

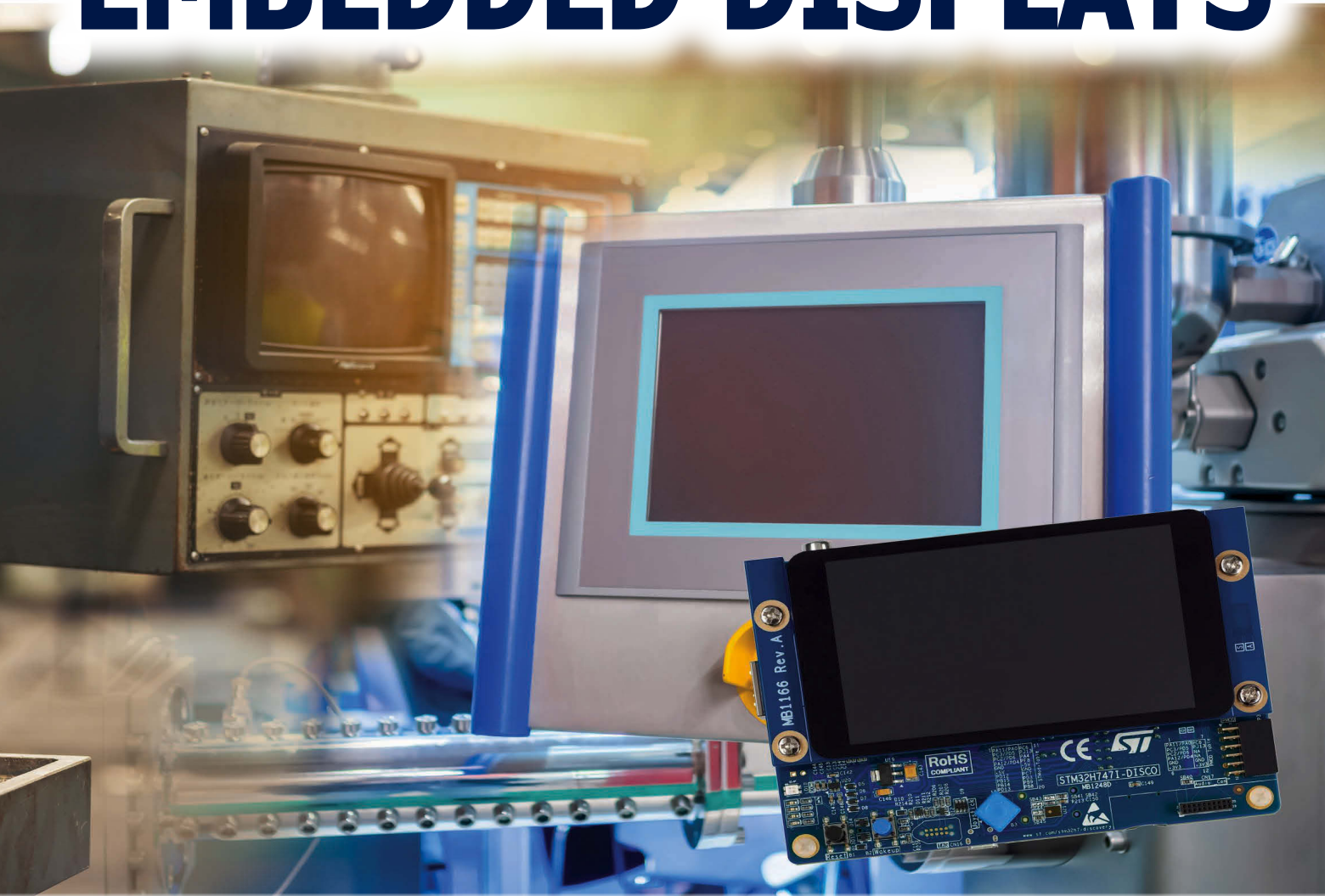
circuitcellar.com



circuit cellar

Inspiring the Evolution of Embedded Design

EMBEDDED DISPLAYS



▶ DC-DC Converters ▶ Holographic Display via Raspberry Pi |

Query a Database in PHP—Backend Web Development |

Playable MIDI Synthesizer ▶ How They Did It Before Transistors |

Secure C/C++ Code with CHERI | NoteCard for Embedded Communication

▶ The Future of RF Surveillance

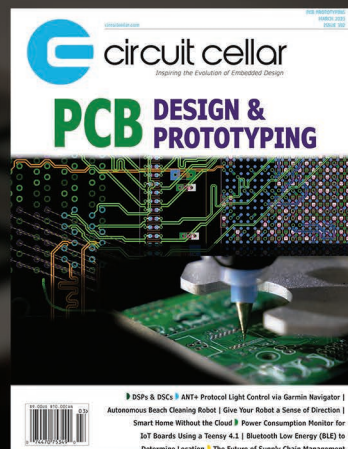
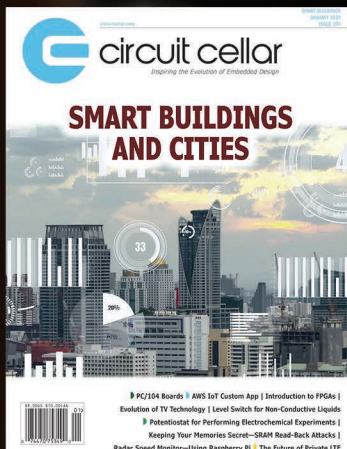


Imagine what you will

Discover



CC VAULT

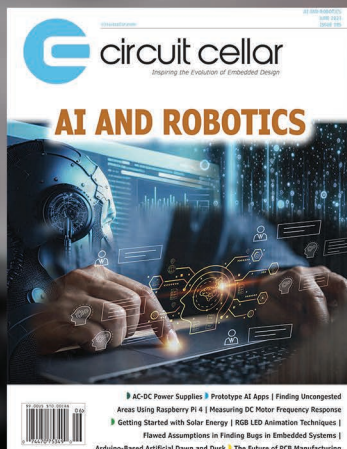
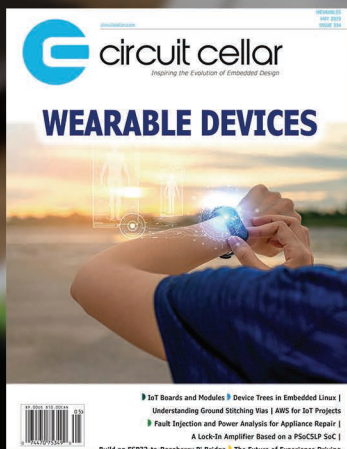


400+ ISSUES

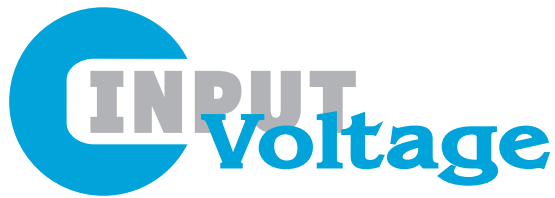


Get the complete *Circuit Cellar* issue archive and article code stored on a stylish, durable and portable USB flash drive. You can easily keep your CC Vault archive up to date by purchasing subsequent issues from our webshop or by downloading issues with a *Circuit Cellar* Digital Subscription. Issues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

cc-webshop.com



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




Circuit Cellar's 400th Issue

It's funny introducing *Circuit Cellar's* 400th issue. By most measures, I'm still new at this magazine. I've been working here a little over a year, and have, as of this publication, been the Editor-in-Chief of *Circuit Cellar* for 14 of its issues. In no way is this milestone—four hundred issues!—my accomplishment.

Nor, for that matter, is any issue I've been a part of "my" accomplishment. I'm the guy at a fancy restaurant who, after a team of highly trained expert chefs prepare a dish, makes sure there are no unsightly food smears on the plate before it goes to the table. *Circuit Cellar's* quality, success, and longevity rests on the shoulders of its tireless writers, some of whom have been with the magazine since its inception. Each issue feels like a low-grade marathon, and I have the easy job. I can't fathom doing 400 of these, as some members of our staff have done.

On the heels of a well-received article he wrote, Steve Ciarcia was hired by BYTE magazine in 1977 to write a column called "Ciarcia's Circuit Cellar," which presented projects he was working on. The column grew in popularity until Steve decided in 1979 to start a company called Micromint that would sell kits based on the projects he wrote about. These two ventures were a hit, and Steve enlisted the help of Ed Nisley, Ken Davidson, and Jeff Bachiochi to contribute their technical expertise to the column and its projects. When BYTE's editorial direction changed a few years after they were bought by McGraw-Hill, Steve founded his own magazine—this one—in 1988. (This means that we are also celebrating *Circuit Cellar's* 35th anniversary this year.) Many of the folks from BYTE followed Steve in this new endeavor. The *Circuit Cellar* magazine team in those days consisted of Steve Ciarcia, Ken Davidson, Jeff Bachiochi, Ed Nisley, Dan Rodrigues, Jeannette Dojan (who later became Steve's wife), Tom Cantrell, Dave Tweed, and many others. We still proudly count Jeff, Ken, and Dave among our staff.

I ran into a wrinkle in this story during my research. Steve posits in his account of *Circuit Cellar's* origins and history [1] that it was Dan Rodrigues who first suggested to Steve, upon hearing of BYTE's redirection, that they start their own magazine. But a few months ago, I received an e-mail from Bob Paddock, a former *Circuit Cellar* writer who, in the '90s and '00s, was a part of the "Ask Us" group for *Circuit Cellar Online*, and who also had his own column for a while. Bob claims that a comment he made to Steve started the whole thing off. He wrote: "I said to [Steve], 'What we really need is a magazine for hardware, like *Dr. Dobb's Journal* is for software.' He responded, 'Good idea,' and over a year or so later the first issue of *Circuit Cellar* magazine was a reality." I think both accounts are true, for the record. In Steve's own telling, he relied on the help of numerous other "hardware nerds" (Bob's term), and I don't doubt that, with the unwelcome changes taking place at BYTE, multiple of these clever engineer-writers were thinking the same thing.

However it happened, we've come full circle. Because I wouldn't be typing these words if it weren't for the decades of clever design, fascinating articles, and sheer engineering *fun* that have made *Circuit Cellar* what it is today. Nor would any of us be doing this if it weren't for our readers, who are, more often than not, experts in an increasingly sophisticated technical field who still find joy or knowledge in this magazine's pages. To borrow Steve's phrase, "we truly have a non-superficial readership." So, yes, it feels funny to introduce the 400th edition of *Circuit Cellar*. But it is no less an honor, a privilege, and a delight. I'm grateful to the *Circuit Cellar* team and to everyone reading. Please enjoy this special issue. 

[1] Steve Ciarcia, "Wondering How It All Began?" *Circuit Cellar's* 25th Anniversary Edition.

Sam Wallace
swallace@circuitcellar.com



Issue 400 November 2023 | 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: product-editor@circuitcellar.com

HEAD OFFICE

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

COPYRIGHT NOTICE

Entire contents copyright © 2023 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. 2023 Printed in the United States

OUR NETWORK



SUPPORTING COMPANIES

Avnet	49
Bel Fuse	65
Bussboard	15
CCS, Inc.	69
EmbeddedTS	69, C4
Microchip	70
OmniOn	47
Per Vices	21
Renesas	37
Siborg Systems Inc.	9
STMicroelectronics	33
TDK	C3
Texas Instruments	29

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

FOUNDER	Steve Ciarcia
PUBLISHER	KC Prescott
CONTROLLER	Chuck Fellows
EDITOR-IN-CHIEF	Sam Wallace
SENIOR ASSOCIATE EDITOR	Shannon Becker
TECHNICAL COPY EDITOR	Carol Bower
CONTRIBUTING EDITOR	Brian Millier
PROJECT EDITORS	Ken Davidson David Tweed
MARKETING MANAGER	Tori Zienka

ADVERTISING SALES REP.

Hugh Heinsohn

ADVERTISING COORDINATOR

Heather Childrey

COLUMNISTS

Jeff Bachiochi (From the Bench)
Stuart Ball (Start to Finish)
Joseph Corleto (The Magic Smoke Factory)
Bob Japenga (Embedded in Thin Slices)
Brian Millier (Picking Up Mixed Signals)
Colin O'Flynn (Embedded Systems Essentials)



4 Building a Holographic Persistence-of-Vision Display

Paint Light Into Ethereal Floating Images Using a Raspberry Pi Pico

Michael Crum, Joseph Horwitz, and Rabail Makhdoom

12 Backend Web Development for MCU Clients

Part 2: Querying a Database in PHP

By Raul Alvarez-Torrico

22 RPiano: A Playable MIDI Synthesizer

On a Raspberry Pi Microcontroller

By Samiksha Hiranandani

30 TECHNOLOGY FEATURE Embedded Displays

By Michael Lynes

38 DATASHEET DC-DC Converters

From the Hyper-Small to the Far Out

By Sam Wallace

42 Picking Up Mixed Signals Before Transistors

How Did They Do It Back Then?



By Brian Millier

52 Embedded System Essentials How CHERI Helps Secure Your C/C++ Code

On an FPGA

By Colin O'Flynn

56 From the Bench Cellular, The Forgotten Wi-Fi

Part 3: Using NoteCard, an Embedded Communications Module

By Jeff Bachiochi

71 TECH THE FUTURE The Future of RF Surveillance Advancements in Drone RF Surveillance

Harnessing High Bandwidth and Wide Tuning Range
Software-Defined Radios (SDRs)

By Brandon Malatest

BONUS Digital Edition Feature Addition Designing Combinational Circuitry

Employing Tiny Logic

By Wolfgang Matthes

66 : PRODUCT NEWS

70 : TEST YOUR EQ



Building a Holographic Persistence-of-Vision Display

Paint Light Into Ethereal Floating Images Using a Raspberry Pi Pico

FEATURES

By
Michael Crum, Joseph Horwitz,
and Rabail Makhdoom

FIGURE 1

Three examples of the holographic persistence-of-vision (POV) display. (Note the Circuit Cellar logo in the center.)

Persistence of vision (POV) is the human brain's ability to perceive light for a brief period after it stops entering the eye. These three Cornell University students exploited the POV phenomenon to create the illusion of holographic images by changing the colors of a rapidly rotating series of LEDs.

Holograms are a common fixture in science fiction, yet remain somewhat of a unicorn for the tech world. While building a "real" hologram might be out of reach for today's technology, we can still strap a horn to a metaphorical horse and make it feel pretty. Persistence of Vision (POV) displays offer one method by utilizing a psychological trick to construct floating images out of light.

Persistence of Vision refers to the brain's tendency to perceive light for a brief period after it stops entering the eye. Through clever engineering, this effect can be exploited to "paint" light onto thin air. A quickly rotating series of LEDs appear to the brain as a full circle, for example, and by changing the colors emitted by the LEDs, we can create the illusion of floating holographic images. These images are ghostly, beautiful, and mesmerizing—perfect for advertising, art installations, or product presentations (**Figure 1**).

The unique design challenges associated with creating a high-speed, fully wireless (both for power and communication), and low-budget

POV display led us down many interesting paths in a variety of engineering disciplines. Our electronics harnessed the Raspberry PI Pico microcontroller (MCU) to drive the display, and we created custom printed circuit boards (PCBs) to house the microcontroller, LEDs, and accompanying electronics. The whole system is powered inductively, removing the need for any wires. Our software consists of embedded C programming for high-speed operation of the Pico, along with a Python TCP client to send images to the display over Wi-Fi. Finally, our mechanical design uses 3D-printed components to enable safe, high-RPM operation.

ELECTRICAL OVERVIEW

In a system experiencing high accelerations, PCBs are king. Made from high-strength PTFE substrate, these boards can stand many thousands of Gs, and soldered connections are extremely resilient to the characteristic forces of a POV display. They are also lightweight and slightly flexible, making them perfect for our use case. **Figure 2** shows the two PCBs we made for our design.

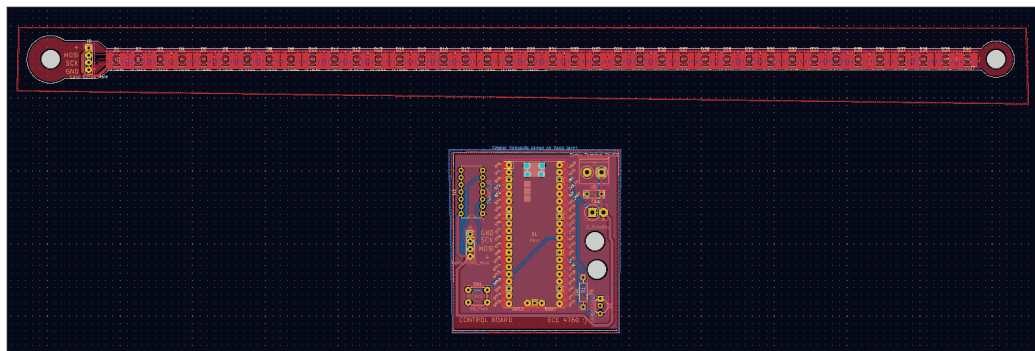


FIGURE 2
Our PCBs laid out in KiCad. Top: "The arm," which holds 40 surface-mounted APA102 LEDs. Bottom: The control board, which holds the Pico W and the power/logic electronics.

We call the first PCB "the arm," shown lit up in **Figure 3**. The arm holds 40 surface-mounted APA102 LEDs, and provides standard 0.1-inch headers for interfacing with the LEDs. We chose the APA102 LEDs because they use a two-wire SPI protocol to communicate with the control board. This allows communication rates of up to 20MHz, more than fast enough for our application. We previously experimented with the popular WS2812B LEDs, but these LEDs are capped at a 1kHz refresh rate due to their single-wire protocol. This would limit the radial resolution of our display. We added an M3-sized hole on each end of the arm, one to connect the arm to the rest of the rotor, and one to attach weights to balance the system.

The second PCB, shown in **Figure 4** is the control board. The control board holds the Pico W and the power/logic electronics to facilitate communication with the LEDs and Hall effect sensor. The Pico W uses 3.3V logic levels, while the APA102 LEDs expect 5V logic. To remedy this disparity, we included a 74AHCT125 Logic Level shifter. This shifter converts our 3.3V signal to 5V, and is fast enough to deal with our 20MHz SPI signals. A 47µF decoupling capacitor is placed across the

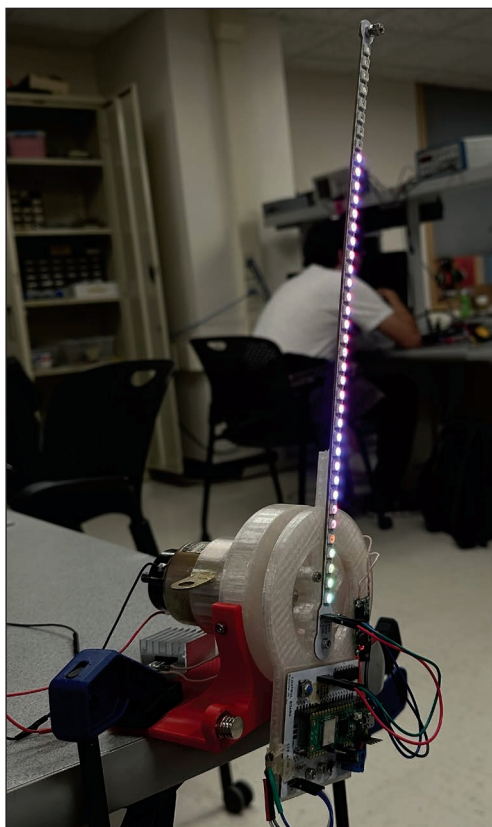


FIGURE 3
"The arm" mounted and with LEDs lit.

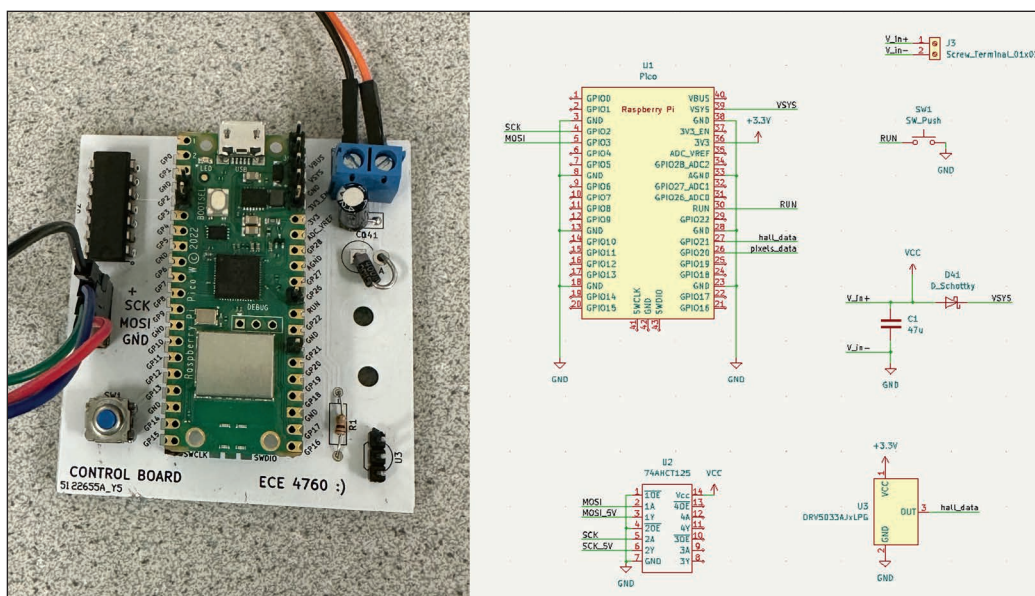
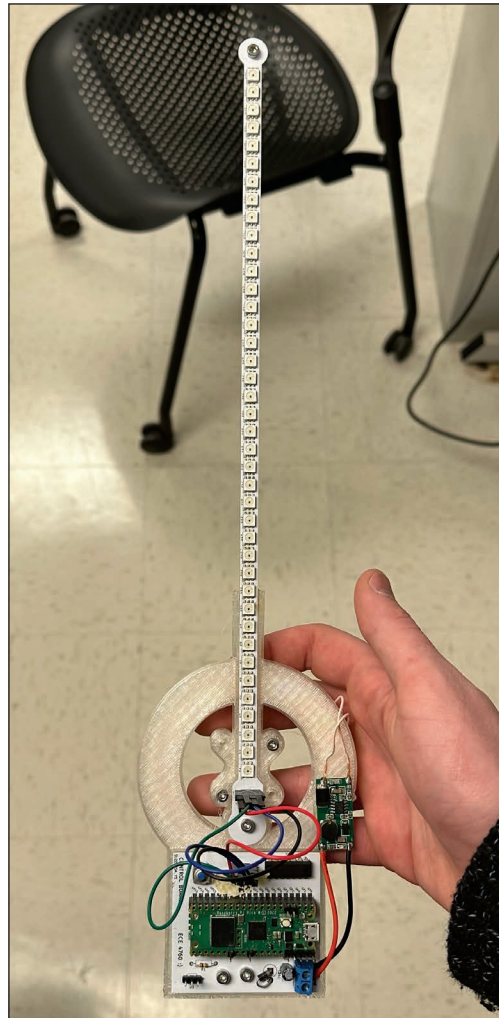


FIGURE 4
The control board PCB and schematic. All PCBs were designed using KiCad, an open-source ECAD software.

FIGURE 5

Inductive coil with 5V level converter hardware mounted on the bottom of "the arm." The coil powers the rotor, which spins 360 degrees.

**FIGURE 6**

The fully assembled rotor.

power supply, which is especially important when dealing with the rapidly changing power requirements of the LEDs. We also added a Schottky diode between the power rail and the Pico's VSYS pin. This diode allows the board to simultaneously take power from screw terminals and the Pico's onboard USB without damaging the Pico or the power supply.

To facilitate programming, we connected a push button between the RUN pin and ground. Pulling the RUN pin down causes the Pico to enter boot-select mode and appear as a programmable USB device. Finally, we wired the Hall effect sensor to a GPIO pin of the Pico with a 10k Ω pull-up resistor. Note that the sensor is active low.

One of the key design choices for a POV display is how to power the rotor. Because it spins 360 degrees, wires cannot be safely routed between the stationary stand and mobile rotor. There are three traditional approaches to this issue: a slip ring, an onboard battery, and an inductive power supply.

Slip rings use brushes and contacts to create connections that can "slip" past each other and rotate. However, they are notoriously unreliable, cause sparks under high load, wear over time, and add friction. An onboard battery adds weight to the rotor and is a potential safety concern at high speed. Finally, there are inductive power supplies. Due to the widespread adoption of wireless charging technology, inductive coils are readily available online. They are frictionless, robust, and are by far the "slickest" solution, if that matters (let's be honest—it does). We picked up a \$25 system on Amazon complete with a 5V level converter hardware, shown installed on the arm in **Figure 5**.

Finally, we need to spin the rotor. We used a spare motor found around the lab, but most motors will do. Our motor used 18W to achieve 1,800rpm (equivalent to 30fps), so look for something in that range if you build this project yourself. This motor is powered by a motor speed controller built from a second PI Pico and an HBridge. This allowed us to control the motor speed precisely, but a bench supply would also suffice.

MECHANICAL OVERVIEW

We started the design process by working on the rotor. As mentioned in the previous section, the PCBs, themselves, were included in the mechanical construction of the rotor. To supplement the PCBs, we needed to create a superstructure that holds the PCBs together and connects them to the motor shaft. This structure also served to mount the inductive coil. Along with the functional requirements, we want to keep weight to a minimum and make the design modular so that design iterations are faster.

Our design is 3D printed with minimal infill to reduce weight. It is only a couple of millimeters thick, and is designed to use the PCBs to supplement its strength. Components are connected using M3 screws that are threaded directly into the PLA. With proper print settings, these connections are remarkably strong, and more than enough for the mostly lateral load of this application.

To interface with the motor, we created an adapter that fits the motor shaft on one end and supplies a 1" square hole pattern on the other. We made this a separate component, so that we could quickly iterate designs in case the fit on the motor was too loose. The final product is shown in **Figure 6**.

The next step was creating a stand to house the motor and inductive coil. The inductive coil has a specific range in which it can operate safely, and we used the stand to enforce this distance. The stand also allows us to clamp the system to a table for testing.

The design is split into two parts to reduce reprinting time. All parts are printed in PLA with 20% infill, which was plenty strong enough for the application. PLA is not ferromagnetic, which means that it does not interfere with the inductive power supply. The motor mount is shown in **Figure 7**.

SOFTWARE OVERVIEW

Using just 40 independently addressable LEDs, we were able to create the illusion of 12,000 pixels at over 30fps. The display is 26" in diameter, and updates over Wi-Fi from our custom Python client.

To display an image, we first use a Python program running on a laptop to convert an image into the display's polar coordinate system. This data gets sent over TCP to the Pico W, where it is prepared to be displayed. We created a browser-based GUI to streamline the process, accepting images or GIFs and handling the full transmission cycle.

The Pico measures its rotational speed using a Hall effect sensor and a magnet mounted to the stand. With this information, it displays the pixels for the "slice" of the image corresponding to its current position in the rotation.

The MCU code executes two processes, each running on its own core. One process handles TCP exchanges and writes the image array with new pixel data. The other process reads the image array and updates the LEDs to maintain a complete image. By using both cores, we can concurrently receive TCP messages and control the LED strip, allowing for seamless operation.

Python Code: Our Python code creates an HTTP server that allows users to submit images to be shown on the display. The front

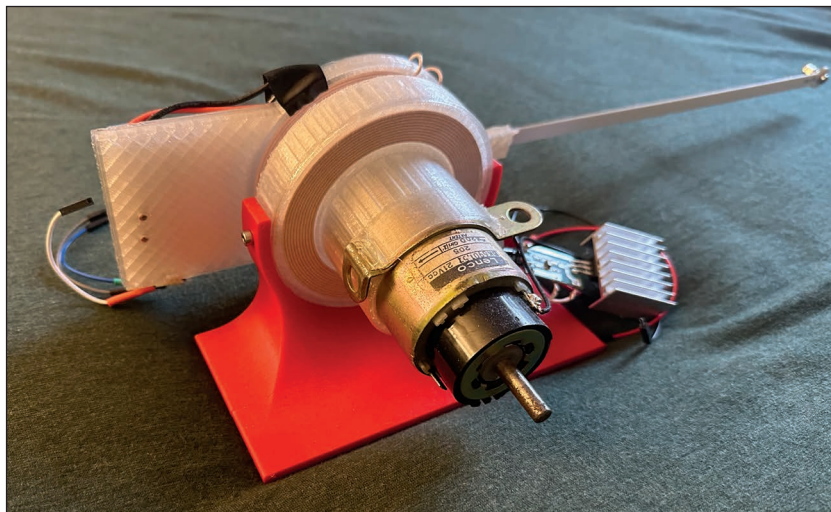


FIGURE 7
The final motor mount.

end (shown in **Figure 8**) uses simple Javascript to POST the user's image to the server, where the server downloads and caches it.

Raster images are typically displayed on rectangular screens, utilizing a rectangular array of pixels. POV displays are unique in the radial arrangement of their pixels, meaning we must pre-process the images from the canonical rectangular system into the display's native polar coordinates.

Our approach is virtually overlaying the location of the display's pixels over the rectangular image. We chose to center the circle and have its diameter be the same as the smallest dimension of the source image. This focuses on the central parts of the image and maximizes the amount of the display utilized. Any pixels outside the circle defined by this radius are ignored. For each of the pixels on the display, the closest pixel of the rectangular image is selected, and that color is used for the radial representation.

When processing the image, we must decide on a resolution. Because the number of LEDs on the arm is physically determined,

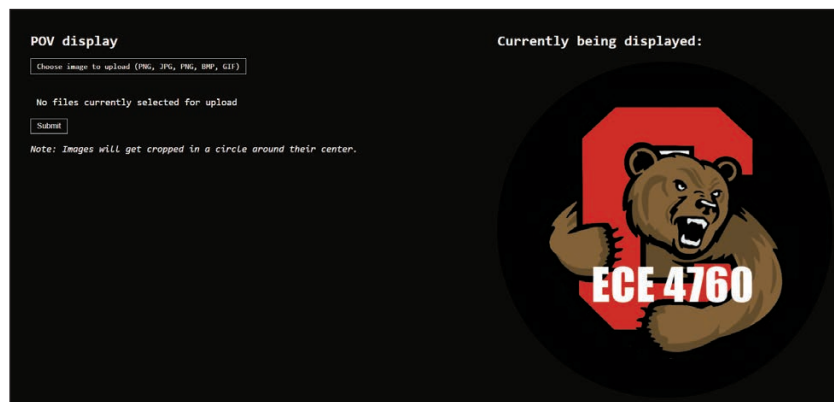


FIGURE 8
The front end for the Python code. It uses simple Javascript to POST the user's image to the server, where the server downloads and caches it.

we can only control the angular resolution. This is the number of times LEDs must change color while traveling in one rotation. While higher angular resolution results in a clearer image, it also strains the processing time of the Pico and the refresh rate of the LEDs. Our experimentation showed diminishing returns with >300 LED changes per rotation, so we stuck with that resolution for our final results.

The pixels are then pushed onto an array that stores the polar image. The array stores a pixel as (THETA, R, COLOR) rather than the traditional (X, Y, COLOR). Theta represents which angle of the arm contains the pixel, R represents the distance from the center of the arm in terms of the number of LEDs, and COLOR is an RGB triple.

Figure 9 shows what the processed images

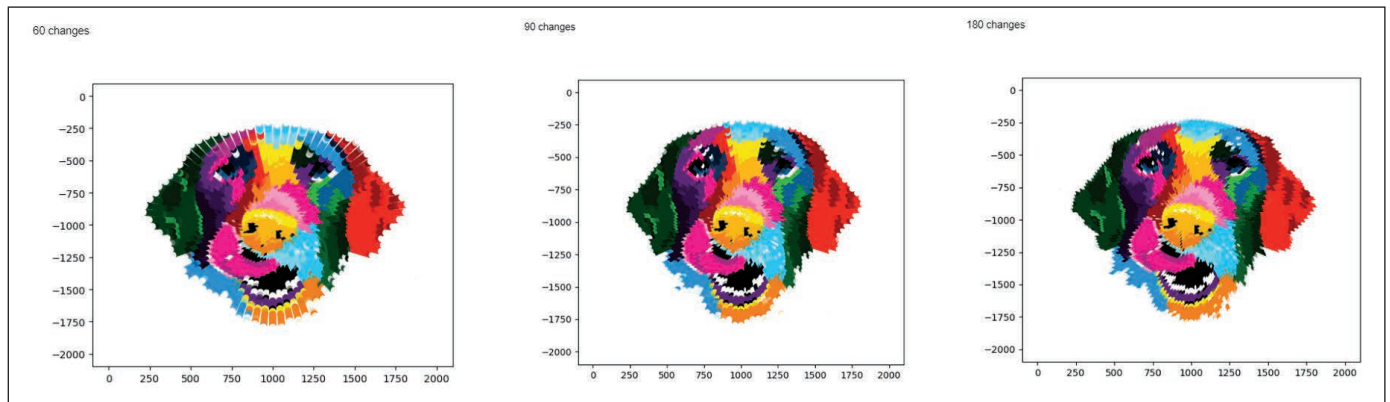


FIGURE 9

A test image produced at angular resolutions of 60, 90, and 180 LED changes per rotation. Angular resolution is the number of times LEDs must change color while traveling in one rotation.

```
# Sample the image at each point that an LED will update at
# We calculate this point in polar space, convert it to rectangular, then sample the image at that point
# The results are stored in rad_img, which is a polar representation. Essentially (theta, r)
for t in range(DIVISIONS_PER_ROTATION):
    for l in range(NUM_LEDS):
        # What angle are we looking at (in radians)
        theta = t * ((2 * numpy.pi) / DIVISIONS_PER_ROTATION) + offset
        # How far out are we (from 0 to 1)
        r = l / NUM_LEDS

        x_raw = numpy.cos(theta) * r
        y_raw = numpy.sin(theta) * r

        # Get the rectangular coord for the current polar coord, centered on the image and going to the edges
        x = numpy.interp(
            x_raw, [-1, 1], [(width / 2) - (min_dim / 2), (width / 2) + (min_dim / 2)])
        y = numpy.interp(
            y_raw, [-1, 1], [(height / 2) - (min_dim / 2), (height / 2) + (min_dim / 2)])

        x = int(x)
        y = int(y)

        assert x < width and x > 0
        assert y < height and y > 0

        rad_img[t][l] = img_array_np[x][y]
```

LISTING 1

The Python code for sampling an image in polar coordinates.

look like at various radial resolutions, and **Listing 1** shows the Python code for our implementation. We were able to run up to 300 LED changes per rotation, but reasonable images can be generated with 180 changes or less.

Once the image has been processed, we must send it to the Pico. This is handled using a TCP connection created by the Socket Python module. A laptop running the Python code presents itself as a TCP server, to which the Pico automatically connects. Once the connection is established, we can send our image as a stream of bytes to the Pico.

Pico Code: The Raspberry Pi Foundation provides an excellent SDK for programming the Pico, including all the build tools necessary for deploying code, and a collection of drivers for the various peripherals of the RP2040. Our work makes extensive use of this SDK, along with the popular Protothreads threading library for concurrent programming [1].

The RP2040 included on the Pico is dual-core processor, which is perfect for our use case. The display can be broken down into two high-level components: networking (talking to the laptop over Wi-Fi); and control (controlling the LEDs and ensuring timing consistency). Running each component on a separate core separates the interrupt-heavy and asynchronous requirements of network programming from the timing critical

and processor-greedy requirements of peripheral control. Additionally, because the control logic only reads data from shared memory (never writing) there is no concern over race conditions.

Core zero is responsible for the networking code, and starts life by initializing its peripherals. The networking on the Pico W is handled by an onboard CYW43439 chip, which has a handy driver packaged into the Pico SDK. After initializing the RP2040's GPIO pins, the CYW43439 driver is initialized and used to connect to a provided Wi-Fi SSID. We then register our custom interrupt handlers to manage TCP-related messages received by the Wi-Fi chip.

When the interrupt signifying a TCP transmission is triggered, data is fed to the interrupt as a packet of bytes (**Listing 2**). While TCP does guarantee the delivery of data, it does not guarantee how many packets the data will be formatted into when sent. This makes it the developer's responsibility to ensure that all data is received, even if it is broken into many packets. This problem can be handled by including a header with the message length, but our packet size is always the same (whatever is required by the resolution of the image) and can be agreed upon before the code is flashed to the Pico.

Because the size of an image is known, the Pico continues listening for packets until enough data has been

```
static int dump_bytes(const uint8_t *bptr, uint32_t len)
{
    unsigned int led_i;
    unsigned int rot_i;
    unsigned char rgb_i;
    uint8_t x;

    for (unsigned int i = 0; i < len; i++)
    {
        x = bptr[i];
        rgb_i = arr_i % 3;
        led_i = (arr_i / 3) % LED_NUM;
        rot_i = (arr_i / (LED_NUM * 3)) % ROTATIONS;
        led_array[rot_i][led_i][rgb_i] = x;
        arr_i++;
    }
    return rot_i + 1;
}
```

LISTING 2

The interrupt handler for TCP packets.

LCR-Reader[®] All-in-One Multimeter

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

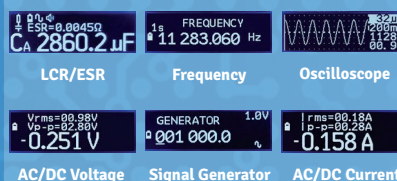
Basic Accuracy: 0.1%

Test Frequency:
100 Hz to 100 kHz

Test Signal Level:
0.1, 0.5, 1.0 Vrms



Optional Bluetooth model for data logging and quick Pass/Fail component assessment



LCR-Reader[®] Budget LCR Meter

LCR/ESR measurements
1 oz. weight
Automatic range selection
One button navigation
OLED display

Basic Accuracy: 0.5%

Test Frequency:
100 Hz, 1, 10 kHz

Test Signal Level:
0.5 +/- 5% Vrms



Siborg Systems Inc.

24 Combermere Crescent, Waterloo, Ontario, Canada N2L 5B1
Phone: 1-519-888-9906 Fax: 1-519-725-9522 www.Siborg.com

www.LCR-Reader.com

SIBORG
SYSTEMS INC.



FIGURE 10

Six different holographic images shown on the display.

received. It then rebuilds the 3D polar array that represents the image, identical to the array sent from the laptop's Python client. This array is stored in memory accessible to both cores, making it available to core one's display logic.

Core one tackles the issue of controlling the LEDs. The core first initializes the relevant GPIO pins for SPI communication, then outputs a test pattern to the LEDs as a visual indicator that initialization was successful. It also registers an interrupt handler for the rising edge of a Hall effect sensor, which is explained below.

Using the image information supplied by core zero, the image is almost ready to be displayed. We still need one more piece of information: the current position of the arm. To display a steady image, we need to know what "slice" of the image is currently being displayed. Instead of trying to measure the position of the arm directly, we use some mathematical trickery to form an estimate. By using a Hall effect sensor to determine when the arm passes a magnet mounted to the base, we get a sub-millisecond measurement of the period of the arm's rotation, and stable a zero point in the viewer's frame of reference. Given that the arm is rotating at a constant speed, the amount of time that each "slice" of the image should be displayed is the period divided by the number of pixels per rotation. This is easy to keep track of using an MCU, and we accomplished it using the yield functionality of the Protothreads library [1]. We determined the yield time using the following equation:

$$\text{yieldtime} = (\text{period of rotation}/\text{changes per rotations}) - \text{LED update time}$$

The Hall effect sensor we chose is active low and pulls a GPIO pin to ground whenever the south pole of a magnet gets close. We set up a falling edge interrupt on the pin, triggering whenever the sensor moves past the stationary magnet on the motor mount. When the interrupt is triggered, the period of rotation is calculated by subtracting the last activation from the current time. We also check that the period is a reasonable value ($>10,000\mu\text{s}$), which helps us reject any high-frequency false positives. We also indicate that we have hit our zero point by setting the relevant flag.

Finally, we can update the LEDs! We chose APA102 LEDs because they use the high-speed SPI protocol to communicate. A common pitfall of POV display design is attempting to use the ubiquitous WS2812b LEDs (also known as Adafruit NeoPixels).

These LEDs use a single wire control protocol and don't have the required bandwidth for high speed refreshes. The SPI interface has the additional benefit of allowing us to use the Pico's SPI peripheral to simplify the driver. The LEDs expect packets that are broken into "frames" of 32 bits. Each message begins with a start frame of 32 0's and ends with an end frame of 32 1's. In between, each frame represents the data for a single LED in the strip. A LED frame starts with 111, then is followed by 5 bits representing the brightness of the LED. This is followed by 8 bits for each of blue, green, and red, giving 256 values for each.

The LEDs are wired in series, with the SCK and MOSI lines of the previous LED leading into the next. When an LED receives a packet, it updates its state, strips the first LED frame off the packet, and then shifts the new packet out of its output SCK and MOSI lines. By doing so the entire strip can be updated from a single message sent to the first LED.

RESULTS OF THE DESIGN

Various holographic images on the POV display are shown in **Figure 10**. We can quantify the performance of our display in terms of several metrics:

1. Resolution: Rotational displays operate slightly differently than traditional grid-based displays. Each "pixel" is actually an arc, and its position is defined in terms of radius and angle rather than x and y. For a POV display, the resolution on the radius is the number of LEDs on the arm, so 40 in our case. The angular resolution depends on how many times the LEDs update per rotation. We experimented and determined that 300 updates produced vivid images without overwhelming our MCU. Multiplying these quantities gives 12,000 pixels, which is much higher than comparable DIY systems.

2. Size: POV displays become exponentially more complicated as they grow larger. Large radius results in higher acceleration, more LEDs required for equivalent pixel density, and more power required. Many POV projects are under 6" in diameter for this reason. Because our goal was to create a visually impressive product, we decided to aim for around the size of a large poster. This resulted in a 26" diameter display. This posed many technical challenges, but the result is absolutely stunning.

3. Image Stability: Due to the high speed of the system and the noisy signals generated by the Hall effect sensor, it can be difficult to determine the exact rotational frequency. This can cause the image to jitter or process around the display. Our display dealt with these issues remarkably, with almost no

visible jitter. Any noise was constrained to within one angular pixel, or under 2 degrees.

4. Usability: Our design emphasizes usability and consistency. Good build quality ensures that repeated use doesn't cause incremental damage, decreasing the life span. The system is powered by a single wall outlet, so no specialized hardware is required. It is resilient to fluctuations in motor speed, so replacement or modification is easy. The custom Python interface allows for use of the display with any image you choose, and updates can be sent over Wi-Fi without slowing down the display.


All the code and design files for this project are available on GitHub [2]. See *Circuit Cellar's* Article Materials and Resources webpage.

FUTURE WORK AND IMPROVEMENTS

One notable issue with our display is the spacing between the pixels. Because of the physical requirements for soldering the arm, there are small gaps between the LEDs. This causes circular interruptions in the image, and could be remedied by using two staggered rows of LEDs.

We also hope to improve the interface for transmitting images and video. It would be interesting to write a display driver that allows the display to mirror a computer screen. This would make the display interface even more intuitive.

SPECIAL THANKS

Special thanks to Cornell's Professor Hunter Adams and Professor Bruce Land for all of their help. 

ABOUT THE AUTHORS

Michael Crum (mmc323@cornell.edu) is an undergraduate Junior studying Computer Science at Cornell University. He is primarily interested in embedded systems and their applications in robotics. See more of his work at <https://michael-crum.com/>.

Joseph Horwitz (jah569@cornell.edu) is an undergraduate Senior in Electrical and Computer Engineering at Cornell University. He is also interested in embedded systems and firmware development.

Rabai Makhdoom(rm857@cornell.edu) is a Master of Engineering (M.Eng) student in Electrical and Computer Engineering at Cornell University. She is primarily interested in analog IC design, power electronics, and robotics.

Additional materials from the author are available at:

www.circuitcellar.com/article-materials

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

RESOURCES

Raspberry Pi | www.raspberrypi.com

Backend Web Development for MCU Clients

Part 2: Querying a Database in PHP

By
Raul Alvarez-Torrico

Proficiency with servers, HTTP, and backend technologies are valuable skills for the embedded systems professional. In Part 2 of this three-part article series, Raul steps us through creating a MariaDB database, how to use SQL queries to store data in the database with a slightly modified PHP script, and how to use a second PHP script to extract data from the database and send it back to a web client.

In Part 1 of this article series I discussed basic concepts regarding full-stack web development and backend/front-end web development. I also discussed a basic backend workflow for working with microcontroller (MCU)-based web clients. I explained how to set up a basic Linux web server with a database by installing the LAMP (Linux, Apache, MySQL/MariaDB and PHP) backend technology stack on a Raspberry Pi board.

I presented an Espressif ESP8266 MCU-based data logger with a Bosch Sensortec BME688 Environmental Sensor, as an example of an MCU-based web client. This data logger periodically sends sensor readings via Hyper Text Transfer Protocol (HTTP) POST requests to the web server. I explained as well a basic PHP script that runs on the server to attend the POST requests from the MCU web client. The script retrieves the sensor data that comes in the HTTP request's body and prepares a String Query Language (SQL) query that can be used store the values in a database.

If you are not familiar with the concepts described above, please refer to Part 1 of this article series ("Backend Web Development for MCU Clients," *Circuit Cellar* 399, October, 2023) so you can follow the topics presented here [1]. Here, in Part 2 of this article series, I discuss the creation of a MariaDB database on the server to store the remote sensor readings. I explain basic SQL queries to perform

diverse operations with the database, and I also discuss a second PHP script to query the database to retrieve previously stored data.

CREATING THE DATABASE

MariaDB is a SQL-based relational database, so to be able to interact with it, you must have a basic understanding of SQL. What is SQL? SQL is a standard language for storing, manipulating and retrieving data from databases. With SQL you execute queries against the database to store and retrieve data, update and delete records, create new databases and new tables, create new users, set access permissions, and so on. The SQL language is intuitive and easy to understand. Once you get acquainted with the most simple queries, using a database server becomes a straightforward activity.

Listing 1 shows the procedure to create a database, a table, and a user with all the necessary access privileges. From Part 1, you should have already set up your Linux-based web server with a MariaDB/MySQL database. On your Linux server, open a terminal window and run the command from line 5 to access MariaDB. Note that you can execute "sudo mysql" instead, to the same end. Next, run the SQL query in line 8 to list all currently available databases. From now on, remember to end all SQL queries with a semicolon (";"). MariaDB won't execute the query until you type the semicolon.

Run line 11 to create a new database named “logger_db.” You could use any other name for your database, but it is advisable to use the same names described here, to follow all procedures avoiding potential confusion. Run line 14 to select the newly created database. From now on, all issued SQL queries will apply to the selected database.

Now, we need to create a table in the database to store the data. To do so, in the command line write and run the query shown in lines 17-23. The indentation tabs are optional. To break the query in many lines, as it is shown in the listing, just hit <Enter> to break each line. Remember that SQL queries only execute when they are ended with a semicolon; thus,

```

1 # Creating a MariaDB/MySQL Database, Table and User
2
3 ### Create a Database and a Table
4 ### Start MariaDB:
5     sudo mariadb
6
7 ### List all available databases:
8     SHOW databases;
9
10 ### Create a database named 'logger_db':
11     CREATE DATABASE logger_db;
12
13 ### Select the created database:
14     USE logger_db;
15
16 ### In the database, create a table named 'sensors':
17     CREATE TABLE sensors (
18         unix_t INT(11),
19         gas_res DECIMAL(8,2),
20         pressure DECIMAL(8,2),
21         temperature DECIMAL(5,2),
22         rel_hum DECIMAL(5,2),
23         id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY);
24
25 ### As a test, insert manually a row in the table
26     INSERT INTO sensors (unix_t, gas_res, pressure, temperature, rel_hum)
27     VALUES('1688160823', '50178.01', '749.45', '25.74', '45.59');
28
29 ### Show all rows in the table
30     SELECT * FROM sensors;
31
32 ### Empty the table (delete all rows without erasing the table).
33     TRUNCATE TABLE sensors;
34
35 ## Create a New MariaDB User. With the user name 'user1' and password 'password1':
36     CREATE USER 'user1'@localhost IDENTIFIED BY 'password1';
37
38 ### Check user status:
39     SELECT User FROM mysql.user;
40
41 ### Grant Privileges to the new MariaDB User
42     GRANT ALL PRIVILEGES ON logger_db.sensors TO 'user1'@localhost IDENTIFIED BY
'password1';
43
44 ### Refresh privileges:
45     FLUSH PRIVILEGES;
46
47 ### Verify permissions for the new user:
48     SHOW GRANTS FOR 'user1'@localhost;
49
50 ### Remove MariaDB User Account:
51     DROP USER 'user1'@localhost;
52
53 ### Exit MariaDB:
54     exit

```

LISTING 1

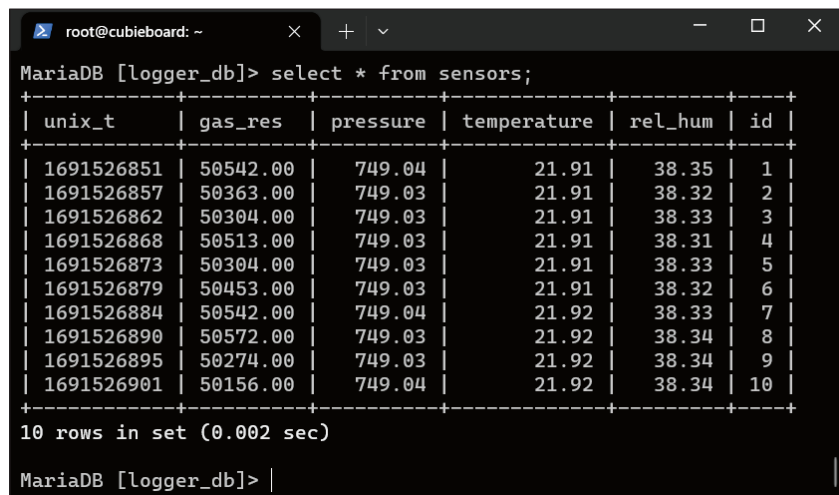
SQL queries to create a database, a table, and a user.

this query will execute only after you type the semicolon in the last line and hit <Enter>.

What does this SQL query do? It creates a new table named "sensors" in the currently selected database ("logger_db"). This table has

six columns: "unix_t," "gas_res," "pressure," "temperature," "rel_hum," and "id." The first column ("unix_t") is of type INT (integer) with 11 digits. The second column ("gas_res") is of type DECIMAL with a precision of 8 significant digits and a scale of 2 decimal digits. This means that this column can store a maximum value up to 999999.99. The same logic applies to the next columns, except for the last one ("id"). The last column (line 23) serves as the primary key that will help to uniquely identify each data row in the table. This column is of type INT UNSIGNED. It can never be null, and it auto-increments with each new row that's inserted in the table.

After the database table is created, run the SQL query from lines 26-27 to insert your first data row manually in the newly created table. The first parenthesis in this query contains the table column names, and the second parenthesis contains the corresponding values to be inserted in each column. Repeat if you want the same query many times, changing values to store additional rows. Next, run line 30 to display all data rows already stored in the table. The asterisk (*) in this last query



```

root@cubieboard: ~
MariaDB [logger_db]> select * from sensors;
+-----+-----+-----+-----+-----+-----+
| unix_t | gas_res | pressure | temperature | rel_hum | id |
+-----+-----+-----+-----+-----+-----+
| 1691526851 | 50542.00 | 749.04 | 21.91 | 38.35 | 1 |
| 1691526857 | 50363.00 | 749.03 | 21.91 | 38.32 | 2 |
| 1691526862 | 50304.00 | 749.03 | 21.91 | 38.33 | 3 |
| 1691526868 | 50513.00 | 749.03 | 21.91 | 38.31 | 4 |
| 1691526873 | 50304.00 | 749.03 | 21.91 | 38.33 | 5 |
| 1691526879 | 50453.00 | 749.03 | 21.91 | 38.32 | 6 |
| 1691526884 | 50542.00 | 749.04 | 21.92 | 38.33 | 7 |
| 1691526890 | 50572.00 | 749.03 | 21.92 | 38.34 | 8 |
| 1691526895 | 50274.00 | 749.03 | 21.92 | 38.34 | 9 |
| 1691526901 | 50156.00 | 749.04 | 21.92 | 38.34 | 10 |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.002 sec)

MariaDB [logger_db]>

```

FIGURE 1

Listing all data from the "sensors" table.

```

1 <?php
2 // Get the body (CSV string) from the incoming request
3 $csv = file_get_contents('php://input');
4
5 $data_array = str_getcsv($csv); //Convert CSV to array
6
7 if($data_array != null) {
8     $unix_t = time(); //Read unix time (GMT) from server
9
10     // Extract received remote values from array
11     $gas_res = $data_array[0];
12     $pressure = $data_array[1];
13     $temperature = $data_array[2];
14     $rel_hum = $data_array[3];
15
16     // Build SQL query string for the database
17     $query = "INSERT INTO sensors(unix_t, gas_res, pressure, temperature, rel_hum)
VALUES" . "('$unix_t', '$gas_res', '$pressure', '$temperature', '$rel_hum')";
18
19     // Connect to the database
20     require_once 'login.php'; // Include login information
21     $conn = new mysqli($server, $user, $password, $database);
22     if ($conn->connect_error) die($conn->connect_error);
23
24     // Query the database
25     $result = $conn->query($query);
26
27     // If the query was unsuccessful...
28     if (!$result) echo "Query error: $query\n" . $conn->error . "\n";
29     else echo "Data inserted into DB!\n"; // Send success message
30 }
31 ?>

```

LISTING 2

Receiving CSV data in the server and storing them in the database.


```

1 <?php // login.php
2 $host = 'localhost'; //Server host name or IP address
3 $database = 'logger_db'; // Database name
4 $user = 'user1'; // Change for your own user name
5 $password = 'password1'; //Change for your own password
6 ?>

```

LISTING 3

PHP script containing database login information.

simply means “all.” So, the query can be interpreted as follows: “Select and display all available rows from table sensors.” After running this query, in the terminal you will see listed all rows previously inserted in the table (**Figure 1**). If you want to get a fresh start with your database table, purge all data from the table by running line 33. The “TRUNCATE” query will empty the database table without erasing its structure. Now if you run line 30 again, you will get an empty table.

Next, we need to create a database user with all the necessary privileges to perform operations in the table. We accomplish this with the query from line 36. Before running this query, replace “user1” with your own user name, and “password1” with its corresponding password. After running line 36, run line 39 to get all available users in the MariaDB database. You should see in that listing the

user you just created. Going forward, run line 42 to grant all privileges that will allow the new user full control over the “sensors” table in the “logger_db” database. Here too, you must replace your own user name and password.

Run line 45 to reload the granted privileges, and run line 48 to verify that the user has received them. If for some reason you need to erase a user, run line 51 to do so. However, we need to keep the user we just created to access the database in the following examples. Finally, type “exit” to exit MariaDB (line 54). If you want to delve deeper into SQL, a suggested online tutorial [2] is available on the *Circuit Cellar* Article Materials and Resources webpage.

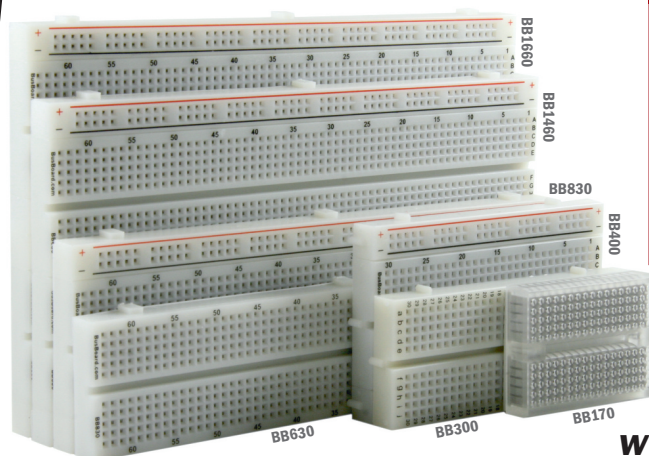
If you are new to the Linux terminal, interacting with a database from the command line can be a bit awkward at first, but it

Experience the BREADBOARD DIFFERENCE

BPS™ BusBoard
Prototype
Systems

Enjoy prototyping with high reliability precision breadboard contacts and avoid the frustration of poor breadboard connections. Usable with square-post headers and a range of wire sizes down to 26AWG.

Available for purchase at your trusted retailer



M MOUSER
ELECTRONICS

DigiKey

NEW

JAMECO
ELECTRONICS

amazon.com

For more details on
BusBoard breadboards and
prototyping PCBs visit:

www.BusBoard.com/CC



pays off in the long run. When working with servers, sometimes the only way to access them is by using a remote SSH connection from a terminal window. There is, however, some Graphical User Interface (GUI) software to interact with databases. Arguably the most popular for MySQL/MariaDB databases is “phpMyAdmin” [3] (written in PHP). It allows you to interact with MariaDB/MySQL from a web browser window.

STORING DATA IN THE DATABASE

Now that we have a working database in our server, let’s store data in it using PHP. How we do this? **Listing 2** shows the same PHP code I discussed in Part 1 of this article series [1], but this new version includes code lines 20–29. These additional lines open a connection to the database, insert the received data in the database table, and check for possible errors. Let’s see how this works.

Line 20 acts in similar way to the “#include” C language preprocessor directive. It includes the “login.php” script, which defines four variables containing database login information, as shown in **Listing 3**. “\$host” contains the host name or IP address of the database server. This will be usually “localhost” if the web server (where the PHP script is running) and the database server, both are running on the same computer. This is our case, and it generally is for most small to medium size web applications. “\$database” contains the name of the database you want to access (“logger_db”), which is the database we created previously (see line 11 in Listing 1). “\$user” and “\$password” contain, respectively, the user name and password for the database user with the required access privileges. In this script, change the user name and password

you chose when creating your database user (see lines 36, 42 from Listing 1).

Now let’s go back to Listing 2. Line 21 opens a connection to the database using the login credentials from the “login.php” script. Next, line 22 checks for any errors from the previous step. If an error has occurred, the `die()` function will terminate the script execution and display an error message. If the connection was opened successfully, line 25 will submit to the database the SQL query prepared in line 17. If the query is unsuccessful, line 28 sends back to the web client an error message containing the query string (“\$query”) and the connection error (“\$conn->error”). Otherwise, a success message is sent instead.

To test the backend so far, follow these steps: First, in your web server’s root directory, create a subdirectory called “backend” and copy into it the “receive_csv.php” and “login.php” files. We will be putting all server files inside this subdirectory. The root directory for Raspberry Pi servers or any other Debian/Ubuntu-based servers will be typically: “/var/www/html/”. So, the full path to our web application will be: “/var/www/html/backend/”. Next, connect the ESP8266 board to your PC, and upload the new version of the “esp8266_http_post_client.ino” Arduino sketch provided for Part 2 of this article series. This file and all other source code files are available on the *Circuit Cellar* Article Materials and Resources webpage.

In the first version of this Arduino sketch (given in Part 1) [1], I used the “ESP8266WiFi” library to manually build and send the HTTP POST requests to the server. By doing it that way, it was clearer how the HTTP requests are structured, protocol-wise. In

```

esp8266_http_post_client_2.ino
101
102 // Create and send sensor data as CSV string
103 csv_data = String(gas_resistance) + "," + String(pressure) + "," + String(temperature) + "," + String(humidity);
104 Serial.print("CSV string to send: ");
105 Serial.println(csv_data);
106 Send_Post_Request(); // Send data in an HTTP request
107 prev_millis = millis(); // Take current time to compute next interval
108 }
109 }
110

```

```

Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM32')
New Line
115200 baud

Temperature = 21.87 °C | Pressure = 749.07 hPa | Humidity = 38.49 % | Gas = 50.22 KOHms
CSV string to send: 50215.00,749.07,21.87,38.49
[HTTP] POST... code: 200
received payload:
Received CSV string: 50215.00,749.07,21.87,38.49
SQL query: INSERT INTO sensors(unix_t, gas_res, pressure, temperature, rel_hum) VALUES ('1691526685', '50215.00',
Data inserted into DB!

```

FIGURE 2

Server response to a POST request.

this second version, however, I'm using the "ESP8266HTTPClient" library that makes the sending and receiving of HTTP requests more straightforward, because of its additional abstraction layer. By comparing the two versions, you will see that the second one is more compact. Besides, in this new version, now we are reading real values from the BME688 sensor, instead of using random values to simulate sensor readings.

Remember to change your Wi-Fi credentials and your server's IP address before flashing the code. After flashing the code, open the Arduino IDE's serial monitor to see debug information. Once the board is connected to your Wi-Fi router, it will automatically start to send data periodically to the web server. Responses from the server will be printed on the serial monitor (**Figure 2**). Now, access MariaDB from the command line on your server by opening

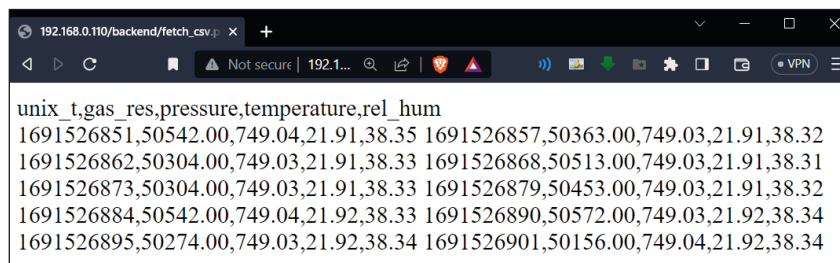
```

1 <?php
2 // If 'from' and 'to' dates arrived as key:value pairs
3 if (isset($_GET['from_date']) && isset($_GET['to_date'])) {
4     $from_date = $_GET["from_date"]; //Read date into local var.
5     $to_date = $_GET["to_date"]; //Read date into local var.
6
7     require_once 'login.php'; // Include DB login info
8     $conn = new mysqli($server, $user, $password, $database);
9
10    if ($conn->connect_error)
11        die("Connection failed: " . $conn->connect_error);
12
13    //Add begin/end hours to dates. Ex: "2023-08-08 00:00:00"
14    $from_date_hour = strtotime($from_date . " 00:00:00");
15    $to_date_hour = strtotime($to_date . " 23:59:59");
16
17    // Fetch data and send it back as CSV
18    Fetch_Db_Csv($conn, $from_date_hour, $to_date_hour);
19    $conn->close(); // Close DB connection
20 }
21
22 // Fetch data and send them back as CSV
23 function Fetch_Db_Csv($conn, $from_date_hour, $to_date_hour) {
24     // Build the SQL query
25     $query = "SELECT * FROM sensors WHERE unix_t BETWEEN '$from_date_hour' AND '$to_
date_hour' ORDER BY unix_t";
26
27     $result = $conn->query($query); // Query the DB
28
29     // If there's at least one row, build the CSV string
30     if ($result->num_rows > 0) {
31         echo "unix_t,gas_res,pressure,temperature,rel_hum\n";
32         $csv_row = "";
33         // Read data from row into local variables
34         while($row = $result->fetch_assoc()) {
35             $unix_t = $row["unix_t"];
36             $gas_res = $row["gas_res"];
37             $pressure = $row["pressure"];
38             $temperature = $row["temperature"];
39             $rel_hum = $row["rel_hum"];
40             // Build CSV string
41             $csv_row = "".$unix_t.','.$gas_res.','.$pressure','.$temperature','.$rel_hum'\n";
42             echo $csv_row; // Send CSV string row to HTTP client
43         }
44     } else { // No results that match the fetch criteria...
45         echo "0 results"; // Send feedback to web client
46     }
47 }
48 ?>

```

LISTING 4

Fetching data from the database and sending them back to web clients.



```

192.168.0.110/backend/fetch_csv.php
Not secure | 192.1...
unix_t,gas_res,pressure,temperature,rel_hum
1691526851,50542.00,749.04,21.91,38.35 1691526857,50363.00,749.03,21.91,38.32
1691526862,50304.00,749.03,21.91,38.33 1691526868,50513.00,749.03,21.91,38.31
1691526873,50304.00,749.03,21.91,38.33 1691526879,50453.00,749.03,21.91,38.32
1691526884,50542.00,749.04,21.92,38.33 1691526890,50572.00,749.03,21.92,38.34
1691526895,50274.00,749.03,21.92,38.34 1691526901,50156.00,749.04,21.92,38.34

```

FIGURE 3

Fetching the CSV file using a web browser.

a terminal window and executing “sudo mariadb”. Run the query, “USE logger_db;” and then “SELECT * from sensors;” to display the contents of the “sensors” database table. You should see listed all data received from the ESP8266 data logger that are stored in the database, similarly to what we saw previously in Figure 1.

FETCHING DATA FROM THE DATABASE

Now that we know how to store sensor data, let’s see how to retrieve data from the database and put them in a suitable transfer format for sending them back to the web client. We will consider basically two types of web clients requesting data from our server: a regular web browser, and an embedded, MCU-based web client, such as our ESP8266-based data logger. For web browsers, XML and JSON formats are generally preferred. For MCU-based clients, however, CSV will be generally more suitable. JSON can also be used for MCU-based clients, but it is less efficient than CSV in terms of memory usage.

In Part 1 of this article series [1], I described CSV as a lightweight format that is appropriate to use with low-memory devices such as MCUs. **Listing 4** shows the PHP script that fetches data from the database, formats them as a CSV string, and sends them back

to the requesting web client. Let’s see how it works.

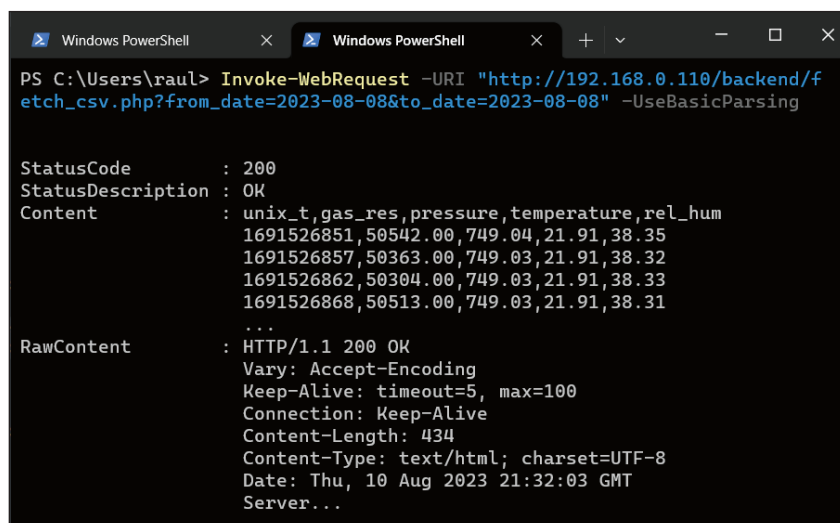
Line 3 verifies that the “from_date” and “to_date” key:value pairs arrived in the incoming HTTP GET request. Only if both keys are set, the script will query the database on the client’s behalf. Lines 4-5 store both dates into local variables.

Line 7 includes the “login.php” script with the database login information, and line 8 opens a connection to the database. Line 10 checks for any connection error; if there’s any, it aborts the script execution (line 11). Line 14 concatenates the “00:00:00” string to the “from_date”, because to search the database we need to specify the begin-hour along with the begin-date. The result will be a string like this: “2023-08-08 00:00:00”. Similarly, with line 15, we specify “23:59:59” as the end-hour for the “to_date”. This includes in the search all available readings until the last second of that day. Next, line 18 calls the `Fetch_Db_Csv()` function defined in lines 23-47. This function queries the database, prepares the retrieved data as a CSV string, and sends them back to the HTTP client. Finally, line 19 closes the database connection.

Let’s look at how the `Fetch_Db_Csv()` function works. The function receives as input parameters the database connection, the “from-date-hour” and the “to-date-hour” (see line 23). In line 25, it builds the SQL query that will be used to fetch data from the database. The “*” in that query means “all,” “sensors” is the table name, and “unix_t” is the column that stores the Unix time for each row in the table (Figure 1). So, in plain English, the full query can be read as follows: Select all rows from the “sensors” table where the Unix time column has a value between “from_date_hour” and “to_date_hour”. Arrange the results by Unix time in ascending order.

Line 27 submits the query to the database, and line 30 checks if there’s at least one row as a result. If so, line 31 sends as a first text row in the CSV file the column names. Lines 34-43 iterate over all available rows, extracting column values and storing them into local variables. For each table row, it then builds a CSV row containing sensor readings (line 41), and sends it back to the HTTP client (line 42). After processing and “echoing” all rows, the client will get a CSV file with all the sensor readings that match the request criteria.

Let’s test the “fetch_csv.php” script from a web browser. First, download the aforementioned PHP script from the *Circuit Cellar* Article Materials and Resources webpage, and copy it to the “/var/www/html/backend/” folder in your server. Next, in the URI below, replace the IP address for your web server’s. Replace as well the start



```

Windows PowerShell
PS C:\Users\raul> Invoke-WebRequest -URI "http://192.168.0.110/backend/fetch_csv.php?from_date=2023-08-08&to_date=2023-08-08" -UseBasicParsing

StatusCode      : 200
StatusDescription : OK
Content         : unix_t,gas_res,pressure,temperature,rel_hum
                  1691526851,50542.00,749.04,21.91,38.35
                  1691526857,50363.00,749.03,21.91,38.32
                  1691526862,50304.00,749.03,21.91,38.33
                  1691526868,50513.00,749.03,21.91,38.31
                  ...
RawContent      : HTTP/1.1 200 OK
                  Vary: Accept-Encoding
                  Keep-Alive: timeout=5, max=100
                  Connection: Keep-Alive
                  Content-Length: 434
                  Content-Type: text/html; charset=UTF-8
                  Date: Thu, 10 Aug 2023 21:32:03 GMT
                  Server...

```

FIGURE 4

Fetching the CSV file from the command line.

and end dates with dates you know for sure you have sensor readings stored in your database:

```
http://192.168.0.15/backend/fetch_csv.php?from_date=2023-08-08&to_date=2023-08-08
```

Open the URI in a web browser. An HTTP GET request will be sent automatically to the web server, which, in response, will send back the CSV file containing the sensor readings, as shown in **Figure 3**. Don't mind the rows don't show broken down properly; that's just because the browser doesn't recognize the "\n" character as a new line.

To get regular dates from the Unix timestamps in your database, convert them

using an online Unix time converter. For instance, I took the first Unix time from Figure 1 ("1691526851"), and after converting it, I got "Tue Aug 08 2023 16:34:11 GMT-0400." So, I used "2023-08-08" as start and end dates in the URI example above. You can use, however, different start and end dates; the server will send whatever data it finds in that time period.

You can also test the backend from a terminal window using the "cURL" library on a Linux machine. After changing relevant details, run the following command to get the CSV file:

```
curl "http://192.168.0.15/backend/fetch_csv.php?from_date=2023-08-08&to_date=2023-08-08"
```

```

1 #define SSID "MyWiFi"
2 #define PASSWORD "MyCatKnowsAssembly"
3 #define PHP_SCRIPT_URI "http://192.168.0.15/backend/fetch_csv.php"
4 String get_query_string; // 'GET' query string with key value pairs
5
6 void setup() { // Regular Wi-Fi initialization... }
7
8 void loop() {
9   static long prev_millis; // Stores time of the last publication
10  long elapsed_time = millis() - prev_millis;
11  if (elapsed_time >= READ_INTERVAL) { // Check time interval
12    String from_date = "2023-08-08";
13    String to_date = "2023-08-08";
14    get_query_string = "?from_date=" + from_date + "&to_date=" + to_date;
15    Send_Get_Request(); // Send the HTTP request
16    prev_millis = millis(); // Take current time
17  }
18 }
19
20 void Send_Get_Request() {
21   if ((WiFi.status() == WL_CONNECTED)) {
22     WiFiClient client;
23     HTTPClient http;
24
25     if (http.begin(client, PHP_SCRIPT_URI + get_query_string)) {
26       int httpCode = http.GET();
27       if (httpCode > 0) {
28         Serial.printf("GET code: %d\n", httpCode);
29         if (httpCode == HTTP_CODE_OK) {
30           String payload = http.getString(); Serial.println(payload);
31           // Parse CSV data here...
32         }
33       } else { Serial.printf("GET error: %s\n", http.errorToString(httpCode).c_str()); }
34       http.end();
35     } else { Serial.printf("Unable to connect\n"); }
36   }
37 }

```

LISTING 5

Code for the esp8266 HTTP GET client.

In a Windows 10/11 machine, open the Windows PowerShell and run:

```
Invoke-WebRequest -URI "http://192.168.0.15/backend/fetch_csv.php?from_date=2023-08-08&to_date=2023-08-08" -UseBasicParsing
```

Figure 4 shows the output from the PowerShell on Windows 10. cURL on Linux will show something similar. If you are not familiar with these command line tools, don't worry. Just use the web browser instead, as explained above. After receiving the CSV file, the HTTP client needs to parse it to get the individual values. There are available CSV parser libraries for virtually every programming language.

FETCHING DATA FROM THE MCU

Now let's see how to fetch the same CSV data using the ESP8266 MCU. To achieve this, we have to send practically the same HTTP GET request sent above, using the command line tools or the web browser. **Listing 5** shows

an excerpt of the "esp8266_http_get_client.ino" Arduino sketch that sends the required GET request to the web server. Before trying the sketch, remember to change the Wi-Fi credentials and PHP script URI in lines 1-3. The `setup()` function contains the same Wi-Fi initialization procedure as in the Arduino sketch that sends POST requests.

Inside the `loop()` function, there's also a non-blocking delay to send requests periodically on a time interval defined by the "READ_INTERVAL" constant. Inside the "if" statement (lines 11-17), the GET query string is built and the request is sent to the server. Lines 12-13 define the start and end dates for the data we are interested in to query the database. These dates must be generated dynamically, depending on the specific application. Here, for simplicity, we are defining them statically in the code.

Line 14 builds the GET query string by concatenating the "from_date" and "to_date" key:value pairs. This string will be appended to the base PHP script URI from line 3. Line 15 invokes the `Send_Get_Request()` function to send the request. Inside the aforementioned function (lines 20-37), a connection to the web server is opened, and the HTTP GET request is sent (line 25). The second argument to the function in line 25 is the string concatenation ("+") of the base PHP script URI and the GET request string. The resulting string will look this:

```
"http://192.168.0.15/backend/fetch_csv.php?from_date=2023-08-08&to_date=2023-08-08."
```

Line 26 retrieves the HTTP response code from the server. It will have a positive value if the server received and processed the request. It will be negative if a communication error occurred. If the code is 200 ("HTTP_CODE_OK"), the server has acknowledged our request and sent a response. So, we retrieve the payload from the HTTP response's body and print it to the serial monitor (line 30). This payload contains the CSV string with the sensor readings fetched from the database. **Figure 5** is a screen capture of the Arduino IDE's serial monitor showing the received CSV string. The first text row shows the HTTP response code. The CSV string begins in the second row, which shows the column names, followed by ten rows of sensor data.

After retrieving the payload, the CSV string must be parsed to obtain all individual sensor values. To keep the focus on the scope of this article, I will not describe here how to do the parsing. Nevertheless, the full source for this example contains parsing code using the "CSV_Parser" Arduino library. You can

```

esp8266_http_get_client_2 | Arduino IDE 2.1.1
File Edit Sketch Tools Help
NodeMCU 1.0 (ESP-12E...)
esp8266_http_get_client_2.ino
51  get_query_string = "?from_date=" + from_date + "&to_date=" + to_date;
52  Serial.print("GET query string: ");
53  Serial.println(get_query_string);
54  Send_Get_Request(); // Send the HTTP request
55  prev_millis = millis(); // Take current time to compute next interval

Output Serial Monitor x
Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E...)' New Line 115200 baud

[HTTP] GET... code: 200
unix_t,gas_res,pressure,temperature,rel_hum
1691526851,50542.00,749.04,21.91,38.35
1691526857,50363.00,749.03,21.91,38.32
1691526862,50304.00,749.03,21.91,38.33
1691526868,50513.00,749.03,21.91,38.31
1691526873,50304.00,749.03,21.91,38.33
1691526879,50453.00,749.03,21.91,38.32
1691526884,50542.00,749.04,21.92,38.33
1691526890,50572.00,749.03,21.92,38.34
1691526895,50274.00,749.03,21.92,38.34
1691526901,50156.00,749.04,21.92,38.34
Ln 91, Col 1 NodeMCU 1.0 (ESP-12E Module) on COM32 2

```

FIGURE 5

Fetching the CSV file using the ESP8266.

Additional materials from the author are available at:

www.circuitcellar.com/article-materials

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

RESOURCES

Arduino | www.arduino.cc

Espressif | www.espressif.com

download it from the *Circuit Cellar* Article Materials and Resources webpage.


It is worth noting, however, that receiving and parsing CSV strings can consume a great amount of the MCU's RAM, depending on the size of the incoming payload. This limits in practice the amount of data you can receive and process with an MCU. The ESP8266-based board I used for my prototype (the "NodeMCU") has 80KB of RAM. After compiling my code with a 5,000-byte buffer for storing the payload, I still had around 46.8KB of free RAM. So, with the provided Arduino sketch, you are limited to 5,000 characters (around 120 rows of sensors data). If you want to receive more data, you must allocate more bytes to the "payload_buf" buffer in the code. But to avoid buffer overflows, it is advisable to start your tests with less data in your database, say around 10-20 rows.

CONCLUSION

Building backends with PHP and MariaDB is a straightforward process once you understand the basics of attending HTTP request with PHP scripts and storing/retrieving data from the database. Up to this point, we used unsecure HTTP instead of secure HTTPS. Thus, because of security concerns, the examples shown here are only suitable for private Local Area

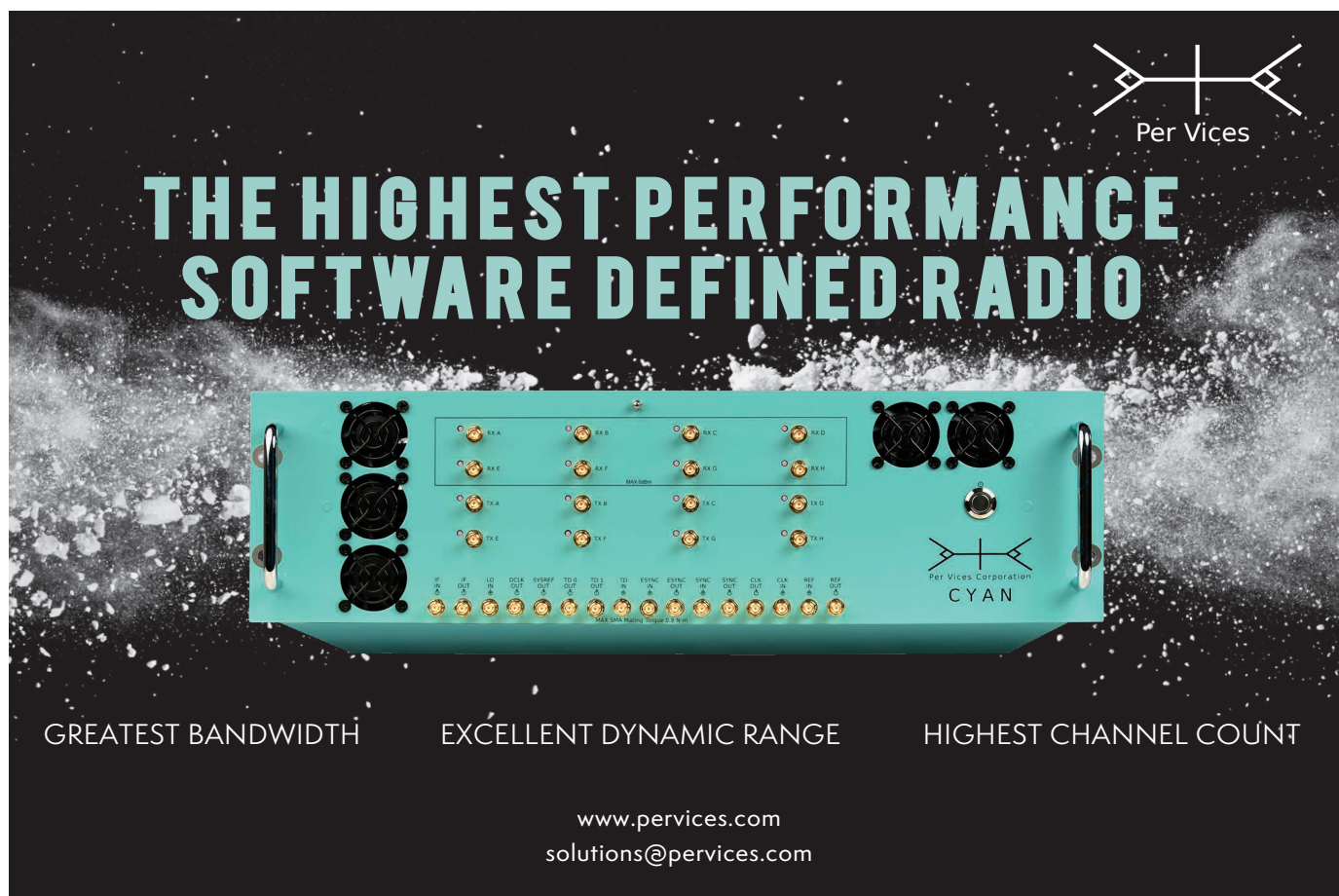
Networks. But adding a security layer on top of what we have done already is not difficult. There are HTTPS libraries for Arduino; and the Raspberry Pi server can be equipped with a "self-signed" SSL certificate to enable HTTPS traffic.

We also set aside concerns regarding one of the most infamous web security vulnerabilities, known as "SQL injection." This is also not very complicated to address in PHP, by following very simple guidelines when building the SQL queries.

Next month, in Part 3 of this article series, I will address some of these concerns, and also will explain how to get data from the server in JSON and XML formats. I will also briefly discuss some workflow guidelines for front-end web development regarding IoT data visualization on a web browser. 

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 site www.TecBolivia.com. You may contact him at raul@tecbolivia.com.



Per Vices

THE HIGHEST PERFORMANCE SOFTWARE DEFINED RADIO

Per Vices Corporation
CYAN

GREATEST BANDWIDTH EXCELLENT DYNAMIC RANGE HIGHEST CHANNEL COUNT

www.pervices.com
solutions@pervices.com

RPiano: A Playable MIDI Synthesizer

On a Raspberry Pi Microcontroller

FEATURES

By
Samiksha Hiranandani

Eager to explore the interface between music and electronics, and the digital representation of music, we created RPiano: a portable, playable MIDI synthesizer on a Raspberry Pi Pico (RP2040). We developed RPiano over the course of four weeks as our final project for Cornell University's course Digital Systems Design Using Microcontrollers. This article details our experience building RPiano.

RPiano, our digital adaptation of the traditional mechanical instrument, consists of a physical keyboard ranging over two octaves. The keys can be played like a regular piano, as shown in **Figure 1**. On pressing a key, our synthesizer produces the note sounds digitally, just like a traditional piano would mechanically. The key shape and size match exactly that of a traditional piano key, making the transition from a traditional piano to RPiano fairly smooth.

RPiano also has several built-in features that can be accessed by pressing the relevant buttons located just above the keys. Our current prototype has five stored songs, each with its own control button (**Figure 2**). Pressing the button for a song plays that song through a pair of attached speakers, digitally producing all the notes in the song, played in time to match the song's rhythm. Additionally, the prototype has three different instrument

modes (with corresponding buttons) that simulate the following three instruments: a grand piano, a harp, and bells. When a particular instrument mode is activated, the key presses on RPiano play the note with the tone of the instrument selected. Lastly, RPiano supports playing both the physical keyboard and a chosen pre-stored track simultaneously. This facilitates duets: the user can play one part on the keys while the in-built synthesizer plays the other.

While the buttons correspond to a few preselected songs, RPiano serves more broadly as a general-purpose synthesizer. It can synthesize any music file stored in the industry standard Musical Instrument Digital Interface (MIDI) format. Thus, RPiano's compatible with millions of existing files—anything from the latest pop hits to Mozart's timeless symphonies—with no additional processing. The user can easily change RPiano's set of songs by supplying MIDI file

paths for each song in the preferred set when compiling the software for the device.

The high-level structure of our project can be seen in **Figure 3**. There are three different types of user inputs: physical key presses on the keyboard, button presses to play a song, and button presses to switch instrument modes. Each of these modifies either the notes played or the kind of sound that is produced.

We used frequency modulation (FM) synthesis to synthesize the audio output and implemented the FM synthesis algorithm in software. The synthesis generates a final output wave using the set of notes to be played, and the kind of sound to be produced (piano, harp, or bells) based on user input. The output wave is sent to a pair of speakers and played out loud.

PROJECT HARDWARE

The project was built using the RP-2040 chip, the chosen microcontroller (MCU) for our course. Its high performance, low cost, and compact size made it ideal for our project. The other hardware components of the project include touch sensors to sense user inputs from buttons and touch-sensitive keys, as well as the hardware used for audio output (DAC, speakers). Our entire circuit schematic is shown in **Figure 4**. The key functions of specific components have been described in further detail in this section.

Touch sensing: To detect key presses, we used human conductance to utilize a similar effect to capacitive sensors. We placed a $1M\Omega$ pullup resistor on the input, and used a metal covering on the keys to make them conductive. When a grounded person touches the contact (metal key), they close the circuit with their body, which pulls the input pin low. By sensing the voltage on the input pin, we could detect whether the key was pressed (circuit complete) or unpressed. Various tests to measure voltage values revealed that the values remained consistently above 3V when there was no contact with the key, while the values ranged between 0.5V and 1.1V when the key was pressed. This left sufficient room for our voltage cutoff to be at 1.2V.

Key set-up and wiring: We made the keys conductive using aluminum foil to wrap the black keys, and copper tape for the white keys. We secured a long copper wire to each key such that each wire was in contact with the metal surface of the key. These wires were connected to the input sensing on the RP2040. We made all our connections on breadboards to ease prototyping, but they could be directly soldered for more reliable connections. We built the keyboard on a cardboard box, with the electronics inside.

Multiplexers: With only 28 GPIO pins on



FIGURE 1
RPiano model

the MCU, we could not attach each of the 29 keys to individual pins. We decided to multiplex the inputs from the keyboard to be able to detect presses on all the keys. We chose to use two 16x1 analog multiplexers. The key inputs were connected to the inputs of the two multiplexers. The multiplexers required 4 GPIO pins to select which of the 16 inputs should be passed through to the common output. The select inputs were varied through software to read each of the 16 inputs (16 keys on each multiplexer). We were able to reuse the same selector signal for both multiplexers. This system enabled us to



FIGURE 2
Keyboard and buttons

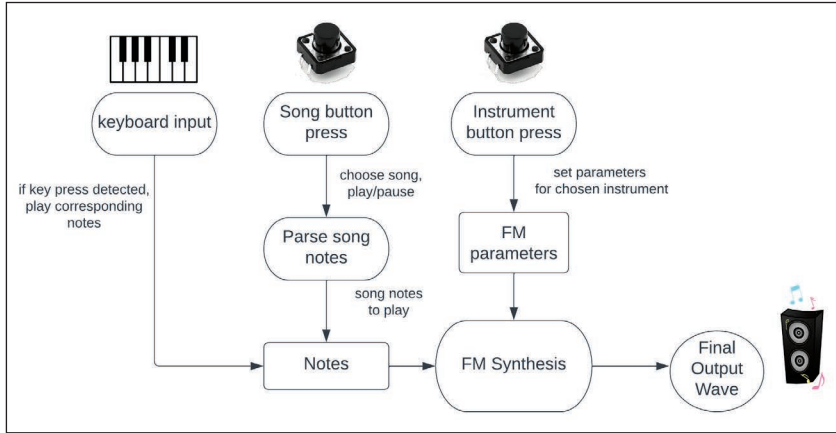


FIGURE 3
High-level overview of the project

utilize 29 analog inputs while using only four selector pins and two input pins, which freed up the other pins on the RP2040 to handle the other button inputs and the speaker output. This also allowed easy extensibility to a larger keyboard. With additional multiplexers, the sensing capabilities can be expanded to 64 keys with only two additional analog input pins.

Digital-to-analog converter: We used an MCP4802 digital-to-analog converter (DAC) to send our output signal from the RP2040 to a set of speakers. The DAC uses the Serial Peripheral Interface (SPI) to take digital input

from the RP2040 and convert it to an analog signal. This allows us to control a speaker from the RP2040.

Speakers: We used standard desktop speakers for output. It was important that the speakers have their own power source since the RP2040 and DAC are incapable of providing the current required to play music at our desired volume.

PROJECT SOFTWARE

On a high level, our software includes three primary components, and is written in C. The block diagram in Figure 3 shows, at a high level, how the different software components fit together.

- Part 1 is the FM synthesis algorithm to compute the wave output written to the DAC to send to the speakers.
- Part 2 comprises the user input detection, detecting piano key presses, instrument mode button presses, and song button presses.
- Part 3 is the software for playing songs. It handles the song notes to be pressed and released based on stored metadata for a selected song.

Another piece of software, separate from

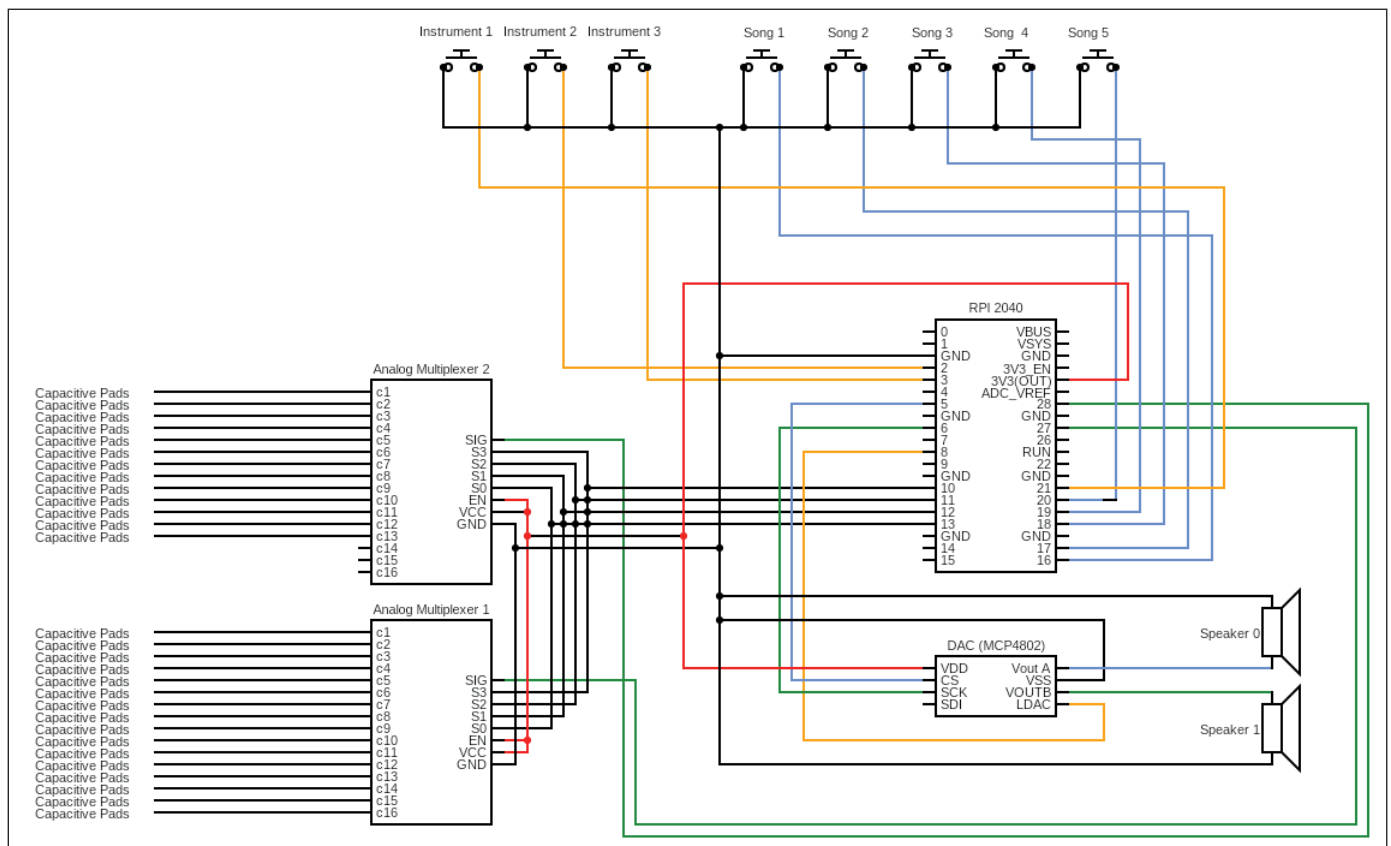


FIGURE 4
Circuit diagram for the project

Congratulations
Circuit Cellar,
on Your
400th Issue!

Saelig
UNIQUE ELECTRONICS
888-7-SAELIG



pico[®]
Technology

PicoScope[®] 2000 Series

Ultra-compact range of 8-bit oscilloscopes and mixed-signal oscilloscopes (MSO). 2000B models offer more memory and bandwidth. All models are USB-powered and have a built-in function generator and AWG.



PicoScope[®] 3000 Series

General-purpose 8-bit oscilloscopes and mixed-signal oscilloscopes (MSO) that combine fast sampling rates with class-leading deep buffer memories. All models have a built-in function generator and AWG.



PicoScope[®] 4000 Series

High-resolution oscilloscopes with 12 to 16-bit resolution. Low noise and distortion provide unmatched signal fidelity. All are USB-powered and most include an AWG. Series includes differential-input models.



PicoScope[®] 5000 Series

Flexible Resolution Oscilloscopes. Breakthrough ADC technology allows a range of hardware resolutions from 8 to 16 bits. Combines the high sampling rate of the PicoScope 3000 Series with the high resolution of the PicoScope 4000 Series.



PicoScope[®] 6000 Series

High-performance oscilloscopes with up to 1 GHz bandwidth, 8 or 8-12 bit flexible resolution and ultra-deep capture memory that delivers 200 ms capture duration at maximum sample rate of 5 Gs/s. Optional MSO pods add up to 16 digital channels



PicoScope[®] 9000 Series

The unique PicoScope SXRTOs and sampling Oscilloscopes for data eye diagram, speed and jitter analysis out to 16 Gb/s. 9.5 GHz optical, clock recovery and differential TDR/TDT options.



the code running on the MCU, is a Python script to parse a chosen MIDI file, and store it in the required format in program memory of the program running on the MCU.

Our implementation is split into several threads, referred to as “protothreads,” using the protothreads library, a light-weight, stack-less, threading library written entirely in C [1].

SYNTHESIS THEORY

We chose to digitally produce the sounds, using FM synthesis to compute amplitude values for a note at a particular frequency, and using additive synthesis to combine notes at different frequencies into a single output waveform.

FM synthesis: FM synthesis is a method of sound synthesis that involves modulating a waveform using another waveform. Two waveforms are generated, and one is used to modulate the other, as shown in **Figure 5**.

The two waveforms are controlled by a logic structure that sets the value of each waveform at every time point. The value is based on how long the note has been played and the relevant attack, sustain, and decay parameters. At each time step the modulating waveform is calculated first, and then its amplitude is used to determine how far to step the main waveform along a precalculated sine table. This causes the main wave to progress through the sine table at different speeds based on the value of the modulating waveform. This modulated frequency can simulate many

instruments better than the single pitch that the basic synthesis algorithm generates [2].

Additive synthesis: For notes played at the same time (such as a chord), we used the principle of additive synthesis to add together all the amplitude values to create a sound comprising all the frequencies. This is simple, and only requires that at each time step we sum the amplitudes of every note that is playing. We then divide by the number of notes playing to normalize the volume. Without normalizing, the output signal could spike in volume when notes are pressed or released.

SYNTHESIS IMPLEMENTATION

The core FM synthesis is done in an interrupt service routine (ISR), computing values for the final wave output that is sent to the speakers through the DAC. Our implementation for the FM synthesis builds on an example by Bruce Land at Cornell University [2].

The wave output values for producing the sound for a particular note cannot be precomputed and pre-stored for each note. This is because the output frequency at which values are written to the DAC needs to be high to achieve reliable (not distorted) sound output, and it would require too much memory to store thousands of samples for each note frequency. Thus, the output values are computed in real time. These values are written to the DAC each time the ISR executes. To write the DAC at the high frequency required for good sound quality, it is essential that the computation is complete before a new value needs to be written. Through experimentation, we arrived at an optimal time interval value of 36 microseconds—large enough to leave sufficient time for completing required computations but small enough that the sound output is smooth and pleasant to the human ear without any distortions. This corresponds to an output sampling frequency value of approximately 27.7kHz. Frequencies lower than this gave a distorted output, leading to sounds that were not smooth, while at frequencies higher than this, with a shorter interval the computation was not completed in time.

The efficiency of our design depended heavily on optimizing the ISR to be as fast as possible, ensuring that a new DAC value was ready every 36 microseconds, without running out of time in the ISR before the next value needed to be completed. To play any MIDI file, our implementation needed to support playing any note in the entire range of the piano (88 keys). Additionally, whether the note is pressed (and needs to be included in the output wave) is controlled by an external input choice (physical keys, or choice of song),

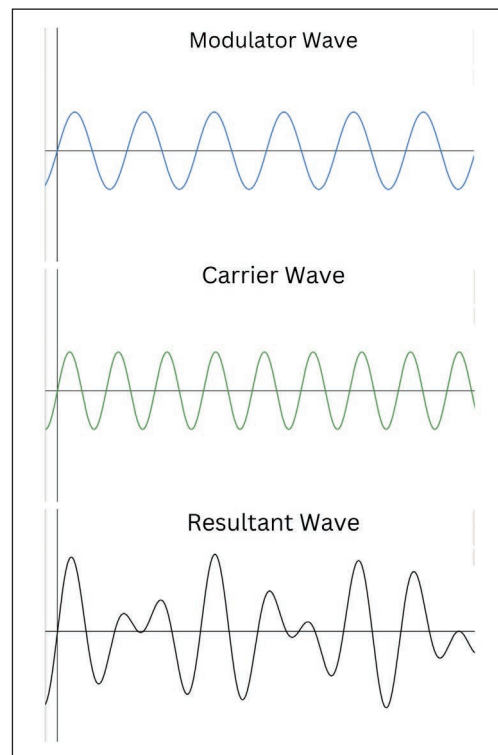


FIGURE 5
Waveform illustrating frequency modulation synthesis

and constantly updated by external threads. Checking all 88 keys in the ISR to see if they were pressed and then computing the required waveform for the frequency corresponding to the key took up too much time and led to the ISR running out of time before completion. This produced distorted sounds.

We approached this problem by adding a buffer for notes that could hold 10 unique notes at a time (corresponding to 10 fingers on the piano). This way, each time a note needs to be played (either on detecting a physical key press on the keyboard or a note play event in the song), its note number is added to the buffer. The ISR now only loops through the 10 notes in the buffer, checks whether they are pressed or not, and then includes them in the synthesis computation. The threads handling user input add to this buffer read by the ISR, as depicted in **Figure 6**.

There is also another FM synthesis control thread that sequences the synthesis ISR, precomputing fixed point constants to make the ISR faster. The buffer is implemented using an array, with each element in the array contained in order sorted with respect to when it was added to the buffer. When the buffer is full and a new note needs to be added, the key that was least recently played is removed from the tail end and the new key is added at the front.

The size of the buffer is a configurable parameter that can be changed based on the number of unique voices needed to be played at the same time. The tradeoff of making the buffer too large would be, however, a lower sampling frequency due to the increase in computation time in the ISR to handle computations for a larger number of notes at the same time.

USER INPUT DETECTION

A separate protothread handles physical key press detections. For a total of 29 keys on the physical keyboard, we use two 16-input multiplexers, each connected to two separate ADC input pins. MCU functions are used to read the input values, and switch between reading the two ADC inputs.

The value read from the ADC is converted to a voltage value by multiplying the read input value by a conversion factor defined through experimentation. We defined a voltage cutoff constant, and the press was detected by checking if the resultant value read was lower than the specified voltage cutoff value. Through experimentation with our physical setup, we determined a value of 1.2V for the voltage cutoff.

If the key press was detected, the note corresponding to the key was set to play. To prevent detecting a single key press twice, we

also stored a previously pressed Boolean value, and the note press was only set if the key was detected to be pressed and was previously not pressed. Additionally, the current state of the key was stored, so that it would be considered “released” in the synthesis computation only when the finger was lifted. This enabled us to store information so that the produced sound’s length was based on how long the key was pressed.

BUTTON PRESSES

Appropriately, a thread called “button press” detects button presses. This thread checks to see if each of the five buttons corresponding to the songs has been pressed by reading the GPIO pin of the button. When a press is detected, the song data corresponding to the chosen song is loaded into a global variable. It also enables pausing and playing the song if it’s currently playing or not playing, respectively.

Instrument button presses are detected in a similar manner. If pressed, the parameters of the FM synthesis are changed to be the values tuned for the instrument corresponding to the button. These parameters are accessed by the ISR for synthesis, generating modified sounds based on the parameter changes.

MIDI FILE REPRESENTATION AND PARSING

MIDI files are the industry standard for passing musical performance information among electronic musical instruments and computers. Unlike an MP3 or WAV file, it does not contain real audio data, but instead, the

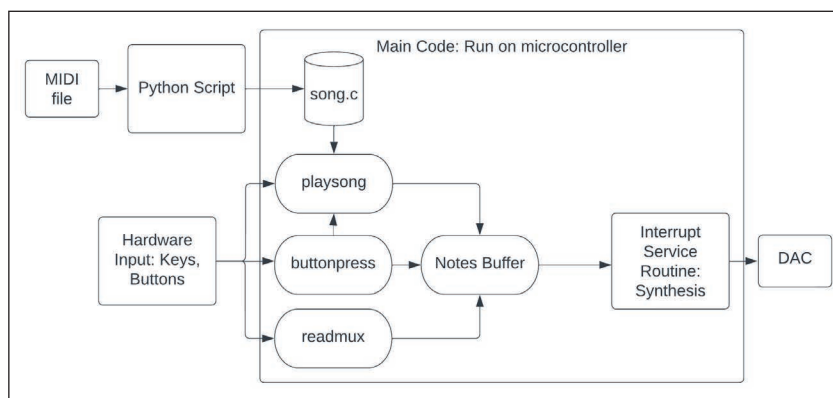


FIGURE 6
Software high-level overview

ABOUT THE AUTHORS

Samiksha Hiranandani (snh44@cornell.edu) is an undergraduate senior at Cornell University studying Computer Science with an external specialization in Electrical and Computer Engineering. She is excited by the integration of software and hardware for engineering solutions.

notes played, their timing, duration, and desired loudness, in sequence. Since it does not store audio data, it is much smaller in size than an alternative MP3 file, so it's ideal for our project with limited data storage. It's also compatible with different instruments—it needs only to play the frequency corresponding to a given note on the chosen instrument. Further, MIDI files make it easy to change tempo based on the user's preference. Each MIDI note number is mapped to a particular frequency that corresponds to a note. For example, MIDI note number 60 corresponds to middle C on the piano (C4). We used Equation 1 to map MIDI note numbers to a frequency used for FM synthesis:

$$f = 440 \cdot 2^{(n-69)/12}$$

The MIDI format consists of a list of events, such as "KeyOn" or "KeyOff," that correspond to note activation and release on a keyboard. We chose to use the Mido library in Python to parse this information [3]. We wrote a script to read any MIDI file and store the required data to play the song on our synthesizer. The script takes in a MIDI file as input, prompts the user to choose a track contained within the MIDI file, and then parses a sequence of MIDI events corresponding to the track selected.

We chose to represent each MIDI event with three fields: the note to press, the note to release, and a hold time (the time to wait before performing this event). The hold time stored is a relative value and is converted to a time in milliseconds by multiplying by a constant conversion factor. Storing this time-based information enabled us to store very concisely enough data to reproduce the song's exact rhythm and playing style (adhering to different elements of music like rests), note values (how long each note is played), as well as the pitch (the frequency of the note).

After reading all events, the script accumulates a list of events, writing this in the form of data to be stored in program memory. On detecting a song play button, the code iterates through each event in the song data corresponding to the chosen song. Before

each event, the hold time in milliseconds is calculated from the relative value stored. This is done by multiplying a delay tick value, 1000ms, by a constant conversion factor for the song. This conversion factor for the song can be changed to speed it up or slow it down.

PERFORMANCE

Over four weeks, we were able to create a playable keyboard that successfully detects key touches and plays the required note. We were also able to play any readily available MIDI file on our synthesizer, making use of the entire range of the piano (88 keys), and handling songs that contain a wide range of notes and different, complicated rhythms. The pieces, when played on our synthesizer, closely modeled the sound of a real piano, and exactly replicated the rhythm and pitch specified in the MIDI file. Circuit Cellar's Article Materials and Resources webpage contains a link to a video of RPiano in action [4].

FUTURE WORK

Overall, our design meets our expectations. In some areas, it even exceeded our expectations—we did not expect to be able to handle more complicated songs with many chords and quick notes smoothly. In terms of the physical design, while wiring the keys with tape and using the breadboard was a quick solution that worked smoothly for the most part, in certain cases a key press would not be registered while testing due to a wire slipping. Soldering the wires onto the metal for the keys and onto a board would improve this issue, and would make the design more foolproof and durable.

There are also multiple extensions that we'd planned as stretch goals that we could implement in addition to the existing functionality. For the songs, we did not use the volume information encoded in the MIDI file since we wanted to be consistent with volume across songs and the keyboard. The code could be altered to include changes in loudness on the keyboard, as well as effects like *piano* and *forte* in sheet music. Another possible extension is to add the three pedals to the piano that create the sustain effect—we could modify the FM synthesis parameters when the pedals are pressed. Finally, we could also enable the user to change the FM synthesis parameters through a bar using a potentiometer. This would allow users to dynamically alter the kinds of sounds produced, rather than just having three different modes.

Acknowledgments

I would like to acknowledge Ben Manninen, a student at Cornell University, who worked with me on this project. 

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
 References [1] to [4] as marked in the article can be found there.

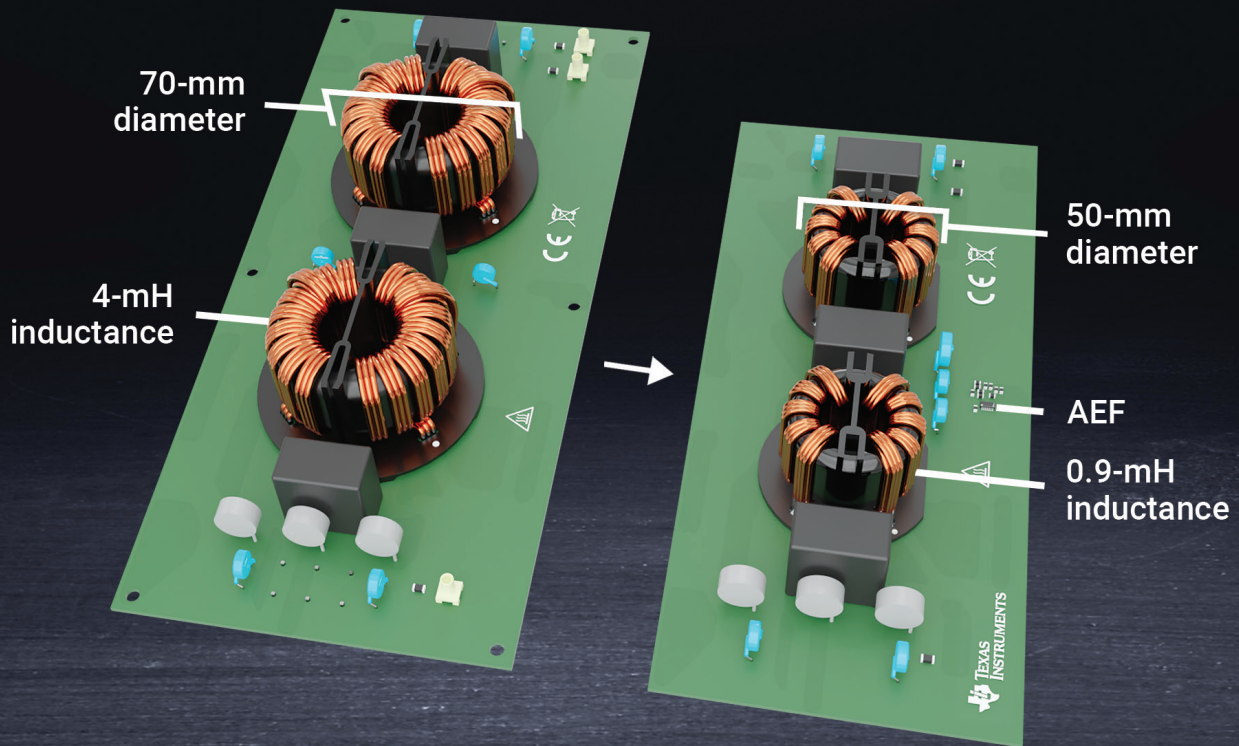
RESOURCES

Microchip Technology | www.microchip.com

Raspberry Pi | www.raspberrypi.com

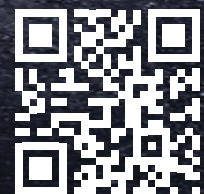
Texas Instruments | www.ti.com

Meet stringent EMI requirements and cut choke size in half.



Our new portfolio of stand-alone active EMI filter (AEF) ICs help designers meet stringent EMI requirements while reducing system size, weight and cost for single- and three-phase AC/DC systems. The TPSF12C1/-Q1 and TPSF12C3/-Q1 allow engineers to shrink the value of common-mode chokes by up to 80% inductance, resulting in over 50% smaller size compared to purely passive filter solutions. Meet your EMI performance standards and increase power density with AEF ICs today.

▶ Learn more > www.ti.com/AEF



Embedded Displays

It's All About Timing

By
Michael Lynes

TECH FEATURE

Timing. For those of us who have had a long association with engineers, or for that matter may in fact be engineers themselves, timing is a very interesting word. Let me explain.

Most people react to the subject of timing in either a negative or at best ambivalent way. For instance, if you say to a person, "Your timing is off," or "The timing is not good for me," their perception will be that there is a problem that needs to be addressed. You see this a lot in business or social situations where people will negotiate a time for a meeting or try to tailor a particular action to best fit in with their other activities. Time management is a skill, and whole industries have been dedicated to the administration of personal and professional time, with scheduling assistants, calendar apps, and cloud-based day-timers.

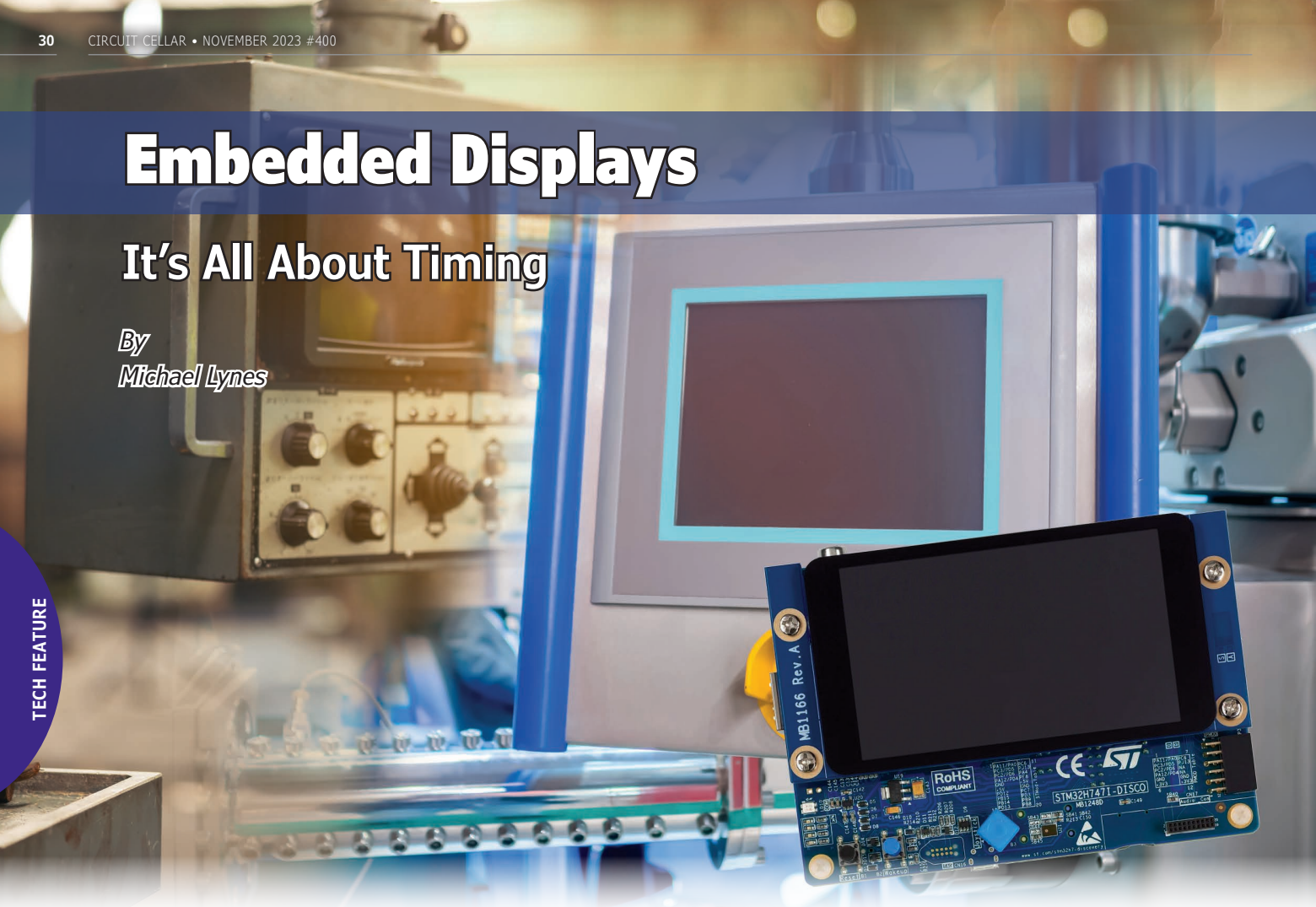
On the personal front, much of our early life experience from the time we enter preschool is about learning the proper timing for various types of interactions. Children are drilled and trained in how to behave, and a lot of the basic instruction will center around the timing of a specific action or behavior—when it is appropriate, and when it may not be.

With all the above said, for engineers, the word timing has a very different meaning.

That is not to say that the prior two meanings are invalid. But one of the charming, some might say challenging, aspects of our sort of folk is our strange fascination with time.

Take the example of social timing given above. Based on my experience, when an engineer finds themselves in a room full of non-engineers, normal people if you will, there is a bit of a disconnect. The conversations in the room will likely revolve around everyday subjects: the weather, sports teams, politics, and so on. This is not to say that engineers don't have concern for these types of topics, but our take on them is different.

The weather is a good example. If an engineer happens to be present when the subject of the weather crops up, you will observe one of two responses. The first and most common will be some form of non-committal social noise. "Hmm," is one of my favorites, as is "Ah...yes," said with a polite nod. The intent is to signal to the speaker that their speech has been received and understood, and that further speech is encouraged. I liken it to a low-level social subroutine, a near-autonomic function designed to passively ignore banal discourse while not causing distress on the part of the speaker.



As a past master of this technique, I sometimes play an internal game to see just how long I can respond in this manner, employing a series of utterances comprised of encouraging word fragments with zero semantic content, while the person I'm interacting with remains unaware that I've checked out of the present moment and instead am engaged in mentally running through a problem from my latest embedded firmware project—say a multi-level finite state machine to be oddly specific, trying to determine the best way to handle the timing of reentrant variable concurrency.

To be clear, I'm not doing this to be disrespectful, and I am in fact listening at some level. The structure of my pseudorandom response subroutine has a built-in priority interrupt that summons my full awareness if the conversation turns to a subject that I am actually interested in, or if the opportunity arises to deftly redirect the flow into a more productive channel of discourse with a well-placed joke or pun. The purpose served is in fact a form of time scheduling, specifically to optimize the conservation of personal time. Engineers love efficiency, getting the most value out of every waking hour. Note that we are back to the subject of timing again, employed as both a defensive shield and a weapon with offensive capability.

The second form of response is triggered if the banal discourse happens to touch on a subject that I find appealing. The unfortunate normal being whose innocent remark sparks this reaction might feel their eyes widen in horror as I launch into near-eidetic recall of everything I've read on the topic. Returning to the example of a weather-related conversation, their offhand comment might bring to mind a recent article I'd consumed on the effects of stratospheric super-heating caused by high-energy particles from solar ejecta. Without the slightest concern for the esoteric nature of the subject or their potential lack of interest, a torrent of words spill forth describing how the charged particles slam into the Earth's outer atmosphere and subsequently influence long-range weather patterns. In this case, my timing could not be worse. Nevertheless, the unfortunate listener will be subjected to a minutes-long diatribe consisting of a highly technical and detailed exposition of my thoughts on the matter. And I will continue regardless of the victim's obvious discomfort or the glazed look in their eyes, the fire-hose flow of information only ceasing if my significant other happens to be near enough to dig a well-placed elbow in my ribs. Her superior timing may be able to save me from myself, but in most cases, the damage has been done. The conversational



FIGURE 1
OGEE VT-100 Display

buzz may resume, but unless another engineer is present, my participation in it will be severely curtailed.

THERE'S ALWAYS TIME TO DO IT OVER...

As the old saying goes, "There's never time to do it right, but there's always time to do it over." And, speaking of do-overs, the subject of this month's Technology Feature, embedded displays, probably sounds familiar to regular readers of this column. In fact, it was only a short time ago—July of this year—that we spent a good deal of time exploring the capabilities of embedded displays from the perspective of digital signage ("Digital Signage," *Circuit Cellar* 396, July 2023) [1]. However, this topic is as broad as the mighty ocean that the Pequod set sail upon, and as deep as the depths to which the white whale himself might dive. So in this month's issue, we are going back into the belly of the beast, so to speak, to look at embedded displays once more.

But, before we delve into the hoary digital guts of modern display technology, let's take a stroll down memory lane and talk a bit about Old Guy Electrical Engineer (OGEE) displays. Back in the ancient days of yore, displays were huge, hot, noisy boxes that consumed a lot of power and precious desk space. As mentioned in another Tech Feature article, the ADM-3A, or perhaps the beast shown in **Figure 1**, would often be your working interface to the DEC VAX 11/780 or PDP-11 minicomputer that



FIGURE 2
DEC VAX 11/780

your current project was cross-compiling on **(Figure 2)**. Fun fact: All modern 102-style keyboards owe their shape and function to the original VT-100 keyboard seen in Figure 1, and the VT-100 screen cursor positioning commands still work to this day.

These displays were not much more than overgrown oscilloscopes with clunky keyboards mounted in front of the tube. The basic technology was comprised of a high voltage cathode ray tube (CRT), enhanced with control circuits driven by a small processor and some RAM memory. CRT itself was an even more ancient display technology that used an electron gun that produced a narrow beam of electrons, and deflection plates that would guide the beam in a scan pattern across the surface of a glass tube coated with phosphor. The beam pattern was a row, or “raster,” scan, and the speed at which the scan would complete one sweep of the entire display area was called the refresh rate.

These displays were able to render images by taking advantage of the human eye’s persistence of vision, the effect that you can most easily perceive when you stare at a bright light for too long and then look away, preferably at a blank sheet of paper. An afterimage of the bright light will appear, seeming to float above the real image of the blank sheet. The human eye is a miraculous device, consisting of an organic light-focusing mechanism and an opto-neuro-chemical interface that we call the retina. The phantom image you see is a side effect of the way your eyes perceive images. Specialized cells of the retina undergo a chemical change in response to various frequencies of light. This causes electrical impulses to be sent along the optic nerve, and ultimately to the visual cortex of the brain for processing. The chemicals that are employed have a response time measured in the millisecond range and can become exhausted by intense exposure to light. The refreshing of these cell chemicals takes time, and has a relatively long hysteresis effect, meaning that there is a period—the cycle time—during which they cannot properly convert the received light into the correct impulses. This effect is perceived as an “afterimage,” or phantom image, in your vision.

This tendency of the eye to preserve an image in this way allows display technologies like movies and television to create the illusion of motion by updating the picture at a higher rate than the eye can perceive. In this case, a complete picture is projected onto a screen. The eye sees the picture for a moment, and then a new picture, the next frame, is projected onto the same spot. For obvious reasons, the alignment of these pictures must

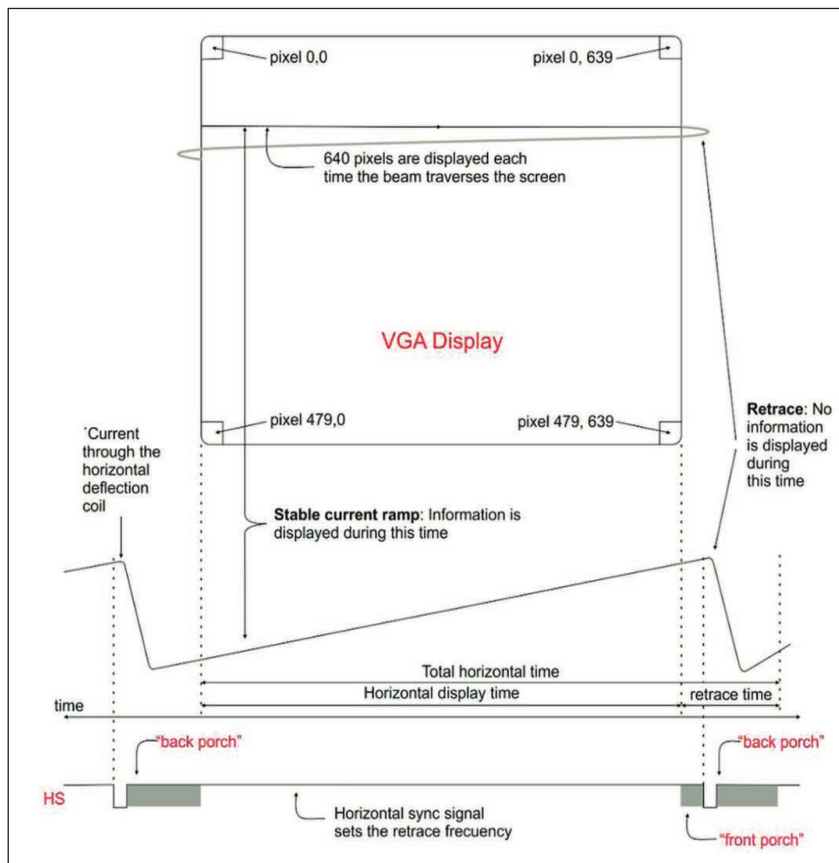


FIGURE 3
CRT raster scan timing

be precise. The optimal picture change rate is something around sixteen frames per second. Once you achieve this rate or higher, the eye no longer sees individual images or jerky motion, but rather the illusion of smooth animation.

So, how is this accomplished with a single dot illuminated by an electron beam you might ask? Well, it's basically explained in the timing diagram in **Figure 3**. As you can see, the screen is divided into a number of elements—let's call them picture elements, or pix-els for short. There are 640 pixels on one line of the screen. Each one is the size of our electron beam's focal dot. Their size is also affected by the grain size and type of phosphor, but let's leave that nit for another discussion. The timing of the scan—and I can almost feel your increase in interest as we return to our favorite subject—is the important part.

Each raster is scanned by the horizontal deflection of the beam. You can see this depicted toward the bottom of Figure 3, which shows the slope of the horizontal plate deflection coil current. As the slope increases, there is greater current and more deflection. The pixels themselves are turned on or off by the activation of the electron beam itself, which is in turn modulated by the values in

the display buffer. The display buffer is a RAM memory in which each "bit" is either a one or a zero, corresponding to the gun being on or off. Turning the gun on excites a pixel, and off allows that pixel's state to remain unexcited. There is a decay rate of excitation, an afterglow if you will, that we depend on to see the image on the screen.

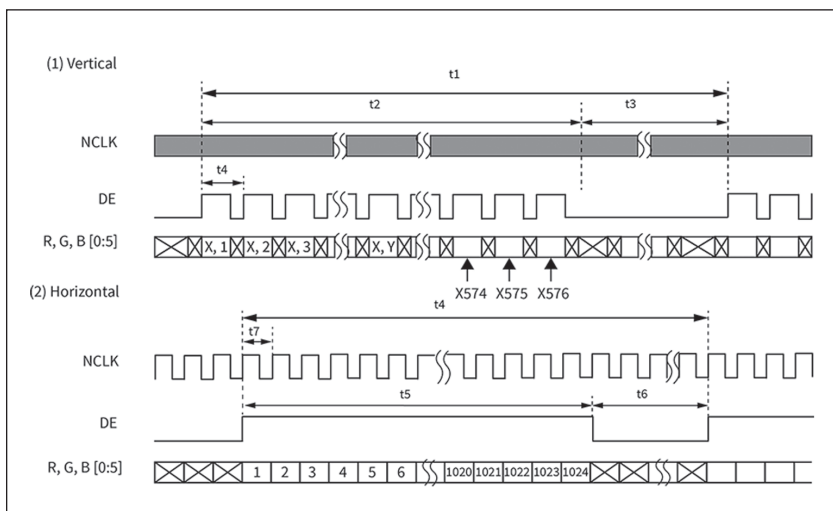


FIGURE 4 Full color LCD display update timing

TECH FEATURE

ST's high-accuracy L9961 Battery Management System controller makes lithium batteries perform better and last longer by providing monitoring, balancing, and protection for industrial applications as cordless power tools, energy-storage systems, portable equipment, and more.

ST
life.augmented

L9961

www.st.com

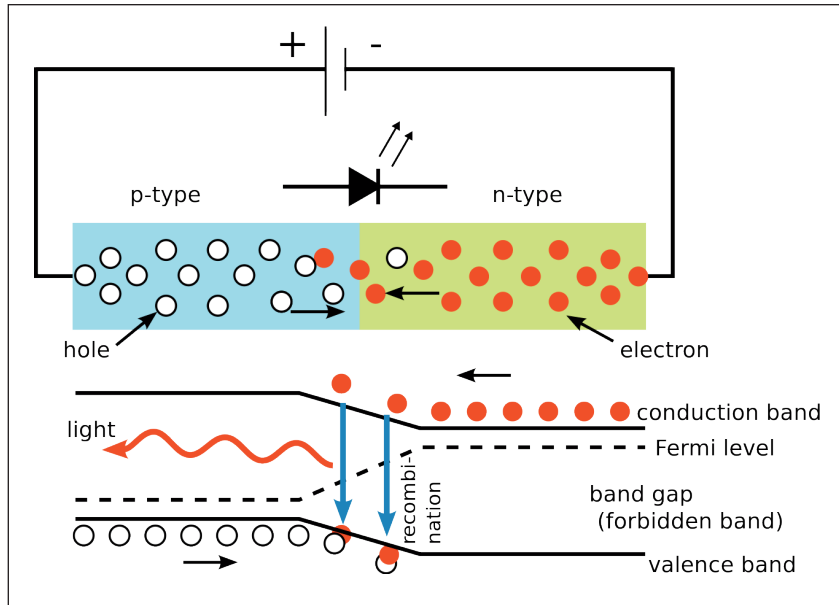


FIGURE 5
LED physics

As you can further glean from the timing diagram, once the scan completes a row, it then goes into the retrace area. During this time, we reset the horizontal deflection to the left-hand side of the screen, and, importantly, we increase the vertical deflection such that we are now going to trace our beam across the next row of our display. Note that our RAM, or display buffer, must have a row of bits for each horizontal sweep and a depth of bits that corresponds to the number of vertical scans. This display is a VGA, 640x480, which means that we can render an image that is, at the most, 640 pixels wide by 480 pixels high. The top left is pixel 0,0 and the bottom right 639,479 (max row, max column). This addressing scheme is still used to this day, and it's a legacy of this old raster scan tech.

Note that we are back to timing, once again. Most of the magic in this diagram happens during the retrace sync and the vertical deflection update—that is to say, while your human information processor is still receiving and assembling the prior frame. During this period, the RAM of the frame buffer is read to allow the next row of dots to be displayed. The entire frame buffer can also be updated—all the rows once per vertical sync pulse, or line by line after each horizontal sync pulse. This understanding of timing is what allows

the display to function efficiently, and to engineers it is both beautiful and crucial.

EMBEDDED DISPLAYS

Now, with that mischief managed, let's move on to our actual topic: modern embedded displays. The critical thing to note here is that all the prior art that was developed for CRT displays—the moving picture frames, the scan, the refresh rates, the colors (we didn't even touch on colors), and so on—are all required for an embedded display to be an effective information transmitter. Humans are still the target audience, so we can still take advantage of our unique visual processing limits. But the technology and the timing that makes embedded displays work is very different.

As you can see in **Figure 4**, LCD-type displays allow both horizontal and vertical read and write access to each individual screen element. In this case, all the updates are digital, and the N-clock is used in place of the high-speed analog raster scan, allowing a full row of display elements, at whatever color bit depth your individual display supports, to be clocked in from the display buffer.

The vertical write update clock timing is slower, but still needs to accomplish what CRT hardware did with analog electronics. Namely, it must refresh the entire screen's worth of bits (AKA the frame) at the specified refresh rate. The LCD can also be read, which means that you can capture elements of the screen by reading the information out of the display itself, or the corresponding frame buffer as you prefer. With modern screens supporting millions of colors, 1080p resolution, and frame rates of 120 frames per second (fps), we can see that the timing will become both a critical and precise part of our design.

THE PHYSICS

Something I learned that I had not known when researching for this article was a bit about the physics involved in creating modern AMOLED displays [2]. As shown in **Figure 5**, the light produced by a light-emitting diode is due to the electroluminescent effect. As current flows across the semiconducting band gap, energy is lost. In standard indirect band gap diodes, this energy is converted into heat. In LEDs, a direct band gap is used, and as

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

References [1] to [10] as marked in the article can be found there.

RESOURCES

Avnet Embedded | embedded.avnet.com

Crystalfontz | www.crystalfontz.com

MIKROE | www.mikroe.com

STMicroelectronics | www.st.com

the gap is traversed, the lost energy produces photons in a characteristic wavelength, which is directly related to the band gap energy of the materials forming the p-n junction. If the casing that covers the junction is transparent, this light will be able to escape and be seen. The color or frequency of the light can range from infrared to ultraviolet, and everything in between (**Figure 6**). Some of that range is in the visible spectrum, and LEDs are chosen to produce the characteristic primary colors (RGB) or secondary colors (CMYK). Producing a specific color consists of activating combinations of these LED dots and varying their intensity to produce one of the millions of colors we can render on modern displays. I found the science to be fascinating, and if you do as well, check out *Circuit Cellar's* Article Materials and Resources web page for a link to an article detailing the incredible manufacturing processes used to create these dense LED arrays [3].

THREE TIMES IS THE CHARM...

A word about criteria: Embedded displays, unlike the digital signage I wrote about in July [1], have some restrictive requirements. A good summary of these requirements can be found on the Predictable Designs website [4]. One of the most important limits is power, in that an embedded project often needs to be configured to run on batteries. Embedded displays must also be designed to give a lot of bang for the buck—high-definition resolution, fast update times, full-motion color, and can be efficiently driven by smaller processors that aren't dedicated to video processing alone. They also have to be small, and, in the same way that it's harder to write a great short story than a novel, this alone is a significant challenge. Last, they must be inexpensive, as the budget for an embedded device can't be dedicated to the display technology alone.

It all seems like a tall order. Luckily there are a lot of manufacturers that have stepped up to supply this need. So, without further, *further* ado, let's look at some of the best examples of embedded displays available on the market today.

MIKROE: MIKROE, founded in 2001 with headquarters in Belgrade, has thousands of embedded products designed with both the industrial IoT market and the hobbyist in mind [5]. It has a full line of embedded displays, and its thin film transistor (TFT) product line (**Figure 7**) features full-color capacitive touchscreens in a wide variety of form factors [6]. Prices range from \$26 for a 4.3" display to approximately \$90 for a 7" diagonal model. Color spectrums are wide and deep, and MIKROE offers comprehensive documentation and support for all its products.

Color	Wavelength [nm]	Voltage drop [ΔV]	Semiconductor material
Infrared	$\lambda > 760$	$\Delta V < 1.63$	Gallium arsenide (GaAs) Aluminium gallium arsenide (AlGaAs)
Red	$610 < \lambda < 760$	$1.63 < \Delta V < 2.03$	Aluminium gallium arsenide (AlGaAs) Gallium arsenide phosphide (GaAsP) Aluminium gallium indium phosphide (AlGaInP) Gallium(III) phosphide (GaP)
Orange	$590 < \lambda < 610$	$2.03 < \Delta V < 2.10$	Gallium arsenide phosphide (GaAsP) Aluminium gallium indium phosphide (AlGaInP) Gallium(III) phosphide (GaP)
Yellow	$570 < \lambda < 590$	$2.10 < \Delta V < 2.18$	Gallium arsenide phosphide (GaAsP) Aluminium gallium indium phosphide (AlGaInP) Gallium(III) phosphide (GaP)
Green	$500 < \lambda < 570$	$1.9^{[22]} < \Delta V < 4.0$	Traditional green: Gallium(III) phosphide (GaP) Aluminium gallium indium phosphide (AlGaInP) Aluminium gallium phosphide (AlGaP) Pure green: Indium gallium nitride (InGaN) / Gallium(III) nitride (GaN)
Blue	$450 < \lambda < 500$	$2.48 < \Delta V < 3.7$	Zinc selenide (ZnSe) Indium gallium nitride (InGaN) Synthetic sapphire, Silicon carbide (SiC) as substrate with or without epitaxy, Silicon (Si) as substrate—under development (epitaxy on silicon is hard to control)
Violet	$400 < \lambda < 450$	$2.76 < \Delta V < 4.0$	Indium gallium nitride (InGaN)
Ultraviolet	$\lambda < 400$	$3 < \Delta V < 4.1$	Indium gallium nitride (InGaN) (385-400 nm) Diamond (235 nm) ^[23] Boron nitride (215 nm) ^{[24][25]} Aluminium nitride (AlN) (210 nm) ^[26] Aluminium gallium nitride (AlGaInN) Aluminium gallium indium nitride (AlGaInN)—down to 210 nm ^[27]
Pink	Multiple types	$\Delta V \approx 3.3^{[28]}$	Blue with one or two phosphor layers, yellow with red, orange or pink phosphor added afterwards, white with pink plastic, or white phosphors with pink pigment or dye over top. ^[29]
Purple	Multiple types	$2.48 < \Delta V < 3.7$	Dual blue/red LEDs, blue with red phosphor, or white with purple plastic
White	Broad spectrum	$2.8 < \Delta V < 4.2$	Cool / Pure White: Blue/UV diode with yellow phosphor Warm White: Blue diode with orange phosphor

FIGURE 6
PN junction colors



FIGURE 7
MIKROE 2168 TFT

TECH FEATURE



FIGURE 8
Crystalfontz TFT resistive touchscreen

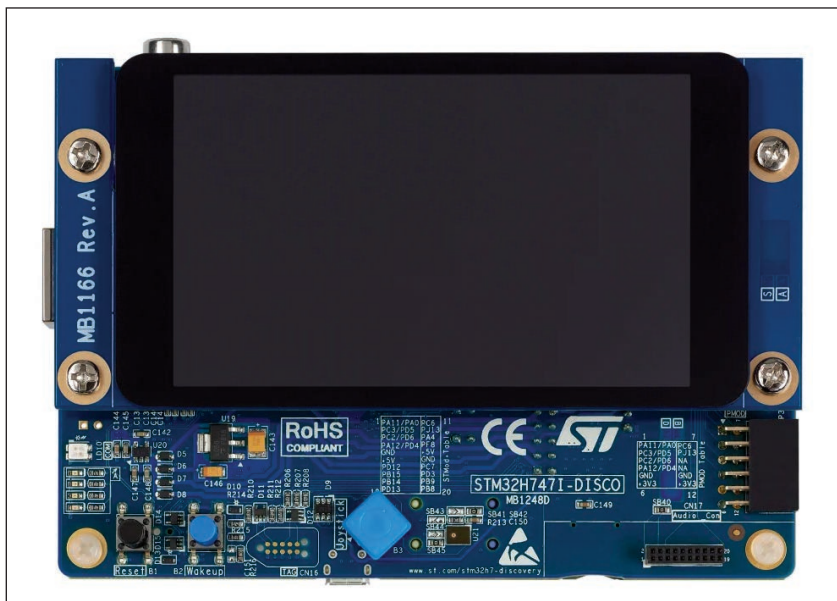


FIGURE 9
STMicroelectronics Disco Kit

Crystalfontz: Crystalfontz is a manufacturer of all sorts of displays, from monochrome transparent OLED (TOLED), to full-color AMOLED, TFT, and ePaper. Its line of resistive touchscreens features enhanced readability in direct sunlight and has 800x400 resolution in a 5" diagonal package (**Figure 8**) [7]. The company offers discounted volume pricing, 16 million colors, and wide-angle viewing thanks to their IPS display technology. Its resistive touch is optimized for more rugged environments where the need to have thick protective coverings is crucial. Screen resolutions range from 80x160 to 1280x800 pixels, and low-power options are available.

STMicroelectronics: ST has a range of Cortex ARM-based Discovery kits with high-resolution displays built in. These kits, such as the STM32H747 (**Figure 9**), feature support for a wide variety of OS and IDE environments, and they support ST's embedded display controller technology, allowing full motion at high frame rates with little to no "tearing" issues due to their ability to sync updates with the screen refresh [8].

A perfect example of timing to the rescue, ST offers the kit through many vendors, with the expected excellent support that ST is known for throughout the industry. Once you've developed your application, it's an easy translation to a wide variety of supported displays. And, as I learned from the company-sponsored webinar on deep learning [9], their TouchGFX framework makes porting a breeze.

Avnet Embedded: Avnet Embedded has deep experience in the embedded environment [10]. It's a division of Avnet GmbH, founded over 100 years ago in Freiburg, Germany. The original Avnet began serving customers as a distributor in New York City in 1921, eventually growing to a huge multinational with offices around the world. The Avnet Embedded division was created in 2001, originally as Avnet EMG, and went through several iterations over the next couple of decades before relaunching as Avnet Embedded in 2021.

ABOUT THE AUTHOR

Michael Lynes is an entrepreneur who has founded several startup ventures. He was awarded a BSEE degree in Electrical Engineering from Stevens Institute of Technology and currently works as an embedded software engineer. When not occupied with arcane engineering projects, he spends his time playing with his three grandchildren, baking bread, working on ancient cars, backyard birdwatching, and taking amateur photographs. He's also a prolific author with over thirty works in print. His latest series is the Cozy Crystal Mysteries. Book one, *Moonstones and Murder*, is already in print, and book two is on its way. His latest works include several collections of ghost stories, short works of general fiction, a collection called Angel Stories, and another collection called November Tales, inspired by the fiction of Ray Bradbury. He currently lives with his wife Margaret in the beautiful, secluded hills of Sussex County, New Jersey. You can contact him via email at mikelynes@gmail.com.



Display technology for the embedded market is one of Avnet's core strengths, and its human-machine interface (HMI) line of displays and touchscreen technology is second to none. They also feature a comprehensive line of SimplePlus TFT displays ranging from 4.3" displays to huge 21.5" diagonals with 1920x1080 resolution, suited for high-definition medical equipment applications (Figure 10).

CODA: CIRCUIT CELLAR'S 400TH ISSUE

And last, speaking of timing—the timing of my association with *Circuit Cellar* over the past twelve months seems to me to be fortuitous, as I have very much enjoyed writing this series and I do hope it continues into the coming new year. In this particular missive, I've really gone on (and on, and on), but I think in this case I can be excused. As you are likely aware, this is the 400th edition of *Circuit Cellar* published since the start of the magazine all those many years ago. I am honored to have had the opportunity to headline it. While my association with this publication is barely a year old, I feel quite at home here, and working with Sam, our Editor-in-Chief, as well as KC, our publisher,

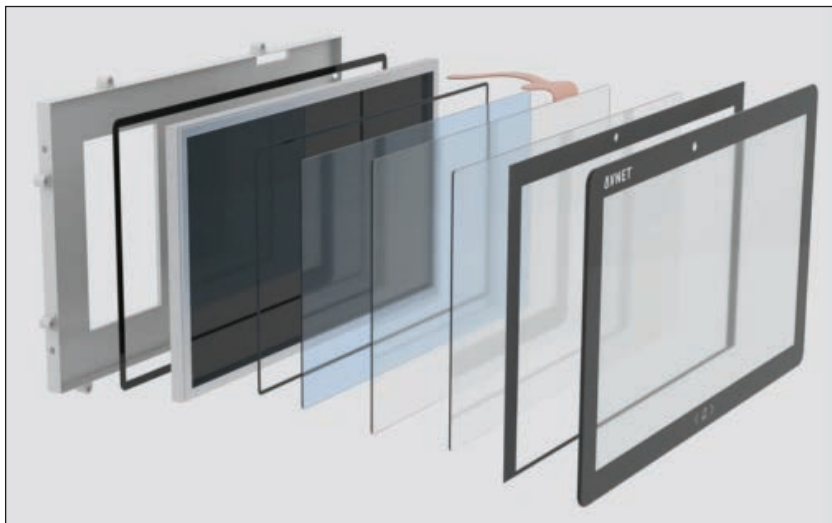



FIGURE 10
SimplePlus from Avnet Embedded

has been both a professional pleasure and a treat. In any case cellar dwellers, happy 400th! Thanksgiving is almost here in the US, and winter is coming, so let's raise an appropriately timed wassail glass and toast to CC and the next 400 editions to come. That's all from me for now. My time is up, and I can see the hook reaching out from stage left as I speak. Until next time... 

FIRMWARE CONFIGURABLE INDOOR AIR QUALITY (IAQ) SENSOR WITH EMBEDDED ARTIFICIAL INTELLIGENCE (AI)

Optimized to support public building air quality standards



Visit renesas.com/zmod4410 to learn more

RENESAS

Datasheet: DC-DC Converters

From the Hyper-Small to the Far Out

By
Sam Wallace,
Editor-in-Chief


DATASHEET

Manufacturers continue to roll out flexible, efficient, and extremely tiny DC-DC converter modules. This month's gallery offers a glimpse of the many offerings currently available on the market for myriad niche embedded solutions.

Nanopower, high-efficiency, ultra-compact, intelligent power sharing—at great risk of repeating myself, today's DC-DC converters continue to evolve in many directions. This is perfectly natural, of course, considering the myriad new, niche applications that sprout up every week, each with its own set of needs. So whether the converter is designed to accommodate multi-voltage electronics, battery-powered devices with long standby times, rugged applications exposed to a wide operating temperature range, or if the module simply needs to be very, very small, manufacturers keep rising to the challenge—producing DC-DC converters with still higher power densities, wider voltage ranges, more advanced filtering, and still tinier footprints.

It's also perfectly natural, then, that a gallery such as this can only capture a minuscule sliver of the huge breadth of options available on the market today. A primary consideration in *Circuit Cellar's* DC-DC converter selection this month was "newness." That is to say, rather than attempt to convey the full range of converter possibilities out there, we went with items that are, for the most part, piping-hot fresh. In the next few pages, we present devices with ultralow quiescent currents,

ultra-small footprints, high configurability, low costs, intelligent capabilities, and high efficiency, to name just some of the gallery's highlighted features. These converters target various sectors, from networking and communications, to battery-powered devices, to industrial solutions, to current-sensing applications, and more. There are several familiar names in the following gallery—ST, TI, and Analog Devices to name a few—and others that might be new to the reader.

As an example of an out-of-this-world application (please pardon the pun), VPT announced in August their SVLFL5000 series built for the missions in space. With "Total Ionizing Dose" (TID) performance and "Enhanced Low Dose Rate Sensitivity" (ELDRS) to 60krad, these wide input voltage range converters are operable over the full military temperature range (-55°C to +125°C) with no power derating. They are suited for applications in low Earth orbit, medium Earth orbit, geostationary orbit, and even deep space missions. I highlight this particular device to drive home the point that DC-DC converter solutions are increasingly wide-ranging, in the most literal sense. And just in case we have any engineers reading this issue in orbit. 

NEXT MONTH'S TOPIC: Tiny Embedded Boards Send related product announcements to editor@circuitscellar.com



nanoPower Boost Converter with Ultralow Quiescent Current

The MAX18000 is a nanoPower boost converter with an input voltage range of 0.5V to 5.5V ($V_{OUT} > V_{IN} + 0.2V$) and a switching current limit of 3.6A. It features an ultralow quiescent current of 512nA which makes it ideal for battery-powered applications requiring a long standby time. The IC operates in nanoPower mode at low loads and transitions into skip and CCM modes of operation at higher load currents to ensure high efficiency over a wide current range. The output voltage can be varied between 2.5V and 5.5V using a single RSEL resistor. The IC features a True Shutdown mode, which disconnects VIN and VOUT when the EN pin is pulled low.

- 0.5V to 5.5V input voltage ($V_{OUT} > V_{IN} + 0.2V$)
- 1.8V minimum start-up voltage
- 2.5V to 5.5V (in 100mV steps) output voltage
- 3.6A cycle-by-cycle inductor current limit
- 512nA IQ supply current into the output
- Output short-circuit protection
- Thermal-shutdown protection
- 95% peak efficiency with 90% or higher efficiency for load $> 20\mu A$
- 1.07mm x 1.57mm, 0.5mm pitch 6-bump WLP
- $-40^{\circ}C$ to $+125^{\circ}C$ operating temperature range

Analog Devices
www.analog.com

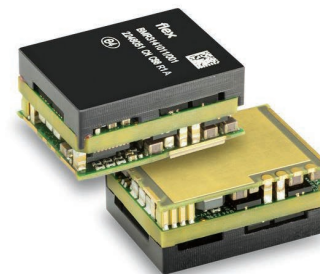


Liquid-Cooled Converter for Hybrid and Electric Vehicles

The Bel Power Solutions 700DNG40-24-8 is a 2nd generation 4kW liquid-cooled DC-DC converter that creates DC voltages in hybrid and electric vehicles suitable to power low voltage accessories. 700DNG40-24-8 converter operates at input voltages from 450 to 900VDC and delivers up to 4000W of output power. Features include very high efficiency, high reliability, low output voltage noise, and excellent dynamic response to load/input changes. This converter is designed for applications in construction equipment, underground mining, ground support equipment, on- and off-highway vehicles, and marine equipment.

- Very high efficiency up to 95 %
- Input voltage range: 450 – 900 VDC
- Output power up to 4 kW
- Parallelable up to 8 unit
- Full galvanic isolation between input and output
- Liquid cooled
- CAN bus serial interface
- Optional UDS functionality, CAN FD & Cyber security
- Adjustable output voltage and over current protection
- Over temperature, output over voltage and over current protection, input and output reverse polarity protection
- IP rating IP67 & IP6k9k
- E-Mark Certification

Bel Power Solutions
www.belfuse.com



Ultra-Small Digital Non-Isolated IBC with 4:1 Conversion Ratio

The BMR314 is a non-isolated, unregulated digital intermediate bus converter (IBC) that delivers 800W of continuous power, and 1.5kW of peak power in an ultra-small package measuring just 23.4 x 17.8 x 9.65mm. Operating over an input voltage range of 38-60V, the 4:1 input-to-output ratio results in an output range of 9.5-15V. At an input voltage of 54V, the efficiency of the module is as high as 97.4% at 50% load (35A), and the part is thermally optimized for cold wall mounting via the attached baseplate. The BMR314 is offered in an industry-standard LGA footprint and pin-out for security of supply and second sourcing. It can deliver a power density of more than 373W/cm³ or 6.1kW/in³ when delivering peak power to the load. Designed for powering cloud-based applications including AI, machine learning, and hyperscale computing.

- Compact non-isolated DC/DC converter
- Input output ratio 4:1
- Digital PMBus interface
- LGA industry standard footprint and pinout
- Halogen-free
- Optimized thermal design for cold wall
- Dimensions: 23.4 x 17.8 x 9.65 mm

Flex Power Modules
flexpowermodules.com

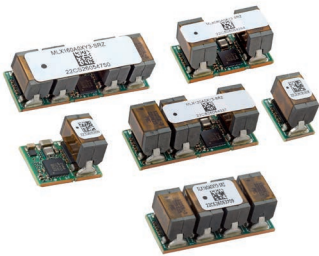
DATASHEET URLS:

Analog Devices MAX18000: <https://www.analog.com/media/en/technical-documentation/data-sheets/max18000.pdf>

Bel Power Solutions 700DNG40-24-8: <https://www.belfuse.com/resources/datasheets/powersolutions/ds-BPS-700DNG40-24-series.pdf>

Flex Power Modules BMR314: <https://flexpowermodules.com/resources/fpm-flyer-dc-dc-converters-for-ai-applications>

Datasheet: DC-DC Converters



Efficient and Flexible Power Modules

The digital DLynx III power modules are easy-to-use, highly configurable non-isolated DC-DC converters that can deliver up to 320A of output current with a master/satellite configuration. These DC-DC converters from OmniOn Power (previously ABB Power Conversion) are ideally suited for networking, industrial, and datacoms applications, and enable increased efficiency and PC board design flexibility through master-satellite groupings for single- or dual-output voltage configurations. The master DC-DC converters can be used as standalone point-of-load modules or with satellites to help meet growing board-level power requirements and power density outputs.

- Minimized board space requirements: High-density footprint (119 to 205A/in², depending on module); Optional satellite phase modules provide increased output voltage or secondary output option
- 90% efficiency at full load (12VIN, 1VOUT, at 25°C); Phase shedding for increased efficiency at low-load operation
- Modules provide maximum rated current for 12VIN, 1VOUT at 200lfm airflow and at 80°C ambient temperature or better
- Overvoltage/undervoltage and overcurrent/undercurrent protection
- Temperature operating range: -40°C to 85°C

OmniOn Power
omnionpower.com

Low-Cost, Low-Profile Isolated DC-DC Single Output Converter

The R05C05TE05S is a low-cost, low-profile, 0.5W SMD isolated DC-DC single output converter with a 4.5-5.5V input range and a semi-regulated 5V output. There is no minimum load requirement which is ideal for applications that switch into very light load operation modes. The device is also able to deliver 600mW for applications requiring additional power for short-peak operation modes. Standard isolation is 3kV_{DC}/1min, and the operating temperature is from -40°C up to +125°C with derating. The fully-automated design, which is equipped with short-circuit, over-current, and over-temperature protection, ensures the highest reliability in applications such as communication, current sensing, and COM port isolation.

- Compact 10.35 x 7.5mm SMD package
- Low profile (2.5mm)
- 3kV_{DC}/1min isolation
- Low EMI emissions
- Ultra-wide temperature range -40°C to +125°C
- Fully automated, high-reliability design
- Semi-regulated 5V output

RECOM
recom-power.com

Synchronous Step-Down Regulator for Industrial Power Systems

The RAA211630 is a DC/DC synchronous step-down (Buck) regulator that supports a 4.5V-60V input voltage range and adjustable output voltage. It can deliver up to 3A of continuous output current with premium load and line regulation performance. The RAA211630 uses peak-current mode control architecture. Its PWM switching frequency is programmable to provide the best trade-off between transient response and efficiency. It supports PFM operation and DEM to maximize light-load efficiency, in addition to an external bias LDO input to further reduce power dissipation across the load range.

- Wide input voltage range: 4.5V to 60V
- Adjustable output voltage: 0.8V to 90% of VIN
- Up to 3A of continuous output current
- Default 400kHz switching frequency and programmable switching frequency range from 200kHz to 800kHz
- ±1% Load regulation accuracy from -40°C-125°C, ±0.5% load regulation accuracy at 25°C
- 95µA typical quiescent current
- Internal compensation
- Internal 0.5ms soft-start in QFN; External programmable soft-start (HTSSOP)

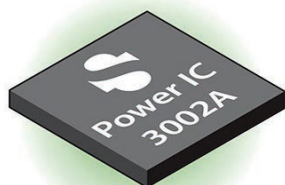
Renesas
www.renesas.com

DATASHEET URLS:

OmniOn Power DLynx III MLX040: https://www.omnionpower.com/assets/pdfs/windchill/data-sheet/mlx040_ds.pdf

RECOM RxxC05TExxS: <https://g.recomcdn.com/media/Datasheet/pdf/.fSiC8jY/.tce0d3579c4367bc096e6/Datasheet-514/RxxC05TExxS.pdf>

Renesas RAA211630: <https://www.renesas.com/us/en/document/dst/raa211630-datasheet>



Integrated Buck Converters Offer Intelligent Power-Sharing Capabilities

Silanna Semiconductor's SZPL3002A DC/DC converter ICs are the world's first integrated buck converters to offer intelligent power-sharing capabilities. The SZPL3002A is a high-efficiency, synchronous buck converter along with a USB-PD controller creating a complete, single IC, downstream facing USB-PD compliant port. The device can supply fixed output voltages as well as Programmable Power Supply (PPS) profiles for fast charging to connected devices. The device also supports the Qualcomm QuickCharge protocols, QC2.0/3.0/4.0/4.0+/5.0, supporting Type-C output ports as well as Type-A ports.

- Synchronous buck regulator with switching frequencies up to 2MHz
- Integrated USB-PD controller supporting USBPD R3.0, PPS, BC1.2, QC 2.0/3.0/4.0/4.0+/5.0 support
- Intelligent multiport power sharing and power re-balancing
- High efficiencies (>98%)
- Selectable power saving mode
- Selectable power contract configurations reduces required programming
- Temperature triggered power throttling
- VCONN power generation for e-Marked Cables
- Wide Input Voltage Range: 7.0V to 27V
- Supports VOUT of 3.3 ~ 21.5V, at 3.25A

Silanna Semiconductor
powerdensity.com



High Efficiency in a Compact Footprint

The STMicroelectronics L6983I 10W isolated buck (iso-buck) converter ensures high efficiency and a compact footprint, with advantages including low quiescent current and 3.5V-38V input-voltage range. The L6983I is suitable for applications that require an isolated DC-DC converter. It implements an iso-buck topology, which uses fewer components than a conventional isolated flyback converter and requires no optocoupler, saving bill-of-materials costs and PCB space. Further benefits of the L6983I include 2 μ A shutdown current and integrated functions such as adjustable soft-start time, internal loop compensation, and power good indicator, as well as protection from overcurrent and thermal shutdown. The selectable spread-spectrum feature improves EMC performance.

- Designed for iso-buck topology
- 3.5V to 38V operating input voltage
- Primary output voltage regulation, no optocoupler required
- 4.5A source/sink peak primary current capability
- Peak current mode architecture in forced PWM operation
- 390ns blanking time
- 200kHz to 1MHz programmable switching frequency. Stable with low ESR capacitor: min 2 μ F
- Internal compensation network
- 2 μ A shutdown current

STMicroelectronics
www.st.com



Synchronous Boost Converter with Average Input Current Limit

The TPS61299 is a synchronous boost converter with a 95nA ultra-low quiescent current and an average input current limit. The device provides a power solution for portable equipment with alkaline batteries and coin cell batteries. This device has high efficiency under light-load conditions to achieve long operation time and the average input current limit can avoid battery discharging with high current. The TPS61299 has a wide input voltage range from 0.5V to 5.5V and an output voltage range from 1.8V to 5.5V. The device has different versions for the average input current limit from 5mA to 1.9A. The TPS61299 with a 1.2A current limit can support up to 500mA output current from 3V to 5V conversion and achieve approximately 94% efficiency at 200mA load.

- Input voltage range: 0.5V to 5.5V
- 0.7V minimum input voltage for start-up
- Input operating voltage down to 150mV with signal VIN > 0.7V
- Output voltage range: 1.8V to 5.5V VSEL pin select output voltage
- Average input current limit: 5mA; 25mA; 50mA; 100mA; 250mA, 500mA, 1.2A, 1.9A (different versions)
- 95nA typical quiescent current from VOUT
- 60nA typical shutdown current from VIN and SW

Texas Instruments
www.ti.com

DATASHEET URLS:

Silanna Semiconductor SZPL3002A: <https://powerdensity.com/wp-content/uploads/2022/08/SZPL3002A-Product-Brief-Prelim.pdf>

STMicroelectronics L6983I: <https://www.st.com/resource/en/datasheet/l6983i.pdf>

Texas Instruments TPS61299: <https://www.ti.com/document-viewer/tps61299/datasheet>

PICKING UP MIXED SIGNALS

Before Transistors

How Did They Do It Back Then?

As part of *Circuit Cellar's* celebration of its 400th issue, Brian looks back at some ingenious electromechanical devices that performed necessary functions, using existing technology. Many were so clever that they are still in use today—even while microcontrollers are used in just about everything.

By
Brian Millier

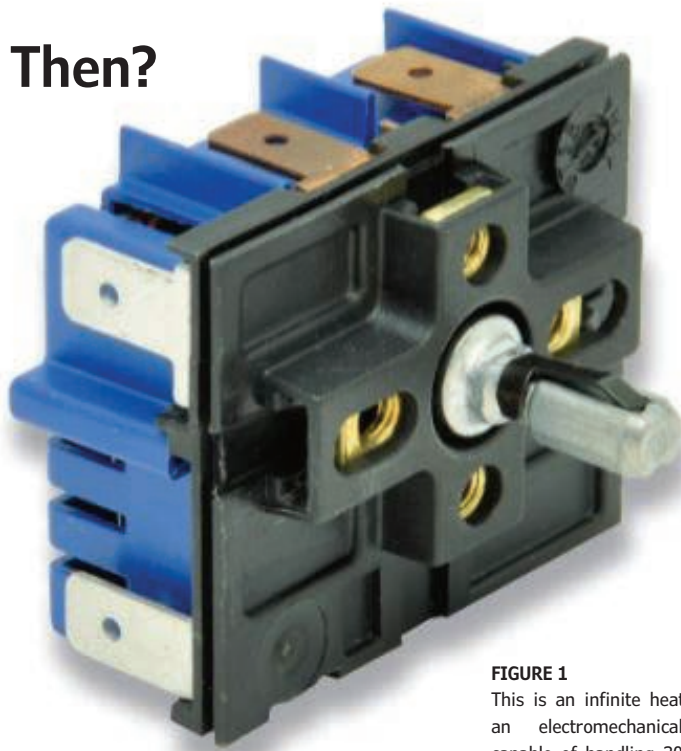


FIGURE 1

This is an infinite heat switch—an electromechanical device capable of handling 3000W with a resolution that would normally be associated with a PWM circuit.

Both at work and at home, we expect that most of the devices we use daily will contain some form of a microcontroller (MCU), or at least electronic circuitry. Sometimes this trend goes overboard, and we hear talk of toasters that are Internet-connected.

Many of these consumer and commercial/industrial products were introduced 50-75 years ago—prior to the invention of transistors, never mind complex digital integrated circuits. How did the engineers and designers of the era design products that, while not as sophisticated as those available now, nevertheless did the job adequately? When I began my career at General Electric Consumer Products Division, I quickly discovered just how cheaply electromechanical components could be made, and that these components could still do the job in a reliable way.

In this column, I'm going to describe several components and circuits—found in industrial, scientific, and consumer products—that I've worked with and found to be extremely ingenious. Let's start close to home.

A HIGH-POWERED KITCHEN

Electric stoves have been around for 100 years. Regardless of their vintage, they all contained a number of "surface units" for

heating foods in pots and skillets. These heaters consume about 3000W, on average, and ideally should be finely adjustable in terms of heating power delivered. Early stoves contained surface units in which the spiral Calrod heating element was made up of three discrete heaters of different wattages. A complex mechanical switch would "dial in" eight different combinations of these three elements, providing a rough control of heating power. This wasn't ideal, and in time, the infinite heat switch was invented (**Figure 1**).

Using modern components and techniques, controlling an AC-powered 3000W heater today would likely be done with a PWM generator feeding a solid-state relay (which is basically an opto-coupled TRIAC mounted in a heatsink enclosure). In the 1970s, when infinite heat switches were introduced for stoves, SCRs were quite new and still expensive, and TRIACS were not available.

Referring to **Figure 2**, SW1 is a set of high-current contacts, one of which is connected to a length of bimetallic strip. A bimetallic strip is made up of two dissimilar metals that each have different thermal coefficients of expansion. When heated, a bimetallic strip will bend in proportion to how much heat is applied. In this device, the bimetallic strip has a small heater coil wound around it. The dial

of an infinite heat switch is connected to a contoured cam. When it is moved from the off position, it closes both SW2, the on-off switch, and SW1 containing the bimetallic strip. At this point, the 240V_{AC} will pass through both switches and power up both the surface unit and the small heater wound around the bimetallic strip. When this small heater heats up it will bend the bimetallic strip so that it opens the contacts of SW1. Both this small heater and the surface unit will then stop heating, and in a few seconds, the bimetallic strip will cool off and SW1 will again close. While this diagram doesn't accurately reflect the shape of the cam, suffice it to say that as you rotate the dial knob for more heat, the cam will adjust the position of the bimetallic strip in such a way that it must heat up more to open SW1's contacts. The whole on-off cycle will take 10-40 seconds, depending upon the dial setting, and the PWM duty cycle will vary from about 10% to 100% in fine increments.

When I first encountered them in the mid-1970s, infinite heat switches were made by Robertshaw, and they still are. Back then, I'm sure they cost no more than a few dollars to manufacture—a tiny fraction of what it would cost to perform the same function using PWM plus a solid-state relay. From experience, I know that they can easily last 20+ years in normal service. They are still in common use in stoves today.

THEY PUT VACUUM TUBES IN CARS?

Even if you are too young to have ever used them, most electronics people know what vacuum tubes are. Compared to transistors, they get hot, draw much more power, and are a lot more susceptible to shocks and vibration. They're not something you would expect to see in a car. However, car radios were a popular option from the 1950s onward. While transistors had been invented in 1947, they were neither robust nor high enough in performance to operate in car radios for many years. So, for at least a decade or so, vacuum tubes were used in car radios.

Vacuum tubes often use 12V for their filaments, which matches the 12V car battery. However, to operate efficiently, they need at least 100V_{DC} for their plate electrode. Using a "B" battery (around 100V), which was employed in old home radios, was not practical in a car. So, the 12V_{DC} battery voltage had to be converted into AC, stepped up using a transformer, and then rectified back to a high enough DC potential to run the vacuum tubes. It would be easier to do this in today's cars since alternators are now used to charge the battery. Internally they produce AC voltage which is rectified before

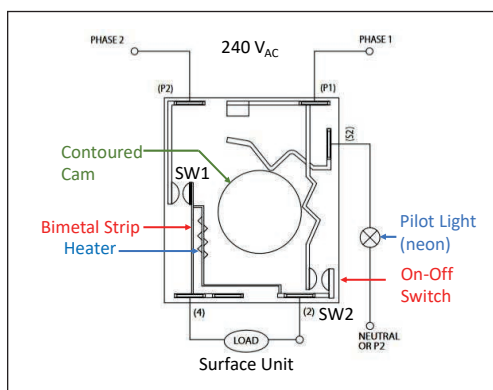


FIGURE 2

This is a block diagram of an infinite heat switch. In the text, I describe how it operates.

it leaves the alternator. That AC voltage could feed a transformer directly, but automotive generators of that era produced DC voltage only.

Figure 3 shows a picture of the device that made vacuum tube car radios possible. It was called a vibrator. This one was made by Cornell Dubilier, which was well-known for its capacitors. I'm guessing Cornell Dubilier got into the vibrator business because it already used those cylindrical cans to house its power supply electrolytic capacitors.

Figure 4 is a schematic diagram of the circuit using such a vibrator in a car radio. Basically, a vibrator is like a two-pole relay, designed to handle being switched on and off rapidly, and for a long duration. The 12V battery voltage is applied to two of the vibrator's four terminals. The current passes through a set of N.C. contacts to the vibrator



FIGURE 3

This is a vibrator unit that was used in early car radios to provide enough voltage to operate the vacuum tubes that were used in car radios at the time.

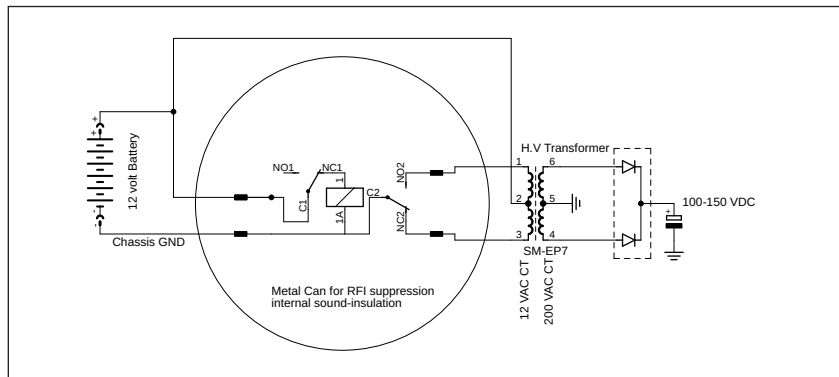


FIGURE 4

This is a schematic diagram of the high-voltage power supply used in older car radios containing vacuum tubes.

coil, which energizes it. Once energized, it opens that N.C. contact, and the relay coil is deactivated. This part of the vibrator acts much like the old electromechanical buzzers used in the past, except that here we don't want to hear the buzzing sound—therefore the can surrounding this vibrator is sound-insulated. I can't recall the frequency that these vibrators operated at, but it was somewhere between 50 and 100Hz. The second set of contacts is SPDT and basically switches opposite ends of the radio's power transformer to chassis ground. With 12V_{DC} from the battery supplied to the center tap of the primary, we are effectively supplying a square wave AC voltage to the transformer's primary winding. The voltage from the transformer's high-voltage secondary is full-wave rectified (by a vacuum tube rectifier, if my memory serves me correctly) and filtered.

It speaks well of the engineers at the time that they could design an AM radio that would pick up distant RF signals clearly while in the presence of electromagnetic interference (EMI) from the spark plugs, distributor, generator commutator, and the contacts in the vibrator itself. Vibrators were not expensive in those days and lasted for many years.

SLOWLY DRIFTING AWAY

After learning about the automotive vibrator in the last section, can you think of another place where such an electromechanical device could be used? Let's consider industrial process controllers, specifically ones in which temperature is controlled—possibly high temperatures in large ovens. The only temperature sensors capable of withstanding high temperatures—rugged and can work with long signal leads—are thermocouples.

Thermocouples produce only low millivolt-level signals even over a high-temperature span. These tiny signals must be amplified greatly before they can be used in some form of PID controller. However, amplifying a slowly changing DC signal by a large amount requires a high-gain amplifier, with a frequency response down to DC. Modern op-amps with zero-drift features are common today, but they weren't 30-70 years ago when such controllers were required. It was difficult to design a high-gain DC amplifier that did not suffer from some amount of drift over temperature/time (especially using vacuum tubes). This drift could severely affect the accuracy of the controller, and many processes are critical regarding process temperature.

If you instead consider a high-gain AC amplifier, you can see that a multi-stage amplifier (needed for high gains) can have its stages coupled via capacitors. Any DC drift in a particular stage will not pass through this capacitor to subsequent stages. Therefore, a good solution is to convert the low-voltage thermocouple signal into an AC voltage, amplify it with a high-gain, drift-free AC amplifier, and then convert it back to DC again at the output.

Today, such switching is generally done using MOSFETs. They have fairly low R_{DS} values, and don't generate any DC offsets of their own (which would be meaningful given the low voltages produced by thermocouples). They can suffer from a phenomenon known as charge injection, but this isn't much of a consideration at the low switching frequencies needed for this type of application.

However, MOSFETs weren't around back in that era. Instead, if you consider the automotive vibrator, it has all the attributes needed to do the DC-AC conversion. In particular, it

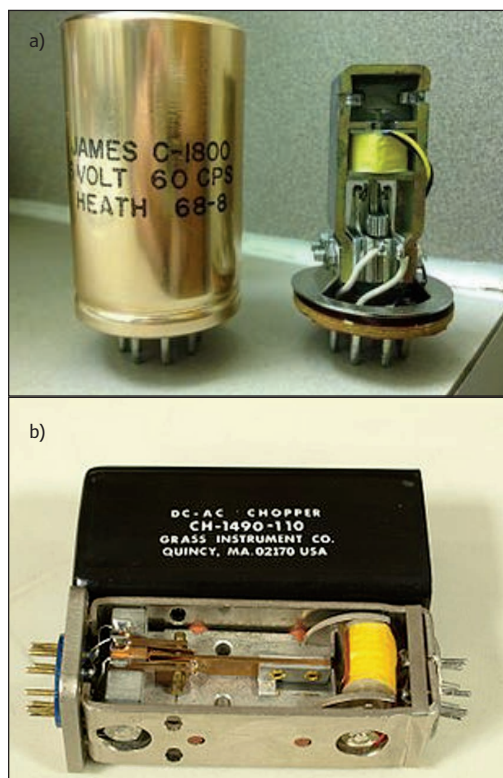


FIGURE 5A,5B

These are two photos of choppers used in early DC amplifiers. They converted the DC voltage into AC, where it was amplified by an AC amplifier, and then synchronously rectified back to DC using a separate set of contacts.

has basically 0Ω contact resistance when the contacts are closed, and it doesn't generate any offset voltages of its own. And there's another bonus: If you add a second SPDT set of contacts to the vibrator design, you can use that set to synchronously rectify the AC signal at the output of your AC amplifier. Again, no offset voltage errors are introduced, as these are only mechanical contacts.

In this case, such devices were called choppers. **Figure 5A** and **Figure 5B** are examples of such choppers. In a car, there was only a DC voltage available, so the vibrator needed its own set of contacts to switch the coil on and off rapidly. My memory is a bit hazy on this now, but I believe that the chopper's coil was fed an AC voltage and the switching was performed at 60Hz (at least here in North America).

I had some spare choppers in my lab, which were used in chart recorders (another device that needed to be able to amplify tiny DC signals in a stable, drift-free manner). They were somewhat smaller than the ones shown in Figure 5. They didn't turn my lab into a museum when I retired, and in time, all of these older parts were disposed of, so I have no photos.

A LIGHTBULB MOMENT

I recently read that the USA is banning incandescent light bulbs. Today, of course, the media tries to make a controversy about everything, so there is some confusion about whether this is a ban or just new, stringent energy efficiency regulations. One way or the other, almost no one uses incandescent lights for lighting any longer, since LEDs are cheap and vastly more energy efficient.

Figure 6 is a photo of a miniature light bulb like that which played a significant role in the first product ever produced by Hewlett-Packard—now a large multinational conglomerate in the computer/electronics industry. No, they didn't start out producing light bulbs. Instead, Bill Hewlett and David Packard's first product was an audio signal generator which was initially manufactured in David Packard's garage in Palo Alto. You can Google "HP 200A" for a photo of the original HP 200A generator, but those photos are not clear enough to meet *Circuit Cellar's* publishing standards.

High-quality audio signal generators were, and still are, essential in the audio industry. In particular, low waveform distortion and a wide frequency range are required. Getting both qualities simultaneously makes the design more difficult, but the Wien bridge oscillator configuration is one of the better choices. **Figure 7** is a schematic diagram of the Wien bridge oscillator in HP's original design. The

frequency-determining components are $R1, C1$ and $R2, C2$, where $R1=R2$ and $C1=C2$. Capacitors $C1, C2$ are two sections of an air-variable capacitor and $R1, R2$ are switched by the frequency range switch. The combination of vacuum tubes $V1$ and $V2$ provides AC voltage gain. For a Wien bridge oscillator, the gain of the amplifier must be >3 for the circuit to sustain oscillation. However, if the gain is too large, the oscillator will saturate. But, even before such saturation, the sine wave amplitude would not remain constant if the gain changed.

Incandescent bulbs have a large positive temperature coefficient. That is, the hotter they get, the higher their resistance becomes. For lighting purposes, this is a double-



FIGURE 6

This is a miniature incandescent lamp like the one that was an integral part of H.P.'s first product: the HP 200A audio signal generator.

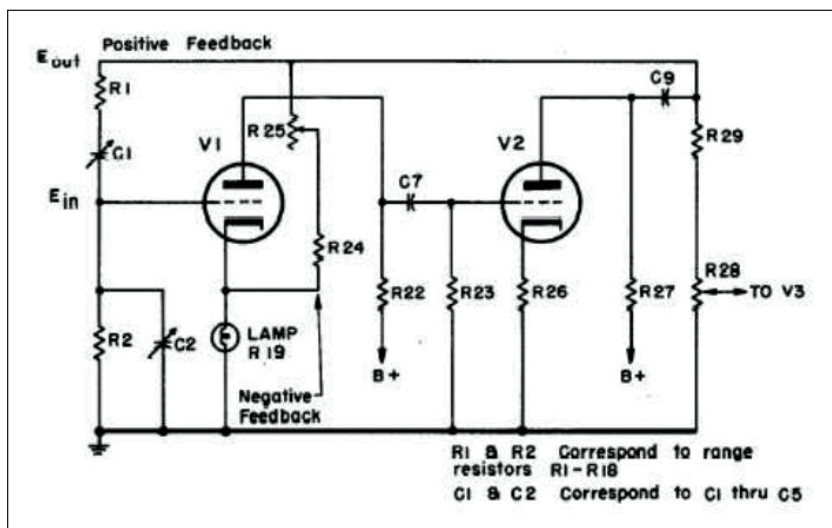


FIGURE 7

This is a schematic diagram of the Wien bridge oscillator circuit used in the HP 200A audio signal generator.

FIGURE 8

This is an HP 200CD audio signal generator. It was a bit later model of the HP 200A. I used one in my lab.



edged sword. On the plus side, as the AC mains voltage varies (from time of day and load conditions), an incandescent bulb will compensate, to a fair degree, in terms of brightness. The downside is that the bulb's filament is always cold when you first turn it on, and its lower cold resistance leads to a momentary current surge at turn-on. That's why incandescent lamps often burn out right after you turn them on.

From Figure 7, you see that the bulb is placed in the cathode circuit of V1. The AC audio signal is coupled to the lamp via R24 and pot R25. When the signal amplitude increases, it feeds more power to the lamp, increasing its resistance. The higher the resistance present in the cathode return path to ground, the lower the gain of V1. This negative feedback, combined with the particular characteristics of the R19 lamp, act to produce a constant amplitude audio signal—even when switching ranges.

While there are other, more complex ways of building a Wein bridge audio oscillator with a constant output amplitude, none of these alternative circuits are even close to the simplicity and cost of the HP 200A's light bulb scheme. The HP 200A was granted US Patent #2268872 in 1942.

Figure 8 is a photo of the somewhat newer HP 200CD which uses somewhat different circuitry from the original HP 200A design, but still uses vacuum tubes and the incandescent light bulb. There was one in my lab when I arrived, which I used, and which was still working when I left 30 years later.

THE GLOVES CAME OFF

I have another clever use of light bulbs. In the Department of Chemistry at Dalhousie University where I worked, there were numerous glove boxes, like that shown in **Figure 9**. In case you're wondering, the gloves don't extend outward like that when you are using them! The enclosure is hermetically sealed and often filled with a gas other than air—for chemical and/or safety reasons. The gas used may be expensive, so for that and safety reasons, it's useful to be able to know if there are leaks in either the enclosure or the gloves themselves. If you cut a hole in the glass envelope of an incandescent lamp, you can operate it with something other than the vacuum under which it's accustomed to operating. If this modified lamp were operated in our normal atmosphere, containing oxygen, it would quickly burn out due to oxidation of the filament. You'll have noticed this if you've broken a light bulb that's powered up. However, many of the gases used with glove boxes do not oxidize the lamp's filament, so you could power up this modified bulb in that atmosphere and it wouldn't burn out.

**FIGURE 9**

This is a glove box apparatus that is used in chemistry labs to handle chemicals either that are dangerous to humans or that must be handled in an atmosphere made up of gas(es) different from Earth's normal atmosphere.

Figure 10 shows a simple schematic of such a leak alarm. The 120V mains power is applied to the incandescent bulb (a 120V_{AC} 7W night light bulb works well) through resistor R5. The AC voltage across R5 is adjusted downward by pot R3 to provide 2.0V_{DC} after rectification/filtering by D1 and C1. Under normal conditions, the lamp is lit up and there is enough voltage across R5 to turn the optocoupler on. Under these conditions, the 2N3904 does not conduct and the alarm buzzer doesn't sound. If the glove box leaks and lets in oxygen from the outside air, the lamp quickly burns out and the optocoupler shuts off. This raises the voltage on the base of the 2N3904 and the SP1 buzzer sounds. Also, a positive 5V can trigger an optional timer module to start accumulating elapsed time. This allows the glove box operator to know for how long the proper gas atmosphere has been lost, and to act accordingly.

Like HP's audio oscillator, this design, using a light bulb, is much simpler than other ways of accomplishing the same function. I built quite a number of these devices for some of the many glove boxes that were used in our chemistry department.

MECHANICAL ACTUATORS

Aside from robotics, there are many other areas in which mechanical actuators are

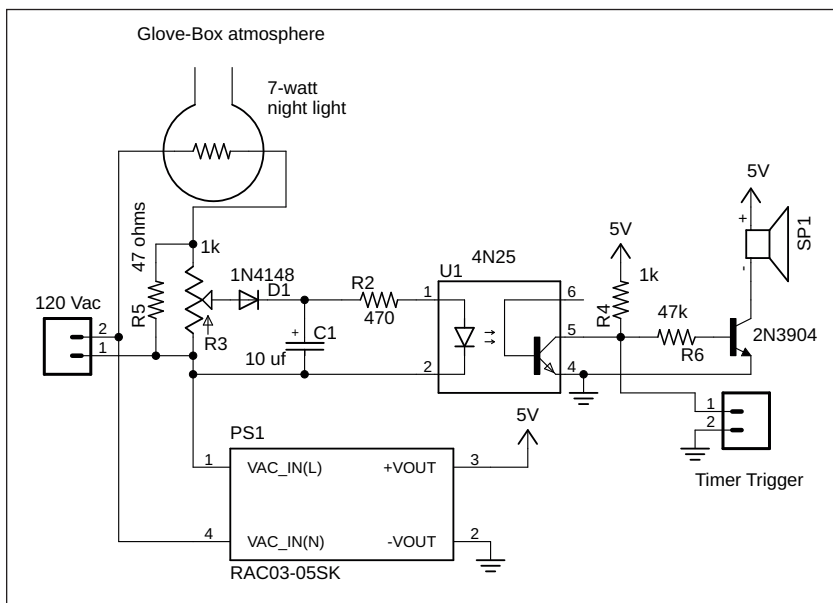


FIGURE 10
This is a schematic of a simple glove box leak detector based on a simple miniature incandescent light bulb. Like the HP 200A, it's a novel use for a simple light bulb.

needed to perform some physical function. Depending upon the task, either stepper motors or servomotors might be the best solution. Linear actuators are another common solution, but if they are electrically driven (not pneumatic or hydraulic), they would generally be driven by either a stepper motor or a servomotor.



OmniOn Power's Dlynx III: Flexible, Reliable Power Design Across the Board

Dlynx III DC/DC converters deliver high-efficiency 40A to 320A power, helping to meet the needs of the latest networking, data center, and industrial equipment. The modules utilize a master-satellite concept to provide board design flexibility and high power density.

Learn more at OmnionPower.com



FIGURE 11

This is a selsyn unit—they were later re-labeled “synchro.” Two such units, wired together, can transmit the rotary position of one unit to be displayed on the other unit.

In almost all cases, you need some positioning information fed back to the controller. This sensor would often take the form of a digital rotary encoder, or maybe just a potentiometer if the motion was limited to <360 degrees of rotary motion. Alternately, a linear potentiometer could be used if the motion was linear.

What if we add the criteria that there must be some haptic feedback as well? In simple terms, think of haptics as the controller needing to know how much resistance the actuator is encountering when it’s moving toward its targeted position. Possibly add to this the need for limit switches so the actuator doesn’t destroy something when it tries to move beyond some mechanical limit. Suddenly, the design of the controller/actuator becomes quite a bit more complicated. Modern controllers with powerful MCUs and intelligent sensors can handle this without too much difficulty. However, before transistors and IC chips, controlling an actuator electrically was difficult enough that pneumatics/hydraulics were often used instead, particularly in industry.

There was, however, one ingenious device called the selsyn, invented back in 1925, that handled all of the following:

- Physical motion (actuator)
- Position feedback
- Haptic feedback

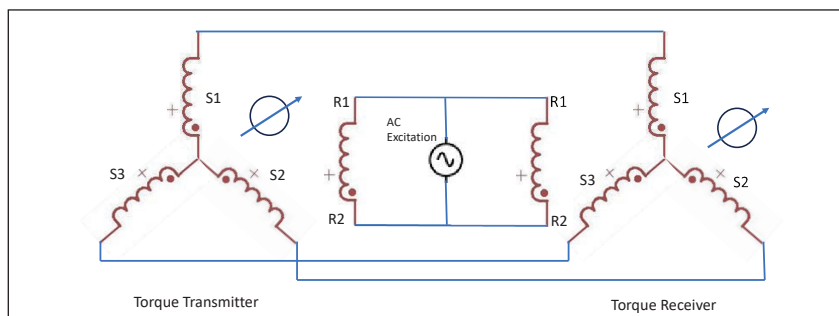


FIGURE 12

This is a schematic diagram showing how two synchros are wired together.

Figure 11 is a photo of a small selsyn. It’s configured like a small three-phase motor, with an additional coil (terminals R1,R2) mounted on the device’s rotor and connected externally via slip-rings. Its mechanical rotary output comes from the threaded shaft on the right. The stator is made up of three Y-connected coils (S1,S2,S3) spaced 120 degrees apart. While a selsyn is a motor, it can equally act as a generator. In fact, selsyns are always used in pairs (one transmitter and one or more receivers). **Figure 12** is a schematic diagram of a basic selsyn receiver-transmitter pair. The transmitter’s S1,S2,S3 coils are connected to like-named coils on the receiver. When an AC excitation voltage is applied to each unit’s rotor coil, it will induce a voltage in the three stator coils. The phase of each of those signals will be displaced by 120 degrees and will vary as the transmitter shaft is rotated. These three voltages are applied to the receiver, which will cause the receiver’s rotor to move to match the position of the transmitter.

Selsyns are commonly used to transmit the rotary position of some remote mechanical device to a receiver, which is configured with a dial, allowing it to be read like a meter. This is referred to as a torque system. However, you can also use them as an actuator: a person can rotate the transmitter selsyn shaft, and the receiver selsyn will move to match that rotational position. If the receiver encounters some resistance in achieving that position, that will be felt by the person rotating the transmitter (haptic feedback). The amount of torque developed by the receiver selsyn is small—somewhat limited by the torque that the operator can exert on the transmitter. This is called a control system. Optionally, the transmitter’s output signals can be amplified and fed to a servomotor if a larger torque is needed. During the Second World War, the term selsyn was replaced by the term synchro.

As a teenager, I was fortunate enough to get a truckload of electronic military surplus equipment removed from the DEW line in northern Canada. I was delighted to find a lot of military-grade 6LC, 12AX7, 12AU7 vacuum tubes, power transformers, and audio power transformers. These all made their way into guitar and hi-fi amplifiers that I built. I came across a lot of selsyns as well, but didn’t know what they were used for until much later in life. There was no Google back then.

MUSICAL INSTRUMENT EFFECTS PROCESSING

I mentioned my interest in guitar amplifiers in the last section. From the 1960s onwards, electric guitars and organs played a large role in rock and popular music. Using only vacuum tubes and early transistors, many sound

“effects” were invented back then that are still popular and routinely used with electric guitars and organs. In general, these fall into four categories:

- Tremolo (amplitude modulation)
- Vibrato (frequency modulation)

- Phasor/flanger (phase modulation/comb filtering)
- Reverb/delay (basically introducing echo into the signal, over time)

Let’s examine the vibrato effect—specifically as it was implemented in the

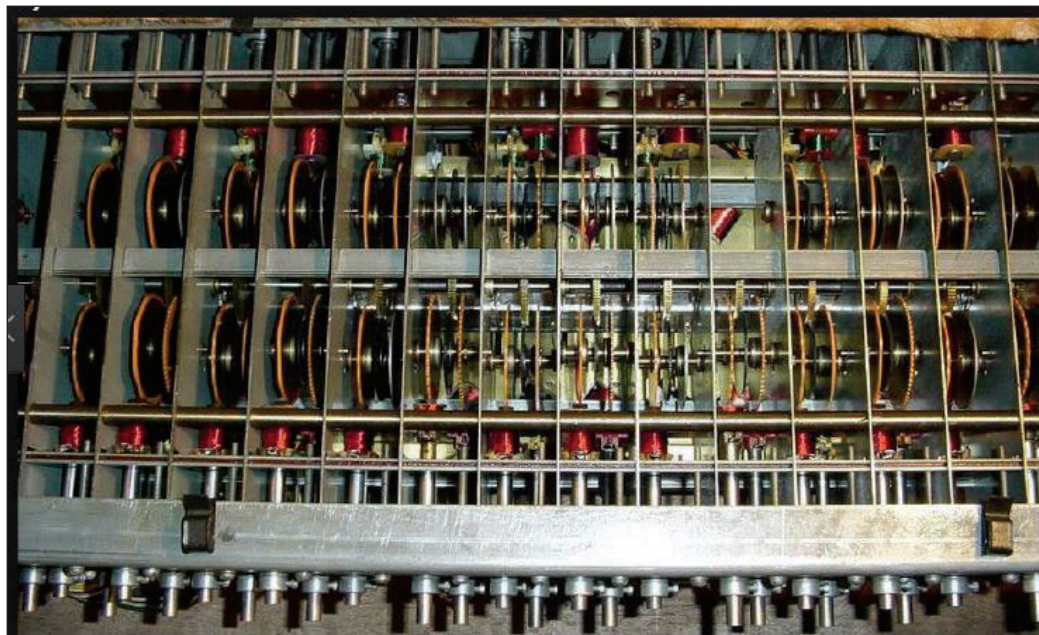


FIGURE 13

This is a photo of the tonewheel generator as was used in the famous Hammond B3 organ. The tonewheels, which have “teeth” something like gears have, are clearly visible, as are the associated pickup coils.

AVNET®

/ LAUNCHING... WHAT'S NEXT!

Launching new technology is more complex than ever. When it comes to navigating today’s design and supply chain challenges, Avnet is at the heart of it all. Whether you’re just starting on a design or working to get your product to market, Avnet delivers the right mix of technology and expertise to help your business succeed. We deliver what’s next in design, supply chain and logistics so you can deliver what’s next for all of us.

Learn more at avnet.com

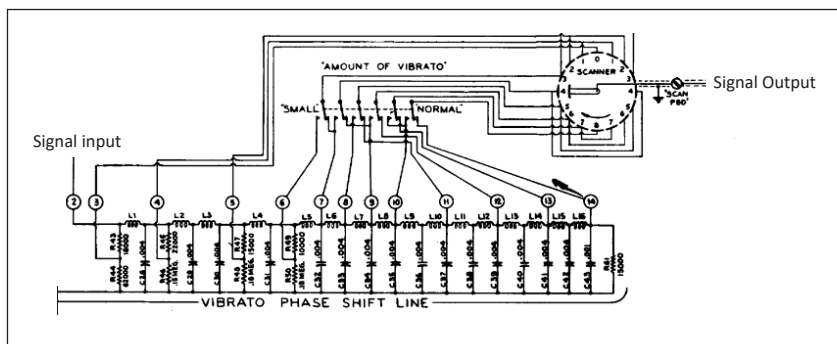


FIGURE 14

This is a schematic diagram of the vibrato circuit in a Hammond tonewheel organ. The heart of this circuit is the cascaded LC network and its associated scanner switch.

famous Hammond tonewheel organs such as the B3. Briefly, Hammond tonewheel organs generate the frequencies needed for each note on the keyboard (plus a lot of selectable harmonics) using metal wheels, which are machined with a sine-wave pattern along their circumference. These wheels are rotated at a fixed speed by a synchronous motor, and a pickup coil is placed near the wheel's circumference, which generates a sine wave at a specific frequency. Depending upon the model, there are 91 or more of these wheels/pickup coils. That amount is needed to produce the fundamental tones for all of the notes within the keyboard's range, plus many user-configurable harmonics. **Figure 13** is a photo of the tonewheel generator assembly with the wheels and coils clearly visible. This sound generation method is known as additive synthesis and is still one of the best synthesis methods available, even using today's complex digital integrated circuits. In *Circuit Cellar* #328, I designed a Teensy MCU-based Tonewheel organ synthesizer. There is much more background on these organs contained in that article ("Simulating a Hammond Tonewheel Organ—Part 1: Mimicking a Mechanical Marvel," *Circuit Cellar* 328, November 2017) [1].

A big advantage of the tonewheel organ was that every note was properly in tune, without any adjustments needed, due to the fact that the tonewheels were rotated by a synchronous motor driven by the very accurate 60Hz power mains. The downside of this is that there was no easy way to introduce vibrato—a slow modulation of the note's frequency.

Today, with fast MCUs and large amounts of RAM and such, we could take the digital signal representation of a fixed-frequency sine wave, and introduce vibrato as follows:

- Feed the digital sound samples into a large circular RAM memory buffer.
- Maintain input and output buffer pointers into that buffer.

- Slowly manipulate the position of the output pointer with respect to the input pointer in such a way as to introduce a phase/frequency variation.

This is a bit of a programming effort but quite doable with today's MCU and memory devices. But how would you do this back in the 1930s when the Hammond tonewheel organ was designed? Hammond's solution was quite ingenious. **Figure 14** is the electrical schematic for what was called the Vibrato scanner. The electrical signals representing the notes being played enter at terminal 2, to the left. You can see that there is a whole series of LC networks that are series-connected. The time constant of each of these LC networks is large enough to introduce a significant phase shift to the incoming signal. This phase shift steadily increases as you move from left to right in the LC network. At each "tap" of the cascaded LC network, there is a signal that goes to what is labeled the scanner. For now, consider that to be a 16-position switch. The scanner switch is rotated by the same synchronous motor that turns the tonewheel generator. As the scanner rotates, it will select various taps of the LC network, and the varying phase shift applied to the input signal will be enough to produce a pleasant vibrato effect. There is also a switch that can select which switch taps are fed by the LC network—this allows for various amounts (depths) of vibrato effect.

Were the scanner to actually be a mechanical switch, it would produce frequency variations in discrete steps. Also, the switch would have to be make-before-break or the signal output would be interrupted with small intervals of silence.

Using a mechanical switch wouldn't work in practice. The digital vibrato solution, that I presented above, would have thousands of elements in the buffer array, making each sample close together in time. Thus, you could achieve a smooth vibrato response. Here we only have 16 "switch" positions on the scanner and that isn't nearly enough resolution.

Figure 15 is a picture of the inside of the Scanner. You can see that at each scanner position, there is stacked a series of copper plates. While not visible, the rotator part of the scanner is also a series of stacked copper plates, and is spaced between the fixed plates. This forms an air capacitor which results in basically a variable capacitor between the rotator and each of the 16 fixed scanner capacitors (three of which are removed in this photo). As the scanner rotates, it will smoothly mix in various proportions from any two adjacent

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

Reference [1] as marked in the article can be found there.

fixed scanner capacitors. This will provide a smooth vibrato signal.

The capacitance of a small air variable capacitor, such as those 16 found here, is quite small. I don't think that specification is available, but I would estimate it to be in the tens of picofarads. Given such a small coupling capacitance, the impedance of the amplifier following it must be high or there would be poor low-frequency response. Since they used vacuum tubes for the amplifiers in these organs, the high-impedance criterion wasn't hard to meet. However, the shielding of both the scanner assembly and the shielded cable leading to the following amplifier had to be good, or there would have been excessive hum in the organ's output signal.

The sound of Hammond tonewheel organs was so exceptional that they produced about 2 million of them between 1935 and 1975. Even though the newest of them would now be 50 years old, many thousands of them are still in use. They were originally designed for churches, but I suspect that the remaining ones are used mostly by rock/pop music bands. If you ever had a chance to see the complex mechanical components inside of one of these organs, as I have, you would be astonished to know that they were sold for about \$1200 dollars when first introduced in 1935.

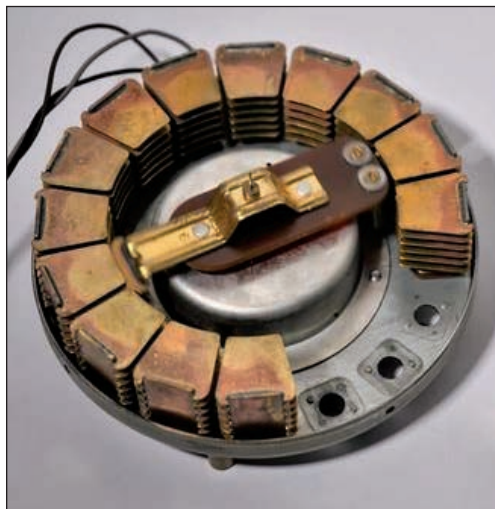



FIGURE 15

This is a photo of the vibrato circuit's scanner. This is basically a "switch" using capacitive coupling. See the text for more details on why this was so ingenious.

CONCLUSION

This article was intended as a special one-off for the 400th edition of *Circuit Cellar*. However, while looking back over the last 50 years in which I've been active in electronics, I collected many other ideas/devices like the ones described here—more than would fit into one column. If there is reader interest, I may sprinkle these other ingenious design ideas into future *Circuit Cellar* editions. I hope you enjoyed the trip back in time. 

COLUMNS

ADuC841 Microcontroller Design Manual

From Microcontroller Theory to Design Projects

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

Buy it today!

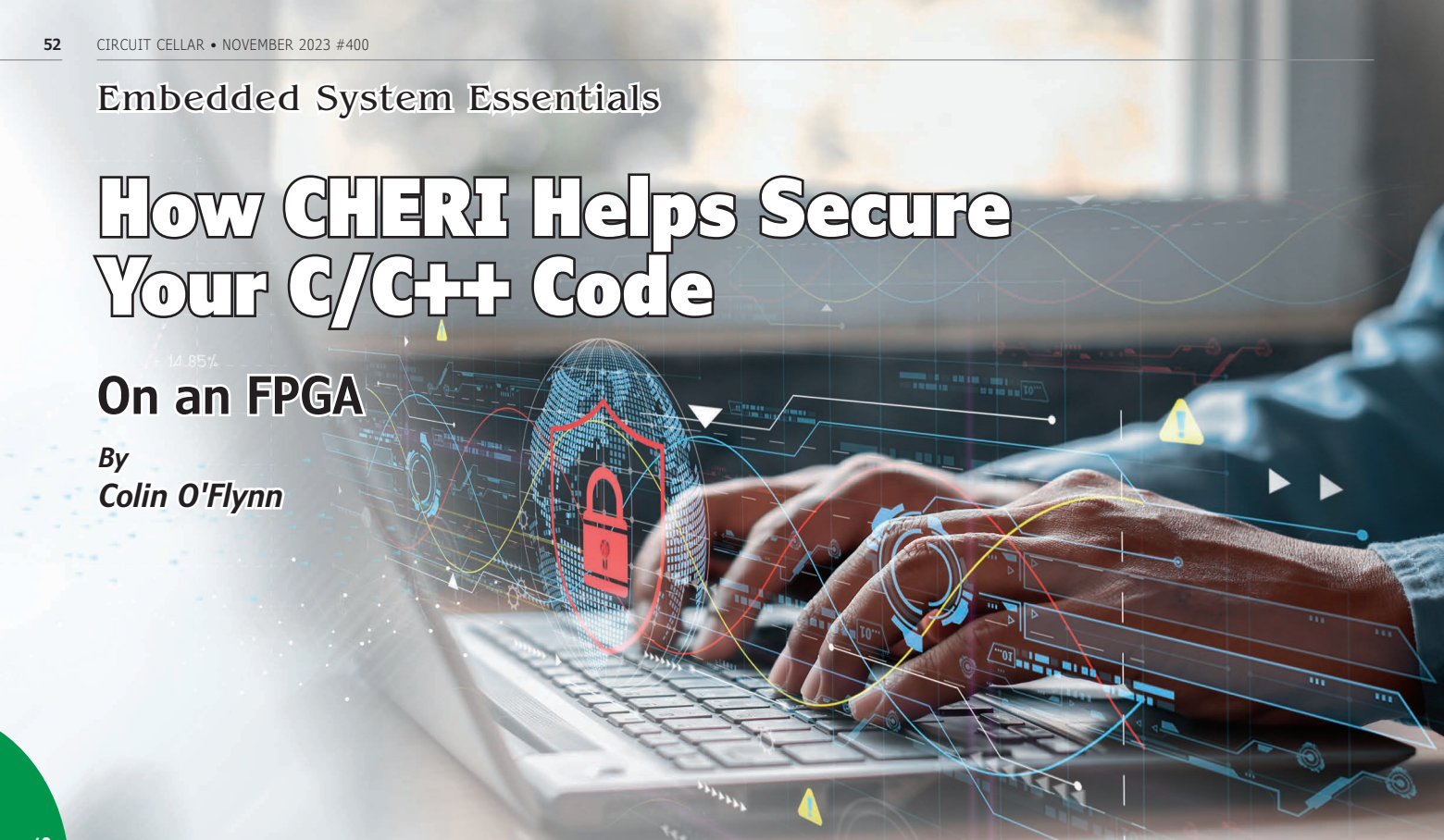
cc-webshop.com

Embedded System Essentials

How CHERI Helps Secure Your C/C++ Code

On an FPGA

By
Colin O'Flynn



Most embedded attacks either start with or end with illegal memory accesses. The typical linear address spaces of most microcontrollers, combined with the many years of non-memory-safe legacy C/C++ code, mean that this will be a threat for many years to come. A newer technology called CHERI is trying to add memory safety to your existing code, and a recent open-source RISC-V version called CHERIoT has turned it into something you can experiment with today.

This column normally covers how to attack embedded systems. For this special issue, I'm taking a step back to look at how to build secure embedded systems. Ultimately, it's the goal of most embedded engineers to improve their systems. I talk about embedded attacks because understanding attacks is an important step in the process. But once you know the attacks, what do you do next?

In this article, I'm going to introduce a new technology called Capability Hardware Enhanced RISC Instructions (CHERI), which is an extension to microcontroller (MCU) Instruction-Set Architectures (ISAs) that builds in capabilities for fine-grained memory protection and software compartmentalization. The exciting thing about CHERI is that it provides a way for you to take existing C/C++ code (which famously ends up with lots of security vulnerabilities) and provide protection against entire classes of attacks, including ones

I've shown you before. This means the "what you do next" step may require little effort beyond recompiling your code (and hoping your RISC-V core has the CHERI extensions).

CHERI technology has been around for a few years, and Arm has even built some demonstration boards (called the "Arm Morello") that include this technology. More recently, an open-source RISC-V specification called CHERIoT was produced, and a demonstration RISC-V core called CHERIoT-Ibex was released which allows you to experiment with this on an FPGA development board. The technology is even easier to access thanks to a new project called the "Sunburst Project," which will have a special-purpose development board (the "Sonata Board"), designed by yours truly. Watch the lowRISC website for future details of this design. The board design will be open-source, so you can build one yourself if you're handy with the soldering iron!

ATTACK THE MEMORY

Before we dive into the details of what CHERI is, let's look at how the most common embedded system attacks work. Most attacks on embedded systems exploit improper access to memory. This works in practice because of two simple facts:

1. Most embedded systems have one memory space containing everything.
2. Memory protection, if enabled at all, may not be fine-grained enough to prevent an attacker from reading (or writing) sensitive data.

Buffer overflows are a good example of a simple attack here. The basic idea of a buffer overflow is shown in **Figure 1**. In a buffer overflow, an attacker overwrites the end of a buffer, which ends up writing data onto the stack. This stack normally includes return addresses, which allows an attacker to change the control flow of the program. In other cases, the attacker is able to write executable code that the victim jumps to and executes.

Other common attacks include reading past the end of memory, or reading memory they shouldn't have access to. This might be possible with logic flaws, such as improperly checking the bounds of a request. But also many of my fault injection attacks exploit this, like when I showed you how I read the private key from a Bitcoin wallet using a fault injection attack. See my article in *Circuit Cellar* #346 ("Attacking USB Gear with EMFI: Pitching a Glitch" *Circuit Cellar* 346, May 2019), or my paper "MIN()imum Failure: EMFI Attacks against USB Stacks," links to both of which are available on *Circuit Cellar's* Article Materials and Resources webpage [1][2].

All of these attacks are successful because the processor executing a read (or store) instruction has no context about what the command should or should not have access to. Generally, a low-level read (or store) instruction has access to a huge range of memory. Processors may have a secure and unsecure (or privileged and unprivileged) mode that provides some bounds, but it still leaves the problem that a single flaw in the secure mode gives access to the entire secure memory space.

EVERYTHING OLD IS NEW AGAIN

In an alternate history, we never would have these problems at all. A friend introduced me to the (failed) Intel iAPX 432 processor from 1981, a processor that was built with object-oriented programming supported in hardware. *Circuit Cellar's* Article Materials

and Resources webpage includes a link to an interesting article detailing this device [3]. It's too much to cover in a few paragraphs.

The processor is described as "anti-RISC" to set the stage for what comes next. As an example of the complexity, the variable-length instructions could be from 6 to 321 bits, and didn't need to be stored byte-aligned. All this complexity did buy you a fully memory and capability-safe processor, long before people were thinking seriously about computer security.

Fundamentally, the iAPX 432 implemented the idea of instructions operating on objects. This means it was impossible to "read beyond" memory, since memory existed only for the given purpose. Like many failed good ideas, the practical implementation left much to be desired. The implementation choices resulted in such excessive performance hits that it simply wouldn't survive in the marketplace.

Forty years later, CHERI offers memory and capability-safe processors as well. But unlike the iAPX 432, it offers it in a RISC format, and with a minimal overhead. Work has been done to ensure this overhead remains small even with practical considerations, such as how the DRAM refresh cycle impacts trying to add memory tagging. This practical focus is what makes CHERI exciting (and what makes it unlike the iAPX 432)—it's not just a research project, but a complete set of tools including specification, compilers, debuggers, reference cores, and more.

TAG YOU'RE IT

When discussing the previous attacks, it often comes down to: an attacker should only be able to access a certain segment of memory. A pointer should point to an 8-byte buffer for example, but an error in the bounds check logic lets them access memory beyond

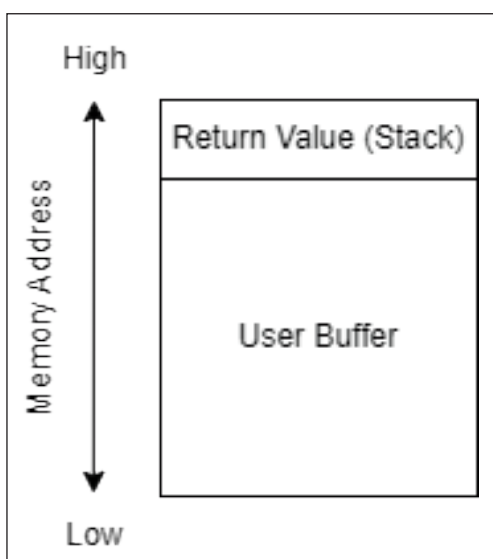


FIGURE 1
Writing to or reading from memory is a constant source of problems in embedded systems.

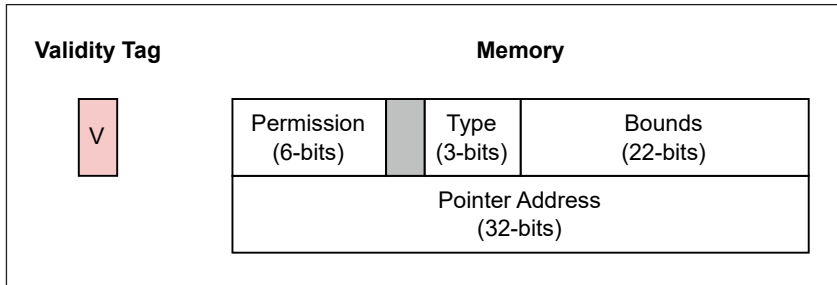


FIGURE 2
CHERI adds bounds and capabilities to memory spaces.

the end of the buffer. Or a user passes a string to a `print()` call which is missing the null, resulting in the `print()` call dumping additional sensitive data.

One way to solve this is by using memory-safe languages (such as Rust). These languages provide the memory with protection as part of the output of the compiled code, and provide language syntax to use these features.

The big downside to using a memory-safe language is it requires rewriting your code in a memory-safe language. If you have many years of legacy code to support, this can be no small feat. Instead of doing this with the compiler output, CHERI does this in hardware with tags.

The tag format in **Figure 2** shows that the pointers being referenced suddenly have a few extra fields. The inclusion of a bounds field means an attacker no longer has access to arbitrary lengths of memory. This bounds is somewhat cleverly encoded to reduce the bit-space needed, by using a “floating-point” or “logarithmic” type encoding. That means you have more precision at smaller boundary

sizes, but for larger blocks must pick the closest boundary.

The validity tag is a single bit indicating if the memory access should be used at all, and the type and permission provide additional granularity. Memory can be made read-only or disallow execution through the permissions.

The validity tag in Figure 2 isn’t shown as being in the same memory space, as it’s held in an out-of-bounds memory. Modifying this requires special instructions, which ensures that an attacker cannot simply mark invalid memory as valid. The validity tag is cleared by hardware when capabilities become invalid as well.

Beyond memory safety, CHERI enables a variety of other security features, including that:

- It makes it easy to compartmentalize your software so that tasks can only access their own memory.
- It makes it possible to easily pass pointers which allow read-only access (enforced by the core itself and not just a polite request).
- It can seal sections of memory to prevent modification.

The best way to see these in action is to look at a few examples, and I’ll use the CHERIOT-RTOS project for that.

USING CHERIOT

The CHERIOT-RTOS project is a Real-Time Operation System (RTOS) that supports CHERI features to provide a high level of security. To be clear, you don’t need to use CHERIOT-RTOS to access the security features of CHERIOT. But it provides a useful framework for experimenting with CHERIOT.

The CHERIOT-RTOS repository includes numerous examples of using the CHERI extension. These can run on the CHERIOT RISC-V core, be it an emulator or the real CHERIOT-Ibex soft core which you can program into an FPGA board. Soon this will be even easier to experiment with on the open-source Sonata Board, which includes all required debugging hardware.

I’ll bring up a simple example so you can get an idea of how the CHERI extensions work. To start with, let’s look at simply printing a few different strings. This is shown in Listing 1, and recreates the hello.cc file from the error-handling examples that are part of the CHERIOT-RTOS repository.

You’ll see this includes three calls to the `write()` function, which sends data out the UART. The implementation is shown in **Listing 2**. Note that there is no special handling at all to check the validity of the passed memory. The only call is one that



ABOUT THE AUTHOR

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

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
References [1] to [4] as marked in the article can be found there.

RESOURCES

lowRISC | lowrisc.org


```
// Copyright Microsoft and CHERIOT Contributors.
// SPDX-License-Identifier: MIT

#include "hello.h"
#include <fail-simulator-on-error.h>

// Thread entry point.
void __cheri_compartment("hello") entry()
{
    // Try writing a string with a missing null terminator
    char maliciousString[] = {'h', 'e', 'l', 'l', 'o'};
    write(maliciousString);
    // Now try one that doesn't have read permission:
    CHERI::Capability storeOnlyString{maliciousString};
    storeOnlyString.permissions() &= CHERI::Permission::Store;
    write(storeOnlyString);
    // Now one that should work
    write("Non-malicious string");
}
```

LISTING 1

The "hello.cc" file from 07.error_handling example

```
// Write a message to the UART.
void write(const char *msg)
{
    LockGuard g{lock};
    Debug::log("Message provided by caller: {}", msg);
}
```

LISTING 2

The write function of "uart.cc"

checks for a lock to prevent concurrent entry (which would be common in most RTOSs).

If a memory error occurs, a handler can capture that to print a useful debug message. But it doesn't require you to add any memory safety check. The hardware provides memory safety checking, which is the entire point of CHERI.


Going back to Listing 1, the first call to `write()` is missing the null terminator. This results in the function attempting to read beyond the allowed memory space, and it shows how CHERI can help with this common problem. The second call to `write()` shows how CHERI's capabilities give you more control over how data is used. Here the passed string doesn't actually have read capability; it's only allowed to be used for storing data. Again, the hardware prevents the `write()` function from reading from this memory.

A SUNNY FUTURE

If the examples in this column have piqued your interest, take a look at the CHERIOT repositories to see all the details of both the RTOS and core [4]. And watch the lowRISC website for more about the Sunburst Project,

which will include the open-source Sonata board to make it easy to run the sort of demos I showed in Listing 1.

When it comes to practical usage, you'll of course need CHERIOT implemented in some physical MCU. Right now, the answer to that isn't as clear—I don't know of any commercial MCUs planned with CHERI support. But hopefully, with a few more accessible examples, we'll see it get picked up. But the soft-core CHERIOT-Ibex that is currently available has the advantage of not locking you into a specific configuration.

CHERI is an exciting technology to me because it doesn't have to be turned on all at once. If you have a CHERI-enabled MCU, you can use your existing code almost as-is. From there, improving the security can be done in stages by adding in the additional features to your code. To me this is the main advantage of CHERI, and why it has a higher chance of finding commercial relevance. It doesn't require you to rewrite your entire codebase at once. For better or worse, it might give all that memory-unsafe C/C++ code another lease on life in a world where more people are demanding security by design. 

From the Bench

Cellular, The Forgotten Wi-Fi

Part 3: Using NoteCard, an Embedded Communications Module

In Parts 1 and 2 of this series, Jeff introduced us to a novel method of providing light in a meeting place located in a remote area without electricity, by using a low-voltage solar energy system. Last month he described the use of Wi-Fi for communicating the system's performance. In Part 3, he uses NoteCard, a cost-effective embeddable cellular module for sending data to his home from a remote area.

By
Jeff Bachiochi

When I got my first Motorola flip cell phone, it was like Star Trek come alive. The Star Ship Enterprise's communicator was similar in size. I installed a sound clip to imitate the familiar opening of the communicator when I opened my Star Tac flip phone. In the beginning you couldn't call to the next town with out incurring toll changes. When you left your cell network coverage, there were extra roaming charges. Today we have unlimited calling, text, and data plans. Streaming your favorite movie/series requires a rather large bandwidth of streaming data. Everything today is data-oriented.

In my September 2023 article ("Local Isolation: Using the Sun's Energy," *Circuit Cellar* 398, September 2023) [1], I described a low-voltage solar energy system I built for lighting my Scout troop's large storage shed. I described the battery I chose, and the Modbus protocol with RS-485 serial communication used for remote monitoring of the system's performance ("sniffing" or listening to

communication on the bus, and decoding each message). Then, in my October article ("Local Isolation Using the Sun's Energy: Part 2: Modbus Client," *Circuit Cellar* 399, October 2023) [2], I expanded on this node's ability to do more than just listen and decode. This month, I discuss my use of a modern module, to make use of cellular networks for communication.

I initially considered repurposing an older cell phone to make a connection with the "Eagle's Nest" solar project. However, the monthly fee for that would be greater than \$100 a year. While that would certainly make a worthy project, there are much less expensive options today. Like the Wi-Fi transceivers that allow us to connect to a wireless network at home, modules are now available to make use of the cellular networks that are growing in connectivity everywhere. The data bandwidth we need for most applications is extremely low. We can therefore take advantage of the cellular system without the steep monthly charges.

RED, WHITE, AND BLUES

One of the companies that provides this service is Blues Inc. [3]. Let's start with development tools. These are divided into two parts: the "Notecard" and the "NoteCarrier." The Notecard (**Figure 1**) is an embeddable communications module; it has all the necessary hardware to handle global cellular communications over LTE-M, NB-IoT, or Cat-1. The NoteCarrier is a prototyping PCB that contains a socket for the Notecard and for a microcontroller (MCU).

While you can put a NoteCard on your own circuitry and save some more cash, I will take advantage of the NoteCarrier, because it supports a 24-pin Adafruit Feather breakout header. You'll recall that I have been using the Adafruit Huzzah 32 module for this project since day one! If you want to use this with other MCUs, there are breakout points for every necessary connection to the NoteCard. **NOTE:** The newer starter kits feature a Feather-compatible MCU, the Swan, based on an ARM Cortex-M4.

The NoteCard can provide communications via one or more of these data networks, LTE-M, NB-IoT, GPRS, LTE Cat-1, and WCDMA. I am using the LTE CAT-M NoteCard for North America, which is perfect for medium-throughput applications requiring low power, low latency, and/or mobility, such as asset tracking, wearables, medical, POS and home security applications. In my application, I have power via solar, but no Internet. Cell coverage in the area makes this solution possible. Let's see how the NoteCard/Carrier combination is used.

To complement this hardware, Blues Inc. has a cloud-based site ("NoteHub") that communicates with the NoteCard and syncs data both to and from the NoteCard. Last month's project ended with our Modbus

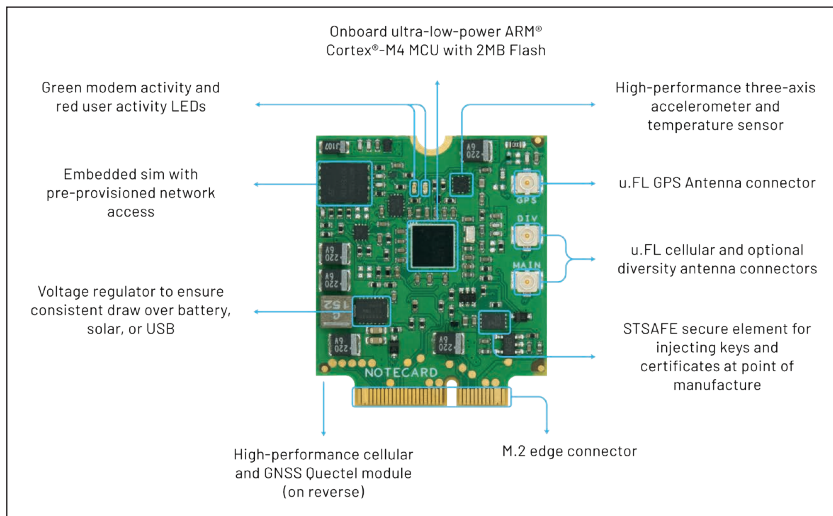


FIGURE 1 The NoteCard contains a complete cellular and GNSS (Global Navigation Satellite System) you can use in your own circuitry or in a NoteCarrier via the M.2 high-density (0.5mm pitch) edge connector.

data sent via Wi-Fi to a MQTT server, using JSON objects [2]. At the time, you may have been thinking that there may have been better ways of sending this data. That was actually a setup for this month's use of the cellular service. A JSON object is simply a way of passing data in a readable form, as in {solarPower:4}. While an object can contain multiple messages separated by commas, this single message indicates the variable solarPower is equal to a value of 4. On the NoteCard end we have functions that can be called to pass and retrieve data. Data from NoteCard is placed into a NoteFile (.qo extension) on NoteHub, and once received is added to a NoteFile with a .db extension. Data from a NoteFile with a .qi

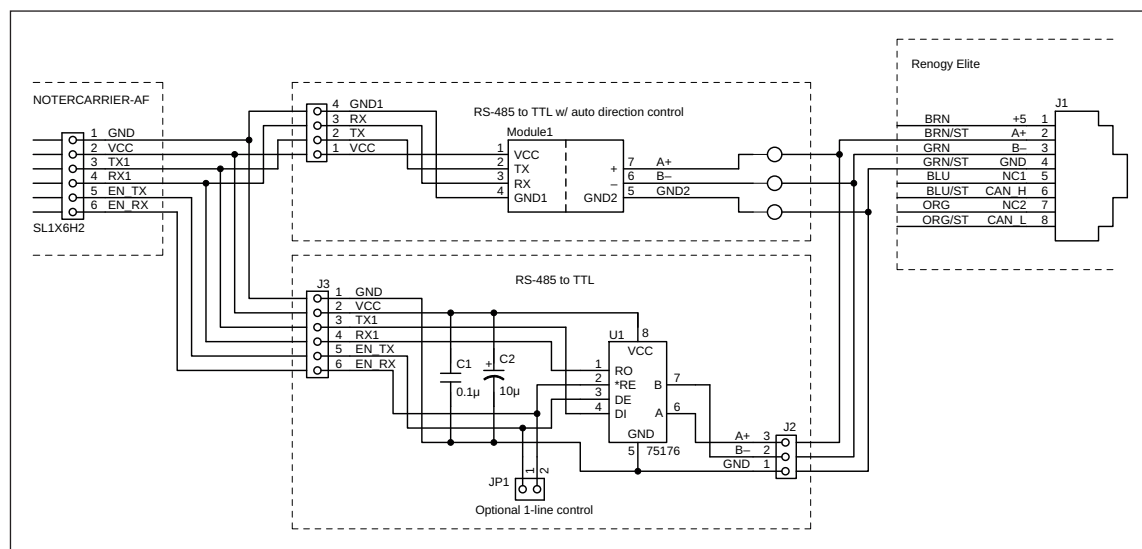


FIGURE 2 Only 6 (4) connections from NoteCarrier are required to interface with the RS-485 converter. The solar energy system used Modbus (using RS-485) for inter-module communications.

COLUMNS

extension on NoteHub is sent to the NoteCard on any sync.

One powerful feature of NoteHub is event routing, which allows one to forward data from NoteHub to a public/private cloud, including AWS, Azure, Google Cloud, a messaging platform like MQTT, or a custom HTTP/HTTPS endpoint. Note that MQTT is supported. Let's take a step back and look at how our solar project is now configured to use the NoteCard.

NOTECARRIER

I'm using a NoteCarrier-AF for this project. The present offering, the NoteCarrier-F, is just a slight variation of this, it uses an external antenna, for instance. The Huzzah32 plugs into the Feather socket. We can continue to use the RS-485 converter described the initial project column [1]. This only needs a 6-wire connection to the NoteCarrier-F. Note that the schematic (**Figure 2**) consists of adding the RS-485 chip to the NoteCarrier, so that we can make the Modbus connection to the Eagle Nest's solar system.

Since I am using the Huzzah Serial1port for Modbus and the serial USB port is being used as the debug port, I will be using the I²C interface to the NoteCarrier/Card. Either serial or I²C can be used to communicate with the NoteCarrier-F. All connections between the Note Card and my Huzzah 32 are made through the NoteCarrier. I just added the six connections to my RS-485 circuit module [1].

You will need to install the note-arduino library available on Blues GitHub page [5]. This library gives the user about a dozen functions. For this project we'll only use six. Like many libraries, one of the first functions you place in the `setup()` function is `Notecard.begin(0x17, 0x20, Wire)`. The parameters for this function define whether the NoteCard interface will be via a serial port or I²C; here the setup is for the I²C interface. Once I've started the primary serial port with `Serial.begin(115200)`, I can command the Notecard to use this port for debug messages with `Notecard.setDebugOutPutStream(Serial)`.

All communication between our micro and the NoteCard will be JSON objects. We've already used JSON objects in our Arduino code to identify a register source and its value. Objects are a group of one or more members (name:value pairs), surrounded by braces, `{}`. An object with multiple members are separated by a comma.

The purpose of the NoteCard is to form a communication path between our micro and the NoteHub, which is the cloud-based receiver of our cellular data. We need to inform NoteHub who we are and how we will communicate with it. This is accomplished by creating a JSON object. A `hubRequest` command is used to configure and monitor the connection between the Notecard and Notehub.

```
J *req = Notecard.newRequest("hub.set");
```

will initialize this object with a member identifying the command.

Next, we can add some JSON members to the object. The second member identifies us and ties us to our specific NoteCard, `JaddStringToObject(req, "product", myproductID)`. The string variable, `myproductID`, will be discussed further in the NoteHub section to follow. Next we'll configure how the communication channel will be used by defining the "mode", `JaddStringToObject(req, "mode", "continuous")`. There are basically two modes: "periodic" (default) and "continuous". Periodic is used to keep NoteCard in a low power mode (μ A) and connect with NoteHub periodically. The continuous mode (tens of mA) keeps a communication path open for immediate transfer of data. In either mode, the current draw is 250mA when the modem is active. I'll be transferring data once per minute.

We have a complete command (JSON object) built now, and it looks like this:

```
{
  "req" : "hub.set",
  "product" : myProductID,
  "mode" : "continuous"
}
```

We are now ready to send this JSON object to the NoteHub. The `sendRequest` function handles this. Initially, we are coming out of a power-up, and we don't know if the NoteCard has finished all of its initialization (it has its own application to execute). If we ask it to send a request to NoteHub before it is ready, it could fail. We can either check for that or use the `sendRequestWithRetry` function. I'll

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@imaginehatnow.com, or at: www.imaginehatnow.com.



let the NoteCard handle any error and use the function `sendRequestWithRetry(req, 5)`. This will retry the send command every 5 seconds.

Let's take advantage of the NoteCard's time service. Once it has completed a sync with NoteHub, it will have a reference epoch (number of seconds since January 1, 1970.) This requires a second JSON object, one that holds the response from the NoteCard. The command `J *rsp = Notecard.requestAndResponse(Notecard.newRequest("card.time"))` is used to initialize the `rsp` object to the response of the function. The NoteCard's `card.time` function returns an object containing multiple members. We want the member "time" : reference epoch, where reference epoch is a number (signed 32-bit integer). This number can be easily extracted from the object using the command `myEpoch = J.getNumber(rsp, "Time")`. After we have what we want, we can delete the `rsp` object from the NoteCard with `Notecard.deleteResponse(rsp)`. I adjust `myEpoch` for my local time zone. I'm using the `TimeLib.h` to handle the time on the Arduino.

We're now finished with all the necessary NoteCard initialization in the Arduino's `setup()` function, so it's on to the `loop()` function where we have previously gathered data via the Modbus connection to the solar system. We don't have a lot of things to do in our `loop()` function. We do need to decide how often to request data from any of our devices in the solar system. The variable `sampleDelay` constant determines this timing, and our

function `requestRegisters()` asks for 20 registers in the solar controller, starting with register address `0x100`. Besides requesting data in a timely fashion, our loop also checks for any activity on the Modbus (excluding any requests we make). If we have activity, hopefully a response to our request, we call the `mbProcess()` function to dissect the response. If this response was due to our request, the data is placed into the appropriate group of defined variables. Our previous programs have initialized local holding registers for every piece of data we might be interested in from any of the devices connected on the Modbus. For this program we are interested in just 20 of these.

With data collected, we are now ready to have this data transferred via cellular communication to the `notehub.io` cloud. We'll use the now familiar `newRequest` function to handle this. The last request was for the `hub.set` command, now our JSON object will use the `note.add` command, `J *req = Notecard.newRequest("note.add")`. We'll add a `sync` member to the object, `JaddBoolToObject(req, "sync", true)` to force the NoteHub to handle this immediately. Then all our data is added as additional members, which will all be sent in one fell swoop (**Listing 1**).

When we've added all the data we want to include, we issue a command to send the data, `Notecard.sendRequest(req)`. You may have noticed that the member data was added to "body" and not to "req" as in our `hub.set` command. This alters the JSON object by adding a member "body" that is

```
JAddNumberToObject(body, "batterySOC", batterySOC);
JAddNumberToObject(body, "batteryVoltage", batteryVoltage);
JAddNumberToObject(body, "batteryCurrent", batteryCurrent);
JAddNumberToObject(body, "controllerTemperature", controllerTemperature);
JAddNumberToObject(body, "batteryTemperature", batteryTemperature);
JAddNumberToObject(body, "loadVoltage", loadVoltage);
JAddNumberToObject(body, "loadCurrent", loadCurrent);
JAddNumberToObject(body, "loadPower", loadPower);
JAddNumberToObject(body, "solarVoltage", solarVoltage);
JAddNumberToObject(body, "solarCurrent", solarCurrent);
JAddNumberToObject(body, "solarPower", solarPower);
JAddNumberToObject(body, "batteryCumulativeChargeHours", batteryCumulativeChargeHours);
JAddNumberToObject(body, "batteryCumulativeDischargeHours",
batteryCumulativeDischargeHours);
JAddNumberToObject(body, "solarCumulativePowerGenerated", solarCumulativePowerGenerated);
JAddNumberToObject(body, "solarCumulativePowerConsumed", solarCumulativePowerConsumed);
```

LISTING 1

This is where each variable is added to the JSON object body. Each member will contain the variable name and its value.

```

{
  "req": "note.add",
  "sync": true,
  "body": {
    "batterySOC": 0,
    "batteryVoltage": 0,
    "batteryCurrent": 0,
    "controllerTemperature": 0,
    "batteryTemperature": 0,
    "loadVoltage": 0,
    "loadCurrent": 0,
    "loadPower": 0,
    "solarVoltage": 0,
    "solarCurrent": 0,
    "solarPower": 0,
    "batteryCumulativeChargeHours": 0,
    "batteryCumulativeDischargeHours": 0,
    "solarCumulativePowerGenerated": 0,
    "solarCumulativePowerConsumed": 0
  },
  "crc": "0002:98F82C2C"
}

```

LISTING 2

You can see a copy of the complete JSON object that's sent via the debug serial output on the HUZAH32 micro. Once connected to the solar system's modbus, these values will be pulled from the solar controller.

an array of the data. Because I enabled the debug port, we can see this sent to the USB serial port (Serial). See **Listing 2**.

Note the JSON object contains four members: "req," "sync," "body," and "crc." The member "body" has an array of members associated with it. The "crc" member ensures that the data is transferred without error.

So far in this project, I've collected data from the solar system installed in Troop 96's equipment shed, which is off the grid. Because we have no way of monitoring the system, except for a Bluetooth app on my cell phone—which only works while I'm near the shed—a cellular connection was added. My MCU, which collects the data, is interfaced with a Blues Inc. NoteCard, which is now sending the collected data to NoteHub.io, Blues' cloud service [4]. Let's now see what NoteHub.io does with this data.

NOTEHUB

When you purchase a NoteCard, you get hardware that you can immediately power up, and follow an online tutorial on how to set up your NoteCard and connect to it from the web browser. Open the Blues.io webpage for NoteCard quickstart and NoteCarrier-F [6]. You will get an error message if your browser is not supported. A supported web browser (

The screenshot shows the Blues.io website's 'NoteCard Quickstart' page. The main content area contains instructions for connecting the NoteCard to a computer, including steps for Mac, Windows, and Linux. Below the instructions are two buttons: 'Having trouble connecting?' and 'Prefer to use the NoteCard CLI instead?'. The 'Validate Serial Connection' section provides a copyable JSON request and a terminal window showing the successful connection and the resulting JSON response. The terminal window shows the following output:

```

~ Connected to serial
~ DeviceUID dev:864475644208469 (NOTE-NBGL500) running
firmware 5.1.1.16026
> {"req": "card.version"}
{
  "version": "notecard-5.1.1.16026",
  "device": "dev:864475644208469",
  "name": "Blues Wireless Notecard",
  "sku": "NOTE-NBGL500",
  "board": "1.11",
  "api": 5,
  "body": {
    "org": "Blues Wireless",
    "product": "Notecard",
    "target": "n5",
    "version": "notecard-5.1.1",
    "ver_major": 5,
    "ver_minor": 1,
    "ver_patch": 1,
    "ver_build": 16026,
    "built": "Apr 3 2023 11:04:31"
  }
}
> {"req": "card.version"}

```

FIGURE 3

The NoteHub Quick Start Tutorial will guide you to quickly get connected using your NoteCard/NoteCarrier without the need to write any code. This is a great way to get some hands-on experience with the hardware before you begin writing your application.

Chrome, Opera, or Edge) will connect directly to the NoteCarrier USB port and manipulate the NoteCard directly with it, as if you had a MCU attached to it. This initial connection with the USB cable is used for the tutorial to make it easy to experiment with the NoteCard/NoteCarrier kit right out of the box.

When you connect the NoteCarrier via USB to your PC, it should be recognized as a new serial port connection. You can now connect to it, and you should see some messages just under the green bar in the upper right of the screen (**Figure 3**). The NoteCard is directed to make contact with NoteHub. Once its sign-in is complete, you should see a sign-on message:

Welcome to the NoteCard In-Browser Terminal.

Start making requests below.

(For advanced info, use the 'help' command.)

~ Connected to serial

~ DeviceUID dev:864475044208469 (NOTE-NBGL500) running firmware 5.1.1.16026

Your NoteCard's UID is shown along with the revision of the firmware running on the card. Now you can enter commands. The quickstart suggests entering the following command:

```
["req": "card.version"]
```

You can copy this into the input terminal at the lower right. Click on ">" to send the request. You'll note that the response is in JSON format, and the info is the same as the sign-on message.

New users are required to set up an account on NoteHub. Under the heading "Set up Notehub," see "Create a Notehub Project" and click on "Notehub Project Dashboard." The quickstart guides you through this by opening a new browser window (NoteHub.io), where you can enter your name, email address, and a password. When you have completed this, you are ready to create a project. Click on "+Create Project" and add a project name such as "quickstart" to the New Project card. Note that you are given an account and a UID prefix. The product UID becomes the product UID prefix plus the project name. Remember that earlier, we initialized a variable `product_UID` in our Arduino sketch? This project's "productUID" is what you will use to define that `product_UID`, so your application will be associated with the NoteCarrier/NoteCard, the NoteCard/NoteHub communication channel, and this project on NoteHub. When you're happy with the information, click on "+Create Project"

and your browser will go to your project page. Make note of the "ProductUID" here, and with it you can continue with the quickstart by entering the following command into the input terminal:

```
["req": "hub.set", "product": "com.your-company.your-name:your-product"]
```

Then substitute your `productUID` for "com.your-company.your-name:your-product" in the right-hand member of the JSON pair. When the command is sent you will see the command and response in the output terminal. A no-error response is sent as "{}". You can switch back to the Blues.io tab and go on with quickstart entering commands including:

```
["req": "note.add", "body": {"temp": 35.5, "humid": 56.23}]
```

At this point, we've requested the NoteCard to send some data to our project in NoteHub. Go back to the NoteHub.io tab and click on "events." You should now see a list of events or communicates you requested via the NoteCard. Select an event and click "view" to see the event data. The last command sent was typical data. This is saved as an event in the .qo file. If you select that event and view it, you will see this event's data under the "Body" tab (**Figure 4**).

You should investigate all the different screens available on your project's page, and become familiar with how things are presented, because we will be coming back to this shortly. You should also experiment with sending other commands using the terminal in the Blue.io window.

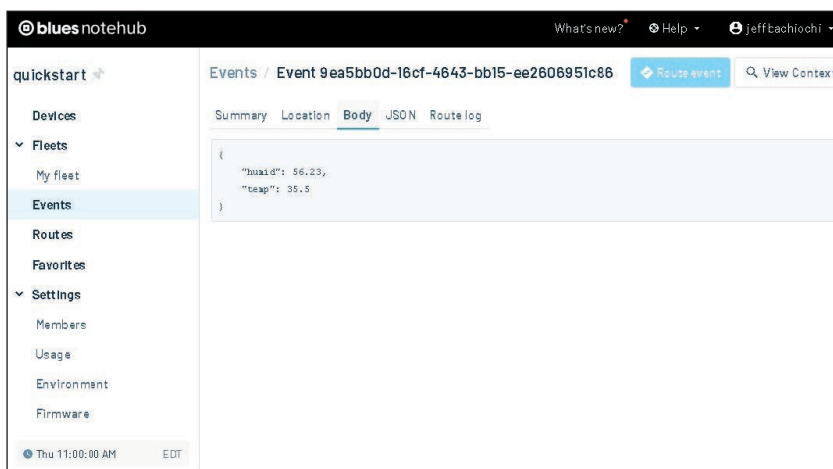


FIGURE 4

The tutorial shows you how to send data from your NoteCard/NoteCarrier to NoteHub and see the actual data arrive as an event.

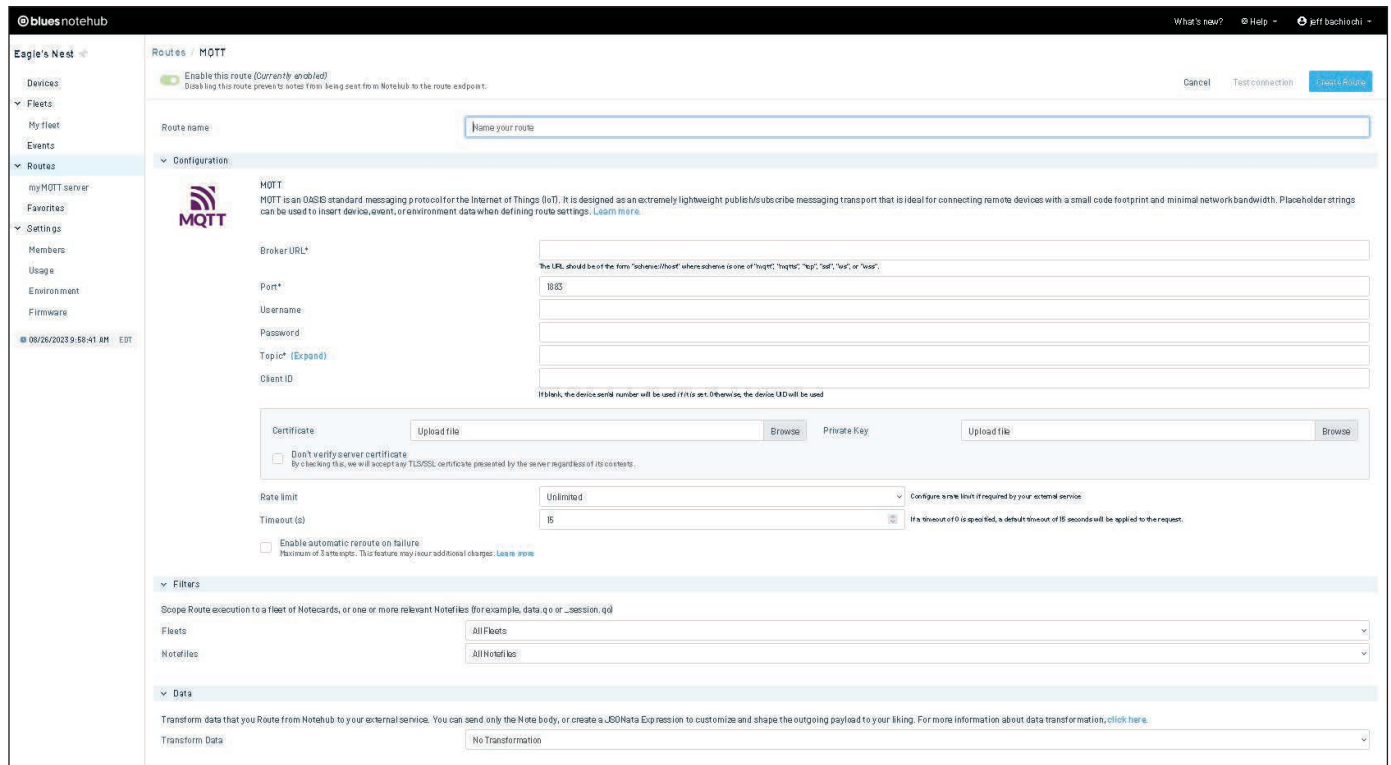


FIGURE 5

My application data wants to eventually end up being sent to my Raspberry Pi, running an MQTT server. NoteHub can be used to route events to an external data sink. My choice is MQTT, and NoteHub makes it happen by answering a few questions.

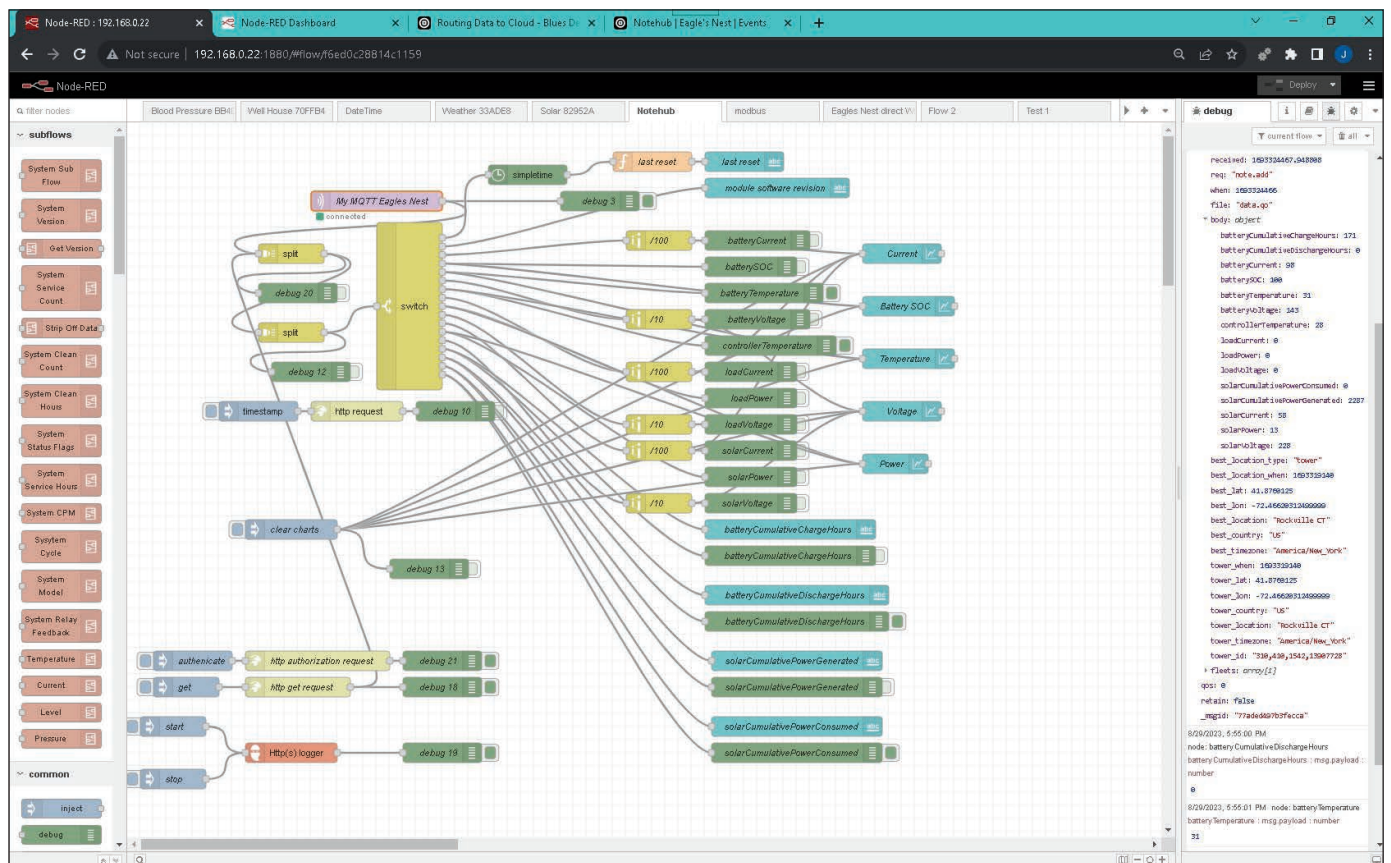


FIGURE 6

Events are routed by NoteHub to my Raspberry Pi. I'm using Node-RED to take each MQTT packet sent and split the JSON object into individual members.

THE REAL THING

We have enough information to proceed with our project. My Huzzah 32 MCU is chomping at the bit to be programmed with this latest application and be placed on the NoteCarrier's "Feather Headers." We discussed the additions to an application to support NoteCard/NoteHub. These have been added to our project from last month [2]. The use of Wi-Fi last month, as part of the ESP32 MCU, is now replaced by the NoteCard's cell service communication. Using the Arduino IDE, this new application can now be programmed into the micro. Upon power up, the Huzzah's application makes itself known to the NoteCard. This NoteCard (NBNA-500) uses a Quectel BG95-M1 modem, which covers the LTE_M Data Networks in United States, Canada, and Mexico. (See other models for global coverage.)

Our product on Notehub.io is receiving our data periodically, based on the constant `reportDelay`, once an hour. However samples are taken periodically, based on the constant `sampleDelay`, here every 10 seconds. Each sample is saved along with a running total, so that when reported once an hour, the average sample is reported. The equation for the first register of interest, `batterySOC`, is:

```
batterySOC = batterySOCTotal /
sampleCount;
```

This is handled in the `averageTotals()` function prior to adding the data members to the "body" object. A final command, `notecard.sendRequest(req)`, will request our NoteCard to send our data to NoteHub.

Assuming this has happened and it has arrived as an event, we need to get the events routed to the Raspberry PI installed on my home network. Click on the route tab and then the "+Create Route" button. Here is a list of the routing destinations you can choose from. Each has a tutorial to help you if necessary.

- HTTP/HTTPS endpoints
- AWS
- Azure
- Google Cloud function
- MQTT
- PROXY
- ThingWorx
- RadResponder
- snowflake
- twilio
- Edge Impulse
- slack
- amazon S3
- Datacake

In this case, I am going to choose MQTT. This leads to the MQTT configuration page in **Figure 5**. Here you can give your route a name. Only two other items are required, Broker URL, and Topic. The Raspberry Pi is on my local WAN, so I can go to my Modem/Router and find my actual IP address. While this is not a static IP, I found it is not likely to change unless you change service provider. You can always use a dynamic DNS (Domain Name Server), if you want to have a static name served to your dynamic IP, like No-IP.com

Topic defines the topic for each messages routed (forwarded to my Pi); the payload will be your event data. I use the `[product]/[device]` which substitutes my ProductUID "/" DeviceUID as the topic. These can be found on the Device tab by double clicking on the appropriate device. You'll need this for the MQTT IN node in your Node-REDflow [7]. Last month I used the Huzzah's Wi-Fi and showed how, if your system was in range of Wi-Fi service, you could send the data directly to the MQTT server. There is little to change to that Node-RED flow. The difference is in how the data comes in.

In the Wi-FI project [2], each JSON object contained one member (name/value pair) per packet. NoteHub will route the whole event, consisting of multiple members (including more than just the data), in one packet. You can see most of this in Node-RED in the debug panel on the right of **Figure 6**. We just need to change the Topic in the MQTT In node to the one used in the NoteHub MQTT setup. Because

The screenshot shows the 'blues notehub' interface. On the left, a navigation menu includes 'Devices', 'Fleets', 'Events', 'Routes', 'Favorites', 'Settings', 'Members', 'Usage', 'Environment', and 'Firmware'. The 'Events' section is selected, showing a list of events. The selected event is 'Event 978ecf97-9afd-4daa-8ae8-4d868fcd0be0'. The 'Body' tab is active, displaying a JSON object with 20 key-value pairs representing sensor data from a solar controller. The data includes battery status, temperature, and solar power metrics.

```
{
  "batteryCumulativeChargeHours": 171,
  "batteryCumulativeDischargeHours": 0,
  "batteryCurrent": 98,
  "batterySOC": 100,
  "batteryTemperature": 31,
  "batteryVoltage": 143,
  "controllerTemperature": 28,
  "loadCurrent": 0,
  "loadPower": 0,
  "loadVoltage": 0,
  "solarCumulativePowerConsumed": 0,
  "solarCumulativePowerGenerated": 2287,
  "solarCurrent": 58,
  "solarPower": 13,
  "solarVoltage": 228
}
```

FIGURE 7

Each of the 20 variables pulled from the Solar Controller via the Modbus and recorded by the Huzzah 32 are transferred to the NoteCard and sent to NoteHub. There you can see an event's body contains 20 members (name/value pairs).



FIGURE 8

Once Node-RED has received the MQTT JSON packet and split it into individual members, each member is sent to the appropriate chart or textbox for display on the Node-RED Dashboard.

each event sent to NoteHub from NoteCard contains all our data in a single packet, the event is routed to our MQTT server all at once. If you viewed an event on NoteHub (events, click on an event, then view), you can choose "body", to see the data members (**Figure 7**), or "JSON" to see the complete event information. Note that the JSON object contains multiple members (name/value pairs), one of which is "body." This member has an object as its value. The "body" object contains multiple members, our data

All of this is sent to the MQTT server, so each event will need to be disassembled into separate messages. The first split node divides the single message into multiple messages. If enabled, Debug 20 will show the separated messages in the debug panel on the right. They are "event," "session,"

"best_id," "device," and so on. Note that the message "body" contains all of our data. The second split node divides the "body" message into separate messages, as can be seen using debug 12. Now we have a whole lot of individual messages, just like the ones sent using Wi-Fi in last month's project. The "switch" node routes only those that are our data messages to separate outputs, where they can be massaged and displayed on the Node-RED dashboard (**Figure 8**).

COST OF SERVICE

Today you can get a starter kit for North America for \$99 or the EMEA (Europe, Middle Asia, and Africa) starter kit for \$109 from Blues.io. This includes NoteCard, NoteCarrier, Swan (Feather-compatible micro), and 10 years of cell service! The cell service includes 500MB of free cellular data and 5,000 free Consumption Credits. Yup! For about a hundred bucks you can get started with a cellular connection that will give you up to 10 years of service. Renewals and extra data are equally valued.

So how can Blues make any money on this stuff? Naturally, offering a development kit and service at such a low cost removes any initial hesitation to try out their hardware. Their real income will come from recurring revenue. This is measured by cellular data limit and the consumption credits (CC). Each of my events is about 1.5KB. If we divide the

Additional materials from the author are available at:

www.circuitcellar.com/article-materials

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

RESOURCES

Adafruit | adafruit.com

Arduino | www.arduino.cc

Blues, Inc. | blues.io

Node-RED | nodered.org

Renogy, Inc. | renogy.com


cellular data cap by my event packet size we get $500,000,000B/15,00B = >300,000$ messages. If I sent this every minute, that would be ~ 200 days. Cellular data and CC are renewed each month, so I should never run out. You can think of consumption credits as NoteHub requests, of which an event route costs 1 credit, that would be 5,000 routes. Again, if we send an event every minute, that would be $60 \text{ minutes} * 24 \text{ hours} * 31 \text{ days} = 44,640$ routes per month. That is a bit over our 5,000 CC budget. 40,000 extra CC's cost about \$30. If you limited the routes to every 10 seconds, you would stay under budget. You'll notice I'm only updating things once an hour so that's only $24 \text{ routes/day} * 31 \text{ days} = 744$ routes/month. It's easy to see that in some applications you may need to purchase additional credits.

UNCOVERED

There are so many things that I have not covered here. I think if you are at all intrigued by this, you should visit the Blues website and look a little deeper. I'm not sending any data to the NoteCard from my Pi. This cellular solution is closed loop, and can be used for both monitoring and control. So I could be using the cellular connection to say, turn ON/

OFF the power to the lighting units, but at this point there is nothing I want to control at the shed. You'll be fascinated with some of the applications listed on the Solutions tab of the Blues website.

If you are contemplating any kind of project that won't have a permanent home on some Wi-Fi network, you should consider using cellular communication. You don't have to fool around with connection requirements of SSID and security key each time you move to a new location. You may have noticed that one of the object members returned in the MQTT packet is the longitude and latitude of the cell tower picking up our cellular transmission.

Once you have learned the fundamentals by following the tutorials, you will find it easy to tack this cellular system onto any project. I'm excited to get to use the new solar lighting system in our Troop's shed. We're starting up regular meetings again after the summer break. Our first meeting is this Monday night, and with the sun beginning to set earlier each night, having adequate lighting will surprise everyone. Everything is in place just in time. I'll be adding weekly meetings and monthly camp outs to my fall schedule. Yep, there is too much to do and so little time. 



2nd Generation DC-DC Down Converter

The 700DNG40-24-8 is a compact and lightweight 4kW liquid-cooled DC-DC converter known for its exceptional efficiency and reliability. It is specifically designed to support the DC voltage needs of hybrid and electric vehicles, making it ideal for powering various low-voltage accessories in these vehicles.

belfuse.com/power-solutions

PRODUCT NEWS by Kirsten Campbell

Saelig Debuts Economical 12-bit Rigol DHO 800/900 Oscilloscope Series

Rigol's newest high-performance 12-bit economical digital oscilloscopes are portable and offer high-resolution, a capture rate up to 1,000,000wfms/s, up to 50Mpts memory depth, and an ultra-low noise floor that allows the detection of even small signal details.

The Rigol DHO 800/900 Oscilloscope Series supports 16 digital channel capture, allowing analysis on both analog and digital signals simultaneously to meet complex embedded design and test tasks. Affordably priced, these scopes provide auto serial and parallel bus analysis, Bode plot analysis, and



many other functions needed for today's test demands in R&D, education, and scientific research.

DHO800 Series Highlights

- Ultra-low noise floor, pure signal depiction, captures small signals
- Up to 12-bit resolution for all the models
- Analog bandwidth of 70MHz & 100MHz, 2 & 4 analog channels
- Max. real-time sample rate 1.25GSa/s
- Max. memory depth 25Mpts
- Vertical sensitivity range: 500 μ V/div to 10V/div
- Max. capture rate of 1,000,000wfms/s (in UltraAcquire mode)
- Digital phosphor display with real-time 256-level intensity grading
- Waveform search and navigation function
- 7" (1024x600) capacitive multi-touch screen
- New user-friendly Flex Knob control
- USB Device & Host, LAN, and HDMI interfaces (std.) allows remote control
- Additional DHO 900 Series Highlights
- 16 digital channels (std. but logic probe purchase is required)
- Max. real-time sample rate of 1.25GSa/s
- Max. memory depth 50Mpts
- Vertical sensitivity range: 200 μ V/div to 10V/div

Saelig | saelig.com

ARTERY Introduces Its First Automotive-Grade MCU to Power Next-Generation Vehicles

ARTERY Technology launched its first automotive-grade microcontroller AT32A403A. Certified according to automotive standard AEC-Q100 Grade 2, and passed qualification tests including accelerated environmental stress test, accelerated lifetime simulation test, package assembly integrity test and electrical verification test.

Automotive-grade chips have the top priority of ensuring driving safety must pass a whole set of qualification tests for automotive applications according to the industry standard specification developed by Automotive Electronics Council (AEC).

AT34A403A—AT32A403A series is based on ARM 32-bit Cortex-M4 core that embeds up to 1MB Flash memory and 224KB SRAM, operating at a frequency of up to 200MHz. Powered by 2.6-3.6V voltage and can operate in a wide operating temperature range from -40°C to 105°C, meets the high computing power, high stability, and high reliability requirements of automotive electronics.

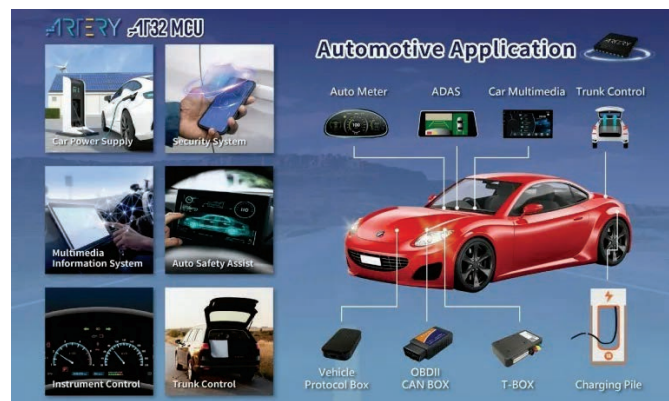
AT32A403A integrates rich peripherals to enhance the connectivity of diverse communication interfaces. It features 8x UARTs, 4x SPIs, 3x I2Cs, 2x I2Ss, 2x SDIOs, XMC, 2x CANs, SPIM, USB 2.0 FS interface supporting Xtal-less mode, 8x 16-bit general-purpose timers, 2x 32-bit general-purpose timers, 2x 16-bit motor control PWM advanced timers with dead-time generator and emergency brake, 2x 16-bit basic timers to drive DACs, 3x 12-bit 2 Msps A/D converters with up to 16 channels. Almost all GPIO ports are 5V tolerant. The AT32A403A series is especially suitable for IoT applications, leading to higher reliability and lower cost in terminal products.

Intelligent Application of Automotive MCUs: Along with the development of smart cars, on-board displays have been digitalized to realize human-vehicle interaction with gesture controls and even voice controls, instead of traditional mechanical buttons. With powerful on-chip resources, higher integration,

cost-effectiveness, and safety functions adhering to automotive-grade MCU standards, the AT32A403A series is considered the preferred choice for automotive applications such as advanced driver assistance systems (ADAS), automotive body control, anti-theft security devices, digital dashboard, automotive motor power supplies, automotive lighting, and automotive battery management system (BMS). It can even be applied to the in-vehicle infotainment (IVI), Shy Tech and AR HUD to enrich audiovisual experience and intelligent human-machine interaction.

Artery Technology is committed to creating automotive grade MCUs that are in line with the development trend of intelligence and digitalization and meet high performance and high safety standards. AT32A403A currently supplies a total of 12 models in 4 different packages and 3 kinds of Flash memory architecture to create highly reliable automotive MCU solutions and accelerate the popularization of intelligent new energy vehicles. ARTERY will also continue to expand the automotive microcontroller market, and provide customers with better services and richer products.

ARTERY Technology | arterytek.com



NEW PRODUCT SUBMISSIONS— E-mail: product-editor@circuitcellar.com

PRODUCT NEWS

Sheba Microsystems Launches Revolutionary MEMS Autofocus Actuator for Active Athermalization in Embedded Vision Cameras

Breakthrough μ Pistons technology uniquely solves decades-long embedded vision camera industry's problem of lens thermal expansion. This novel product unlocks unparalleled resolution and consistent high-quality imaging performance for automotive, action, drone, mobile robotics, security and surveillance, and machine vision cameras.

The first-of-its-kind solution tackles the long-standing industry problem of embedded vision cameras' inability to maintain image quality and focus stability during temperature fluctuations as optics undergo thermal expansion.

While smartphones use autofocus actuators and electromagnetic actuators including voice coil motors (VCMs), these actuators are unreliable for achieving active athermalization in embedded vision cameras due to extreme environmental conditions. Embedded vision camera optics are also 30 times larger than smartphone optics. Other autofocus systems in-market such as tunable lenses lack thermal stability and compromise optical quality.

"MEMS actuators are fast, precise, and small in size, and are actually uniquely suited to solve thermal expansion issues, because they are thermally stable and maintain consistent performance regardless of temperature changes," said CEO and co-founder Dr. Faez Ba-Tis, PhD. "Because of these known advantages, there have been previous industry attempts at incorporating MEMS actuators into cameras, but because they failed drop tests they were quickly abandoned. Sheba's new design solves for all of these previous blockers, which opens up limitless possibilities for embedded vision camera innovation."

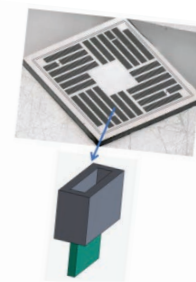
Sheba's proprietary technology compensates for thermal expansion by uniquely moving the lightweight sensor, instead of moving the lenses. The silicon-based MEMS actuator platform actuates the image sensor along the optical axis to compensate for thermal expansion in the optics. The weight of the image sensor represents only 2-3% of the optical lens weight, which makes it easier to handle, enabling ultra-fast and precise autofocus performance even when temperatures fluctuate.

Sheba's novel piston-tube electrode configuration takes advantage of a larger capacitive area, allowing for substantial stroke and increased force. Sheba's μ Pistons design makes the MEMS actuators uniquely resilient against severe shocks, since the electrodes are well-supported and interconnected.

Sheba's new MEMS actuator has successfully passed drop tests as well as other reliability tests, including thermal shock, thermal cycling, vibration, mechanical shock, drop, tumble, and microdrop tests. It is also highly rugged, which helps maintain image focus during high shocks in action cameras or machine vision environments.

Sheba's MEMS actuator offers lens design flexibility and is suitable for near and far-field imaging. It is easily integrated into existing systems and scaled up on mass production tools for automotive, action, drone, mobile robotics, security and surveillance, and machine vision cameras.

Sheba Microsystems | shebamicrosystems.ca

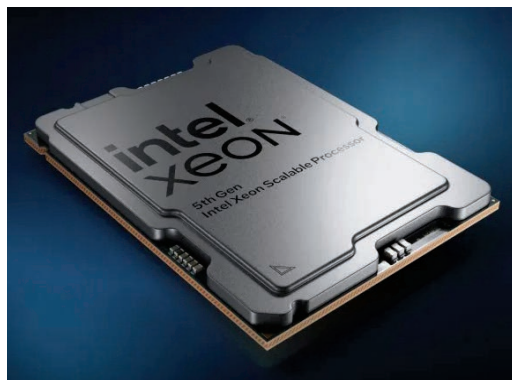


Intel Reveals New 288-Core Sierra Forest CPU, Specifically Designed for High-Density Servers

Intel has unveiled a formidable 288-core CPU, featuring a dual-chiplet configuration with 144 cores on each die, resulting in an impressive 288 cores and 288 threads.

The 288-core CPU, as part of the Sierra Forest lineup, promises to deliver unparalleled processing capabilities for data-intensive tasks, while the compatibility with the Birch Stream platform opens the door for advanced server configurations and scalability, setting the stage for more efficient and powerful computing solutions. This move reflects Intel's dedication to innovation and its ongoing efforts to address the evolving needs of diverse sectors within the technology landscape, promising exciting developments on the horizon for both consumers and enterprise customers.

Intel has unveiled a formidable 288-core CPU, featuring a dual-chiplet configuration with 144 cores on each die, resulting in an impressive 288 cores and 288 threads. This announcement positions Intel to compete directly with AMD's EPYC



Bergamo CPUs, known for offering up to 128 Zen 4C cores, and Ampere's 192-core AmpereOne processors, both of which made their debut earlier this year. Notably, there is speculation within the industry that Intel may even explore the possibility of launching a tri-chiplet SKU with a staggering 432 cores, although the feasibility of such a technological advancement remains uncertain and will undoubtedly draw significant attention.

Intel's latest offering in the form of the 288-core CPU underscores the fierce competition in the high-performance computing market, with companies constantly striving to push the boundaries of core count and processing power. As the technology landscape continues to evolve, this development signifies Intel's commitment to staying at the forefront of innovation, promising exciting prospects for users in need of unparalleled processing capabilities for a wide range of demanding applications.

Intel | intel.com

PRODUCT NEWS by Kirsten Campbell

Dusun Introduces DSGW-290 IoT Edge Computing Gateway Specially Designed for IoT Hardware Developers

The DSGW-290 Dusun Pi4 is a multifunctional IoT gateway hub that supports multiple protocols: BLE/ZigBee/Z-Wave/Sub-G to Wi-Fi/LTE/Ethernet.

Dusun has introduced a highly versatile smart home mini PC—DSGW-290 smart home hub, featuring exceptional multimedia processing capabilities suitable for both home and office media centers and entertainment purposes. This mini PC also offers a comprehensive selection of wireless protocol options, making it an ideal candidate for use as a smart home hub.

It features a high-performance processor designed for robust and reliable performance, the RK3568, equipped with an independent NPU boasting an impressive 1T computing capability.

The 64-bit quad-core Cortex-A55 processor, with a maximum clock speed of 2.0GHz, ensures consistent and efficient data processing for the DSGW-290. Includes 4GB of DDR4 RAM and 64GB of eMMC storage, guaranteeing both speed and stability for your devices. Option to insert a TF card (up to 1TB) and an M.2 SSD (up to 512GB) via the PCIe interface.

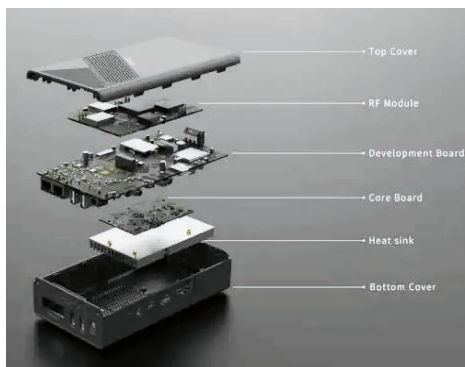
The DSGW-290 mini PC comes fully equipped with a diverse range of integrated wireless modules, featuring Zigbee 3.0 (with optional Tuya Zigbee support), Bluetooth 5.2 (BLE), Z-Wave, Sub-GHz, Wi-Fi (2.4G/5G), and 4G LTE Cat4 connectivity.

The DSGW-290 incorporates a GMAC Ethernet controller that extends the capabilities with two RJ45 Gigabit Ethernet ports, delivering a significant advantage in data transfer speed and meeting the demands of high-speed networks. Simultaneously, the presence of dual Gigabit Ethernet network ports enables users to seamlessly transmit and access data within both internal and external networks. Additionally, the DSGW-290 features a USB3.0 port, a USB2.0 port, and a Type-C port, further enhancing its data transfer capabilities and enabling faster data transfer speeds.

The DSGW-290 distinguishes itself as nearly pure computer hardware, offering developers the flexibility to customize firmware logic from the ground up. Users can select from a range of operating systems, including Debian 11 and Android, as well as leverage programming languages such as C, C++, Python, and Java. The DSGW-290 runs on a Linux-based operating system.

The DSGW-290 represents a significant leap forward in the realm of smart home technology. Delivering unmatched multimedia processing capabilities, a comprehensive suite of onboard wireless modules for seamless connectivity, and exceptional programmability allowing for high customization, it stands as a true pioneer in the world of smart home mini PCs and hubs.

Dusun | dusuniot.com



AMD Introduces EPYC 8004-Series 'Siena' CPUs

AMD has unveiled EPYC 8004-series processors for edge servers. Previously disclosed under the codename Siena, the EPYC 8004 series is AMD's low-cost sub-set of EPYC CPUs, aimed at the telco, edge, and other price and efficiency-sensitive marketing segments. Based on the same Zen4c cores as Bergamo, Siena is essentially Bergamo-light, using the same hardware to offer server processors with between 8 and 64 CPU cores. The new CPUs come in an all-new SP6 form-factor, pack up to 64 Zen 4c cores, and feature a six-channel DDR5 memory subsystem. AMD's EPYC 'Siena' processors are designed for edge and communications servers that rely on one processor and require advanced I/O and power efficiency more than raw performance.

"The new EPYC 8004 Series processors extend AMD leadership in single socket platforms by offering excellent CPU energy efficiency in a package tuned to meet the needs of space and power-constrained infrastructure," said Dan McNamara, senior vice president and general manager, Server Business, AMD.

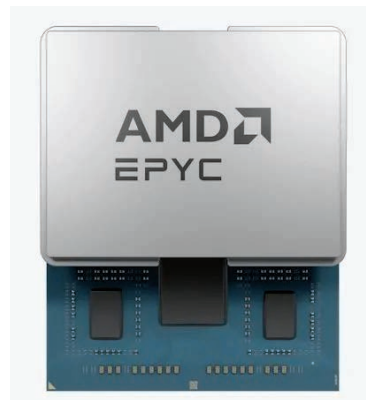
Besides the reduced excitement that comes with the launch of lower-end hardware, there is, strictly speaking, no new silicon involved in this launch. Siena is comprised of the same 5nm Zen 4c core complex die (CCD) chipllets as Bergamo, which are paired with AMD's one and only 6nm EPYC I/O Die (IOD). As a result, the EPYC 8004 family isn't so much new hardware as it is a new configuration of existing hardware—about half of a

Bergamo, give or take.

And that half Bergamo analogy isn't just about CPU cores; it applies to the rest of the platform as well. Underscoring the entry-level nature of the Siena platform, Siena ships with fewer DDR5 memory channels and fewer I/O lanes than its faster, fancier counterpart. Siena only offers 6 channels of DDR5 memory, down from 12 channels for other EPYC parts, and 96 lanes of PCIe Gen 5 instead of 128 lanes. As a result, while Siena is still a true Zen 4 part through and through (right on down to AVX-512 support), it's overall a noticeably lighter-weight platform than the other EPYC family members.

"AMD has delivered multiple generations of data center processors that provide outstanding efficiency, performance, and innovative features," added McNamara. "Now with our 4th Gen EPYC CPU portfolio complete, that leadership continues across a broad set of workloads—from enterprise and cloud, to intelligent edge, technical computing and more."

AMD | amd.com



IDEA BOX

The Directory of PRODUCTS & SERVICES

AD FORMAT:

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

All text and other elements MUST fit within a 2" x 3" format.

E-mail adcop@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

Microprocessor Design Using Verilog HDL

by *Hanno Sander*



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

C Workshop Compiler For PIC® MCUs



- 444 built-in functions
- 177 External Peripheral Drivers
- 156 Example Programs
- 1 Amazing Price

C Workshop Compiler : ONLY \$99!


Professional grade, feature rich compiler for a fraction of the cost

Full support for 13 popular processors

Perfect for students, hobbyists, or those looking to learn C for PIC® MCUs

Add an additional chip for \$20

Sales@ccsinfo.com
(262)522-6500 Ext. 35
www.ccsinfo.com/cc1023



ALL 2021 Issues on CD

cc-webshop.com



www.embeddedARM.com

TS-7100

NXP i.MX 6UL 696 MHz ARM CPU with FPU

Our smallest single board computer measuring only 2.4" by 3.6" by 1.7"



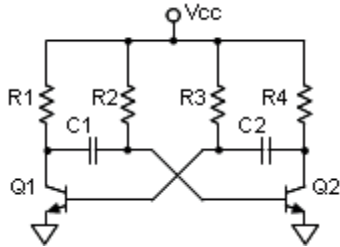
Starting at **\$269** QTY 100



TEST YOUR EQ

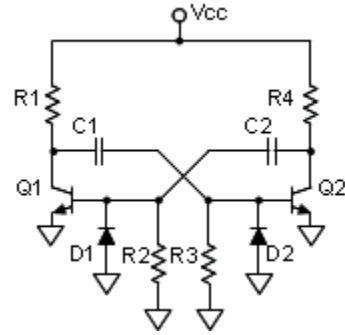
Contributed by David Tweed

Problem 1—The classic two-transistor a stable multivibrator is shown below. Typically, R2 and R3 have at least 10 times the value of R1 and R4. This circuit oscillates, with Q1 and Q2 turning on alternately. From the point in time in a cycle where Q1 first switches on, describe what happens until Q2 switches on.



Problem 2—What determines the time of one half-cycle of the oscillation? Does this depend on VCC?

Problem 3—Recently, a different circuit appeared on the web, shown below. Again, R2 and R3 are significantly larger than R1 and R4. The initial reaction of one observer was that this circuit can't work, because there's no DC bias path for either transistor. Is this assessment correct?



Problem 4—What role do R2 and R3 play in this circuit?

Analog Solutions for Battery Management Systems



microchip.com/BMS



The Future of RF Surveillance

Advancements in Drone RF Surveillance

Harnessing High Bandwidth and Wide Tuning Range Software-Defined Radios (SDRs)

Software-defined radios (SDRs) represent a paradigm shift in wireless communication systems. Unlike traditional radios, which rely heavily on dedicated hardware components to perform specific functions, SDRs leverage software to control and configure radio functions. This flexibility allows SDRs to adapt to various communication standards, frequency bands, and signal processing techniques, making them a versatile solution for modern wireless applications.

Drones have proven to be invaluable tools for surveillance across a multitude of industries. In the realm of RF surveillance, drones equipped with specialized sensors and SDRs can intercept and analyze wireless signals emitted from various sources. These sources can include communication devices, Internet of Things (IoT) devices, and even illicit transmitters. This capability makes drones equipped with SDRs vital assets for spectrum monitoring, threat detection, and intelligence gathering with a specific focus on the incorporation of high-bandwidth and wide tuning range SDRs.

The integration of high-bandwidth SDRs introduces an array of advantages, empowering drones to capture and analyze a broader range of frequencies than ever before. These specific advantages include improved signal detection, real-time analysis, and spectrum mapping.

Improved Signal Detection: High-bandwidth SDRs empower drones to capture a wider range of frequencies simultaneously. In the past, narrowband radios limited the ability to monitor only specific frequencies at a time. With high-bandwidth SDRs, drones can now conduct comprehensive spectrum analysis, identifying potential threats and anomalous activities across multiple frequency bands (**Figure 1**). This is particularly advantageous in scenarios where

various wireless technologies coexist, such as Wi-Fi, Bluetooth, cellular, and IoT.

Real-Time Analysis: Traditional RF surveillance systems often required post-processing of captured data for analysis. High-bandwidth SDRs enable drones to process and analyze complex RF signals in real time using on-board field programmable gate arrays (FPGAs). This immediate analysis provides operators with actionable insights, allowing them to swiftly respond to emerging threats or anomalies. For example, a drone equipped with a high-performance, high-bandwidth SDR with on-board FPGA for DSP can identify unauthorized drone communication attempts in real time, helping security personnel intervene before any potential threat materializes.

Spectrum Mapping: Drones equipped with high-bandwidth SDRs can create detailed spectrum maps. These maps illustrate signal strength and frequency distribution across a geographical area. By identifying signal congestions and dead zones, network operators can optimize wireless deployments for better coverage and performance. Additionally, these maps aid in detecting unauthorized transmissions and interference sources. For instance, during a large event, a drone can monitor signal congestion and ensure that critical communications remain unaffected.

Similar to the benefits of high-bandwidth SDRs outlined above, it is equally important



Brandon Malatest
COO and Co-Founder of
Per Vices

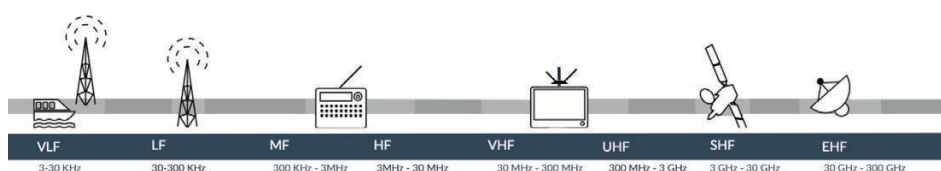


FIGURE 1

High-bandwidth SDRs allow drones to scan multiple bands of the frequency spectrum, shown here.



FIGURE 2

High-bandwidth, wide tuning range SDRs like Per Vices Cyan can improve drone-based RF surveillance.

to ensure the SDRs being utilized offer wide tuning ranges. These systems offer additional flexibility and interference mitigation along with the use for covert operations.

Frequency Flexibility: Wide tuning range SDRs provide drones with the ability to scan a broad spectrum of frequencies. This flexibility is crucial in scenarios where the frequency of interest might change frequently. Whether it's monitoring licensed communication bands or searching for rogue signals, wide tuning range SDRs ensure that drones can adapt to dynamic RF environments. This adaptability is vital in situations like disaster response, where communication frequencies may shift due to damaged infrastructure.


Interference Mitigation: Drones equipped with wide tuning range SDRs can identify sources of interference and assess their impact on communication systems. By pinpointing interfering signals, operators can take proactive measures to mitigate the effects of interference and maintain the reliability of critical wireless networks. For example, in urban areas with high levels of electromagnetic interference, drones can help pinpoint sources of interference, enabling authorities to optimize signal distribution and minimize service disruptions.

Covert Operations: The wide tuning range of SDRs allows drones to intercept and analyze signals across a range of frequencies, including those that may be used for covert communications or illicit activities. This makes drones equipped with such SDRs essential tools for law enforcement and security agencies, enabling them to detect and counter unauthorized communication attempts. In scenarios where criminals use encrypted

communication on various frequencies, drones with wide tuning range SDRs can help decode and analyze such communications, aiding law enforcement efforts.

While the benefits of using high-bandwidth and wide tuning range SDRs for drone RF surveillance are undeniable, there are challenges to address. Power consumption is a critical consideration, as processing high-bandwidth signals demands substantial energy. Efficient power management solutions, and standard rack mount solutions optimized for signal processing, are essential to extend drone flight times.

Moreover, the complexity of signal processing algorithms and data analysis must be managed efficiently to ensure real-time responsiveness. Advanced signal processing techniques, including machine learning and artificial intelligence, can aid in quickly identifying patterns in intercepted signals and distinguishing between legitimate and potentially malicious activities.

The integration of high bandwidth and wide tuning range SDRs in drone-based RF surveillance marks a significant leap forward in capabilities (**Figure 2**). These advancements empower drones to efficiently detect, analyze, and respond to diverse RF signals, making them indispensable tools for spectrum monitoring, threat detection, and intelligence gathering. As technology continues to evolve, the synergy between SDRs and drones is set to reshape the landscape of RF surveillance, unlocking new possibilities and enhancing our ability to monitor and secure the wireless world. Through effective collaboration, innovation, and responsible deployment, high-bandwidth and wide tuning range SDRs promise to revolutionize the field of drone RF surveillance, creating a safer and more connected future. By addressing challenges and leveraging the full potential of SDRs, the integration of drones and RF surveillance holds the promise of enhanced situational awareness, improved communication reliability, and a stronger foundation for public safety and security. 

ABOUT THE COMPANY

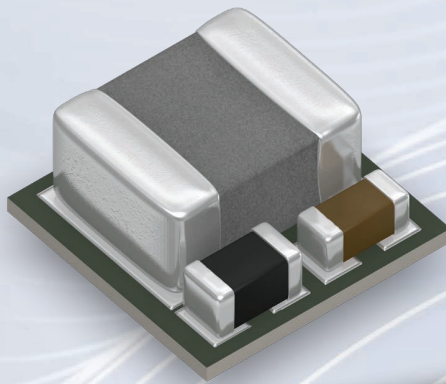
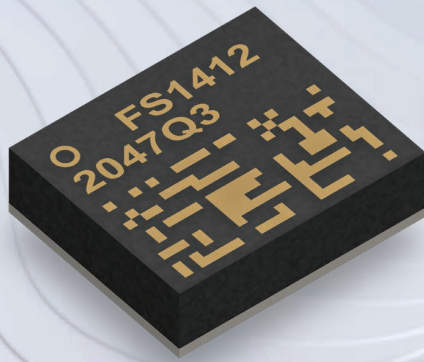
Per Vices, a leader in software-defined radios, offers high-bandwidth and wide tuning range SDRs for RF surveillance. Its high-performance radios come with exceptional signal processing capabilities and advanced hardware features, so that customers can leverage Per Vices' SDRs to achieve superior performance, enhanced flexibility, and extended capabilities. Contact solutions@pervices.com to learn more about the different options available.

RESOURCES

Per Vices | www.pervices.com

ABOUT THE AUTHOR

Brandon Malatest is the COO and co-founder of Per Vices Corporation, a leader in software-defined radio technology. Brandon has an honor's degree in Physics with a specialization in Experimental Physics from the University of Waterloo in Ontario, Canada. On graduating, Brandon started his career as a research analyst and statistician at one of the largest market research firms in Canada and later joined Victor Wollesen to co-found Per Vices. Since starting Per Vices, Brandon has authored many thought leadership articles based on software defined radio technology.

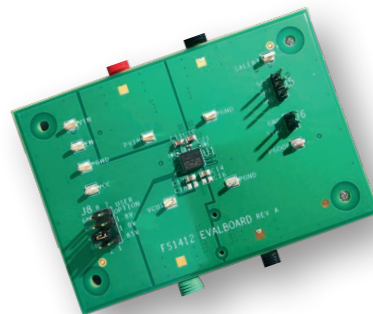


μ POL™ Chip-Embedded Power Modules

A Simple Solution for High Power Density Applications

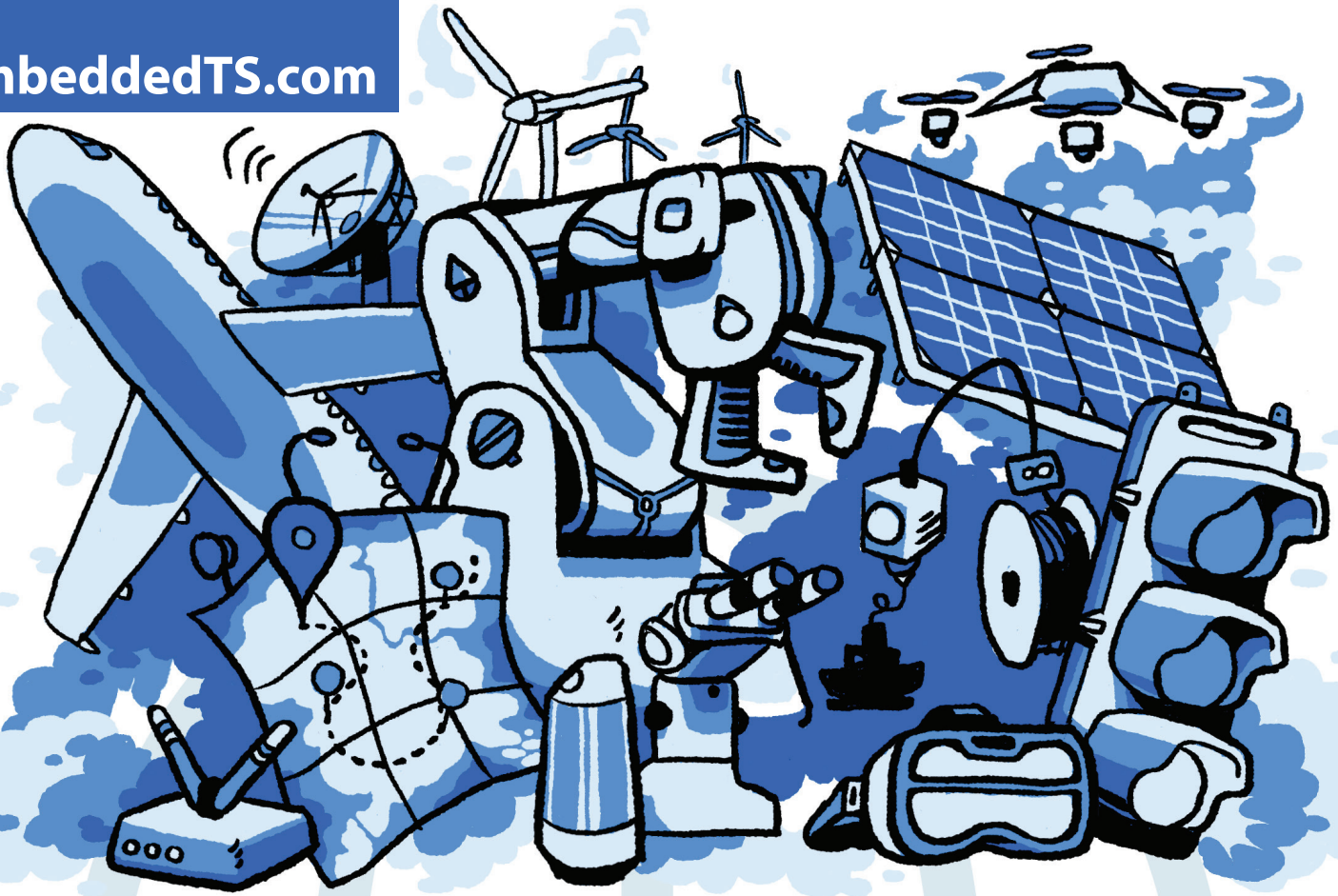
TDK μ POL™ DC-DC converters are compact and highly integrated point-of-load converters for powering CPUs, MCUs, ASICs, FPGAs, DSP, and other advanced digital logic devices, providing the high performance, fast load transient response, and high accuracy voltage regulation needed by these devices.

- Technology Includes Inductor, DC-DC regulator with MOSFETs and Driver
- Ultrathin: 3.3 mm x 3.3 mm x 1.5 mm or 4.9 mm x 5.8 mm x 1.6 mm
- Plug & Play (No Compensation Required)
- DC-DC Analog & Digital Bus Options (I2C / PMBus)
- Current Output: 3 A, 4 A, 6 A, 12 A
- Wide Input Voltage (up to 16 V)
- Adjustable Vout \pm 5m V
- Output Voltage, \pm 0.5% Initial accuracy

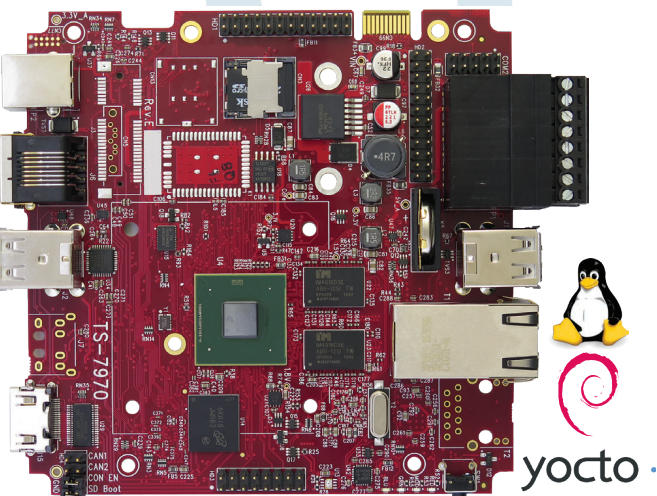


**REGISTER FOR
A FREE μ POL
EVALUATION
BOARD
(Up to 5 winners)**





GO BUILD IT



Powered by the NXP i.MX6 SOC with the Arm® Cortex®-A9 core, the TS-7970 industrial Single Board Computer (SBC) stands out from the crowd with its high performance components, connectivity options, and multimedia capabilities. It's a general purpose, low-power SBC ready to tackle demanding applications including digital signage, HMI's, data acquisition, Edge IoT, industrial automation, and anything in between.

- 800 MHz Solo or 1 GHz Quad Core Arm based CPU
- 1 to 2 GB DDR3 RAM
- 4 GB MLC eMMC Flash
- SATA II, MiniCard, DIO, ADC, ModBus, USB, CAN
- Multimedia Solution with HDMI, LVDS, and Audio I/O

TS-7970

NXP i.MX6 Arm® Cortex®-A9 ARM CPU
Single Board Computer



Made in USA
with Global Parts

Designing Combinational Circuitry

Employing Tiny Logic

FEATURES

Tiny or little logic components belong to the staple portfolio of semiconductor manufacturers. For some special purposes, they offer compelling advantages. Components that are barely visible on the printed circuit board connect, for example, ASICs to microcontrollers (MCUs), or allow a reduction in the pin count and hence the cost of the more flashy ICs. Therefore, designing gate by gate is not an outdated art. It is, however, different from the gate-level design of the past. Here we give an overview of components, design rationales, and particular solutions.

By *Wolfgang Matthes*

Sometimes, digital or logic design tasks require more than one gate, but are not so complex that a CPLD or even an FPGA is deemed necessary. When small-scale digital design is only an occasional challenge, encompassing only a minor part of the total circuitry, one may have concerns about the expenditures for a CPLD/FPGA integrated development environment (IDE), programming equipment, and so on. Thus, it may be expedient to resort to traditional logic design. This is not just a matter of tinkering. On the contrary, elementary logic circuitry is also

used in large-volume fields of use like automotive systems. Consequently, semiconductor manufacturers offer a broad portfolio of appropriate devices (**Figures 1 to 5**) [1-15].

Packages are, so to speak, a science in itself. There

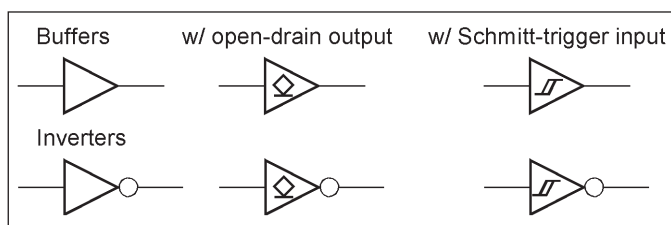


FIGURE 1 Buffers and inverters. One, two, or three of those devices are housed in one IC package (single, dual, or triple devices).

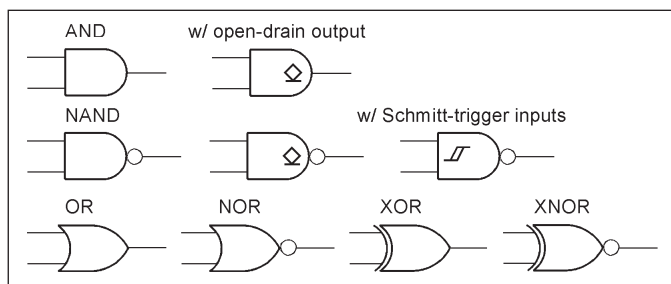


FIGURE 2 Gates with two inputs. There are single and dual gates.

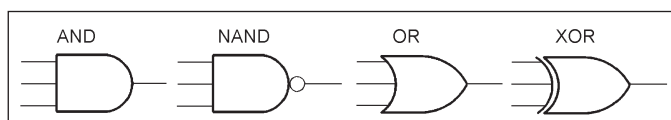


FIGURE 3 Gates with three inputs. In tiny packages, only single devices are available.

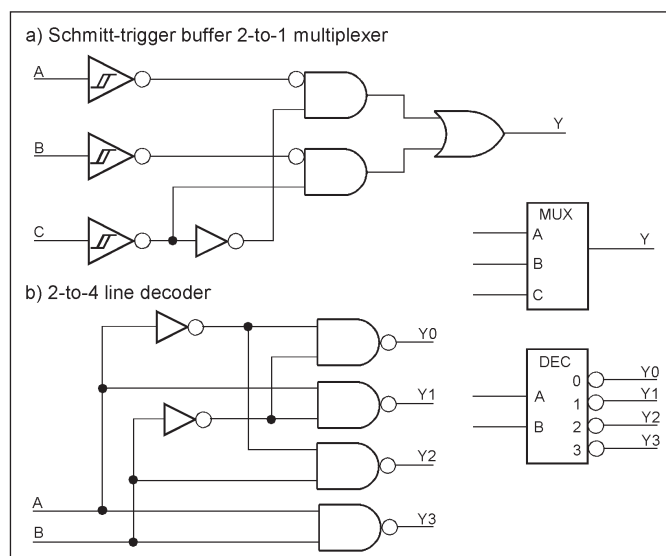


FIGURE 4 Two examples of somewhat more complex devices. Above a 2-to-1 data selector/multiplexer with Schmitt-trigger inputs (74AUP1T157), below a 2-to-4 line decoder (74LVC1G139). The similar multiplexer 74AUP1T158 has an inverted output.

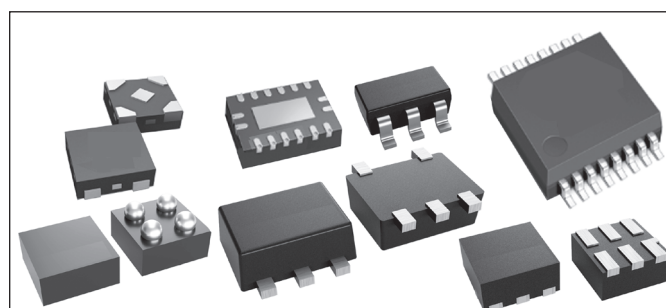


FIGURE 5 A few of the tiny IC packages (not to scale).

are many package types around, the manufacturers have different designations, suffixes, and trademarks, and they are always busy inventing something new, making things smaller and smaller. The latest types are indeed very tiny. They have no leads, and the backsides are covered with solder contacts. To give a first impression of what we're talking about, Figure 5 shows a few examples. Beyond that, refer to the manufacturer's catalogs, application notes, datasheets, and cross-reference tables (see, for example, [2-4] and [7-14]).

The "tiny" or "little" IC series comprise sets of different gates: AND, NAND, OR, NOR, XOR, and XNOR. Additionally, there are multipurpose devices that can be configured to perform the desired logic function. So, the problem of implementing all the combinational functions by a single type of gate does not exist (in contrast to the distant past, where designers had to get by with only NANDs (TTL) or NORs (ECL)). Designing with such components could be dubbed trickery in the small. What is taught in introductory digital engineering courses may not be that helpful. Therefore, we will not proceed by explaining Karnaugh-Veitch diagrams and the like. Devices with three-state outputs, flip-flops, analog switches, and so on we have omitted here. Instead, we will concentrate on straightforward combinational circuits.

A FEW APPLICATION EXAMPLES

Tiny logic solves small or straightforward logic tasks on the spot. There is no talk of implementing arithmetic-logic units (ALUs) or finite state machines (FSMs).

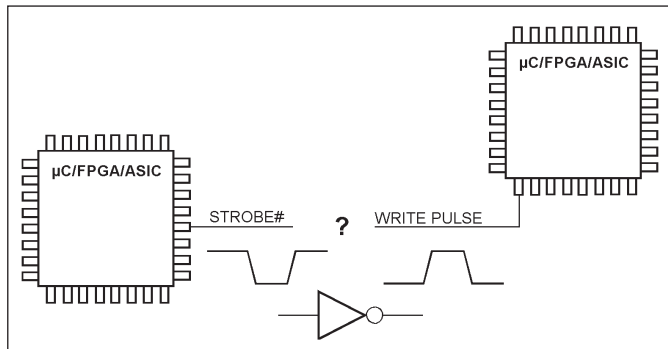


FIGURE 6
A tiny inverter adapts an active-Low output to an active-High input.

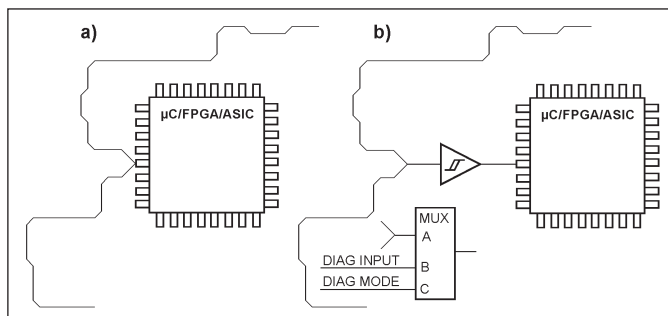


FIGURE 7
Lengthy traces impede signal integrity. A tiny device in the near vicinity ensures a clean signal at the IC's input. If the signal's edges are deteriorated, a buffer with a Schmitt-trigger input is the obvious choice. The multiplexer beyond (the device shown in Figure 4) is an alternative if diagnostics or PCB testing are to be supported.

A simple application is patching, as illustrated in **Figure 6**. A complex IC generates an output signal whose behavior fits well with an input of another highly integrated device. Unfortunately, the signal is generated active-Low, but the input of the receiving IC is active-High. A tiny inverter is the most straightforward solution to this problem.

Figure 7 depicts a long signal trace running across the board. Such traces may pick up noise and may cause the signal edges to deteriorate. The integrated circuit shown here needs, however, a clean signal with steep edges. Think, for example, of a clock or reset input. A tiny buffer, placed in the near vicinity, would solve the problem. The alternative shown concerns diagnostics and PCB testing. Here, a tiny multiplexer allows for the injection of a diagnostic signal (think of clock pulses excited by a tester or a service processor) if the circuitry is switched to a diagnostic mode.

The example in **Figure 8** concerns an application where direct-acting (that is, non-programmable) hardwired logic is a mandatory requirement. In case of an emergency, signals are to be brought to determined levels. All further activities are to be inhibited. Low levels can be enforced by AND gates, high levels by OR gates.

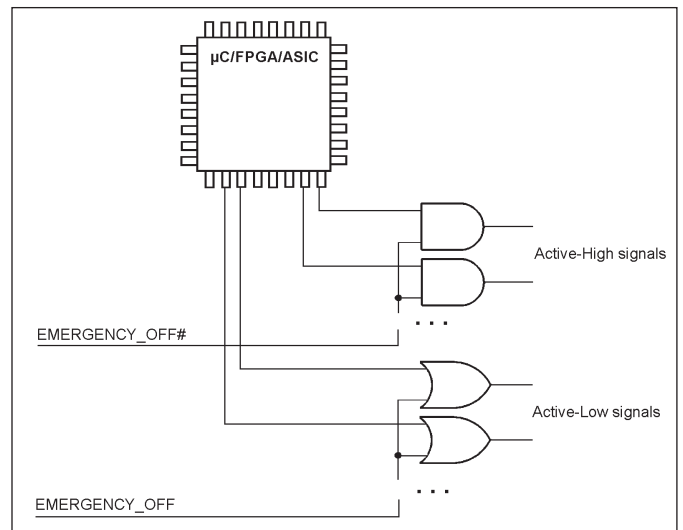


FIGURE 8
Tiny gates enforce particular signal levels in case of an emergency.

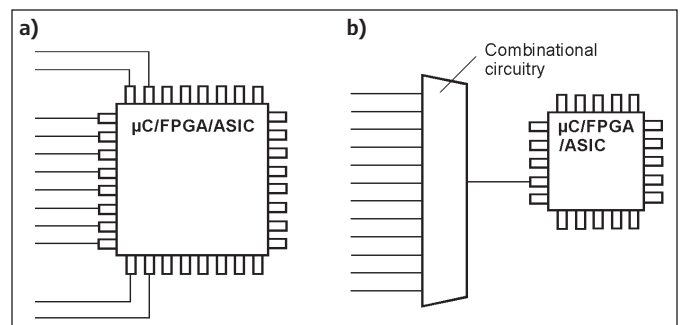


FIGURE 9
Particular conditions or bit patterns on signal lines are to be detected. It could be done by a decoder in the FPGA or ASIC or by comparing read-in bit patterns by software. This, however, requires connecting all signal lines, thus wasting precious I/O pins (a). If the conditions are detected by external circuitry, only one or a few pins are needed. Thus we could get by with an MCU, FPGA, or ASIC in a smaller package (b).

The package contributes crucially to the cost of a complex integrated circuit. So it's understandable to want to get by with fewer pins and a package that's less costly to purchase and process. Occasionally, external circuitry can save on the required number of pins considerably, as illustrated in **Figure 9**.

For example, some or even many sensor signals may be OR-ed together to trigger an interrupt in a microcontroller (MCU) (**Figure 10a**). A further example is detecting conditions on signal lines for conditional branching or to trigger interrupts. For that, we must implement so-called product terms. That means AND-ing together the particular signals, either true or inverted (**Figure 10b**).

To house this kind of circuitry, we may think of a CPLD or even a low-cost FPGA. Occasionally, this approach is recommended by manufacturers of programmable logic [46-48]. The obvious advantages are that functional complexity is not restricted, and you can master such tasks without being a seasoned digital designer, at least in most cases. (I recommend resorting to the Verilog hardware description language (HDL) and leaving the rest to the IDE.) The benefits of tiny logic appear if only straightforward combinational functions are to be implemented. Then you will get by without HDL and IDE at all.

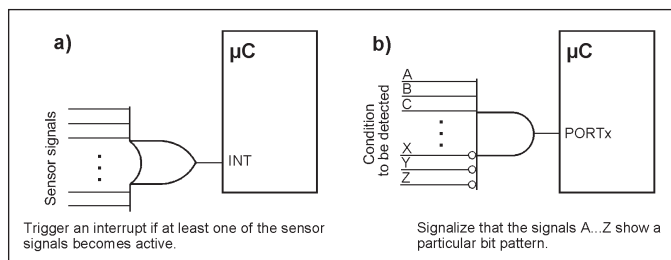


FIGURE 10
An apparent advantage of the external combinational circuits is that we could get by with an MCU in a cheaper package (that is, one with fewer pins).

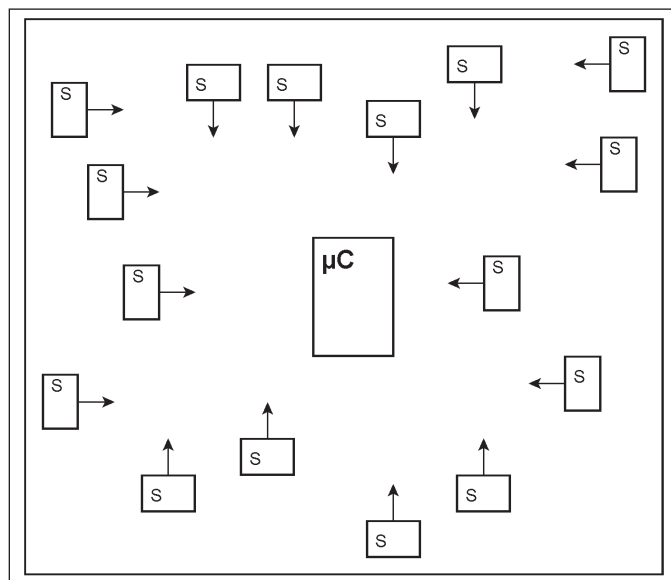


FIGURE 11
Tiny logic may be dispersed over the printed circuit board (PCB). Here, sensors (S) are shown. They are to be OR-ed together to trigger interrupts in the MCU. When done in a CPLD, for example, all sensor signals must be routed to this device. When spreading OR gates in the vicinity of the sensors, only a few traces need to be run to the MCU.

Figure 11 illustrates a further advantage. Imagine a somewhat larger PCB with sensors (S) or other signal sources spread over the total real estate. When all those signals are to be OR-ed or AND-ed by a single CPLD or FPGA, all the signal lines have to be routed to this device. Therefore, it could make sense to also distribute the combinational circuitry over the PCB, especially if cost is a primary concern and the number of PCB layers should be kept as low as possible.

DESIGNING WITH TINY LOGIC

Our primary design challenges are twofold. The first is to choose tiny components wisely. The second consists of cascading such components so that more, or even many, input signals can be attached. For both goals, the sharpest tool in our box is DeMorgan's law (**Figure 12**). There is an uncountable number of sources that deal with Boolean algebra and basic gate-level design. More often than not, however, Boolean functions are not treated as tools for problem-solving but solely as objects of minimization. I recommend looking first into the technical documentation the semiconductor

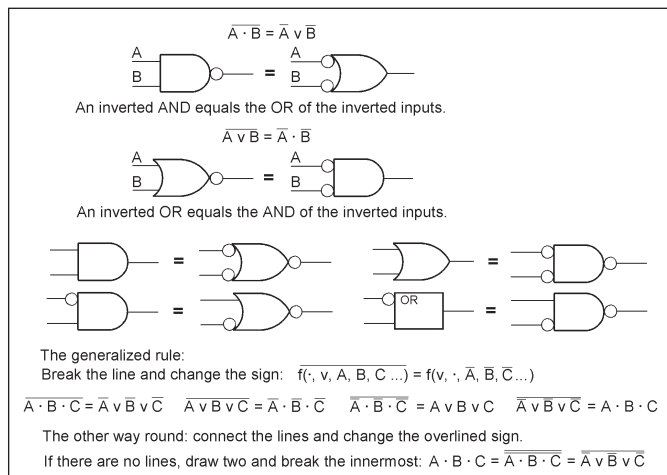


FIGURE 12
DeMorgan's law describes the so-called duality between AND, OR, the inversion of the outputs, and the inversion of the inputs. AND and OR can be swapped against each other, provided non-inverted signals are inverted and vice versa.

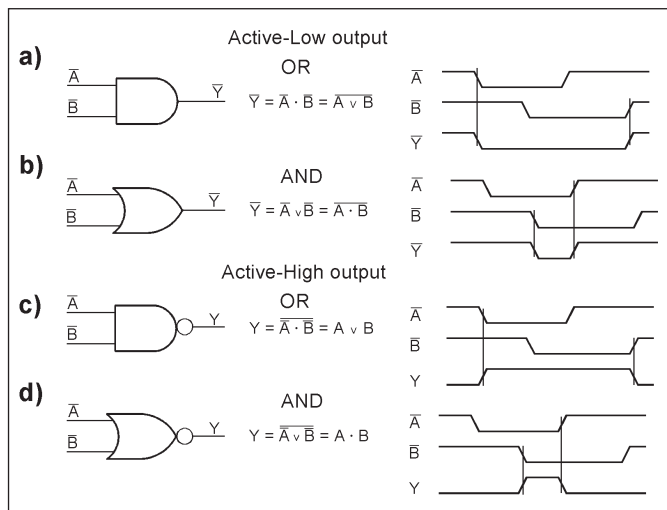


FIGURE 13
How basic gates act on active-Low signals.

manufacturers provide (like [4] or [52]), and not to begin with college-level textbooks.

Capturing the problem and understanding the design task: The problems to be solved are not that complicated. Nevertheless, they must be understood in their intricate details. A well-proven approach is to describe the problem as painstakingly as possible using the terms AND, OR, and NOT. This way, we will obtain at first colloquial and then formalized Boolean expressions. They are to be implemented by our tiny components. Mostly, it could be done best by assembling the combinational circuits step by step from small basic gates, without dealing with two-level canonical forms, Karnaugh-veitch diagrams (K-maps), and the like. A well-proven overall approach is to first solve the pure logical design problem, assuming that all types of tiny devices may be applied. The

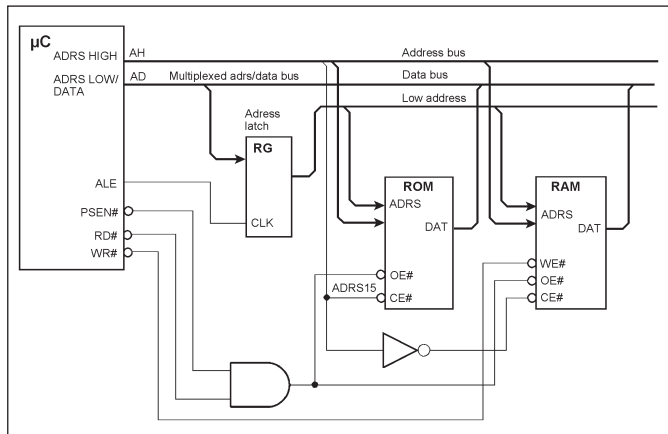


FIGURE 14 Turning the 8051 into a v. Neumann machine where instructions and data are located in the same address space can be done by two tiny devices, an AND gate, and an inverter.

<table border="1"> <thead> <tr> <th>Inputs</th> <th>Output</th> <th>AND</th> </tr> </thead> <tbody> <tr><td>Low</td><td>Low</td><td>Low</td></tr> <tr><td>Low</td><td>High</td><td>Low</td></tr> <tr><td>High</td><td>Low</td><td>Low</td></tr> <tr><td>High</td><td>High</td><td>High</td></tr> </tbody> </table>	Inputs	Output	AND	Low	Low	Low	Low	High	Low	High	Low	Low	High	High	High	<table border="1"> <thead> <tr> <th>Inputs</th> <th>Output</th> <th>OR</th> </tr> </thead> <tbody> <tr><td>Low</td><td>Low</td><td>Low</td></tr> <tr><td>Low</td><td>High</td><td>High</td></tr> <tr><td>High</td><td>Low</td><td>High</td></tr> <tr><td>High</td><td>High</td><td>High</td></tr> </tbody> </table>	Inputs	Output	OR	Low	Low	Low	Low	High	High	High	Low	High	High	High	High	<table border="1"> <thead> <tr> <th>Inputs</th> <th>Output</th> <th>XOR</th> </tr> </thead> <tbody> <tr><td>Low</td><td>Low</td><td>Low</td></tr> <tr><td>Low</td><td>High</td><td>High</td></tr> <tr><td>High</td><td>Low</td><td>High</td></tr> <tr><td>High</td><td>High</td><td>Low</td></tr> </tbody> </table>	Inputs	Output	XOR	Low	Low	Low	Low	High	High	High	Low	High	High	High	Low
Inputs	Output	AND																																													
Low	Low	Low																																													
Low	High	Low																																													
High	Low	Low																																													
High	High	High																																													
Inputs	Output	OR																																													
Low	Low	Low																																													
Low	High	High																																													
High	Low	High																																													
High	High	High																																													
Inputs	Output	XOR																																													
Low	Low	Low																																													
Low	High	High																																													
High	Low	High																																													
High	High	Low																																													
<table border="1"> <thead> <tr> <th>Inputs</th> <th>Output</th> <th>NAND</th> </tr> </thead> <tbody> <tr><td>Low</td><td>Low</td><td>High</td></tr> <tr><td>Low</td><td>High</td><td>High</td></tr> <tr><td>High</td><td>Low</td><td>High</td></tr> <tr><td>High</td><td>High</td><td>Low</td></tr> </tbody> </table>	Inputs	Output	NAND	Low	Low	High	Low	High	High	High	Low	High	High	High	Low	<table border="1"> <thead> <tr> <th>Inputs</th> <th>Output</th> <th>NOR</th> </tr> </thead> <tbody> <tr><td>Low</td><td>Low</td><td>High</td></tr> <tr><td>Low</td><td>High</td><td>Low</td></tr> <tr><td>High</td><td>Low</td><td>Low</td></tr> <tr><td>High</td><td>High</td><td>Low</td></tr> </tbody> </table>	Inputs	Output	NOR	Low	Low	High	Low	High	Low	High	Low	Low	High	High	Low	<table border="1"> <thead> <tr> <th>Inputs</th> <th>Output</th> <th>XNOR</th> </tr> </thead> <tbody> <tr><td>Low</td><td>Low</td><td>High</td></tr> <tr><td>Low</td><td>High</td><td>Low</td></tr> <tr><td>High</td><td>Low</td><td>Low</td></tr> <tr><td>High</td><td>High</td><td>High</td></tr> </tbody> </table>	Inputs	Output	XNOR	Low	Low	High	Low	High	Low	High	Low	Low	High	High	High
Inputs	Output	NAND																																													
Low	Low	High																																													
Low	High	High																																													
High	Low	High																																													
High	High	Low																																													
Inputs	Output	NOR																																													
Low	Low	High																																													
Low	High	Low																																													
High	Low	Low																																													
High	High	Low																																													
Inputs	Output	XNOR																																													
Low	Low	High																																													
Low	High	Low																																													
High	Low	Low																																													
High	High	High																																													

FIGURE 15 Positive (above) and negative (below) logic.

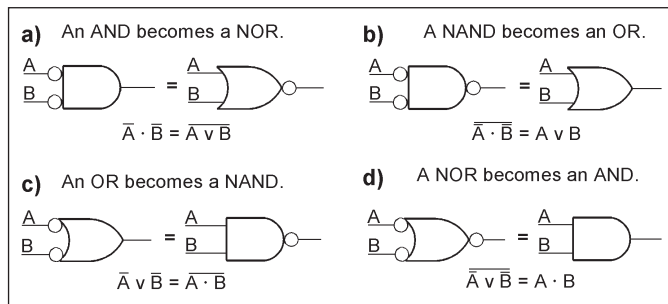


FIGURE 16 Inverted inputs will alter the function.

levels and supply voltages, the IC families, packages, and so on are dealt with in a second pass.

Logic levels and signals: Propositional logic knows only two values. Applying them to digital design seems to be the most straightforward thing on earth. Those values are, however, to be assigned to the problem to be solved, and it's easy to mix something up, causing annoying design errors. So be careful and better look once more. In the beginning, we will assume that all is without a hitch. The supply voltage (V_{CC}), the Low and High levels, and the IC families fit well together. The problems we will discuss later.

The logic levels are physical facts. If a level is nearer to minus infinity ($-\infty$), it is called Low; if nearer to plus infinity ($+\infty$), it is called High.

Propositional logic is concerned with truth. George Boole has equated truth with 1 and falsehood with 0. In addition, 0 and 1 are the digits of binary numbers. If the 0 is represented by the Low level and the 1 by the High level, we speak of positive logic. The opposite assignment is called negative logic.

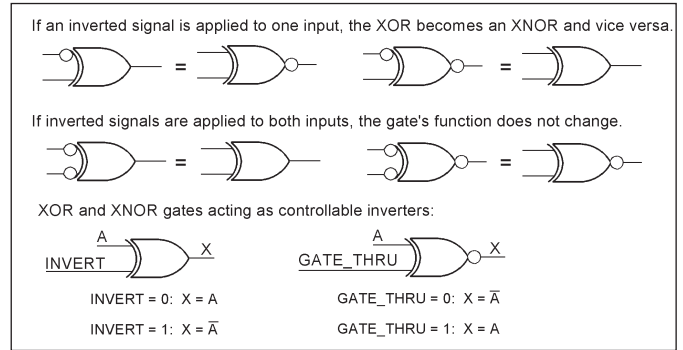


FIGURE 17 Some particular properties of the XOR and XNOR functions.

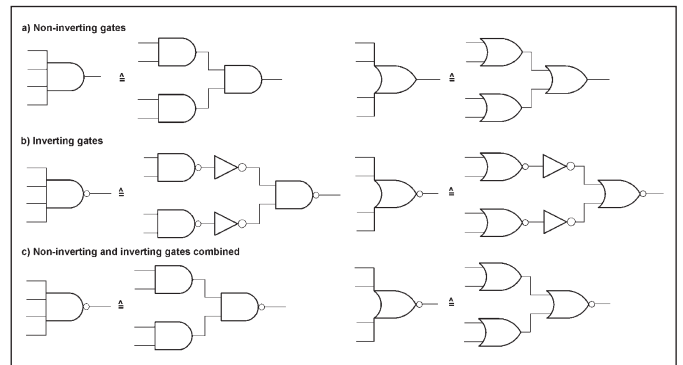


FIGURE 18 Extending the number of inputs by cascading.

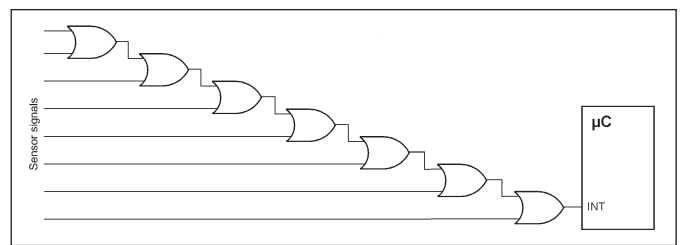


FIGURE 19 Cascading by daisy-chaining.

particular function, we begin with its truth table. The control inputs of all AND gates that correspond to ones in the result column are connected to High, the remaining to Low.

An ensemble of 2^n AND gates, one for each bit pattern, is essentially a decoder. So let us look for basic types of components in which 2^n product terms are readily decoded. There are three such basic types, the binary (1-out-of- n) decoder, the multiplexer, and the addressable memory. Integrated decoders (like the venerable 74x138) contain

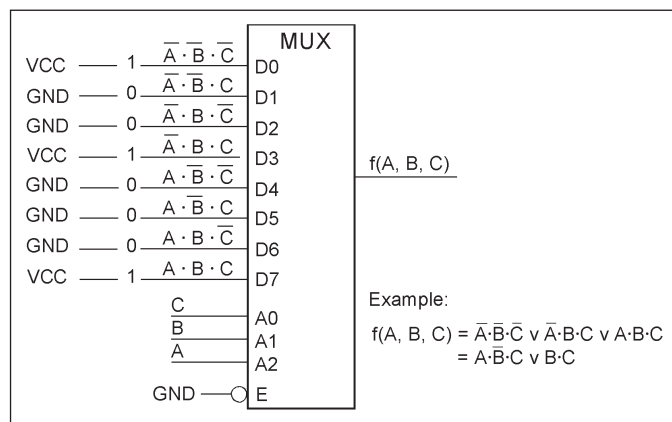


FIGURE 32
A Boolean function of three variables implemented by a multiplexer. Its address decoder acts as the decoder depicted in Figure 31. Suitable devices are 74x151 8-to-1 multiplexers or 8-channel analog switches, like the NX3L4051 [36]; this device guarantees break-before-make, so the inputs may be connected directly to VCC.

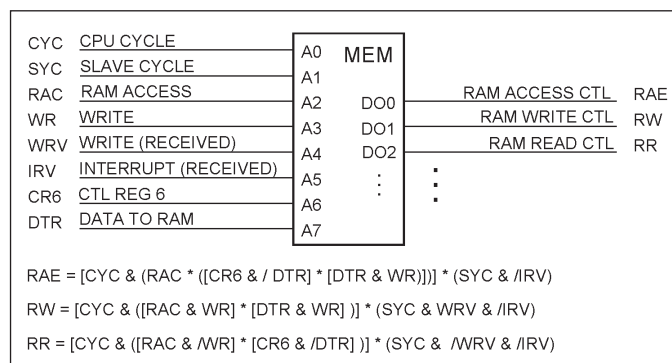


FIGURE 33
A historical example of a ROM housing the truth tables of some combinational functions. Thus, it substitutes a considerable number of gates. Some of the Boolean equations are shown here, as they have been entered into the development system. However, as straightforward as the design idea seems, there are some gotchas to observe (which I had—decades ago—learned the hard way).

Family	VCC range	Output drive	Input tolerance	IOFF protection
AUP	0.8 to 3.6 V	4 mA	3.6 V	Yes
AUC	0.8 to 2.7 V	8 mA	3.6 V	Yes
LVC	1.65 to 5.5 V	24 mA	5.5 V	Yes
AHC	2.0 to 5.5 V	8 mA	5.5 V	Yes
LV1T	1.8 to 5.5 V	7 mA	5.5 V	No

TABLE 1
Logic families comprising tiny gates (according to [2]).

only the AND gates. OR-ing is to be done outside. Thus, for practical reasons, the decoder-based solution may be omitted here.

The multiplexer as a universal combinational building block: The multiplexer is a combination of a data selector and an address decoder. A multiplexer with n address inputs selects one of the data inputs to be gated through to the output. Thus, one can use a 2^n -to-1 multiplexer to implement any combinational function of n variables. It requires only wiring the data inputs to Low or High, according to the result column of the corresponding truth table (**Figure 32**). Thus, the multiplexer becomes a small read-only memory (ROM). In some FPGA families, the logic cells are implemented this way. To be programmable, the multiplexer inputs are attached, for example, to the flip-flops of a shift register or to flash memory cells.

Unfortunately, the series of tiny logic components contain no multiplexers with a useful number of inputs (4, 8, or even 16). Therefore, you must resort to components in larger packages. Occasionally, analog multiplexers may work, too. If break-before-make behavior is not guaranteed, I recommend not wiring the inputs directly to VCC but applying the High voltage via a pull-up resistor to limit eventual shoot-thru currents.

The addressable memory—a universal combinational building block: The memory cells are selected by addressing. n address bits correspond to 2^n memory cells. Referring to the expansion theorem, the address decoder corresponds to the AND gates, the stored bits correspond to the control inputs, and the bit line implements the OR function. Thus, implementing a combinational function requires nothing more than storing the result column of the truth table.

The address inputs are connected to the input signals; the memory cells are filled with ones or zeros according to the truth table. In this way, any combinational function can be implemented, limited only by the storage capacity.

In some FPGA families, the logic cells contain small RAMs, the so-called lookup tables (LUTs), to implement the combinational functions. Larger FPGAs also contain dedicated RAM structures, like distributed RAMs (that are LUTs operated as addressable RAMs instead of combinational circuits) and block RAMs. In an example LUT, a RAM has a storage capacity of 64 bits. Hence it can accommodate a combinational function of 6 inputs. Dedicated RAMs can be configured for different word lengths, for example, 16k x 1, 8k x 2, 4k x 4, and so on. A 16k x 1 block RAM could accommodate a combinational function of $\text{ld } 16k = 14$ inputs.

Outside the FPGAs, the principle of stored truth tables can be implemented by ROMs with an asynchronous memory interface, thus limiting the number of inputs between, say, 8 to 20. Here we take it as a matter of course that we want to get by with a simple design and low cost.

The idea may occur to let a ROM absorb some combinational functions that otherwise would be spread over the PCB (**Figure 33**). There is, however, a caveat. Memories with an asynchronous interface are internally clocked devices. They have sequencers built-in that detect when an address or control signal changes its level. Then they start a new access cycle. In the course of this cycle, the data outputs may become temporarily

Generally, there are two kinds of signals. The first carries binary digits, ones or zeros. They have nothing to do with truth, falsehood, activity, idleness, and the like, but are simply two values of equal significance. Signals of the second kind exert activities. The logic levels represent two states, idle (or off or deasserted), and active (or on or asserted). In this regard, we speak of signals that are active-Low or active-High.

As a first example, **Figure 13** shows how two active-Low signals can be combined by AND, NAND, OR, and NOR gates. If the output is to be active-Low, an AND gate acts as an OR (Figure 13a), and an OR gate as an AND (Figure 13b). If the output is to be active-High, a NAND is to be used instead of the AND (c), and a NOR instead of an OR (d).

In one of the earliest applications of tiny logic, the AND gate of Figure 13a is employed to turn the venerable 8051 microprocessor (MPU) into a von Neumann machine (**Figure 14**). Architecturally, the 8051 has separate memories for programs and data (Harvard architecture). It is obvious to store programs in the ROM and data in the RAM. In some applications, however, it is desirable to have a unified memory (von Neumann architecture). PSEN# signalizes that instructions are to be fetched. RD# signalizes that data bytes are to be read. To get access to both memories for instructions as well as for data, a joint output enable (OE#) signal is generated by OR-ing both low-active signals. To select the ROM or the RAM, the highest-order address bit is used here, requiring, in addition, a tiny inverter.

Positive and negative logic: An AND in positive logic corresponds to an OR in negative logic and vice versa. The same correspondence applies to NAND and NOR, as well as to XOR and XNOR (**Figure 15**).

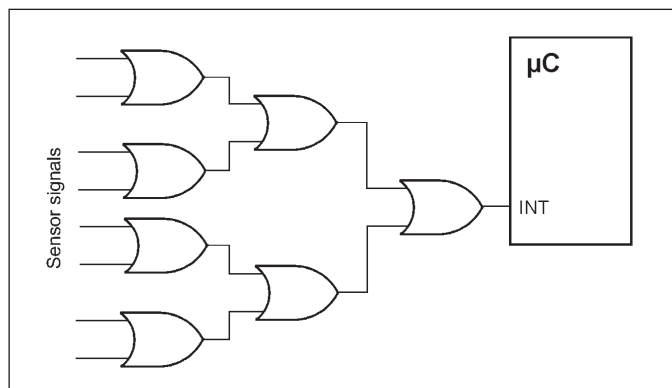


FIGURE 20
Cascading by connecting the gates according to an inverted-tree topology.

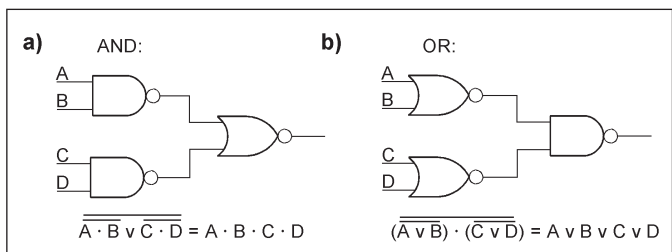


FIGURE 21
Making good use of DeMorgan's law. a) shows an AND, b) an OR with four inputs each. NAND and NOR functions are obtained by not inverting the output.

Gates with inverted inputs: If the inputs of a gate are inverted or if inverted signals are applied, the gate's function will change according to DeMorgan's law, as depicted in **Figure 16**.

XOR and XNOR: XOR stands for exclusive OR; XNOR is an XOR with the output inverted. An XOR gate with two inputs signalizes inequality, and a corresponding XNOR signalizes equality. In other words, the XNOR behaves as a single-bit equality comparator. A positive logic XOR corresponds to a negative logic XNOR and vice versa. Both gates can be operated as controllable inverters (**Figure 17**). The output of an XOR with an arbitrary number of inputs signalizes a one if the number of ones at the inputs is odd (odd parity).

Cascading: Cascading means composing a gate with many inputs from gates having few inputs. Non-inverting gates of the same type can be cascaded easily (**Figure 18a**). To cascade inverting gates, additional inverters must be interspersed (**Figure 18b**). With the comprehensive assortment of gate types available nowadays, the most straightforward solution is to combine non-inverting and inverting gates (**Figure 18c**).

There are two basic topologies to cascade gates: the daisy chain (**Figure 19**) and the inverted tree (**Figure 20**). For a particular number of inputs, both need the same number of gates. Only the propagation delay is different. In a daisy chain, it increases linearly with the number of cascaded gates. In the tree, it increases logarithmically. If propagation delay is not that important, you may prefer the topology that is most expedient for routing.

By using NAND and NOR gates and applying DeMorgan's law, you can implement AND, OR, NAND, and NOR functions with an arbitrary number of inputs (**Figure 21** and **Figure 22**).

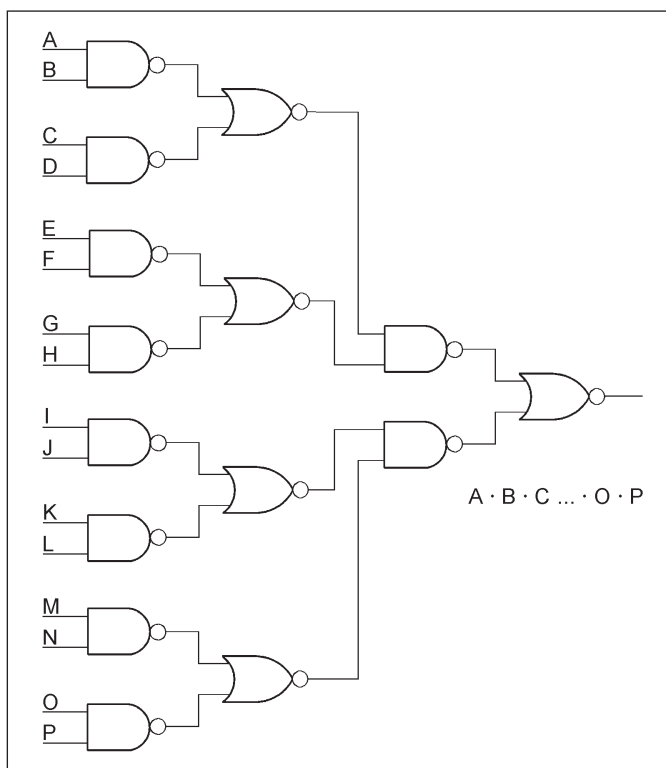


FIGURE 22
An example of a DeMorgan tree. The AND shown here has 16 inputs. It consists of two levels or layers of circuits according to Figure 21a. A similar structure, built with circuits according to Figure 21b, would yield a corresponding OR gate.

A NOR corresponds to an AND of inverted signals. An AND with many inputs can thus be implemented with NAND gates whose outputs are connected to a NOR gate. A NAND corresponds to an OR of inverted signals. An OR with many inputs can thus be implemented by NOR gates whose outputs are connected to a NAND gate. XORs are cascaded like non-inverting gates. A wide XNOR can be built from cascaded XORs with an inverter downstream.

To implement gate functions with even more inputs, cascade an appropriate number of the circuits shown in Figure 21 (Figure 22). If all gates have two inputs, such so-called DeMorgan trees may be built with four, 16, 64 (and so on) inputs. Using gates with three inputs, the smallest DeMorgan tree would have nine inputs. A two-level tree (similar to Figure 22) would have 81 inputs, and so on. (See, for example, [52] for a comprehensive description of DeMorgan trees.)

AND-ing and OR-ing true (not inverted) and inverted signals: Figure 23 depicts the problem together with the solution. Some of the input signals are attached directly, some are to be inverted. Typical applications are to detect particular conditions, like a bit pattern on a data bus and some control signals on, some off, or to combine sensor signals, some of them active-High, others active-Low. The solution follows from DeMorgan's law. A NOR acts as an AND of inverted variables, and a NAND as an OR. Thus, all input signals to be inverted are connected to a NOR or NAND gate, respectively.

Expanding with diodes: I discussed diode gates in my previous article ("Solving Level-Translation and Logic Problems: Using Discrete Components," *Circuit Cellar* 395, June 2023) [37]. Here, where only CMOS buffers or gates are to be driven, the static load current may be neglected. Occasionally, diodes could be a viable solution for expanding the number of inputs (Figure 24). They are small and cheap, and they need no power supply (GND/ V_{CC}) traces on the PCB.

The approach has, however, some caveats. A diode AND increases the Low level, and a diode OR decreases the High level by one forward voltage drop (V_F). Because of the low voltages, we cannot be as generous as in a 24V environment (as done in [37]). The output levels of the diode gates must comply with the input specification of the downstream device.

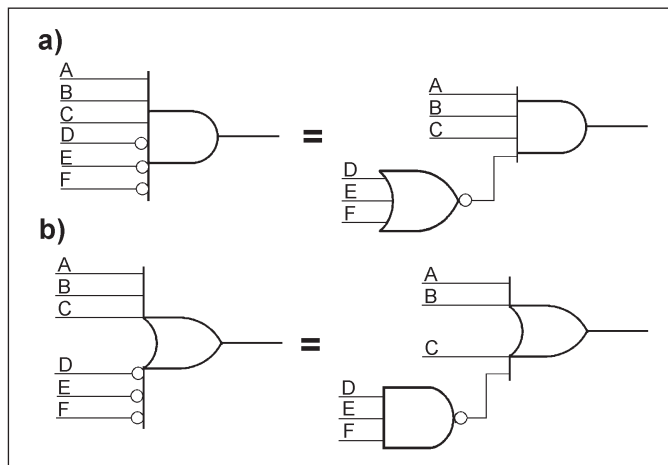


FIGURE 23

True and inverted signals are to be AND-ed or OR-ed. Resorting to DeMorgan's law, we can save on inverters and get by with fewer components. a) shows how to implement a minterm or product term, b) how to implement a maxterm.

The low level must be well below V_{ILmax} , the high level well above V_{IHmin} . Due to their low forward voltage drop, Schottky diodes are an obvious choice. On the other hand, if there are more than a few diodes wired together, their reverse current could be a problem. As a rule of thumb, Schottky diodes could work in CMOS environments with supply voltages well above 2V. The example in Figure 24 illustrates that for a supply voltage (V_{CC}) of 2.5V and a V_F of approximately 0.4V, the levels of the Y output come dangerously near the specified ranges of input levels of the downstream device. When contemplating this solution, strive to keep V_F low by selecting appropriate components. Small-signal Schottkys may be a good choice [33, 34]. They are also available in packages containing, for example, two diodes (isolated or with the cathodes or anodes connected). Bus termination arrays contain more diodes, but their V_F may be too high [35]. RL is to be dimensioned according to $(V_{CC} - V_F)/I_F$. Setting the diode's forward current I_F is a compromise: not too high to keep V_F low, but high enough to ensure proper diode operation and sufficiently fast charging and discharging of the parasitic capacitances (say, between 0.1mA and 1mA). The rise and fall times should be within the limits of the downstream circuit's specification. Beware that Schmitt-trigger inputs may be no remedy here because their high-to-low threshold voltage is considerably lower than $V_{CC}/2$. So, it's wise not to neglect a worst-case analysis.

Logic by wiring: Open-drain outputs can be wired (dotted) together (Figure 25). If at least one of the output transistors

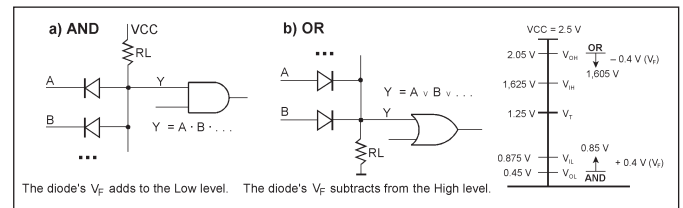


FIGURE 24

Diode gates. a) AND; b) OR. The example on the right shows the influence of the diode's forward voltage, assuming worst-case output voltages of the upstream gates and a V_F of 0.4V. For the level specifications of the 2.5V logic, see Figure 34.

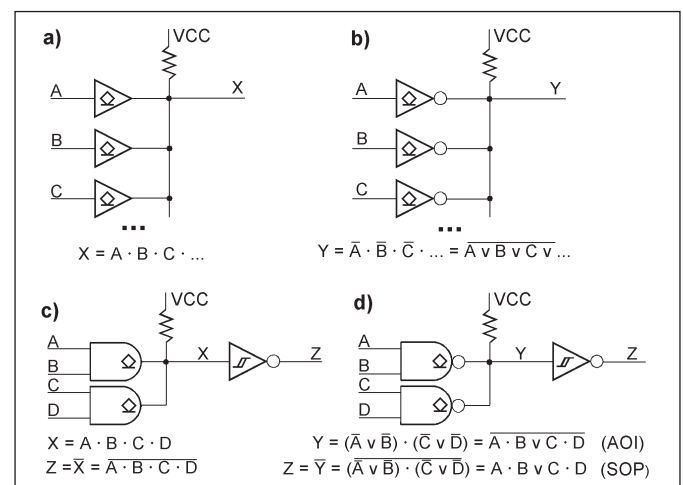


FIGURE 25

Wired (dotted) logic. a) Dotted active-High signals yield an AND function. b) If the dotted signals are active-Low, a NOR results (colloquially called the wired-OR). c) The AND function can be implemented by dotting AND gates. Inverting the output yields a NAND. d) Dotting NAND gates results in an AND-OR-INVERT (AOI) Function. Inverted, it is the sum-of-products (SOP) function.

is switched on, the output level will be low. If all transistors are switched off, the output level depends on the voltage drop across the load resistor. Properly dimensioned, the output voltage will remain in the region of the High level. In a nutshell: Low is caused by the transistor switched on, and High by the load resistor if the transistor is switched off. The term “wired-OR” is widely known. It may be, however, somewhat misleading. It is only correct when we speak of negative logic or signals that are active-Low. If the output is active-High, the circuit acts as a NOR. Concerning positive logic or active-High inputs, the circuit behaves like an AND.

In tiny-logic IC series, the assortment of open-drain devices comprises buffers (non-inverters), inverters, NAND gates, and AND gates. The outputs are mostly specified for voltages higher than V_{CC} (for example, up to 3.6V or even above 5V). Therefore, such components may also be used for level translation. Combinational functions with many inputs can be implemented by wiring together (dotting) an appropriate number of open-drain components. To implement a wired-AND, active-High-signals are to be attached via AND gates or non-inverters, active-Low signals via inverters. The wired-OR function results if active-Low signals are attached via non-inverters and active-High signals via inverters. Strictly speaking, it is, however, a NOR because each active signal will enforce a Low output. Dotted NAND gates yield an AND-OR function with an inverted output, the so-called AND-OR-INVERT (AOI) function.

Dimensioning the load Resistor R_L : In general, this problem has been discussed in my previous article [37]. Here, where only CMOS buffers or gates are to be driven, the static load current may be neglected. More significant is that the Low-to-High transitions are not too slow. This depends on the RC time constant. Therefore, R_L should be as low as possible. As a rule of thumb that leaves a sound margin, you may spend half of the rated Low-level output current I_{OL} of a driving upstream device. Thus, R_L will be calculated

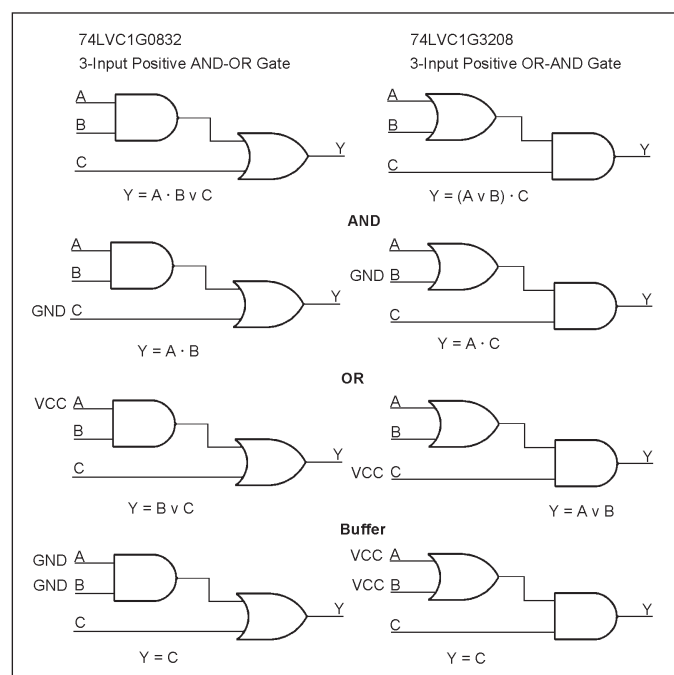


FIGURE 26 Two straightforward multipurpose devices [31] [32].

according to V_{CC} divided by half of the datasheet value of I_{OL} . You may also contemplate getting by with a lower current and compensate for the less steep Low-to-High edges by a downstream Schmitt-trigger.

A straightforward example: Imagine a PCB similar to Figure 11 and assume that 20 sensor outputs are to be OR-ed to excite an interrupt input of an MCU. Let’s begin with active-High sensor outputs. When cascading OR gates with two inputs, 19 devices would be required. A diode-OR would require 20 diodes, the load resistor, and a buffer (with a Schmitt-trigger input, if appropriate). A wired-OR would require 20 open-drain inverters, the load resistor, and the final buffer (with a Schmitt-trigger input). If all sensor outputs are active-Low, the desired OR function can be implemented by cascading AND gates, by a diode-AND, or by a wired-AND built with open-drain non-inverters. If there are sensor outputs of both types, an appropriate solution should be found by making good use of DeMorgan’s law. The wired-OR is the most straightforward solution because it is only necessary to select appropriate open-drain buffers, that is, non-inverters for the active-Low and inverters for the active-High sensors. Sensors with active-Low open-drain outputs

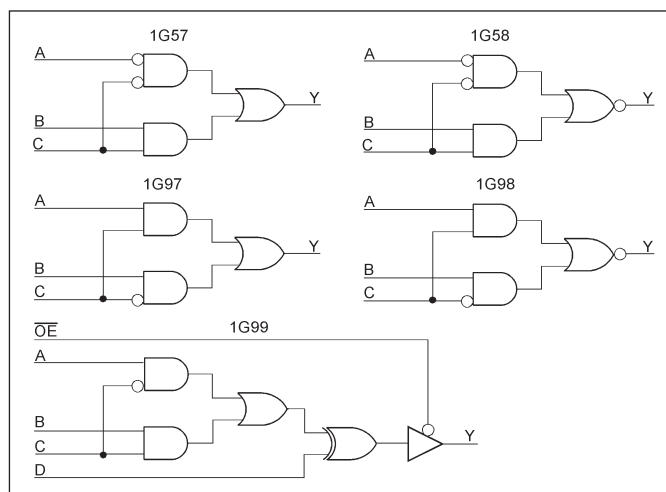


FIGURE 27 Examples of configurable multiple-function gates. They have Schmitt-trigger inputs (that are not shown here).

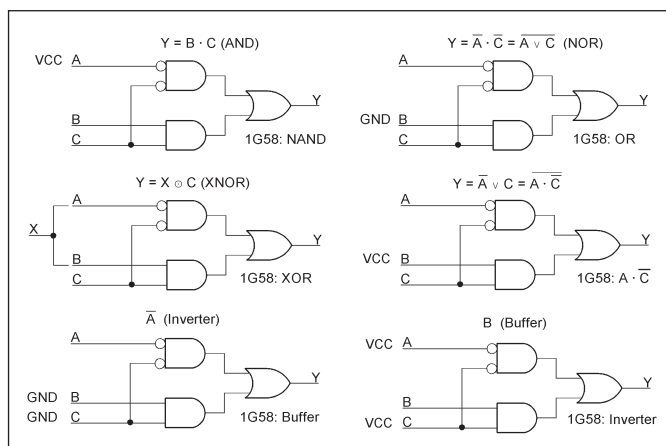


FIGURE 28 Configuration examples (1). The 1G57 [26]. The 1G58 [27] implements the inverted function.

may be wired without additional buffering, provided their output specifications permit.

CONFIGURABLE LOGIC

Semiconductor manufacturers offer some types of tiny configurable gates that can be turned into inverters, buffers, ANDs, ORs, and so on simply by connecting the pins appropriately to signals, ground (= Low), or the supply voltage (= High). Such components (configurable multiple-function gates) can be used wherever straightforward combinational functions are required. They combine some functions that are often needed. In many applications, a configurable gate replaces two or even more single gates. Another advantage is that they can substitute single gates and so reduce the inventory. We begin with two straightforward devices, shown in **Figure 26**. Occasionally, such a 3-input function (AND-OR or OR-AND) will come in handy. For example, an AND-OR could be the last device downstream of cascaded gates implementing a sum-of-products (SOP) function (like $A \cdot B \vee C \cdot D \cdot E \vee \dots$). Beyond that, the devices can substitute AND gates, OR gates, and buffers. Because they lack inversion, their versatility is, however, somewhat restricted.

The theoretical foundation of the more advanced configurable devices are so-called lattices of Boolean functions. Such a lattice results from a single Boolean function by feeding each input with a signal, an inverted signal, a Low level, or a High level. If the original function has n inputs, we may imagine the entire lattice described

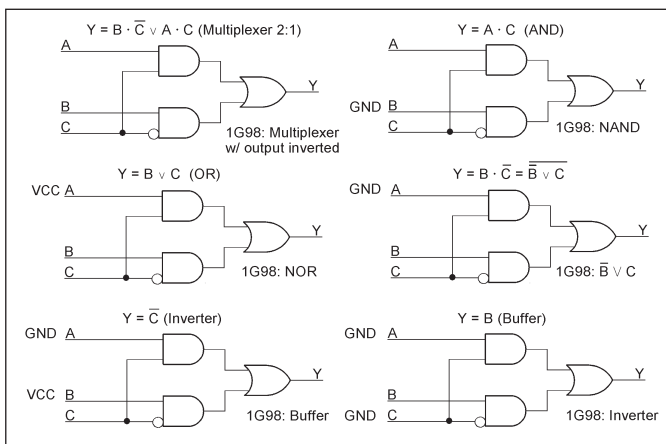


FIGURE 29

Configuration examples (2). The 1G97 [28]. The 1G98 [29] implements the inverted function.

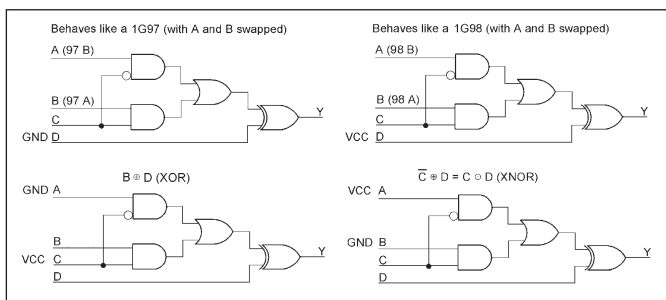


FIGURE 30

Configuration examples (3). The 1G99 [30]. The tri-state output is not shown here. To enable the output, connect OE to GND.

given by 4^n truth tables corresponding to all the combinations mentioned above. The obvious quick-and-dirty approach is to try out all combinations. (Yes, complexity of the order 4^n is far from being quick, and the theory provides more elegant approaches.) The real trick is to find universal Boolean functions whose lattices contain as many usable functions as possible. Additional combinational functions result from making use of DeMorgan's law. But you can't have everything at once. The manufacturers are primarily concerned with getting by with a single package for many typical applications that is also as small as possible.

The really universal component for all possible functions of two variables would be a 4-to-1 multiplexer. That would, however, require a larger package with at least 9 pins (including GND and V_{CC}). According to Boolean lattice theory, to be fully universal occasionally requires inverting input signals. To avoid separate inverters, the manufacturers offer some devices in pairs, with the output and one of the inputs either non-inverted or inverted (**Figure 27**). **Figures 28 to 30** show a few configuration examples. For more details and exhaustive descriptions, we refer to the corresponding datasheets (for example, [25-30]).

The pair 1G57/1G58 has an AND gate with two inverted inputs. The corresponding AND gate of the pair 1G97/1G98 has only one inverted input. With the 1G57/1G58 you can build XOR and XNOR gates, but not a 2-to-1 multiplexer; with the 1G97/1G98 it is the other way around. The 1G99 is basically a 1G97 enhanced with an XOR gate and tri-state output. The functions of the 1G97 or 1G98 can be emulated by wiring the XOR input D to Low or High, respectively. In addition, the circuit can be configured as an XOR or XNOR gate.

UNIVERSAL LOGIC

What we strive for are universal or general-purpose integrated circuits to implement arbitrary combinational functions. In contrast to CPLDs and FPGAs, however, they should do without programming.

The theoretical foundation is Boole's and Shannon's expansion theorem. On n signal lines, 2^n different combinations of Low and High levels may occur. For each of those combinations, an AND gate—in other words, a product term—is provided. Each of the 2^n AND gates has an additional control input. If this input is active, the AND gate will contribute to the function to be implemented; otherwise, it will remain idle. All AND gates are OR-ed together (**Figure 31**). To implement a

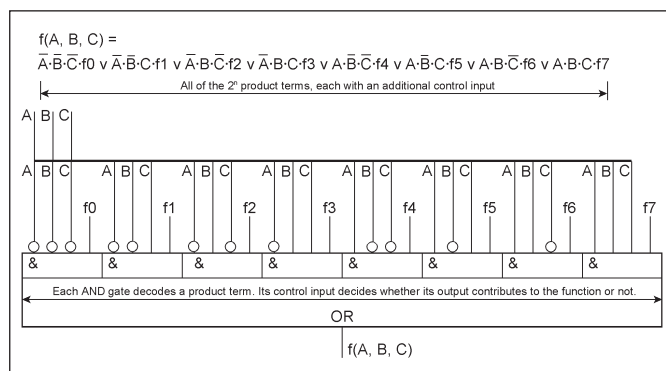


FIGURE 31

Boole's and Shannon's expansion theorem explained. How an arbitrary Boolean function is mapped to a universal sum-of-products (SOP) function.

unstable. These oscillations affect all data outputs, regardless of the combinational function they belong to. Think, for example, of an AND function with the inputs A, B, C, and other functions depending on input signals D, E, F, and so

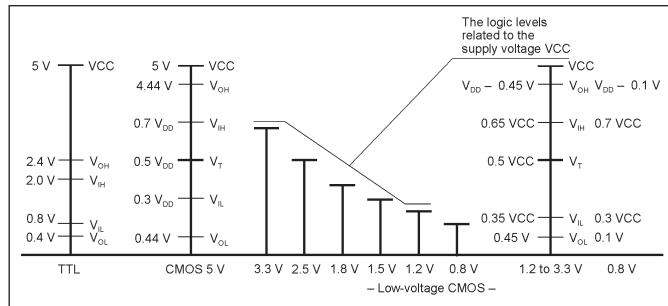


FIGURE 34

Logic level specifications at a glance (some minor differences neglected). TTL and 5-V CMOS are shown for reference.

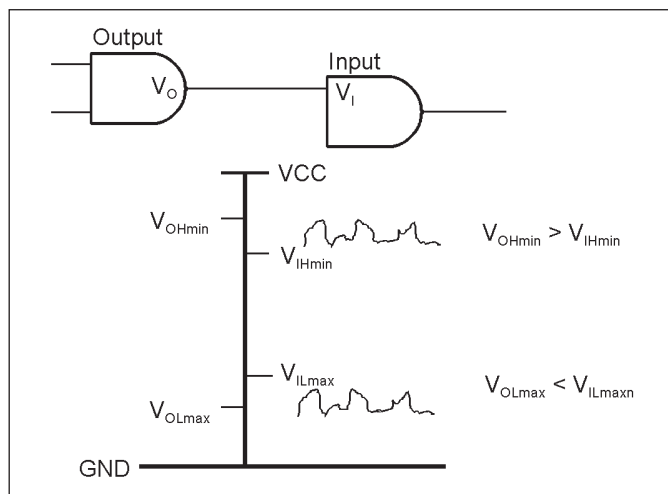


FIGURE 35

Basic requirements for output and input levels. The downstream device should see definite Low and High levels even when noise, ground bounce, and the like are present. So, leave reasonable margins for the maximum Low and the minimum High levels.

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

References [1] to [59] as marked in the article along with product and information sources can be found there.

RESOURCES

Nexperia | www.nexperia.com

NXP Semiconductors | www.nxp.com

Onsemi | www.onsemi.com

STMicroelectronics | www.st.com

Texas Instruments | www.ti.com

Toshiba | www.toshiba.com

on. Implemented with gates, the AND depending on A, B, C will not be affected if, for example, the signal E switches. In the ROM implementation, however, the AND output may show pulses, although the AND function does not depend on the input signal that has changed. On the other hand, ROM-based lookup tables are an expedient solution for FSMs, code conversion, trigonometric functions, and so on. What all such applications have in common is that the stored words and hence the output signals belong together and are subject to synchronous operation.

Why not use an MCU?: Since the advent of the microprocessor, emulating combinational circuitry has occasionally been a topic in application notes [49-51]. When microseconds do not matter, it could be a viable approach because programming MCUs is a much more widespread skill than CPLD/FPGA design. Moreover, there is no need to purchase new development software, programming equipment, and so on.

The most straightforward approach would be to store the truth tables and let the MCU act like a ROM addressed by the input signals (only slower, of course). Such a program must read the input signals, assemble the memory address, read the addressed truth table entry, and emit the output signals. Bit processors, digital simulators, or even fully-fledged Boolean machines belong to the more demanding projects.

SOME GENERAL DESIGN CONSIDERATIONS

Selecting the logic family: The components must fit into the overall design. Above all, it relates to the supply voltage and the logic levels (Table 1 and Figures 34, 35). Additional stipulations to which we (as designers) must comply may concern the IC family, power consumption, speed, packages, soldering processes, testability guidelines, and so on.

When the gates are to be used in a circuit with different supply voltages and logic levels, appropriate level-translation solutions are to be found. For some design challenges, well-suited components are readily available. So skim first the catalogs and selection tables (on the Internet) before trying to find a tricky solution on your own.

Partial power down: A problem may occur when the supply voltage of particular functional units is switched off, for example, to reduce power consumption. Our gates could be without power in an otherwise powered environment or vice versa. Voltages at the inputs of conventional CMOS devices powered off (that is, with a V_{CC} of 0V) may cause short-circuit currents to flow. Provisions to prevent this are called IOFF protection. Most of the low-voltage logic families have this feature (as mentioned in Table 1).

Overvoltage-tolerant inputs: Overvoltage/input tolerance means that the input voltage V_{IN} may rise beyond the supply voltage V_{CC} . Typically, the limit is the rated maximum supply voltage.

Voltage-level translation: Let us assume a particular supply voltage (V_{CC}). If the input voltage is lower, you need a compliant device, or you will have to interpose a level-translation circuit. If the input voltage is higher, you should check whether your logic family tolerates it (Table 2, Figure 36). Otherwise, you may resort to level-translation devices or try some trickery, like current-limiting via series resistors ([4] [40]).

General design rules: They are to be followed even when the digital design task seems straightforward. Semiconductor

Logic family, characteristic feature	Direction	Remarks
LV1T devices	Up: 1.2V to 1.8V 1.8V to 2.5V 1.8/2.5V to 3.3V 2.5/3.3V to 5.0V Down: 2.5/3.3/5.0V to 1.8V 3.3/5.0V to 2.5V 5.0V to 3.3V	Output level corresponds to VCC between 1.8 to 5.0V Up translation due to Schmitt-trigger inputs accepting appropriately low input voltages as High levels. Down translation due to overvoltage-tolerant inputs.
AUP1T devices	Up: 1.8/2.5V to 3.3V 1.8V to 2.5V Down: 3.3V to 2.5V	Output level corresponds to VCC between 2.5 to 3.3V. Principles of operation similar to VL1T, but reduced voltage ranges.
Open-drain outputs	Up and down	By connecting the load resistor to a supply voltage lower or higher than the device's VCC
Overvoltage-tolerant inputs	Down	The device tolerates input voltages higher than its VCC.


TABLE 2

Voltage-level translation by tiny devices (according to [45]).

manufacturers provide ample literature to be studied ([52-59] are only a few examples). The most basic rules concern unused inputs, ground and power supply routing, and bypass capacitors. Gross errors that beginners sometimes commit are leaving unused inputs open, letting the auto-router handle the ground and V_{CC} traces like signals, and locating the bypass capacitor far away from the integrated circuit, perhaps in the opposite corner of the PCB.

Testability: When designing in earnest, that is, for manufacturing in series, this aspect should not be neglected. Especially if you contemplate somewhat tricky solutions, like diode gates, open-gate outputs, or universal logic based on multiplexers, ROMs, or even MCUs, you should team up early with the test people.

SUMMARY AND SUGGESTIONS

Unassuming tiny components still play a significant role. They support sophisticated MCUs, FPGAs, and ASICs. In some design projects, where only minor digital problems are to be solved, they may allow to get by without programmable logic, like a CPLD or an FPGA, components which would require you to purchase programming devices and development software. Here we gave an overview of tiny gates and some characteristic peculiarities of designing with them. Furthermore, we discussed basic principles of configurable and universal logic devices. The proposals of substituting gates with ROMs and even MCUs seem to defy our intent not to program. Our excuse is that such components are less costly than FPGAs and that employing them requires only run-of-the-mill computer programming skills without being familiar with digital design, hardware description languages, and CPLD/FPGA programming. The programmable universal Boolean machine is a topic in itself (to be dealt with later). 

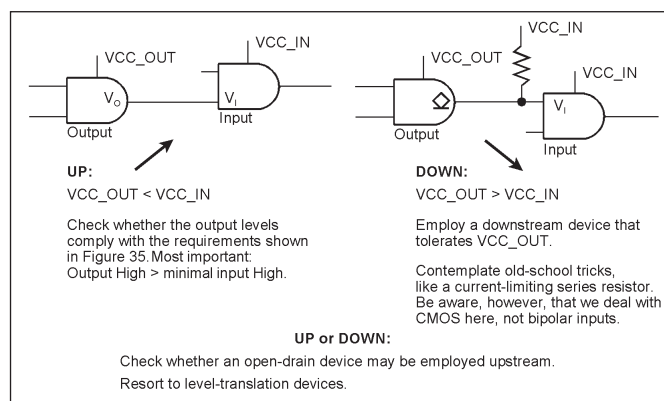


FIGURE 36

A few hints on how to solve level-translation problems.

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 over 50 patent applications and written seven books. (www.realcomputerprojects.dev and www.controllersandpcs.de/projects).