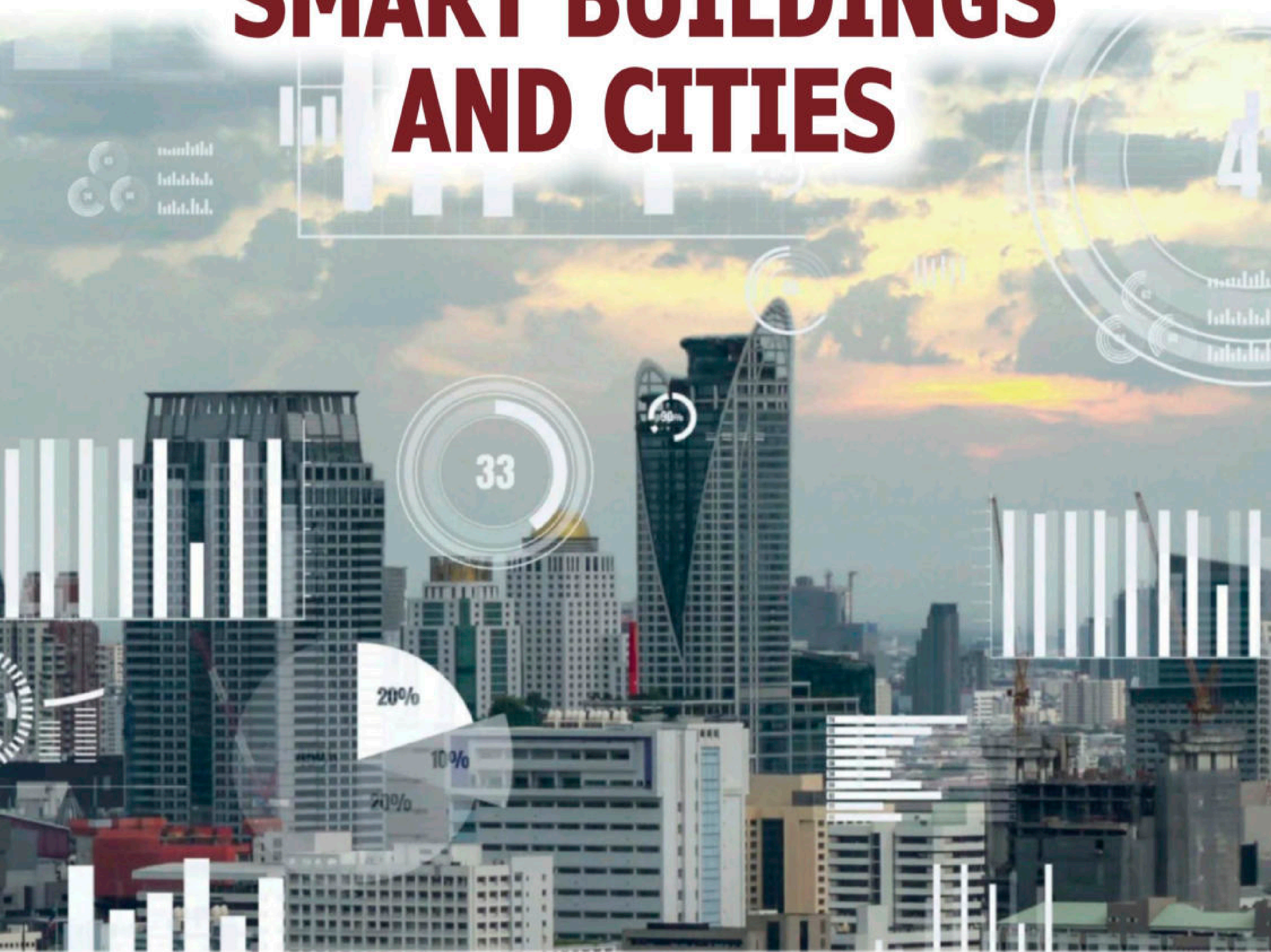




circuit cellar

Inspiring the Evolution of Embedded Design

SMART BUILDINGS AND CITIES



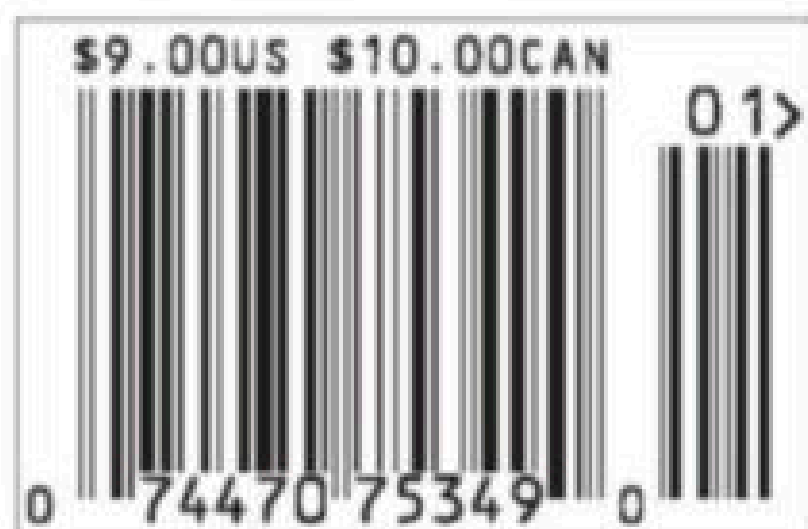
▶ PC/104 Boards ▶ AWS IoT Custom App | Introduction to FPGAs |

Evolution of TV Technology | Level Switch for Non-Conductive Liquids

▶ Potentiostat for Performing Electrochemical Experiments |

Keeping Your Memories Secret—SRAM Read-Back Attacks |

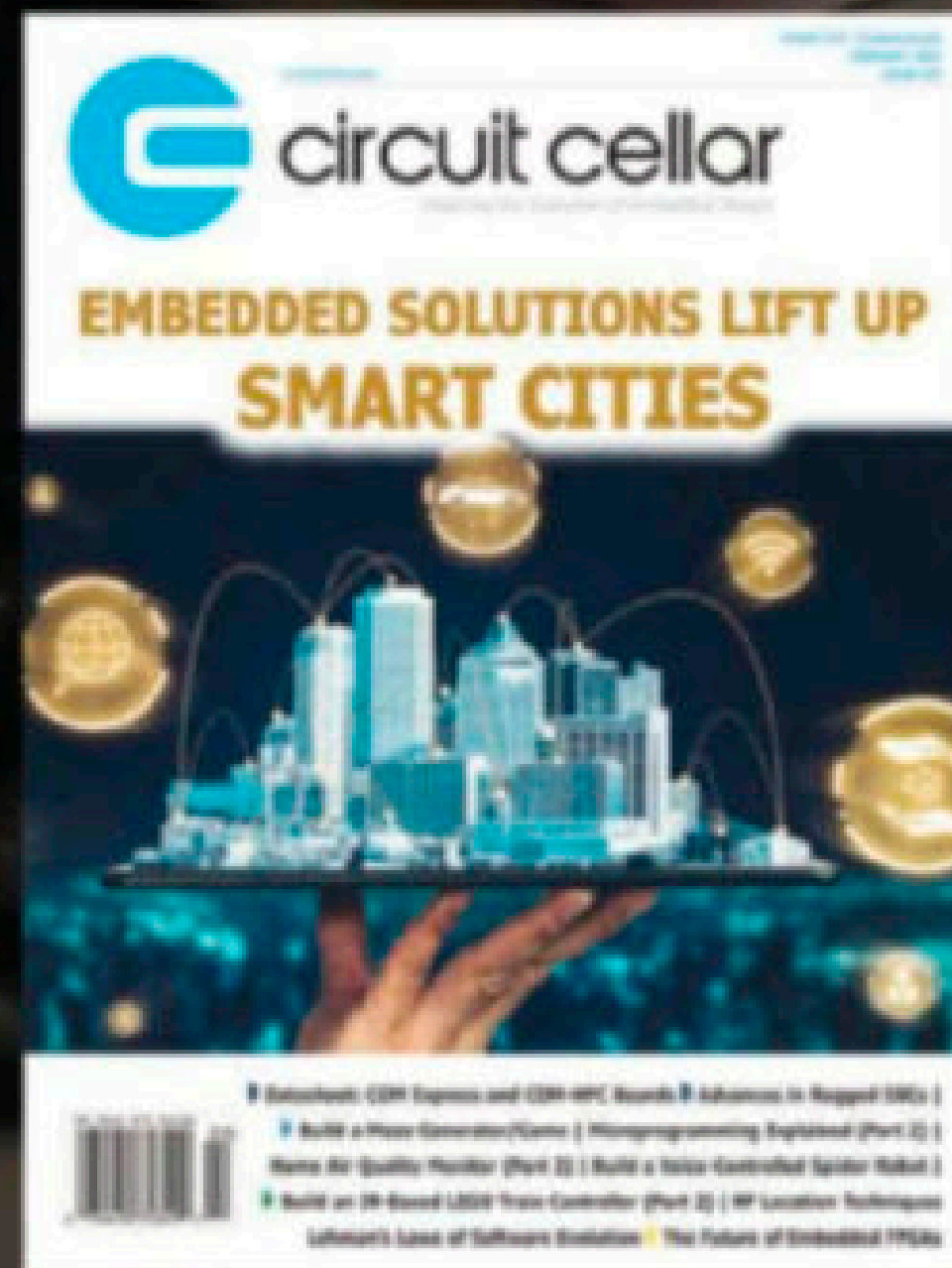
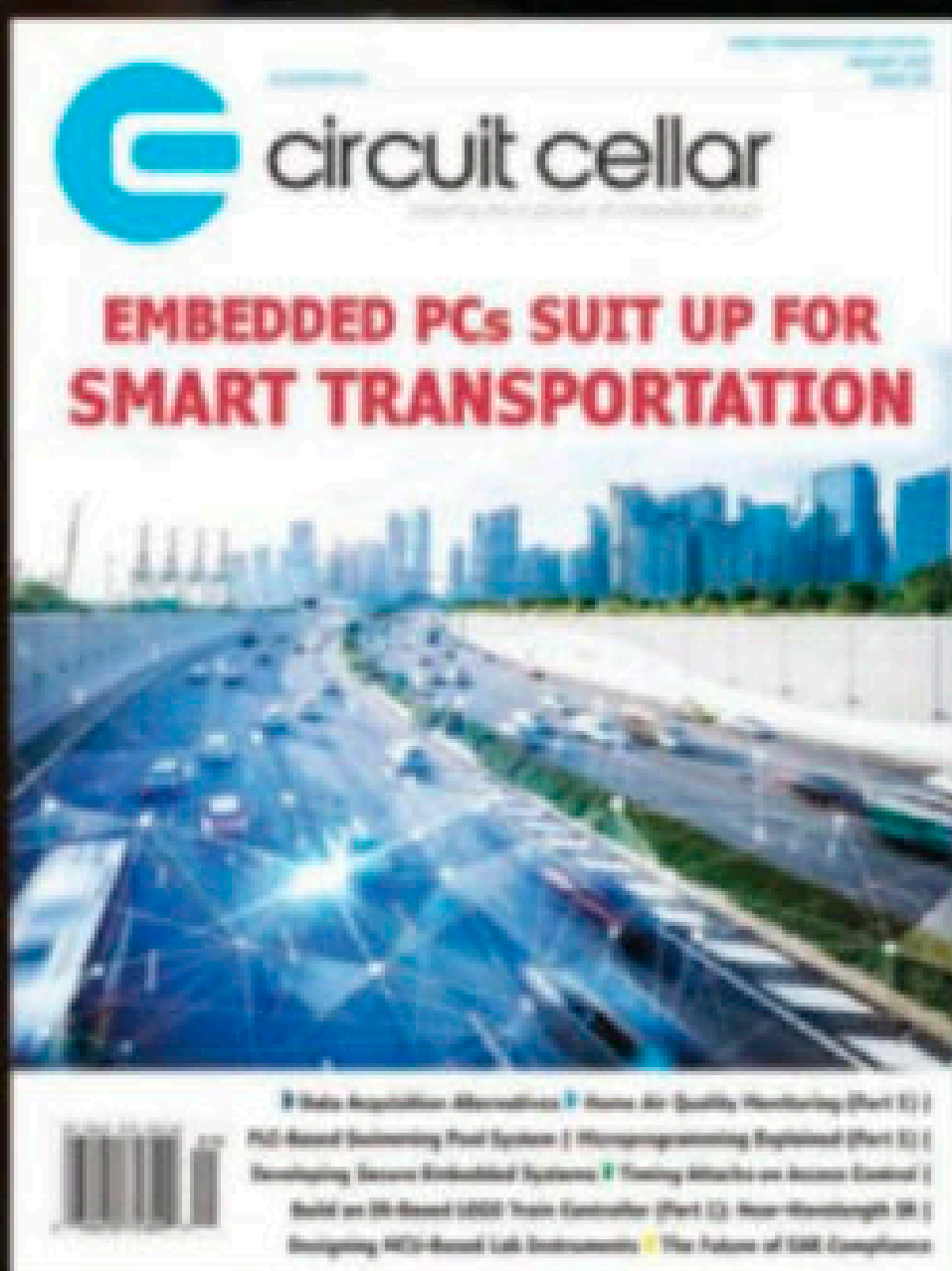
Radar Speed Monitor—Using Raspberry Pi ▶ The Future of Private LTE



Imagine what you will



CC VAULT



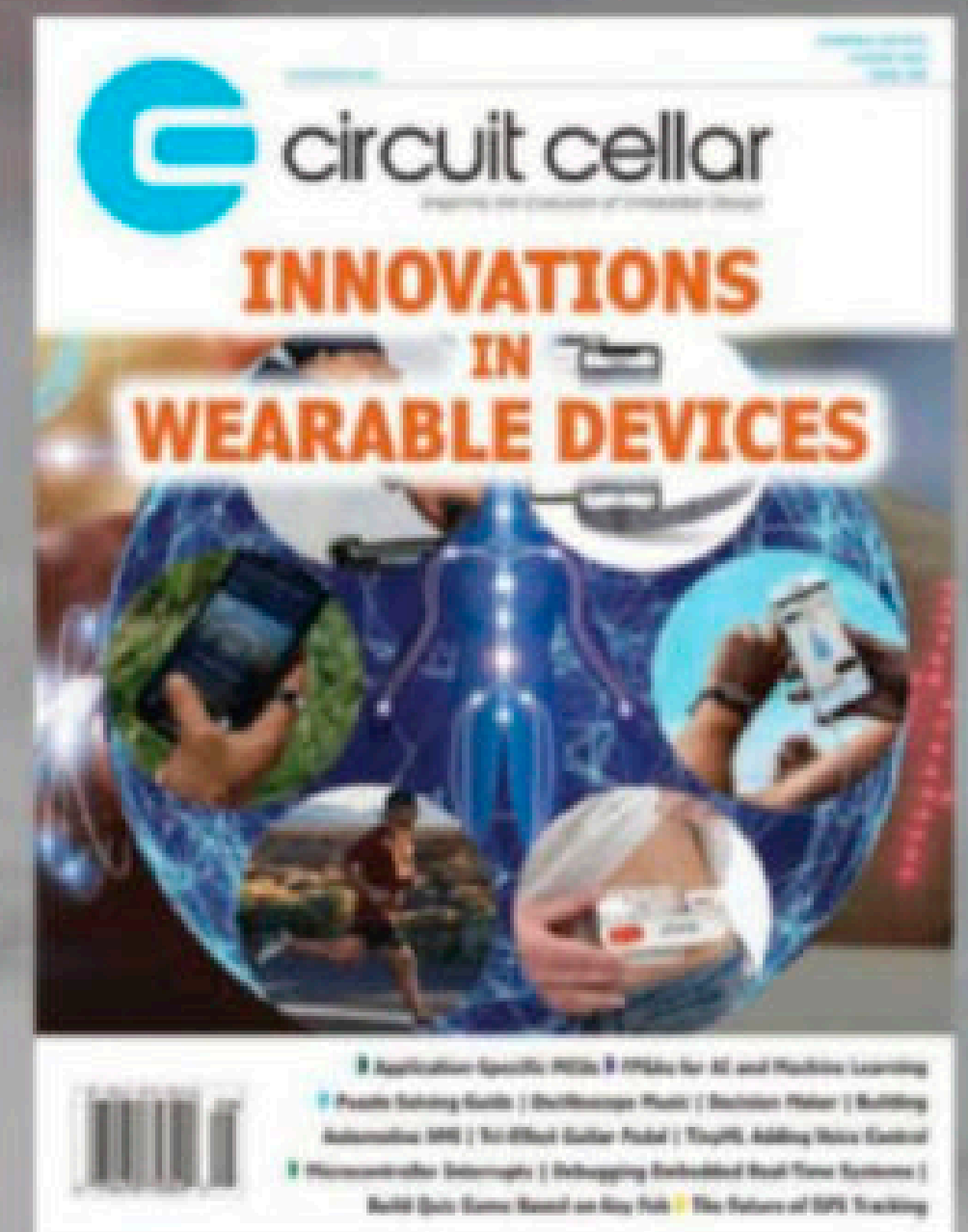
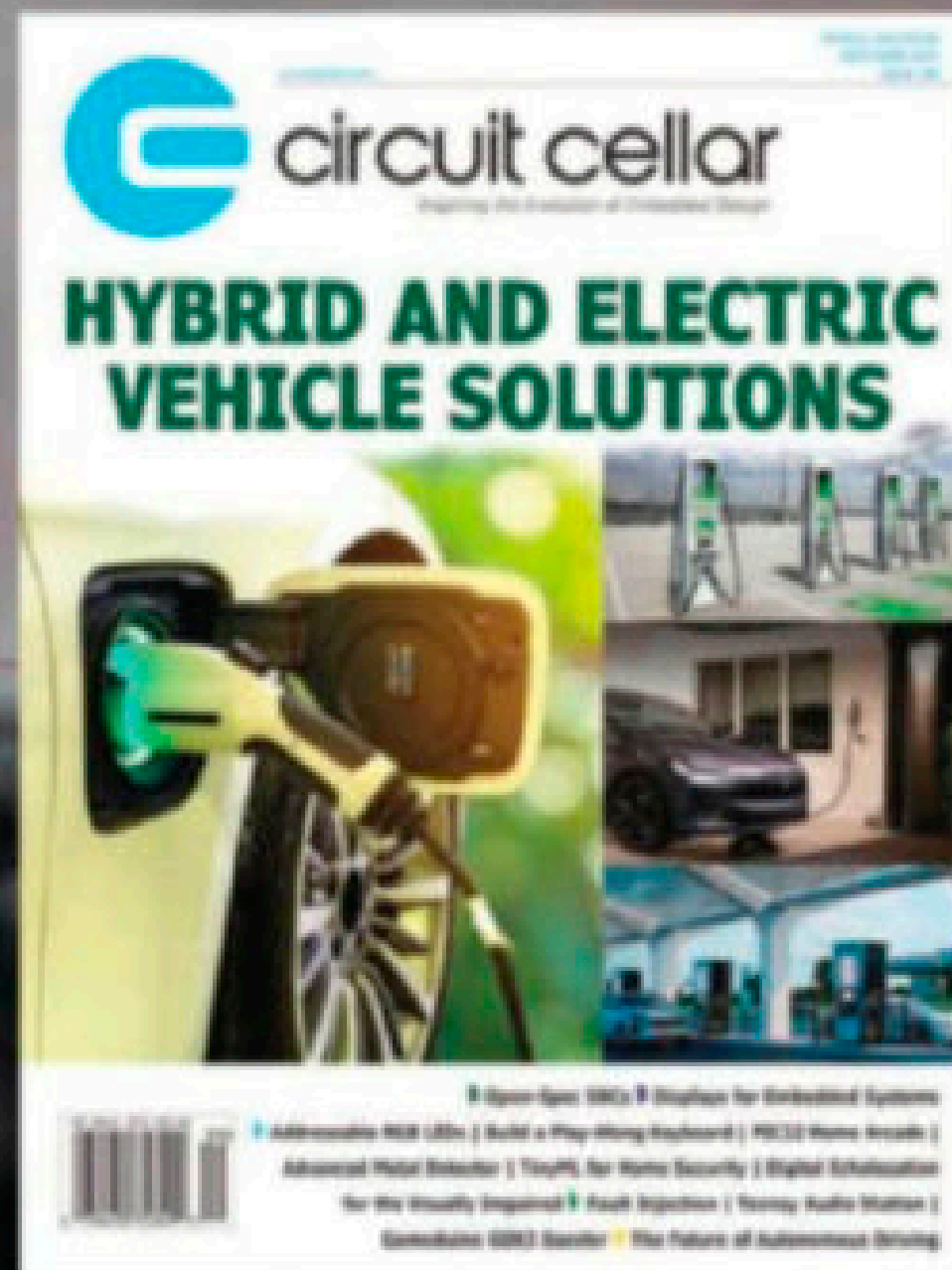
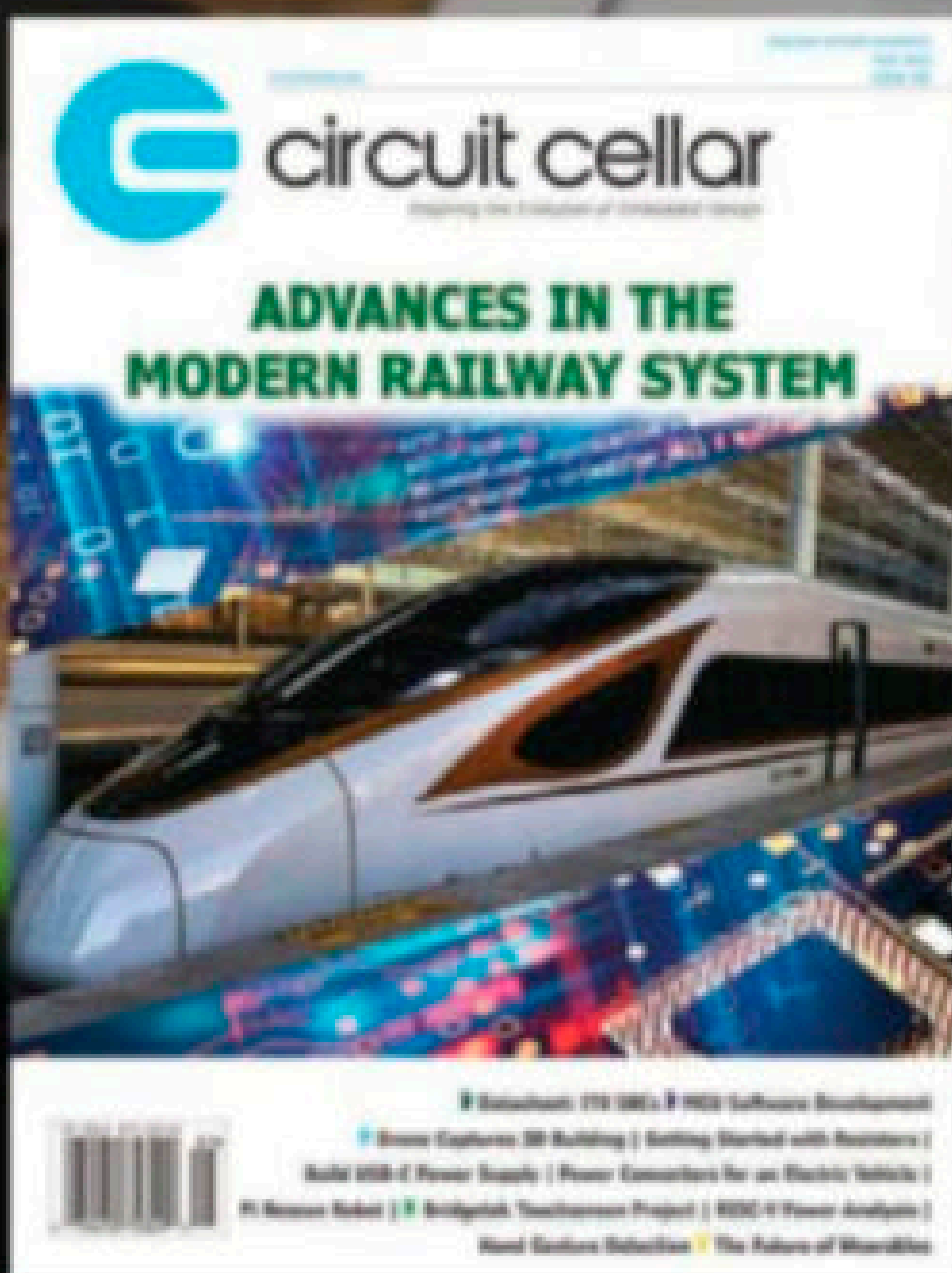
Discover



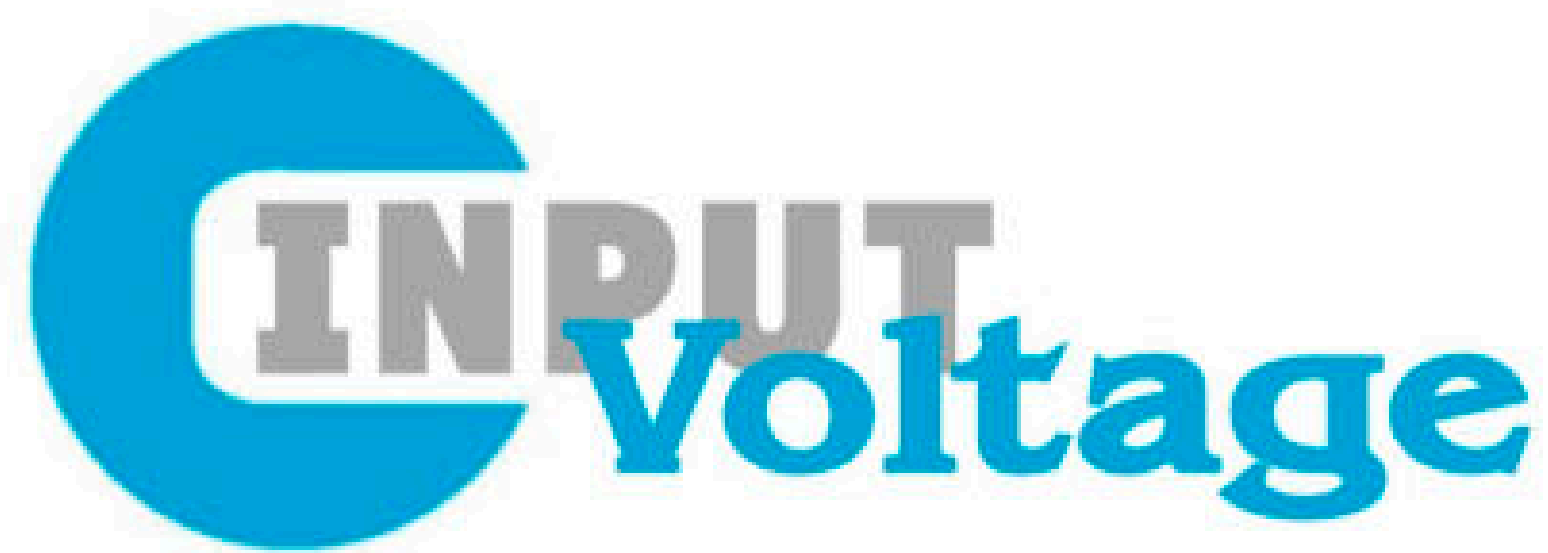
circuit cellar

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.



A New Feature for a New Year

Happy New Year! I hope 2023 will bring you exciting projects, successful designs, and nary a single bug in your code.

To mark this season of making—and promptly forgetting—resolutions, we have made a few small changes to *Circuit Cellar* as well. Namely, we are excited to introduce a new monthly section of our magazine entitled “Technology Feature.” Helmed by industry insider Michael Lynes, Tech Feature will shine a light on the emerging trends and technologies most critical to developing microcontroller- and embedded processor-based systems and design topics that are driving today’s embedded electronics industry. Check out his article this month on smart buildings and smart cities, and what it takes to transition your business to a smart building today.


Remember to check out David Tweed’s section “Test Your EQ” in the back of each issue. In it, he poses questions that challenge your embedded systems knowledge, with answers published the following month.

Field-programmable gate arrays (FPGAs) are an essential technology that can offer more power and flexibility than a typical microcontroller, although they can be more difficult to work with. Thus, Nishant Mittal provides a primer on FPGA basics. He covers topics ranging from their architecture to some aspects of FPGA-based design. Read his piece “An Introduction to FPGAs” on page 14.

Alex Pozhitkov and Brian Millier both write up their experiences in building tools for research, this month. Alex wrote a piece focusing on a component of a gas conditioner used for fuel cell research that his company built. They needed to repurpose a liquid level probe designed for conductive liquids to work with non-conductive liquids. Read more in “Level Switch for Non-Conductive Liquids” on page 23. Brian, meanwhile, discusses the design details of building a potentiostat for electrochemical experiments, something he was tasked with in his time as an instrumentation engineer in the Department of Chemistry at Dalhousie University. His piece, part of his Picking Up Mixed Signals column, can be found on page 36.

Faiz Rahman and Jeff Bachiochi both revisit the past, albeit in different ways. Faiz goes in depth in covering the development of flatscreen TV technology, in “The Evolution of Flat Panel TV Technology” on page 18. As a follow-up to his last piece, Jeff builds a second radar speed monitor, now using a different MCU—the popular Raspberry Pi Pico. Check it out in his column From the Bench, on page 52.

And Colin O’Flynn writes about SRAM read-back attacks in his Embedded System Essentials column. These can occur because many debug lock or security features in microcontrollers allow read-back of SRAM. He demonstrates the attacks themselves, as well as some possible countermeasures. It’s an insightful piece.

I, for one, am excited to see what new technologies the new year will bring, and what’s on the horizon for embedded systems. Thanks for joining me on the path of discovery. I hope 2023 brings you success in all your embedded systems endeavors. 



Sam Wallace

Issue 390 January 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 © 2022 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. 2022 Printed in the United States

OUR NETWORK



SUPPORTING COMPANIES

CCS, Inc.	61
Datakey	C3, 61
DesignCon 2023	13
Digi ConnectCore	62
Micro Computer Control Corp.	61
PCBWay	41
Technologic Systems, Inc.	C4, 61

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
GRAPHICS	Grace Chen
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 Build a Custom App with AWS IoT

Part 1: Embedded IoT Device

By Raul Alvarez-Torrico

14 An Introduction to FPGAs

From Flip-Flops to Applications

By Nishant Mittal

18 The Evolution of Flat Panel TV Technology

From CRT to OLED and Beyond

By Faiz Rahman

23 Level Switch for Non-Conductive Liquids

By Alex Pozhitkov, PhD

26 TECHNOLOGY FEATURE Smart Buildings and Cities

Proactive Planning for Your Business

By Michael Lynes

32 DATASHEET PC/104 Boards

Vital in Harsh Environments

By Sam Wallace

36 Picking Up Mixed Signals A Potentiostat for Performing Electrochemical Experiments



By Brian Millier

46 Embedded System Essentials Keeping Your Memories Secret

SRAM Read-Back Attacks

By Colin O'Flynn

52 From the Bench A Radar Speed Monitor

This Time Using Raspberry Pi Pico

By Jeff Bachiochi

63 TECH THE FUTURE The Future of Private LTE Private Cellular Networks in Buildings

By Stephen Kowal

59 : Product News

62 : TEST YOUR EQ

Build a Custom App with AWS IoT

Part 1: Embedded IoT Device

Using the Amazon Web Service (AWS) IoT EduKit as a starting point, I will show you how to build a custom system to monitor the air quality of an electronics workspace. This is the first of a two-part article series.

By
Raul Alvarez-Torrico

FEATURES

In a previous article, “Build IoT Secure Apps with AWS Services” (*Circuit Cellar* issue #383, June 2022) [1], I introduced Amazon Web Service’s (AWS) IoT EduKit learning program, advertised by its creators as an easy and cost-effective way to learn how to build secure Internet-of-Things (IoT) applications using the AWS IoT Core service. The EduKit learning program uses as a reference hardware the Core2 ESP32 IoT Development Kit, developed and manufactured by M5Stack. It offers a software framework and sample code in the form of learning tutorials, along with some community-developed projects for additional reference.

The EduKit learning program aims to help developers easily get into building secure IoT applications. Part of the idea behind it is to streamline the process of writing, compiling and testing embedded IoT firmware, to help developers focus on creating real-world applications. Secure communications at the embedded hardware level is not overlooked within the framework. This could help facilitate its adoption for commercial, industrial, medical and other safety-critical applications.

In my previous article [1], I discussed the EduKit learning program’s workflow, the reference hardware specifications, cloud services, libraries, and development tools. I also made a walk-through of the first three examples presented in the program, explaining key ideas regarding the hardware, software, and cloud services.

In the present two-part article series, I take the EduKit learning program’s “Smart

Thermostat” example as a starting point for building a custom system to monitor the air quality of an electronics workspace, where soldering fumes typically pollute the air. The system will monitor the room air quality and control an air extractor to automatically clean the air when needed. The solution I give is somewhat trivial, because the project is not intended for real-world use. Its main aim is to showcase the interfacing of off-board sensors not supported directly by the Software Development Kit (SDK) and the implementation of custom hardware driver code. I will also discuss the required embedded firmware project configurations to include the new hardware and the build of a custom, serverless application to support the system.

To follow the topics discussed here, you must be familiar with the AWS EduKit platform. If that’s not the case, please first read my previous article [1], or check the first three examples on the EduKit learning program’s website [2].

In Part 1 of this article series, I discuss the interfacing of the off-board sensors selected for this project, the development of custom hardware drivers for them, the required project configurations to include the new hardware for a successful compilation, and a basic testing with the AWS IoT Core server. In Part 2, I will discuss the inclusion of actuators, particularly an air extractor, and the build of a custom, serverless application to support the system on the AWS IoT Core Cloud.

The block diagram for the Smart Air Cleaner system is shown in **Figure 1**. The Core2 ESP32 IoT Development Kit (from now on, “Core2

device” for short) is the embedded controller used for the IoT device in this project. It is based on the Espressif Systems ESP32-D0WDQ6-V3 microcontroller (MCU), featuring dual Xtensa 32-bit LX6 cores that run at 240MHz. The Core2 device comes with a Microchip ATECC608B Trust&GO pre-provisioned, secure element (SE) integrated circuit (IC) that facilitates network authentication and the use of secure connections.

The proposed system will use a carbon dioxide (CO₂) sensor and a particle-density sensor to monitor air quality. An air extractor will clean the air if it becomes too polluted. Sensor measurements will be sent to the AWS IoT Core server via the Message Queueing Telemetry Transport (MQTT) protocol. A serverless application in the same server will generate and send back commands to the Core2 device to control the air extractor.

Amazon’s AWS IoT Core platform uses the MQTT protocol to exchange data with IoT devices. MQTT is great for interconnecting remote devices with small code footprints (such as MCUs) and low network bandwidth. MQTT is designed around the publish/subscribe messaging model, where message types are defined as “topics.” In this model, some devices publish to topics to send data, and others subscribe to topics to receive data. Data exchanged between publishers and subscribers can be text, numbers, binary data, JSON strings, and other data. JSON data format is recommended for interacting with AWS IoT Core platform.

The EduKit learning program showcases the use of the “AWS IoT Device SDK for Embedded C” libraries for writing firmware for the Core2 device. We will be using the same SDK here. With this SDK it is possible to write embedded applications that securely connect to the AWS IoT Core platform via authenticated TLS connections [3]. It also greatly simplifies access to the platform’s MQTT broker to publish and subscribe to topics. The SDK was built with resource-constrained devices in mind—typically MCUs—and facilitates the

interaction with the Core2 device’s SE for easy security authentication.

The AWS IoT Core platform has a feature called “device shadows” that’s used to exchange and synchronize data between MQTT clients. A device shadow is just a JSON document that the platform stores in the cloud and contains current state information of the IoT device (for instance, the Core2). This JSON document is published on a special MQTT topic, and contains “key:value” pairs that store the latest state of the mirrored IoT device. Any system that has access to the device shadow can obtain real-time status updates from the mirrored device. Other systems can even push their own key:value pairs to the device shadow, so the mirrored device can receive data from them as well.

For this project we will use a device shadow to mirror the state of a number of key:value pairs in the Core2 device. Some of them will be “reported” state values—for instance, the readings obtained from the CO₂ and particle-density sensors. Some of them will be “desired” state values, such as commands coming from “topic rules” and the “detector model” (serverless application) running on the AWS IoT cloud platform.

Topic rules are an AWS IoT Core feature that allows reported state variables to be received from an MQTT client (for example, the Core2 device), via its shadow device. It then generates new, desired state values using SQL queries with conditional logic. Those generated state values are then inserted back into the shadow device, so the MQTT client can automatically receive them when a synchronization action is performed. These new, desired state values could be, for example, commands to control actuators.

A detector model is another AWS IoT feature that helps easily implement a serverless application. A detector model is just a Finite State Machine (FSM) with conditional logic capable of receiving input from topic rules, compute state changes, and publish them back to the device shadow as

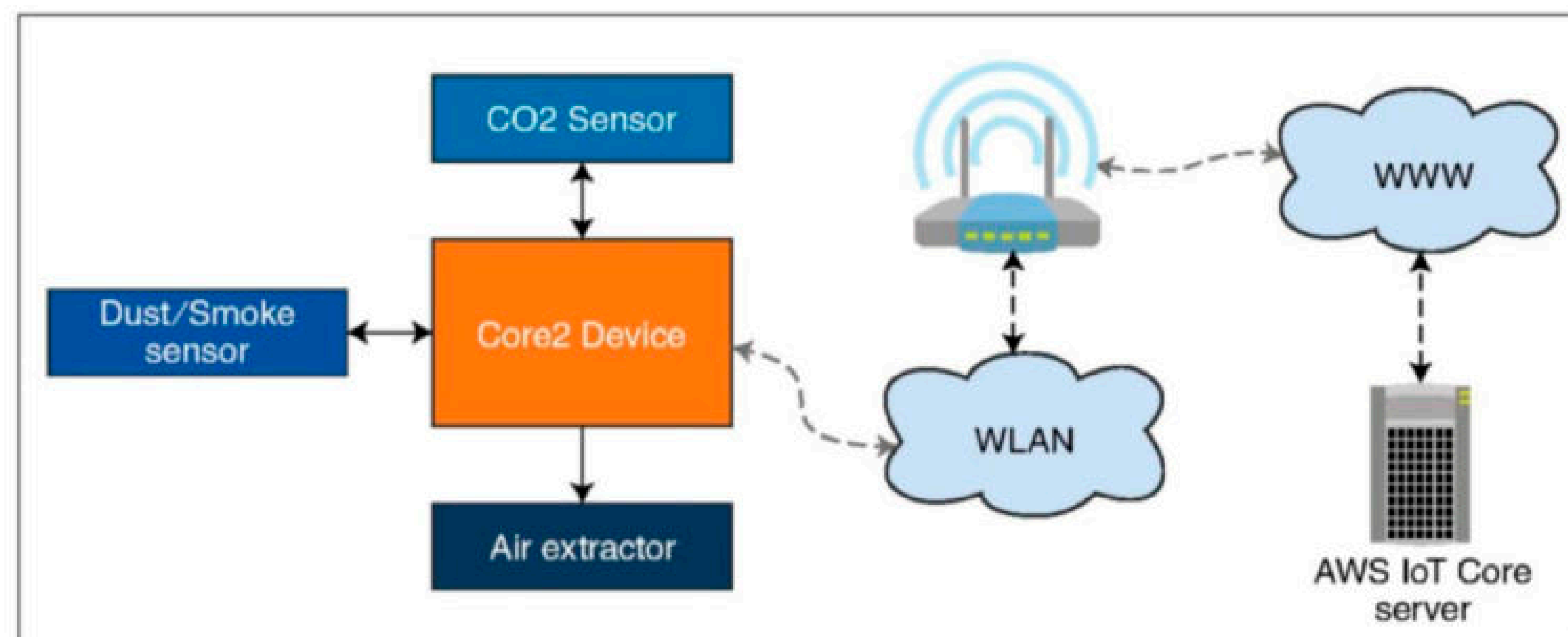


FIGURE 1
Block diagram for the Smart Air Cleaner system

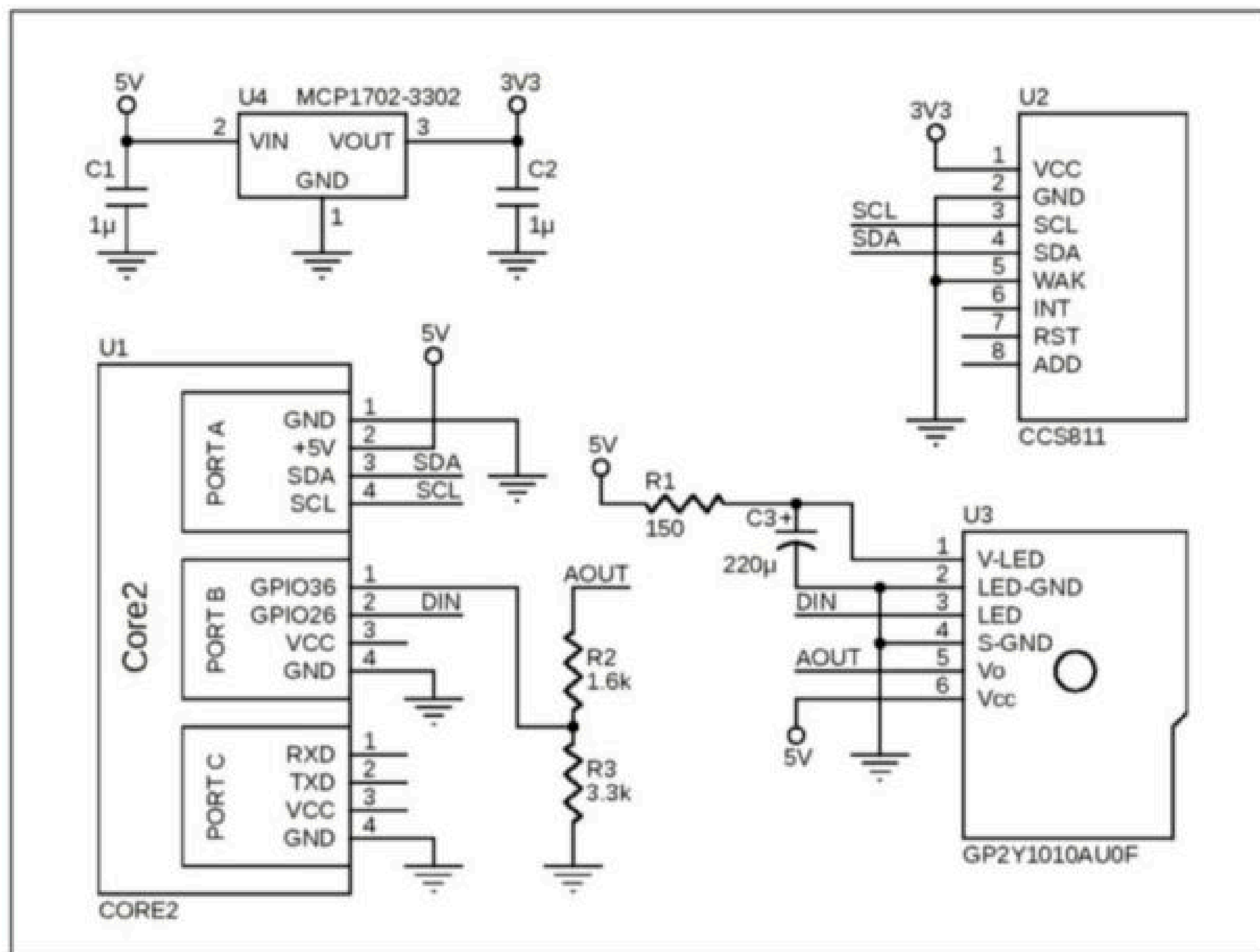


FIGURE 2
Circuit diagram for the Smart Air Cleaner system

desired states. The behavior of the detector model is defined by using JSON syntax.

HARDWARE

Figure 2 is the circuit diagram for the Smart Air Cleaner system without the air extractor. The system is composed of a CJMCU-811 sensor module carrying an ams OSRAM CCS811 ultra-low-power, digital gas sensor. This sensor uses a metal oxide (MOX) gas detector to sense a wide range of Volatile Organic Compounds (VOCs), and it is recommended for indoor air quality monitoring. The sensor uses an I²C port to interface with an application MCU.

The CCS811 implements internally intelligent algorithms that process raw measurements and output values for the Total

VOCs (TVOCs) and the equivalent CO₂ (eCO₂) detected in the air. The computed eCO₂ value fairly represents the real CO₂ concentration, as long as the detected VOCs come from humans. This sensor also implements multiple optimized measurement modes for low power consumption, and an “idle” mode for extending battery life in portable applications. The CCS811 is connected to the Core2 device’s Port A, which exposes pins GPIO32 (SDA) and GPIO33 (SCL) from the ESP32 processor (see Figure 2).

The SHARP/Socle Technology GP2Y1010AU0F is a compact optical dust sensor capable of detecting smoke and other very fine particles. It is an analog-output sensor composed of an infrared (IR) LED-phototransistor pair, diagonally arranged in the device. The sensor detects reflected light from dust or smoke particles in the air, and provides a voltage output level that corresponds to the particle density in µg/m³.

The GP2Y1010AU0F sensor interfaces with an MCU via two pins: a digital LED input pin and an analog Vo output pin. Through the LED input pin, the sensor receives a short pulse that turns on the internal infrared LED. After 280µs, the Vo output pin provides an analog voltage proportional to the air particle density. To read this sensor, the Core2 device must provide first the short digital pulse, and then sample the analog output voltage with an analog-to-digital converter (ADC) input. Next, by applying a transfer function provided by the device’s datasheet, the corresponding µg/m³ particle density value can be calculated.

I connected this sensor to the Core2 device’s Port B, which exposes pins GPIO26 (DAC) and GPIO36 (ADC) from the ESP32 MCU. GPIO26 is used as a digital output to generate the required digital pulse, and pin GPIO36 is configured as an ADC input to sample the output voltage. The hardware prototype for the system is shown in **Figure 3**.

In Part 2 of this article series, Port C from the Core2 device, which exposes pins GPIO13 (RXD2) and GPIO14 (TXD2), will be used to connect the air extractor.

SOFTWARE

I took the Smart Thermostat project from the EduKit learning program as a starting point for this project [4]. Then, I added and modified source code and configurations for the custom sensors and actuators. Because I couldn’t find any suitable CCS811 and GP2Y1010AU0F driver libraries readily available for the platform, I had to port custom ones myself, using libraries from other platforms as references. The AWS IoT EduKit development environment is based on the FreeRTOS real-time operating system.

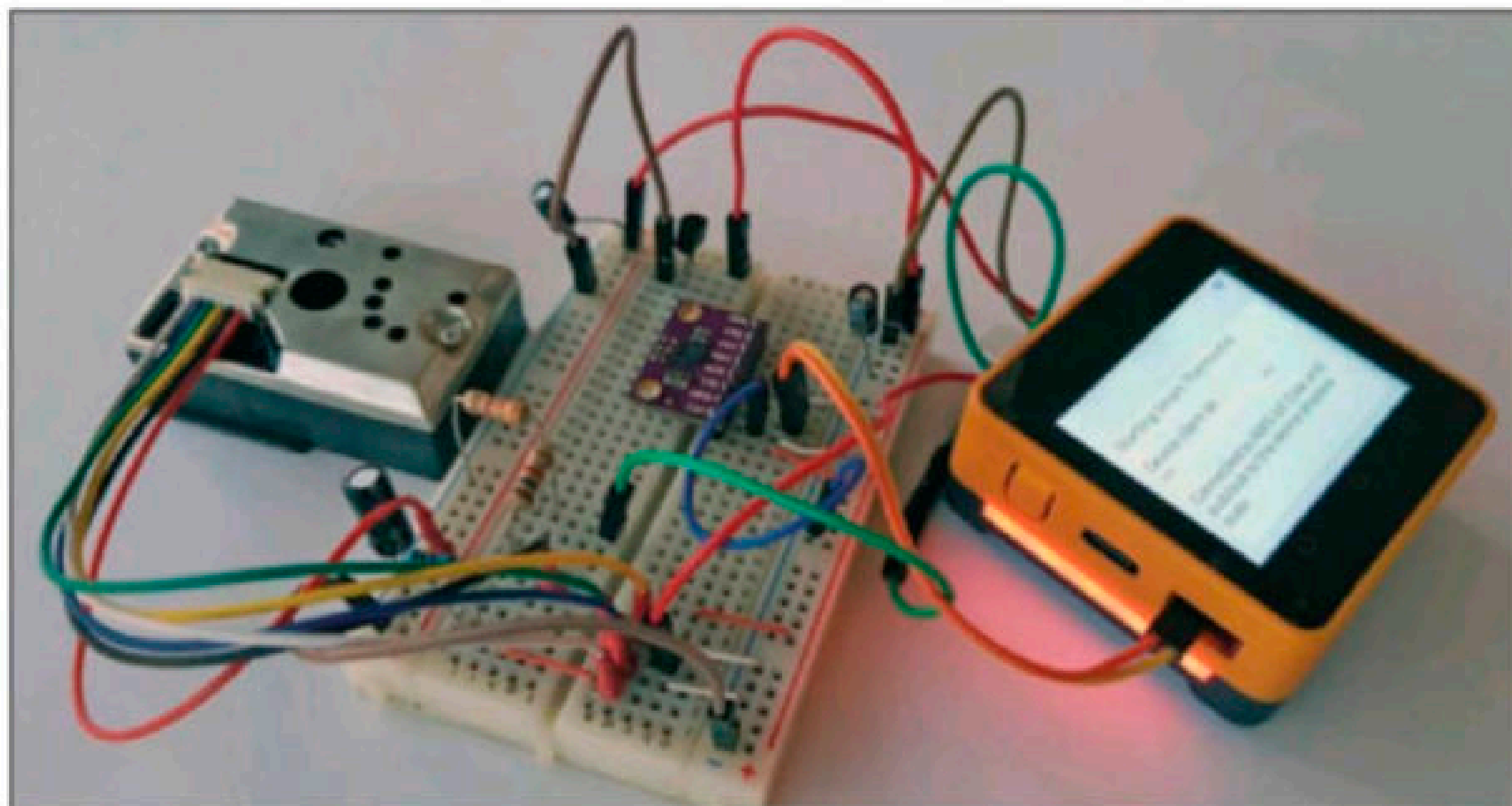


FIGURE 3
Hardware prototype for the Smart Air Cleaner system

Any driver/library must be thread-safe in that context, to avoid race conditions and other synchronization issues.

The driver library I ported for the CCS811 is very simple with no-frills. It implements the minimum requirements to get readings from the sensor. It is composed of a 'ccs811.c' file located inside the project's "...\\Core2-for-AWS-IoT-EduKit\\Smart-Air-Cleaner\\main" folder and a 'ccs811.h' file inside the "...\\Core2-for-AWS-IoT-EduKit\\Smart-Air-Cleaner\\main\\includes" folder.

To compile the drivers without errors, the project's 'CMakeLists.txt' file, also located inside the "main" folder, must be edited to include the new library modules. In that file, the first configuration line must be edited to include the libraries as follows: `set(COMPONENT_SRCS "main.c" "ui.c" "fft.c" "wifi.c" "ccs811.c"`

"gp2y1010.c"). The last two parameters are the libraries for the two sensors in the system.

Source code from the CCS811 driver library is given in **Listing 1**. Lines 14-51 define the `ccs811_Init()` function that initializes the sensor. With line 21, a handle to the I²C communications port is created specifying the I²C address and baud rate. With line 24, a "software reset" is performed in the sensor by writing a reset code to the corresponding register in the device. Line 33 switches the sensor to "application mode" by writing to it a specific code. According to the datasheet, a destination register is not required for this transaction; hence, the 'I2C_NO_REG' value as the second argument. Line 43 sets the sensor's default measurement mode as "mode 1s" (constant power mode, measurements every 1 second).

```

14 void ccs811_Init(void)
15 {
16     const static uint32_t sw_reset = 0x8a72e511; // Software reset code
17     static uint8_t app_start = CCS811_REG_APP_START; // App start register
18     static uint8_t mode_1s = CCS811_MODE_1S; // Mode 1S code
19     esp_err_t err; // ESP error type
20
21     my_port_A_peripheral = Core2ForAWS_Port_A_I2C_Begin(CCS811_I2C_ADDRESS_1, PORT_A_I2C_STANDARD_BAUD);
22
23     // Perform a software reset
24     err = Core2ForAWS_Port_A_I2C_Write(my_port_A_peripheral, CCS811_REG_SW_RESET, &sw_reset, 1);
25     if(!err){
26         ESP_LOGI(TAG, "----->> sw_reset OK");
27     } else {
28         ESP_LOGI(TAG, "----->> sw_reset Error");
29     }
30     vTaskDelay(100); // Wait 100 ms after the reset
31
32     // Switch to sensor's application mode
33     err = Core2ForAWS_Port_A_I2C_Write(my_port_A_peripheral, I2C_NO_REG, &app_start, 1);
34     if(!err){
35         ESP_LOGI(TAG, "----->> app_start OK");
36     } else {
37         ESP_LOGI(TAG, "----->> app_start");
38     }
39
40     vTaskDelay(100); // Wait 100 ms after starting the app
41
42     // Set default measurement mode to "1s"
43     err = Core2ForAWS_Port_A_I2C_Write(my_port_A_peripheral, CCS811_REG_MEAS_MODE, &mode_1s, 1);
44     if(!err){
45         ESP_LOGI(TAG, "----->> CCS811_MODE_1S OK");
46     } else {
47         ESP_LOGI(TAG, "----->> CCS811_MODE_1S Error");
48     }
49
50     vTaskDelay(100); // Wait 100 ms after setting the meas. mode
51 }

```

LISTING 1

Here are lines 14 to 51 of the source code for the ams OSRAM CCS811 ultra-low-power, digital gas sensor's driver library. (The complete listing is available on the Circuit Cellar article materials webpage.)

For the following code line references, please refer to the complete listing available on the Circuit Cellar article materials webpage. Lines 54-81 define the `ccs811_ReadData()` function that reads the eCO₂ and TVOC values from the sensor. Lines 60-

68 can be used to troubleshoot the I²C communications with the sensor, in case the obtained values appear to be incorrect. This code block reads the hardware ID from the sensor and prints it to the terminal window in hexadecimal format. According to

```

10 void gp2y_Init(void)
11 {
12     // Initialize GPIO pins for GP2Y1010AU0F particle sensor
13     Core2ForAWS_Port_PinMode(PORT_B_ADC_PIN, ADC); // ADC input
14     Core2ForAWS_Port_PinMode(GPIO_NUM_26, OUTPUT); // IR LED output
15 }
16
17 void gp2y_Read(float* part_con_val)
18 {
19     static double particle_concentration;
20     static int adc_reading_mv;
21
22     particle_concentration = 0;
23     adc_reading_mv = 0;
24
25     for (int x = 0; x < NUM_ADC_READS; x++)
26     {
27         // GP2Y1010AU0F reading sequence
28         Core2ForAWS_Port_Write(GPIO_NUM_26, false); // Turn on IR LED
29         ets_delay_us(280);
30
31         // Read ADC input
32         adc_reading_mv = Core2ForAWS_Port_B_ADC_ReadMilliVolts();
33         // ESP_LOGI(TAG, ">>>>>>>>> adc_reading_mv : %d", adc_reading_mv); // Just for debugging
34
35         // Sum readings to calculate an average (convert mV to V)
36         particle_concentration = particle_concentration + adc_reading_mv/1000.0;
37
38         ets_delay_us(40); // Wait 40 us
39         Core2ForAWS_Port_Write(GPIO_NUM_26, true); // Turn off IR LED
40         ets_delay_us(9680); // Wait 9680 us (GP2Y1010AU0F's reading period is >= 10 ms)
41     }
42
43     particle_concentration = particle_concentration/NUM_ADC_READS; // Compute average
44     // GP2Y1010AU0F runs with 5V. We are using a voltage divider to downscale the reading
45     // to an equivalent voltage between 0V-3.3V
46     float rdiv_ratio = 3.3 / (3.3 + 1.6); // Voltage divider resistor ratio
47
48     particle_concentration = particle_concentration/rdiv_ratio; // Upscale reading to a 0V-5V range
49     // ESP_LOGI(TAG, ">>>>>>>>> GP2Y10: Voltage [V]: %.2f", particle_concentration);
// Just for debugging
50
51     // Linear equation: http://www.howmuchsnow.com/arduino/airquality/ to convert read
voltage to ug/m^3
52     // Chris Nafis (c) 2012
53     particle_concentration = 170 * particle_concentration - 0.1;
54
55     // The equation is not adjusted for this particular sensor, so sometimes
56     // negative values of particle concentration are seen when it is near zero:
57     if(particle_concentration < 0) {
58         particle_concentration = 0;
59     }
60
61     *part_con_val = particle_concentration;
62     // ESP_LOGI(TAG, ">>>>>>>>> particleCon: %.2f", particleCon); // Just for debugging
63 }

```

LISTING 2

Source code for the SHARP/Socle Technology GP2Y1010AU0F compact optical dust sensor's driver library

the datasheet, the sensor's ID is '0x81.' Line 71 reads four bytes from the sensor. The first two bytes must be combined to get the eCO₂ value, and the last two bytes to get the TVOC value (not used in this project). Lines 79-80 combine the corresponding bytes into 16-bit numbers by using bit-shift and "OR" logic operations.

Reading the GP2Y1010AU0F sensor is also straightforward. The basic procedure is to generate a digital pulse, sample an analog voltage, and apply a transfer function equation. **Listing 2** shows the driver code for this sensor. Lines 10-15 define the `gp2y_Init()` function that initializes the required ADC input and digital output. Lines 17-63 define the `gp2y_Read()` function that samples the sensor's analog voltage output and computes the particle concentration value. The 'for' loop in lines 25-41 samples the analog voltage 'NUM_ADC_READS' times, and computes an average. This helps to filter some high-frequency noise in the sensor data.

The suggested reading sequence in the datasheet is as follows: First, turn on the sensor's internal IR LED. The IR LED circuit works with negative logic, hence the writing of a 'false' value in line 28. Wait 280µs (line 29), and then sample the

sensor's analog output voltage (lines 32-36). After waiting for 40µs (line 38), turn off the IR LED and wait for at least 9,680µs before repeating the reading procedure (lines 39-40). The datasheet recommends a sampling duty cycle of no less than 10ms. That's the obtained period after adding the three delays.

The ESP32 ADC works with voltage inputs between 0V to 3.3V, but the sensor's maximum output voltage is around 4V. To avoid overloading the ADC input, a voltage divider is implemented for the ADC input at GPIO36, with two resistors of 3.3kΩ and 1.6kΩ (see Figure 2). Line 46 computes the resistor divider ratio, and line 48 divides the averaged voltage by this ratio to obtain the voltage value in the sensor's original output scale. I hard-coded the ratio calculation here for practicality, because my goal was to obtain a working driver as soon as possible. Line 53 applies the transfer function (a linear equation) to the sampled voltage to obtain the particle density value.

DATA SYNCHRONIZATION

Listing 3 contains the most relevant source code I added in the project's 'main.c' file for the custom sensors and

```

31 void aws_iot_task(void *param) {
32     static double particle_concentration = 0;
33     static int adc_reading_mv = 0;
34     static uint16_t eco2 = 0;
35     static uint16_t etvoc = 0;
36     // ...
37
38     // My custom sensor and actuators structs
39     jsonStruct_t eco2Handler;
40     eco2Handler.cb = NULL;
41     eco2Handler.pKey = "roomCo2";
42     eco2Handler.pData = &roomCo2;
43     eco2Handler.type = SHADOW_JSON_UINT16;
44     eco2Handler.dataLength = sizeof(uint16_t);
45
46     // ...
47
48     jsonStruct_t airExhaustActuator;
49     airExhaustActuator.cb = airExhaust_Callback;
50     airExhaustActuator.pKey = "airExhaust";
51     airExhaustActuator.pData = &airExhaust;
52     airExhaustActuator.type = SHADOW_JSON_BOOL;
53     airExhaustActuator.dataLength = sizeof(bool);
54
55     // ...
56
57     // register delta callback for airExhaust
58     rc = aws_iot_shadow_register_delta(&iotCoreClient, &airExhaustActuator);
59     if(SUCCESS != rc) {
60         ESP_LOGE(TAG, "Shadow Register Delta Error");
61     }
62
63     // ...
64
65     // loop and publish changes
66     while(NETWORK_ATTEMPTING_RECONNECT == rc || NETWORK_RECONNECTED == rc || SUCCESS == rc) {
67         // ...
68     }

```

LISTING 3

Source code added in the project's 'main.c' file for the custom sensors and actuators

actuators. For simplicity, I excluded the rest of the code from the original Smart Thermostat project, but full source code and schematics are available on the *Circuit Cellar* article material webpage. Please download the complete listing to check all code line references mentioned.

Lines 31-82 in this listing define the `aws_iot_task` FreeRTOS task function that handles all sensor readings and communications with the AWS IoT Core server. To include the custom sensors and actuators, structs of type `'jsonStruct_t'` must be defined for each one of them. For instance, lines 39-53 define those structs for the CO₂ sensor and the air extractor. The first line in each block defines a handler name for the device. The second line defines a callback function that will execute when a value change in the handler's data member (`'eco2Handler.pData'`) is detected—for example, after receiving an update from the shadow device on the AWS IoT Cloud. We don't need a callback function for the CO₂ sensor, because that data is being generated locally (see line 40). For the air extractor we do need a callback function that will execute commands received from the device shadow (see line 49). Those commands will arrive in its handler's data member (`'airExhaustActuator.pData'`)

The third line in each code block (lines 41, 50) define key names for the key:value pairs in the shadow device containing state data for this device. Lines 42 and 51 set the local variables used to store values for the key:value pairs. For these handlers, those variables are defined in lines 26 and 27. `'roomCo2'` is the integer variable that will store the CO₂ readings in parts-per-million (PPM) from the CCS811 sensor. `'airExhaust'` is the Boolean variable that will store "true/false" commands received from the AWS IoT serverless application, to activate/deactivate the air extractor. Lines 43, 44, 52 and 53 set the type and size for those variables.

Next, for each actuator, you should register the corresponding callback function that will execute when a delta (a difference) in an incoming value is detected. Lines 58-61 register the air extractor callback function, and lines 7-14 define the callback function in question. For now, in this callback we are just printing the received command or state to the terminal window.

For the particle sensor, a similar set of steps is repeated. I omitted those lines in the listing for simplicity. Open the `'main.c'` file in the Smart-Air-Cleaner project folder to see the complete source code (on the *Circuit Cellar* Article Materials webpage).

```

1 while(NETWORK_ATTEMPTING_RECONNECT == rc || NETWORK_RECONNECTED == rc || SUCCESS == rc) {
2     rc = aws_iot_shadow_yield(&iotCoreClient, 200);
3     if(NETWORK_ATTEMPTING_RECONNECT == rc || shadowUpdateInProgress) {
4         rc = aws_iot_shadow_yield(&iotCoreClient, 1000);
5         // If the client is attempting to reconnect, or already waiting on a shadow update,
6         // we will skip the rest of the loop.
7         continue;
8     }
9
10    // START get sensor readings
11    // ...
12
13    // Read the CO2 sensor
14    ccs811_ReadData(&eco2, &etvoc); // read the data from sensor;
15    roomCo2 = eco2;
16
17    // Read the particle density sensor
18    gp2y_Read(&partcon);
19    particleCon = partcon;
20
21    // END get sensor readings
22
23    ESP_LOGI(TAG, "*****");
24    ESP_LOGI(TAG, "On Device: roomOccupancy %s", roomOccupancy ? "true" : "false");
25    ESP_LOGI(TAG, "On Device: hvacStatus %s", hvacStatus);
26    ESP_LOGI(TAG, "On Device: temperature %f", temperature);
27    ESP_LOGI(TAG, "On Device: sound %d", reportedSound);
28    ESP_LOGI(TAG, "On Device: roomCo2 %d", roomCo2);
29    ESP_LOGI(TAG, "On Device: particleCon %f", particleCon);
30    ESP_LOGI(TAG, "On Device: airBlow %s", airBlow ? "true" : "false");
31    ESP_LOGI(TAG, "On Device: airExhaust %s", airExhaust ? "true" : "false");
32
33    rc = aws_iot_shadow_init_json_document(JsonDocumentBuffer, sizeof(JsonDocumentBuffer));
34    if(SUCCESS == rc) {
35        rc = aws_iot_shadow_add_reported(JsonDocumentBuffer, sizeof(JsonDocumentBuffer), 8,
36            &temperatureHandler,
37            &soundHandler, &roomOccupancyActuator, &hvacStatusActuator,
38            &eco2Handler, &particleConHandler, &airExhaustActuator, &airBlowActuator);

```

LISTING 4

These are the details of `aws_iot_task()` infinite while loop from Listing 3. (The complete listing is available on the Circuit Cellar article materials webpage.)

Lines 66-68 show the place where the infinite while loop for this task is defined. Lines 70-79 show code for printing meaningful debugging messages, in case the infinite loop breaks due to any irrecoverable error—such as any persistent network communications error, or if some data buffer exceeds its capacity.

The contents of the while infinite loop from Listing 3 (available on the *Circuit Cellar* article materials webpage) is shown in detail in **Listing 4**. Lines 1-8 ensure that the main body will execute as long as the Core2 device is still connected to the AWS IoT Core server, and there's no shadow device update currently in progress. If there's an update in progress, the while loop will be skipped. If the device fails to reconnect to the server, the loop will be terminated. Line 14 reads the CO₂ and TVOC values from the CCS811 sensor. Line 18 reads the particle concentration value from the GP2Y1010AU0F sensor. Lines 23-31 print the current sensors' values and current actuator states to the terminal window. I'm keeping the Smart Thermostat's original sensors and actuators in the project for a comparative reference with the implementation of the new ones.

Line 33 initializes the JSON document that will contain the key:value pairs to be sent to the AWS IoT Core broker. Those values will then be stored in the corresponding device shadow. Particularly, in lines 35-37 we choose which values we want to send to the cloud. The third argument inside the function contains the number of values we are sending, and the rest of the arguments, from the fourth onwards, are the handles of the values themselves.

DEVICE SHADOW TEST

To test the system so far, we follow the same procedure outlined in the EduKit tutorials for testing the Smart-Thermostat project.

1. Log into your AWS account.
2. Navigate to the AWS IoT console.
3. Go to the "Test" section in the navigation pane and click the option "MQTT test client" (see **Figure 4**).
4. Confirm that the "Subscribe to a topic" tab is active in the "MQTT test client" window.
5. Enter the following topic filter:

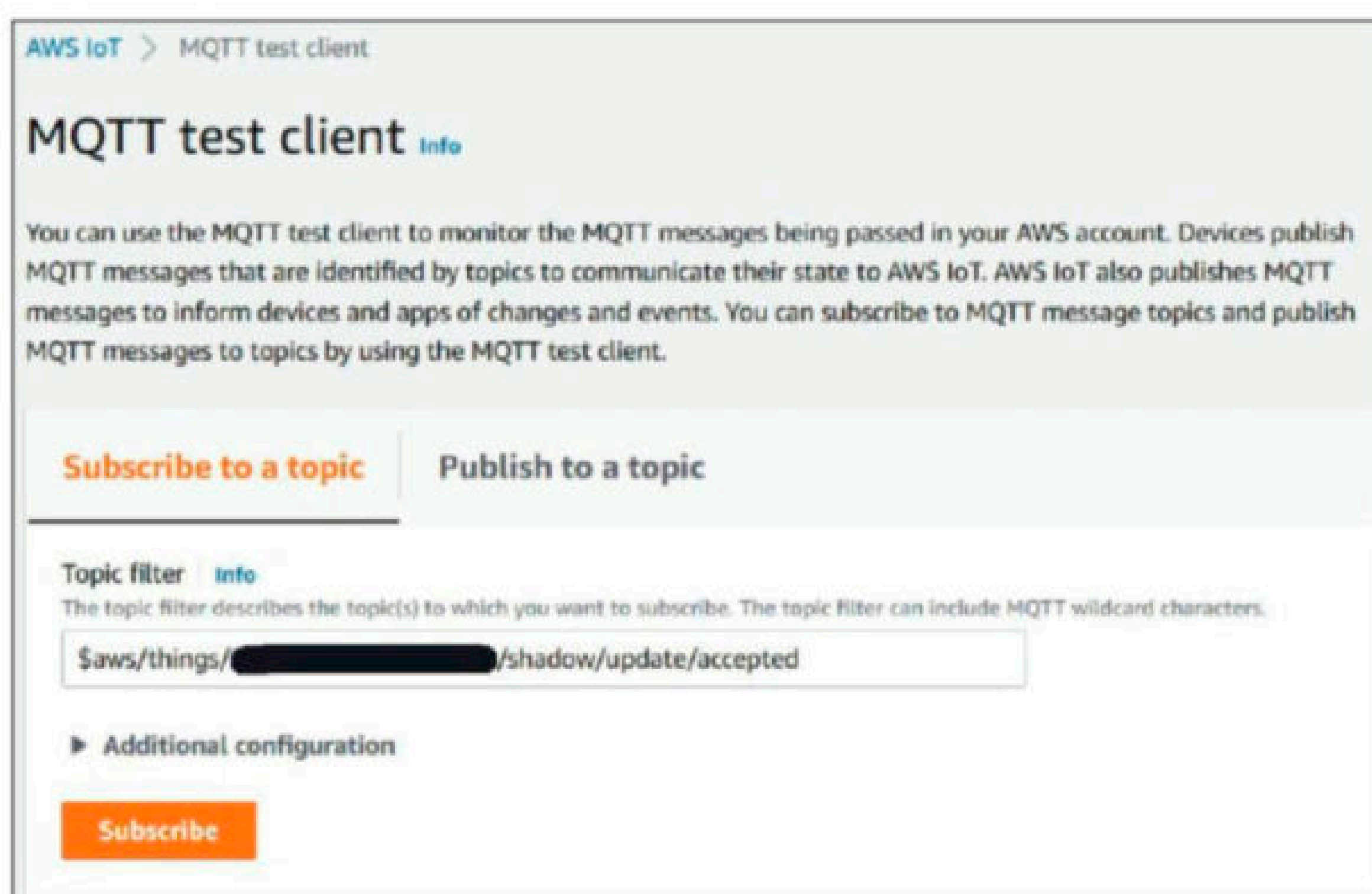


FIGURE 4
MQTT test client—subscribe to topic

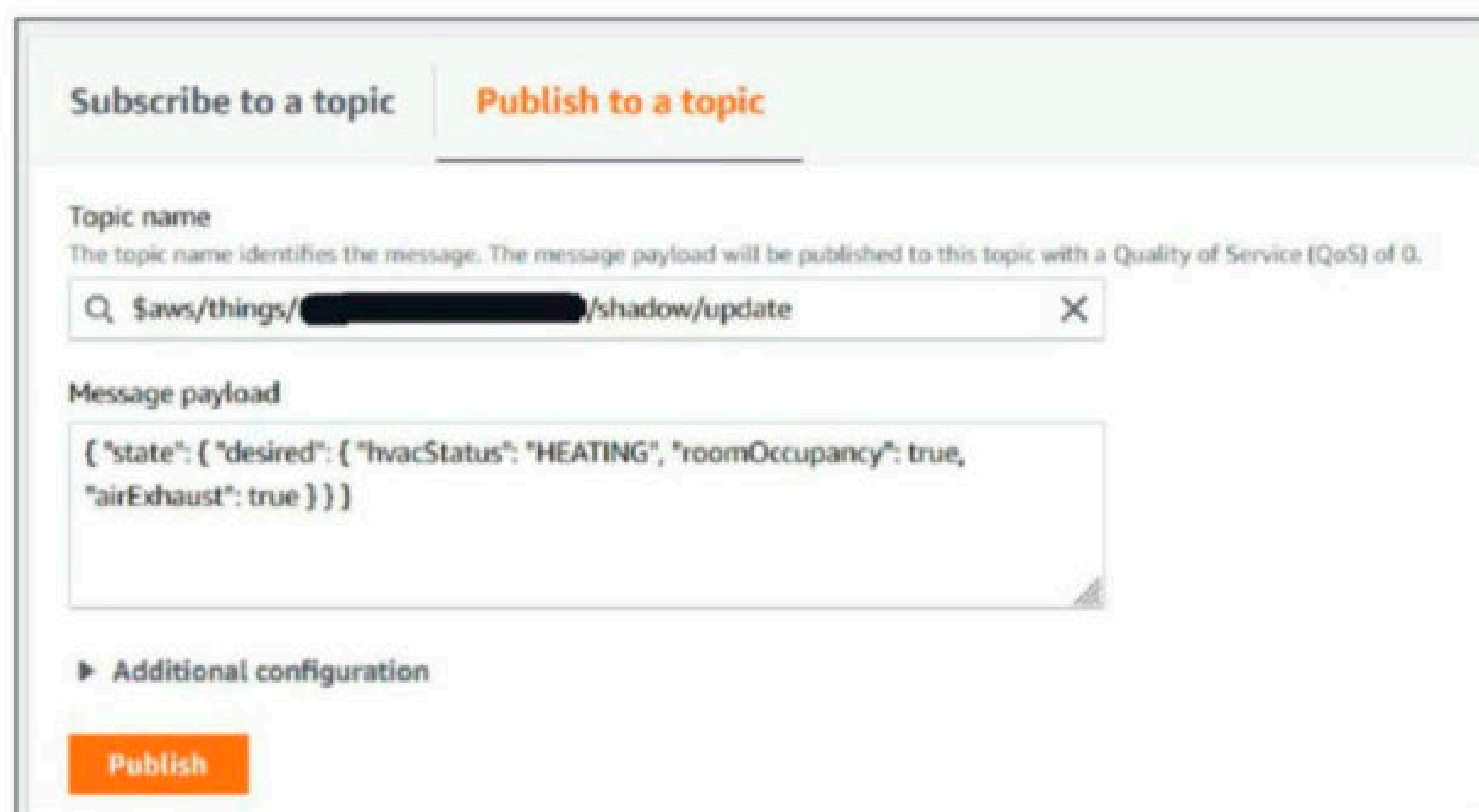


FIGURE 6
MQTT test client—publish to topic



FIGURE 5
MQTT test client—shadow update

FIGURE 7

VS Code terminal window update shadow

```

es[0;32mI (34257) MAIN: *****
*****es[0m
es[0;32mI (34257) MAIN: On Device: roomOccupancy truees[0m
es[0;32mI (34267) MAIN: On Device: hvacStatus HEATINGes[0m
es[0;32mI (34267) MAIN: On Device: temperature 59.805389es[0m
es[0;32mI (34277) MAIN: On Device: sound 18es[0m
es[0;32mI (34277) MAIN: On Device: roomCo2 400es[0m
es[0;32mI (34287) MAIN: On Device: particleCon 35.744244es[0m
es[0;32mI (34287) MAIN: On Device: airBlow falsees[0m
es[0;32mI (34297) MAIN: On Device: airExhaust truees[0m
es[0;32mI (34297) MAIN: Update Shadow: {"state":{"reported":{"temperature":59.805389,"sound":18,
"roomOccupancy":true,"hvacStatus":"HEATING","roomCo2":400,"particleCon":35.744244,"airExhaust":
true,"airBlow":false}}, "clientToken":"
"}es[0m
es[0;32mI (34327) MAIN: *****
*****es[0m

```

“\$aws/things/⟨⟨CLIENT_ID⟩⟩/shadow/update/accepted”.

Replace ⟨⟨CLIENT_ID⟩⟩ with your device ID and choose “Subscribe.” After a few seconds you should see in the lower part of the window new messages arriving in JSON format (see **Figure 5**).

- Go to the “Publish to a topic” tab (see **Figure 6**).
- Enter the following topic filter:

“\$aws/things/⟨⟨CLIENT_ID⟩⟩/shadow/update”
 Replace ⟨⟨CLIENT_ID⟩⟩ with your device ID.

- Replace the “Message payload” window with the following JSON string:

```

{ "state": { "desired": {
  "hvacStatus": "HEATING",
  "roomOccupancy": true,
  "airExhaust": true } } }

```

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.

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

References [1] to [5] as marked in the article can be found there.

RESOURCES

MQTT | www.mqtt.org

Sparkfun | www.sparkfun.com

Visual Studio Code | code.visualstudio.com


Click on “Publish.” The Core2 device’s LED bars should go red for “HEATING,” blue for “COOLING,” and off for “STANDBY” values in “hvacStatus”. In the VS Code terminal window, you will also see the JSON string you just sent as part of a larger device shadow JSON string (see **Figure 7**). Please don’t mind the weird line endings in Figure 7, it’s just a configuration issue in my VS Code installation.

- Change the JSON string published before to an “airExhaust” value of “false,” and publish again. Verify that change appears in the terminal window after a few seconds. Change the other values as well, if you want.

CONCLUSION

One of the first challenges I faced with the implementation of my custom sensors was the lack of readily available drivers for them in the AWS IoT EduKit environment. Because this platform is relatively new, few driver libraries are available for sensors and actuators. Chances are you won’t find drivers available for a particular set of sensors and actuators. So, you will probably have to write or port your own. For people with little experience on embedded systems firmware, sure enough, that’s a downer—especially if the platform aims at helping people with little such experience.

There’s an “EduKit Content Library” [5] webpage with some projects from the AWS team, and some others contributed by community developers. The last time I checked, there were only around 24 projects available. Nevertheless, they can provide an additional reference if you are interested.

In Part 2 of this article series, I will explain how to set up the cloud solution that completes the rest of the system. In particular, I will discuss the configuration of topic rules and a detector model suitable for this application. I will also elaborate further the use of the air extractor and the logic involved in controlling it. 

The Nation's Largest Event for Chip, Board & Systems Design Engineers



Created by engineers for engineers, North America's largest chip, board, and systems event, DesignCon 2023, returns to Silicon Valley. This annual event brings together designers, technologists, and innovators from the high-speed communications and semiconductor communities for three jam-packed days of education and activities.

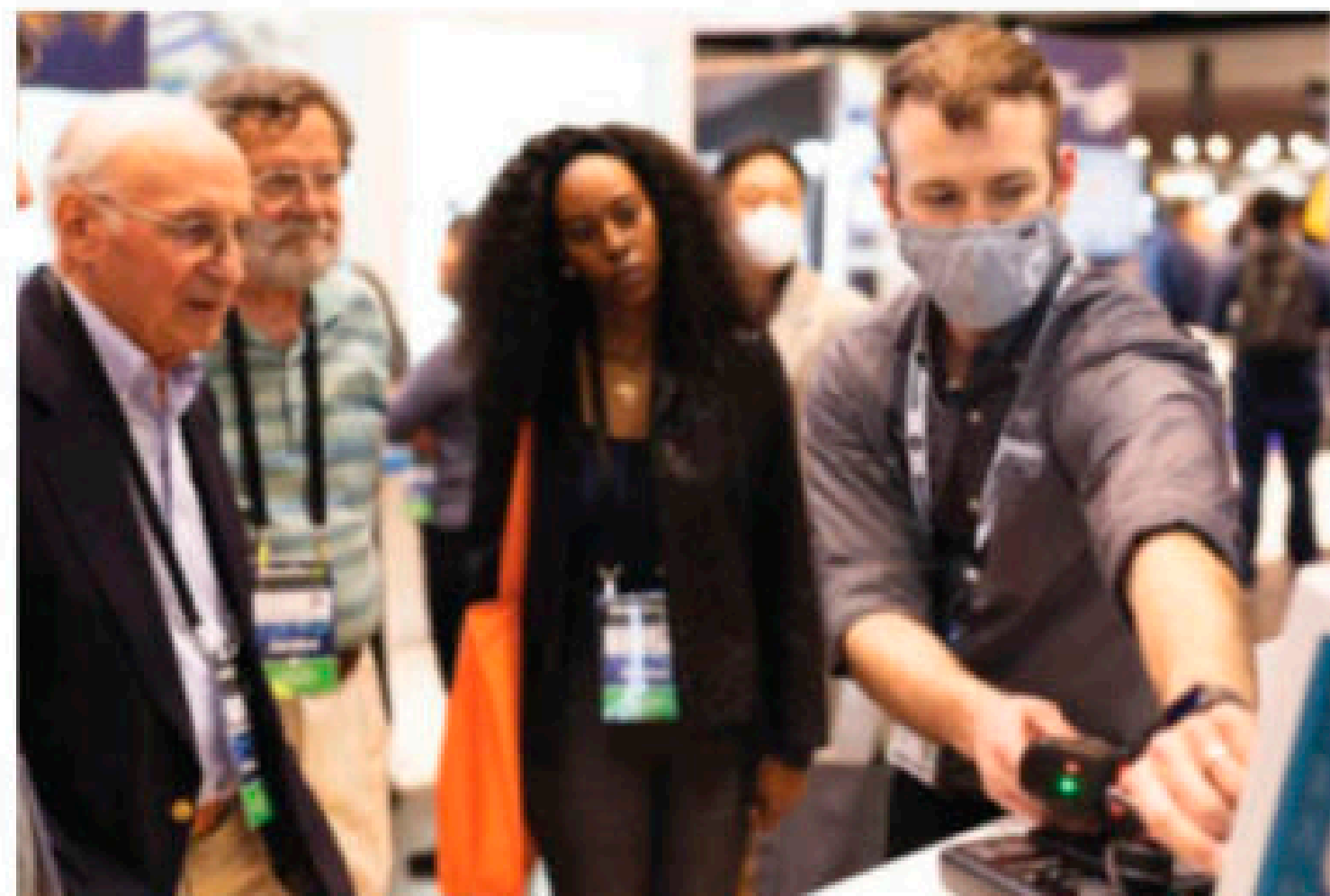
DesignCon is a must-attend opportunity to share ideas, overcome challenges, and source for designs. Join DesignCon at the Santa Clara Convention Center Jan. 31 – Feb. 2, 2023!

Education

- DesignCon's 14 Track Technical Conference
- Drive World Conference Track
- Expert Speakers

Expo

- Keynote Presentations open to all attendees
- Free Education at Chiphead Theater
- Expo Hall with Leading Suppliers
- Interactive Demos



Save 15% on conference registration or receive a free expo pass with code **KCK**

Register at

DesignCon.com

Host Sponsor:

Amphenol

An Introduction to FPGAs

From Flip-Flops to Applications

FEATURES

By
Nishant Mittal

FPGAs are popular with designers around the world for their flexibility and reprogrammability. In this article, I will cover some of the basics of FPGAs—from their architecture to some aspects of FPGA-based design.

Field-programmable gate arrays (FPGAs) are semiconductor devices that contain an array of configurable logic blocks (CLBs) which are connected via programmable interconnects. They were developed in the late '80s to address processing, industrial, automotive, and even aerospace applications. Since then, the number of applications centered around FPGAs has increased exponentially. In this article we will explore the architecture of FPGAs, as well as some aspects of creating an FPGA design.

FPGAs are categorized based on their different end-user applications, such as those in automotive, defense and space. **Table 1** shows some examples of devices for each of those industries.

FPGA resources are categorized by the following parameters:

- Number of Available I/O's (Differential and Single-Ended).
- Amount of Internal memory (RAM)
- DSP Slices (Complex Multipliers)
- Connectivity Ports (Serial Interfaces, Memory Interfaces, etc.)

Pin-compatible devices with more resources than are needed should be considered while choosing an FPGA for a new design, to allow for enough design margin.

As a basic rule, 60% to 75% utilization of an FPGA's logic resources—like slices and LUTs—can be considered a higher limit, beyond which congestion and timing issues can arise.

FPGAs are used in a broad range of applications in technologies like 5G wireless, embedded vision, industrial Internet-of-Things (IoT), and cloud computing. This is made possible by the availability of ARM processor cores and C-based compilers for a given FPGA platform.

Another important application of FPGAs is in the field of platform engineering, an increasingly in-demand discipline in the industry. In platform engineering, SoCs and ASICs can be prototyped on FPGAs and validated even before the tape-out of the final device. Let's now try to understand FPGA architecture and its workflows.

ARCHITECTURE

Modern FPGAs come with their programmable logic integrated with the processor system core in the same chip—converting an FPGA into a programmable SoC. **Figure 1** is the various logic components inside a Zynq Ultrascale+ FPGA. Typically, these consist of a configurable logic block (CLB), digital signal processing (DSP), a transceiver, input/output (I/O) blocks, memories, and Federal Development and Certification Environment (FDCE) blocks.

CLBs: A CLB's logic function is defined and configured by the FPGA user. A typical CLB contains a set of lookup tables (LUTs) and D-type flip-flops with clock enable (FDCE). When the logic is programmed into the FPGA, each CLB takes a part of the logic and configures itself to perform that function. **Figure 2** is a typical CLB contained in an FPGA.

DSP Slices: Many algorithms, such as AI, require a lot of math and signal processing

to handle a specific scenario. DSP broadens the scope of the overall FPGA structure, so that complex algorithms such as filtering or matrix multiplication are performed with significantly greater efficiency than they would be using many CLBs.

Transceivers: Several transceivers available in complex FPGAs can transmit and receive data at a high data rate via a Serializer/Deserializer (SERDES), a pair of functional blocks which can rapidly send thousands of signals through a single transceiver path. A set of high-speed transceiver blocks can be connected using a GT cable to transmit and receive data at a rate of tens of Gbps.

Figure 3 is a block diagram of a SERDES.

I/O Blocks: I/O blocks are a vital part of an FPGA—they are where the FPGA’s data connects to external circuitry. I/O ports are defined in a Xilinx Design Constraints (XDC) file with the extension .xdc. In an XDC file, one needs to provide a pin number along with a logic level voltage (such as LVCMOS or LVTTTL) based on the external device to which it’s connected. It’s important to refer to the I/O bank voltages of the I/O ports to select the correct port.

Consider the Zynq UltraScale+ device as an example. Each I/O bank contains 52 SelectIO interface pins. In some devices, there are high-range (HR) I/O mini-banks containing 26 SelectIO pins, each with their

Category	Device
Xilinx Automotive (XA)	XA Zynq UltraScale+ XA Zynq-7000 XA Artix-7 Kintex
Xilinx Defense-Grade	Artix-7Q Kintex-7Q Virtex-7Q
Xilinx Space-Grade	Virtex-5QV Virtex-4QV

TABLE 1

Some FPGA devices, organized by three of the major industries in which they are commonly found

own independent power supply and VREF pin. The SelectIO pins can be configured to various I/O standards, either single-ended or differential. Single-ended I/O standards are, for example, LVCMOS, LVTTTL, and POD. Some examples of differential I/O standards are LVDS, SLVS, and LVPECL.

Certain rules must be obeyed while combining different input, output, and bidirectional standards in the same bank:

- Output standards with the same output VCCO requirement can be combined in the same bank.
- Input standards with the same VCCO and VREF requirements can be combined in the same bank.
- Input standards and output standards

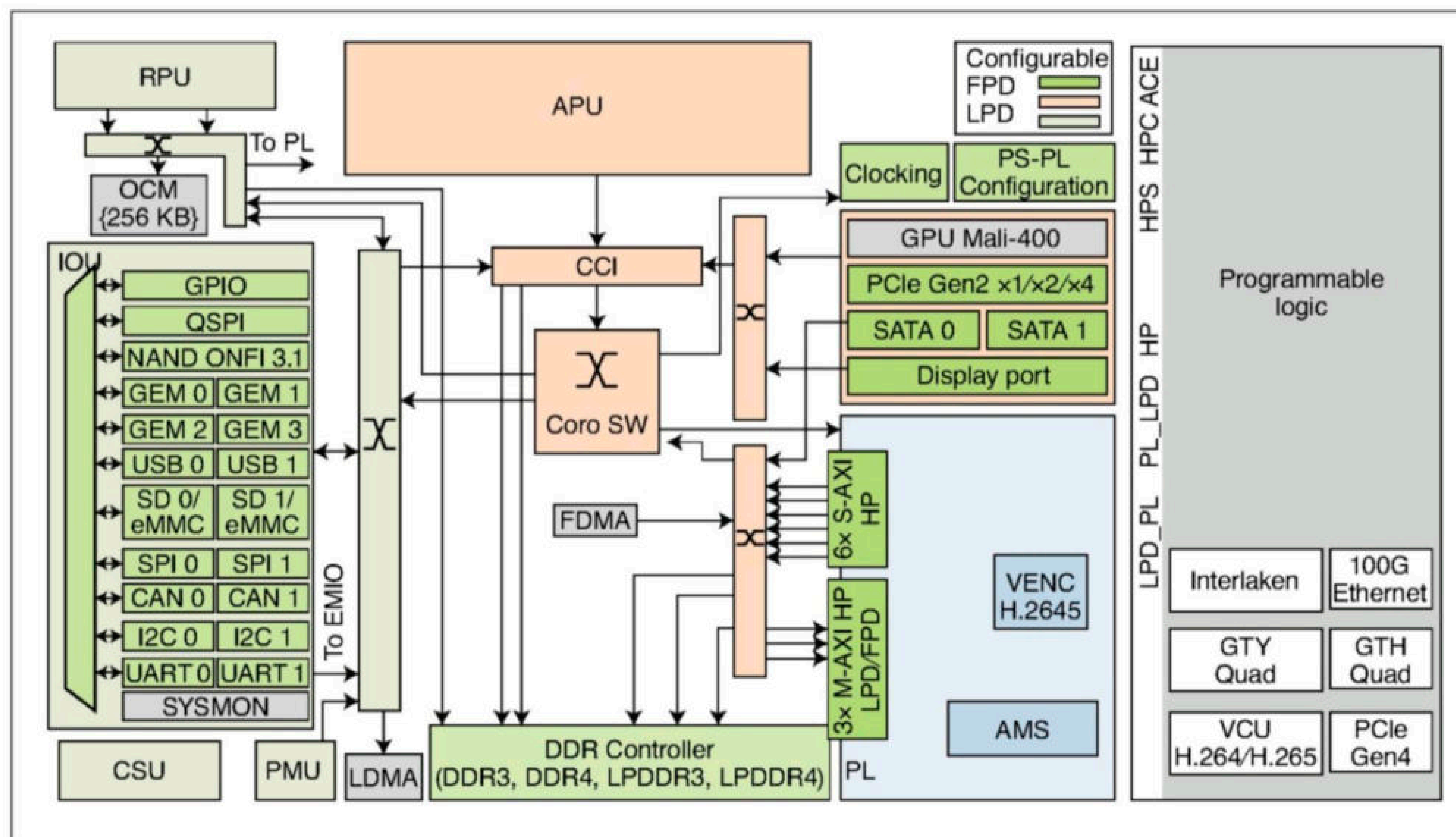


FIGURE 1 Here is an FPGA Block Diagram (Xilinx.com). Note: Interlaken is an interconnect protocol to tackle high-speed signaling between chips in network applications.

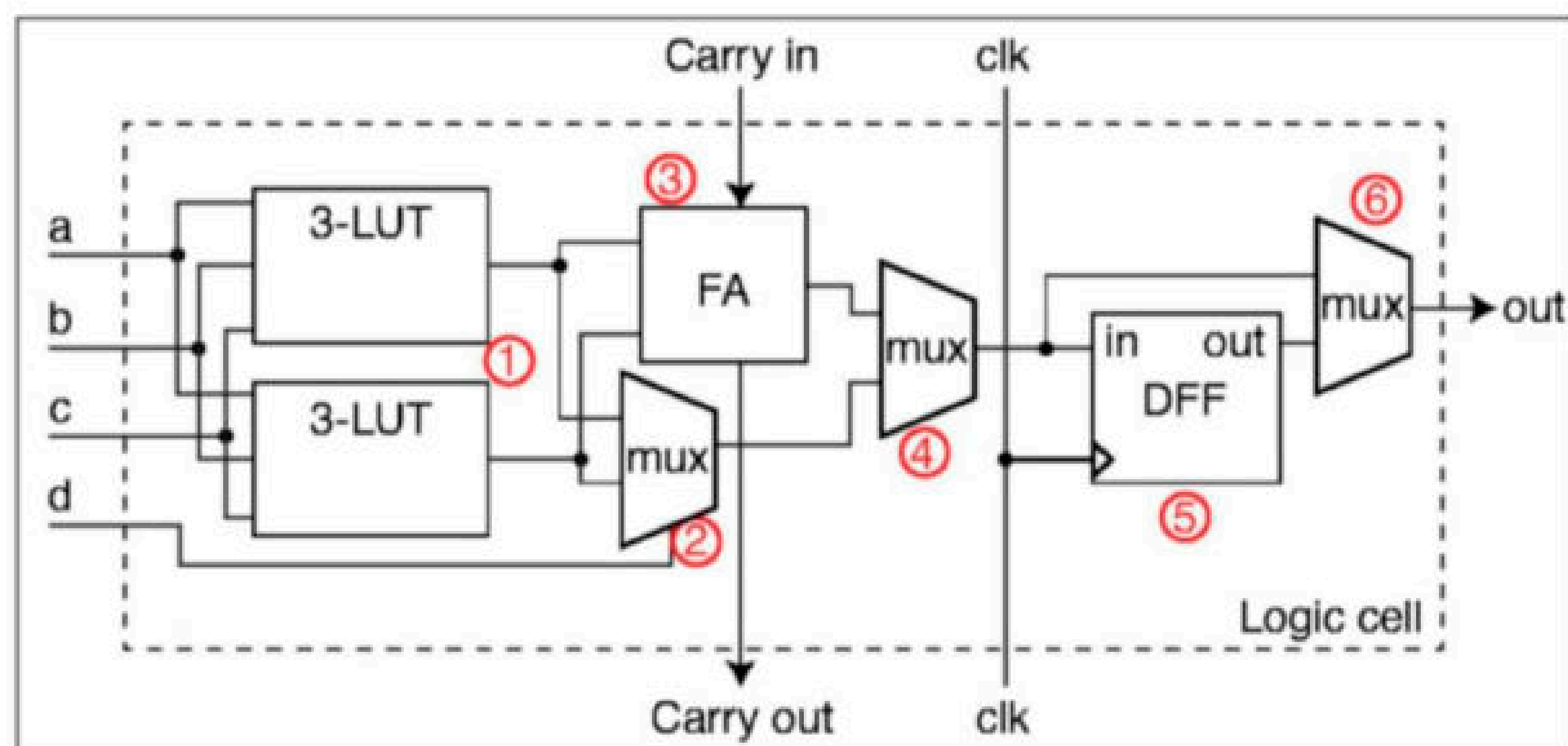


Figure 2
CLB representation diagram [1]

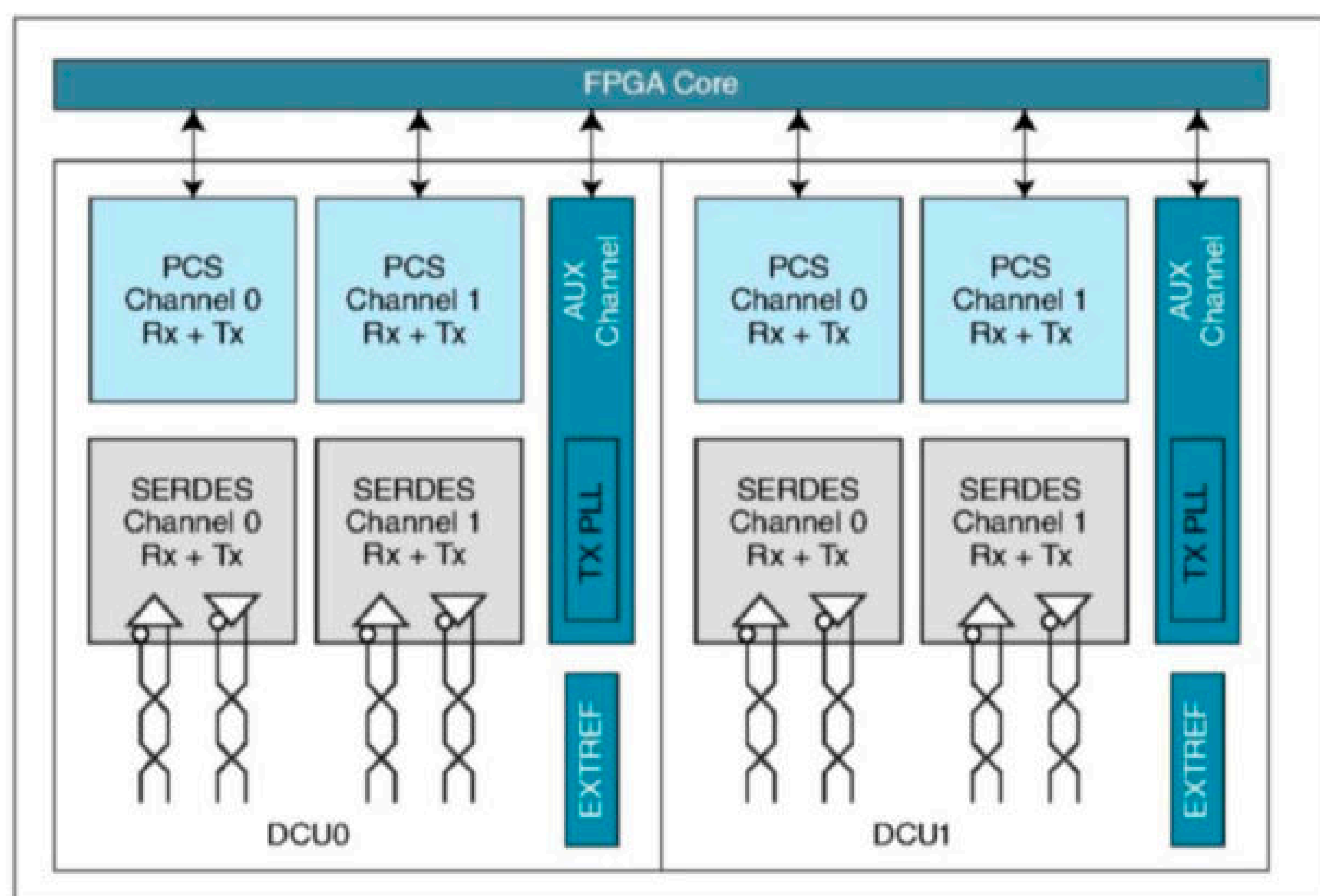


Figure 3
Representation block diagram of SERDES [2]

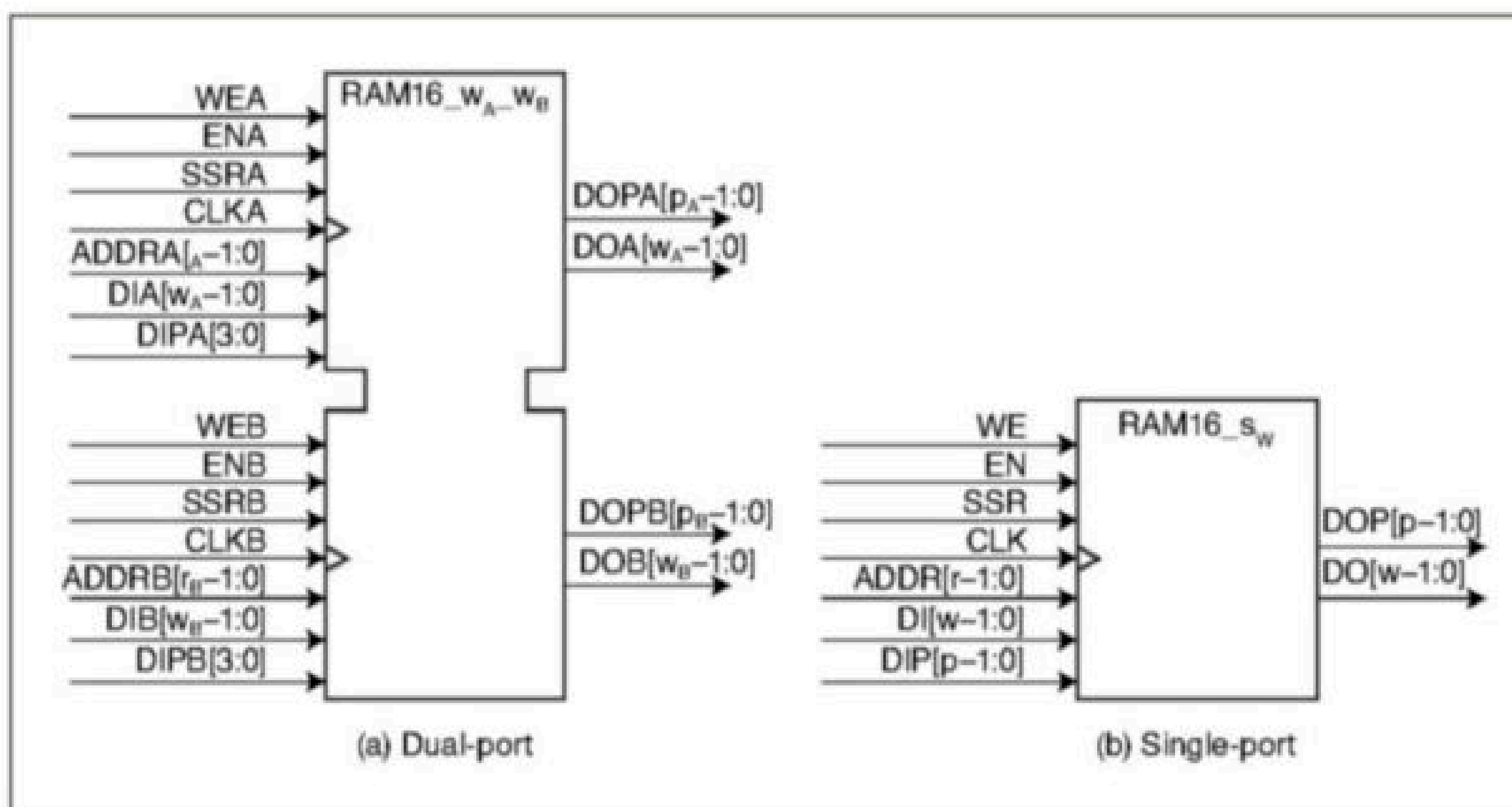


Figure 4
FPGA inferred BRAM

with the same VCCO requirement can be combined in the same bank.

Block Random Access Memory: There are various types of memory available which can be interfaced with an ASIC. But in FPGAs with limited die area, memory is more constrained. The dedicated memory on the chip itself is referred to as block RAM (BRAM). There are other types of RAM, like unified RAM (URAM), or distributed RAM (DRAM), which is part of SLICEM. While each individual block is a fixed size (36Kb for Xilinx 7 series chips), these blocks can be subdivided or cascaded to make smaller or larger BRAM blocks as needed. They can also be configured to support special functionality such as error-correction. BRAMs are a major component of FPGAs, and a high percentage of utilization can result in congestion and non-routable scenarios. So, BRAM usage must be planned efficiently to prevent this.

A typical FPGA BRAM instance is somewhat different from the hard memories found in an ASIC. When writing code for a BRAM to be inferred, the ports mapping and instantiation must be correctly implemented in the HDL code to ensure it's synthesized properly. **Figure 4** shows a typical FPGA-inferred BRAM. BRAM can be either synchronous or asynchronous. When we say that BRAM is synchronous, we mean that reads and writes are synchronous with the clock. **Listing 1** is a small snippet of Verilog code defining synchronous BRAM.

FDCE: FDCEs are the flip-flop blocks present in an FPGA. These are limited in number, and they play a major part in the overall FPGA utilization. FDCE blocks play a critical role in logic design, as well as in timing constraints and placement. An efficient FDCE placement promotes an accurate timing design.

FPGA DESIGN FLOW

FPGA design flow is typically spread across four major stages: elaboration, synthesis, routing, and device programming. In the elaboration stage, the design is compiled, checked for any syntax errors, and converted into circuitry. During elaboration, behavioral simulations can be performed to assess if the design meets the logic requirements. A test bench can be added, and the design assessed.

Once the design meets the logic requirements, it's parsed in the synthesis stage, where it's converted into a flattened netlist. This netlist file then translates, maps, and finally performs placement inside the FPGA. If the design over-utilizes the FPGA's resources in the synthesis stage, it remains unplaced and throws an error in the design tool.

RESOURCES

Xilinx.com | www.arrow.com


```

module singleportram(
  oe,
  address,
  we,
  clk,
  dout
);
parameter DATA_WIDTH = 8 ;
parameter ADDRESS_WIDTH = 8 ;

input [DATA_WIDTH-1:0] din;
input cs;
input oe;
input we;
input [ADDRESS_WIDTH-1:0] address;
input clk;
output [DATA_WIDTH-1:0] dout;

reg [DATA_WIDTH-1:0] memoryelement[ADDRESS_WIDTH-1:0];
reg [DATA_WIDTH-1:0] d_out[ADDRESS_WIDTH-1:0];
always @(posedge clk)begin
  if (cs && we && !oe) begin
    memoryelement[address]<=din;
  end
  else if(cs && !we && oe) begin
    d_out[address]<=memoryelement[address];
  end
end
assign dout =d_out[address];
endmodule

```


LISTING 1
FPGA BRAM code

In this case, one needs to clean up the design and restart all the stages. When the design is placed properly, it passes through the routing stage, in which all of the paths are routed. During the routing stage, the impact of timing constraints shows up. There may be issues like SLR crossing, SLL issues, or congested nets due to high slack. This can result in heavily negative slack or a design exhibiting level six (or higher) congestion, which will never converge. In this stage, timing must be analyzed appropriately, and necessary actions need to be taken on individual paths. **Figure 5** shows the FPGA design flow diagram.

All the necessary timing constraints are written in the XDC file, which is attached to Vivado or any other FPGA tool. Note that the FPGA has a unique Debug Hub feature which allows users to debug the signals using an interface between the FPGA's JTAG Boundary Scan and the Vivado Debug core. The user can probe signals listed during the compile. All those signals will appear in the hardware manager once the bit file and ltx files are loaded.

CONCLUSION

So, we've discussed various areas of FPGAs, from architecture to design flow. We

also covered different issues which arise while creating an FPGA design. Needless to say, this is only the tip of the iceberg when it comes to working with FPGAs. But I hope this article is a satisfactory primer on the topic. 

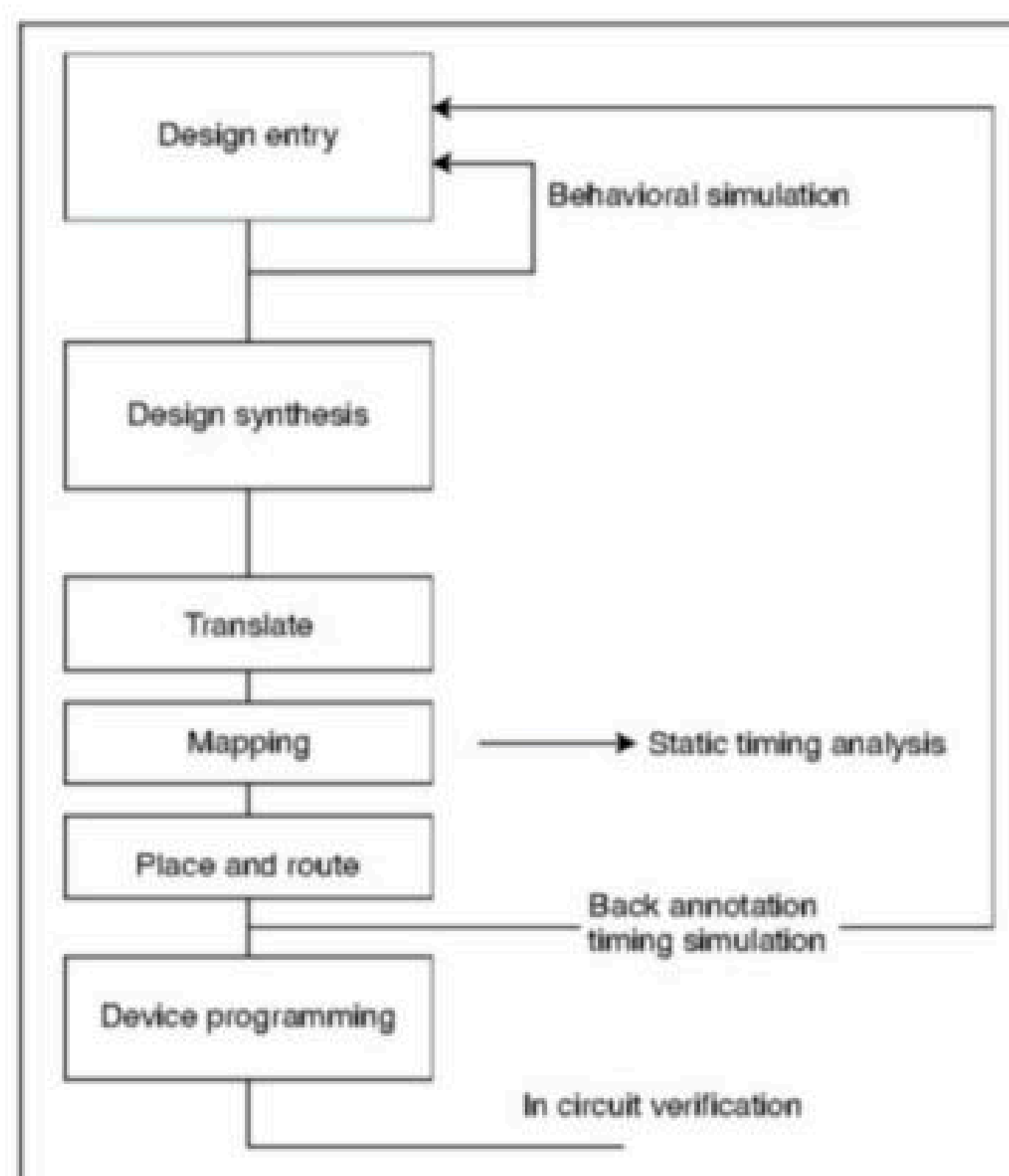


Figure 5
FPGA design flow

ABOUT THE AUTHOR
Nishant Mittal is a hardware engineer in Bangalore.

The Evolution of Flat Panel TV Technology

From CRT to OLED and Beyond

FEATURES

By
Faiz Rahman

Flatscreen TVs are ubiquitous, but even tech-savvy readers might not know of some of the engineering feats that continue to push this technology forward. Here, I cover the history of their development, and look at what's ahead.

We are not far from the centenary of one of the most important inventions in modern history—one that brought moving pictures to living rooms all across the world. On October 2, 1925, the Scottish inventor John Logie Baird, often called “The Father of Television,” succeeded in demonstrating the first proper television. It was a truly groundbreaking invention that laid the foundation for the British Broadcasting Corporation (BBC) in the years that followed. Recognizing the impact of his invention over the course of nearly a hundred years, the UK Royal Mint recently released a coin celebrating this achievement (**Figure 1**).

Baird’s original invention was an electromechanical device that was not really suitable for long-distance TV transmission. In later years, with the advent of valve-based electronics, Baird and others re-invented the TV in a form that could be beamed through radio waves to reach audiences over large geographic areas. Later still, the technology embraced solid-state electronics, and even the ability to transmit and display images in color. Thereafter, up until the 1980s TV technology remained relatively stagnant, before another wave of innovations took it by storm.

TV display technology changed dramatically from the late 1980s to the early 1990s when the so-called flat panel displays (FPDs) started to appear. Prior to that, the venerable cathode ray tube (CRT) had served that role since the emergence of TVs, themselves, in the 1930s.

The FPDs were immediately perceived as a disruptive technology, because of the several clear advantages they offered over the old CRT displays. The main benefit was easy scale-up of display dimensions. Before the advent of FPDs, increasing the size of CRTs was challenging, because of their complicated construction and method of operation.

This is easy to understand if one keeps in mind that CRTs are basically oversized vacuum tubes. Just like any vacuum tube, CRTs are made from fragile materials, painstakingly assembled together into a heavy and bulky image-display device. Flat displays—not based on the use of accelerated electron beams (cathode rays)—are inherently easier to construct, transport, and use. As it happened, the FPD revolution brought a proliferation of display technologies as the years went by. In what follows, I examine the main features of FPD technologies, focusing on both their operating principles and attendant benefits.

NEON STARTS A REVOLUTION

The first FPD technology was based on plasma display panels (PDPs), which are essentially an array of miniature neon lights. A PDP consists of a two-dimensional array of tiny chambers, each coated with a red, green, or blue phosphor, sandwiched between two thin glass sheets. These dielectric sheets are printed with a pattern of parallel conductive lines, with lines on one sheet running orthogonal to lines on the other sheet. These tracks are called “address

and display electrodes." This assembly is enclosed between a pair of glass sheets and hermetically sealed all around. During the panel sealing process it is filled with a mixture of neon and xenon gases and mercury vapor, at a low pressure.

As shown in the schematic cut-away diagram in **Figure 2**, the chambers or cells are formed by rib partitions, and each is coated with a thin red, green, or blue phosphor layer. A trio of RGB cells forms a single color pixel, where each R, G, and B sub-pixel is individually addressable. To make any sub-pixel light up, its corresponding row and column tracks are energized, such that a glow discharge takes place at their intersection.

The gas composition and pressure are chosen so that most radiation in the discharge is emitted in the UV region. The UV photons strike the phosphor, and visible red, green, or blue light is emitted. This is similar to the way fluorescent neon lamps operate. The intensity of emitted light can be varied by controlling the voltage on the display electrode, through a technique called pulse width modulation. By controlling the intensities of the red, green, and blue sub-pixels, any desired color can be displayed. The entire display is scanned at high speed to show video frames at 50 or 60 frames per second.

Plasma displays were the leading television display technology during the 1990s. PDPs can be made in very large screen sizes and have many advantages over other display technologies. Perhaps the most important is the extremely high contrast exhibited by plasma panels. Since any pixel can be completely dark when not addressed, PDPs have especially high contrast values. PDPs are also "fast," because their pixel cells can be switched between lit and un-lit states quickly. For this reason, they do not show any motion blur in high-speed action scenes. PDP screens don't rely on polarization of light to switch pixels on and off, and thus, can be viewed from any angle. While PDPs offer all these benefits, their sealed, gas-containing construction makes them heavy, fragile, and difficult to assemble. These issues caused their ultimate demise, when other displays that were easier to manufacture started to be commercialized in later years.

LIQUID CRYSTALS ENTER THE SCENE

During the late 1990s, PDP technology began to be supplanted by a different FPD technology. It was based on a light valve or switch that could be made transparent or opaque under electrical control, and relied on control of the polarization of light.

Light consists of oscillating electric (and magnetic) fields that are oriented at right



FIGURE 1
UK Royal Mint 50p coin celebrating TV pioneer John Logie Baird (Image courtesy of The Royal Mint)

angles to the direction of light propagation. These oscillating fields can be oriented at any angle around the light's propagation direction. If the oscillations take place along only a certain direction perpendicular to the light's travel direction, then the light is said to be "plane polarized." This can be easily achieved by passing ordinary unpolarized light through a polarizing material, such as a sheet of Polaroid plastic, which has a well-defined polarizing direction. Another Polaroid sheet placed close to the first sheet will either allow light to pass through it or get blocked, depending on whether the polarization directions of the two Polaroid sheets are parallel or perpendicular to each other.

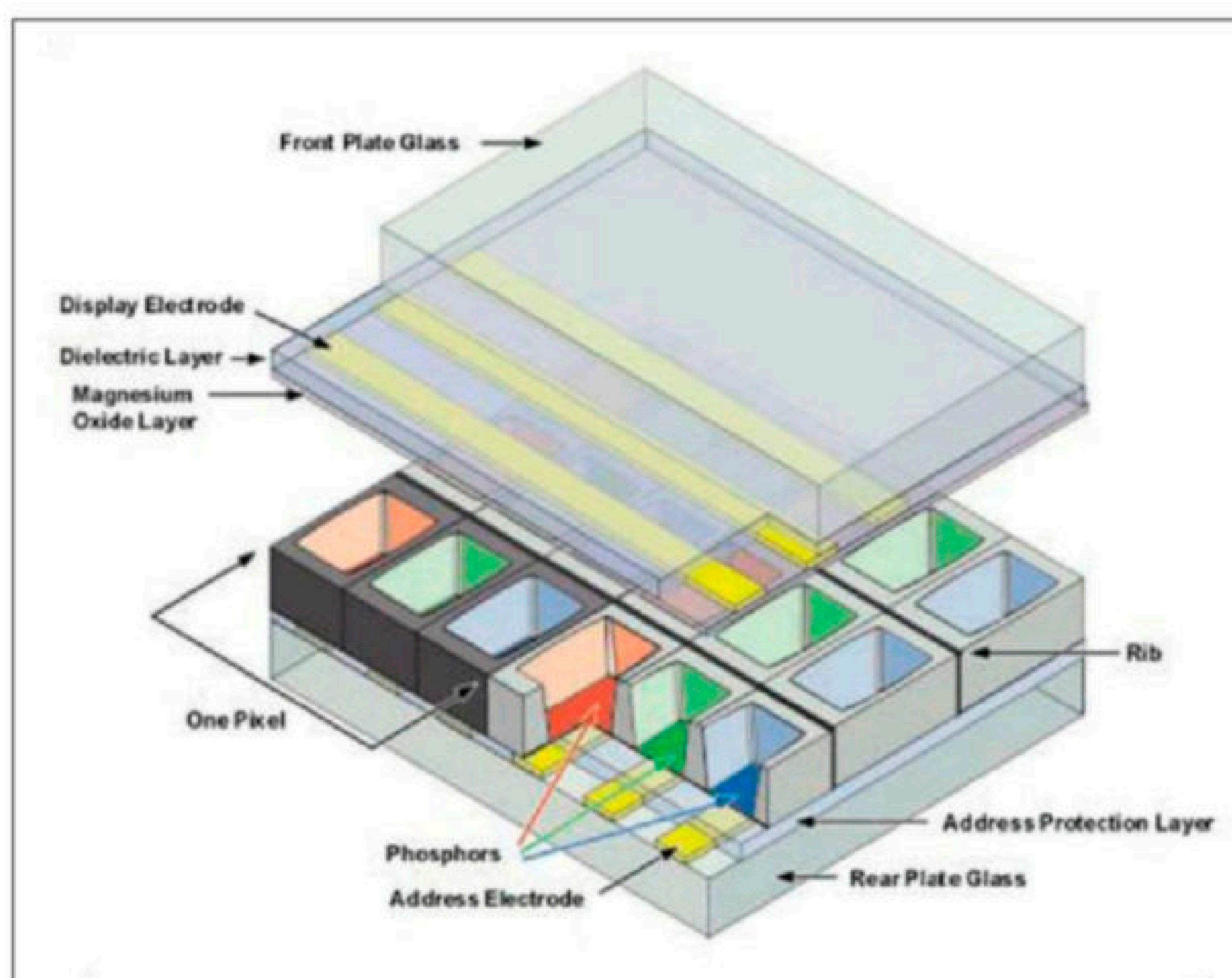


FIGURE 2
Schematic structure of a television plasma-display panel

This basic principle can be used to construct a display device, by using a Polaroid sheet together with an electrically-controllable, polarization-inducing device. Such a device is made by sandwiching a material called a “liquid crystal” between two sheets of electrically-conducting glass plates. Liquid crystals are materials with long, rod-like molecules that can twist under the action of an applied electric field, and rotate the plane of polarization of light passing through them.

A liquid-crystal display (LCD) panel consists of a two-dimensional grid of liquid-crystal cells that can be controlled individually. Applying a voltage to any cell causes the plane of polarization of light passing through that cell to align with the polarization direction of the Polaroid sheet. This allows the arrangement of the liquid-crystal cell and the Polaroid sheet to become locally transparent to the passage of light. A flat, uniformly-lit white screen placed at the back then becomes visible at that point, displaying a picture element (pixel). If the voltage is removed from a chosen liquid-

crystal cell, then light is not able to pass through, and that pixel becomes dark. By selectively turning pixels on and off, both still and moving images can be displayed. Color pictures can be displayed by using red, green, and blue plastic filters, uniformly distributed on the LCD panel so that the pixels appear colored (**Figure 3**).

LCD technology has now matured to the extent that LCD-based TVs have become commodity products—manufactured in large numbers and sold at affordable prices. Early LCD TVs exhibited problems with LCD cell-switching speeds, which caused “motion streaks” to appear in scenes where moving bright lights were shown. This issue was overcome by the use of better liquid-crystal materials and optimized liquid-crystal cell design. These advances also helped to increase the viewing angle of the displays, so that people not seated directly in front of the screen can also see a normal, undistorted image.

The one problem that has taken longer to solve is the limited image contrast delivered by LCDs. This arises because “off” LCD pixels are not completely opaque, and some light passes through them. Thus, black pixels are not entirely black, but are rather a deep shade of gray. Better materials and device design and optimization have greatly improved image contrast in modern LCD panels, but there is still room for improvement.

Traditional LCD technology has spawned a number of more advanced offshoots that feature better image reproduction. While these are still based on the basic LCD panel, the innovations lie in the construction of the backlight, which differentiates various derivative technologies. This fact has been somewhat cleverly exploited by TV manufacturers by advertising them as distinct TV technologies—LED, QLED, Nanocell, and others. Nevertheless, it is a fact that these are definite improvements over the traditional simple LCD technology. So, let’s explore these more modern variants of the LCD technology.

TVs WITH LED BACKLIGHTING

The original LCD TVs made use of a backlight that was basically just a sheet of diffuse white plastic, lit by slim, cold cathode fluorescent light (CCFL) tubes from the sides. CCFL edge-lit backlights were mass produced in several sizes for the TV industry. With the arrival of high-brightness, white LEDs in the late 1990s and early 2000s, fragile CCFL tubes were replaced by lighter, cheaper, and more controllable white LEDs. Thus began the era of LED backlights—with LEDs first positioned around the edges, and later directly behind the light diffuser screen, as shown in **Figure 4**.

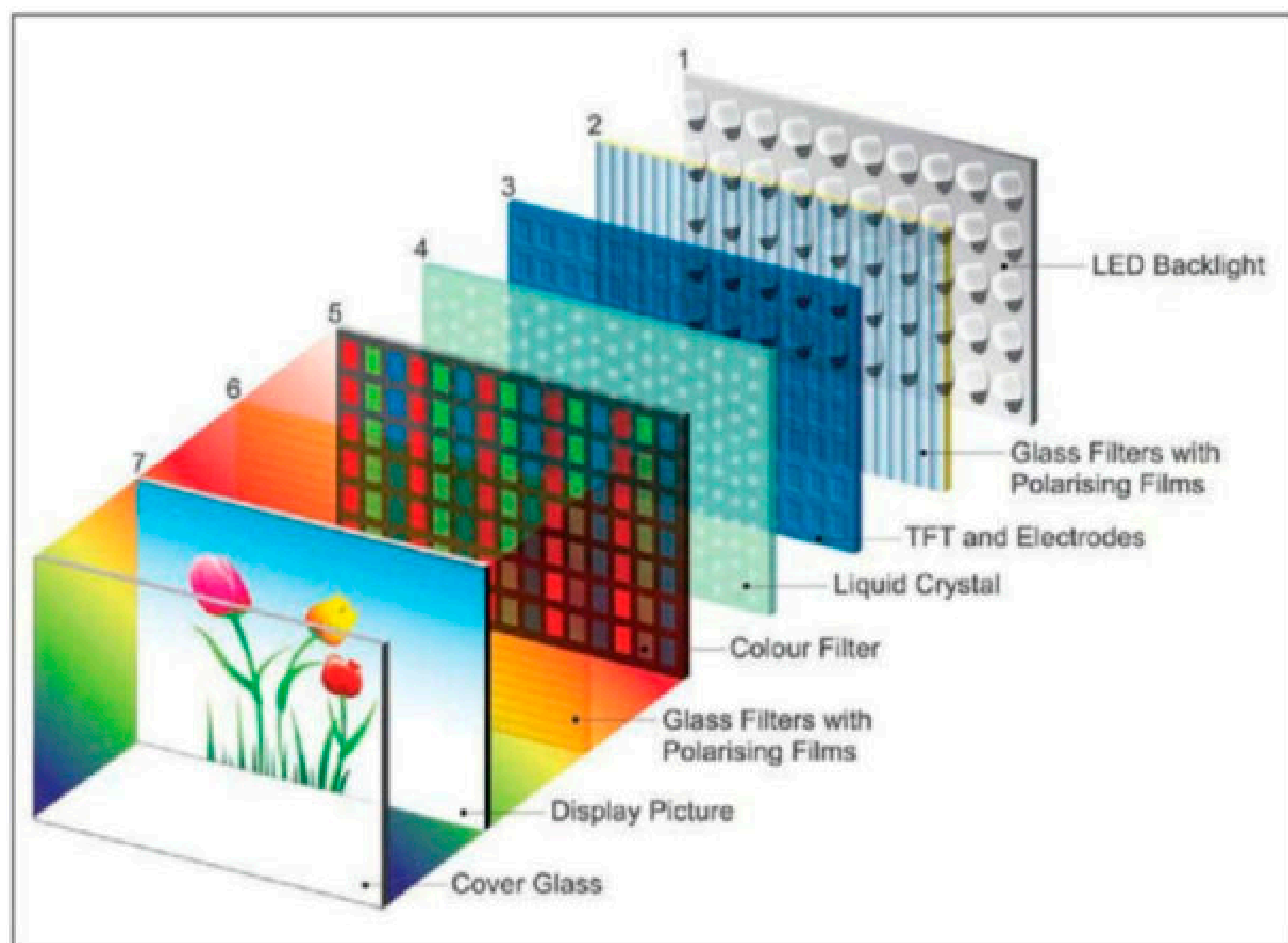
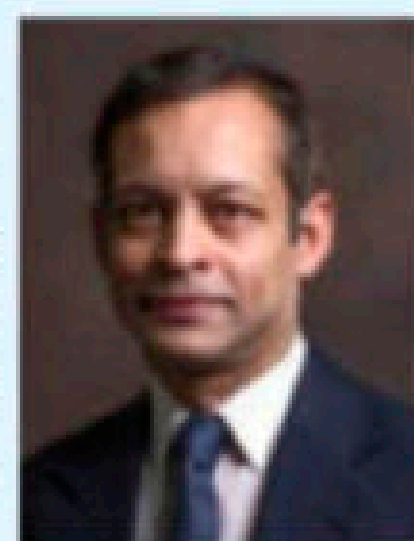


FIGURE 3

Schematic diagram showing the basic construction of an LCD panel (Image courtesy of Toyo Ink Group, Ltd.)

ABOUT THE AUTHOR

Faiz Rahman, PhD, is Associate Professor of Electrical Engineering at Ohio University, Russ College of Engineering and Technology. Dr. Rahman’s research interests are in semiconductor materials and devices for electronics and photonics, in particular in the development of novel radiation emitters and detectors. He also conducts research on various semiconductor manufacturing processes. Email: rahmanf@ohio.edu



In this role, LEDs offer several advantages. Low power consumption is at the top of the list. LEDs consume only a fraction of the power dissipated by CCFLs. LEDs also provide better color balance, which makes colors more realistic. But perhaps the biggest advantage offered by LED-backlit TVs is the ability to provide local dimming. These devices can be individually controlled, and their brightness can be made to mimic the dark and bright locations of the image being displayed. LEDs behind dark parts of an image can be dimmed appropriately. This greatly enhances the perceived image contrast. Today, most LCD TVs make use of this feature. Because LED brightness can be modulated at high speeds, this works well with modern image displays with high frame rates.

During the past four years, a new generation of TVs has started using dense, two-dimensional arrays of small LEDs to illuminate their backlights. These “mini-LED” TVs offer the ultimate in local contrast control that is possible with LCD technology. This feature, together with their competitive pricing, has made it one of the best-selling TV technologies of recent times.

QUANTUM DOTS AND NANO PARTICLES STEAL THE SHOW

The RGB color combination scheme lies behind all color display systems. This is how it works. Narrow wavelength spread (high color purity) in the red, green, and blue light components enables the creation of a wider palette of displayed colors, or a “wide color gamut.” Traditional white LED TV backlight technology does not allow narrow wavelengths to be selected by simple filtering of white light. However, TV manufacturers have found a way to achieve primary color sources with narrow wavelength spread by the use of quantum dots.

Quantum dots are very pure, nanometer-sized crystals of substances such as cadmium sulfide, cadmium selenide, indium phosphide, and other binary compounds, synthesized through techniques that produce crystals of precise dimensions. By irradiating quantum dots with blue or ultraviolet light, one can obtain other visible colors that depend on the size of the crystal. The smaller the dimension of the quantum dot crystal, the longer its emitted wavelength. Thus, with increasing size, quantum dot emission shifts toward the red end of the spectrum. The same material, grown in different sizes, can produce a range of emission colors, as shown in **Figure 5**.

Quantum dots have been used in quantum-dot LED (QLED) displays that benefit from the narrow wavelength emission from quantum dots. These displays use a plastic

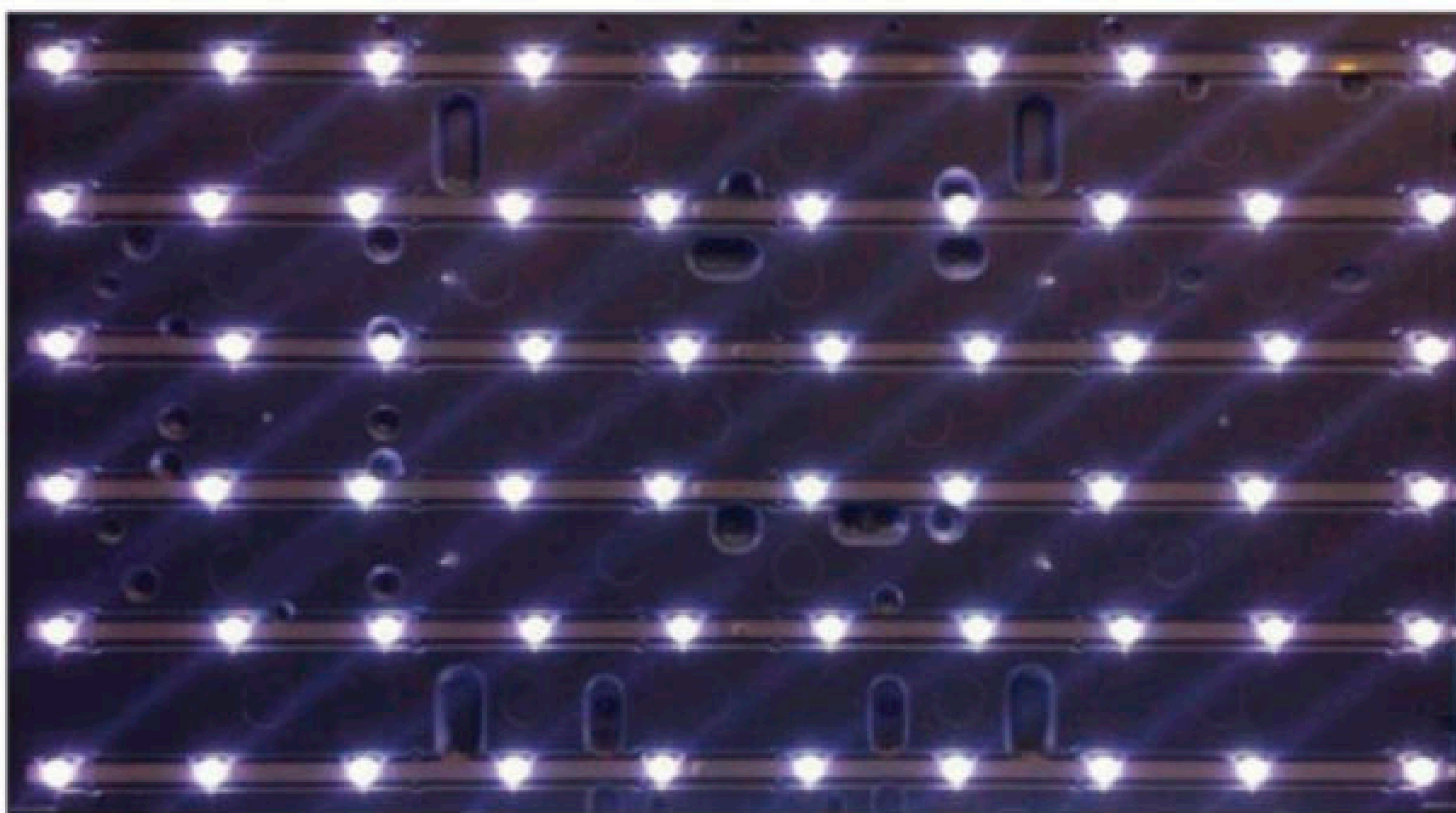


FIGURE 4
White LEDs positioned in a 2D array behind the backlight diffuser of an LCD TV

film containing red- and green-emitting quantum dots. The film is placed in front of a backlight illuminated by blue LEDs. Quantum dots absorb blue light and convert it into red and green light. Together with the residual blue light from the backlight, the combination produces white light with narrow red, green, and blue spectral peaks. The QLED light source thus produces the ideal spectral emission for color image reproduction.

QLED TVs, popularized by Samsung, are renowned for their excellent color rendering. They produce shades that are close to natural, for the rendition of challenging colors, such as skin tones. A comparable technology is that of “Nanocell” displays, developed by LG Electronics. Here, too, a polymer sheet with dispersed nanoparticles is used, but in this case the particles are used for filtering light from a white backlight. This produces a spiky RGB spectrum that can be used for more accurate color rendering than is possible from a simple LED-lit white backlight. Both QLED and Nanocell TVs produce exceptional color reproduction, and have gradually become highly successful products.



FIGURE 5
Suspensions of quantum dots of different sizes glowing under ultraviolet light



FIGURE 6
OLED TVs produce the deepest blacks of any current TV technology

TV BECOMES ORGANIC

All FPDs based on the LCD technology utilize a backlight as the source of display illumination. It is light from the backlight that is visible at the location of lit pixels. The LCD panel simply acts as a matrix of light valves that make the backlight locally visible, or not.

In contrast, a self-emissive display has a point source of light at each pixel location that can be selectively turned on or off to display brightness or darkness at that point. Widely used LED matrix boards are examples of self-emissive displays, where each pixel location is an LED, or a trio of RGB LEDs. Implementing this scheme in TVs requires much tougher engineering, because of the need to produce miniaturized, light-emitting elements at individual pixel locations. This approach has been successfully developed through the technology of organic, light-emitting diodes (OLEDs), to make a new breed of self-emissive TVs.

OLED TVs make use of special panels manufactured by LG Electronics. The panels are constructed from electrically conducting sheets of organic, light-emitting semiconductors patterned into arrays of organic LEDs—each paired with a driving transistor that can switch the corresponding LED on or off. Using a so-called “thin film transistor” (TFT) to control each OLED is called an “active matrix approach.” This strategy allows fast scanning of the entire OLED panel, leading to high image-refresh

rates. The OLEDs can be made from different materials that can emit red, green, or blue light. The most outstanding advantage of this design is that, when not lit, pixels can be completely dark, and thus can reproduce perfect black levels in scenes being displayed. OLED TVs are, therefore, renowned for their high dynamic contrast levels—something that even the best LCD-based TVs cannot match (**Figure 6**).

The near-perfect black levels, combined with rich color reproduction, wide viewing angles, high frame-rate capability and the possibility of making thin and lightweight curved screens, all have contributed to the widespread popularity of OLED displays.


Quantum dots are also being integrated with OLED displays, to further enhance color reproduction accuracy. Q-OLED displays could be made from a single blue-color, active-matrix OLED array, with red-, green-, and blue-emitting quantum dots printed on adjacent pixels to provide full-color capability.

Compared to LCDs, OLED TVs command a price premium, due to the involved nature of their production and their somewhat low manufacturing yield. Nevertheless, these displays have grown in popularity, and have become available in ever-larger screen sizes—currently approaching 100 inches!

THE ROAD AHEAD

The television industry thrives on introducing new display technologies through continuous technological innovations. Commercially successful technologies are improved and re-marketed, while new technologies are developed and promoted on a regular basis. This is now a well-established trend, and has resulted in ongoing improvements to display picture quality and related performance metrics.

Micro-LED, self-emissive displays based on tiles of gallium nitride LEDs are the most advanced alternative technology at this time. Beyond it, several new technologies are being investigated in corporate and academic R&D labs. These include the use of laser diodes for providing blue illumination for QLED TV backlights. Replacing blue LEDs with blue laser diodes can improve the spectral characteristics of light by narrowing down the blue component, too. New luminescent materials for generating colored light, such as charge transfer compounds, are also being investigated for electronically altering colors produced at each pixel location, instead of using filters to produce component colors.

There is no doubt that we will continue to see more exciting TV display technologies in years to come—some with capabilities that are hard to imagine today. 

RESOURCES

LG Electronics | www.lg.com

Samsung | www.samsung.com

Polaroid | www.polaroid.com

Level Switch for Non-Conductive Liquids

In this article, I discuss how our company repurposed a liquid level probe intended for conductive liquids to work with non-conductive liquids, as well.

By
Alex Pozhitkov, PhD

Not too long ago, our company developed and built a gas conditioner for fuel cell research. Before supplying hydrogen and air to the fuel cell, the gases had to be humidified and warmed up to a controlled level. Humidification was performed using a membrane-based, flow-through system, which required distilled or deionized water. As a result, the level of distilled water had to be monitored throughout the operation of the gas conditioner, and the water had to be replenished automatically to compensate for the water carried away with the gas. The task seemed to be quite simple, and there was no shortage of various level sensors and switches available at Grainger or McMaster-Carr. Unfortunately, they did not fit one way or another—either too big, or too expensive, or outside the temperature range. Therefore, we came up with our own solution.

It is important to understand that distilled water is not conductive, but rather “polar.” This means that a water molecule is a dipole having partial positive and negative charges. Other molecules, including ethanol, ammonia, and acetic acid (vinegar) are also polar. **Figure 1** shows the distribution of charges across the water and ammonia molecules. The dipole characteristic of these molecules is going to be very useful for us.

When a polar liquid is placed into an electric field, the molecule dipoles orient themselves along the field lines, effectively reducing the electric field within the liquid. This phenomenon is described through a dielectric constant, ϵ . For water at 20°C,

$\epsilon=80$; for liquid ammonia at -34°C, $\epsilon=22$. If such liquid is placed into a capacitor, its capacitance would increase ϵ times, compared to an air-filled capacitor. Our level switch utilizes this phenomenon according to the diagram shown in **Figure 2**, which is effectively a voltage divider.

The voltage on the resistor part of the divider is applied to the sensor, whose capacitance depends on the liquid level. As the liquid reaches the sensor's rod, the capacitance increases, and the voltage drops. A good candidate for an airtight sensor is the Dwyer CLP-1 liquid level probe (**Figure 3**). Contrary to our requirements, the probe is specifically designed to work with conductive liquids! But as we will see, our solution makes it perfectly suitable for both conductive and non-conductive liquids. (Although Dwyer is not producing such sensors anymore, there is a wide variety of similar products at Grainger under the category “Conductivity Level Probes.” For example, item #6EJP7, Liquid Level Probe: Industrial, 1/2 in NPT, 304 Stainless Steel.)

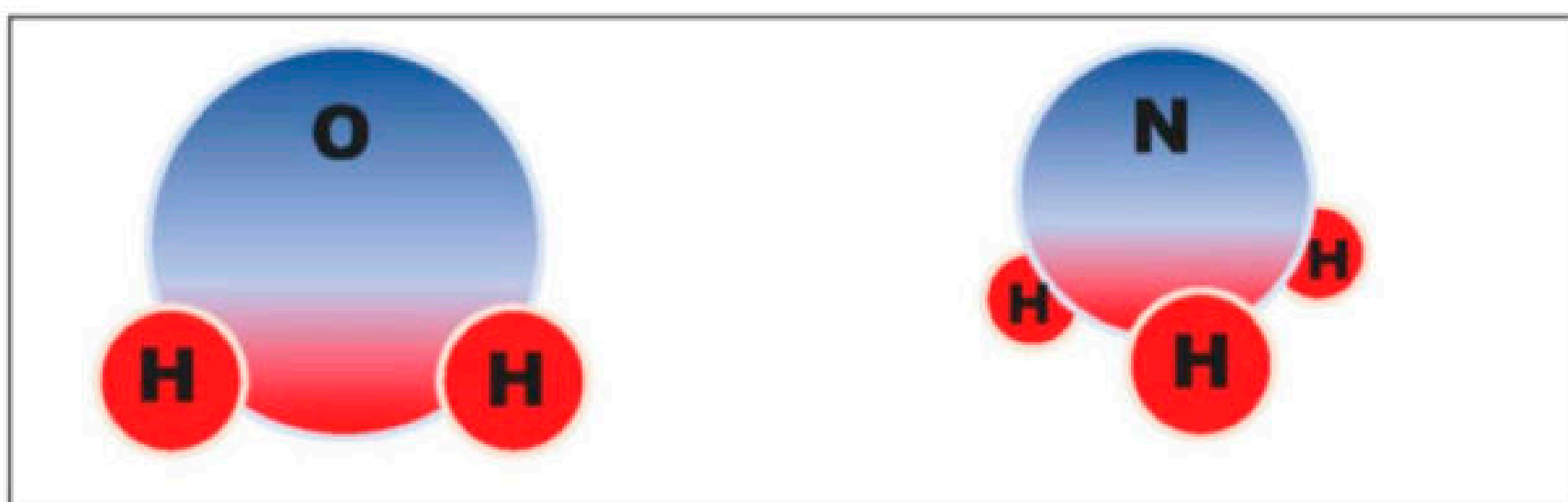
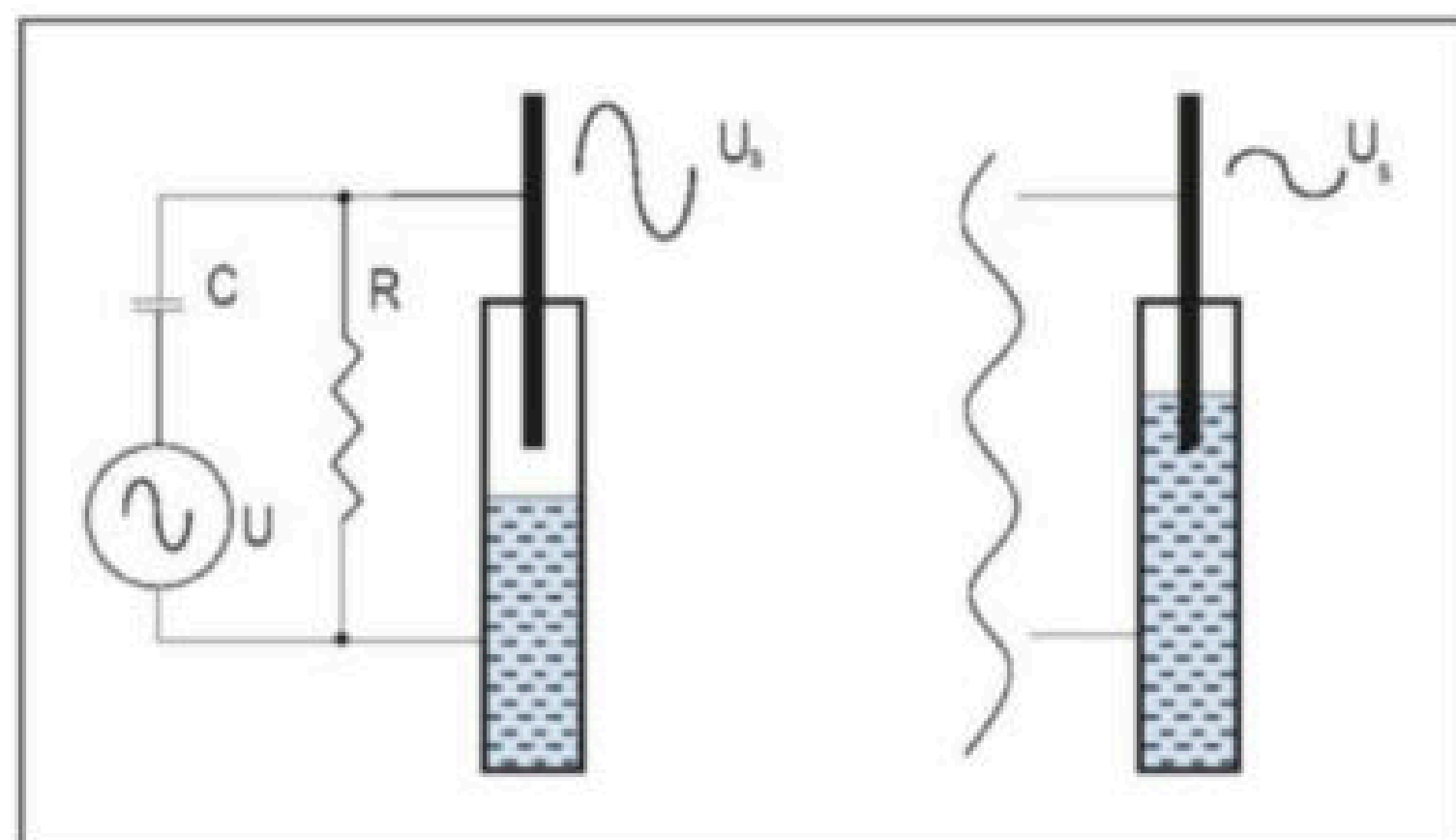


FIGURE 1

This is a schematic representation of electron density distribution in water (left) and ammonia (right) molecules. Blue to red indicates the transition from negative to positive partial charges.

FIGURE 2

Outer shell and a sensor rod make a liquid-filled capacitor, which is the liquid level sensor. The capacitance increases as the rod is being immersed.



Condition	U_s , Vp-p	C_s , pF
Air	8.80	14.0
Distilled water, 1-2 mm immersion	0.72	1288.0
City water, 1-2 mm immersion	0.13	7592.0

TABLE 1

The results of our experiments with a sine wave generator to determine the capacitance of our system with and without distilled water or city water

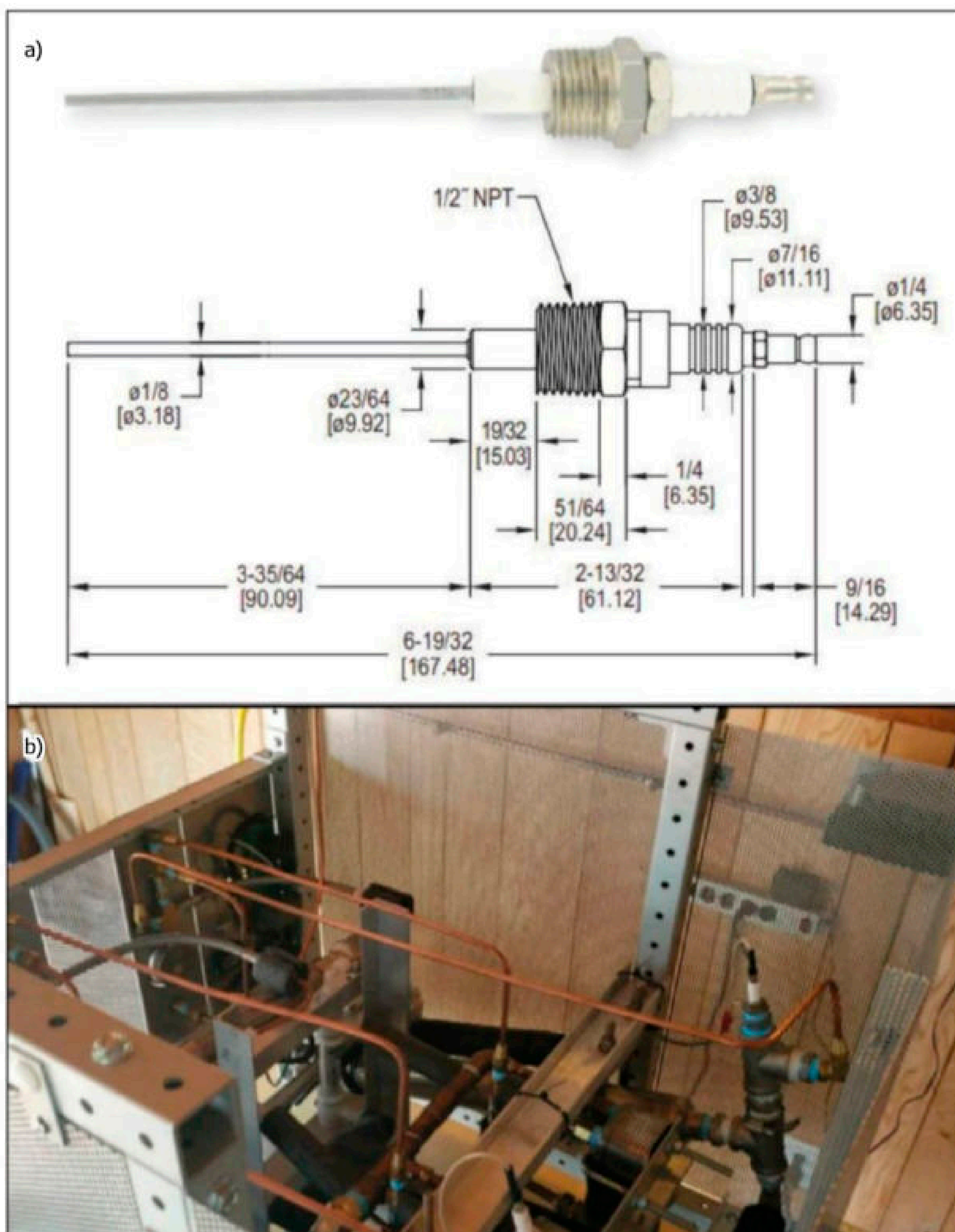


FIGURE 3

a) Dwyer Instruments CLP-1 liquid level probe (Image adapted from dwyer-inst.com); b) The probe placed into the gas conditioner

We did not know the capacitance of our system with and without the distilled water. Therefore, we conducted initial experiments with a sine-wave generator, 500Hz, $U=10V_{p-p}$, $C=100pF$ and $R \rightarrow \infty$ (removed). Capacitor C and the liquid level sensor with capacitance C_f make a divider:

$$U_f = \frac{UC}{C+C_f}; \text{ therefore } C_f = \frac{C(U-U_f)}{U_f}$$

Measurements produced the results shown in **Table 1**. These findings are quite revealing. First, our physical model seems correct. Indeed, the distilled-water-filled capacitor is about 80 times higher than the air-filled capacitor! Second, as a side note, the capacitance in the case of distilled water is quite high—1.3nF—which explains why any digital electronics should not be turned on wet, even if the wetness is due to non-conductive distilled water or dew. Such capacitance at the megaHertz-scale frequencies presents itself as low impedance of a few hundred ohms.

Knowing the order of magnitude of the capacitance in our configuration, the final design for a two-channel liquid level switch was created and tested. The schematic for it is shown in **Figure 4**. A sine-wave oscillator is built around U1.1, which produces approximately 3kHz. The oscillator is followed by the buffer U1.2. The signal is split into two identical channels built around U2.1 and U2.2. (In our design we had to humidify two different gases. In principle, there may be one, two, or more channels).

Let's focus on one of the channels in Figure 4 outlined with a red-dashed rectangle. The 100pF capacitor, as explained above, is a part of the voltage divider. The 390kΩ resistor shunts statics and electromagnetic noise that may occur in the sensor and wires leading to it. The outer shell of the sensor is tied with ground (GND_COMMON). Measured amplitudes on the sensor with respect to ground were 0.18V and 2.88V, with and without distilled water, respectively.

The U2.2 digitizes the sine-wave signal such that when water is present, the output of the comparator becomes high impedance. In this case, this is because the voltage on the inverting input is below that of the non-inverting input. Diode D5 acts as a very high-impedance device, preventing any significant current flow into the base of transistor Q3. Hence, the transistor Q3 is not conducting, resulting in low voltage at the gate of the FET Q4. Thus, the Q4 is blocking the current through the solenoid. When water falls below the sensing rod, the output of the comparator produces negative voltage pulsing. These pulses are integrated with the 470kΩ resistor R9 and the 22nF capacitor C7. The integrated voltage affects (lowers) the voltage

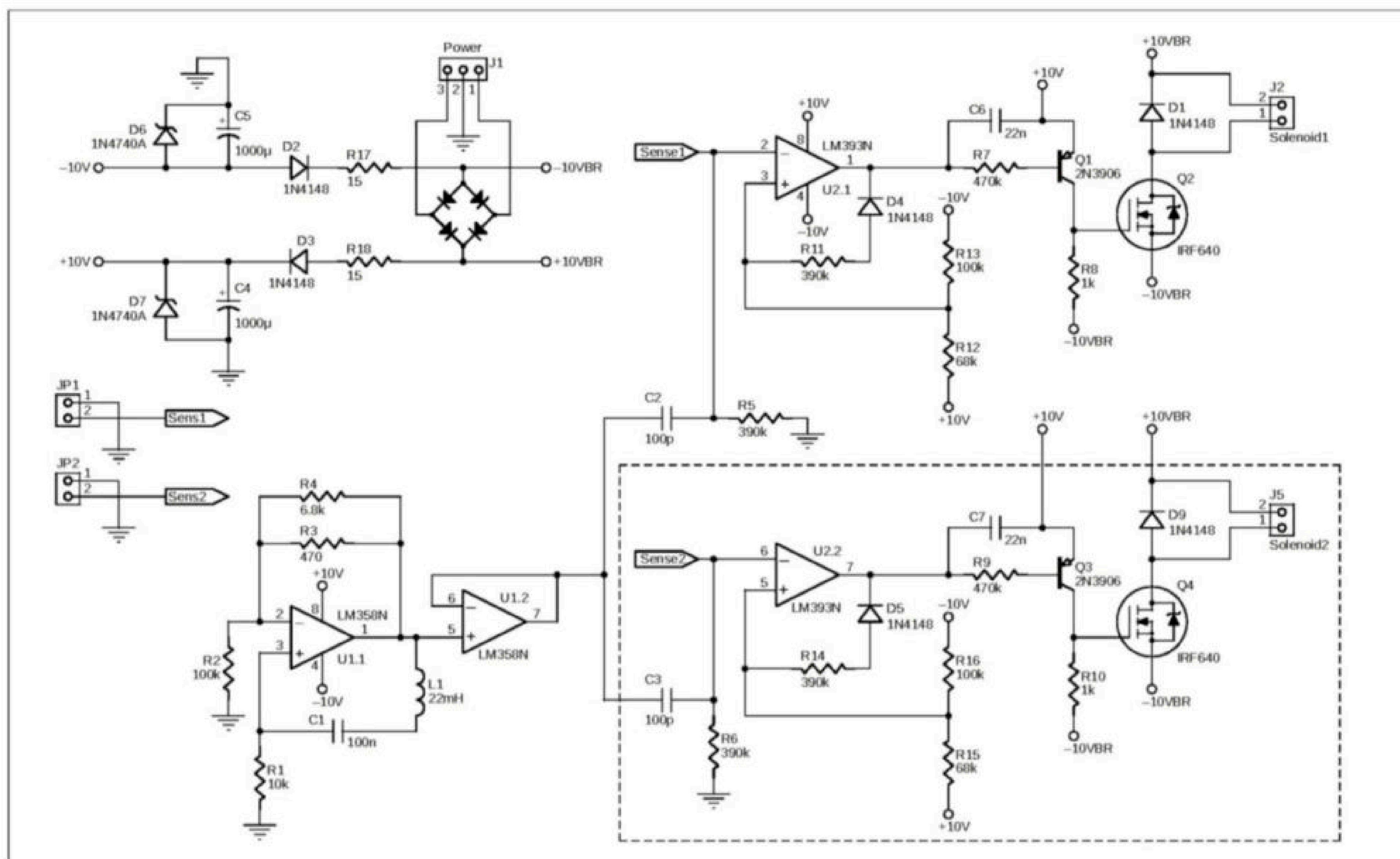


FIGURE 4
Schematic of a two-channel liquid level switch

on the non-inverting input of the comparator through R14, D5, thus providing a hysteresis. Hence, the transistor Q3 turns on, resulting in increased voltage at the gate of the FET Q4. Thus, the Q4 opens and the current is flowing through the solenoid. The current through the Q4 energizes the solenoid, and water begins filling up the system. Note that the solenoid is fed with unregulated pulsing voltage “VBR” (20V). The power source for the level switch was a 14VAC transformer with a center tap.


The liquid level switch has been shown to work well with distilled (and city) water. But how will it perform with other liquids, and what is the appropriate frequency range? Let us theoretically analyze the divider in Figure 2 and make some predictions. Simple manipulations of complex math reveal the voltage on the sensor as shown in the equation:

$$U_s = \frac{UC(C+C_s)R^2w^2}{1+(C+C_s)^2R^2w^2}$$

where w is angular frequency ($w=2\pi f$), and

C_s is the capacitance of the sensor, which depends on the presence or absence of liquid and its dielectric constant. For a system being designed, one has to experimentally determine C_s in air and in liquid. After that, it is possible to come up with optimal C and w ,

such that the voltage on the sensor is within reasonable limits, and the difference between U_s^{air} and U_s^{liquid} is greater than noise. R should be at hundreds of kilohms.

Detecting the level of conductive and non-conductive (but polar) liquids was accomplished by utilizing the difference in the dielectric constants of air and the liquid. There is a wide variety of liquid level probes on the market, which may be labeled “for conductive liquids.” But fear not—our solution will repurpose them for the non-conductive liquids as well! 

ABOUT THE AUTHOR

Dr. Alexander Pozhitkov, PhD, has an MSc degree in Chemistry and a PhD in Genetics from Albertus Magnus University in Cologne, Germany. His expertise is in interdisciplinary research involving molecular biology, physical chemistry, and electrical engineering. Alexander has worked in academia (University of Washington and Planck Institute) and in the private sector (MidNite Solar). He owns a small family business, Buddy Engineer, which specializes in the design and manufacturing of research equipment. He is a researcher at the City of Hope Medical Center in Los Angeles CA.

RESOURCES

Buddy Engineer | www.buddyengineer.com

Dwyer Instruments | www.dwyer-inst.com

Grainger | www.grainger.com

Smart Buildings and Cities

Proactive Planning for Your Business

By
Michael Lynes

Does your business need to move to the smart city of tomorrow? Learn how to avoid potholes and traffic jams, and navigate confidently into your smart building today.

WHO LET THE ROBO-DOG OUT?

Sure, you're thinking, smart buildings...that's just what humanity needs. We can't figure out how to postpone Windows updates, so let's add cybernetic warehouse workers with a penchant for somersaults, and fido-droids with killer frisbee skills into the mix. In fact, what we really need is a whole new dystopian cyborg breeding ground, ala *I Robot* by Isaac Azimov (one of my favorite sci-fi stories, though it was ruined in the movie adaptation, in my humble opinion, by Will Smith and that lady that played Doctor Calvin, but I digress).

The concept of a smart city, comprised of many interconnected smart buildings, isn't new, but has evolved significantly over the last couple of decades. And, whether it was brought into existence by intention, or from one-too-many wireless sprinkler systems accidentally short-circuiting with discarded fit-bits during a thunderstorm, it seems like it's here to stay. All the same, if the thought of a billion-ton steel and glass behemoth looming over you as you stroll down the street listening to your favorite techno, and using the Bluetooth signals bouncing off of your head to calculate if its window washer-bot could dump a bucket of grimy suds on you as you pass doesn't send a chill down your spine, then maybe this is the wrong article for you. (Just kidding. You can see I've been thinking about this a lot.)

Anyway, nightmare scenarios aside, the proliferation of smart buildings—and their big brothers and sisters, smart cities—is an undeniable trend in our modern, ever-more-coupled society. Over the past several years the need for diversified, highly connected, and efficient workspaces has grown exponentially. Recent events like the global pandemic cannot be entirely blamed for this, but it certainly threw saltwater onto the already raging smart building lithium fire—which brings us to the knob of this article.

So, there you are one fine Monday morning in November, basking in the cool glow of your twin twenty inch LED monitors,

scrolling through Twitter as you think about changing this week's line-up for your fantasy football team, and wondering if Doordash is up for delivering lunchtime nachos, when your boss casually pauses in the doorway of your cube-world happy place and says, "Hey, you're an IT guy—what do you think it would take to make our building 'smart'? The big boss asked me to figure it out, so now it's your problem. Do some research and get back to me with an executive summary, and a biz case by say, ah, Friday. That would be gr-r-reat. . ."

And then she takes a sip of her triple-mocha-half-skim-half-soy latte, and wanders off in the direction of the break room as the panic sets in. Fortunately for you, you're reading this comprehensive guide on just that subject. So, sit down, my Padawan, and let me edu-ma-cate you.

THE IOT—TWENTY YEARS ON

Twenty years? Can it really be that long? Yes, believe it or not, the term IoT (or Internet of Things), is more than twenty years old. According to Wikipedia [1], the term Internet of Things first appeared in a speech by Peter T. Lewis in September 1985. Lewis defined it as "[the] integration of people, processes and technology with connectable devices and sensors to enable remote monitoring, status, manipulation and evaluation of trends of such devices." The term was even re-coined independently by Kevin Ashton of Procter & Gamble in 1999, though he prefers the phrase Internet for Things, because prepositions, or something.

Even further back, before it was called the Internet (and way before the Things), legend has it that the primary neural node of this IoT-powered proto-Matrix was born in the late nineteen-seventies within a lowly soft-drink vending machine located on the campus of Carnegie Mellon University.

HAL, as he prefers to be called, became the first ARPANET connected appliance, able to report the weight and the pimple density of his college-age patrons, as well as to monitor the

contents and temperature status of his soft-drink inventory. And, as HAL would remind you, the value of any network increases as a function of the number of nodes that are connected to it. Fast forward from that lowly beginning to today and IoT class devices have become legion. They are the worker bees of the Internet, whether wired or wireless, and include everything from key fobs to kitchen sinks. With the advent of IPv6, the number of things that can be linked together using IoT is almost without limit. All modern building systems—HVAC, water, sewer, solar power, telephones, lighting, emergency notification and recovery, maintenance, electrical, IT, fire, safety, access management, and maintenance—can be controlled and monitored by IoT. These devices are the fundamental components for our smart building, and their utility has grown in proportion to their ubiquity.

The push to add intelligence to the corporate environment is no longer a question of why, but rather how, when, and at what cost versus benefit? And, since you've been tasked with planning out your company's migration into this smarter world, the first question becomes: build new, or retrofit? There are business cases to be made for both scenarios, and a savvy Industrial Systems Engineer (ISE) can be a good source of advice on the finer details. There are consulting firms whose sole focus is smart building roll-out planning, and even a modicum of research will turn up a slew of choices. Rather than dive into the gritty depths, however, the rest of this article will be devoted to important points of the decision tree, references to the players in the space, and potential partners.

TO RETROFIT OR GREENFIELD?: THAT IS THE QUESTION

Okay, we've decided that we want a Smart Building, but before we start mapping out the plan, there are a couple of major decision points to navigate. First, we need to understand the scope of our project. Buildings, like homes, have one incredibly difficult component that must be dealt with—namely, people.

People are awful, full stop. The most terrible thing about them is that they all have an opinion. Getting the people in your organization to agree on how their office space is going to be configured can be a thankless task. As every facilities manager can attest, no one is ever happy, ever. No matter what you propose, someone is going to want it a different way, so the recommended attitude to cultivate is a flexible and calm one. That said, there are some techniques which can smooth the process. Getting management to buy-in is key.

Another important pre-consideration is to make your proposal fit-to-purpose. Simply put, this means that although you can find IoT

devices that will track the CO₂ levels and heat signatures of every person in the building, or will manipulate the exterior lights to display Pac-Man, it may not be appropriate for your organization, nor add anything to the bottom line. As with any project, identifying the major points of pain—the issues that need to be addressed versus those which can be—will point you in the right direction.

Next, what level of disruption will be acceptable? Is this an actively occupied building, and if so, how critical to the company workflow is the activity in the space? An in-place upgrade can be implemented in incremental steps, but will the retrofitting and time spent in systems cutover be offset by the productivity boost and energy savings? If that answer points you towards a greenfield approach, then the cost of moving from one space to another must be added into the mix. Moving costs—especially the often-underestimated cost of packing and unpacking—can be significant, and the disruption moving causes can be equivalent to that from in-place upgrades.

Careful calculation, with plenty of leeway built in, is a crucial part of planning your smart building project. Most organizations, and very likely your boss, are happier if potential snags and cost overruns are identified early in the process. Late-breaking delays can undermine the success of the project and negatively affect morale, especially if they are perceived by management as something that should have been foreseen. They can also make you the target of people's frustration when things go sideways. Again, a flexible and calm attitude, combined with proactive and frequent communication with all the stakeholders, is going to make your smart building project, and your people, happier.

Back to the greenfield versus retrofit question. Both paths have benefits and drawbacks, and since there is no exact right answer, it may come down to philosophy. A smaller organization which is more tolerant of upheaval may prefer to upgrade and retrofit an older space, saving on the cost of new construction, pacing the budget burn rate, and scaling as needed. A larger organization may elect to take the hit and pay upfront for the latest tech in a brand-new space, so as to avoid the maintenance and recurring costs of an incremental approach in an older building. There are arguments to be made on both sides. Let's explore two real world scenarios.

Greenfield—Duke Energy Center: The Duke Energy Center, located in Charlotte North Carolina, was a brand-new construction, costing north of \$800 million (**Figure 1**). In this case the building was purpose-built to showcase the latest energy efficiency modalities, and to have a low carbon footprint. IoT devices were

employed to add intelligence to almost every aspect of the building.

As you can see in **Figure 2**, 16 separate building systems were integrated into one seamless net, reducing operational expenses and energy consumption by 22% overall. One of the most intriguing features is the intelligent elevator bank system called “Destination Dispatch.” This system assigns elevators to occupants with common destinations, making the fewest possible stops while transporting passengers. In addition, the building’s emergency systems, irrigation and water use, access control, and disaster recovery and lighting equipment were all integrated in this new construction.

Retrofit—Empire State Building: The barriers that seem to be in the way of retrofitting—for example equipment cost, workflow disruption, time, and rewiring—have

all been reduced with the newer generation of IoT and networking technologies. The case study below from Building Engines Blog [3] shows how an iconic building, the Empire State Building, was converted to a smart operation (**Figure 3**).

As you can see in **Figure 4**, even a high-rise built almost 100 years ago with little to no consideration of networks or modern connectivity can be retrofitted without destroying its gothic 1930s charm. In this case wireless IoT devices and networking technology were used, connecting the disparate building systems together in one seamless high-speed net. Employing these technologies, the old building realized a 38% reduction in overall energy consumption, saving over \$4 million dollars a year in energy costs (**Figure 5**).

SMART BUSINESS: MAKING THE CASE

In this section we’ll go over the value proposition of smart buildings. Critical to any final decision will be the return on investment (ROI). Management will want to see how your proposal will affect the bottom line, and more importantly, how long it will take to recoup the required investment (**Figure 6**). Once you’ve spent the time to identify what must be done and at what cost, a key selling point for the project will be an estimate of the duration of this recovery period.

Traditionally, one of the greatest incentives to invest in a smart building is the increase in asset value. A smart building can be a great selling point for a future owner, and the overall energy savings provide a year-over-year boost to the bottom line. According to commercial real estate (CRE) industry metrics, building automation and integrated control systems can realize anywhere from 10% to 40% energy cost savings for the building occupants. However, there is another, even greater factor that is often overlooked, and it can seal the deal.

In a typical pre-pandemic working environment, employees spent around forty hours per week in the office. This adds up to around two thousand hours on an annual basis. As any HR manager will tell you, human capital makes up the lion’s share of operating expense in any business, and can be upwards of 90% of recurring costs in some tech-heavy industries. As a rule of thumb, JLL [6], a Canadian commercial real estate firm, has developed the 3-30-300 rule.

Simply put, this equates to \$3 per square foot per year for utilities, \$30 for rent, and \$300 for payroll. Using this rule, JLL claims that the greatest financial savings that result from optimizing a workplace do not lie in energy but in productivity.

According to their model, smart buildings provide direct benefits to the people who occupy



FIGURE 1
Duke Energy Center, North Carolina (Image Source: Intelligent Building Europe [2])

<p>Originally designed as a replacement for Wachovia’s One Wachovia Center headquarters in Charlotte, North Carolina, it was intended to be a state-of-the-art, sustainable and efficient corporate base for the firm.</p> <p>Although that vision was realised, the building was instead occupied by Duke Energy, and has become an iconic fixture in the centre of Charlotte. Each time the Carolina Panthers, the city’s American football team scores a touchdown, LED lights colour the tower blue, the team colours.</p>	<p>WHAT MAKES IT SMART?</p> <ul style="list-style-type: none"> ■ 16 separate building systems are integrated on one network, including HVAC, security, metering, lighting, water management and emergency preparedness; this reduces operational expenses and energy consumption by 22% ■ Digital sensors are used to manage energy consumption, allowing building temperatures to be adjusted within a predetermined range ■ A lighting control system reduces energy consumption by dimming perimeter lights in response to available natural light and turning lights off when the room is vacant; the lighting control system learns occupant behaviour over time and automatically adjusts according to long the office will be occupied ■ “Destination Dispatch” elevator controls assign elevators to passengers with common destinations, which will then transport them while making the fewest possible stops ■ The building reuses approximately 10 million gallons of harvested water each year including groundwater, rainwater and HVAC condensation, meeting roughly 80% of the cooling tower’s water needs and 100% of the building’s irrigation needs
--	--

Year	2012
Architect	Thompson, Ventulett, Stainback & Associates
Owner	Wells Fargo
Tenant	Duke Energy
Key numbers	880 million, cost in US dollars

FIGURE 2
Duke Energy Center Project (Image Source: Intelligent Building Europe [2])

them, increasing morale, which in turn produces significant positive impacts on the company's bottom line. In numbers, they estimate that 43% of the total value of a smart building project comes from enhanced employee productivity, and another 41% from increased employee retention. In addition, they say that another 7% results from improved employee wellness, 7% from utility cost savings, and only 2% from maintenance savings! This surprising result is because there is a 100-fold factor weighting the bottom-line cost reduction on the company's human capital expenses. Basically, having happy employees equates to business success.

As per JLL [6], "[the] productivity gains can be achieved by making workplaces physically comfortable, enabling fewer distractions and the ability to concentrate fully on tasks. Furthermore, it has been proven that there are direct links between human-focused, intelligent building systems and a company's ability to recruit the brightest talent. Not to forget, active participation and signed consent of the employees are vital to a system's success. However, if all things are considered, the promise of energy efficiency, better access control, greater comfort and environmental responsibility all come down to a high return on investment (ROI) for smart buildings."

PLANNING IT OUT: A SAMPLE WORKSHEET

As we can now see, the question of how to plan your smart building project has many moving parts. Losant has a comprehensive guide on their website (**Figure 7**) [4]. Some of the tangibles are cost and time. These can be further broken down into:

Upgrade old

- Planning and Logistics
- Equipment Acquisition
- Labor/Installation
- Disruption to normal workflows (Opportunity/Productivity losses)
- Incremental Upgrades/Maintenance
- Ongoing Maintenance

Buy new

- Building Site Acquisition
- Financing
- Consultant/Design Partner
- Construction Costs and Time
- Packing/Moving/Unpacking
- Disruption to normal workflows (Opportunity/Productivity losses)

Each of the above need to be estimated, and it helps tremendously to have a partner who can assist in getting you the proper numbers, and who has the experience to avoid common pitfalls. So, let's look at some partners.

PARTNERS

Whether you're going to build a whole new facility, or just upgrade the one you are in, the partner you choose for assistance is important. Depending on which way you go, you may have a monolithic turnkey solution from a single large

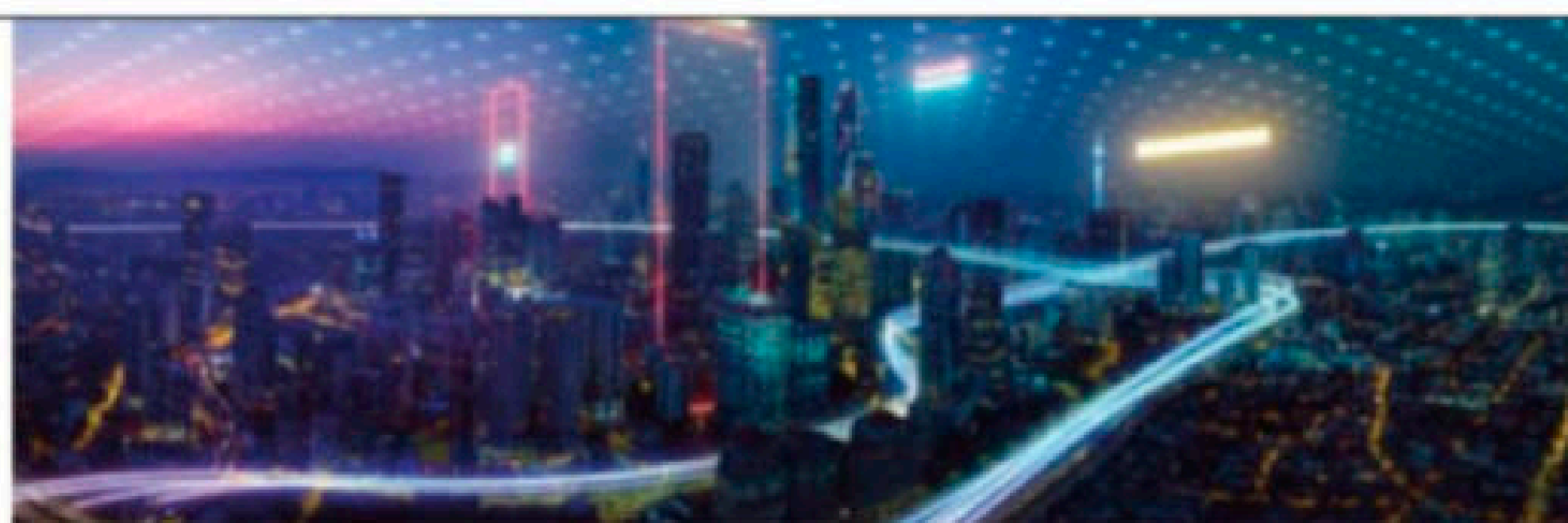


FIGURE 3
Old to SMART (Image Source: Building Engines [3])

Take the example of one of the world's most iconic older buildings, the Empire State Building. Built in 1931, in the past decade it was retrofitted with smart technology. It has since seen a 38 percent decrease in energy consumption and enjoyed [\\$4.4 million in energy savings](#) every year, according to Jones Lang LaSelle (JLL).

While ten years ago retrofitting a building may have been prohibitively expensive for most, that's not necessarily still the case. You can connect modern sensors wirelessly, without installing expensive hard-wiring. Many sensors are low energy and low maintenance, with some harvesting energy from the light and movement around them, and others being ultra-low power, long battery life.

FIGURE 4
Retrofitting Grandpa (Image Source: Building Engines [3])



The size of the global smart building market was \$57.3 billion in 2020, and is projected to grow to [\\$67.6 billion this year](#). According to Fortune Business Insights, it is forecast to reach \$265.37 billion by 2028 at a compound annual growth rate of 21.6 percent.

Some more old school CRE professionals are a little more skeptical of smart buildings however. And for understandable reasons.

FIGURE 5
Market Size (Image Source: Building Engines [3])

supplier, or a collection of equipment sources that will interconnect using a specified standard protocol to a unified management center. Here are some of the vendors I found, along with a synopsis of their capabilities. This list is far from exhaustive, but it represents a cross-section of the market space.

Cisco: As the saying goes—no one ever got fired for choosing Cisco. Known worldwide for their networking and productivity solutions, Cisco is a natural player in this market. As you can see on their website [9], they have a whole division called Spaces, which is focused on building and supporting the smart modern workplace (**Figure 8**). Their eBLE line of IoT devices are a rock-solid way to move your project forward. And they offer consulting services to help scope and identify your best-case solution.

Siemens: Siemens is a global player in a wide variety of industries (**Figure 9**). They too have a full suite of products and solutions for the workplace of tomorrow (**Figure 10**). They have a very informative video that explains their philosophy: "Building the Future Today" [7]. More information is also available on their website [8].

Cohesion: On a different level there are companies like Cohesion [10]. Cohesion's product is a state-of-the-art management and integration suite, allowing the integration of many types of IoT devices and systems under a single platform. For a more incremental and standards-based system, especially one where a phased roll-out is desired, Cohesion, and other similar vendors, is a good option to explore.

IotaComm: Another player in the integration space is Iota Communications. Their value add is a comprehensive integration and management platform called Delphi360. More information is available on their website [11].

Aurecon: International in scope, this Singapore-based design and construction company has a global presence, with a strong focus on the Asian and Oceania marketplaces. Aurecon can provide a soup-to-nuts contract, including consultancy, design, architecture, building and site development and more. For the greenfield experience they are a great partner.

Locatee: Locatee is a Swiss company that specializes in consulting and optimizing your workplace and efficiency. They are a great resource to use to scope out your project and arrive at the right-size solution that meets your company's needs. Locatee's website offers whitepapers, business case studies, partner references and more [5].

RISK FACTORS

Okay, things are looking good! You've got a plan and it's only Wednesday afternoon. You can kick back and put your brain in neutral, right? Hold on there Anakin, you forgot one more thing about people. People hate change.

Assuming your organization has not been exposed to a smart building in the past, all the things you are about to propose are going to be viewed with a range of suspicion. You may have the most logically thought-out proposal, but nevertheless be prepared for a few curveballs. Not least will be the fear of the unknown, as outlined in the Smart CRE website [12]. There they list a few of the most common concerns and objections that you may run into.

Cyber security: Connecting nearly every device in a building to the Internet has an obvious risk. Cheap IoT devices are not equipped with the best security software, and can potentially be hacked. To fight this problem, many countries and organizations are implementing regulations or laws to force companies to make their IoT devices more secure. However, as every parent

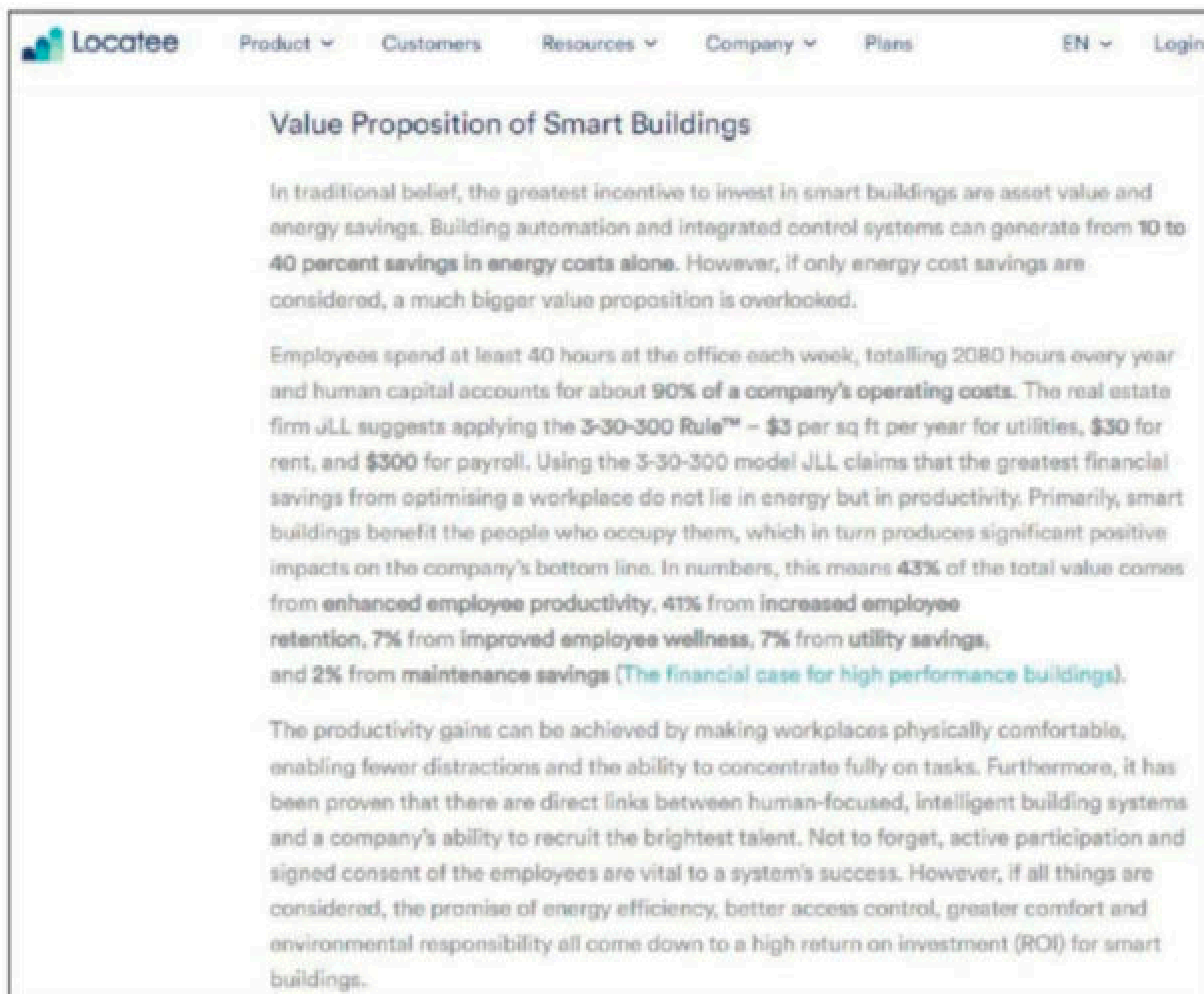


FIGURE 6
Making the Biz Case (Image Source: Locatee [4])

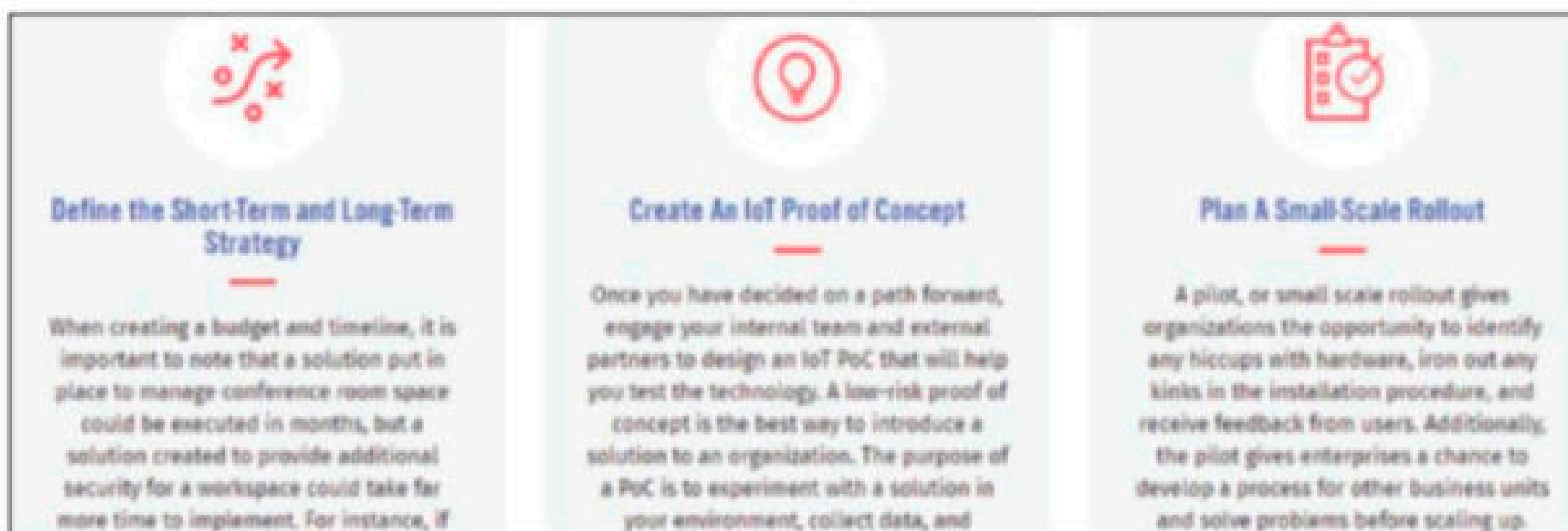


FIGURE 7
Losant Comprehensive Smart Environment (Image Source: Locatee [4])

Additional materials from the author are available at:
www.circuitcellar.com/article-materials
 References [1] to [12] as marked in the article can be found there.

who has had the neighborhood whiz kid hack into their doorbell camera knows, any connected device can be attacked.

Initial installation costs—sticker shock: The costs of installing smart technologies can be tens or even hundreds of thousands of dollars. This can be a large investment for smaller companies. Of course, this also depends on the size of the building and the number of solutions installed. Even though there must be some investment in the beginning, you want to be able to confidently assure your team that there will be a positive return within a few months or one to two years. According to JLL [6], it is one of the biggest myths that smart buildings are not a good investment decision.


Internet connection: Another disadvantage of smart buildings is the need for a constant internet connection. A reliable network needs to be established to get the full potential out of the intelligent technology. If your building becomes a pumpkin because the internet happens to be down, expect that you may get a lot of annoyed early-morning wakeup calls. A robust fallback plan is key.

Usability: Even though many children nowadays are growing up with technology, there is still a large pool of people that are not accustomed to it. One goal of smart technology and IoT must be that it can be easily used by everyone without prior knowledge. Training your folks on what to expect and giving leeway for adoption will help.

DRAWBACKS: MAYBE IT'S TOO SMART

Given the above, you may meet some resistance from your organization. Concerns that are often raised about smart buildings center around the hype versus the reality, security, and privacy. While it may seem ideal to be able to track your employees' every movement and mouse click, putting the shoe on the other foot and imagining Big Brother looking over your shoulder every moment of the day injects a dose of reality. Employees need to feel that they're being treated with respect, and that they are trusted with their jobs' responsibilities without a corporate nanny-bot spying on their every move. The adoption rate and ultimate success of the new smart environment will depend on how comfortable you can make it for everyone.

CONCLUSION

Well, that's all for me. I sincerely hope that this article will help get you started on your smart building project. If you were inclined to, say, email a couple Starbucks coupons my way as you bask in the warmth of the hero-adulation you're going to get from your boss as she presents your work to the board, I would not refuse. Go forth smarter and better-prepared, and to quote Miracle Max: "Have fun stormin' da castle!" 

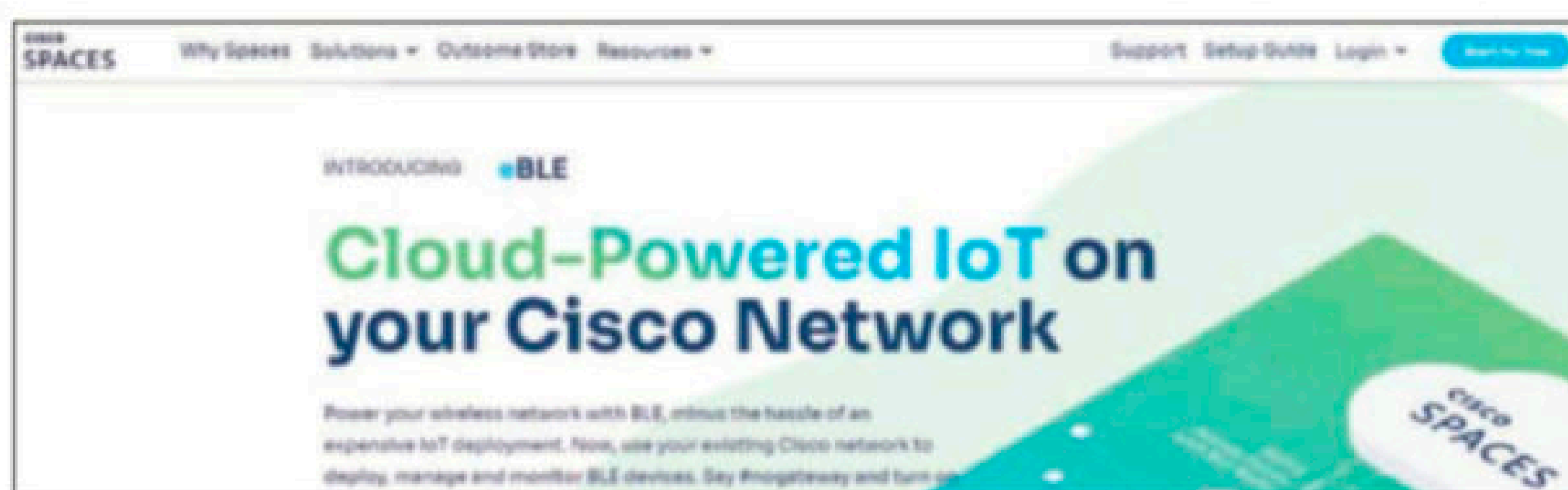


FIGURE 8
Cisco Spaces (Image Source: Cisco Spaces [9])

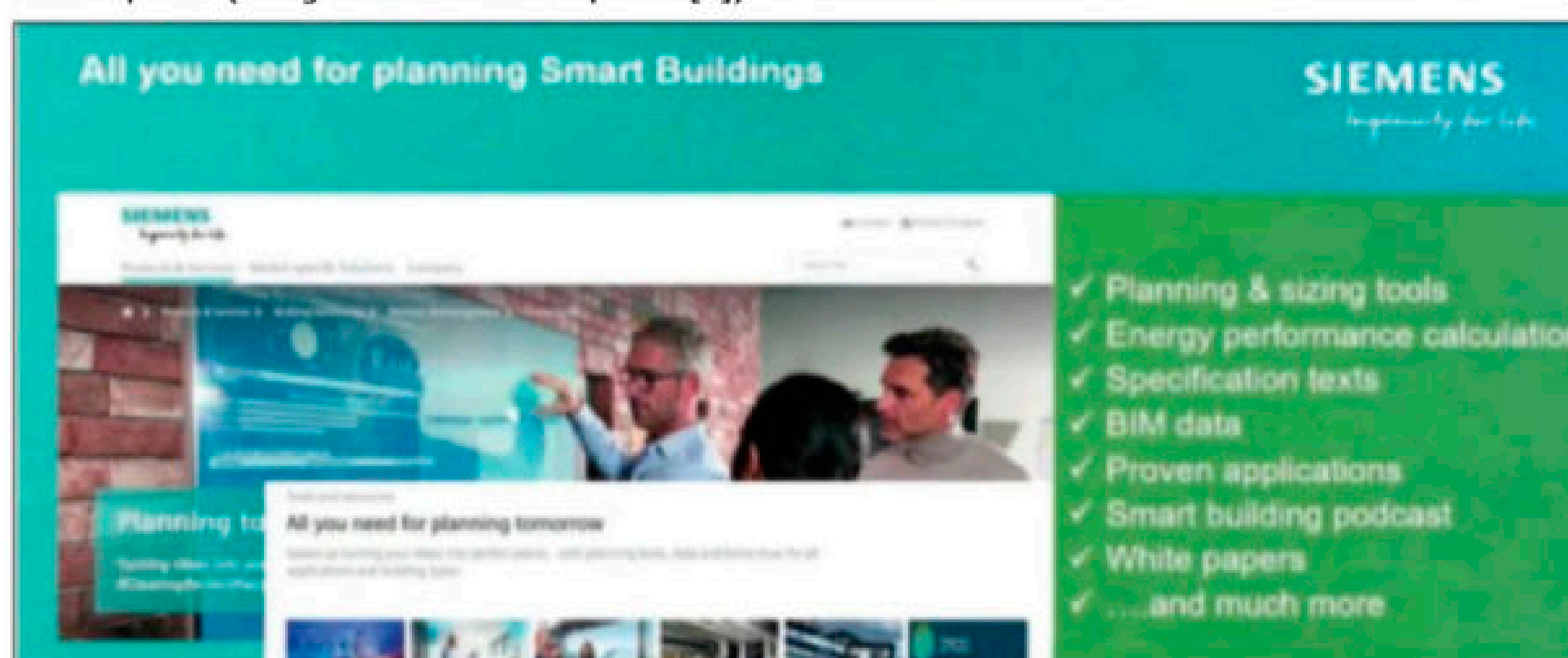


FIGURE 9
Siemens Partnership (Image Source: Siemens [8])

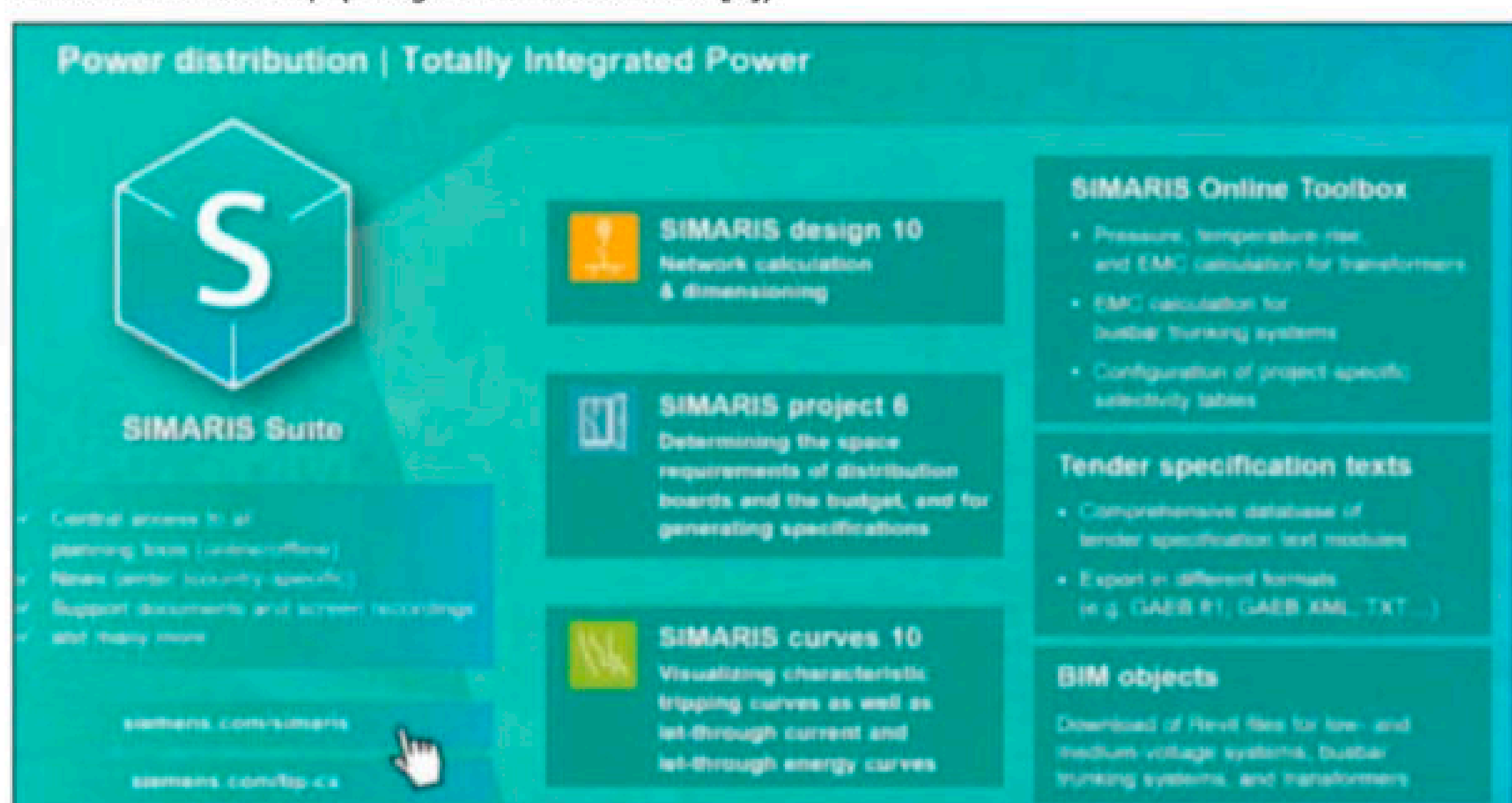


FIGURE 10
Smart Building Metrics (Image Source: Siemens [8])

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.



Datasheet: PC/104 Boards

Vital in Harsh Environments

DATASHEET

By
Sam Wallace,
Editor-in-Chief

FIGURE 1

PC/104 boards are inherently rugged—built to sustain high levels of shock and vibration—and thus ideal for applications in harsh environments, such as military equipment.


Over thirty years since its inception, the PC/104 form factor family is alive and well. For rugged applications with space constraints, PC/104 and its variants continue to deliver.

Last year marked three decades since the PC/104 Consortium standardized the PC/104 form factor, so called for its 104 pins on the interboard connector (ISA) in the original specification. The PC/104 standard is noteworthy for introducing the embedded stackable computing concept. Since its beginning with the ISA bus, it's grown to include more recent desktop computing technologies with PCI and PCI Express—and subsequently spun off its wider family of related technologies, including the PC/104-Plus, PCI-104, PCI/104-Express and the popular PCIe/104.

PC/104 was designed to be rugged (**Figure 1**). That, coupled with the ability of its architecture to implement rapidly evolving PC technology into embedded computing products—by taking advantage of PC market adoption, performance, scalability and growing silicon availability worldwide—has led to its permeation in a

wealth of industries, including industrial IoT, aerospace, medical, defense, transportation, and more.

Recent years have seen several additions to the PC/104 family and standard. A few years ago, the PC/104 Consortium introduced a revision to PCI/104-Express and PCIe/104 that provides an additional option called “OneBank,” which utilizes a smaller, lower-cost bus connector that is compatible with full-sized PCIe/104 connectors. Other recent innovations include Mini PCIe sockets. These let system designers mix and match add-on functions, leveraging the emerging ecosystem of Mini PCIe peripheral cards.

Datasheet last covered PC/104 boards in 2019, and as such the products in this article's gallery were not released exclusively within the last twelve months—with some exceptions, such as the recently released VL-EPM-E9 from Versalogic. These products prove that PC/104 remains a vital and viable form factor. 

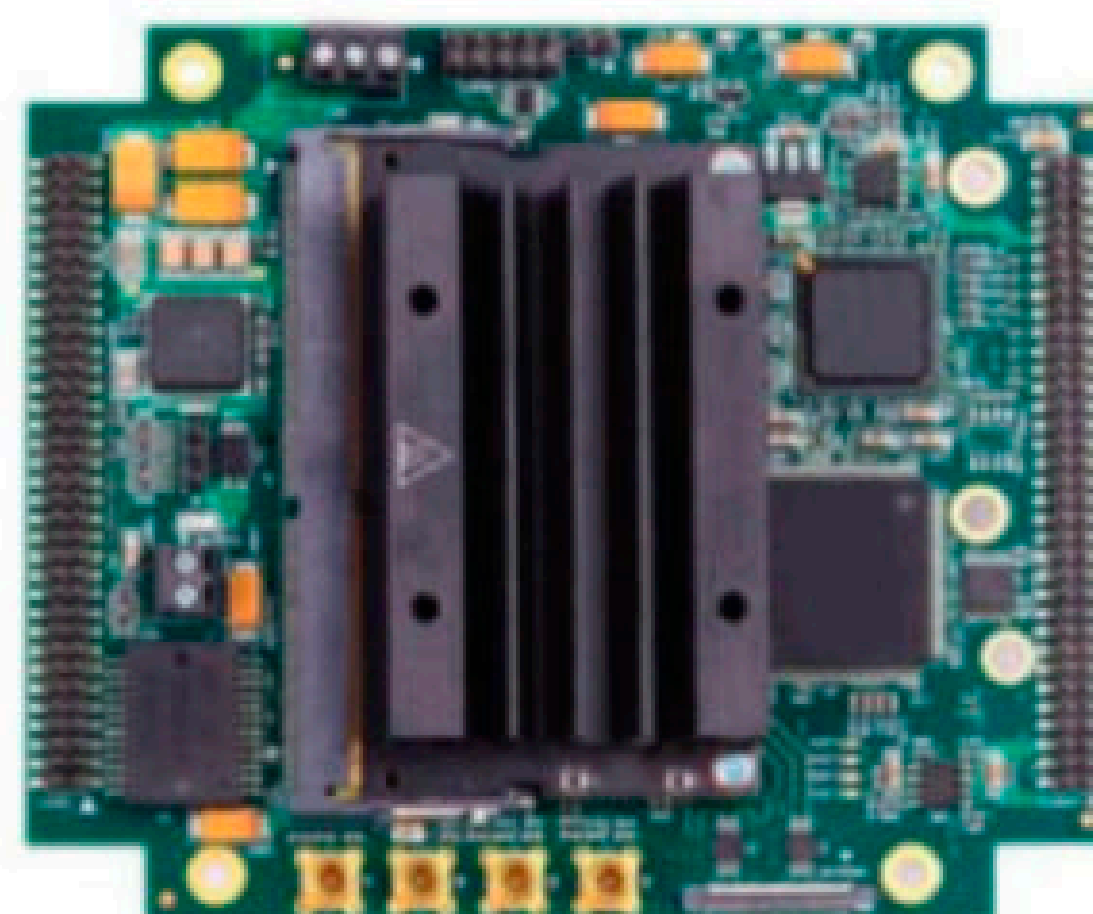


First PCIe/104 Module with NVIDIA's Quadro P1000

Adlink's CM5-P1000 is a PCIe/104 Type 1 module designed for AI-enabled solutions. It adheres to size, weight and power (SWaP) restrictions for aerospace and defense applications. Adlink touts this module as the first PC/104 board to be equipped with Nvidia's Quadro P1000 graphics, which brings with it 640 CUDA cores and 1.8 TFLOPs. The CM5-P1000 is very rugged, with a -40°C to $+85^{\circ}\text{C}$ operational temperature range, high humidity tolerance, and the ability to cope with shocks and vibrations.

- NVIDIA Quadro P1000
- 4 GB GDDRS
- 640 CUDA cores
- 1.8 TFLOPs
- 4x DisplayPort
- Operating temperature of -40°C to $+85^{\circ}\text{C}$
- OS support: Windows 10, Windows 7, Linux drivers, 64 bit
- 116mm x 96mm

ADLINK Technology Inc.
adlink.com

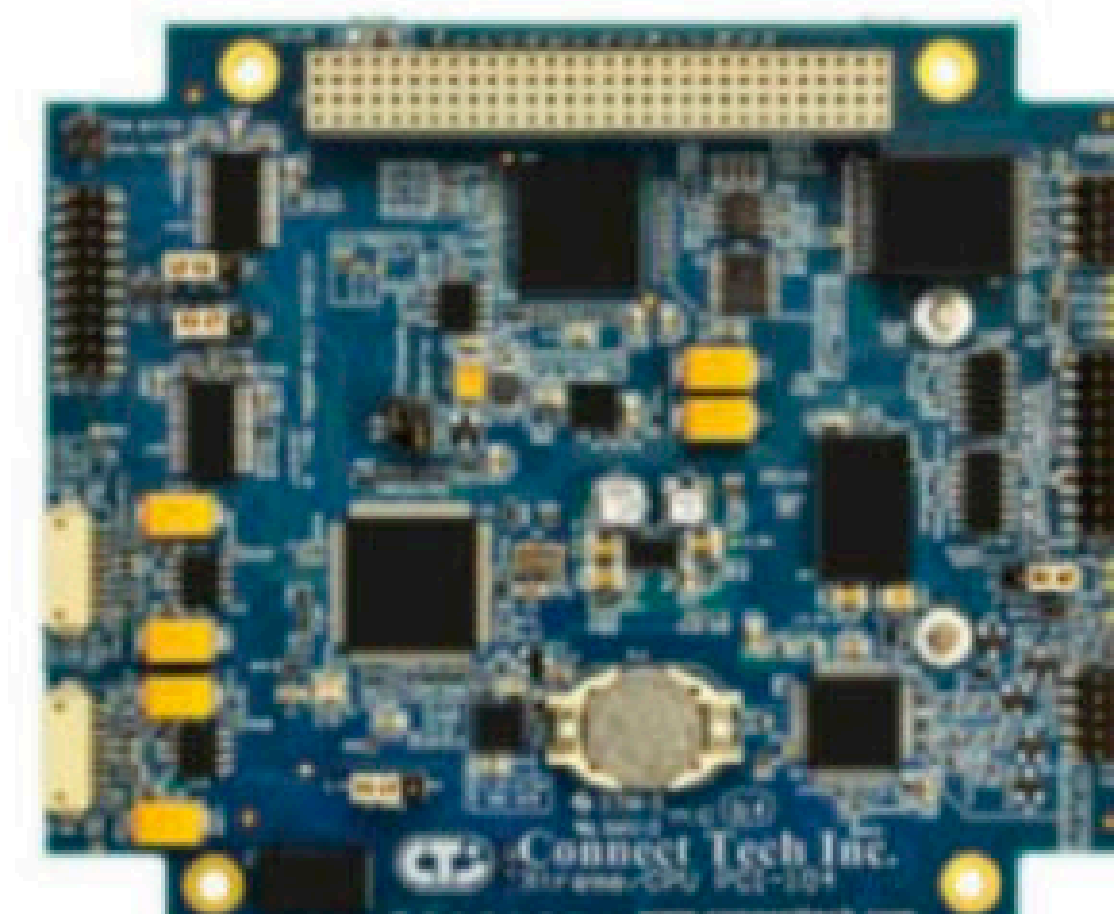


Carrier Board Suited for Vision-Enabled AI Applications

Advanced Micro Peripherals' JetStream is a multi-channel video AI carrier board for NVIDIA Jetson Xavier NX. With dual HDMI and eight composite video inputs allowing multi-channel video AI applications to be rapidly developed and deployed, this PCIe/104 form factor, SWaP-optimized solution is geared for applications in military, communications, transportation, mining, and energy industries. The JetStream is supported by a comprehensive board support package and application notes designed for fast development of AI vision applications.

- 2x HDMI video inputs at up to 1080p60
- 8x Composite PAL/NTSC/RS-170 video inputs
- HDMI 2.0 Display at up to 4K resolution
- Multi-channel AI Video Inference
- H.264/H.265 Video Encoding and Streaming
- Rugged, real-time video capture/analyze/stream solution
- M.2 (Key M) NVMe storage
- Gigabit Ethernet

Advanced Micro Peripherals
ampltd.com



A PCIe/104 SBC with Multiple Processor Options

Connect Tech's Xtreme/SBC is a small single-board computer (SBC) that utilizes the PCIe/104 form factor with 4 x1 PCIe lanes. Connect Tech offers the Xtreme/SBC with a variety of embedded processors, including Intel Atom, Freescale i.MX51, TI OMAP, and NVIDIA Tegra. This SBC allows instant access to PCIe/104 peripherals such as FPGA solutions, multi-port serial, and frame grabbers. The modular and scalable Xtreme/SBC has access to most current embedded processors and can be upgraded with future generations of processors such as ARM-based options or Intel Atom. The specifications listed below are for the Atom Z500 processor option.

- 1.1GHz to 1.6GHz
- 4x PCIe x1
- 1 GB memory
- Up to 4GB on-board flash disk
- Intel GMA 500 graphics
- VGA, LVDS display interface
- 1x SATA external storage
- 4x USB 2.0
- Ethernet 10/100/1000
- ATX Supply Input

Connect Tech
connecttech.com

DATASHEET URLS:

ADLINK CM5-P1000 Datasheet: <https://www.adlinktech.com.cn/Products/PC104SBCs/PCIe104/CM5-P1000?lang=en>

Advanced Micro Peripherals Jetstream Datasheet: <https://www.ampltd.com/products/pc104-h264-jetstream/>

Connect Tech Xtreme/SBC Datasheet: <https://connecttech.com/ftp/pdf/CTI-X-Xtreme-SBC-PCIe-104-Single-Board-Computer.pdf>

Datasheet: PC/104 Boards

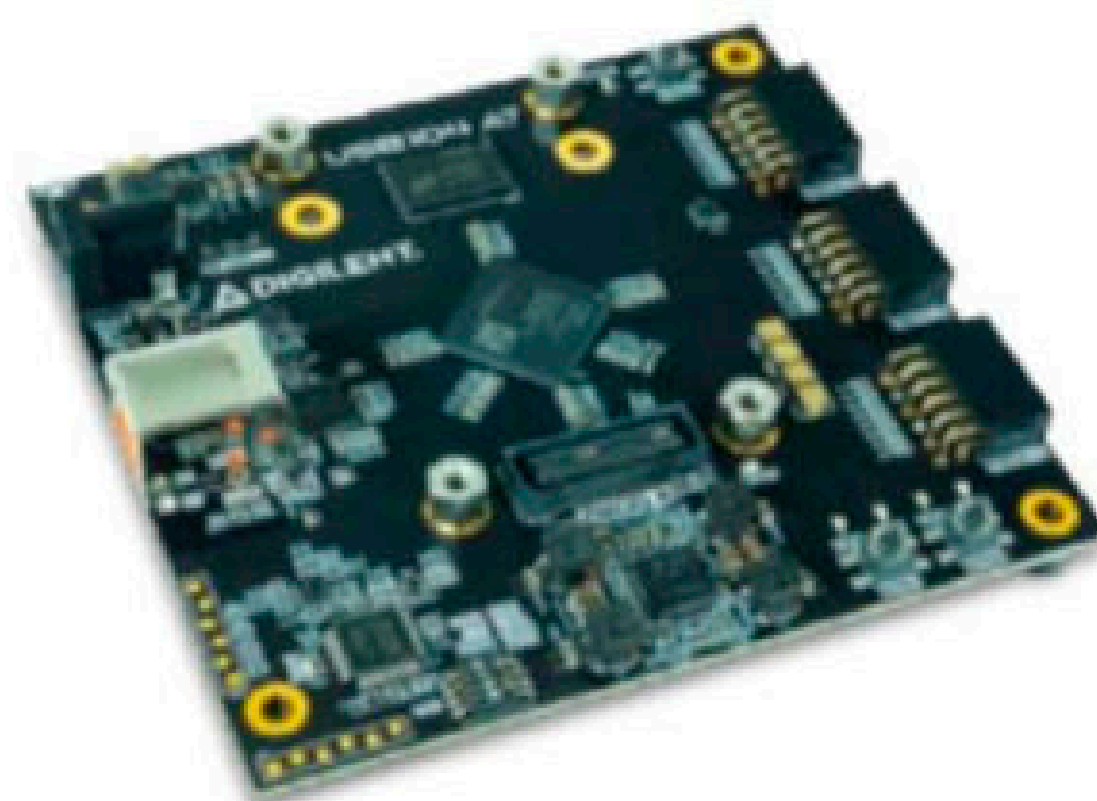


Fourth-Generation SBC Built for Long-Lifecycle Products

The original Athena debuted from Diamond Systems in 2004, and as the name suggests, the Athena IV is the fourth generation in this family of rugged, wide-temperature PC/104 SBCs with integrated data acquisition. It maintains the same physical shape and most of the I/O connectors and features as the previous generations, but with certain key updates to keep up with the state of the art. This has allowed Athena platform customers with product lifecycles that extend beyond that of their CPUs to transition painlessly to each subsequent generation, and indeed the Athena IV is an ideal choice for customers that wish to build a PC/104 system with a long life expectancy.

- Heat sink cooling
- 5VDC input voltage
- Dimensions: 106mm x 114 mm
- -40°C to +85°C operation fanless
- Windows and Ubuntu Linux OS support
- Choice of E3845, E3940, or E3950 Atom CPU
- 4-8GB RAM soldered on COM
- Up to 128GB eMMC Flash integrated onto COM

Diamond Systems
diamondsystems.com

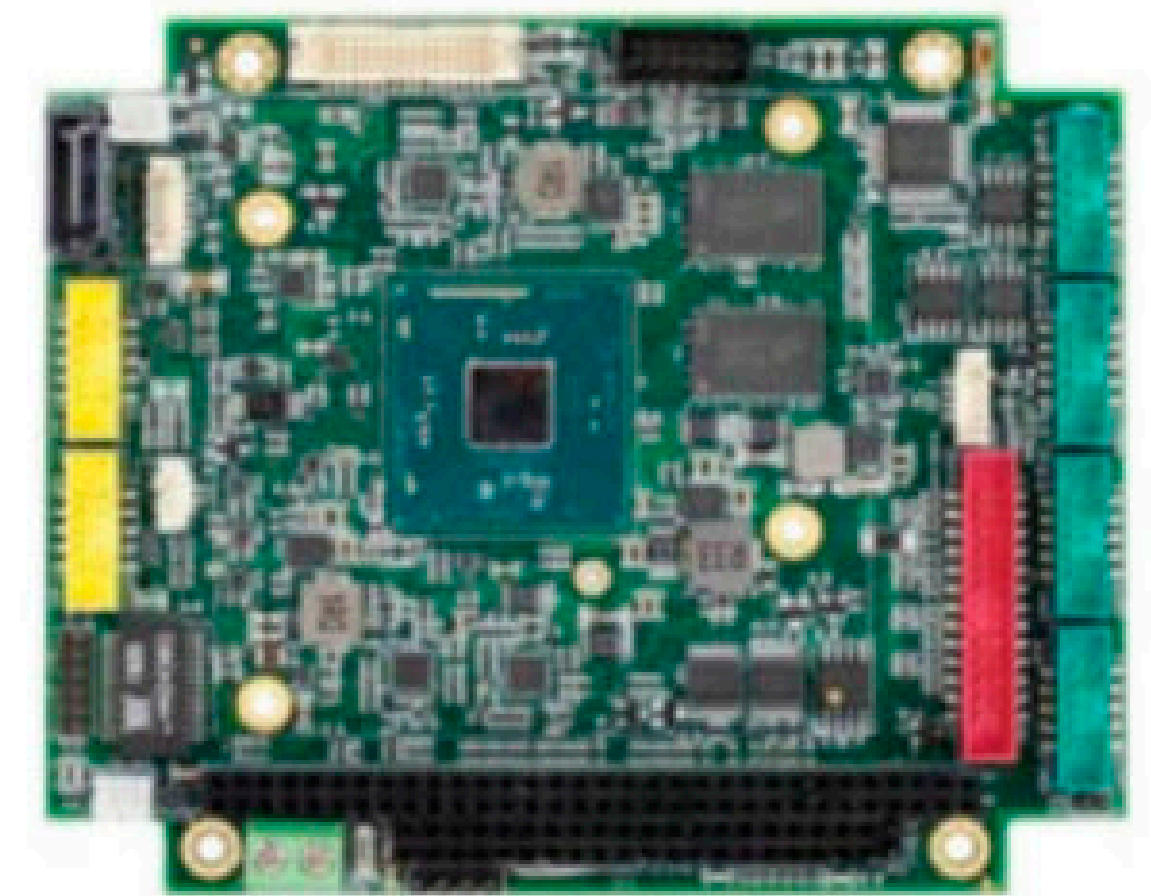


Rugged, Compact, SWaP Package with a Wide Operating Temperature Range

Digilent's rugged and compact USB104 A7 brings versatility and power in an industrial SWaP package. It features the Xilinx Artix-7 XC7A100T that has an operating temperature of -40°C to +100°C, and also a SYZYGY port to allow for high-speed data acquisition through the addition of Zmod expansion modules. The USB104 A7 is ideal for military aerospace, embedded OEM, telecommunications and industrial automation applications.

- FPGA features include 101,440 logic cells, 240 DSP slices, 4,860 Kb memory, 300 I/O pins, and 8 GTP 6.6Gb/s transceivers
- Memory: 512MB DDR3/DDR3L with 16-bit bus @ 800Mbps (1.5V) or 667Mbps (1.35V)
- 16MB Quad-SPI Flash for storing FPGA configuration
- Powered from a 5V external DC adapter attached to barrel jack or from USB header
- 2 push-buttons and 4 LEDs
- One standard SYZYGY port (Zmod) and three standard Pmod ports

Digilent
digilent.com



UEFI and Legacy Bios Support in This Stable PC/104 Board

ICOP's IBW-6954 is a stable and powerful PC/104 embedded solution based on the Intel Braswell Quad-Core CPU with 4GB DDR3L RAM onboard. ICOP chose Braswell as it's an Intel platform that can support both UEFI and Legacy BIOS. The IBW-6954 supports Gigabit LAN, 4x COM, 4x USB, I2C, Parallel, Mini PCIe, HDMI, LVDS, VGA, and SATA interface for storage for development use.

- 4GB DDR3L 1600 RAM
- UEFI and Legacy BIOS
- ISA Bus x1, MiniPCIe (full-size) x1
- 5VDC power requirement
- 116mm x 96mm
- 81g in weight
- Windows 10 OS support

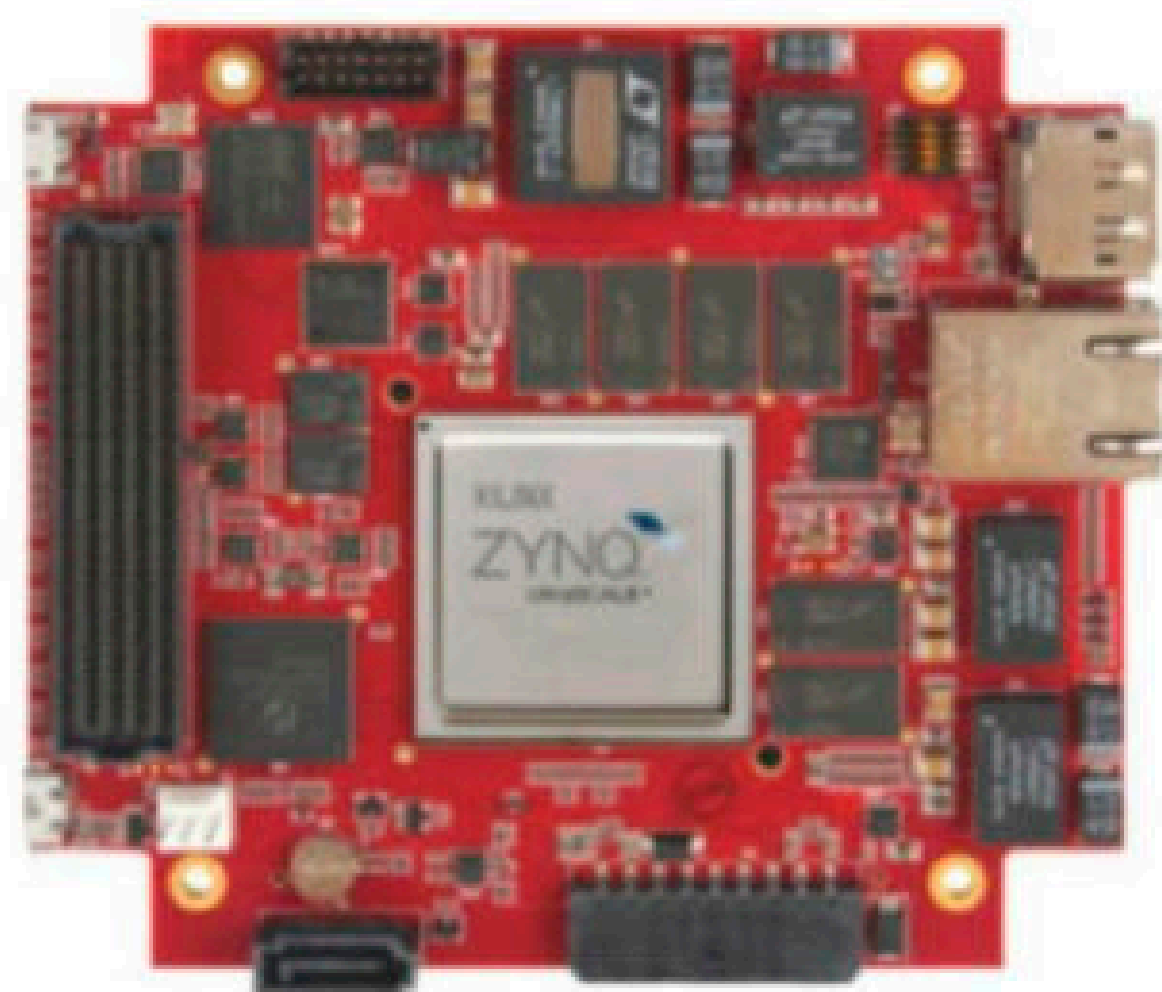
ICOP Technology
icop.com.tw

DATASHEET URLS:

Diamond Systems Athena IV Datasheet: https://www.diamondsystems.com/products/display_datasheet.php?handle=athenaiv

Digilent USB104 A7 Datasheet: <https://digilent.com/reference/programmable-logic/usb104a7/reference-manual?redirect=1>

ICOP IBW-6954-E4 Datasheet: https://www.icop.com.tw/download_resource/IBW-6954-E4?tags=34&selected=34

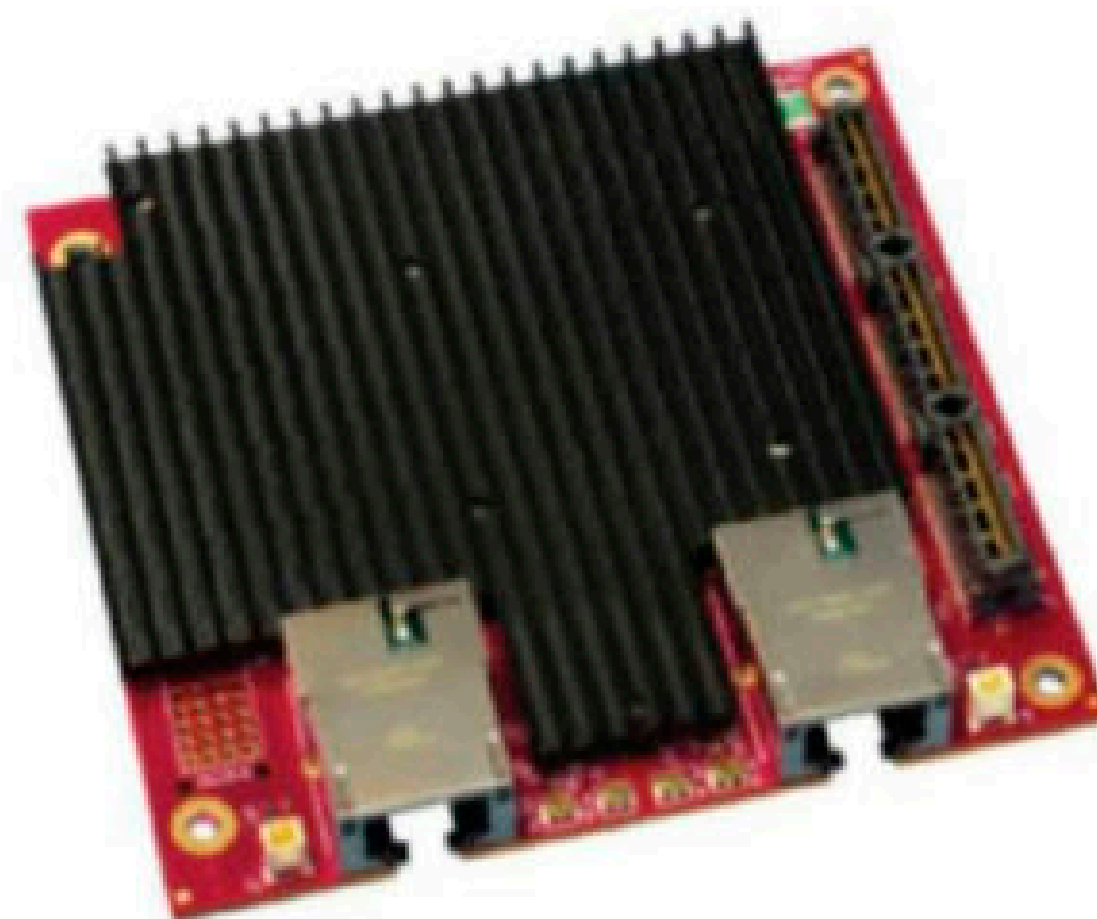


Versatile Board Based on the Zynq MPSoC Family

Sundance Digital Signal Processing Inc.'s PC104Z is based on the Xilinx Zynq UltraScale+ multiprocessor system-on-a-chip (MPSoC) family. It comes in a PCIe/104 form factor, and depending on the choice of Zynq MPSoC, it can be used for digital communication, image processing including AR/VR applications, and even high performance computing (HPC). The PC104Z offers four Gen 2.0, x1 lane PCIe interfaces through a switch which allows four PCIe/104 cards to be connected to the ARM on the Zynq which acts as the host.

- Up to 4GB of DDR4 to PL
- Flash memory for user application bitstream
- Micro SD card for running OS and embedded applications
- Two x4 lane PCIe Gen4 to PL
- FMC expansion site with 10 GTH at 16.3Gb/s transceivers and 80 LVDS IO pairs
- Video Codec Unit H.265/H.264 with XCZU7EV
- GTH, GTY, 100EMAC and Interlaken, when fitted with XCZU11EG

Sundance DSP Inc.
sundancedsp.com



Extremely Rugged Ethernet Expansion Card

The VL-EPM-E9 module is an extremely rugged dual 10 Gigabit Ethernet add-on expansion card, for use with copper cabling. With its stack-down three-bank PCIe/104 pass-through connector, it is ready for inclusion in a PC/104 stack. On-board magnetically isolated RJ45 ports make for easy connection to other network nodes.

- Dimensions: 90mm x 96mm x 23mm
- Weight: 148g (5.22 oz.)
- +12V input voltage from PCIe/104 three-bank connect
- Typical power of 6.9W
- Bus interface: PCIe/104 three-bank, stack-down
- -40° to +85°C operating temperature range
- Meets MIL-STD-202H specifications for shock and vibration
- Network: Two 10GBASE-T copper ports (100/1000/2.5G/5G/10GBASE-T auto detect)
- Compatible with Windows and Linux

VersaLogic Corporation
www.versalogic.com



Rugged SBC Featuring Intel Apollo Lake-I

Winsystems' PX1-C441 single-board computer (SBC) comes in a PC/104 form factor but with PCIe/104 OneBank expansion. It features the Intel Apollo Lake-I Dual-core or Quad-core SOCs for graphics processing. With 8GB LPDDR4 system memory and an eMMC for solid-state storage of an OS and applications, this small, rugged board is geared to industrial IoT, control, transportation, Mil/COTS, and energy markets.

- Intel Apollo Lake-I E3900 Processor (Dual- or Quad core)
- Up to 8 GB Soldered LPDDR4 System Memory
- Time Coordinated Computing
- Precision Time Measurement
- -40°C to +85°C Operating Temperature Range
- PC/104 Small Form Factor with PCIe/104™ OneBank™
- Multiple Displays Supported

Winsystems
www.winsystems.com

DATASHEET URLS:

Sundance DSP Inc. PC104Z Datasheet: <https://www.sundancedsp.com/products/fpga-boards-modules/pcie-104/pcie104z-with-xilinx-zynq-us-mpsoc/>

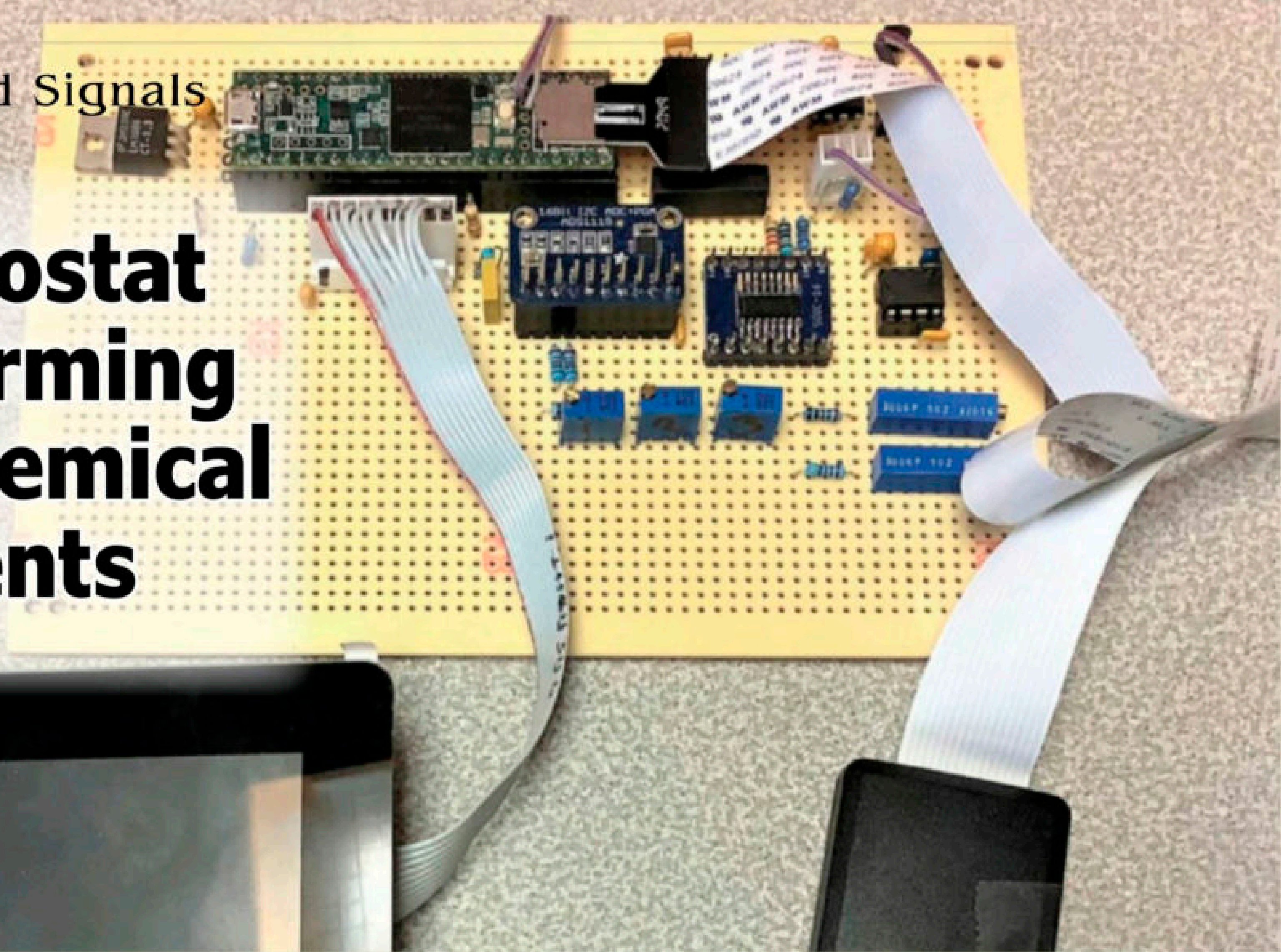
VersaLogic VL-EPM-E9 Datasheet: <https://www.versalogic.com/wp-content/themes/vsl-new/assets/pdf/DS-EPM-E9.pdf>

Winsystems PX1-C441 Datasheet: <https://resources.winsystems.com/datasheets/px1-c441-ds-v1.2.pdf>

Picking Up Mixed Signals

A Potentiostat for Performing Electrochemical Experiments

By
Brian Millier



The powerful Teensy 3.5 microcontroller (MCU) module, combined with a Bridgetek-controlled thin-film transistor (TFT) touchscreen display and some analog components, makes for a low-cost Potentiostat instrument.

Over the years I've designed and built a number of different potentiostats to be used mainly for teaching labs, as well as a few specialized ones for researchers at the Dalhousie University Department of Chemistry where I worked. My job and interests were in electronics instrumentation, not chemistry, so my knowledge of electrochemistry lies more in its measurement than in the construction of electrochemical cells, or the ultimate goal of their measurement.

Basically, electrochemistry examines the relationships between electrical potential and chemical phenomena. This is done by placing specialized electrodes into a chemical solution and measuring the relationship between the applied voltage and the resulting current flow. **Figure 1** is a drawing of such a cell, with the electrodes labeled. Note that the electrode naming convention wouldn't be familiar to an electronics person, but this is how electrochemists label them. What I refer to in the article as "cell bias" goes to the working electrode, and the "current sensing" is connected to the counter electrode. I'll explain the reference electrode later. The instrument that performs this measurement was called a voltammograph by the faculty members with whom I worked, but the term potentiostat is synonymous and more commonly used today.

A potentiostat is an instrument that provides a linear voltage ramp with a starting potential and a final potential. In some cases, it provides for a two-part ramp where the intersection of the two ramps is defined as the switching potential. The definition of these three voltages is a bit of puzzle for an electronics person like myself. When an electrochemist talks about a starting voltage of positive 1.0V, for example, the ramp

voltage must be programmed to put out a *negative* 1.0V. The same applies for the switching and final voltage polarities. I take care of this inversion in my software.

The "ramp speed" is specified in volts (or millivolts) per second. A "step" value, in volts (or millivolts) is defined as the ΔV between each cell current measurement. Cell current can vary widely, so the current-measuring circuit must be capable of measuring currents less than $1\mu A$ and up to 10mA. The 10mA upper range would not apply to all electrochemical measurements, but it does handle all of the situations that one would run into in a teaching lab, which is generally where my instruments were used.

For some measurements, it is necessary to superimpose a square wave (of a user-selected amplitude value) upon the linear voltage ramp. The period of this square wave is fixed: the upper portion is applied for one current sample and the negative portion is applied for the next sample.

The ramp voltage is generated by a digital-to-analog converter (DAC). The time between when the DAC has been set to a new voltage and when the cell current is actually measured by the analog-to-digital converter (ADC) is called the "measurement delay." This is chosen by the operator, and is measured in milliseconds. The execution of a full single-segment or two-segment ramp is called a "cycle." The operator enters the number of cycles to be run and the current vs. voltage data is collected and stored in a file, separated out into cycles. The final parameter that can be set is the "cell relaxation" time (seconds). The potentiostat contains a relay which connects and disconnects the counter electrode from the potentiostat's circuitry. The cell relaxation time is the interval between the connection of the counter electrode

to the potentiostat's current-sensing circuitry and when the voltage ramp actually begins.

In electrochemistry, the range of ramp voltages that are needed doesn't exceed $\pm 2V$. Not being a chemist, I don't know why this is, but it's a nice coincidence. Since the total range is less than 5V, the analog circuitry required can be implemented with devices that work on a single 5V power source. This is the same voltage supplied by the USB specification and is also the voltage used to power the microcontroller (MCU) module that I use.

POTENTIOSTAT CIRCUITRY

When I started building potentiostats for both teaching labs and researchers, good-quality op-amps were not available with single-supply, rail-to-rail specifications. At the time, good-quality op-amps were generally powered by $\pm 12V$ power supplies. Even though the voltage range needed in electrochemistry was less than $\pm 2V$, in my earlier designs, I used quality op-amps, mentioned above, which operated on $\pm 12V$ power supplies. A side effect of this was that several of the op-amps had to be able to dissipate a significant amount of power, due to the large differential between the power supply and the normal working range of potentiostat output voltages.

Today, with the advent of single-supply op-amps having rail-to-rail inputs and outputs,

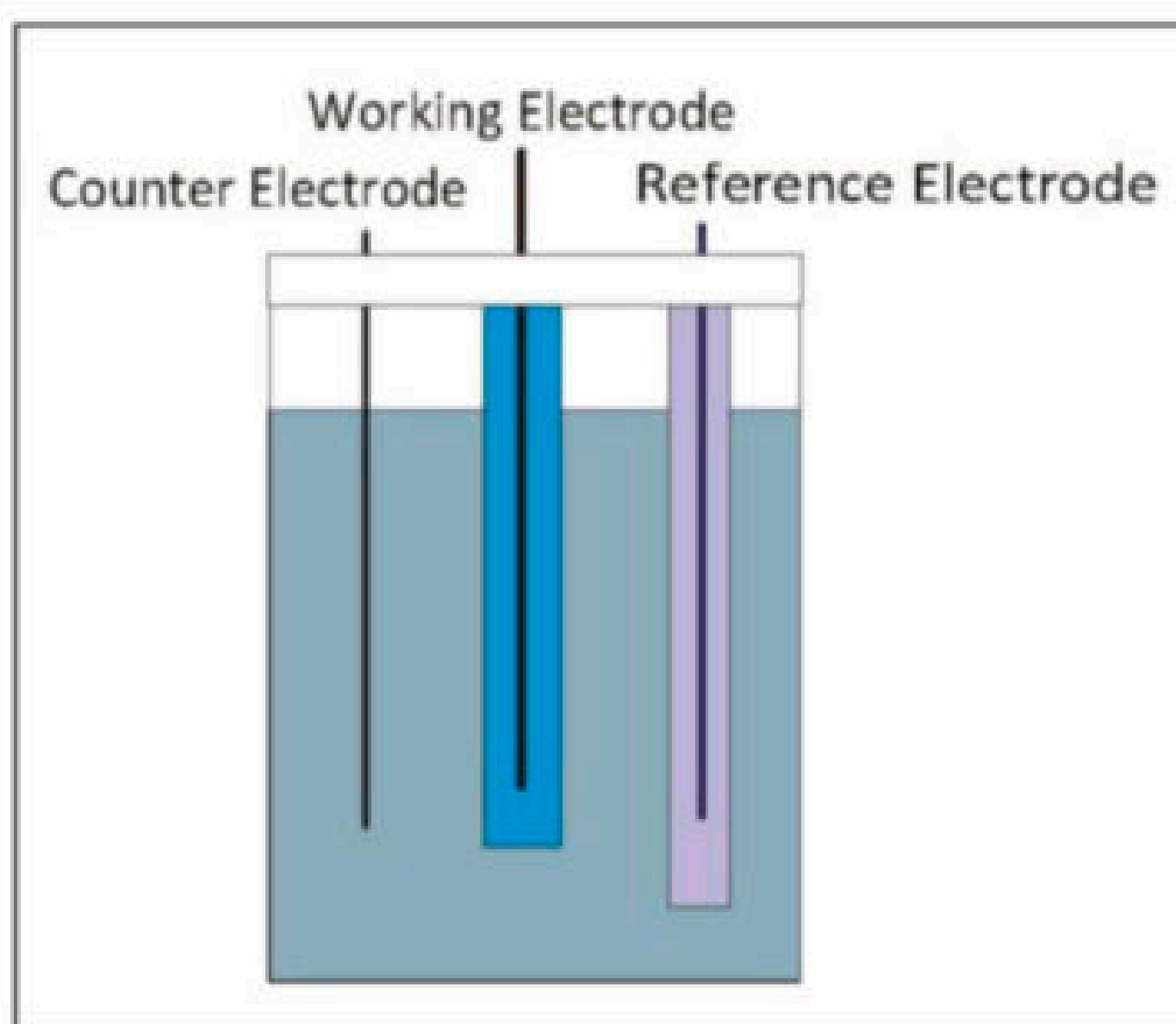


FIGURE 1
This is a diagram of an electrochemical cell. The electrode naming convention is strange to an electronics person, and is described in the text.

the design can be a lot simpler. Although the potentiostat must be capable of providing a bipolar output range, the electrochemical cell itself is not grounded. Therefore, the potentiostat can generate cell bias voltages, and measure the resulting current, using a "virtual" ground (common) with a potential of 2.5V. The resulting $\pm 2V$ cell bias voltages will then fall into the 0.5V to 4.5V range. This allows for the use of a single 5V power supply for the potentiostat's analog circuitry—assuming that one uses op-amps that are capable of rail-to-rail operation and have a 5V power source. This 5V power is readily supplied by a USB charger which connects to the micro-USB socket on the Teensy 3.5 MCU

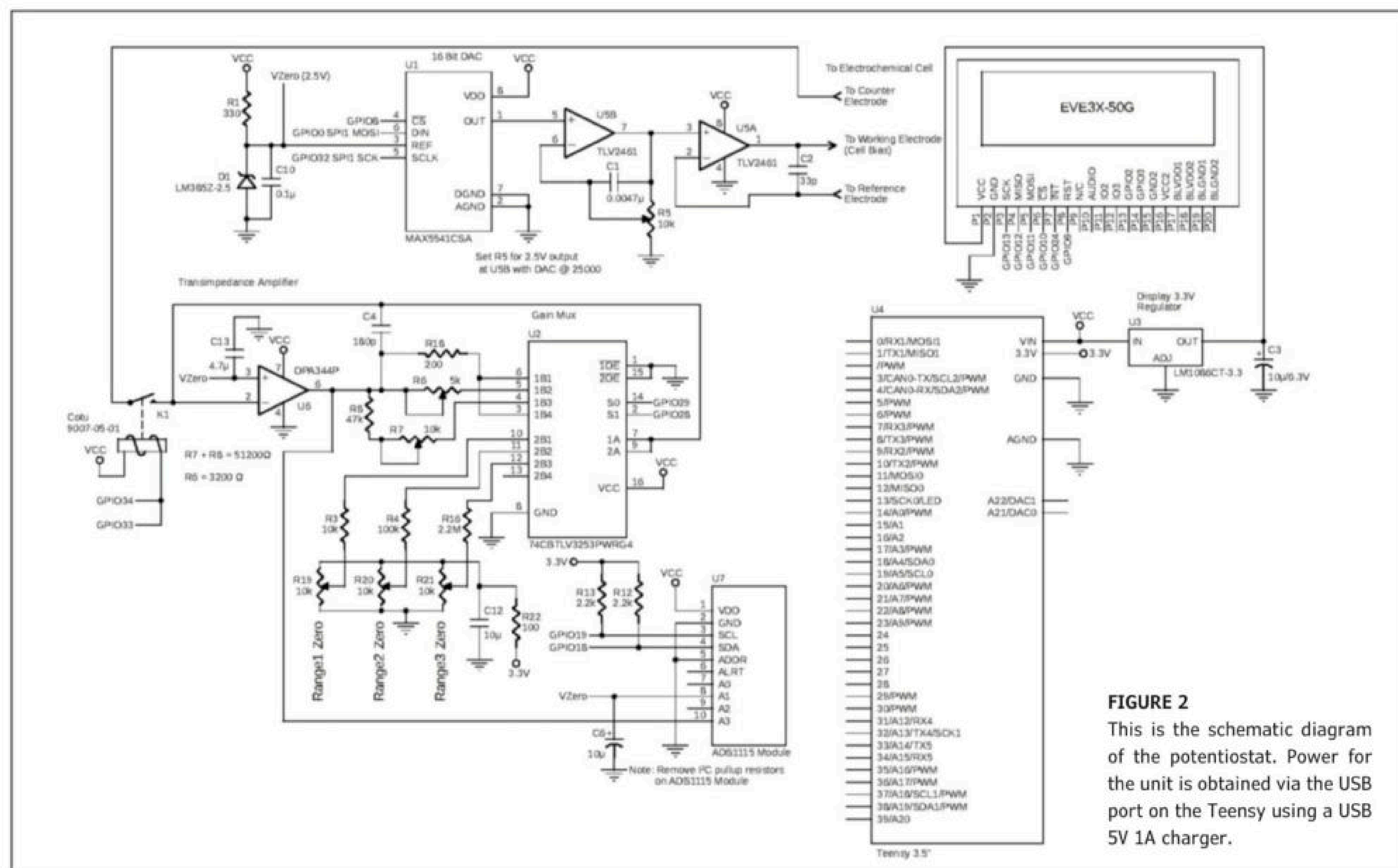


FIGURE 2
This is the schematic diagram of the potentiostat. Power for the unit is obtained via the USB port on the Teensy using a USB 5V 1A charger.

COLUMNS

module which runs the potentiostat. The 5V USB power is pinned-out on the Teensy 3.5's VIN pin. The Teensy 3.5 module contains a 3.3V LDO regulator which powers the MK64FX512VMD12 ARM MCU on that module, and this 3.3V is also used to power a few things on the external mixed-signal board.

With the power supply requirements out of the way, let's look at the potentiostat's analog circuitry. Please refer to **Figure 2**. To provide a voltage ramp in the $\pm 2V$ range with a resolution of 1mV would require a 12-bit DAC (that is, a range of 0V to 4.096V in 1mV intervals). Because the voltage ramp might have a variable amplitude square wave added to the ramp value, under user control, I decided to instead use a 16-bit DAC—the Maxim MAX5541. This DAC can be powered by 5V and it requires an external reference. I use an LM385Z-2.5 2.50V reference device for this purpose. The DAC output, over its full 16-bit range, will be 0V to 2.50V. For convenience, I wanted the resolution of the DAC to be 10 counts per millivolt. As mentioned earlier, I am using a virtual ground set at 2.50V (also provided by the LM385Z-2.5 reference). I use an amplifying buffer (U5B) with an adjustable voltage gain, set by pot R5. During initial calibration, the MAX5541 DAC is sent a value of 25000, and R5 is adjusted to provide a 2.50V output at the U5B buffer's output pin. Given that I've chosen a "virtual" ground of 2.50V, this 25000 reading corresponds to a "zero-volt" cell bias voltage. Similarly, a DAC value of 5000 represents a DAC output of -2V, and a DAC value of 45000 represents a +2V DAC output. So, each DAC count represents 0.1mV. The virtual ground node is labeled Vz0 in Figure 2.

The bias voltage is fed to the non-inverting input of U5A (a TLV2461 dual op-amp). This op-amp has rail-to-rail inputs and outputs. The TLV2461 couldn't source/sink the 10mA currents needed for the potentiostat all the way out to each rail. But, the voltage range used by the potentiostat is only 0.5V to 4.5V, and this op-amp can provide the necessary current over that reduced voltage range, since it never gets within 0.5V of either rail.

As mentioned earlier in the article, the cell potential is not actually the voltage between the working electrode (bias voltage) and the

counter electrode (current sense). Instead, there is a third electrode called the reference electrode, and the cell potential is the voltage between it and the counter electrode. To accomplish this, the inverting input of U5A is connected to the reference terminal, and this feedback loop adjusts the voltage going to the working electrode (cell bias) accordingly. The DAC, dual op-amp and software make up the ramp voltage generator (with provision for a square wave modulation of the ramp, in software, under user control).

As mentioned at the start of the article, how one specifies the cell bias voltage (connected to the working electrode) is different from how electronics people would think. In electrochemistry, when one specifies a cell bias voltage of +1V (for example), the actual voltage that the potentiostat must supply to the working electrode is -1V. Since that's the way electrochemists express the cell bias voltage, my software inverts the polarity of the operator-selected ramp voltages before sending the applicable values to the DAC.

All that's left is to provide circuitry to measure the cell current. This could be accomplished by placing a low-value shunt resistor in series with the cell's counter electrode, to the "virtual ground" node and then measuring the voltage across it. This would result in a small discrepancy between the ramp generator's output voltage and the true cell potential, due to the small voltage drop across the shunt resistor. Instead, a potentiostat generally incorporates a trans-impedance amplifier (TIA). A TIA converts a current into a voltage output without a shunt resistor, so there is virtually no error due to a voltage drop across it.

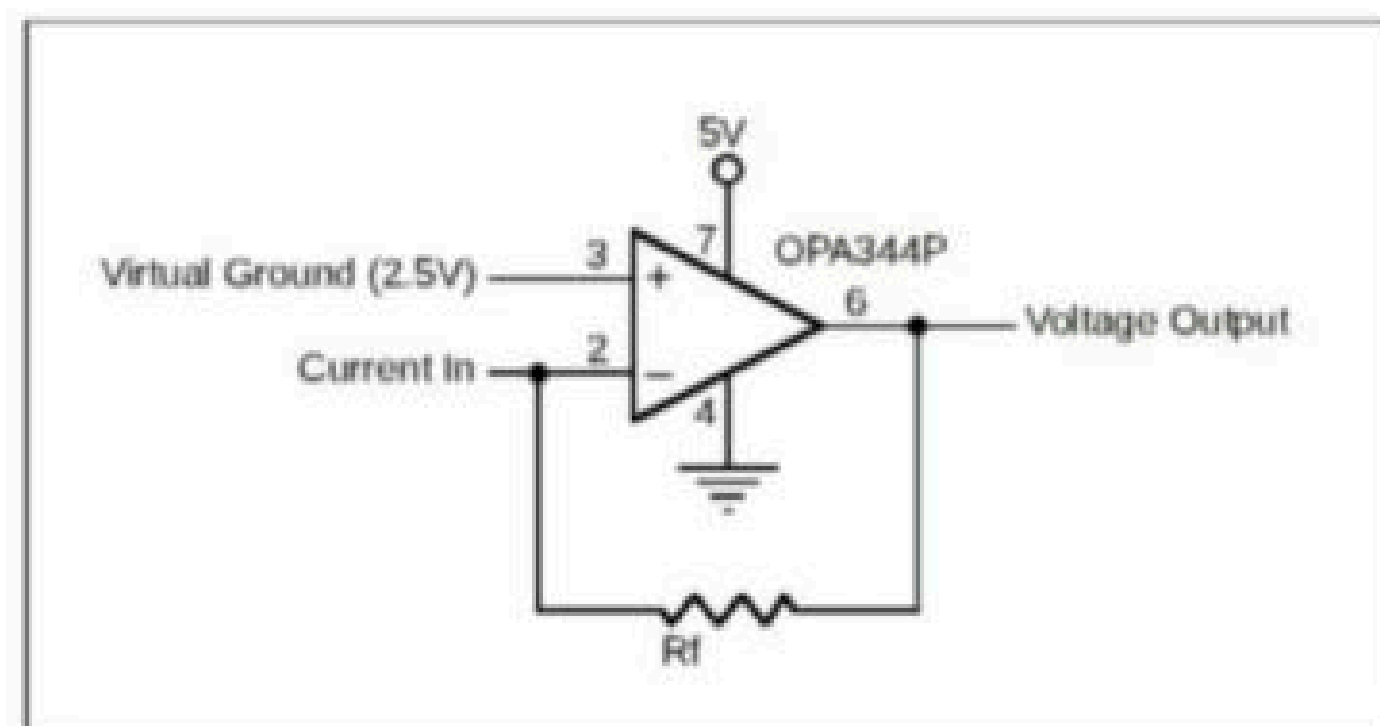
Figure 3 shows a basic TIA circuit. Note that the non-inverting op-amp input is connected to a virtual ground, which in this project is 2.50V. Due to the presence of negative feedback resistor R_F , and the extremely high open-loop gain of the op-amp, the voltage between the inverting and non-inverting inputs will be virtually zero. Therefore, the current coming in will be measured with respect to the 2.5V virtual ground potential. Since the sum of the currents at the TIA's inverting input node must be zero, the op-amp's output voltage will equal:

$$V_{OUT} = -I_{SENSE} \times R_F$$

I chose a T.I. OPA344P op-amp for the TIA. It's a rail-to-rail op-amp that operates on 5V. The TIA's output pin must be capable of sinking the 10mA maximum cell current that the instrument is designed to measure. The OPA344P is capable of $\pm 15mA$ short-circuit output current, so it's just powerful enough to handle the maximum current being sensed in this design.

FIGURE 3

This is the basic trans-impedance amplifier (TIA) circuit that I use to measure cell current.



To handle a wide range of currents from the low μA range to 10mA, two methods were used. For the TIA itself, I provide three different feedback resistors with ratios of 16 between them: 200 Ω , 3,200 Ω and 51,200 Ω . The 74CBTLV3253 multiplexer switches have a resistance of about 5 Ω , so the 3,200 and 51,200 values are trimmed slightly by pots R6,R7 to maintain 16:1 and 256:1 ratios. The 200 Ω resistor's slightly higher effective value is due to the multiplexer's switch resistance.

The TIA voltage output is measured by a ADS1115 16-bit ADC. This ADC has an internal PGA providing six full-scale ranges—I use the 256, 512, 1,024 and 2,048 millivolt ranges in this project.

The combination of the above two scaling methods results in nine current ranges with full-scale values between 10 μA and 10mA. With a 16-bit ADC, accurate measurements below 1 μA are possible.

For each of the three TIA current ranges, there is a separate zero offset adjust using the second half of the 74CBTLV3253 multiplexer and pots R19 through R21. There is a software routine "offset_Adjust_Screen()" which I call during initial calibration. It switches among the three multiplexer settings and displays the raw ADC output. With nothing connected to the current sense pin, you adjust these pots, one per TIA range, to produce ADC outputs as close to zero as possible.

The ADS1115 is a real gem of an ADC. It is capable of 16-bit readings at sample rates up to 860sps. The PGA ranges all have exactly a 2.000 ratio between adjacent ranges, so software trimming and calibration is not required when using the various PGA ranges. It uses an I²C interface to the MCU, which is handy in this project as both the DAC and the touchscreen display use the SPI interface. I run the I²C interface at 400KHz. The ADS1115 chip is housed in a tiny 10-pin MSOP package which would be tough for me to mount. Luckily, Adafruit (and many other vendors) sell this chip on a small breakout board with a 0.1 pin header, costing virtually the same price as the chip itself.

Since the analog circuitry feeding the ADS1115 is capable of providing signals up to 5V, I use 5V to power the ADS1115 module. The I²C lines on the Teensy 3.5 operate at 3.3V, so I removed the two 10k I²C pull-up resistors on the Adafruit ADS1115 breakout board and added pull-up resistors to the 3.3V power supply on my main board (R12,R13).

TOUCHSCREEN DISPLAY

I had a Matrix Orbital Eve3-50G display on hand which I had used for my two-part *Circuit Cellar* article "Building a Touchscreen Display Using a Bridgetek BT81x" (*Circuit Cellar* May

2022 issue #382 and July 2022 issue #384) that covered the Bridgetek BT81x series of touchscreen controllers. This display has a 5" screen with 800 x 480 pixel resolution and a capacitive touchscreen. A Bridgetek BT815 controller chip handles the display and also provides an interface to the capacitive touch controller chip. All communication between the host MCU and the display occur over a high speed SPI bus. I use two separate SPI ports on the Teensy 3.5—one for the display and a secondary one for the MAX5541 DAC.

Note that in this article, references to the Bridgetek BT815 controller may involve references to FT81x as well as BT81x. The reason for this is that the FTDI company originally designed this line of display controllers and later sold or spun off the product line to Bridgetek. Therefore, even the latest available Bridgetek documentation and library/function names use a mix of the FT and BT naming conventions. This dual naming convention is carried through into this article.

Having spent a lot of time working with this display for my aforementioned articles, I decided to use this display for the potentiostat project. For a potentiostat, you need at least an 800 x 480 pixel display to properly graph the data. There are also many user-entered parameters which must be routinely entered or changed. I consider a 5" touch screen display the smallest practical display for this project. Having used resistive touch screens often in the past, I knew that the capacitive touch screen found on the Eve3-50G display was much more accurate and user-friendly.

The Matrix Orbital Eve3-50G display is meant to connect to the host MCU using an FFC-20P cable. These 0.5mm pitch flex cables/sockets don't lend themselves to the Vector protoboards that I use to build projects. The Eve3-50G also contains a PCB footprint (on 0.1" centers) containing all the necessary signals (likely there for factory tests). I was able to solder a header for a 10-conductor ribbon cable to the Eve3-50G board, and connect it to my host board this way. However, while working with this display for the earlier *Circuit Cellar* articles, I ran into a strange anomaly. The Eve3-50G would work fine when connected to a Teensy 3.5 module, but wouldn't work when I connected it to any other member of the Teensy family (Teensy 3.2, 3.6, 4.0, 4.1). All of these Teensy modules are pin-compatible and I tried all of them out

ABOUT THE AUTHOR

Brian Millier runs Computer Interface Consultants. He was an instrumentation engineer in the Department of Chemistry at Dalhousie University (Halifax, NS, Canada) for 30 years.

using the same Vector protoboard, wiring and interconnect cable. I ruled out that the Teensy's execution speed could be too high, since the Teensy 3.2 is a much slower 72MHz Arm Cortex M4. The Teensy 3.6 is identical in execution speed to the Teensy 3.5, and the Teensy 4.0 and 4.1 are much faster 600MHz Arm Cortex M6 MCUs. When I connected my logic analyser to the SPI port lines on the display, I could see no difference in the data being sent to the Eve3-50G display, regardless of what Teensy MCU I had connected. Neither could I see any meaningful differences in the SPI signals on my 'scope either, using all four Teensy MCUs. In the end, I was never able to solve this issue, and I spent a lot of time at it.

While the Eve3-50G display module works well with the Teensy 3.5 used in this project, I would recommend using the newer Matrix

Orbital Eve3x-50G module. I was in contact with Matrix Orbital for some technical information while writing the earlier articles. They sent me a sample of the newer Eve3x-50G module and it worked fine with all of the Teensy MCUs mentioned above. It also contains a separate 20-pin socket for a standard ribbon cable, in addition to the FFC socket.

The Eve3x-50G display comes with some added features:

- Haptic Feedback—This is a tiny “shaker” motor which you can turn on/off to act as haptic feedback when you touch the screen, for example.
- A low-pass filter, low-power amplifier and a small loudspeaker—The BT815 controller itself contains a basic sound synthesizer function. Its audio output is a Sigma-Delta signal with a carrier frequency of about 30MHz. The Eve3 display sends this raw sigma/delta signal to the FFC-20P socket, but only the Eve3x modules contain this added circuitry.

The Eve3x models contain a significantly brighter screen than the EVE3 models—1000 nit Sunlight readable versus 300-600 nit for the Eve3. This higher brightness requires a more powerful LED backlight. Therefore, the Eve3x displays require more power. Since I wanted to use the USB port on the Teensy 3.5 to supply the power for the whole project, I was limited to the 500mA that the USB specification calls for. I was able to keep the project's entire current draw to roughly 500mA by dropping the Eve3x-50G LED backlight power to 50% using the following line of code:

```
wr8(REG_PWM_DUTY + RAM_REG, 64);
// Backlight PWM duty 50 %
```

The REG_PWM_DUTY register expects values between 0 and 128 for duty cycles between 0 and 100%. The RAM_REG constant is the base address that must be added to all register address definitions to form the true memory address. Even at 50% backlight intensity, the display is perfectly readable indoors. **Figure 4A** and **Figure 4B** show the front and back of the Eve3x-50G display.

Although I covered the topic of BT815 graphics programming in depth in my earlier *Circuit Cellar* articles, there is one graphics feature that was a prerequisite for this project: fast real-time plotting of the data to an X-Y graph. The BT815 controllers use a display list architecture to render graphics to the screen. Using this method, and a collection of high-level graphics “widgets,” it's possible to render sophisticated GUI screens. The BT815 also contains high-level

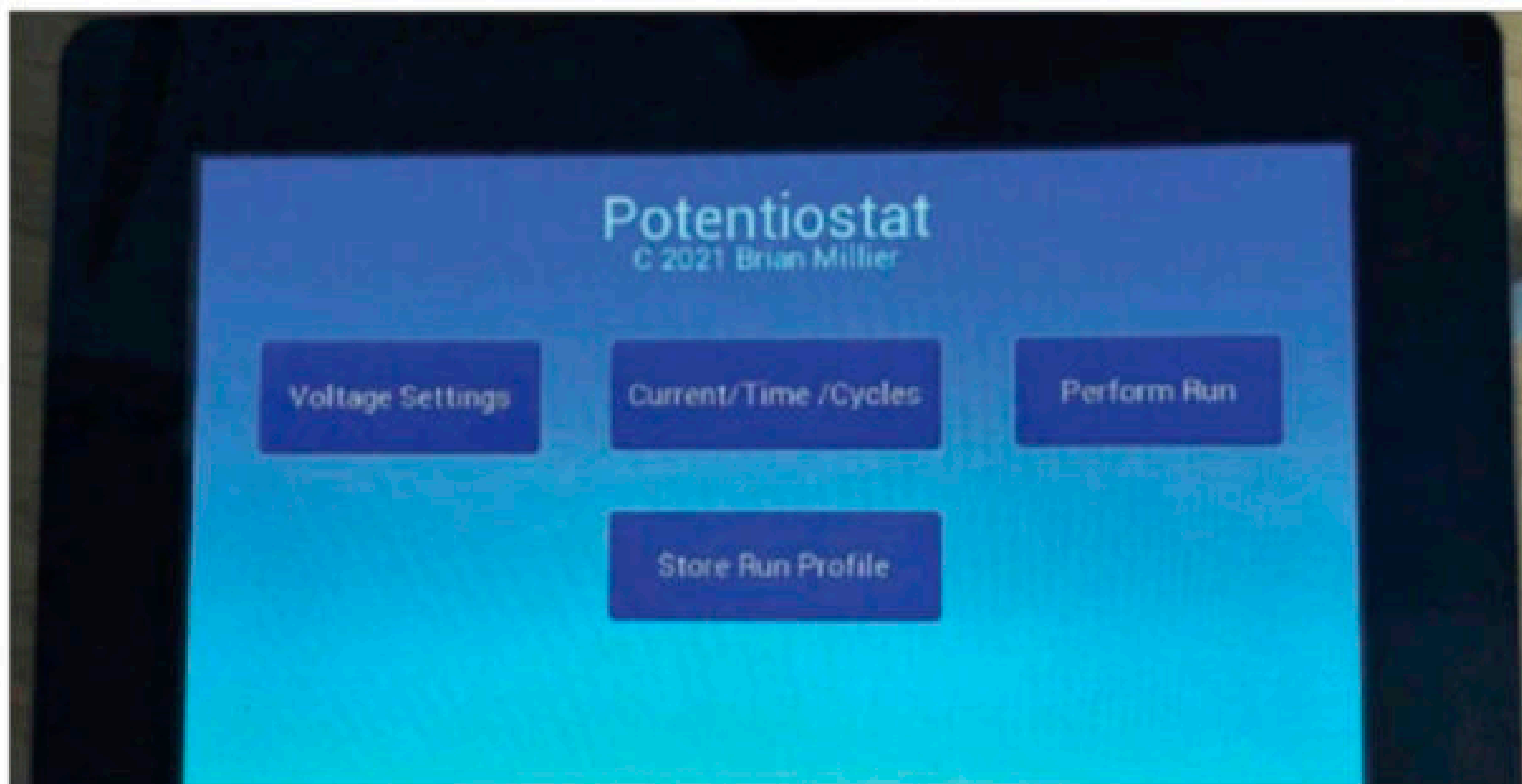


FIGURE 4A

This is the front view of a Matrix Orbital Eve3x-50G display, showing the main menu. As an example of the high-level graphics capability of the BT815 controller, the color gradient you see on the screen can be done using one BT815 instruction.



FIGURE 4B

This is the circuit board side of the Eve3x-50G display. Both the standard 20-pin ribbon cable header and FFC socket are at the left. The haptic motor and audio circuitry/speaker are at the middle right of the board.

hardware handling of touch events, which greatly simplifies the handling of touch events by the host MCU's code.

The BT815 contains the equivalent of a "draw-line" command (VERTEX) which can be used to draw a graph box and place axes markers, etc. The same command could also be used to plot the data points on the graph—assuming that all of the data points were available in advance.

In a potentiostat, however, the cell voltage is ramped up/down over time, and the current is constantly monitored. The cell voltage is plotted on the X-axis and the cell current is on the Y-axis. For measurements involving fast ramps, one could just wait until the complete ramp up/down was finished and perform one plot of the complete dataset. However, collection runs with very slow ramps are also common, and the operator will want to see the partial data as it is being collected. That way, if there is something wrong in the electrochemical cell configuration, they won't wait until the end of a possibly long cycle to see the problem.

In a conventional LED display controller, there would be a large block of RAM called the display buffer which contains all of the pixel data to be displayed. The display controller would constantly read sequentially through this buffer and transfer that data stream to the LED panel's pixel elements. To plot a graph in real time, all one needs to do is to write to a specific location in that RAM array, to turn a specific display pixel on/off. For a plot that is evolving in real-time, at any time you are only changing the specific RAM locations corresponding to the last data point received—the rest of the RAM array remains unchanged. Other parts of that array which would remain static would include the graph box, axes and labels.

The BT815's display list method is very different from the above. Standard high-level graphics widgets and primitives are used to render the screen, including the plotting of graphical data. Plots which evolve over time would normally require you to send a new display list to completely redraw the whole display screen, every time a new data point arrives. That is, clearing the screen, drawing a new graph box, inserting the axes and labels, and plotting the available data, point-by-point. The BT815 controller, coupled with a 16MHz SPI clock rate, is fast, but not nearly fast enough to perform all this at the fastest 50sps rate that I wanted to achieve in my design.

After several attempts at coding a routine to perform this real-time plotting, using the standard functions described in the BT815's programming manual, I wasn't able to achieve anywhere near the screen refresh rate that I wanted. Finally, I came up with a scheme that worked beautifully at the fastest data acquisition rates expected in the project.

BT815 REAL-TIME PLOTTING "TRICK"

While the BT815 uses a display list methodology, and a relatively small 8k FIFO buffer memory to store the display list, it also contains 1024k of general purpose RAM labeled RAM_G. This RAM memory is normally meant to hold the following:

- Images—either un-compressed bitmaps or JPEG, PNG types
- Fonts—over and above the ROM fonts contained in the BT815 (that is, larger or more specialized fonts)
- Sound wave files—for use with the sound-synthesizer section of the BT815

PCBWay

Advanced PCB Fabrication



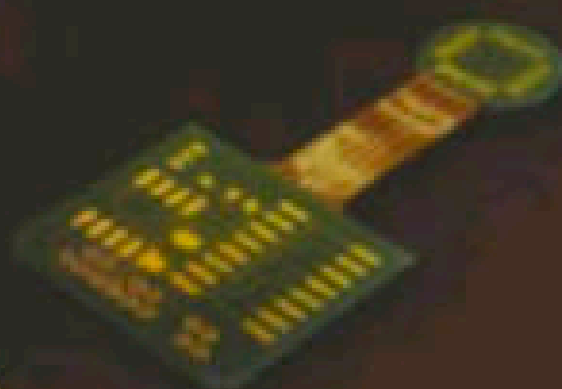
Instant quote online & User-friendly web interface
Fast turnaround in 24 hours
Perfectly implement your idea with different types of PCB!

High-Quality PCB Assembly



Turnkey PCB Assembly Service
Start from only \$30
Free stencil & Free shipping all over the world

15% off for Flex PCB



Professional Flex/Rigid-Flex PCB.
Quality assurance with the inspection and test report
Make high-end products the easy way

3D printing & CNC machining service is also available now!



More information:
www.PCBWay.com

Phone: (0571) 8531 7603

Email: service@pcbway.com

```

void putPixel(int x1, int y1) {
    int graphOffset;
    uint8_t rem;
    uint8_t l;
    uint8_t tmp;

    graphOffset = x1 >> 3; // div by 8 as 8 pixels per
byte = number of bytes across the full width of graph
    rem = x1 % 8;
    l = 128 >> rem;
    graphOffset += y1 * GraphX>>3;
    tmp = rd8(graphOffset);
    l = l | tmp;
    wr8(graphOffset, l);
}

```

LISTING 1

This is the snippet of code that converts a pixel with co-ordinates X,Y into a pointer to the BT815's RAM_G bit-map memory, and modifies that byte to light the pixel.

In the case of images, these image arrays would normally be loaded into RAM_G from either the Host MCU (program flash or SD card) or from a Flash memory chip located on the Eve3-50G board itself (this is a purchase option that can add a 32 or 256 megabit Flash device to the board).

The “trick” that I eventually came up with involves setting up a display list which contains instructions and parameters to display a bitmap with dimensions equal to the X and Y sizes of the rectangle enclosing the plot. I then define a bitmap header which contains these X,Y dimensions, the type of bitmap, and a pointer to the position in RAM_G memory that contains the bitmap image. The type of bitmap is “L1”, which is black/white and maps eight pixels (horizontally) into 1 byte of memory. Prior to collecting any data, I issue a CMD_MEMZERO command with the proper parameters to zero out the area of RAM_G that is used for this bitmap. Thereafter, as data points arrive, this data is incrementally plotted by manipulating bytes in this bitmap array, as needed to form the required plot trace.

Also included in this display list are a series of high-level commands or widgets which:

- Draw the rectangle defining the graph area on the screen
- Draw axes ticks and descriptive labels
- Insert Min, Max value labels for the axes
- Provide an “Exit” button to halt the data collection
- Provide any necessary status message(s)

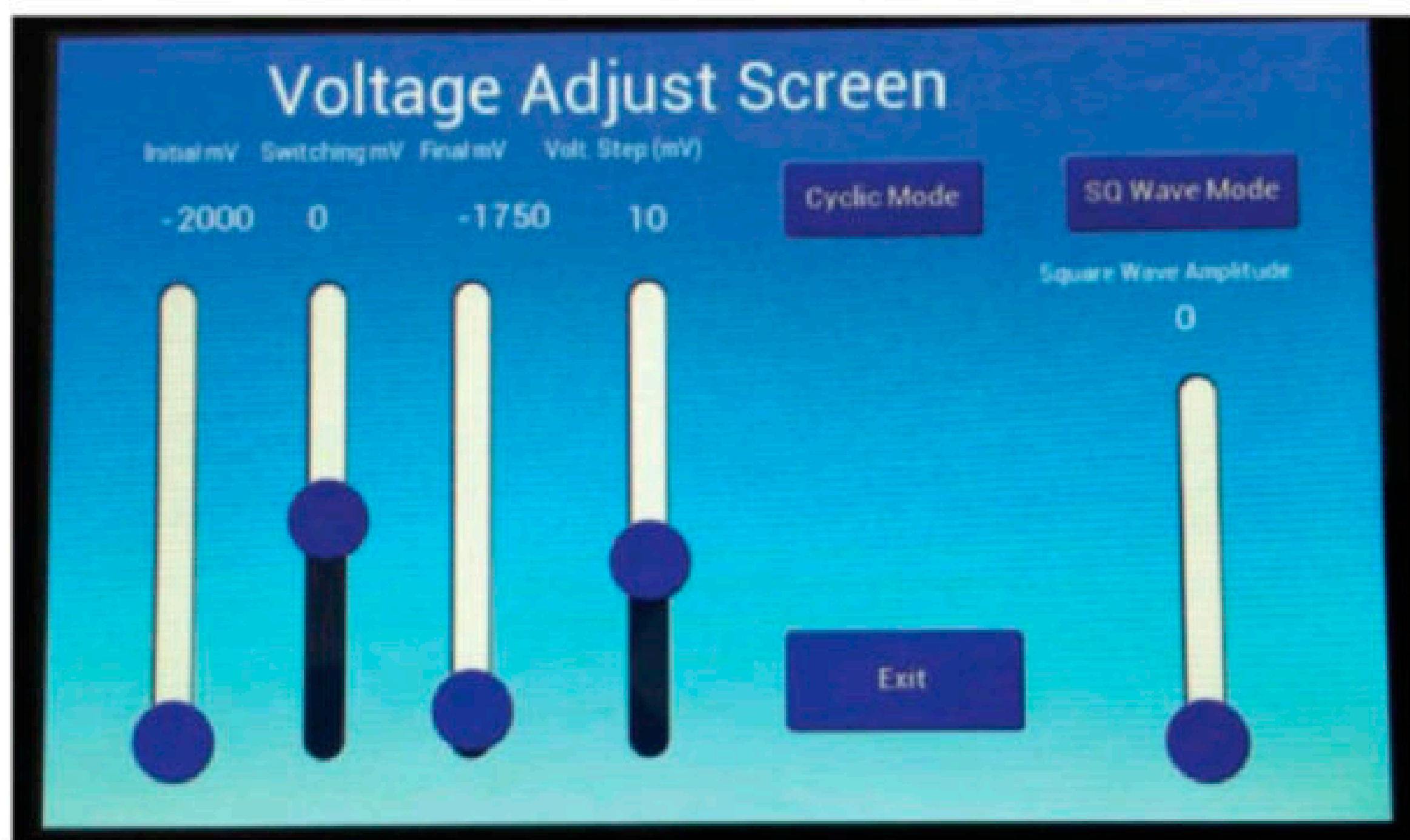
At the start of the run, the display list described above is sent to the BT815 controller. Once sent, it will continuously execute that list, at the display's refresh rate—giving what would normally be a static image.

What I discovered however, is that the host MCU can issue instructions to write data directly to the RAM_G area assigned to the bitmap. Whatever changes are made to this RAM_G memory are seen in the graph area of the screen, on the next screen refresh.

Therefore, as data comes in, I calculate a line connecting the newest data point with the previous data point. I use Bresenham's algorithm to break this line up into a number of X,Y points. I then do a calculation to map each point's X,Y coordinate values into the proper bitmap memory location in RAM_G. Due to the nature of a potentiostat run, the X value of a data point will be incrementing/decrementing slowly as the ramp progresses. The Y value of the data point (cell-current) will also involve only gradual changes. Therefore, for each new data point, there will be only a short line segment needed, and very few RAM_G memory locations will need to be modified. Even if there were large value changes from data point to data point, transferring those new memory values to RAM_G is done very efficiently using the BT815's memory read/write commands, coupled with the 16MHz SPI bus speed.

The routine that does the above is a routine labeled FT81x_DG(parameter list) and is located in my BT81x.cpp library file. The routine which takes two adjacent data points and converts them into points on the graph is called BresenhamLine(parameter list). The routine which maps a point's X,Y coordinate into the bitmap RAM_G memory is putPixel(parameter list). Both of these routines are in the main “C” program file.

Listing 1 is the putPixel routine which maps the X,Y coordinates of a point on the graph into the bitmap array in RAM_G. It is quite simple since the L1 bitmap format maps eight horizontal pixels into 1 byte in memory. Also, I chose the graph's X dimension to be 640, which is evenly divisible by eight. Since any given byte represents eight adjacent horizontal pixels, one has to read in the proper RAM_G memory byte (using the BT815's rd8

**FIGURE 5**

This is the screen which is used to enter all of the voltage-related parameters.

routine), or in the new pixel bit, and then write it back to RAM_G (using wr8).

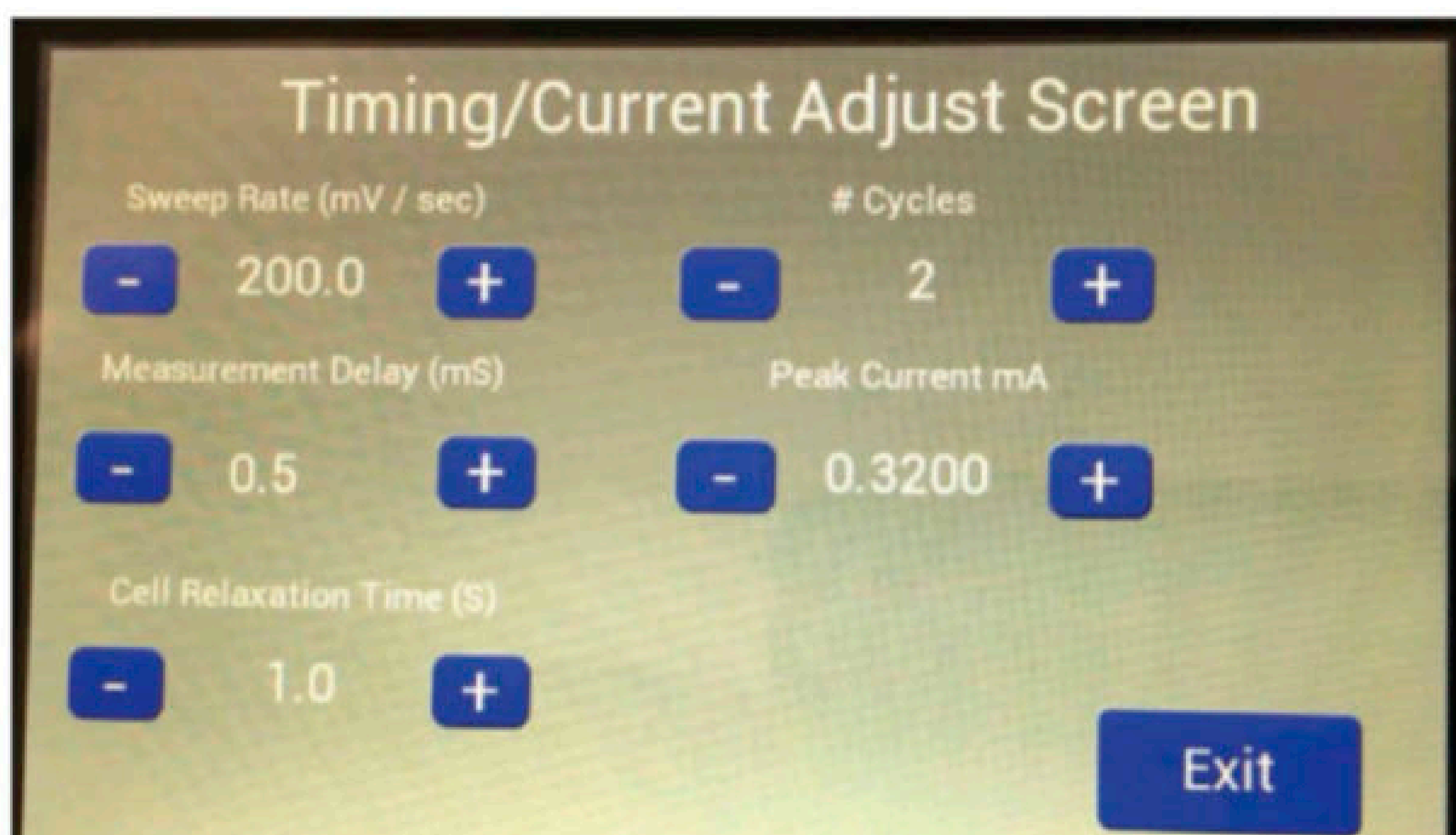
Using my scheme allows one to get all of the advantages of the BT815's display list architecture and high-level widgets, while utilizing the much faster and more efficient graphics frame memory buffer concept for the plotting routine. I'll also mention that the 8kb size of the BT815's display list memory, while sufficient for lots of high-level graphics commands and widgets, is not big enough to allow you to plot more than a few hundred data points—using the BT815's lower-level Vertex commands (the equivalent of draw-line commands).

TOUCH SCREEN OPERATIONS

There are quite a few parameters that are needed to define a potentiostat data collection run. The potentiostats that I built in the past all interfaced to a PC computer. I wrote a Visual Basic application which simplified entering these parameters using a GUI screen that contained text box fields for each parameter.

The 5" Eve3x-50G display has a high resolution of 800 x 480 pixels, and the capacitive touch screen is very accurate and sensitive to user touch. However, I had to separate the parameter entry process into two separate screens to allow the necessary controls to be large enough for easy manipulation. I broke the run configuration process up into two screens:

- **Voltage Screen (Figure 5).** This includes sliders to select starting, switching and final voltages. If you don't activate the Cyclic Mode button, you only have to specify start and final voltages, and the run will consist of just one cycle. Otherwise, you must also specify a switching value and the run will consist of multiple cycles (the number of cycles are set on a separate screen). You also specify the voltage step that you want between taking current readings. Finally, if you want to superimpose a square wave onto the voltage ramp, you can select the SQ Wave Mode button, and enter the square wave's amplitude using a slider.
- **Current/Time Screen (Figure 6).** This screen allows you to select the speed of the voltage ramp in mV/s. The measurement delay is the time in milliseconds between the setting of the DAC at a new voltage value and the triggering of the ADC to measure the cell current value. Cell relaxation time is the time between the closing of the TIA input relay (K1) and the start of the ramp. This happens at the beginning of each cycle of the run. The number of cycles is set on this screen using the +, - buttons.



The full-scale current setting for the TIA/ADC combination is also selected on this screen and ranges between 0.01mA and 10.24mA in 9 separate ranges.

For some of the above settings, you are selecting from a limited, achievable list of values. For others, there are limits defined. However, there are combinations of parameters which could result in "impossible" run configurations. After using both screens to set the parameters, you must return to the Main Menu screen and hit the "Store Run Profile" button. This will result in a check of all the parameters as they interact with each other, and an error message will be issued if there are problems. Assuming things are OK, they will be stored as a data structure in the Teensy 3.5's EEPROM memory. That is, they will remain in force even after the power is cycled off/on.

FIGURE 6

This is the screen which is used to enter all of the timing and cycle-related parameters. Because the time parameters can vary over a wide range, a pre-defined array of values is included in the program, and the + and - buttons cycle through these choices.

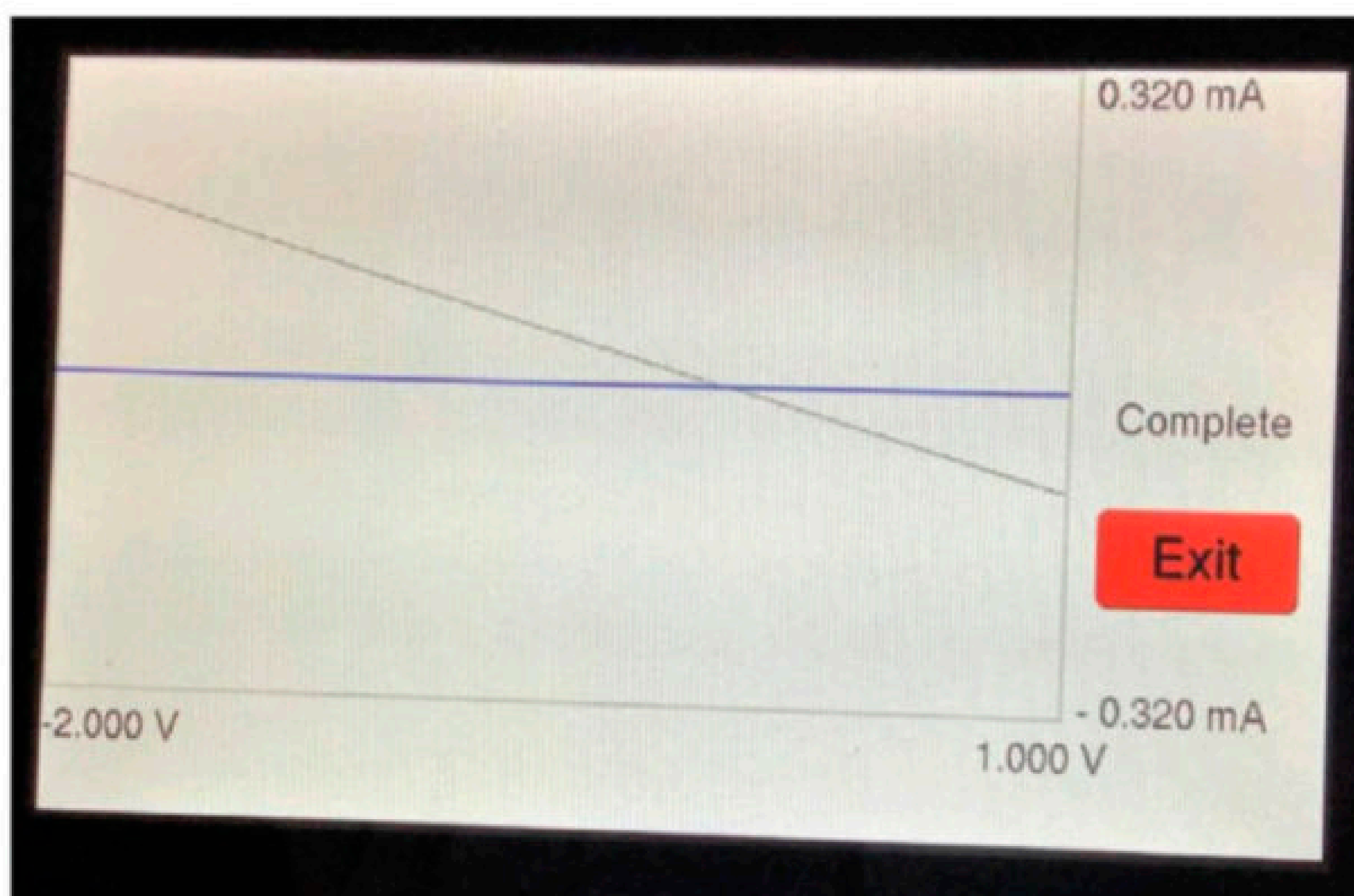


FIGURE 7

This is a photo of the display when it has completed a cycle. The faint outline of square blocks in the photo are not visible to the eye, in actual use.

Figure 7 is a photo of the potentiostat screen after performing a run. Due to COVID-19, the teaching lab that I work with is closed, and I did not have access to a real electrochemical cell. So, I used a resistor to simulate the cell, for this run.

One valuable feature of the Bridgetek BT815 display controller is its high-level handling of touches and gestures on the thin-film transistor (TFT) display screen. With resistive touch screen controllers, you generally get an indication when the screen has been touched or released, and an X,Y coordinate where the touch occurred. You would have to interpret this coordinate to determine what button was pressed. Evaluating what the user has done with a slider, for example, is a more complicated software routine.

The Eve3x-50G display contains a capacitive touch screen, and a Goodix GT911 touchscreen controller. It's not necessary for a programmer using a Eve3x-50G display to have any knowledge of the operation of the GT911. Rather, when the display is initialized (using the `FT81x_Init` routine), a block of data is loaded from the host MCU's flash memory to the BT815 controller chip. This block of data is not described in Matrix Orbital's documentation or as comments in the driver library they supply. Whatever it does, it allows the BT815 to communicate with the GT911 touch controller, and it transfers touch status and X,Y coordinates to the BT815's touch screen registers.

There are two main ways to get information about whether and where the screen has been touched:

- Low-level access to the touch screen registers—If you read the 32-bit register `REG_CTOUCH_TOUCH_XY` it will return the X coordinate of the touch in the MS 16-bits and the Y coordinate in the LS 16-bits. A 32-bit value of `0x80008000` is returned when the screen is not being touched.
- Higher-level commands—When you place any widgets on the screen, you would normally use the TAG instruction ahead of the command that places that widget. A parameter of that TAG command is a single character/byte that you wish to associate with that widget.

For example, the following code will place an "exit" button on the screen and assign it a TAG value of 'e':

```
Send_CMD(TAG('e'));
// assign TAG ID for following widget(s)
Cmd_Button(444, 366, 141, 59, 27, 0, "Exit");
```

If you then read the 8-bit register `REG_TOUCH_TAG` it will return a value of `0x65` (ASCII value of 'e') whenever that button is touched. The valid TAG IDs are byte values from 1 to 255, and a zero is returned if the screen has not been touched.

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

RESOURCES

adafruit | www.adafruit.com

Maxim Interated | www.maximintegrated.com

Matrix Orbital | www.matrixorbital.com

Texas Instruments | www.ti.com

Reading sliders is only a bit more complicated. For example, you can place a slider on the screen as follows:

```
Send_CMD(TAG('F'));
Cmd_Slider(243, 164, 24, 265, 0, vf, 200);
Cmd_Track(243, 164, 24, 265, 'F');
```

This tags the slider with the TAG ID 'F,' sets its default value at `vf` and its range as 0-200. The `Cmd_TRACK` statement defines the area of the screen occupied by the Slider, tracks any touches in that area and assigns them to the TAG 'F.' To read the value of a slider at any time, the following statement is used:

```
uint32_t tracker = rd32(REG_TRACKER + RAM_REG);
tracker = tracker >> 16;
```

The tracker value is a 16-bit value scaled such that the minimum slider value is 0 and the maximum slider value is 65,535 (`0xFFFF`). This returned value has the same range regardless of what you define to be the range in the `Cmd_Slider` command. Therefore you will have to scale this value to map it into the range that you want the slider to operate within. This is somewhat inconsistent, since you specify a "real world" range using a parameter of the slider command and assign it a default value within this range. However, subsequent monitoring of the slider returns a 16-bit value that must be scaled.

TOUCH SCREEN CALIBRATION

It's not emphasized in the BT815 controller's Programming Guide that you must calibrate the touch screen prior to using it. Fortunately, Bridgetek provides the `Cmd_Calibrate()` command which handles most of the process for you. This command places three target dots on the screen, one at a time, which you have to press. That results in the BT815 controller generating a six-member transform array containing the calibration constants. There is some additional code required to use this command, and this can be found in the `MakeScreen_Calibrate()` function, found in my `BT81x.cpp` file.

You could run this routine at every startup, but that would be inconvenient. Therefore, I save these six 32-bit values to the Teensy 3.5's EEPROM memory. At startup, I call the `loadTouchMatrix()` function, which reads this EEPROM array. If the six 32-bit values are all `0xffffffff`, that indicates that the EEPROM has not yet been written to, and I therefore call the `MakeScreenCalibrate()` function. Otherwise, I take these six values and transfer them to the BT815 at the memory location reserved for these values:

```
REG_TOUCH_TRANSFORM_A + RAM_REG;
```

The Eve3x-50's capacitive touch screen is very stable—once you have performed this calibration, you won't have to do it again. This may not be true if you were operating the display over a wide temperature range, but I have not tried this. What I have tried, by accident, is swapping between the Eve3-50G and Eve3X-50G displays without remembering to do a re-calibration. In this case, the touch screen appears to be non-responsive, as the transform array values are definitely way off.

Here, I have covered a few relevant topics regarding BT815 controller programming. A trade-off for its powerful graphics capabilities is the fact that its architecture and command set are rather unusual. If you are considering using the Eve3 or Eve3x series of displays, I'd recommend that you refer to my two-part article in *Circuit Cellar* issues #382 and #384, as I previously mentioned.

SD CARD FILE SYSTEM

All of my earlier potentiostat designs were interfaced to a PC computer via a serial port. Therefore the run data, which could be large if the cycle count was high, could simply be stored on the PC's hard disk. For this stand-alone unit I had to provide some form of permanent/removable storage. The Teensy 3.5, 3.6 and 4.1 modules all contain a uSD socket which is connected to the MCU's SDIO port. This provides much quicker access than you can achieve if you were to use the SPI port, which SD cards also support.

All of the Teensy modules are meant to be programmed in C++ using the Arduino IDE. There are a number of libraries available to handle SD cards. The early libraries were targeted for the 8-bit Atmel AVR MCUs that were mounted on Arduino boards. When the Arduino IDE was extended to work with 32-bit ARM MCUs, these libraries were ported to the more powerful ARM MCUs, with much the same functionality. That is, they only handled 8.3 DOS format file names, and did not work with newer high capacity (HC) SD cards.

Today there are more powerful SD card libraries available for the 32-bit ARM MCUs. I am using the SdFat library (written by Bill Greiman) which is now included with the latest version of Teensyduino (V1.56). This library works with the fast SDIO port that Teensy 3.5 uses for its SD card socket. It allows for long filenames—it's not limited to the 8.3 DOS format. It handles FAT16, FAT32 and exFAT disk formatting.

Midway through the `setup()` function, I call the following function: `SD.sdfs.begin(SdioConfig(FIFO_SDIO))`.

This routine will return a false value if a formatted SD card is not detected. The display will show an "SD Card not inserted" message, and then go into an endless loop. Power must be cycled off and a card inserted, to proceed.


The run data that is collected is stored in the Teensy 3.5's RAM memory until a cycle is complete. It is then sent to the SD card as a text file with a filename of `Cyclexxx.txt`, where xxx is a leading zero-padded cycle number. The data format is a simple comma-separated table:

Voltage (mV), current (mA)

I do not include the user-specified run parameters in this file.

Since I am using the on-board uSD Card socket, it wasn't possible to provide access to a uSD card from the external enclosure. To get around this issue, I bought a flexible uSD extension cable. **Figure 8** shows the potentiostat circuit board with the flex extension cable attached to the Teensy 3.5's uSD socket.

CONCLUSIONS

The Eve3x-50G display is the first reasonably-priced TFT touch screen display that is usable for instruments that require a graphics display of data in any significant amount or resolution. Its capacitive touchscreen and 5" size also make it relatively easy to provide a GUI which allows for easy entry of parameters, and so on. Since I obtained the newer Eve3x-50G display after the project was substantially finished, I did not make use of the Eve3x-50G haptic feedback or its audio synthesizer/speaker. There are occasions, when using a touch screen display, where it would be better to use haptic feedback rather than waiting until the user stopped touching the screen, before proceeding. Alternately, a short "beep" from the speaker would help in this respect. 

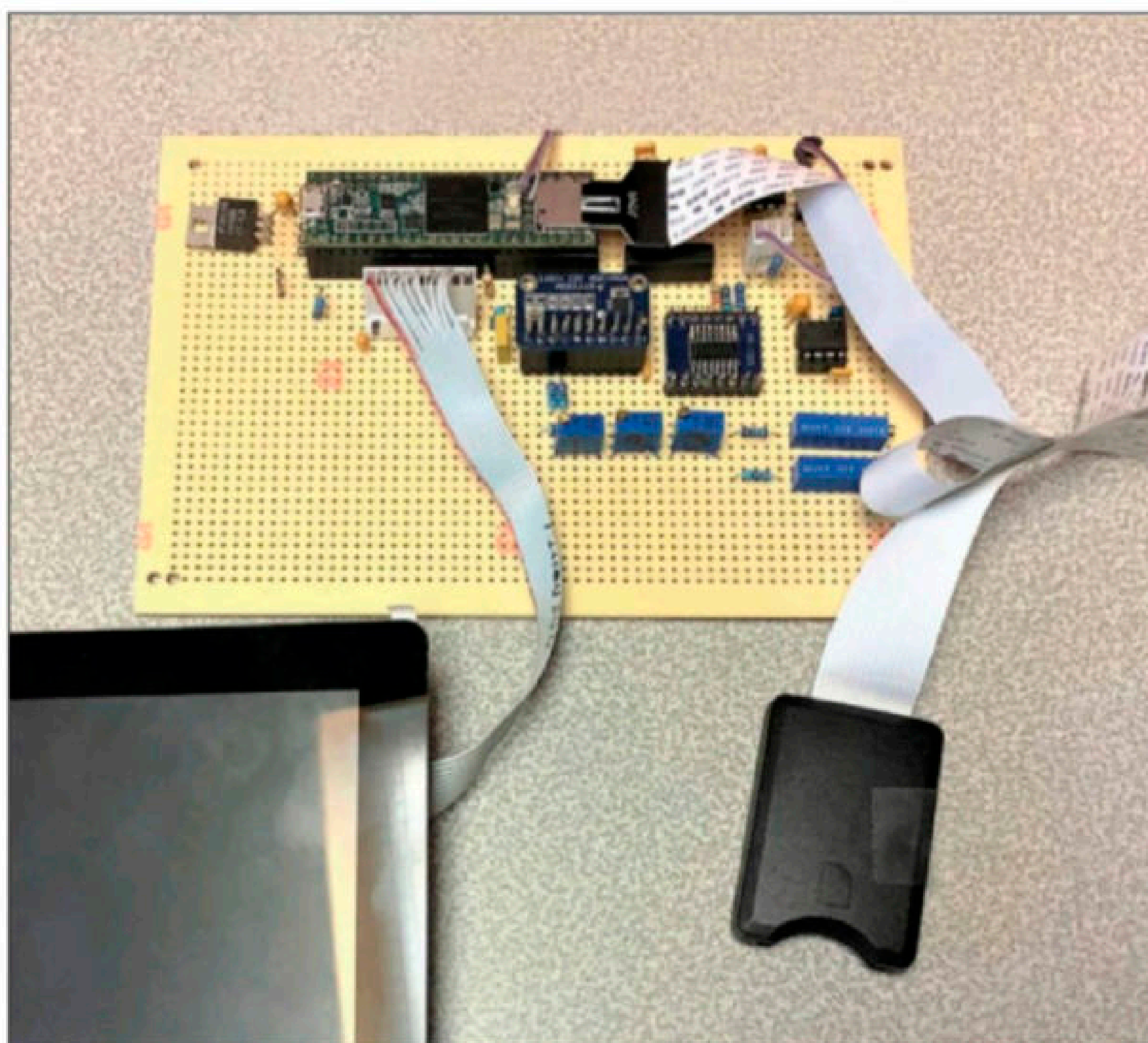


FIGURE 8

This is the display and the unit's circuit board. The Teensy 3.5's uSD socket has a flex extension cable plugged into it to extend the socket so it can be accessed from the outside of the unit's enclosure. The cable here is much longer than needed—about 18"—but my first choice of a shorter cable was defective when it arrived.

Embedded System Essentials

Keeping Your Memories Secret

SRAM Read-Back Attacks

By
Colin O'Flynn

In this article, I demonstrate attacks which rely on reading back SRAM from a microcontroller (MCU). These attacks occur because many debug lock or security features in MCUs allow read-back of SRAM, even when code memory is secure. Taking advantage of the known structure of certain features such as the Advanced Encryption Standard (AES) key schedule allows an attacker to easily detect where these sensitive keys are stored. We can complicate these attacks by clearing or obfuscating memory, and this article gives some practical demonstrations of both attacks and countermeasures.

AES is the most likely symmetric security algorithm you'll be using on your embedded system. It's fast, secure, and code-efficient. Even if your application doesn't use it, you might also find it used in a bootloader or other part of your system.

In this article I'm going to show you how an attacker could recover your AES keys by taking advantage of the fact that many microcontrollers (MCUs) don't correctly protect SRAM access, even when the MCUs do protect FLASH access.

In practice, this means an attacker may be able to attach a debugger, dump a snapshot of your device SRAM, and then run a simple script to detect AES keys. If the attacker has more knowledge of your device internals they may be able to extract even more secrets.

It's also something you can easily test yourself—it only requires access to a debugger, which you're likely already using as part of your embedded development. This makes it a very accessible test you can do on your own products. There are also simple countermeasures you can apply, which mostly concentrate around zeroing out critical data, and storing the critical data in an obfuscated format.

In previous articles, I've shown you how to break AES with power analysis and fault

injection. I've also talked about both hardware AES (such as that implemented by an MCU peripheral) and software AES (implemented in your firmware or a library). Power analysis and fault injection are much more advanced attacks than what I'm going to show you here, but I recently had a chance to use this "classic" attack, and it was a good reminder that it's important to cover the basics as well as the more advanced attacks.

This attack has two parts: the first is that an attacker can read SRAM from an otherwise secure device using a couple simple techniques. The second is that the structure of how most AES implementations store the key allows it to be easily detected in memory. I'll summarize those two points first before we get into the real-life examples and work.

READING MEMORY FROM LOCKED DEVICES

If you're using a secret, such as an encryption key, in your device firmware, you almost certainly know that at minimum you should enable the vendor-provided firmware protection. This is typically called something like "code read protection" or "debug lock."

There are two related problems with the typical debug lock. The first is that in some devices the debug lock only prevents read

access from the FLASH memory which holds the program code. Even with the device locked, an attached debugger can read the SRAM memory. An example of that is the STM32F1 series of devices—these older devices have only a single “level” of code protection, as described in ST Programming Manual PM0075.

More recent devices often have a way of disabling SRAM access as an option. For the ST devices, application note AN5156 provides an overview of which devices offer specific security features. Here we can see one of the RAM banks can be protected on a more recent part such as the STM32F4, so if we are storing sensitive secrets in that RAM bank an attacker cannot read them out like they could with a STM32F1.

In this case we rely on a more destructive attack: we disable the code read protection, and then try to read the SRAM. On many devices the code read protection can be disabled with a device erase. The attack works because sometimes the device erases only the flash memory, and not the SRAM. While this “kills” our target, it allows an attacker the ability to get a copy of the SRAM at a specific moment in time.

Note that since the attacker needs to kill a device to get a copy of the SRAM, if they aren't sure when the sensitive data is stored in memory, it becomes more complex and expensive to automatically try multiple times.

Some devices have features that help you counteract this attack. The two main things to check for are seeing if the device will erase SRAM at the same time it erases FLASH memory, and seeing if the device clears SRAM at boot. When devices support this feature it may only be a specific portion of the SRAM that it applies to.

For example, the ST describes in reference manual RM0090 how the STM32F415 treats the special “backup SRAM” memory. This backup SRAM memory *is* protected against read-out when the flash read protection is enabled, and this memory section *is* erased when you disable code read-out protection. While the backup SRAM is designed to be used with a battery for storage when power is off, you could use it like any other SRAM section even without the backup battery feature. A smart choice would be to store all secrets to this SRAM, by defining a memory section and ensuring your linker script puts all of the AES state (along with similar sensitive data) into this memory section.

DOWNGRADE ATTACKS ON CODE READ PROTECTION

While you're investigating the manufacture claims for SRAM read protection, you should also consider if there might be attacks that

```
#include "hal.h"
#include <stdint.h>
#include <stdlib.h>

volatile uint8_t test_value[] =
{0xFE,0xED,0xFA,0xCE,0xCA,0xFE,0xBE,0xEF};

int main(void)
{
    platform_init();
    init_uart();

    while(1) {
        test_value[0] = test_value[0];
    }
}
```

LISTING 1

This basic C code uses the ChipWhisperer Hardware Abstraction Layer (HAL) to compile this simple code of a variety of target MCUs.

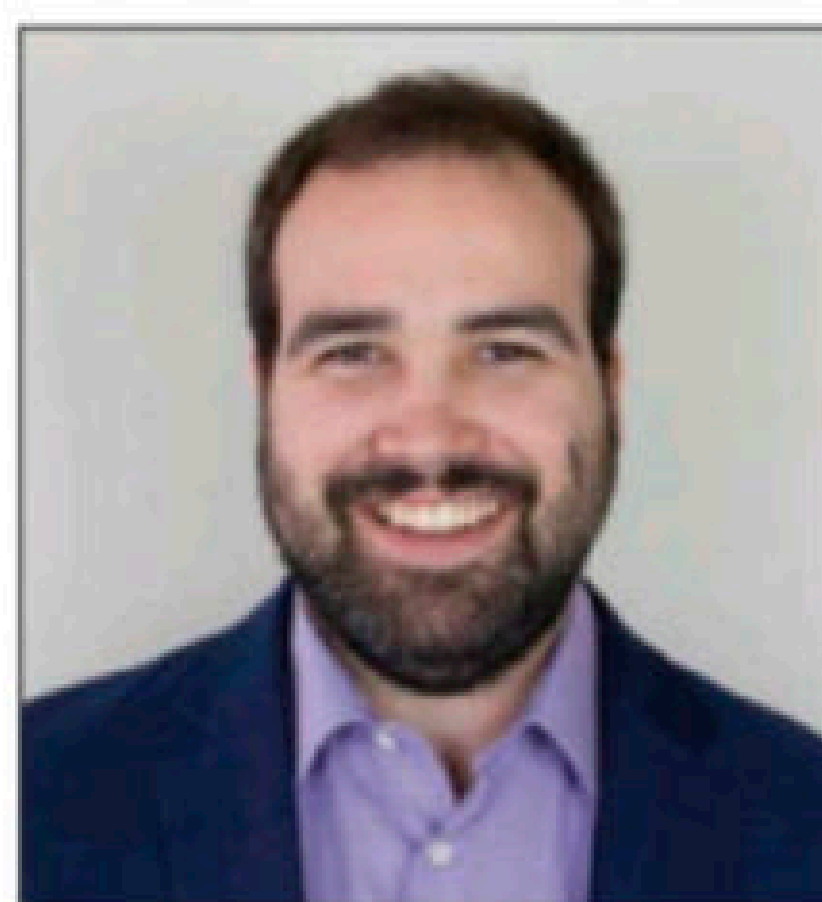
reduce the claimed security levels. Devices from different manufactures have proven to be vulnerable to various types of “downgrade attacks.” As an example, some of the STM32F devices I mentioned can be glitched from the debug level which has the SRAM access disabled (RDP2 in the ST world), to a lower level (RDP1 in the ST world).

While the ST documentation claims such change is impossible, this attack has been widely demonstrated in practical scenarios. One detailed example is given in the blog post “Kraken Identifies Critical Flaw in Trezor Hardware Wallets,” which uses an STM32F215. In fact, this also shows how you can further read flash memory once in RDP1 level, which I similarly demonstrated in my article “Revisiting Code Readout Protection Claims” in *Circuit Cellar* July 2022 Issue #384.

Such attacks are more difficult than those using just a debugger, but may still be in scope for your threat model, so I want to make sure you're aware of the broader picture. But for now, let's look at how you can evaluate devices with only a debugger attached.

TESTING YOUR OWN DEVICES

One thing I promised at the start of this article is how easily you can test your



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.

```
*(.data)
.data          0x0000000020000000      0x8 objdir-CW308_STM32F0/simpleserial-base.o
              0x0000000020000000      test_value
```

LISTING 2

A MAP file created by our toolchain lets us see where the symbol was placed in RAM.

own devices for the sort of simple SRAM read-out. To demonstrate this, I'll use an STM32F0 "target" device that is part of my ChipWhisperer platform. You could use any sort of development board or other board you have with your target device.

To do this, I compiled the simple code from **Listing 1**. This code places some known byte sequence (FE ED FA CE CA FE BE EF) into SRAM, which we can confirm by looking at the MAP file, as shown in **Listing 2**. Here it shows this object was placed at address 0x20000000

(the start of SRAM). Watch to ensure the toolchain doesn't remove the object once it detects that it's never used—simply declaring it as volatile may be enough to keep it in the object file during compilation, but it may still be removed by the linker. In this case I've purposely accessed one element of the array, which caused the entire array to stick around.

To test the device, I programmed this binary file into my target device. I then connected with a debugger and confirmed that, as expected, my byte sequence is shown in SRAM memory at the expected location. This step confirms that, without a doubt, the data has been placed into memory at the correct location.

Finally, we use our device programmer to "secure" the device. I was using Segger J-Flash, and it has a secure option that programs the device to RDP2. For the STM32 device you'll need to power cycle it for the setting to take effect. At this point, the device is running our code and placed our test value in SRAM at address 0x20000000. But the JTAG access is locked, preventing us from reading the flash memory.

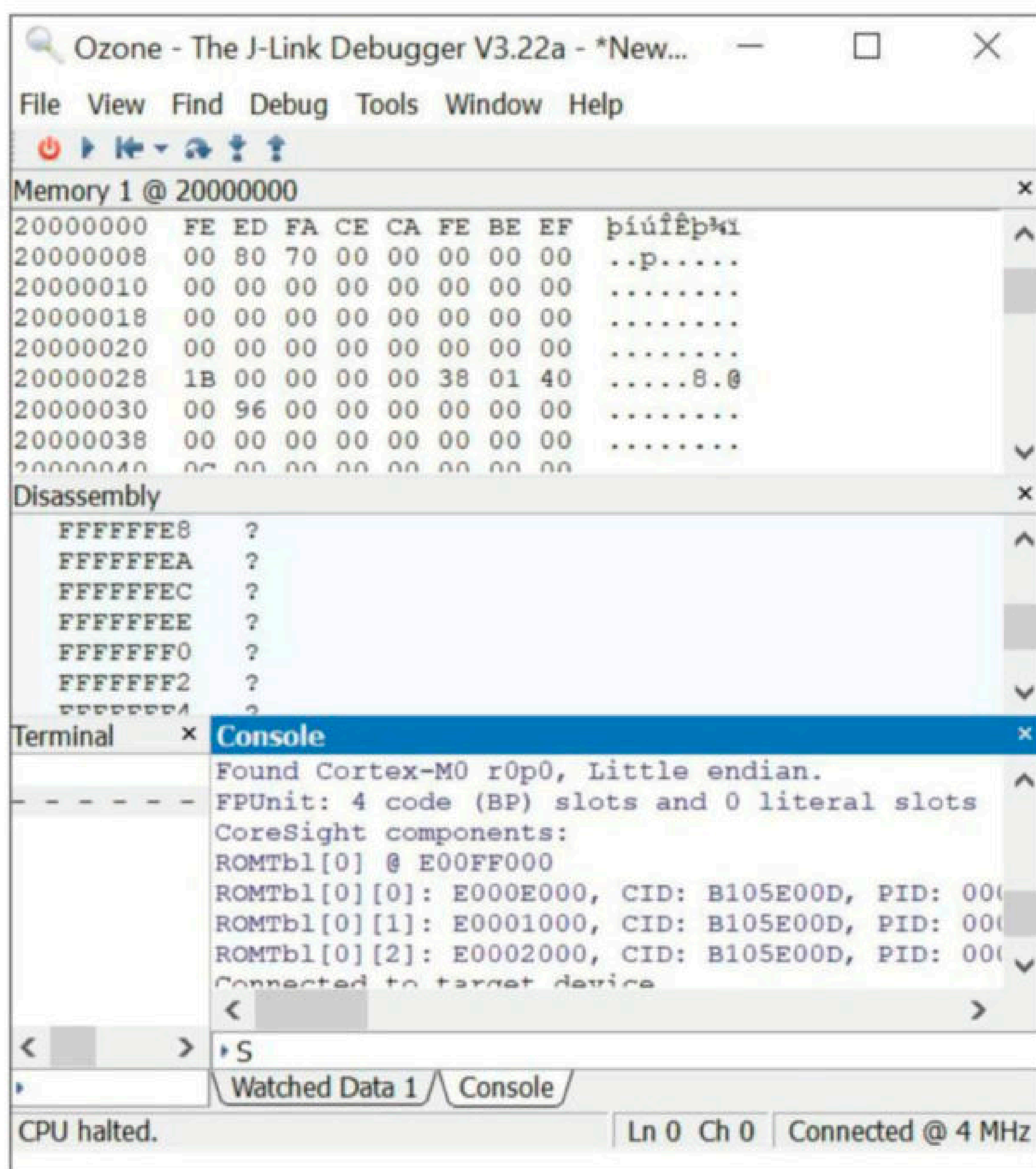
Next, we unlock the device, which will trigger a flash erase. Without power cycling it, we connect the debugger and read the SRAM memory. With any luck, it looks something like **Figure 1**. Here you can see I'm using Segger Ozone as a debugger, and the memory window shows the value FE ED FA CE CA FE BE EF at address 20000000. Sharp-eyed readers might notice that the window showing "Disassembly" is pointing to invalid code, since the code has been erased and the device has an invalid boot address.

But the important finding here is that by erasing the device and connecting a debugger, the general-purpose SRAM has not been cleared. Any sensitive data left in SRAM will be readable by an attacker.

Of course, there is no structure to this data. If an attacker has reverse engineered your firmware or has some inside knowledge they may know exact addresses that store your secrets, but let's look at the more generic case where an attacker has no such knowledge and needs to exploit the structure of the secret keys.

FINDING AES KEYS

We don't need to dive into the details of AES to understand how you can find its keys in memory. But you need to understand that

**FIGURE 1**

The Ozone Debugger shows the SRAM memory in the top window, with SRAM starting at address 0x20000000, and the secret value clearly visible.

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

RESOURCES

ST | www.st.com

AES takes 16 bytes of input and uses a secret key to generate 16 bytes of output. Depending on the type of AES, the key could be 16 or 32 bytes long (AES-192 could also be in use, which means a 24-byte key, but is almost never found in practice).

Depending on how AES is used, we use different "modes" of AES. The most basic form of AES is called Electronic Code Book (ECB), referred to as AES-ECB. In this mode encrypting the same input (plaintext) will always map to the same output (ciphertext). This means it's often insecure for many applications, since an attacker can see patterns in the encrypted ciphertext, even if they don't know the exact ciphertext-to-plaintext mapping.

The other aspect of AES-ECB is that we can decrypt a random chunk of 16-byte data, without decrypting the entire file. While there are better modes of encryption to use when random access is required (such as AES-XTS used for disk drive encryption), you still often find AES-ECB used in embedded systems. If your system uses AES-ECB, this also means you can perform a brute-force attempt to decrypt a piece of ciphertext using every possible 16-byte key from memory.

A simple example of doing this is shown in **Listing 3** using Python. You'll need some way to identify a successful decryption—this will depend on your system. If it's a firmware file for example, you may expect to find some specific strings in the decrypted file. So, you could look for a string that you know exists in the file, such as a debug message that says "System booted." I've done exactly that in my example from Listing 3.

One caveat here is that how the key is stored in memory will vary depending on the AES implementation. For this reason, I also do a brute-force test of the most common variations and their permutations: the key could be stored Most Significant Byte (MSB) first, or Least Significant Byte (LSB) first. Each 4-byte word (assuming a 32-bit system) could itself also be mirrored. This more complex search is shown in **Listing 4**.

Note both Listing 3 and Listing 4 assume a 16-byte key, meaning AES-128 is in use. You could switch this to use 32-byte keys and AES-256. Another change you might consider is searching with a 1-byte step size instead of the 16-byte step size. The 16-byte search step size is faster, but might miss the start of an array.

You can also eliminate unlikely keys—for example, in practice, large blocks of the SRAM will be 00, which you could skip.

If you know something about what the device is doing, such as AES for a standard communications interface, you can also

feed details of that to build your own model into which you feed the brute force key search. Remember to include the various permutations of how the key could be stored in memory!

But if the device is running AES in software,

```
from Crypto.Cipher import AES

def chunks(l, n):
    return [l[i:i+n] for i in range(0, len(l), n)]

sram = open("sram_dump.bin", "rb").read()
ciphertext = open("fw_update_file.bin", "rb").read()

keys = chunks(sram, 16)

for offset, k in enumerate(keys):
    cipher = AES.new(bytearray(k), AES.MODE_ECB)
    dec = cipher.decrypt(bytearray(ciphertext))

    if b'System' in dec:
        print(offset*16)
        print(k)
        break
```

LISTING 3

An example Python script to test all possible keys for decrypting a file

```
from Crypto.Cipher import AES

def chunks(l, n):
    return [l[i:i+n] for i in range(0, len(l), n)]

def word_reverse(k):
    return k[0:4][::-1] + k[4:8][::-1] + k[8:12][::-1] + k[12:16][::-1]

def byte_reverse(k):
    return k[::-1]

def test_result(ciphertext, k):
    cipher = AES.new(bytearray(k), AES.MODE_ECB)
    dec = cipher.decrypt(bytearray(ciphertext))
    if b"System" in dec:
        print(k)
        return True
    return False

sram = open("sram_dump.bin", "rb").read()
ct = open("fw_update_file.bin", "rb").read()
keys = chunks(sram, 16)

for offset, k in enumerate(keys):
    if test_result(ct, k):
        break
    if test_result(ct, byte_reverse(k)):
        break
    if test_result(ct, word_reverse(k)):
        break
```

LISTING 4

We don't know how the AES key is stored in memory, so try to capture several likely permutations.

you can also take advantage of the AES "key schedule." The key schedule is how the AES algorithm transforms the 16-byte key into a longer 176-byte sequence used by the full AES-128 algorithm (or converts a 32-byte key into a 240-byte sequence for AES-256).

To take advantage of that, we simply perform a search in memory by taking the 16-byte key, calculating the next 16 bytes that would be stored in memory, and seeing if we find that data in our device SRAM. An example of how you can implement that is shown in **Listing 5**.

```
def xor(s1, s2):
    return tuple(a^b for a,b in zip(s1, s2))

class AES(object):
    class __metaclass__(type):
        def __init__(cls, name, bases, classdict):
            cls.Gmul = {}
            for f in (0x02, 0x03, 0x0e, 0x0b, 0x0d, 0x09):
                cls.Gmul[f] = tuple(cls.gmul(f, x) for x in range(0,0x100))

Rcon = ( 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a )
Sbox = (
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
)

@staticmethod
def rot_word(word):
    return word[1:] + word[:1]

@staticmethod
def sub_word(word):
    return (AES.Sbox[b] for b in word)

def key_schedule(self):
    expanded = []
    expanded.extend(self.key)
    for i in range(self.nk, self.nb * (self.nr + 1)):
        t = expanded[(i-1)*4:i*4]
        if i % self.nk == 0:
            t = xor( AES.sub_word( AES.rot_word(t) ), (AES.Rcon[i // self.nk],0,0,0) )
        elif self.nk > 6 and i % self.nk == 4:
            t = AES.sub_word(t)
        expanded.extend( xor(t, expanded[(i-self.nk)*4:(i-self.nk+1)*4]) )
    return expanded

def add_round_key(self, rkey):
    for i, b in enumerate(rkey):
        self.state[i] ^= b

def sub_bytes(self):
    for i, b in enumerate(self.state):
        self.state[i] = AES.Sbox[b]
```

(CONTINUED)

LISTING 5

We can search for the AES key schedule in memory without having any knowledge of the AES mode or data being used.

This attack is powerful since we don't need to know anything about what the device does—if we can find an AES key schedule in memory, we know exactly what the AES key is, and can then try to figure out what it's used for.

HIDING YOUR SECRETS


What happens if your device is vulnerable to this type of attack? Obviously, the best solution would be to use a more secure device, but you likely have devices in the field, or are constrained by the ongoing supply chain problems and have no other stock.

But even with such a device, we can make improvements to our implementation. The first choice is to scrub data from memory when it's not needed. You might currently calculate the AES key schedule once and keep that in memory for example—a reasonable choice for speed reasons. But you've just made an attacker's life very easy, since they don't need to decide on a specific time to read SRAM. They can do it at any point in time. By clearing data immediately after you use it, you leave a much smaller window an attacker can exploit.

Of course *you* may not leave the key schedule around in memory, but a library you are relying on might. You can use the code from Listing 4 to see if you find AES key schedules in your own memory, and if you have source code for the library you can likely inspect it to see if it clears the key schedule after use.

An additional countermeasure is to obfuscate data in memory. This can be relatively simple yet effective—for example you could XOR the data read and written to an array with the index of the array element along with a constant. An example of this is shown in **Listing 6**. This would completely break my brute force attack in Listing 5.

You can also store data in unusual orders to make brute-force searches more complicated. Again, these countermeasures can be relatively easily undone if an attacker gets access to your code, but with only SRAM access it will take more effort to discover your specific byte ordering.

Hopefully this article has given you some additional topics to consider when attempting to secure your system. 

(LISTING 5—CONTINUED)

```
class AES_128(AES):
    def __init__(self):
        self.nb = 4
        self.nr = 10
        self.nk = 4

class AES_256(AES):
    def __init__(self):
        self.nb = 4
        self.nr = 14
        self.nk = 8

def word_reverse(k):
    return k[0:4][::-1] + k[4:8][::-1] + k[8:12][::-1] +
k[12:16][::-1]

def byte_reverse(k):
    return k[::-1]

def test_key_schedule(aes, key, rlsched):

    aes.key = key
    test_round_1 = aes.key_schedule()[16:32]
    rlsched = list(rlsched)

    if test_round_1 == rlsched:
        return True, key
    return False

if __name__ == "__main__":
    #Here is the main search program itself

    aes = AES_128()

    sram = open("sram_dump.bin", "rb").read()

    for o in range(0, len(sram)-31):

        #Round-1 key is +16 bytes from initial key
        kin = sram[o+0:o+16]
        rlschedin = sram[o+16:o+32]

        k = word_reverse(kin)
        rlsched = word_reverse(rlschedin)
        if test_key_schedule(aes, k, rlsched):
            print("Key FOUND == {}".format(k.hex()))

        k = kin
        rlsched = rlschedin
        if test_key_schedule(aes, k, rlsched):
            print("Key FOUND == {}".format(k.hex()))
```

```
static uint8_t[176] aes_state;

/* XOR means the same function used for hiding or unhiding the AES state */
void hide_or_unhide(void){
    for(int i = 0; i < 176; i++){
        aes_state[i] = aes_state[i] ^ 0xAB ^ i;
    }
}
```

LISTING 6

A very simple obfuscation XOR's the data with both a constant and a changing byte.

From the Bench

A Radar Speed Monitor

This Time Using Raspberry Pi Pico

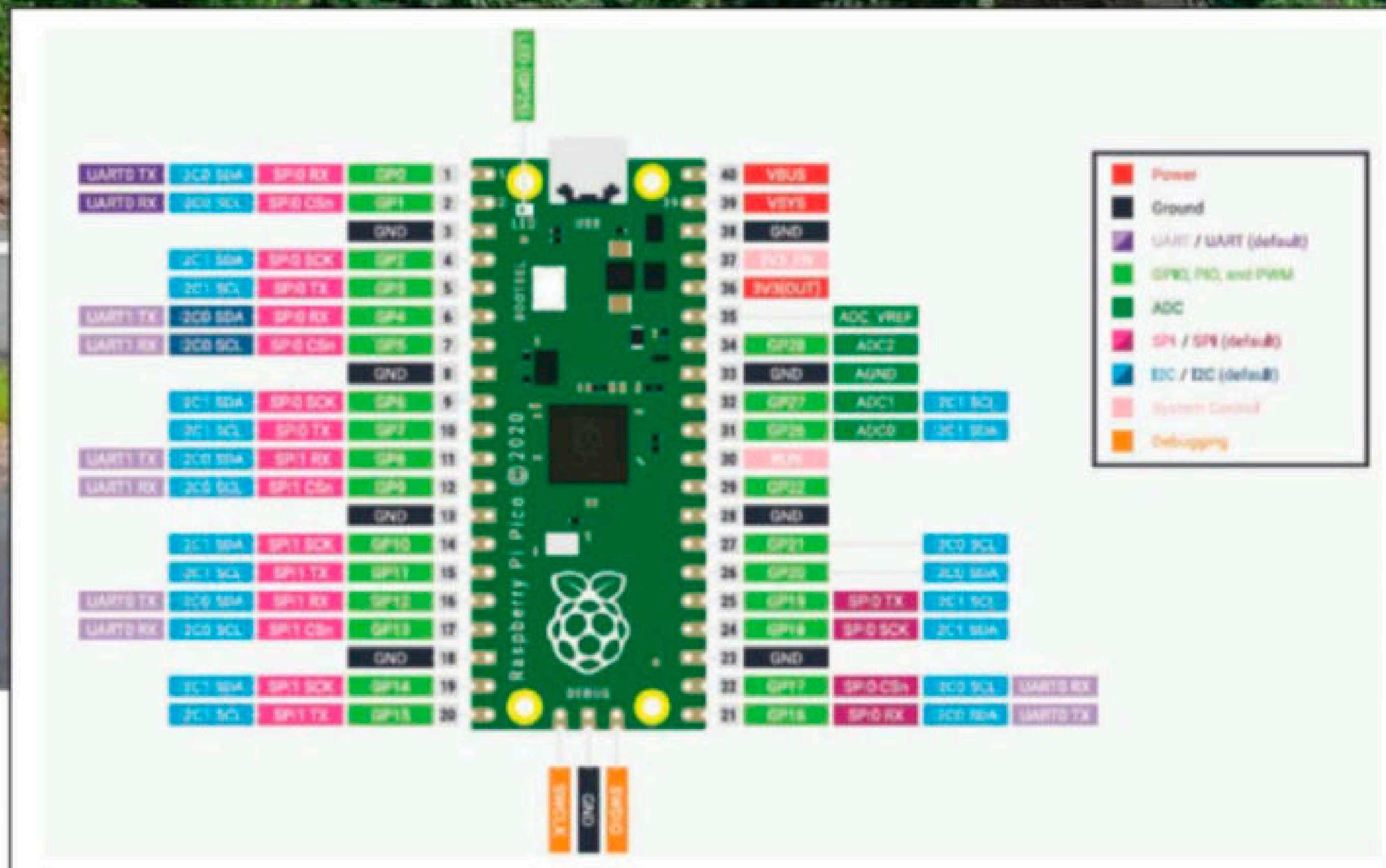


FIGURE 1

This is the \$4 Raspberry Pi Pico microcontroller (MCU) module. The 2x20 0.7" dip format is compact, yet brings out all of the IO to castellated 0.1" pads. It's a 3.3V device that can be powered by its on-board micro-USB connector.



By
Jeff Bachiochi

Who hasn't heard of the Raspberry Pi? As a follow-up to my previous article and a personal challenge, in this article I build another radar speed monitor, this time using the \$4 Raspberry Pi Pico.

I have to try something I've never done before. I'm going to replicate last month's project ("Just How Fast Was I Driving, Officer?," *Circuit Cellar* #389, December, 2022) [1], using a whole different microcontroller (MCU) and radar module—one that I haven't used before, the Raspberry Pi Pico (**Figure 1**). There was a lot of press about it a while ago because it was being sold for \$4. It was difficult to get your hands on one for quite some time. The supply has loosened up, but I haven't seen a lot of projects using it.

The Raspberry Pi Pico is an MCU board based on the Raspberry Pi RP2040 MCU chip (**Figure 2**). It was designed to be a low-cost, yet flexible development platform for RP2040. **Table 1** lists the key features and peripherals.

The Pico provides minimal external circuitry to support the RP2040 chip, comprising flash (Winbond W25Q16JV), crystal, power supplies and decoupling, and a USB connector. The majority of the RP2040 MCU pins are brought to the user IO pins on the left and right edges of the board. Note that four of the RP2040 IOs are used for internal functions: driving an LED, on-board Switched Mode Power Supply (SMPS) power control, and sensing the system voltages.

SPRECHEN SIE DEUTSCH?

Unlike other PI boards, the Pico board doesn't run a full operating system. You must embed your own application, written in either MicroPython or C. Since the release, there is also an installation for the Arduino IDE. I won't be using any of these—instead I'll be writing the app using an interpreter, MMBasic, which is a Microsoft BASIC-compatible implementation of the BASIC language, with floating point, integer and string variables, arrays, long variable names, and many other features.

When you plug in the Pico while holding down the BOOTSEL button, a virtual drive is created called "RPI-RP2" (the same as if you had plugged in a USB memory stick). Drop the `PicoMiteV5.07.04.uf2` file onto the drive, and the Pico will restart and automatically create a virtual serial port. The LED on the Raspberry Pi Pico will blink slowly, indicating that the PicoMite firmware with MMBasic is now running.

The emphasis with MMBasic is on ease of use and development. The development cycle is very fast, with the ability to switch instantly from edit to run. Errors are listed in plain English, and when an error occurs a single keystroke

will invoke the built-in editor, with the cursor positioned on the line that caused the error.

A note about the on-board editor: You also need a terminal emulator program running on your desktop computer. The terminal emulator should support VT100 emulation, since that is what the editor built into the PicoMite expects. There is more on this in the PicoMite user's manual [2].

For Windows users, it is recommended that you use Tera Term, which has a good VT100 emulator and is known to work with the XModem protocol—which you can use to transfer programs to and from the PicoMite. I'm most comfortable with a mouse-style editor, so I will not use this internal editor. Instead I will use MMEdit as an alternate tool, suggested by PicoMite authors Peter Mather, Geoff Graham, and Mick Ames [2]. MMEdit is a Windows editor that will allow you to edit, store, and retrieve your file, then automatically download your basic text file to Pico, save and run it. Seven programs, 1-7, can be saved to the Pico, and you can have any one of these autostart on power up. They can also be chained or called by others without losing variable contents.

Ctrl-C exits a running program to a command prompt. In Command mode ">" you can enter direct commands such as `memory`, and get some real-time feedback about the resources available (**Figure 3**). Arduino users will feel comfortable, because many commands will be familiar. Although the syntax is a bit different, the user manual will explain all!

Let's get into the first program, which will allow us to investigate a new radar module, the K-LD2 Radar Transceiver from RFbeam Microwave GmbH (St. Gallen, Switzerland). This unit is smaller than the Grove BGT24LTR11 I used in last month's project [1]. While the K-LD2 uses a UART communication interface, the format is a bit different. It has eight command classes that separate registers into groups (**Table 2**). The communication format is shown in **Table 3**.

A typical Read Command request/response might be:

```
$S06<CR>
$S0601<CR/LF>
```

A Write Command request/response might be:

```
$S0602<CR>
$S0602<CR/LF>
```

APPLICATION 1: INVESTIGATE K-LD2

Of the two applications written for this month's column, the first is by far more

complicated, because I want to allow users to play around with all the registers of the K-LD2, so they feel comfortable adjusting any default parameter that may not be the best selection for their application. The complications arise because a user needs to choose any register, and make adjustments and report the result using words, rather than just a value.

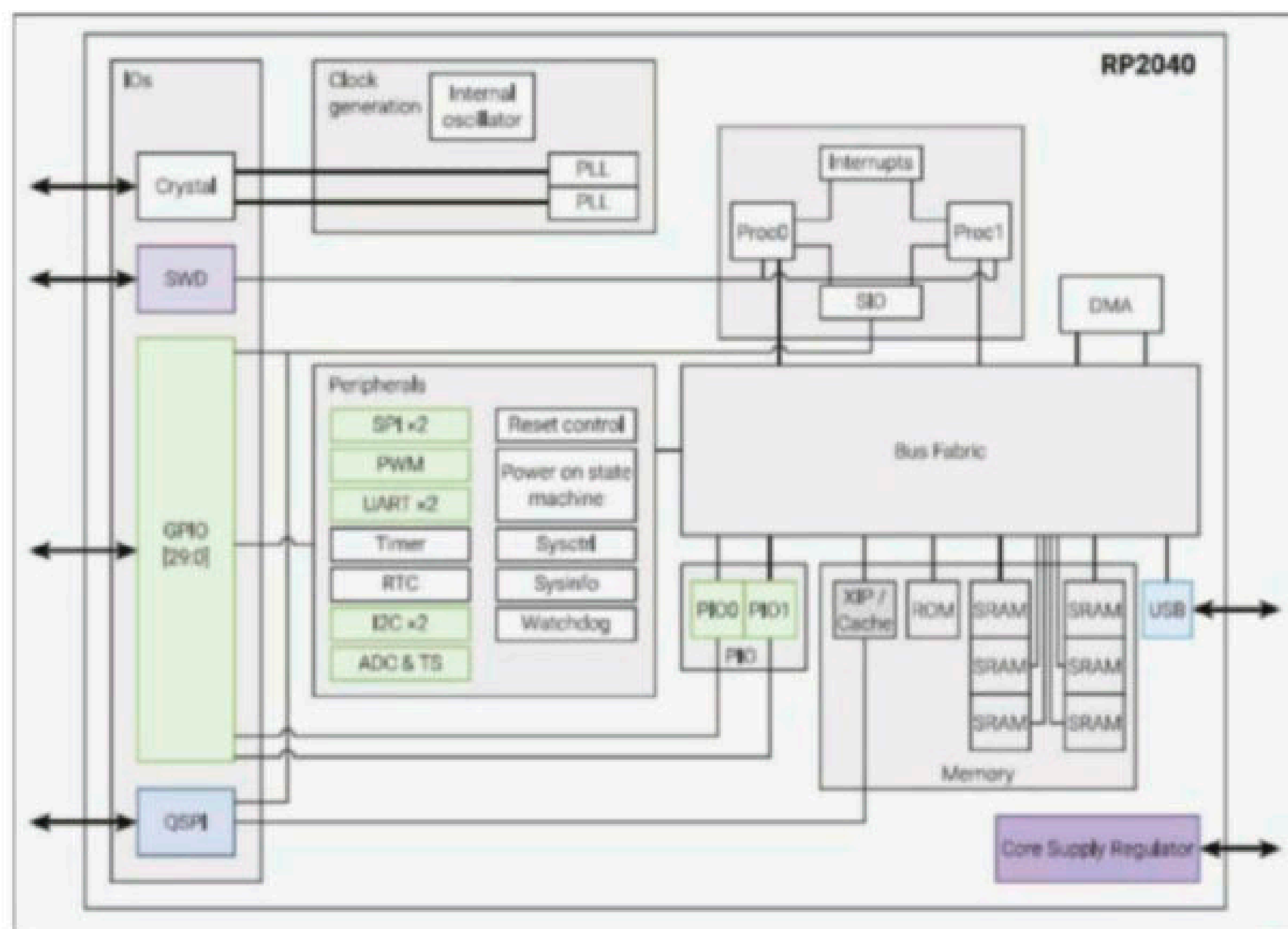


FIGURE 2

This is a block diagram of the RP2040 MCU. Note that each of the two cores and DMA can access separate ROM, (6) SRAM, FLASH, IO, and peripheral devices simultaneously. With up to four simultaneous 32-bit instructions per cycle, this little guy has some serious bandwidth.

Key Features:
Dual-core cortex M0+ at up to 133MHz with 2MB flash
Micro-USB 1.1 B port for power and data (and for reprogramming the flash)
ARM Serial Wire Debug (SWD) port
264KB multi-bank high performance SRAM
External Quad-SPI flash with eXecute In Place (XIP) and 16KB on-chip cache
30 multi-function General Purpose IO (four can be used for the ADC)
Peripherals Include
12-bit 500Ksps Analog-to-Digital Converter (ADC)
Two × UART
Two × I ² C
Two × SPI
16 × PWM channels
One × Timer with four alarms
One × Real-Time Counter
Two × Programmable IO (PIO) blocks, eight state machines total

TABLE 1

The Raspberry Pi Pico key features and peripherals

As shown in **Listing 1**, we begin the application with some variable initialization. A second serial port is set up for the K-LD2. Finally, subroutine `showMainMenu` displays the main menu and looks for some user input.

VT100 commands allow some basic screen formatting. After each user input, the screen is cleared and a menu is displayed. The main menu presents all the classes (**Listing 2**). The subroutine `mainMenuChoice` gets user inputs, and if it matches one of the classes, a different menu is displayed. All menus work in a similar way.

If you choose "C," you would then see the "Class C Menu" and choose "4." You would see the results shown in **Figure 4**.

Note that if the register can be written to, you will get this prompt; just hit ENTER to continue without changing anything. You can investigate all of this further if you like. It's worth mentioning here the ability to change two

parameters, `Hold time` and `Sensitivity`, in real time, via two potentiometers connected to two input pins. These inputs are then read, and the analog conversion value is used to choose one of 10 values available for the actual register parameter. While this is the default, you can disable these pots (inputs) and change these register parameters values directly through the interface (that is, this application).

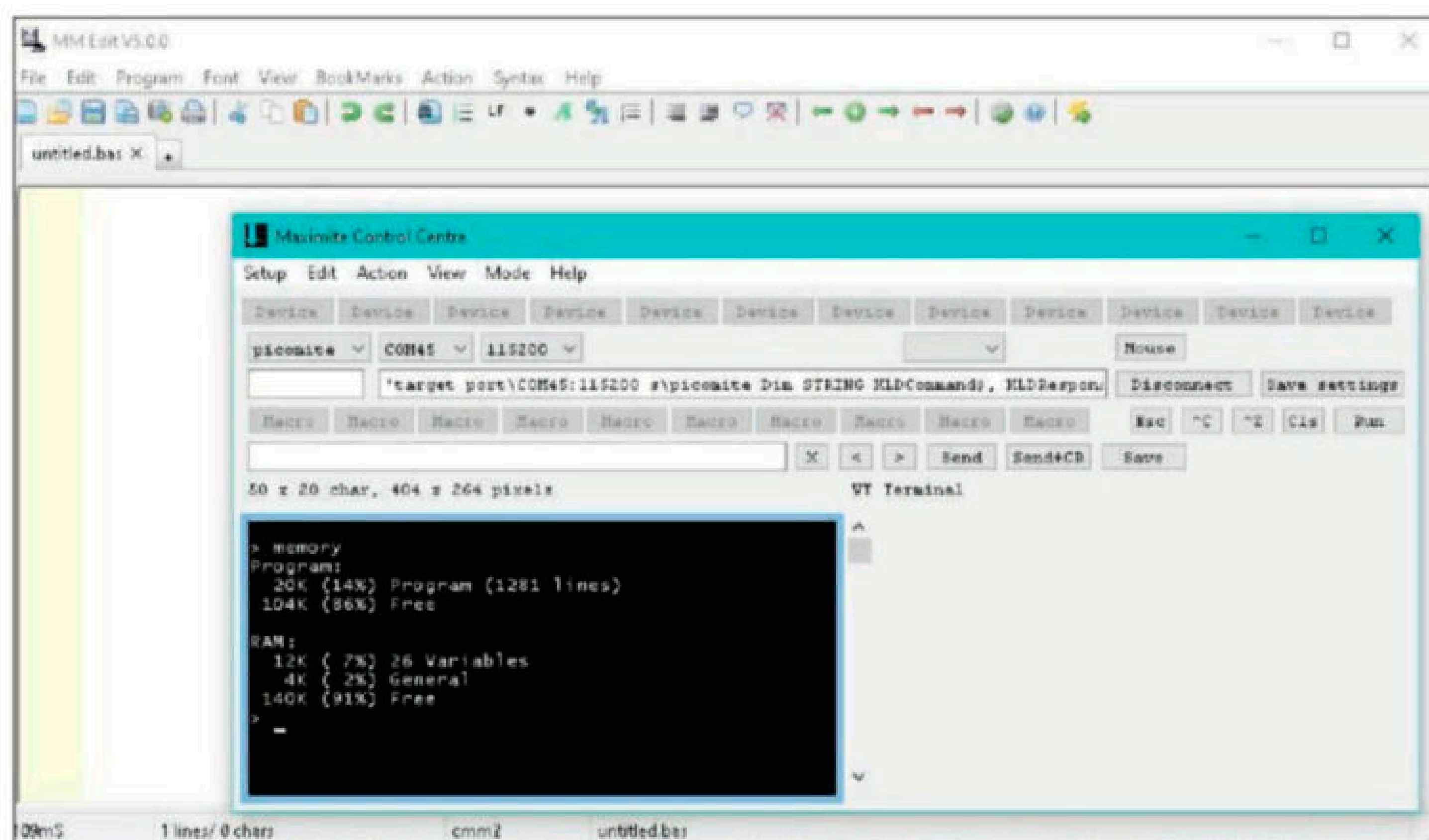
`Hold time` is the number of seconds to hold a detection. It can be one of ten values—0.2 to 160 seconds. The `Hold time` register is a pointer to one of those 10 values held in registers A00:A09.

`Sensitivity` is a value of radar signal in decibels (dB) at which it is considered detected. It can be one of 10 values—0dB to 34dB. The `Sensitivity` register is a pointer to one of those 10 values held in registers A10:A19.

Note that registers A20:A27 hold eight potential filters. A

FIGURE 3

I'm using MMEdit to write my MMBasic application. Upon clicking the action menu item and selecting deploy, the Maximite Control Centre pops up. Here an application is loaded into the Pico, saved and executed. If an application is running, you can exit with a Ctrl-C. This ">" prompt is the command mode. I've entered the memory command, which displays the real time memory stats.



Parameter Type	Cmd Class	EEPROM R/W	Registers	Purpose
System parameters	S	Yes-R/W	S04:S0C	System relevant parameters to configure the sampling and interference suppression
Detection parameters	D	Yes-R/W	D00:D08	Specific parameters to configure the detection algorithm
Array parameters	A	Yes-R/W	A00:A27	System-specific tables
Flash read parameters	F	Yes-R	F00:F01	Read only parameters
Real-time read parameters	R	No-R	R00:R06	Real-time system and detection information
Basic write parameters	W	No- R/W	W00:W02	Basic write parameters to configure the system
Complex read parameters	C	No-R	C00:C06	Advanced read-out parameters
Testing parameters	T	No-R/W	T00:T02	Parameters to test the hardware

TABLE 2

K-LD2 radar module Commands are separated into like-function register groups. EEPROM indicates stored values.

TABLE 3

Communication format

Prefix	Command Class	Register number (HEX)	Byte/Word Value (HEX)	Carriage return
<\$>	ASCII Letter	Register Number	Value	<CR>


```

'target port\COM4:115200 s\picomite
Dim STRING KLDCOMMAND$, KLDRESPONSE$
Dim STRING SignOn$ = "PI PICO Radar"
Dim FLOAT version!
Dim INTEGER C03Array%(1023), verticalPosition = 1, horizontalPosition = 1, myPause = 100, myDebug = &B00000001
.
SetPin GP21, GP20, COM2 ' assign TX and RX for COM2 (K-LD2 radar PCB from Rfbeam.ch)
Open "COM2:38400,4096" As #2
Print SignOn$
hitEnterToContinue ' routine to wait for user input
.
Do While 1
    showMainMenu ' routine to display main menu and look for user input
Loop
End

```

LISTING 1

The first application begins with some initialization which includes variables and a second serial port. The main loop (do while 1) contains a single routine in which all else is based, showMainMenu.

value of 0 disables a filter, and other values 1-127 are identified by one of the filter “bins,” which are determined by the sampling rate. Register D08 determines the bin width for all filters. This allows the elimination of false triggers from an object within the radar field of view that may be causing detections, such as a rotating fan. This is an advanced function, and you will most likely want to know more about the relation between sampling rate, bins, and speed. (See the K-LD2 datasheet [3] available on the RFbeam website.)

Figure 5 is a block diagram of the K-LD2 circuitry. Since it is similar in function to the Grove radar that I described in *Circuit Cellar* last month [1], you might also refer to that column for more background information.

APPLICATION 2: RADAR AND DOTSTAR LEDS

With an understanding of the K-LD2 radar unit, we can get on with the digit-display application. Note that no libraries are used for either the radar module or the DotStar LED interface. The DotStar requires an SPI interface. Adafruit’s DotStar is a continuous string of individually-addressable RGB LEDs; it is an input-only device. While each device has an output, this is a shifted output meant to daisy chain additional DotStar devices. As explained in my column last month [1], I’ve taken a DotStar strip of 60 devices and cut it up into four device segments. These segments become the seven parts of a seven-segment digit, shown in **Figure 6**. Each digit is wired to the next to create two large, seven-segment digits.

The SPI output requires the updating of 56 LEDs (4 LEDs/segment x 7 segments x 2 digits = 56 LEDs). Each LED requires a 4-byte data field consisting of 1 brightness byte, plus 1 byte for each of the three colors red, green, and blue. The total packet requires 4 bytes (start field), followed by 56 x 4 bytes (data field), and 4 bytes (stop field) for a total of 232 bytes. I have separate routines for sending the start and stop data fields. The variable `received_data` is just a dummy variable,

since nothing is connected to the SDI input. As you can see, the the `SPI()` function is simple—it sends the byte (in parentheses) out the `SD0`, receives a byte from SDI and places it in `received_data`.

```

Sub showMainMenu
    clearScreen
    homeCursor
    horizontalPosition = 10
    positionCursor
    Print "Main Menu Class"
    Print "A - Array Parameters"
    Print "C - Complex Read Parameters"
    Print "D - Detection Parameters"
    Print "F - FLASH Read Parameters"
    Print "R - Real Time Read Parameters"
    Print "S - System Parameters"
    Print "T - Testing Parameters"
    Print "W - Basic Write Parameters"
    Print
    Print "Enter a Class Choice"
    mainMenuChoose
end sub

sub mainMenuChoose
    userEntry
    Select Case myString$
        Case "A"
            showMenuA
        Case "C"
            showMenuC
        Case "D"
            showMenuD
        Case "F"
            showMenuF
        Case "R"
            showMenuR
        Case "S"
            showMenuS
        Case "T"
            showMenuT
        Case "W"
            showMenuW
    End Select
End Sub

```

LISTING 2

The showMainMenu subroutine displays choices to the user, and then calls the mainMenuChoose routine to get user entry and determine what to do next. The choice will direct the program flow to display a sub menu.

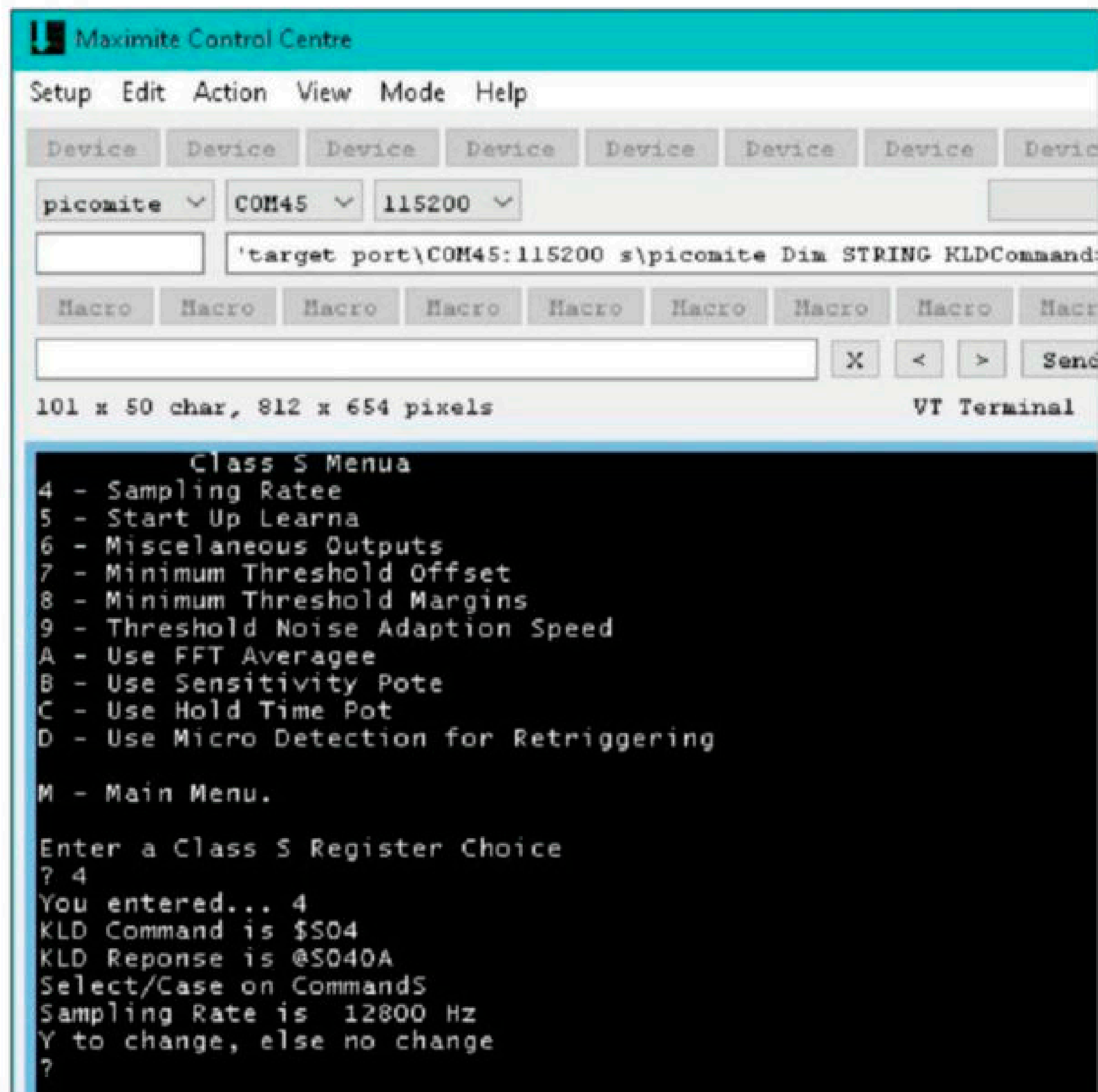


FIGURE 4

The Maximite Control Centre is a pop-up VT100 terminal that is called up by executing a deploy action from the MMEditor. The application from the text editor would be automatically downloaded, saved and run. Here I've halted the program with Ctrl-C. At the command prompt ">" I've entered the memory command to see real-time stats.

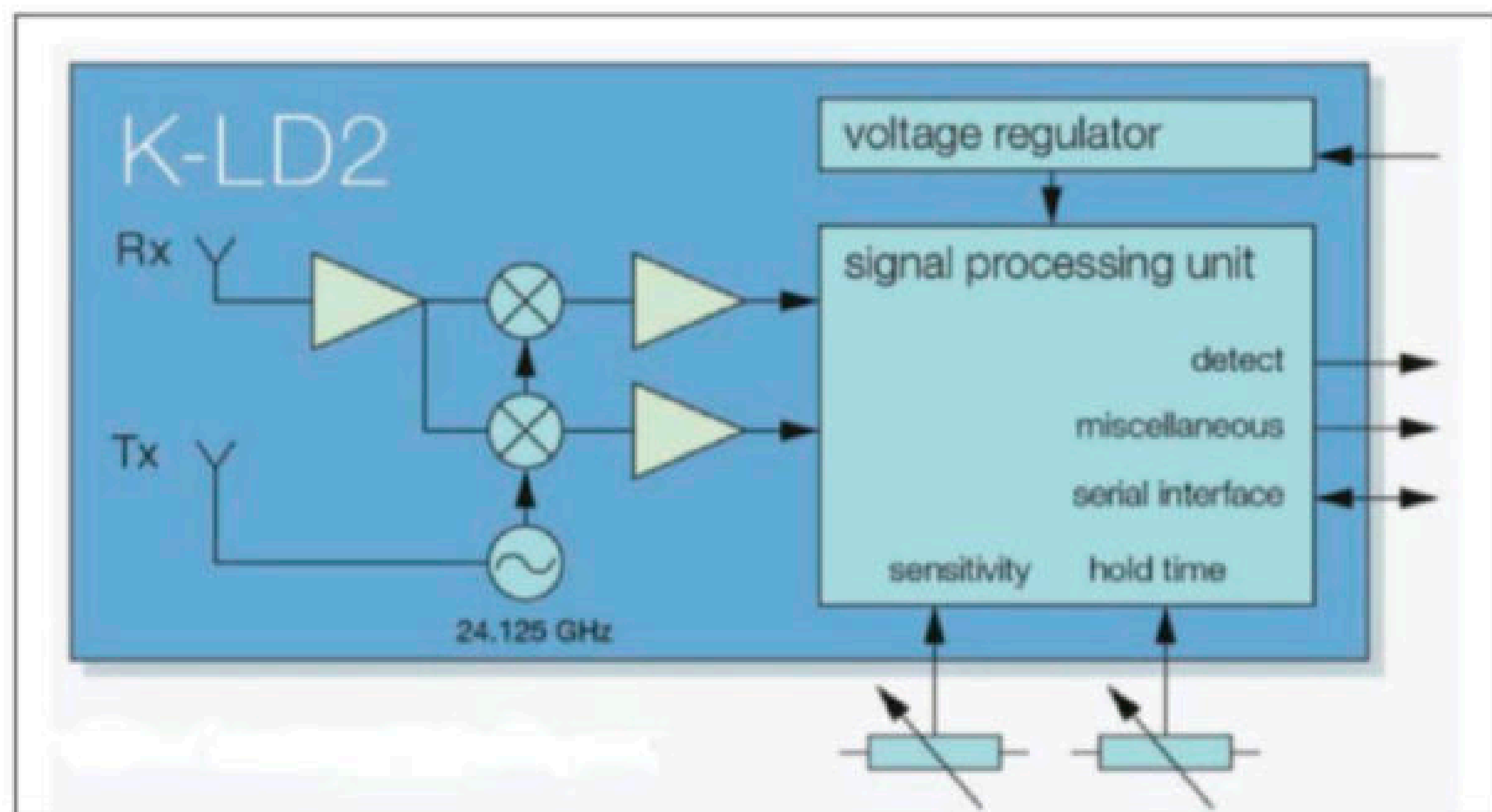


FIGURE 5

This is a block diagram of the K-LD2 radar module. Note the similar RF front end with internal I/Q inputs to the signal processing unit.

ABOUT THE AUTHOR

Jeff Bachiochi (pronounced BAH-key-AH-key) has been writing for *Circuit Cellar* since 1988. His background includes product design and manufacturing. You can reach him at: jeff.bachiochi@imaginethatnow.com, or at: www.imaginethatnow.com.



```
SUB SendEndFrame
received_data = SPI(&HFF)
received_data = SPI(&HFF)
received_data = SPI(&HFF)
received_data = SPI(&HFF)
END SUB
```

```
SUB SendStartFrame
received_data = SPI(&H00)
received_data = SPI(&H00)
received_data = SPI(&H00)
received_data = SPI(&H00)
END SUB
```

The routine `SendLEDFrame` handles the whole process. The `LEDArray` of LED data is just shifted out the SPI port. Note that my debugging print statements in **Listing 3** show the actual bit stream being sent. These can be disabled by changing `myDebug = 0`.

But the true work is handled in the `processValue` routine at the beginning of the `SendLEDFrame` routine. Because I'm shifting bits into the right side of the display, the first data in will go through digit '0' to the last segment in the LED chain of digit 1. `DigitArray()` will hold the tens digit and the units digit of the variable `myValue`. I've predefined the 10 possible 7-bit shifts that will produce the digits 0-9 in `segDefArray()`. Note that Bit0 is not used.

```
Dim INTEGER segDefArray(9)
= (&B11101110, &B10001000,
&B01111100, &B11011100, &B10011010,
&B11010110, &B11110010, &b10001100,
&B11111110, &b10011110)
```

As shown in **Listing 4**, the value in `digitArray()` indexes the data in `segDefArray()` to populate the `segmentArray()`. Finally, for each LED in a segment `LEDsPerSegment 3:0`, for each segment in a digit `Segments 7:1`, and for each digit in the chain `Digits 1:0`, we determine what is actually stored in the `LEDArray()` for each LED based on the `Segments` bit data in `segmentArray()`. If the bit is '1' the segment is lit up (with some color). If the bit is '0' the segment is unlit (blank or black).

All of this takes place any time some speed (in miles per hour) needs to be updated. And that, my friends, comes from the K-LD2.

After we do the required initialization of the serial port, the SPI port, and any registers in the K-LD2 radar module that we need to change from the default values, we come to the last command that forms an endless loop, `DO WHILE 1... LOOP` (**Listing 5**). Here, the flag reflects the state of the GP19 pin, the `detectOut` of the radar module. When a detection is valid, this output will

go high for the holdTime from register SOC. Then, if mySpeedMPH is not the constant SpeedLimit (20mph), it is set to SpeedLimit. This prevents the rest of this routine from executing more than once.

The rest of the routine blanks the display when mySpeedMPH is not the constant SpeedLimit. If it seems as though something is missing here, you're right. I have two timer interrupts set, and they handle the digits and colors displayed.

```
SETTICK 10000, showSpeedLimit, 1
SETTICK 200, showSpeed, 2
```

The first executes the routine showSpeedLimit every 10 seconds. We simply return, unless flag = 0 (no detection), then set myValue to SpeedLimit. The routine getSpeedLimit sets the digit color to WHITE, and the routine sendLEDFrame shifts out the myValue digits. Finally, mySpeedMPH is set to zero to trigger the blanking of the display. It is ON for just a quick flash. When there is no detection, the posted speed limit is flashed every 10 seconds.

The second routine, showSpeed, is executed every 200ms. We simply return, unless flag = 1 (detection); otherwise we send Command \$C00 to the radar module. This gives back three items—the detection register, the detection bin, and the detection strength. If bit 1, the direction of detection register, is "toward" the radar, then the speed is calculated from the myBin number and mySamplingSpeed using the equation:

$$v(\text{KPH}) = \frac{\text{bin} \times \text{sample rate} \times 1,280}{256 \times 44.7\text{Hz}}$$

And so:

```
mySpeedKMH = (myBin * mySamplingRate
* 1280) / (256 * 44.7)
mySpeedMPH = mySpeedKPH * 1.60934
```

Before we send this speed to the LEDs, we check to see if it's greater than the SpeedLimit, and make the color RED; otherwise we make the color GREEN. myValue = mySpeedMPH and the routine sendLEDFrame shifts out the myValue digits. Finally mySpeedMPH is set to zero to trigger the blanking of the display. It is ON for just a quick flash. Flashing the display allows for a lower average current draw. While there is a detection active, the detected speed is flashed multiple times per second.

CONCLUSION

I hope I've been able to demonstrate that you can take many paths to accomplish a task.

```
SUB SendLEDFrame
  processValue
  IF myDebug AND 1 THEN
    PRINT "Shifting Data Out"
  ENDIF
  SendStartFrame
  FOR i = 0 TO (NumberOfPixels * 4)-1 STEP 4
    received_data = SPI(LEDArray(i))
    received_data = SPI(LEDArray(i+1))
    received_data = SPI(LEDArray(i+2))
    received_data = SPI(LEDArray(i+3))
    IF myDebug and 1 THEN
      IF LEDArray(i+1) = 0 AND LEDArray(i+2) = 0 AND
      LEDArray(i+3) = 0 THEN
        print "0";
      ELSE
        print "1";
      ENDIF
    ENDIF
  NEXT
  IF myDebug AND 1 THEN
    PRINT
  ENDIF
  SendEndFrame
END SUB
```

LISTING 3

In the second application we have data placed in LEDArray() via subroutine processValue. We need to transmit this data to the DotStar LED strip via SPI() command. Each of the two display digits are made up of seven segments with four LEDs per segment. Since each LED requires four bytes, LEDArray() holds 2 x 7 x 4 x 4 or 224 bytes.

```
SUB processValue
  FOR i=Digits-1 TO 0 STEP -1
    digitArray(i) = myValue\10 'integer division
    segmentArray(i) = segDefArray(digitArray(i))
    IF myDebug AND 16 THEN
      print "myValue is ";myValue
      print "digitArray(:";i;") is ";digitArray(i)
      print "segmentArray(:";i;") is ";segmentArray(i)
    ENDIF
    myValue = (myValue - (digitArray(i) * 10)) * 10
  NEXT
  i=0
  FOR D=0 TO Digits-1
    FOR S = Segments TO 1 STEP -1
      FOR L = LEDsPerSegment-1 TO 0 STEP -1
        'D0 = D * Segments * LEDsPerSegment
        'S0 = (S-1) * LEDsPerSegment
        'L0 = L * LEDsPerSegment
        if(segmentArray(D) AND 1<<S) THEN
          LEDArray(i) = &HE0 + Brightness
          LEDArray(i+1) = BlueColor
          LEDArray(i+2) = GreenColor
          LEDArray(i+3) = RedColor
        ELSE
          LEDArray(i) = &HE0
          LEDArray(i+1) = 0
          LEDArray(i+2) = 0
          LEDArray(i+3) = 0
        END IF
        i=i+4
      NEXT
    NEXT
  NEXT
END SUB
```

LISTING 4

The processValue subroutine loads the LEDArray() with the proper data depending on the digits and color needed. Digit segment data is held in segDefArray. Each array byte is the data for a digit 0-9, and each bit indicates whether that segment of the digit should be illuminated or not.

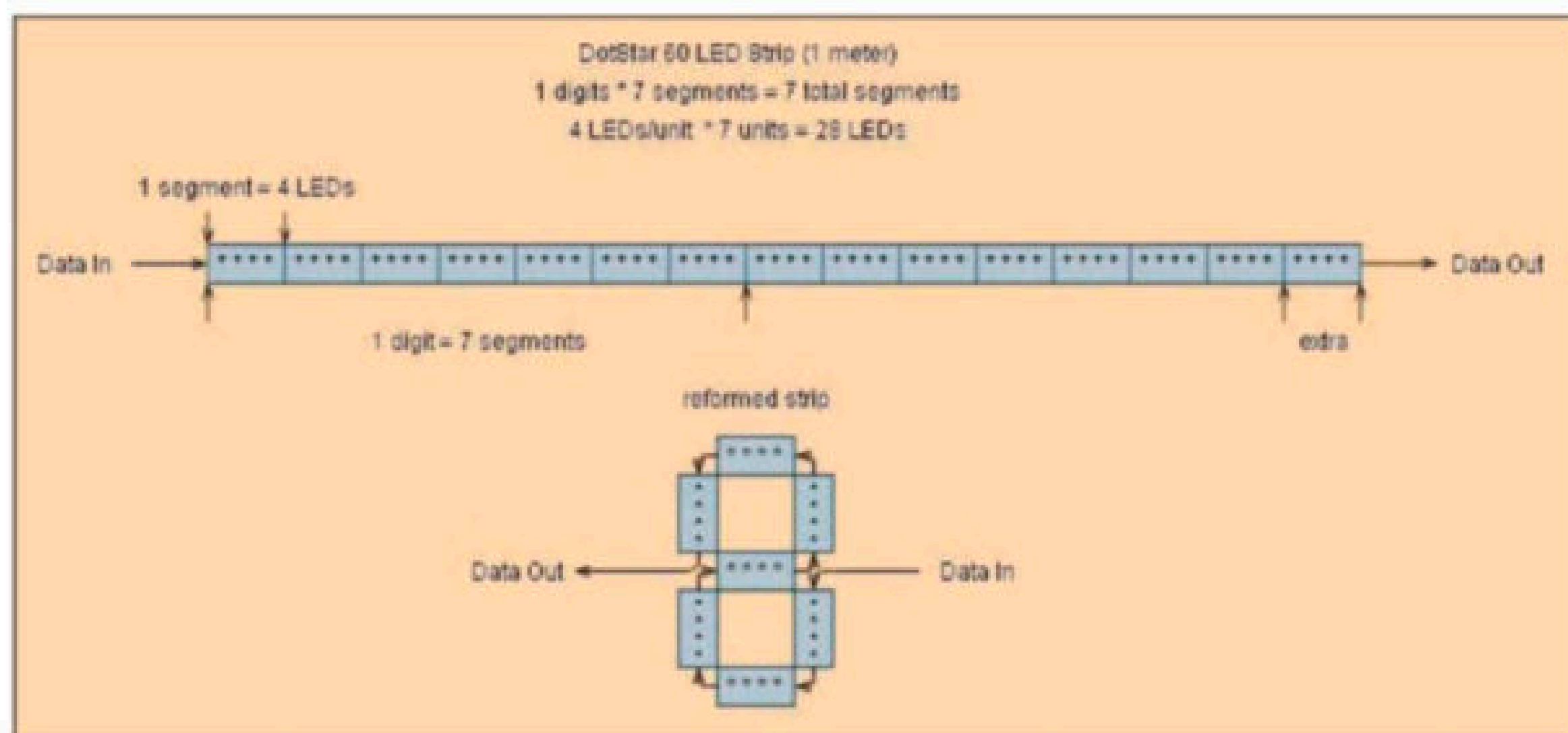


FIGURE 6

I made tiny square PCBs to interconnect the segments at right angles, which make an S-shaped path through the seven-segment digit. Digits can be cascaded. Make sure you arrange the in and out ends of the segments correctly. If you flip them all around from this orientation you can connect from the left. Just make sure that your data is arranged correctly, based on your choice of direction flow!

FIGURE 7

This project is ready to mount inside the enclosure I described in my column last month [1]. The Gel-cell serves as the 6.0-7.2V supply. The solar panel has a built-in charger. A buck/boost switching regulator gives the DotStar LEDs a hard 5V, whereas the Pi Pico and K-LD2 run on 3.3V.



LISTING 5

The input pin GP19 indicates when a legal detection has taken place and sets the flag bit. If mySpeedMPH is not = SpeedLimit (in my case 20), then make mySpeed and myValue = SpeedLimit and turn off all the LEDs. This keeps the display blank most of the time. The display is updated by one of the two timer interrupts—one to continuously flash the measured speed if flag = 1, and the second to flash the speed limit every 10 seconds if flag = 0.

```
DO WHILE 1
  IF PIN(GP19) = 1 THEN
    flag = 1
  ELSE
    flag = 0
  ENDF
  IF mySpeedMPH <> SpeedLimit THEN
    mySpeedMPH = SpeedLimit
    myValue = mySpeedMPH
    getblank
    sendLEDFrame
  ENDF
LOOP
END
```


While you may choose a path based on the best tool for the job, often there are many possible options, all of which will get you over the finish line. Don't be afraid to jump right in and try something new. The learning curve might require a bit more effort, but the rewards are considerable. Adding to your repertoire will make you a more valuable resource.

Figure 7 shows the completed project, ready to mount inside the enclosure described previously [1].

My fond memories of programming the TRS-80 Model 1 in the late 1970s began with the supplied BASIC interpreter. It is good to see that this old standby is still around, by making its presence known to a new MCU and a new crowd that may not even know of BASIC's existence. Congratulations to the Raspberry Pi crew on the release of the RP2040, their flagship MCU chip.

Recently the Raspberry Pi Pico W was released, adding Wi-Fi to the Pi Pico 40-pin DIP format. A Cypress CYW43439 chip enables a 2.4GHz WLAN IEEE 802.11 b/g/n MAC/baseband/radio, and Bluetooth 5.0 compliance. For \$6, I'm sure it will be a big hit as well. That peripheral uses an SPI communication channel, so you should still be able to use MMBasic with the device. Presently MMBasic has direct support for these peripherals:

- SD Cards with FAT16 or FAT32 file systems up to 32GB
- LCD, OLED and e-Ink display panels from 0.96" to 3.5" (diagonal), with resolutions up to 480x320 pixels
- Touch Sensitive LCD panels
- Real-Time Clocks using the PCF8563, DS1307, DS3231 or DS3232 chips
- Infrared Remote Control support allowing a Sony or NEC IR remote control
- Temperature and humidity measurement using the DS18B20 or DHT22/DH11 sensors
- Distance measurement using the HC-SR04
- Numeric keypads with a 4x3 or a 4x4 keypad layout
- WS2812 multi-color LED chips

All these features are built into the BASIC interpreter, so there is no need to load libraries or write special code. It's possible we may see Wi-Fi support added in the future, but until then, don't be afraid to write your own support routines. For Arduino usage you might take a look at the Earle Philhower Arduino core, which is used to to put Arduino on all your favorite RP2040 devices [4]. It has lots of peripherals supported, including Wi-Fi. So many options, so little time. 

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 adafruit | www.adafruit.com • Arduino | www.arduino.cc • Raspberry Pi | www.raspberrypi.com

PRODUCT NEWS *by Stephen Vicinanza*

The VEGA-300 Series is Built for AI at the Edge

Advantech's VEGA-300 series AI acceleration modules are built for AI at the Edge. They provide highly efficient add-on modules in different form factors including M.2, mini PCIe, and PCIe4. They can be easily integrated into existing systems. The integration is seamless and will simultaneously reduce CPU workload. The Edge AI Suite software allows users to quickly enable AI inference on edge devices. That allows the realization of AI and vision analytics applications in many industries including retail, transportation, medicine, and manufacturing.



AI-power smart street lights—There are thousands of street lights in cities throughout the world. They help drivers and pedestrians find their way to their destinations safely. When equipped with AI technology, street lights can be used to not only light up roads but also support city services such as air quality, temperature monitoring, humidity, and also parking and traffic management.

The solution is the M.2 AI Modules: Vega 320. The city planners had already implemented the edge computing system, to serve as a gateway computer for collecting data, including humidity and temperature, which helped facilitate operational analysis. That system was updated to the VEGA-320 edge AI acceleration modules, embedded into cameras for analyzing multi-streams of video data. The system also performed real-time tasks such as finding available parking spaces. The results were encouraging with the city seeing a 10% to 15% increase in the utilization of roadside parking.

Advantech has developed a suite of tools they call the Edge AI Suite, for accelerated deep learning inference on edge devices. The systems are integrated with Intel's Distribution of OpenVINO toolkit. The Edge AI Suite provides a deep-learning model optimizer, pre-trained models, a user-friendly GUI, and a model optimizer.

Advantech | advantech.com

Onsemi Launches Top-Cool MOSFETs That Could be Innovative

onsemi launched a top-cool MOSFET that could be an innovation to other MOSFETs. These new MOSFET devices were announced just in November 2022, featuring a new top-side cooling to assist designers who are tackling those challenging automotive applications.

The focus for these MOSFETs is on motor control and DC-DC conversion. The company is showing new devices at electronica, this year.

The MOSFETs are housed in a TCPAK57 package measuring 5mm x 7mm, the latest Top Cool devices feature a thermal pad on the topside, measuring 16.5mm(2). This allows for direct heat dissipation directly into the heatsink rather than into the Printed Circuit Board (PCB). This enables both sides of the PCB and decreases the amount of heat going into it and the TCPAK57 allows for increased power density.

There is an overall extension of the system lifetime with the improved reliability of the new design.

"Cooling is one of the greatest challenges in high power design and successfully addressing it is the key enabler to reducing size and weight, which is critical in modern automotive design," said Fabio Necco, vice president and general manager, of Automotive Power Solutions at onsemi. "With excellent electrical efficiency and having eliminated the PCB from the thermal path, the design is significantly simplified while reducing size and cost."

The delivery of higher electrical efficiency these devices deliver makes them ideal for high-power applications, with RDS(ON) values down around 1mΩ. The R(DS) x Qg is low (65nC) reducing losses in the high-speed switching

applications

This solution can leverage onsemi's expertise in packaging and also provide high power density setting an industry standard. The initial portfolio for TCPAK57 includes 80V, 60V, and 40V devices. The set of devices are capable of operating at junction temperatures of 175° and are AEC-Q101 qualified and PPAP capable.

Along with the gull wings, this allows for the inspection of solder joints and superior board-level reliability. That makes the MOSFETs ideal suited for the more demanding increased heat of automotive applications. This would be especially true in target applications of high and medium level power motor control. Some of the use cases would be electric power steering and oil pumps.

onsemi | onsemi.com



EPC Launches new eGaN Technology Doubling Performance

EPC launches a new eGaN technology that doubles the performance of previous solutions. Efficient Power Conversion (EPC) introduces the 80V, 4mOhm EPC2619 GaN FET in a 1.5mm x 2.5mm package. This FET offers higher performance and smaller solution sizes than the traditional MOSFETs found on the market today.

The EPC2619 offers high power density for applications including DC-DC conversion, motor drives, and synchronous rectification for the 12V – 20V ranges.

The footprint of the EPC2619 is based on an $R_{DS(on)}$ of only 4mOhms in the tiny 1.5mm x 2.5mm form factor. The footprint area of the EPC2619 is $15m\Omega \cdot mm^2$, which is five times smaller than any other 80V silicon MOSFETs.

The EPC2619 was designed with a wide variety of motor drive applications in mind. For example solar optimizers and synchronous rectification converting 12V to 20V for chargers, TV power supplies, and adaptors. 28V – 48V for conversions of power tools, eBikes, and eScooters. Also for such solutions as DC-DC converters.

The usual $R_{DS(on)} \times Q_{GD}$ indicative of the power losses in hard-switching applications, is overall ten times better than 80V silicon MOSFETs. This allows for switching frequencies that are increased ten times higher than the silicon MOSFETs. There is no loss of efficiency in this frequency and hard switching resulting in a higher overall power density.

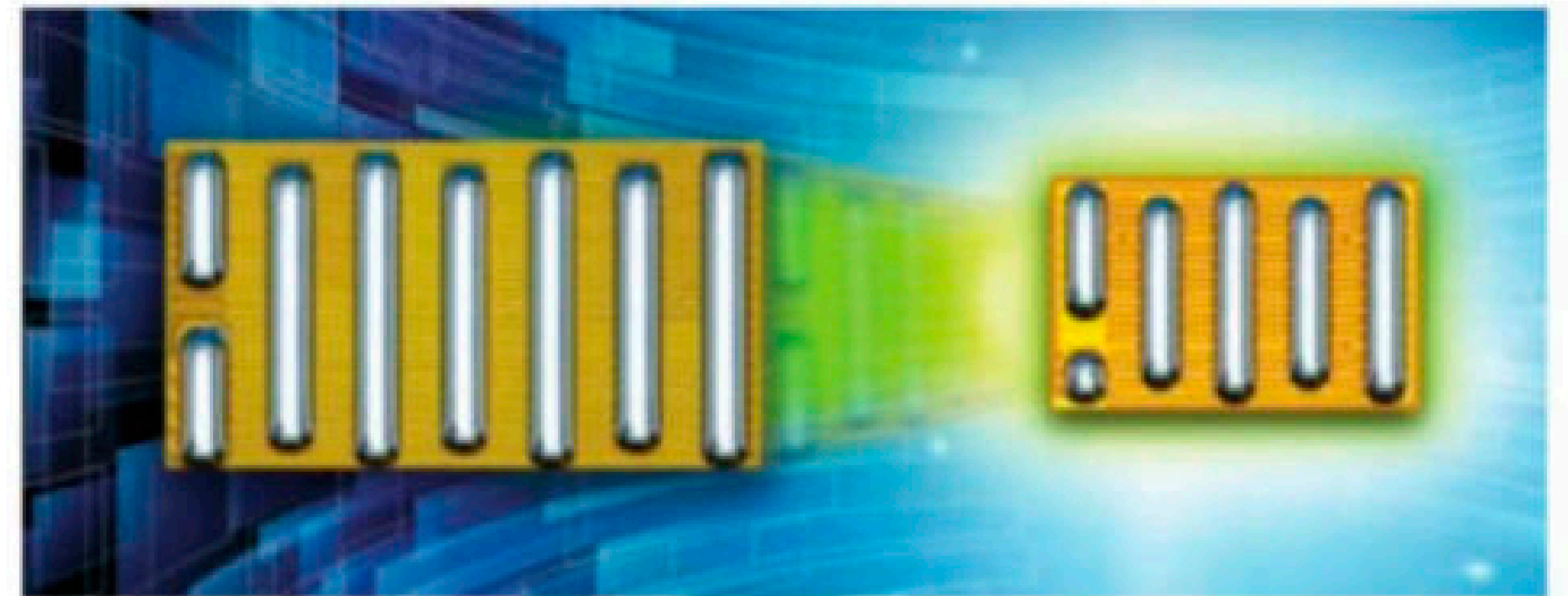
The combinations of hard switching and higher frequencies make the FETs ideal candidates for high-frequency hard switching 24V – 48V applications that include boost converters, boost, and buck-boost scenarios.

In a soft-switching environment, where Typical $R_{DS(on)} \times Q_{oss}$ are indicative of a power loss, the range is 87 mOhms*nC twice as good as an 80V silicon MOSFET. This increase in rates is giving the EPC2619 the capability to work well in soft-switching applications. This could be in the primary rectification full bridge for LLC-based DCX DC-DC converters.

“This is just the first product of a new generation of discrete transistors and integrated circuits for EPC. With the launch of the EPC2619, EPC continues to keep GaN power devices on a path reminiscent of Moore’s Law,” noted Alex Lidow, EPC CEO and co-founder.

The EPC90153 is a development board half bridge featuring the EPC2619 GaN FET. The board is 2" x 2" (50.8mm x 50.8mm) and designed for optimal switching performance. It carries all the critical components necessary for an evaluation of the power systems designers are searching for today. The aim is to speed the design process from concept and the time-to-market.

EPC | epc-co.com



Extreme Engineering Solutions New SBC is Focused on Intelligence and Surveillance

Extreme Engineering Solution’s (X-ES) XCalibur4840 6U VPX-REDI Module, is a new single board computer (SBC) sporting an intel D-2700 series, which was formerly an Ice Lake-D chip, series of processors. These SBCs are focused on computational heavy applications requiring maximum data and information protection. It is applicable in the areas of command, control, communications, computers, intelligence, surveillance and reconnaissance (C4ISR), radar, and many other mission-critical defense systems.

The Intel Xeon D-2700 processors are power-efficient System on Chip (SoC) packages with integrated 40 Gigabit Ethernet for high-speed connectivity. Significant improvements over the existing line of Xeon D chips, in terms of sheer processing power and memory density, make the Ice Lake-D the ideal solution for computationally heavy applications.

The Xeon D-02700 series products have as many as 20 cores and four memory channels to provide maximum processing capability in a high-performance computing environment. These new processors come with full support from Intel’s Internet of Things Group (IOTG). The Xeon D-2700 processors have Intel’s commitment

to 15-year availability. This is an added benefit and increases the reliability of this family of processors.

The XCalibur4840 is equipped with extremely fast connectivity access. It holds two 40GBASE-KR4, two 100BASE-X, and two 10/100/1000BASE-T Ethernet ports. It can handle up to 64GB of DDR4 ECC SDRAM in four channels, and also 32GB of onboard SLC NAND flash in addition to the many I/O ports. The SBC comes with additional expansion through two integrated XMC/PMC sites. These two sites include an x8 PCIe connection to the Xeon D Processor and X12d+X8d I/O mapped directly to the VPX backplane connectors.

Each mezzanine site offers a single PMC connector, which provides the build option for the P64s, or X38s to the VPX backplane connectors.

The XCalibur4840 has an integrated SecureCOTS technology integration with a Microsemi PolarFire FPGA for hosting custom functions, to protect data from outside modification or observation and provides the ideal solution for when the strictest security measures are required.

X-ES | www.xes-inc.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 adcopy@circuitcellar.com with your file.

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



Datakey
RUGGEDrive™
Memory Tokens

USB Flash Drive or SD Card Functionality
in a Robust Proprietary Form Factor

Need a USB Stick or SD Card...
That Doesn't Look Like One?

Register to Win a RUGGEDrive™ Development Kit
circuitcellar.com/datakey



MCC
Micro Computer Control

I²C Bus Adapters

USB / Ethernet / RS-232

Free
Visual Studio, LabVIEW & ASCII APIs
Win, OSX & Linux Drivers
Msg & EEPROM Program Apps

Switch Selectable Options
Voltage Sense (0.5v to 5v)
Power Source 100mA (3.3v or 5v)
Pull-Ups (Enable/Disable)
Over/Reverse Voltage & ESD Protection

shop-mcc-us.com



E3mini Board & Book Bundle
Learn to Code in C Today!

Bundle Contents	
E3mini Prototyping Board	Embedded C Programming Textbook
USB Cable	Single-Chip Compiler

Companion Hardware

E3mini Advanced Accessories Kit	53217-1530 \$25
Sensors Explorers Kit	5-205 \$69

Only \$84.99

A complete introduction to C programming

Channels knowledge into real-world examples

Low cost companion parts kits available

For additional information see the URL below!

Sales@ccsinfo.com
(262)522-6500 Ext. 35
www.ccsinfo.com/cc123




UNLOCK CC 2022
12 Issues On Your Device
Now at CC-Webshop.com

Advances in the Modern Railway System
TEST and MEASUREMENT TOOLS
EMBEDDED SYSTEMS ENABLING CONDITION MONITORING
HYBRID AND ELECTRIC VEHICLE SOLUTIONS
DESIGN
INNOVATIONS IN RABBIT DEVICES
ONE DEVELOPER POWER HANDS ON
EMBEDDED SOLUTIONS LEFT UP SMART CITIES
CONTROL SOLUTIONS FOR ROBOTICS
EMBEDDED PCs SUIT UP FOR RY TRANSPORTATION



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—What's the distinction between a microprocessor (MPU) and a microcontroller (MCU)?

PROBLEM 2—Can you name some examples of MPUs and MCUs?

PROBLEM 3—What's the distinction between a System-on-Chip (SoC) and a System-on-Module (SoM)?

PROBLEM 4—What's the difference between an SoM and a single-board computer (SBC)?

DIGI enabling the edge

Connect with Confidence

Partnering with Digi for your IoT project can deliver exceptional ROI - from critical insights to cost reductions

“Digi ConnectCore is very small and power-efficient, so it fits nicely in our device. We received extraordinary technical support from the Digi team. We're very pleased to have a successful long-term relationship with Digi.”
- Aditya Bansal, Kinetic Co-founder and CTO

“As Wi-Fi has gained momentum in healthcare, we quickly shifted our designs to connect wirelessly, which hospitals prefer for reasons of security and convenience. As we've transitioned Digi has been right there with us.”
- Ned Buffington, Avancen Chief Executive Officer

“We created a privacy platform so patients can securely access and provide sensitive healthcare data without using a smartphone or the Internet in a way that completely meets legal requirements.”
- Dieter Rittinger, DeGIV Founder and CEO

KINETIC

AVANCEN
IMPROVING PATIENT CARE AT THE EDGE

DeGIV
genial digital

Scan this QR code to register for our Digi ConnectCore® 8M Mini Development Kit giveaway
www.circuitcellar.com/digi

The Future of Private LTE

Private Cellular Networks in Buildings

Reliable in-building mobile connectivity is essential to everyday living, as people and businesses use it to communicate and operate their remote devices. Connectivity is so important that it is increasingly considered the fourth utility—along with electricity, water, and gas. The demand for reliable mobile coverage will only intensify as building owners and businesses continue to implement Internet-of-Things (IoT) technologies, such as robots, cameras, and sensors to foster business productivity and growth. In 2022, the IoT market is expected to grow by 18% to 14.4 billion active connections [1].

As IoT networks become more complex, traditional LTE/4G coverage and Wi-Fi are not always enough to drive these IoT devices. When coverage is spotty and unreliable within a building, sometimes due to the building materials themselves or the contents of the building interfering with the signals, building owners and business leaders can turn to a customizable private LTE network to ensure seamless integration of all their devices.

BENEFITS OF PRIVATE LTE

Private LTE can be a better alternative to public LTE and Wi-Fi, with lower bandwidth costs and additional benefits such as:

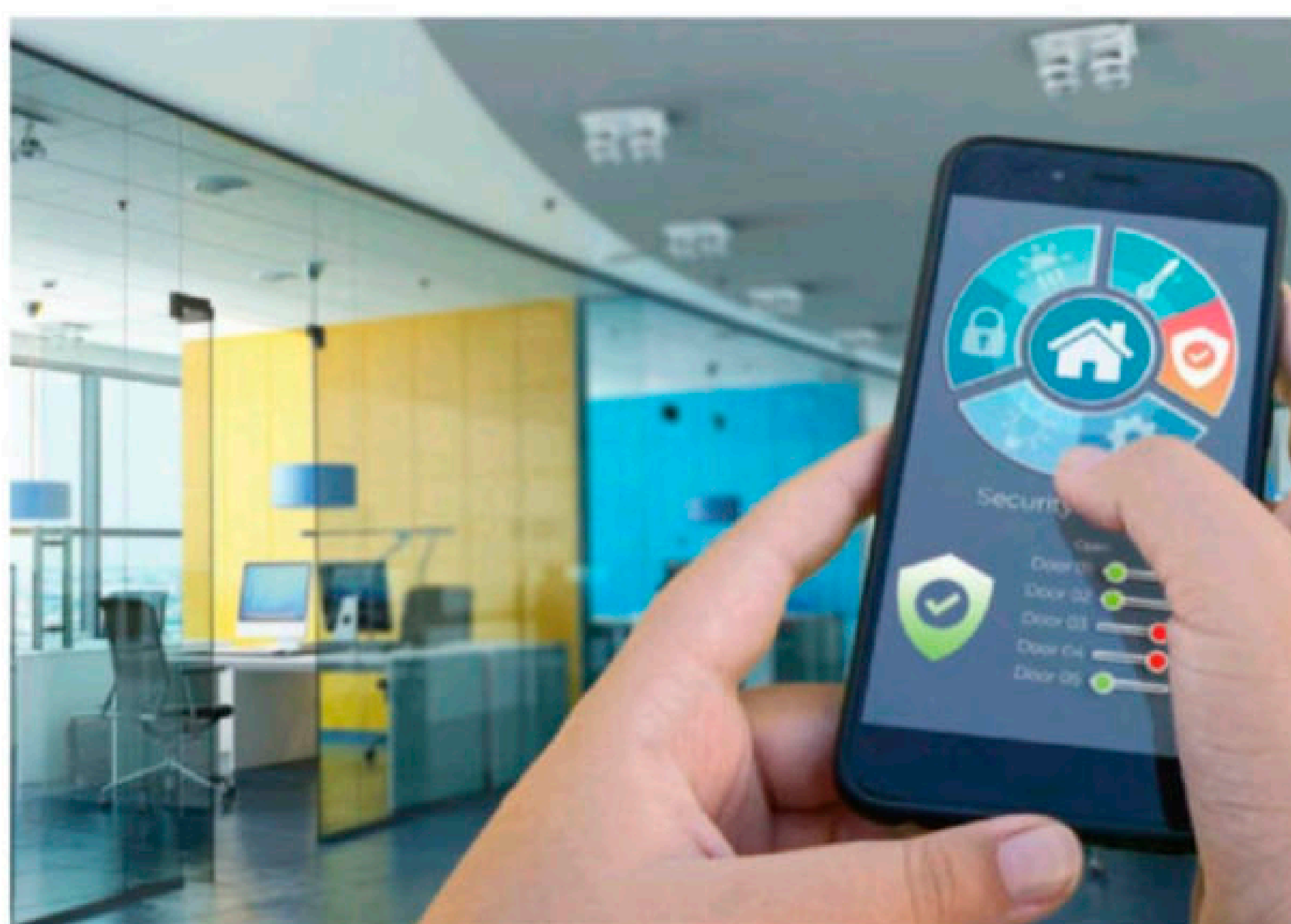
- **Reliable coverage:** Unlike Wi-Fi, where coverage is easily disrupted and can be spotty due to network hand-offs—even with the latest signal-boosting technology—private LTE networks are inherently more rugged. With the help of a professional team of system integrators and installers, private LTE networks are designed to deliver

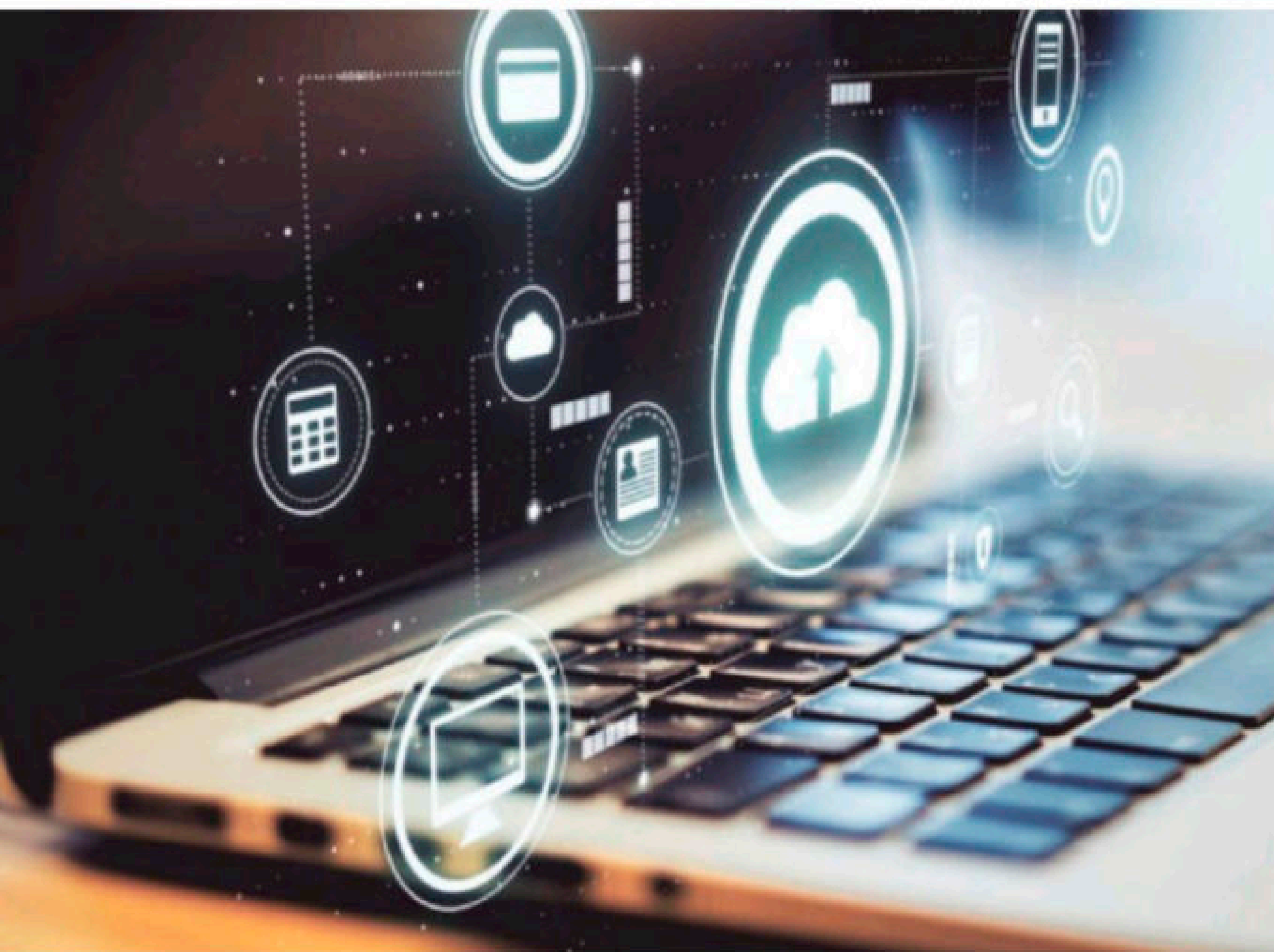
low-latency performance for specific applications.

- **Customization:** Private LTE allows businesses to control data usage and prioritize which devices and applications get faster speeds by executing service level agreements (SLAs) for throughput and latency.
- **Security:** As the name states, the main benefit of private LTE is a private network. It keeps an organization's data confidential by limiting access to the network. And it has greater security policies—with 24/7 centralized encryption—than traditional Wi-Fi, which only asks for a password and allows any stranger to try to enter the network. This feature is especially important as data breaches continue to rise, with average costs increasing 2.6% from 4.24 million dollars in 2021 to 4.35 million dollars in 2022 [2].



By
Stephen Kowal
Chief Commercial Officer
at Nextivity





Due to these benefits, enterprises are increasingly transitioning their traditional in-building networks to private LTE. Currently, there are 1,000 private LTE networks, with the deployment of private LTE set to increase tenfold by 2026 [3].

While building owners and business leaders may be aware of private LTE's benefits, they may not always know where to start with the implementation process and may be presented with various in-building network options, such as traditional passive distributed antenna systems (DAS) or an active hybrid DAS.

CHALLENGES OF DEPLOYING PRIVATE NETWORKS WITH TRADITIONAL NETWORK SYSTEMS

Although both traditional passive DAS and active DAS hybrid solutions provide companies with robust coverage and reliability, they differ in their infrastructure and time to implement.


Traditional DAS has many disadvantages, but the two primary drawbacks are high implementation and equipment costs and long deployment times (sometimes upwards of 12 to 18 months). Long deployment times ultimately drive costs up even further as businesses look for temporary coverage solutions in the meantime—and these solutions lack the benefits private LTE offers in the first place, such as security and reliability.

Traditional DAS may also require a complicated infrastructure with fiber backhubs and base stations, which adds even more to the bill and hurts the bottom line.

IMPLEMENTING PRIVATE LTE WITH AN ACTIVE DAS HYBRID SYSTEM

An active DAS hybrid system, on the other hand, can be deployed in weeks instead of months or years. This is because its system already has integrated features, such as category cable architecture with no signal attenuation, meaning there is no need to add additional heavy frameworks to the network.

The process of active DAS implementation starts with a system integrator doing a site walk-through to take measurements and determine existing coverage, so that they can design the best solution for the building. This can include cabling, antenna, and equipment configurations, as well as signal-aiming tools that optimize signal quality at the site. They will also look at the floor plan and the building materials to ensure there is no signal attenuation. If the business wants capacity in addition to coverage, then it can also add a small cell to achieve this goal.

As more companies and building owners begin to value security and control over their networks, they need a robust private LTE coverage solution that can enable all their devices. Private LTE puts the power back in the hands of companies. It lets them decide how the network functions and delegates service to devices, so they can get ahead of the competition across a variety of industries such as smart cities, Industry 4.0, and more. 

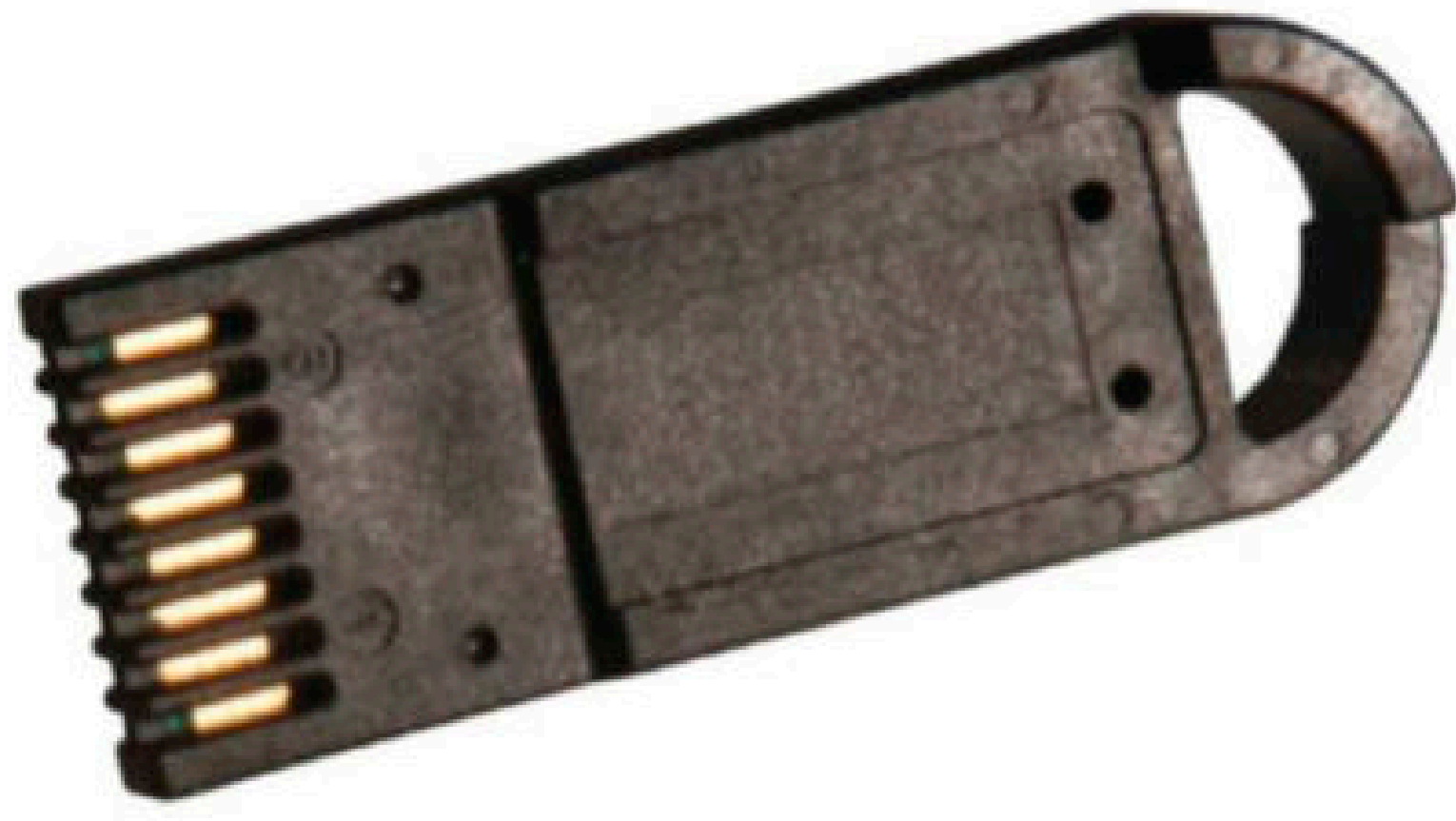
Additional materials from the author are available at: www.circuitcellar.com/article-materials
References [1] to [3] as marked in the article can be found there.

RESOURCES

Nextivity | www.cel-fi.com

ABOUT THE AUTHOR

Stephen Kowal is a technology industry veteran who has held strategic roles in sales, channel, and global accounts for nearly 25 years. As Chief Commercial Officer at Nextivity, Stephen is responsible for the company's customer- and partner-facing teams, specifically those focused on sales, business development, marketing, product management, customer service, and order management.



RUGGEDrive™ Memory Tokens

USB Flash Drive or SD Card Functionality
in a Robust Proprietary Form Factor



Need a USB Stick or SD Card...
That Doesn't Look Like One?

RUGGEDrive Tokens:

Choose from SD or USB
Memory Capacities: 4, 8, 16, 32 or 64 GB
Redundant Contacts – can't insert it upside down

RUGGEDrive Receptacles:

Rated for 50,000 insertion/removal cycles
Reduce vulnerabilities by eliminating accessible USB ports
Standard USB sticks and SD cards don't plug in



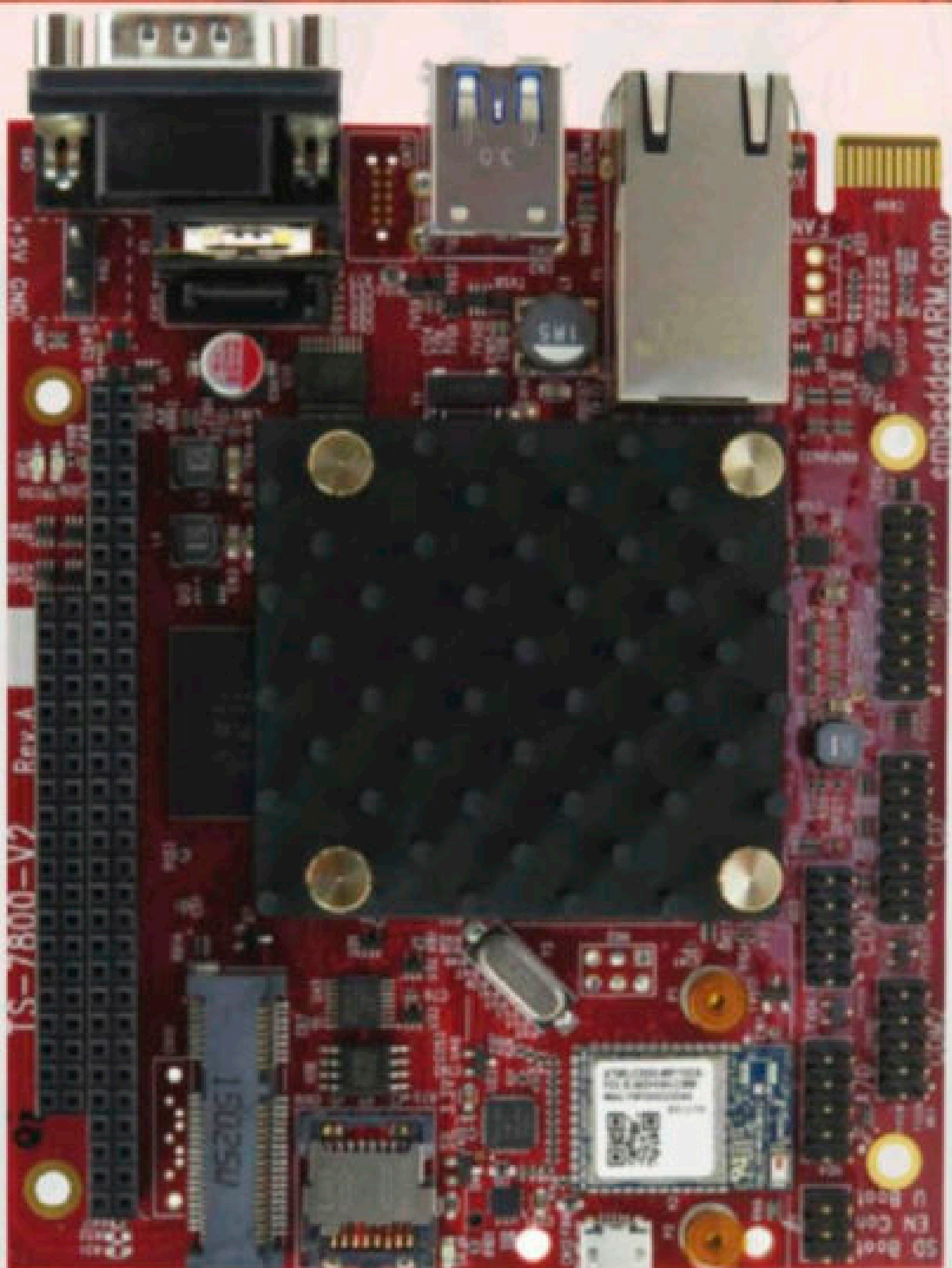
Register to Win

a RUGGEDrive™
Development Kit



circuitcellar.com/datakey

FROM THE OUTBACK TO OUTER SPACE



TS-7800-V2

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

Starting at
\$229
Qty 100

Extreme Remote Deployed Assets.

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

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

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

Where does your project need to go?



Made in USA
with Global Parts

