cpp” file. A small amount of remaining data is stored in an associated “.h” file.

- 4) Choose a primary filename for the “.cpp” and “.h” files.

The developers call this program “decoder.py.” I commend them on using Python for this task. When it comes to handling/parsing out complex file structures, Python is an excellent choice. If you look at the original decoder.py program, you can see that it’s doing a whole lot of parsing and conversions in a relatively small program—a tribute to the efficiency of the Python language.

Python is available for free for Windows computers, Mac OS X and Linux. Personally, I had only used Python sporadically several years ago, as part of some work I did with the Raspberry Pi (running Linux). I had never used it in Windows. I was optimistic that I could remember enough Python to convert this program to something that would work the way I needed for this project.

I have been using Microsoft’s Visual Studio for 5 years, and am currently using the latest version—Visual Studio 2019. In the past, I had only used Visual Studio with the Visual Micro plug-in, which allows me to develop programs on any MCU platform that the Arduino IDE supports. For me, that includes AVR, Arm (Teensy 3.x and 4) and the Espressif ESP8266 and ESP32. However, Visual Studio also supports Python development. I was able to add this functionality by running the Visual Studio Installer (from the Windows Start Menu) and adding Python support. This puts the Python executable program into the folder:

Program Files (x86) Microsoft Visual Studio\Shared\Python37_64

You will have to make a PATH to this location, so that you can run it from other folders (where your Python programs will be located). If you don’t know how to make a windows PATH, I describe how to do this in the “Python Resources help file” that is included with the rest of the source files for this project.

FOCUS ON FILES

Once you have Python installed, you can get the SoundFont decoder.py program (and the associated controller.py program, which provides a Windows GUI which runs it) from the GitHub source [3]. There are some dependencies that need to be installed, and this is also described on that GitHub site and summarized in my “Python Resources help file.” While you will need the other files from

this GitHub repository, you must use the decoder.py file that I provide with the rest of the source code, because I modified it to work for this project. This is available on the *Circuit Cellar* code and files webpage.

Why can’t you just use the original decoder.py program written by the developers of the AudioSynthWavetable library? They designed the decoder.py program to decode the SoundFont file data in such a way that it could be “#included” into a Teensy C language program— as “.cpp” and “.h” files. Basically, this makes the chosen SoundFont voice an integral part of the synthesizer program—it is stored in flash memory. Their program only produces the one voice that you have selected and embedded in the Teensy program. I wanted to be able to choose various voices at will, loaded from an SD card.

The format of the .cpp and .h files from the original decoder.py program was dictated by how the compiler could handle the various data structures. In fact, many of my metadata parameters were defined in such a way that certain Audio library constants were imbedded in the parameter, itself. For example:

```
uint32_t(431*SAMPLES_PER_
MSEC/8.0+0.5) , //RELEASE_COUNT
```

```
struct sample_data {
    int16_t* sample;
    bool LOOP;
    int INDEX_BITS;
    float PER_HERTZ_PHASE_INCREMENT;
    uint32_t MAX_PHASE;
    uint32_t LOOP_PHASE_END;
    uint32_t LOOP_PHASE_LENGTH;
    uint16_t INITIAL_ATTENUATION_SCALAR;
    // VOLUME ENVELOPE VALUES
    uint32_t DELAY_COUNT;
    uint32_t ATTACK_COUNT;
    uint32_t HOLD_COUNT;
    uint32_t DECAY_COUNT;
    uint32_t RELEASE_COUNT;
    int32_t SUSTAIN_MULT;
    // VIBRATO VALUES
    uint32_t VIBRATO_DELAY;
    uint32_t VIBRATO_INCREMENT;
    float VIBRATO_PITCH_COEFFICIENT_INITIAL;
    float VIBRATO_PITCH_COEFFICIENT_SECOND;
    // MODULATION VALUES
    uint32_t MODULATION_DELAY;
    uint32_t MODULATION_INCREMENT;
    float MODULATION_PITCH_COEFFICIENT_INITIAL;
    float MODULATION_PITCH_COEFFICIENT_SECOND;
    int32_t MODULATION_AMPLITUDE_INITIAL_GAIN;
    int32_t MODULATION_AMPLITUDE_SECOND_GAIN;
};
```

LISTING 1

The part of the SoundFont file structure that contains what I call the “metadata” for the voice, in addition to the actual wavetables that define the voice.

In equation form, the release count parameter is defined as:

$$\text{release count} = 431 \times \frac{\text{samples per millisecond}}{8.0} + 0.5 \quad (1)$$

Expressing it in this way is fine when the expression evaluator used in the pre-compile phase will translate this expression into a number. However, I did not want the SoundFont data to arrive in my program in a format that would require a lot of expression evaluation. Instead, I modified

the original decoder.py program to format its parameters as actual numbers, with no system-dependent constants such as `SAMPLES_PER_MILLISECOND` embedded in the parameter.

I made other modifications to simplify the processing of the SoundFont data by the Teensy 4 program, and to eliminate some other lines of code that was originally needed to allow the ".cpp" and ".h" files to be directly included in the Teensy program code.

Figure 3 shows a screen capture of the controller.py program in action. What you see here is the controller.py program running. This is a Windows GUI skin over the decoder.py program (a Command Line Python program). In the upper left portion, you can see a black rectangle, the Python interpreter (py.exe) running from the command line and interpreting the controller.py program. When the controller.py program has all the required user input, it passes that information to the decoder.py program to do the actual decoding. Running an interpreted language like Python is different from what most of us are used to with compiled (.exe) applications!

In Figure 3, I've loaded a SoundFont file containing a whole set of General MIDI programs or voices, and selected the Baritone Sax voice. All that's left to do is choose how many of the samples (regions) you're going to select. This will depend upon how many keyboard regions will fit into the Teensy 4's 480,000-byte sample waveform memory. The size of the regions that you have selected is reflected in the "Sample Stats" window. Finally, you must pick a filename and select a folder in which to store the files.

For this project, all voices must be named in sequence from 1.cpp to 127.cpp—plus the like-named .h files. You can save these files to the PC hard disk, itself, or insert an SD card adapter into your PC and save it to SD card directly. One way or the other, you need all the sound files on the SD card. Now that the SoundFont files have been converted into a form that this project will accept, let's look at the Wavetable Synth object in the Teensy Audio library.

WAVETABLE SYNTH OBJECT

The Teensy audio library object `AudioSynthWavetable` was written by some Portland University graduate students. It is documented on the GitHub site [3]. **Figure 4** shows an instance of this object placed on the Teensy Audio System Design Tool workspace. I've provided a link to this on-line tool [4]. In Figure 4, the Help screen for it is on the right. The seven available functions are shown. Basically, you set the instrument up by passing a string describing it to the

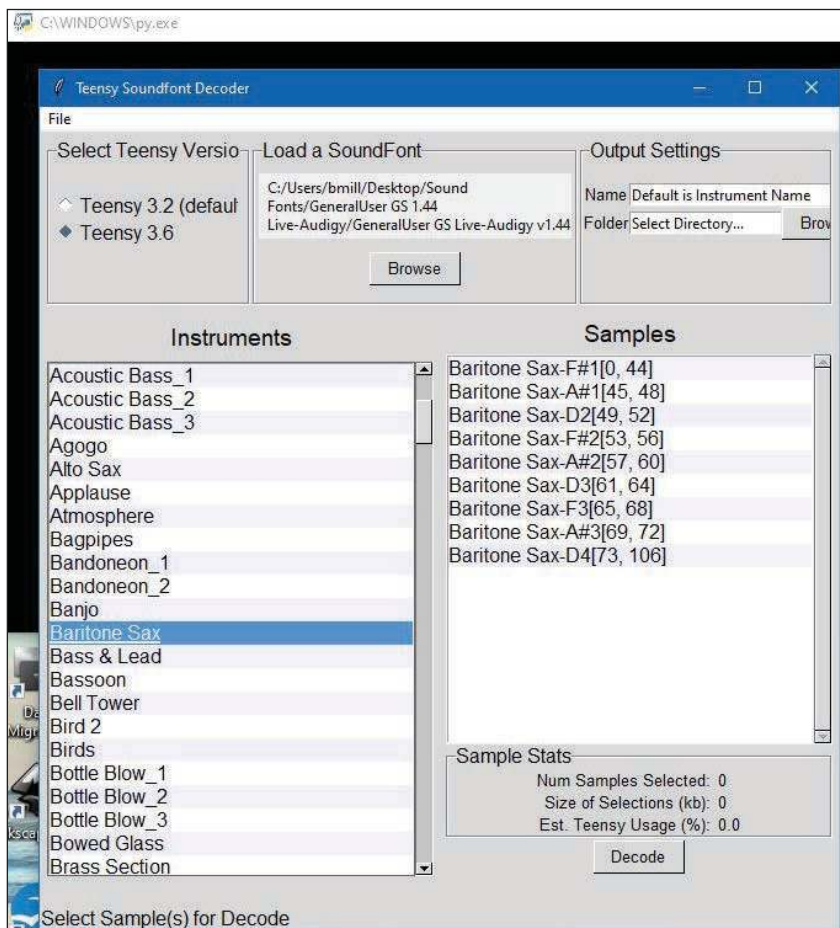


FIGURE 3

The GUI for the Python program that converts standard SoundFont files into a format that can be used by the C code running on the Teensy MCU.

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

Microsoft | www.microsoft.com

PJRC | www.pjrc.com

`setInstrument()` function. That string was defined in the SoundFont's ".h" file that has been "#included" in the program. (after having run the decoder program described in the last section) You use the `playFrequency()` function to start a note playing at a defined frequency and amplitude. You use the `stop()` function to end a note, and the `isPlaying()` function to know when the final release phase of the note is over.

The Wavetable synth object is monotonic—it can only play one note at a time. You must define many instances of this object to handle many notes at a time. Considering that a note may linger for up to a few seconds during its release phase, I chose to include 48 separate Wavetable objects in my program, to handle many notes being played simultaneously. Note that you would generally use the Audio System Design Tool to arrange/wire up your various audio objects, and let it produce the C code that integrates these objects into your program. However, in this case, there are 48 discrete AudioSynthWavetable objects needed, along with 17 mixers to combine all the Wavetable objects. In this case, it was easier just to use the code that was written by the original developers in their sample program (lines 15-67 in my program).

I must commend the students who wrote this library object. Generating sounds based upon the SoundFile template requires the following considerations:

- 1) Scan through the waveform table (from the region corresponding to the given note) using the DDS method, to provide the basic waveform.
- 2) Perform the above scan using the initial section of the waveform table for the attack phase of the note.
- 3) Scan through the loop portion of the waveform repeatedly, during the time that the key is being depressed on the MIDI keyboard controller.
- 4) Control the amplitude of the signal throughout the loop duration, using the decay time and then the sustain time envelope parameters.
- 5) When the `stop()` function is called, continue scanning the waveform, but exponentially decay its amplitude to silence, according to the release time parameter.
- 6) Keep track of when the sound has decayed to zero amplitude, for use by the `isPlaying()` function.
- 7) Apply modulation, such as vibrato, the depth of which will often start at zero and increase while the note is still sounding.

Those are the main functions. There might be some minor ones implemented that I am unaware of. While the original AudioSynthWavetable object was very nicely implemented, in my opinion, it had two significant shortcomings.


First, you must load a specific SoundFont voice into flash memory—so it's a part of the Teensy program. Therefore, the program can reproduce just that one particular voice. It's possible to "#include" more than one set of .cpp and .h files into the program. This could give you a several voices, but in practical terms, there is not enough memory space to handle more than one reasonably rich voice, even with the Teensy 4.

Second, although the original library object can handle vibrato (and in some SoundFonts a delayed vibrato), it cannot produce vibrato that is triggered by the musician moving the Modulation controller (the "mod wheel"), or from Channel

Aftertouch messages. This "expressiveness" is commonly used by keyboard musicians.

When the Teensy 4 became available, it greatly increased the available amounts of both Flash and SRAM memory, compared to earlier Teensy modules. Given that the Teensy 4 had 1MB of SRAM, I felt it would be possible to convert the AudioSynthWavetable object to allow the following activities:

- 1) Move the waveform/metadata memory from program memory, where the original object placed it into SRAM. Note: This was a lot more time consuming than I had anticipated, mainly because I was using someone else's code instead of writing my own.
- 2) Use an SD card to store up to 127 separate voices, which could be selected and loaded into SRAM on demand.
- 3) Allow the "mod wheel" to modulate the vibrato amount in real time. Also allow MIDI Channel Aftertouch messages to do the same thing.

In Part 2 of this article (*Circuit Cellar* 360, July 2020), I'll describe the programming I wrote to accomplish this, and show the circuitry that I built to implement the MIDI wavetable synthesizer. 

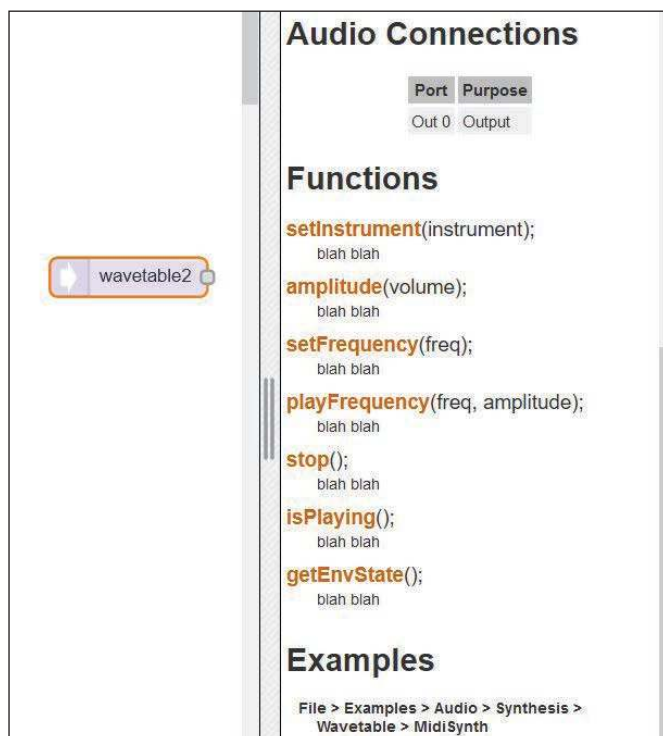


FIGURE 4

The Teensy Audio System Design Tool is used to interconnect the various sound modules that have been written for the Audio library. This shows the Wavetable object in the Design Tool. I made extensive changes to it, to make it do what I expected from the 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.

Embedded System Essentials

Broad Market Secure MCUs

Spotlight on the MAX32520

By
Colin O'Flynn



The new MAX32520 MCU from Maxim provides a number of security features that are normally found only in devices without public information available. Here, Colin explores a few of those features, and performs some hands-on testing to see how they hold up to the types of attacks they claim to defeat.

Previously in this column, I've covered a few new interesting security products. This article will be another along those lines. It follows my experience with a new microcontroller (MCU) from Maxim Integrated Products. This product has only recently become available, and seems to come with a rather loaded press releases extolling how it will create the secure device of your dreams. But, behind all that is a pretty interesting product that is worth finding out more about—with the usual caveats that things might not be as rosy as the marketing department wants you to believe.

SECURITY MARKETS

To understand what makes this device interesting, it's also useful to understand how secure products were typically distributed in the past. There have been many secure MCUs in existence before this device, and there will continue to be them afterwards.

These secure devices had datasheets available only under non-disclosure agreement (NDA). Part of the reason for this is the method used to assign security levels to devices gives a certain amount of points to having information not publicly accessible. The idea being the less an attacker knows about the target, the more difficult it will be for the attacker to exploit it.

Getting access to the parts under NDA required a certain amount of interest from the MCU manufacture in your application. This may mean minimum volumes in the millions or more units before access to the NDA parts was even considered.

This also meant that many companies had secure devices and peripherals that were very well tested. So, it's clear many of these MCU manufacturers have the expertise and know-how to produce a secure device. But the design teams producing non-NDA devices available in the open market—or "broad market"—did not get to re-use those secure MCU blocks or peripherals.

This has left a soft spot in embedded security: Until relatively recently, the security of open-market devices you can simply buy from distributors has been rather poor. This has been shifting in recent years with the slow introduction of power-analysis resistance and other types of countermeasures. If you are developing a product and not building millions of units, you should be interested in what the most security you can get from non-NDA parts.

The Maxim MAX32520 appears to be an interesting experiment by Maxim to step further in that direction. The device includes many features that are typically found in their secure (under NDA) parts such as active tamper detection, internal shields, and fault injection monitors. There is limited talk of DPA resistance, but it's something we can explore a little bit with their development board.

Let's take a look at the parts and then see how we can evaluate a few of the claims! I need to warn you: At the time of writing this article, the SDK and documentation had only just become available. A full reference manual is not available, for example. Hopefully, more details will hopefully become available so you can use all the features of the device.

MAX32520 SUMMARY

On the security front, there are a few interesting features. The first is what they call "ChipDNA"—which is Maxim's take on a Physically Unclonable Function (PUF). The idea of a PUF is that you have some unique "key" generated as a physical artifact of the device. The PUF can be designed such that the characteristics of the circuit will become unstable if certain invasive attacks are performed on the device. This can be used to record what are valid devices at manufacture time, or form part of a secure cryptographic attestation protocol. This is a good step to prevent against someone who clones your product

from having access to online resources such as firmware updates. A remote server can ensure only known valid devices are given access to such services.

Another use of the PUF is to encrypt firmware on the device. If an attacker is able to read the firmware out of the MCU, it will be encrypted with a key that is unique per device. This means the internal flash can be encrypted/decrypted on-the-fly with AES-256 using the PUF key. In fact, the datasheet claims this gives you the “ultimate resistance against reverse-engineered based attacks.”

This sounds great, but the only guaranteed protection is against an attacker who is reading flash memory out by physically opening the case of the device and reading the flash memory out. This is shown in **Figure 1** at location A. In reality, probing the die is a fairly complicated attack vector. But if an attacker finds a vulnerability that allows them to read from the flash memory-space the MCU will already be loaded and using that magic PUF key. (An example of such a fault attack reading unintended memory is described in my article “Attacking USB Gear with EMFI” in *Circuit Cellar* 346, May 2019.) This means the decryption block will happily pass the decrypted data to the MCU, who is then passing it to the attacker, as in location B in Figure 1.

THERE'S NO MAGIC

The ChipDNA PUF feature alone does not magically protect against reverse engineering attacks. While it's a useful additional defense against certain attacks, don't think it will somehow stop an attacker from easily stealing your firmware if you accidentally leave a door open! Carefully enabling the decryption only when needed and ensuring your clear the decryption key when under attack can be helpful in making the most of the feature.

ChipDNA does provide a useful method of ensuring only valid devices having access to online or value-add services, such that cloned devices are not competitive in the market. This requires more effort on your back-end services—assuming your device connects to them—to lock out counterfeit devices.

Another feature that is highlighted in the datasheet is a serial flash emulation. Many MCUs will boot from SPI flash, and the MAX32520 can emulate those SPI flashes. This feature is interesting as it allows you to retrofit an existing unsecure MCU with a secure boot platform. So, if you already have a design booting from SPI flash, the MAX32520 can replace that SPI flash, and now the MAX32520 can first verify that a given firmware image should be booted.

Of course, this feature still means you'll likely be sending unencrypted firmware to your “dumb” host MCU that doesn't support

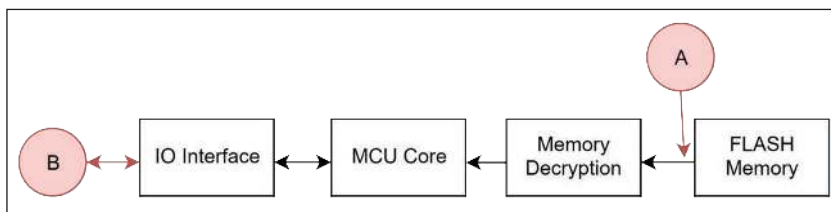


FIGURE 1

The internal encryption key is useful to prevent FLASH read-out attacks from (A), but doesn't stop those attacks occurring after the decryption is already enabled at (B).

secure boot. So, keep in mind this feature doesn't immediately protect someone from copying your design! Can they simply write the encrypted firmware into an old dumb SPI flash in their cloned product?

A much smarter design is to implement some functionality in the MAX32520, such that your system would only function if both the “dumb” MCU alongside the MAX32520 are copied. It should be much more difficult to copy the MAX32520 compared to observing the unencrypted SPI flash data transfers.

The MAX32520 uses entirely internal oscillators to avoid allowing an attacker the ability to control an external oscillator. This makes it impossible to perform attacks such as clock glitching, and may make power analysis attacks more difficult, since the device could internally be adjusting its clock frequency. We'll investigate that a little bit later. The MAX32520 also supports several tamper sensors. Let's take a look at them and see how well they work in practice.

TAMPER SENSORS

External Tamper: The external tamper sensor is fairly straightforward. A random

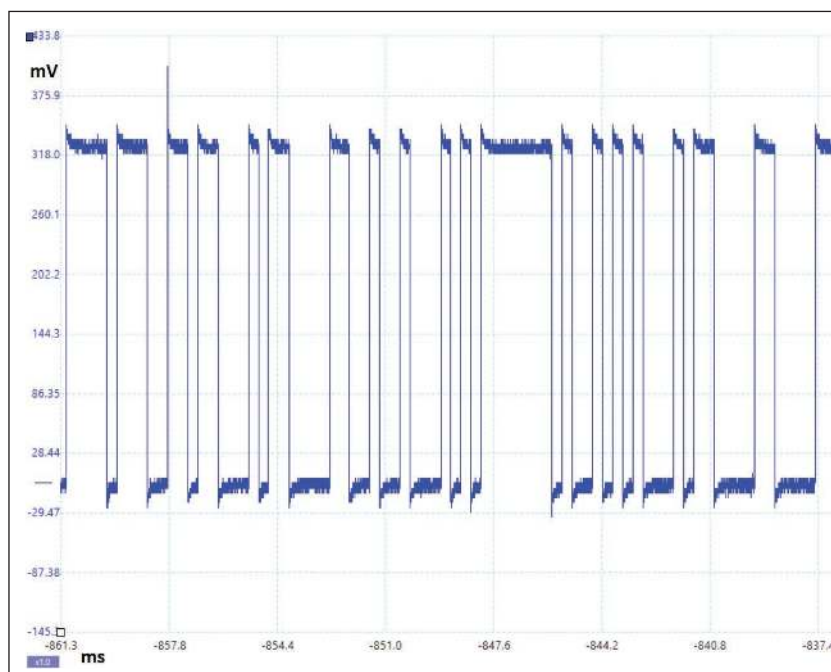


FIGURE 2

The pattern on the external tamper signal is a random signal that must be detected being fed back into the MCU.

```

volatile unsigned int i, j, t, k;
loop_cnt = 0;
while(1){
    t = 0;
    loop_cnt++;
    for(i = 0; i < 1000; i++){
        for(j = 0; j < 1000; j++){
            t++;
        }
    }
    flags = MXC_SMON_GetFlags();
    printf("%d %4d %x\n", t, loop_cnt, flags);
}

```

LISTING 1

This simple double-loop uses volatile variables to make faults in instruction flow and memory access cause the final value of "t" to be incorrect.

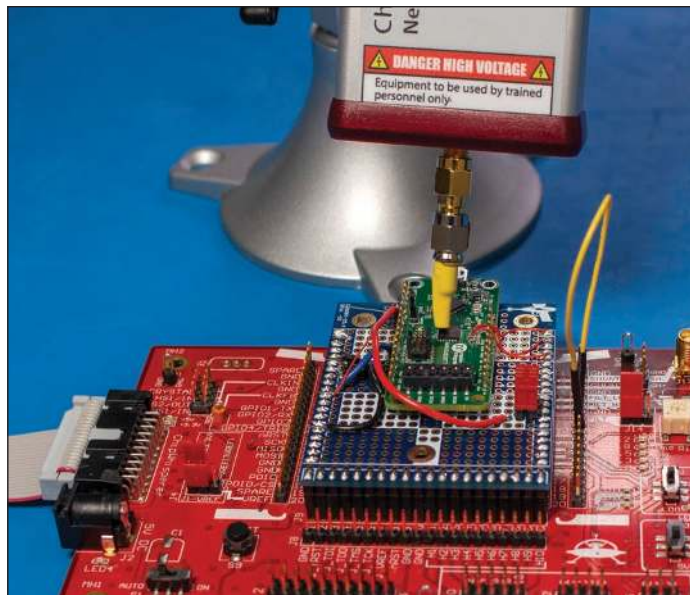


FIGURE 3 Electromagnetic Fault Injection (EMFI) can be used to test features such as the fault detector.

```

1000000    1 0
1000000    2 0
993687     3 0
960420     4 0

```

LISTING 2

If the loops of Listing 1 execute correctly, "t" will have been incremented 1,000,000 times. Note: Two outputs with incorrect values showing faults were injected into the flow.

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

RESOURCES

Maxim Integrated | www.maximintegrated.com

NewAE Technology | www.newae.com

pattern is sent out on one pin, and must be observed on the second pin in order for the device to function. I captured a portion of that waveform in **Figure 2**, you can see it looks like a random binary pattern at about 3kHz fundamental rate with the default SDK example. You can change the frequency via the register setting.

I didn't evaluate the random pattern—let's assume that is OK! The main issue with the external tamper is going to be limited to your use case. This sensor is only active when the device is powered. If an attacker can fully remove the power supply, open the case and short the two tamper pins to each other—your device will happily boot. If you plan on using the external tamper wire, ensure you have sufficient battery power to keep the MCU running with the tamper system. Of course, as a low-power MCU this means running off a small battery for a long period is reasonable. But that does mean you need to perform a careful low-power design for the external tamper feature to be useful.

Internal Tamper: The device also claims multiple internal tamper sensors. Some of them are pretty straightforward, such as checking the device is running inside valid temperature ranges and voltages. It also adds more advanced features such as an active die shield, as well as a various fault detectors. The provided API lists only the "digital fault detector," but the security monitor (SMON) header files and SVD file (SVD provides register bits for debuggers) tease a lot more interesting information.

The "Digital Fault Detector (DFD)" has an API call in the provided peripheral library with the SDK, and it becomes enabled with the `MXC_F_SMON_INTSCN_DFD_EN` bit. In addition, there is a low and high voltage detector for the V_{DD} core, and a voltage glitch detector.

First, I'll use the provided SDK to enable the DFD. I've implemented a simple double-loop that will help me see if any faults have affected the program flow, as in **Listing 1**. I previously discussed this type of fault attack look in the column. I've also printed the status of the security monitor register to see if any changes occurred.

To start with, I used electromagnetic fault injection (EMFI) using the setup of **Figure 3** to see if I could affect the loop. This has the development board mounted onto my UFO target board. See the ChipWhisperer project (www.newae.com/chipwhisperer) for more details of that board. The example output in **Listing 2** shows that a few loops appeared to have invalid counts without the security monitor tripping. Unfortunately, I can't give you a more definitive result. That's because there are few details on how this is implemented, and the documentation shows only functions to enable the DFD feature and read a status. While I experimented with all settings—as well as some hidden flags such as a "voltage glitch enable"—it never fully stopped the glitches from working.

The potential for an effective glitch detector is something worth keeping the MAX32520 in mind for. But be sure to help validate how it works in practice in your product. This is important not only to test the effectiveness of the MAX32520, but also to confirm you have configured it correctly. As you can see from my experiments, following the bare minimum configuration may not be enough.

CRYPTO ACCELERATORS

What would a secure device be without a series of crypto accelerators? The MAX32520 is no different of course! It lists features such as AES, SHA, DES, ECC and RSA accelerators.

You should note that the RSA “accelerator” is actually a software-based (presumably ROM code) accelerator, but, as of yet, there is no documentation for using the RSA accelerator.

With that in mind, let’s see how we can measure the power consumption of their AES implementation. This should help us see if complicated counter-measures are implemented in the device, and if we can see an interesting power signature that is worth exploring in more detail.


A close-up of their development board mounting on my CW308 UFO Board is shown in **Figure 4**. To help reduce noise, I’m feeding in 1.2V to the V_{Core} supply pin, and have removed the “regulator output” capacitors. Because I’m feeding in slightly higher than the specified core voltage, the internal regulator should turn off resulting in a clean power trace.

You can see the power traces of an AES encryption in **Figure 5**. Unlike many classic MCUs, there is no strong or obvious signature of the accelerator block. This is a good sign that the accelerator may have a harder power signature to detect, but more effort would be needed to see if it’s possible to detect the operation of it, and also to break the accelerator itself. We can hope this device isn’t vulnerable to normal attacks such as a classic differential power analysis attack.

TOWARDS BROAD MARKET MCUs

The MAX32520 appears to be a shift toward more open secure MCUs. As you can see from this article, there are still many features with too little documentation. This hampers the ability to perform critical external security evaluations.

In addition, devices like this will always be vulnerable to the classic problem of the user misconfiguring it. For example, while the device has internal flash memory encryption, a software (logic) bug that issues read requests to the flash memory will be automatically decrypted by the security system. To correctly use these devices, you need to Understand what the security features are buying you.

All that said, the inclusion of many more advanced features, without requiring an NDA to see some details of them, is a strong step in the direction we need to help design secure products. I know many people (myself included) are not designing products in industries that involve the volumes needed to get classic secure MCUs under NDA. So, keep an eye out for more details of the MAX32520 as they become available. You might find that you can finally get some real security features in a broad market MCU. And we can hope that other vendors will be in hot pursuit of this more open approach to secure devices, giving us the ability to evaluate and choose between multiple such devices. 

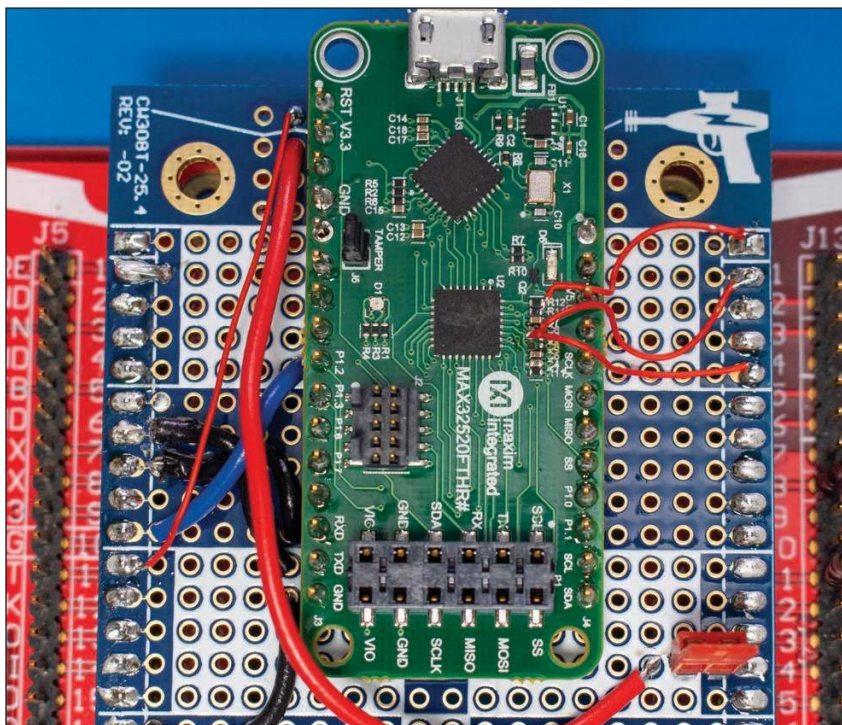


FIGURE 4
A close-up of the development board mounted onto a ChipWhisperer CW308 UFO target board

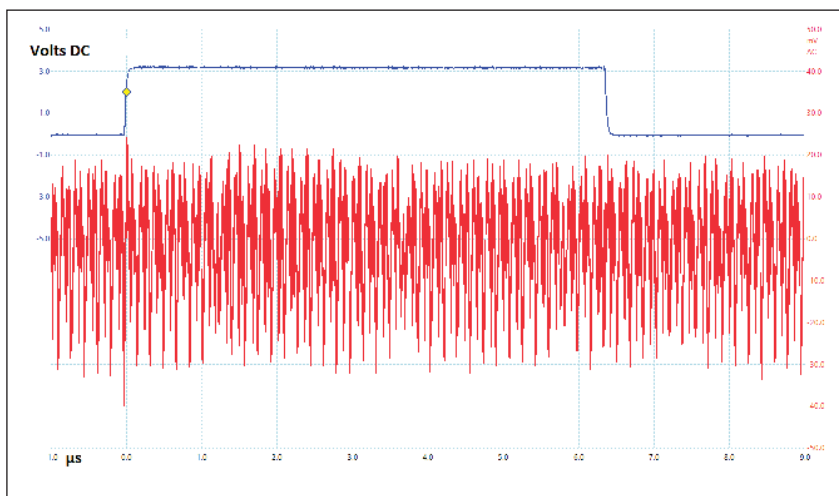


FIGURE 5
The trigger showing when the AES block should be active is on the upper trace, and the lower trace shows the power trace.



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.

From the Bench

Upgrading the Weather Tree With I²C Interfacing

Argent Data Systems makes a pole-mounted sensor unit for collecting wind speed, wind direction and rainfall. It's a great weather measurement system with which you can do whatever fancy interfacing you choose. In this project article, Jeff creates an I²C slave that does all the sensor work and presents the results in table form that can be accessed with a simple I²C connection.



FIGURE 1

The sensor tree contains tipping rain gauge, anemometer and wind direction sensors. Sensors connect to your circuitry via two 4-conductor phone cable connectors.

By
Jeff Bachiochi

Anyone who delves into a weather project has most likely run across what I like to call the “weather tree”—a pole-mounted sensor system for collecting wind speed, wind direction and rainfall. One unit is available from Argent Data Systems for about \$70 (**Figure 1**). Nothing else on the market gives an experimenter these kinds of weather tools at such a reasonable price. The interface is simple for each sensor. All the sensors use reed switches to detect their specific properties.

The wind speed and rain gauge both employ a single reed switch. Reed switches require a magnetic field to change state. These particular switches are open until a magnet comes within range. With a magnet attached to a moving mechanism, the reed switch closes each time the magnet and switch come into close proximity. No actual touching is necessary. The reed switch, itself, is contained within a glass package and can be easily damaged, so some care is necessary. Only two wires are required for each sensor. Connecting each sensor pair between an input pin with a pull-up resistor and ground, the pin will be at logic high, unless the magnet is closing the reed switch, in which case the input is grounded.

The wind direction sensor uses eight reed switches. These are mounted at equal distances around the direction vane's axis of rotation. The vane enables a switch depending on its position. To differentiate one switch from another, resistors of different values are placed in series with each switch. Although this sensor also uses two wires, we are unable to count contact closures, as with the wind speed and rain gauges. Instead, we need to measure resistor values. We can simply connect this sensor as we did the others. Digitally,

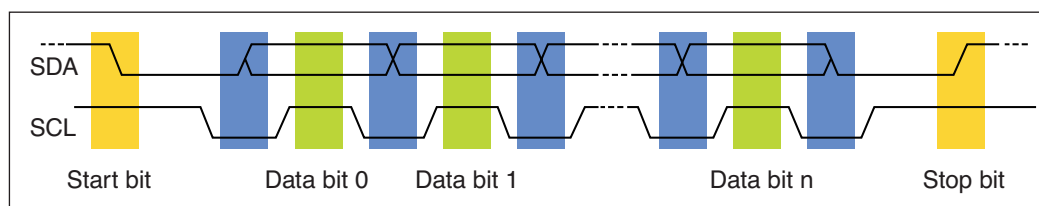
we would see the input being pulled up, if the vane's magnet were not closing a switch.

When closed, depending on the value resistor in series with that switch, the input voltage may or may not be pulled below the threshold for the input to see a logic low. We must use an analog-to-digital (ADC) to detect direction with this sensor. In fact, the design is such that one reed switch will always be closed. It is possible for two switches to be closed while the vane is between reed switches! All resistors in the sensor are 1%, so their actual values are dependable enough that with the values selected, each of the 16 points (N, NNE, NE, ENE, E, ESE, SE SSE, S, SSW, SW, WSW, W, WNW, NW and NNW) can be determined from the divider's voltage.

What could be simpler than counting pulses and measuring voltages? Not much really. So why fiddle with simplicity? Why climb a mountain? Cross an ocean in a row boat? Because you can. It's not the feat, it's the journey! In this journey, I want to create an I²C slave that will do all the sensor work and present the results in table form that can be accessed with a simple I²C connection. If you've used I²C in the past, you know how easy it is to use. If not, then you ought to add this as another tool in your kit of experience.

THE I²C BUS

The interface is simple—four wires, power, ground, clock and data. Both signal lines are open collector with external pull-ups. Although I²C was originally designed for use among devices on the same PCB board, the bus is applicable for wiring together devices at length from one another. For instance, I have a system that polls multiple I²C devices using twisted cables of over 120', using 2.2kΩ resistors on both clock and data lines.

**FIGURE 2**

I²C communication requires two signals: SCL (clock) and SDA (data). Except for the start and stop bits (yellow), you can be assured that the data won't change while the clock line is high (green). After the start bit, an acknowledge ninth bit is added for every 8 data bits. This is feedback that someone is listening to the communication.

The open collector nature of the bus connections allows multiple Master devices to take hold of the bus and request data from multiple Slaves. Bus arbitration is required when using multiple Masters. In most situations, you will have a single Master requesting data from one or more Slave devices. Each device is given a 7-bit address. (10-bit addresses are available in extended addressing, but I've never seen one or needed one.) If you were to purchase an I²C device, it would already have a fixed address. (Some devices have alternate addresses, so you can use multiple devices on the same bus.) There cannot be more than one device on a bus using the same address, otherwise they would both try to answer a request.

The first byte of any I²C transmission contains the 7-bit address plus a read/write bit. So, in reality, a 7-bit address is shifted left 1 bit. The least significant bit becomes 0 (even) to write to the device, and 1 (odd) to read from the device. The 7-bit address 0x13 (00010011) becomes address 0x26 (00100110 = write) and 0x27 (00100111 = read). This is probably the most confusing part of the protocol. I'll be calling this new address the "address value," so as not to be confused with the 7-bit address.

A Master is responsible for driving the clock line. The clock line is used to determine when to interpret the data line. The data is allowed to change state only while the clock is low, and are stable when the clock line is high. Should the data line fall while the clock is low, this indicates a start bit (communication will commence). Should the data line rise while the clock is low, this indicates a stop bit (communication has ended). These rules are easily understood by examining **Figure 2**.

The first data byte is always sent by the Master and is the address value. The hardware in all the I²C devices wakes upon seeing a start bit on the I²C bus. All devices read the address value clocked by the Master. Each device has been given a unique 7-bit address. If an address match occurs, that device continues to pay attention, while all other devices disregard further communications. We've already discussed that the LSB of the address value indicates the mode of the transmission—1 for a read and 0 for a write.

At this point, only one device (or none if there were no device match) is still paying

attention and knows whether the Master will be sending more data (writing) or will be expecting data from the Slave (reading). So, a Slave either continues collecting data, reading it from the I²C register when it is full after each 8 clock bits, or places data into the I²C register before the next 8 clock bits. Where these received bytes go or from where these bytes come can be specific to each device. The standard approach is to read from a table where they are written. The table pointer is reset at each start bit and increments for each new byte. Let's leave it at that for now.

Since the bus is open collector (pulled high when no device is driving either line low), any device can pull a line low. For the data line, it's easy to see that if SDA (the data signal) is left undriven, any device looking at SDA will interpret this as a 1 (logic high), and if driven (pulled) low, any device looking at SDA will interpret this as a 0 (logic low). For the clock line, the Master will control this, but also pays attention to it. When it stops driving the clock line low, it expects it to go high, since Slaves are not in charge of the clock. However, when a Master wants to read data from a Slave by toggling SCL, it must wait until the SCL is high before clocking in that data from the Slave. In this way if a Slave isn't ready to load its data into the I²C register, it can hold down the clock line, which causes the Master to wait while it prepares its data.

If the clock line doesn't return high when the Master stops driving it, the Master pauses until it is released by the Slave. This acts as a handshake that data is ready!

SETTING THE TABLE

The Slave device we are creating for this project will do all kinds of data computations,



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.

so that a Master can receive weather data without having to interpret the reed switch functions of each sensor. Here's where we decide what that data will be. For the rain gauge, each time we get a low pulse, the tip bucket has flip-flopped, passing its magnet past the reed switch that momentarily grounds the pull-up on the sensor's input. According to the datasheet [1], each tip equals 0.011" of liquid. While 1 byte (0-255) could cover $0 \times 0.011''$ (0.000") – $255 \times 0.011''$ (2.800"), this is probably sufficient for most locations. However, the Master would still need to do the conversion to inches from the byte of data. Instead, we'll use 2 bytes. The high byte is whole inches and the low byte is hundredths of an inch. This covers up to 256.99" and requires no external computations, other than placing a decimal between the upper and lower bytes. In most cases, this may never exceed an inch if you are resetting the values every

Sensor Table	
Offset	Register Description
0	Whole inches of Rainfall
1	Fractional inches of Rainfall (hundredths)
2	Present Wind Speed (mph)
3	Average Wind Speed (mph)
4	Peak Wind Speed (mph)
5	Wind Direction (Cardinal Character 1)
6	Wind Direction (Cardinal Character 2)
7	Wind Direction (Cardinal Character 3)

TABLE 1

This project requires reading 7 bytes from this table to get real data from all three binary weather sensors.

Direction		Resistance	Voltage	Conversion
Degrees	Cardinal	Ohms	V=5V, R=10k	8-bit value
0	N	33k	3.84V	197
22.5	NNE	6.57k	1.98V	101
45	NE	8.2k	2.25V	115
67.5	ENE	891	0.41V	21
90	E	1k	0.45V	23
112.5	ESE	688	0.32V	16
135	SE	2.2k	0.90V	46
157.5	SSE	1.41k	0.62V	32
180	S	3.9k	1.40V	72
202.5	SSW	3.14k	1.19V	61
225	SW	16k	3.08V	158
247.5	WSW	14.12k	2.93V	150
270	W	120k	4.62V	237
292.5	WNW	42.12k	4.04V	207
315	NW	64.9k	4.33V	222
337.5	NNW	21.88k	3.43V	176

TABLE 2

Values taken from datasheet. I added the Cardinal and Conversion columns.

hour or day (writing the values of 0 to the table).

The wind speed sensor works on the same principle: a magnet passes a reed switch every complete rotation of the anemometer. The datasheet says that one rotation/s = 1.492mph, so we must include time in our computations for wind speed. I began thinking that we can use the same 2 table bytes as those that we used in the rain gauge, and be able to show wind speeds of up to 255.99mph using this format. However, I decided that fractions of miles per hour were unnecessary. Instead, I could show meaningful wind speed data in a single byte 0-255mph. This led me to realize that other wind speed data might be useful, so I set aside 3 bytes in the table for present, average and peak miles per hour.

Wind direction, as we've previously seen, is an analog value that can be boiled down to one of 16 directions. Each direction can have from one to three characters made up of the same four characters (N, E, S and W). Alternately, we might want to show degrees, again three characters—in this case digits (000-337). Remember, degrees are in increments of 22.5 degrees, which is the best we can do with this sensor. I'll stick to the 16 cardinal directions, thus, three table entries. **Table 1** is the complete table.

READING THE SENSORS

Sensor data for rainfall and wind speed are based on changes in the reed relay's state, and each state change has an opposite edge. The interrupt structure of this microcontroller (MCU), the Microchip Technology PIC16F18313, is two-fold. A dedicated interrupt input pin and a COS (change of state) interrupt can be enabled for any input pin. COS interrupts are all lumped together, so additional registers are required to see through the trees. Separate registers—IOCAP (positive or rising edge) and IOCAN (negative or falling edge)—for each input pin define what will trigger an interrupt. All inputs that have one or both edges enabled will affect the IOCAF register.

The COS interrupt is triggered when the reed switch of the rain sensor and/or wind sensor changes state, due to the position of its associated magnet. The input will drop and rise as the magnet passes the reed switch. We need to recognize only one of these edges. When the IOC (interrupt on change) interrupt is set, we must consult the IOCAF register to determine which input has caused it, and therefore, which routine to run. When the rain bucket tips, its interrupt will increment the variable RainBucket (word). When the wind speed rotates, its interrupt will increment the variable SpeedTick (word).}

The wind direction is handled in the main loop. Execution of any routines in the main loop are based on the passage of time. A separate timer interrupt handles this. It sets a flag once a set amount of time has passed, in this case 1 second. Back in the main loop, we wait to see this flag set before proceeding. An A-D conversion measures the resultant voltage that is presented to the RA5 analog input. The external 10kΩ 1% pull-up to V_{CC} on the RA5 input creates a voltage divider with any resistors that are enabled by the wind direction's vane magnet. All vane resistors are connected to the RA5 analog input, but only those grounded by a reed switch closure become part of the voltage divider. The resultant measurement is transferred to the ADCDIR register. A conversion flag can be polled to determine when it is appropriate to read this value.

CONVERTING SENSOR READINGS

Rainfall is by far the easiest sensor reading to convert. All conversions use fixed-point, multiply and divide routines. Each bucket tip is equal to 11 thousandths of an inch, so we multiply the number of counts in `RainBucket` by 11. A 16-bit by 8-bit multiply gives a potential 24-bit result, in thousandths. By dividing this by 1,000 we get both the whole and fractional parts of the result. The whole part will be transferred to the table offset 0 (as 0-255 inches), and the fractional part/10 will be transferred to the table offset 1 (as hundredths of an inch).

Converting wind speed is similar. Each anemometer rotation/s is equal to 1.49mph, so we multiply the number of counts per second in `SpeedTick` by 149. A 16-bit by 8-bit multiply gives a potential 24-bit result, which is in hundredths of mph. By dividing this result by 100, we lob off the fractional part and can transfer the whole part to the table offset 2 (as present MPH). Detecting the peak speed is just a matter of comparing the present value (table offset 2) to the peak value `MPH_P` (table offset 3), and saving the larger back into the peak value (table offset 3).

The average speed is another story entirely. Like rainfall, average wind speed will most likely be reset at some point, on an hourly or daily basis, so we'll want to be able to keep samples until this reset occurs. If samples are taken every second for 86,400 seconds (1 day), then we must be able to total a possible 255 mph × 86,400 seconds/day. That's a 32-bit value for the `MPHTotal` and a 24-bit value for the seconds count `MPHSeconds`. These registers allow the running average `MPH_A` to be calculated using a 24-bit divide routine.

Calculating wind direction from the sensor's voltage divider requires several comparisons. See **Table 2** for what we can expect from the 8 reed switch-resistor combinations. The voltage presented to the 10-bit ADC will convert to a 10-bit value. We will be using only the most significant 8-bits of each conversion. Note that the values are not in ascending order. A quick sorting of the list into **Table 3** allows Check values to be calculated based on the closest neighbor (conversion-wise), which will then be used in comparisons to determine the Cardinal direction characters to be placed into the table at offsets 5, 6 and 7 (Table 2).

I²C EVOLUTION

I²C addresses were once assigned by the developer of the I²C bus, Philips Semiconductor (now NXP Semiconductor), in the 1980s. Little did they realize how widespread the use of I²C would become outside of its intended local use. Even with its expanded world, the fact remains that each I²C device on a bus must have a unique address. In addition, the end user must know how each I²C Slave device operates, to use it effectively. The data sent to each device must follow the specific rules for that device.

I've discussed the simple transfers in which a Slave can have multiple registers where data can come from and go to. In this case, an internal pointer always begins at offset 0 at the beginning of any communications. This pointer automatically increments for every data byte sent or requested. The only issue here is that you can't get or put any arbitrary byte of data, without first getting or putting all the data up to that arbitrary offset. You can stop at any point, but you must always begin at offset 0.

Around 1995, Intel wanted to improve some of the shortcomings of the I²C bus, and the SMBus (System Management Bus) was developed. The SMB is based mainly on the principles of operation of the I²C bus. The SMBus protocols expand the usefulness of the bus without interfering with the simple Read/Write protocols of I²C. I encourage you to investigate more about these by reading the SMB specifications [2].

One of the improvements adds a "command" byte as the first byte following any write. This could be used for many things, as you will see in looking through the specs. I could have explained this without bringing SMB into the picture, but if you like I²C, SMB is worth checking out.

When a Slave device follows this format, the special use of the first write data byte in any I²C write, the data can be assigned to the register pointer. This

Direction	Conversion	Check
Cardinal	8-bit value	8-bit value
ESE	16	
		18
ENE	21	
		22
E	23	
		28
SSE	32	
		39
SE	46	
		54
SSW	61	
		67
S	72	
		87
NNE	101	
		108
NE	115	
		133
WSW	150	
		154
SW	158	
		166
NNW	176	
		187
N	197	
		202
WNW	207	
		215
NW	222	
		230
W	237	

TABLE 3

With the Conversion values sorted in ascending order, the Check column was calculated to find the average value. This value can be used for comparisons in the code, to determine to which Cardinal direction the vane's position refers.

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] and [2] as marked in the article can be found there.

RESOURCES

Argent Data Systems | www.argentdata.com
 Microchip Technology | www.microchip.com

```

// This example code is in the public domain.
String SignOn = "I2C Weather Probe 1/7/2020";
#include <Wire.h>
byte I2CADDRESS = 0x26;
byte BYTECOUNT = 8;
byte rainfall_W;
byte rainfall_F;
byte windspeed;
byte windspeed_A;
byte windspeed_P;
String winddirection = "  ";
//
void setup()
{
  Wire.begin();
  // join i2c bus (address optional for master)
  Serial.begin(115200); // start serial for output
  Serial.println(SignOn);
}
//
void loop()
{
  Serial.print("Checking I2C address " + String(I2CADDRESS));
  if(checkI2C()==0)
  {
    Serial.print(" ... reading");
    requestData();
    Serial.println(" ... Got it!");
    displayData();
  }
  else
  {
    Serial.println(" ... No one home at this address!");
  }
  Serial.println("Resetting rainfall!");
  resetrainfall();
  Serial.println("Resetting Average and Peak wind speeds!");
  resetwindspeed();
  Serial.println("Waiting 10 seconds before reading
    data again");
  delay(10000);
  Serial.println();
}
//
boolean checkI2C()
{
  Wire.beginTransmission(I2CADDRESS);
  return Wire.endTransmission();
}
//
void writePointer()
{
  Wire.beginTransmission(I2CADDRESS);
  Wire.write(0); // offset pointer=0
  Wire.endTransmission();
}
//
void resetrainfall()
{
  Wire.beginTransmission(I2CADDRESS);
  Wire.write(0); // offset pointer=0
  Wire.write(0); // rainfall_W=0
  Wire.write(0); // rainfall_F=0
  Wire.endTransmission();
}

```

(CONTINUED ON NEXT PAGE)

allows a user to point to any register, prior to additional data transfers. You can therefore zero in on the register you want, and not have to begin with offset 0 every time. Writing to a specific register would require only a write function of the format:

```

I2CWrite
      (7-bit Address*2)+0
      Pointer
      Data

```

To read from a specific register would require a write function (to set the pointer) and a read function of the format:

```

I2CWrite
      (7-bit Address*2)+0
      Pointer

I2CRead
      (7-bit Address*2)+1
      Data

```

You may still write or read multiple bytes, but the total number of bytes can be significantly less, depending on the offset. I'm sure you can see the efficiencies of using this format.

REGISTER READS & WRITES

I implemented registered addressing (using the first write to set the table pointer) for this project. This means the MCU's pointer is set by the first write data byte, as opposed to resetting it automatically to zero for each new read or write. Let's use an Arduino to read the data table from this MCU. The complete application is in **Listing 1**.

Now we can look at the I²C interrupt routine for this project (**Listing 2 - see p. 74**), and see how it handles the Arduino requests. The first Arduino routine, `checkI2C()`, checks to see if the I²C device is at a specific address by looking to see if the device acknowledges its address. This is handled by the I²C hardware in our project's MCU when its address (register `SSPIADD`) matches what is sent by the Master.

When the Arduino requests data, `requestData()`, this read must be preceded by a write to set the pointer to a specific offset. In this case the `writePointer()` routine always is zero. I've colored various sections of Listing 2 for reference. Looking at Listing 2, note that in the MCU's interrupt routine, when an address match has been made, the `Flag.Pointer` is set (red). The next time through, when the first data byte is received, this flag is tested. The branch here either uses this value as the new pointer, or on subsequent data it is stored into the table at the pointer (blue). The second part of the

LISTING 1


The Arduino makes a good "test bed" for interfacing to the project.

request data routine actually retrieves data from the table. Note that the MCU's hardware automatically holds down the SCL line on a read function, so your routine can have time to fetch the required data. This is released by setting CKP in the SSP1CON1 register as the code exits (orange) for each byte received. It's not necessary for writes, but it doesn't hurt, either.

To clear data, separate Arduino routines are used to handle data at different positions in the table—`resetrainfall()` at offset 0 and `resetwindspeed()` at offset 2. Your application should keep track of time and selectively clear registers as needed to give stats by hour or day. Our Slave device must recognize when writes are requested to specific registers (blue). Any writes to these will call routines that actually clear the appropriate registers required. `Resetrainfall` clears the counter `RainBucket0:1`, `Resetaveragewind` clears `totals MPHTotalU:H:L` and `MPHSecondsU:H:L`, and `Resetpeakwind` clears table register `MPH_P`. It can be noted for this example that the actual data values written are not used. In this case, the location written to triggers the clearing routines.

AND IN THE END...

As shown in **Figure 3**, I've built the circuitry on a protoboard that sits inside the rain gauge (Figure 1). If this were a PCB, I could spray it with coating like FINE-L-KOTE AR aerosol conformal coating, to protect it from moisture. For my hand-wired prototype, I plan to put a small plastic bag over it, leaving the bottom open. Note the two threaded standoffs epoxied to the PCB. This creates a firm mounting connection to the rain gauge base, keeping it from interfering with the tipping bucket. Since I had to debug each function separately on alternate I/O pins, `RA0:1` are required for debugging/programming. The jumpers allow the sensors to be temporarily rerouted until the code is debugged. **Figure 4** shows the circuit for debugging compared to the circuit for a code-ready PCB.

The kind of work that can be done with an 8-pin MCU continues to amaze me. Many sensor modules available today use one of the serial protocols—UART, SPI or I²C—for communications. I hope you can put this knowledge of I²C into practice in your projects. Like a physical tool in your workshop, it never hurts to add a new protocol to your repertoire. My “weather tree” is no longer dumb. The “smarts” added here help distribute the computing power of your system. This allows you to do more, because you need to do less. So little time, so much to do! 

LISTING 1: Continued

```

}
//
void resetwindspeed()
{
  Wire.beginTransmission(I2CADDRESS);
  Wire.write(2); // offset pointer=2
  Wire.write(0); // windspeed_A=0
  Wire.write(0); // windspeed_P=0
  Wire.endTransmission();
}
//
void requestData()
{
  writePointer();
  Wire.requestFrom(I2CADDRESS, BYTECOUNT);
  while(!Wire.available());
  winddirection = "";
  for(byte i=0; i<=BYTECOUNT; i++)
  {
    switch (i)
    {
      case 0:
        rainfall_W = Wire.read();
        break;
      case 1:
        rainfall_F = Wire.read();
        break;
      case 2:
        windspeed = Wire.read();
        break;
      case 3:
        windspeed_A = Wire.read();
        break;
      case 4:
        windspeed_P = Wire.read();
        break;
      default:
        winddirection = winddirection + Wire.readString();
        break;
    }
  }
}
//
void displayData()
{
  Serial.println("Today's rainfall " + String(rainfall_W) + "." +
String(rainfall_F));
  Serial.println("The wind is out of the " + winddirection + " at "
+ String(windspeed) + " MPH");
  Serial.println(" with an average windspeed of " + String(windspeed_A)
+ " MPH and peak of " + String(windspeed_P) + " MPH");
}

```



FIGURE 3

My “weather tree” outfitted with this project hand-wired PCB. I used a 4-wire twisted cable for the I²C connection to, in this case, the Arduino inside my shed.

```

;*****;*****
;
;       I2C INTERRUPT SERVICE ROUTINE
;
;*****;*****
;
Interrupt_I2C
    banksel    PIR1
    bcf        PIR1,          SSP1IF    ; clear the interrupt
    banksel    MyFSROL_Pointer
    movf       MyFSROL_Pointer,    W    ; retrieve the stored pointer
    movwf      FSR0L           ; use Pointer
    clrf       FSR0H           ; always in Bank 0
    banksel    SSP1STAT
    btfsc     SSP1STAT,    D_NOT_A    ; skip next if received byte was our address
    goto      Interrupt_I2C_Data     ; else it must be a data byte
;
Interrupt_I2C_Address
    movf       SSP1BUF,    W          ; dummy read of address
    bsf        Flags,    Pointer     ; set Pointer flag
;
    btfss     SSP1STAT,    R_NOT_W    ; skip next if Master needs to read data
    goto      Interrupt_I2C_Exit     ; else I need to read data from Master
;
Interrupt_I2C_AddressRead
    movf       INDF0,    W           ; get table value
    movwf      SSP1BUF           ; send it to Master
    goto      Interrupt_I2C_BumpPointer
;
Interrupt_I2C_Data
    btfsc     SSP1STAT,    R_NOT_W    ; skip next if Master needs to read data
    goto      Interrupt_I2C_DataRead ; else I need to read data from Master
;
Interrupt_I2C_DataWrite
    ; Data from Master
    btfss     Flags,    Pointer     ; skip next if the first byte
    goto      Interrupt_I2C_DataWriteContinue
;
Interrupt_I2C_DataWriteSetPointer
    movlw     low StartOfI2CData     ; bottom of table
    movwf     FSR0L                   ; as new Pointer
    movf      SSP1BUF,    W           ; get received value
    addwf     FSR0L,    F             ; use it as the new Pointer offset
    bcf       Flags,    Pointer     ; clear Pointer flag
    goto      Interrupt_I2C_CheckPointer
;
Interrupt_I2C_DataWriteContinue
    movf      SSP1BUF,    W           ; get received value
    movwf     INDF0                   ; save it to the table @ Pointer
;
    movlw     Rain_F
    subwf     FSR0L,    W             ; test for table offset 1
    btfss     STATUS,    Z           ; skip next if no match
    call      resetrainfall         ; else reset rainfall
;
    movlw     Wind_A
    subwf     FSR0L,    W             ; test for table offset 3
    btfss     STATUS,    Z           ; skip next if no match
    call      resetaveragewind     ; else reset average wind
;
    movlw     Wind_P
    subwf     FSR0L,    W             ; test for table offset 4
    btfss     STATUS,    Z           ; skip next if no match
    call      resetpeakwind        ; else reset peak wind

```

(CONTINUED ON NEXT PAGE)

LISTING 2

The I²C interrupt routine that handles communication requests from a Master (in this case the Arduino). Reading and writing are handled by the indirect pointer FSR0H:L. This pointer is auto-incremented after each data byte.

LISTING 2: Continued

```

;
Interrupt_I2C_BumpPointer
    incf    FSR0L,    F        ; increment the table Pointer
;
Interrupt_I2C_CheckPointer
    movlw   low EndOfI2CData    ; top of table
    subwf   FSR0L,    W        ; subtract top from Pointer
    btfss   STATUS,    Z        ; skip next if equal
    goto    Interrupt_I2C_Exit  ; else not a top so we're good to go
;
Interrupt_I2C_PointerRollover
    movlw   low StartOfI2CData  ; bottom of table
    movwf   FSR0L                ; as new Pointer
    goto    Interrupt_I2C_Exit
;
Interrupt_I2C_DataRead
                                ; data to Master
    banksel SSP1CON2
    btfsc   SSP1CON2,    ACKSTAT ; skip if ACK
    goto    Interrupt_I2C_Exit  ; else no more data is wanted
;
    movf    INDF0,    W        ; get value from table @ Pointer
    movwf   SSP1BUF        ; send it to Master
    goto    Interrupt_I2C_BumpPointer
;
Interrupt_I2C_Exit
    banksel SSP1CON1
    bsf     SSP1CON1,    CKP    ; release clock
    movf    FSR0L,    W        ; get the Pointer
    banksel MyFSR0L_Pointer
    movwf   MyFSR0L_Pointer    ; save it for next byte
    return
    
```

COLUMNS

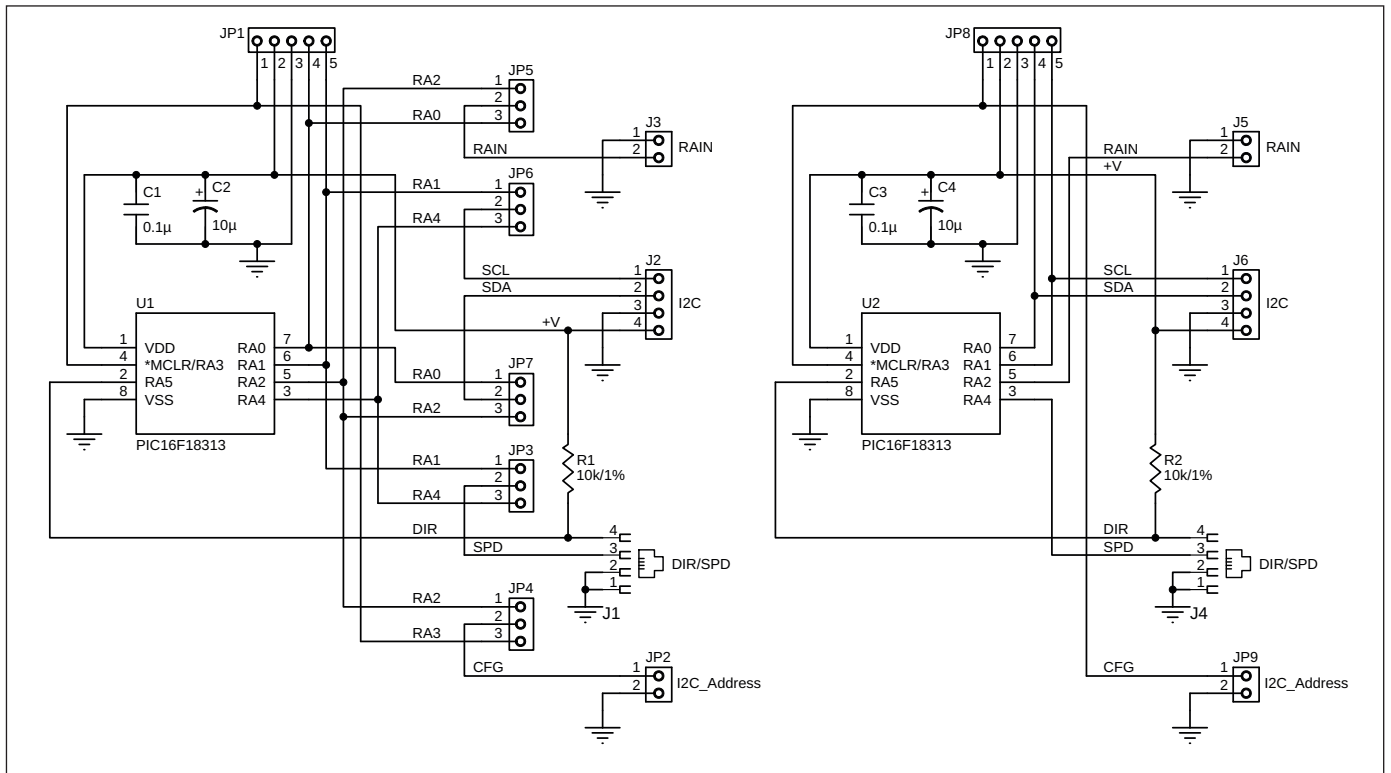


FIGURE 4

The schematic is drawn twice. The circuit on the left allows each sensor to be routed to a couple of different I/Os on the 8-pin microcontroller, so inputs RA0:1 can be dedicated for debugging. The circuit on the right has these sensors assigned to their final I/O, once the use of debugging pins can be released.

PRODUCT NEWS

Tiny Dual H-Bridge Control for Brushed DC Motors

Robot Power has announced the availability of the Scorpion Nano dual H-bridge motor control for brushed DC motors. The Scorpion Nano is the smallest member of Robot Power's Scorpion family. Designed for controlling motors in small robots, toys and other mechanisms, the Nano features two independent reversible motor drive circuits. Robot Power claims that the Scorpion Nano is one of the smallest, if not the smallest available dual RC motor controls with channel mixing for skid steer robot control.

The Nano features robust automotive grade motor drive ICs with full fault protection and a wide operating range. These motor drivers allow up to 1.2A of continuous current to each motor. Greater than 2A peak current is supported for short-term loads. RC wires are pre-installed along with a BEC function to power the RC receiver or microcontroller (MCU).

Battery and motor wires are also pre-installed for easy connection without soldering to the Nano PCB. A standard Scorpion family feature is a calibration button to allow the Scorpion Nano to adapt to the input signal range from either

an RC radio or MCU. A cut-able jumper allows the Scorpion Nano to be configured for RC channel mixing or separate (tank mode) control.

Robot Power | www.robotpower.com



Motor Controller Features Arm Cortex-M0 MCU

Infineon Technologies has released its new IMC300 motor controller series. It combines the iMOTION Motion Control Engine (MCE) with an additional microcontroller based on the Arm Cortex-M0 core. IMC300 complements the IMC100 series and aims at variable speed drives that require very high application flexibility. Both families, IMC100 and IMC300, share the same implementation of the MCE 2.0 providing ready-to-use motor and, optional, PFC control. Applying the MCE for controlling the motor, customers can focus on their system application that runs fully independently on the embedded Arm microcontroller. Infineon's field-proven MCE 2.0 implements highly efficient field-oriented control (FOC) of permanent magnet synchronous motors (PMSM).

IMC300 derivatives are offered for motor drives with and without PFC control. Devices in LQFP-64 packages (shown) are in mass production, and LQFP-48 types will be released in the second quarter 2020. Rapid prototyping of a drive inverter is enabled via two new control boards for the iMOTION Modular Application Design Kit (MADK). MADK is a modular and flexible development platform providing a range of control and power board options for motor drive applications up to 1kW.



Infineon Technologies | www.infineon.com

5G-Specific Cellular Amplifiers Roll

Wilson Electronics has announced the opening of pre-orders for the company's first 5G cellular amplifier, the Pro 710i. The Pro 710i, a single-band, commercial-grade amplifier by WilsonPro, will boost cellular signals on Band 71 (600MHz)

and will be publicly available for purchase in summer 2020. The telecommunications industry has been laying the groundwork for 5G for several years, which includes increased traffic on the Band 71 (600MHz) cellular frequency, says the company.

The Pro 710i's single, powerful indoor antenna port boasts +23dBm of downlink power and provides up to 100,000 square feet of indoor coverage, making it a perfect fit for large commercial buildings looking to improve their 5G cellular connectivity. This includes hotels, healthcare facilities, schools, manufacturing plants, commercial and residential real estate properties and more.

The Pro 710i cellular amplifier can be used as a standalone product to support the ongoing 5G rollout on Band 71 for T-Mobile, U.S. Cellular and other carriers. It can also be easily installed to run in parallel with any existing WilsonPro amplifier system without the need to replace existing amplifiers.



Wilson Electronics | www.wilsonelectronics.com

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 adc copy@circuitcellar.com with your file.

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

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

Get it today, cc-webshop.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

Adding TCP/IP..
Start with 3.3V Ethernet Development Kit

NOW \$25 OFF

CCS Development kits combine the powerful CCS optimized C compiler and ICD-U64 In-Circuit Programmer/Debugger with prototyping boards and hardware accessories.

Example Programs:

- Complete Web Server
- E-mail Generator
- SD Card Read/Write

www.ccsinfo.com/CC520
sales@ccsinfo.com - 262-522-6500 x 35

Circuit Cellar 2019 Digital Archive

With this digital subscription, you have access to all 12 issues of Circuit Cellar 2019 from any computer or tablet at anytime or download the PDF. Other years also available.

Order yours today
cc-webshop.com

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

TEST YOUR EQ

Contributed by David Tweed

Problem 1— A sequence detector receives serial binary data, grouped into 4-bit blocks. In any given 4-bit group, it must determine whether there were exactly two 1s. What is the minimum number of states required to accomplish this?

Problem 2— A similar sequence detector receives serial binary data, grouped into 4-bit blocks. But in any given 4-bit group, it must determine whether there were two or more 1s. What is the minimum number of states required to accomplish this?

Problem 3— A very long shift register needs to have at least one "1" circulating in it at all times. Obviously, one way to accomplish this is to wire up an enormous NOR gate to all of the stages (except the last), whose output drives the first stage. What is a different way to achieve the same result, while monitoring only a single point in the loop?

Problem 4— Why does the X-10 power line communications protocol specify that its tone bursts should be synchronized to the zero crossings of the power line voltage?

Circuit Cellar 2019 Archive

circuit cellar
circuitcellar.com

2019

Issues # 342-353 — CD #24

Copyright 2019, KCK Media Corp.

Order yours today

cc-webshop.com

The Future of Linux Security

Securing Linux-Based Systems in 4 Steps



Glenn Seiler,
VP of Linux Solutions,
Wind River

The world is increasingly interconnected and, as a result of this, the exposure to security vulnerabilities has dramatically increased as well. The intricacies of maintaining today's Linux-based platforms make it very challenging for developers to cover every potential entry point. In 2019 there was an average of more than 45 Common Vulnerabilities and Exposures (CVEs) logged per day.

How does a development organization keep up with that? In order to stay on top of this, developers must increasingly spend more time and effort integrating CVE patches into their solutions, at the cost of spending time developing their applications.

AUTHORITATIVE CVE SOURCE

Among other efforts, The MITRE Corporation [1] maintains the CVE system and is the authoritative source of CVE content, which is located on the CVE website [2]. MITRE functions as the editor and primary CVE Numbering Authority (CNA). CVE is well known across the industry for cyber threat sharing, vulnerability priorities and exposure names.

Security attacks come in many forms and use various entry points. Each attack type comes in several flavors, as there is usually more than one way that they can be configured or camouflaged based on the experience, resources and determination of the hacker.

While some threats are more prevalent than others, a developer needs to protect against all vulnerabilities. **Figure 1** shows the increase in CVEs over the last 6 years, and how many of those CVEs actually impact any given distribution.

MANAGEMENT PROCESS

To reduce threats on Linux-based systems, it helps to have a management process and four-step procedure to: monitor, assess, notify and remediate CVE threats. Now let's examine the process in more detail:

1. Active Monitoring: Monitoring is essential and required due diligence for staying ahead of threats in this ever-changing world. Obviously, this alone doesn't solve any problems, but does provide critical insight of potential vulnerabilities and is a differentiator with a trusted-vendor.

Neglect remains a big risk and some Linux providers are vulnerable from the very beginning with inferior due diligence. A solid security team's approach would include active monitoring, rapid assessment and prioritization, proactive customer notification and timely remediation to achieve a strengthened security posture.

It is important to constantly monitor the CVE database for potential issues. In addition, it is advisable to monitor specific security notifications from US

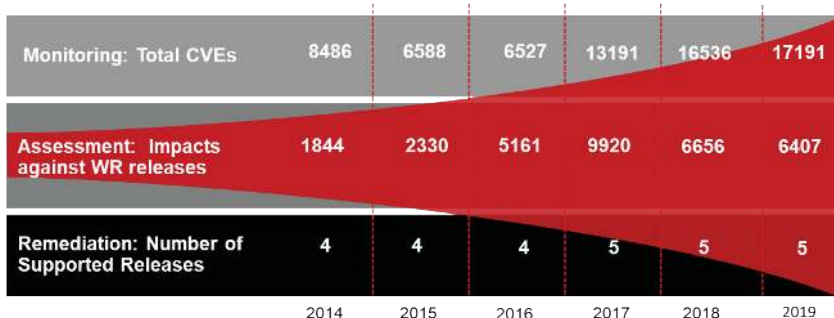
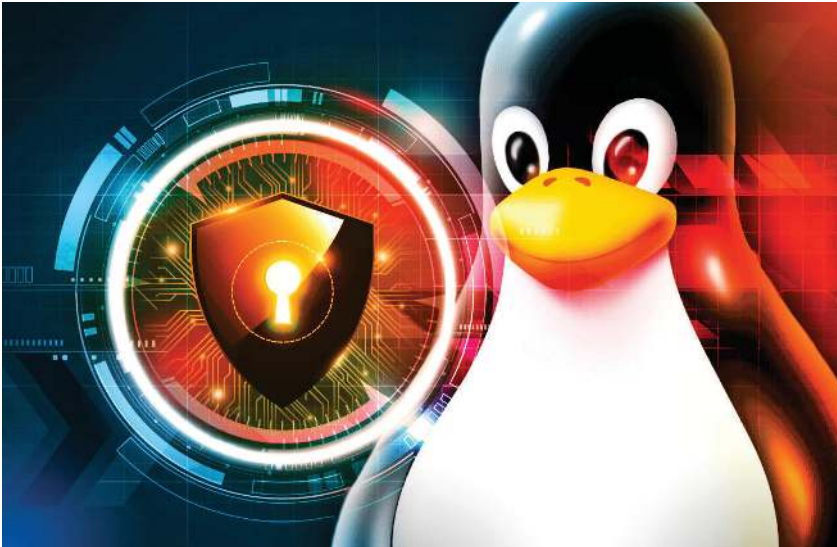


FIGURE 1
Increase of CVEs and 4-step process



government agencies and organizations like NIST, U.S.-CERT, as well as public and private security mailing lists, for alerts from each of these organizations whenever a new security threat arises.

2. Rapid Assessment: Awareness is only the first step. As soon as a potential threat is uncovered, the level of danger associated with the threat, as well as which parts of the Linux version are exposed or vulnerable, must be determined. The vulnerability is categorized and prioritized based on impact and ranked in order of importance based on the CVE priority level and the severity of impact to a business, system performance or exposure of data.

As noted in the next steps, mitigation of the vulnerabilities in this context typically involves coding changes, but could also include specification changes or even specification deprecations—for example, removal of affected protocols or functionality in their entirety.

3. Proactive Customer Notification: Once an assessment is complete, the system reports back to any affected subsystems or users. The report provides sufficient detail about the vulnerability, as well as a plan to thwart the threat. The harvested information also synchronizes with the remediation process.


The notification process is more vital than ever, and may involve employing outside tools and people. This is a determination that would be made based on the assessment of the vulnerability. However, the critical element of this step is the timely handling of notifications with customers to keep damage or data loss to a minimum.

4. Timely Remediation: The remediation process occurs, triage style. Teams should gather all the information relevant to the problem so that it can be analyzed. Based on the severity, threats are either dealt with immediately or handled in a timely "bug fix" manner, which would be deployed in a later update.

Companies can clearly benefit from using a commercially supported Linux. Commercially supported Linux offers low-costs, long term support and maintenance along with comprehensive development lifecycle services. A commercial vendor can supply the training, services, maintenance and support needed. This, in turn, increases productivity and reduces the overhead associated with maintaining a unique Linux distribution.

Regular maintenance, including the four-step CVE process, can radically reduce the hassles, lower the costs and protect customers from the risks of security vulnerabilities across their entire lifecycle.

FUTURE PLANS

While the above management process and steps discussed here won't ensure that all threats are avoided, they do help mitigate customer risk and reduce a system's exposure. Developers may want to apply a critical lens when deciding to build and support their own Linux distribution or when choosing a commercial Linux OS provider. Supporting security vulnerability needs to be considered and compared to how they follow the steps outlined here. 

For detailed article references and additional resources go to: www.circuitcellar.com/article-materials References [1] through [3] as marked in the article can be found there.

RESOURCES

Wind River | www.windriver.com

Glenn Seiler is Vice President of Linux Solutions for Wind River. In this role Glenn is responsible for defining and taking to market Linux and Open Source solutions for customers in embedded markets including Telecom, Industrial and Aerospace and Defense.



Start your learning journey.

IAR Academy is a technical training program offering a unique opportunity to boost your skills in embedded development, speed up project efficiency and meet tight deadlines easier.

IAR Academy On Demand provides you with courses through a self-service online training portal which allows you to access training at your desk or on the go, in a pace that fits your learning preferences. The portal also offers opportunities to register for live seminars, purchase different training modules and request custom on-site training for your organization.

With IAR Academy On Demand you have a choice in where, when and how to learn, with the possibility to easily get reference material from previous courses.

**START YOUR LEARNING JOURNEY!
SIGN UP NOW TO GET 3 FREE COURSES, 15 MIN EACH:**



www.iar.com/academy

FROM THE OUTBACK TO OUTER SPACE



TS-7800-V2

Single Board Computer with
Marvell Armada 385
Dual Core 1.3 GHz ARM CPU

Starting at
\$229
Qty 100

Extreme Remote Deployed Assets.

Technologic Systems computers are engineered to be deployed to some of the most remote places on, and off, the Earth.

The TS-7800-V2 was designed to provide extreme performance for applications demanding high reliability, fast startup, and connectivity at low cost and low power.

Packed with interfaces such as USB 3.0, SATA Gigabit Ethernet, CAN, and RS-232, RS-485, and TTL UART Serial Ports plus a full compliment of digital and analog I/O. With so many features integrated into one computer, you will accomplish more while reducing your payload weight.

Where does your project need to go?



Made in USA
with Global Parts

