SEPTEMBER 2015
VOLUME 13
#6
EMBEDDED-COMPUTING.COM

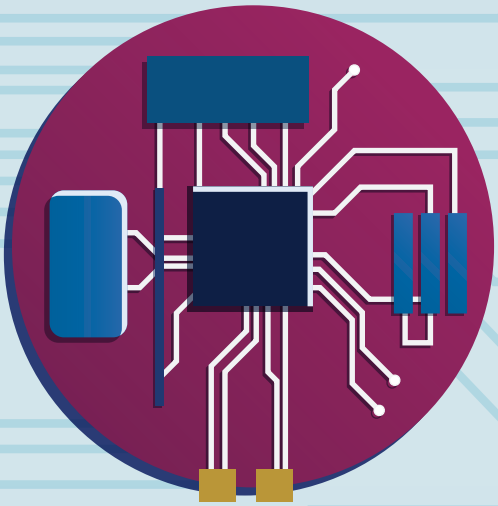
Embedded COMPUTING DESIGN

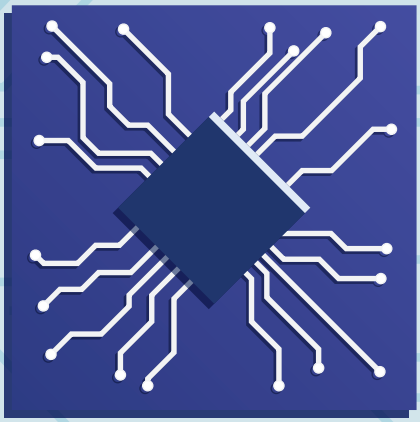
Connecting Silicon, Software, and Strategies for Intelligent Systems

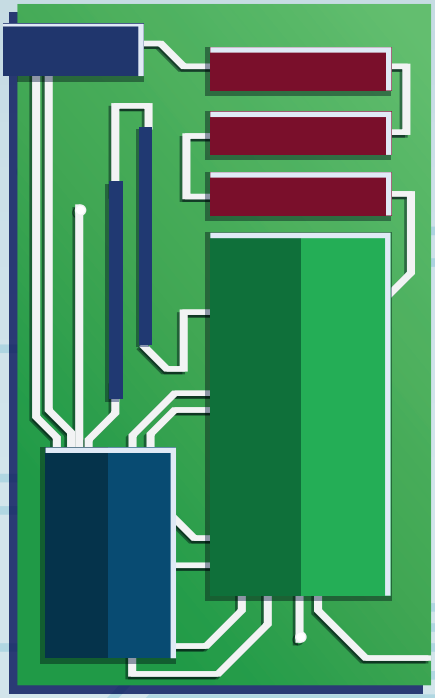
DIY ISSUE
PG. 29

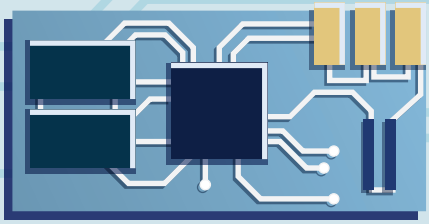
DEVELOPMENT KITS: Low-cost, quick, and easy

PG. 12









PLUS

DIY CORNER

WINDOWS 10 FOR IOT ON THE RASPBERRY PI 2
PG. 8

SOFTWARE

A SURVEY OF OPEN NETWORKING STANDARDS
PG. 20

Experience extraordinary training for embedded systems professional at ESC Conference 2015!

The **Embedded Systems Conference (ESC)** is the industry's largest, most comprehensive technical conference for embedded systems professionals in the U.S.
ESC is excited to announce its return to Midwest.

ESC

EMBEDDED SYSTEMS CONFERENCE '15

MINNEAPOLIS

NOVEMBER 4-5 • MINNEAPOLIS CONVENTION CENTER

ESC Minneapolis, known for strong mix of manufacturing and hi-tech companies, is offering extraordinary training for embedded systems professionals. Join us and learn about the latest techniques and tips for reducing time, cost, and complexity in the embedded development process.

Get your Conference Pass today and enjoy access to Conference Sessions, Network Receptions, Demo Hall, Vendor Sessions and six industry expos going on concurrently at the Minneapolis Convention Center.

Come join us at the Minneapolis Convention Center and check out some of our sponsors for the event:

Rohde & Schwarz | Green Hills Software | Infineon Technologies North America Corp.
Lauterbach | Symmetry Electronics | WolfSSL | Ironwood Electronics | Teledyne Lecory
Rigol Technologies | Altium Inc. | Code Corporation | AMP Display

Register today at **www.embeddedconf.com/minneapolis** with promo code **OSM15MN** and get 15% off a Conference Pass. The offer ends October 15, 2015.

UBM

A history of development kits and community support

By Rory Dear, Technical Contributor *rdear@opensystemsmedia.com*

Back as a fresh-faced 18-year-old in his first days in the embedded industry, I quickly learnt students and hobbyists were considered little more than an inconvenient drain on support resources, be that commercial or technical. In fairness this was at a time when embedded computing technology remained a specialist niche and well before the term or even the philosophy of "ease of development" was conceived.

Hobbyists were viewed as motivated by paying as little as possible and placing as much a burden of support as they could on the respective supplier of their "toy" and of course only wanting one. Students alike were perceived as a drain on technical support, leaning on any resource they could justify to aid their learning with no awareness or care for their encumbrance.

Back then, a PC/104 card was the very height of technology and initial samples invariably were only available packaged within a £995 development kit, always a peculiarity for me as the PC/104 module itself, surely the most expensive component, only accounted for a quarter of this cost. Manufacturers of such exorbitant development kits put heavy emphasis on the elevated cost of the "custom" I/O cables bundled within such a kit, which of course the PC/104 card couldn't function without as all I/O signals were accessible purely through pin headers.

The reality is that such cabling does not extend to hundreds of pounds; the murky secret is the remainder of that cost was an up-front charge for development technical support, whether you used it or not. Those proficient in embedded computing technology would baulk at these costs, but unless you wanted to take it upon yourself to make your own custom cables at the hardware evaluation stage, you were left with no choice.

This practise of "pre-charging" for support continued for some time, then disappeared altogether as the industry recognised its short-termism. In business, the instant gratification of a sizeable purchase order today is, especially by the bean counters, considered more important than an even larger one in X years' time. The passage of time, during which those students graduated and became design engineers and key decision makers even further up the chain, started to cause shockwaves.

Those manufacturers who had "shunned the little guy" were now being shunned themselves and in a very big way. Those students and hobbyists that were priced out of the market by unaffordable development kits, or companies that spurned and belittled their support requests at critical stages of their fledgling engineering career, became brands permanently tarnished in that key decision maker's eyes.

The embedded industry had an epiphany and swiftly removed those barricades. Gone were the prohibitively expensive development kits and cables became mass produced and bundled with every product, so each product was effectively that development kit of old. The negativity and reluctance to satisfy and support those who weren't going to treble turnover today, but may well tomorrow, shifted into one of genuine long-termism, recognising the massive future influence he may yield.

Simultaneously, product designers, who were once proud of the complexity of the elitist embedded engineering faculty, recognised that same attitude was directly why so much support was needed. The embedded market began to recognise the advantages of what could be termed an "abstraction layer", hiding the necessary complexity behind a wall of relative simplicity and enabling a far wider audience to get involved in pushing innovation.

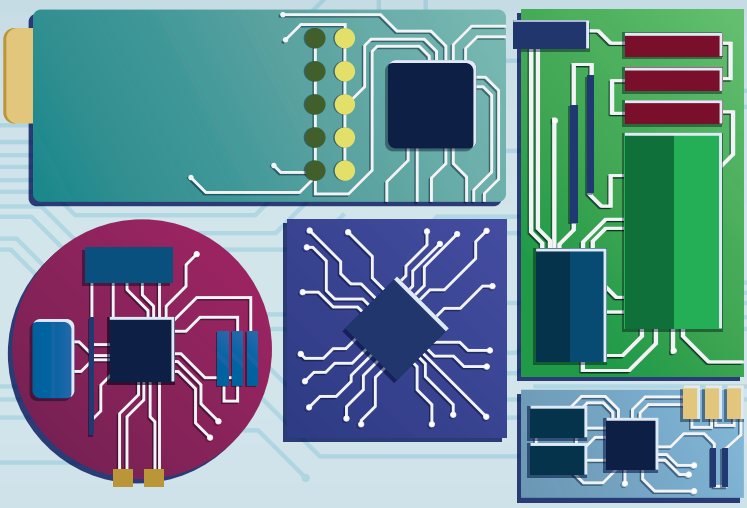
That wider audience of course included those students and hobbyists, so today's embedded manufacturers, well aware of the importance of tomorrow's customers, will not make the mistakes of their forefathers. You'll see evidence of this at any embedded exhibition, with effortless development suites screaming loudly from industry behemoth's stands – with many even offering free editions of historically expensive software to get them on board at an early age.

Not only are today's manufacturers willing to support, but often they don't even have to. The emergence of the Internet as a key resource led to manufacturers setting up online communities where avid developers literally help each other. Not only does this push brand awareness and search engine rankings, but also fosters the community spirit that we see in our industry in 2015. ECD

Embedded COMPUTING DESIGN

SEPTEMBER 2015
VOLUME 13
#6

embedded-computing.com | @embedded_comp | opsy.st/ECDLinkedIn



12

Departments

Silicon

Software

Strategies

Special Features

APP EXTRAS

Download the Embedded Computing Design app:
iTunes: itun.es/iS67MQ
Kindle Fire: opsy.st/kindlefireamaz

ADVERTISER INDEX

Get your free digital edition at embedded-computing.com/emag

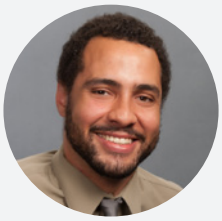
Subscriptions:
embedded-computing.com/subscribe
subscriptions@opensystemsmedia.com

OpenSystems media.

2015 OpenSystems Media ®

ISSN: Print 1542-6408, Online: 1542-6459

ENVIROINK™
The inks used to print the body of this publication contain a minimum of 20% by weight, renewable resources.



Bringing up Sensor Puck shows semis doing more for less

By Brandon Lewis, Assistant Managing Editor *blewis@opensystemsmedia.com*

Step 1: Turn on Sensor Puck.

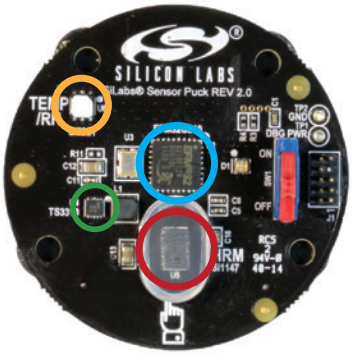
Step 2: Download "Silicon Labs Sensor Puck" app.

Step 3: View environmental and biometric readings.

That's it. I'm free to explore.

This is more than a passing trend – it has become an embedded development de facto. More and more we see vendors like Silicon Labs wrapping their expertise in microcontrollers, analog sensors, multiprotocol radios, and associated communications stacks and processing algorithms into neat little packages that allow app developers to begin adding value after the simple flip of a switch – and doing so at sub $30 price points. Of course, if you're the hands-on type that wants to modify firmware, source code is typically provided free of charge in these kits and can be accessed through development environments that are available at little or no extra cost (in the case of Sensor Puck, Silicon Labs' Simplicity Studio can be downloaded from their website, and a Segger 9-pin ARM Cortex debug cable and MCU evaluation board are both available from Mouser). In some cases, vendors even go so far as to set you up with working applications, such as the heart rate monitor based on the Sensor Puck's integrated optical sensor.

Sensor Puck



- Si7021 Temperature/Humidity Sensor
- Si1147 Optical Sensor
- TS3310 Boost Converter
- EFM32 Energy Friendly MCU

Figure 1 The Silicon Labs Sensor Puck is outfitted with the Si7021 Temperature/Humidity Sensor (yellow), Si1147 Optical Sensor (red), TS3310 Boost Converter (green), and EFM32 Energy Friendly MCU (blue).

So voila! What once consumed a majority of the product development life cycle has now been abstracted down to a couple of minutes, most of which is consumed during the mobile app download and install.

More for less

Short of magic (though the heart rate monitor is pretty damn cool), the advent of the Internet of Things (IoT) is changing business models and prompting semiconductor suppliers to do more for less. "The next big thing" could come from anywhere, and rather than waiting for it to show up on their doorsteps, silicon vendors are getting proactive by pushing their technology out into the world in the hope that it ships in some high-volume end product. If reducing barriers to entry means assuming a level of risk, so be it.

As Ross Sabolcik, VP and GM of Silicon Labs' Analog, Power, and Sensors division explained in a meeting at the company's Austin headquarters, the level of competition in high-volume markets means semicons have to provide a range of IP and offerings that are suitable for multiple applications, and what better way of illustrating these capabilities than through an agnostic development platform that showcases a microcontroller, two sensors, and a power management IC in the same package (Figure 1)? Silicon Labs' acquisition of wireless connectivity shop Bluegiga earlier this year also allows the addition of in-house connectivity to the Sensor Puck and other dev kits via modules such as the Blue Gecko Bluetooth Smart SoC, meaning the company can tightly integrate and optimize the entire solution so developers don't have to.

I am not an engineer. But anymore, how much do you have to be? ECD

Leveraging the (free) power of Windows 10 for the IoT on the Raspberry Pi 2 and other development kit platforms

By Monique DeVoe, Managing Editor *mdevoe@opensystemsmedia.com*

Inexpensive development kits and free/open source software and tools are making project development cheaper and easier than ever. The low-cost dev kits and open source tools have been around for a while now, but one thing I never expected to hear this year is that the new version of Windows would be free! Microsoft seems to be getting into the maker community spirit (and embedded IoT development space) by releasing its Windows 10 IoT Core operating system (OS) for maker platforms – free.

The IoT Core is an edition of the Windows 10 OS for low-cost, small-footprint embedded devices and is compatible with the Raspberry Pi 2, Minnowboard Max, and Galileo boards. This seems like a good fit as a lot of maker/dev kit projects are IoT applications at least in part, and Microsoft's established suite of development tools can be a real help to developers.

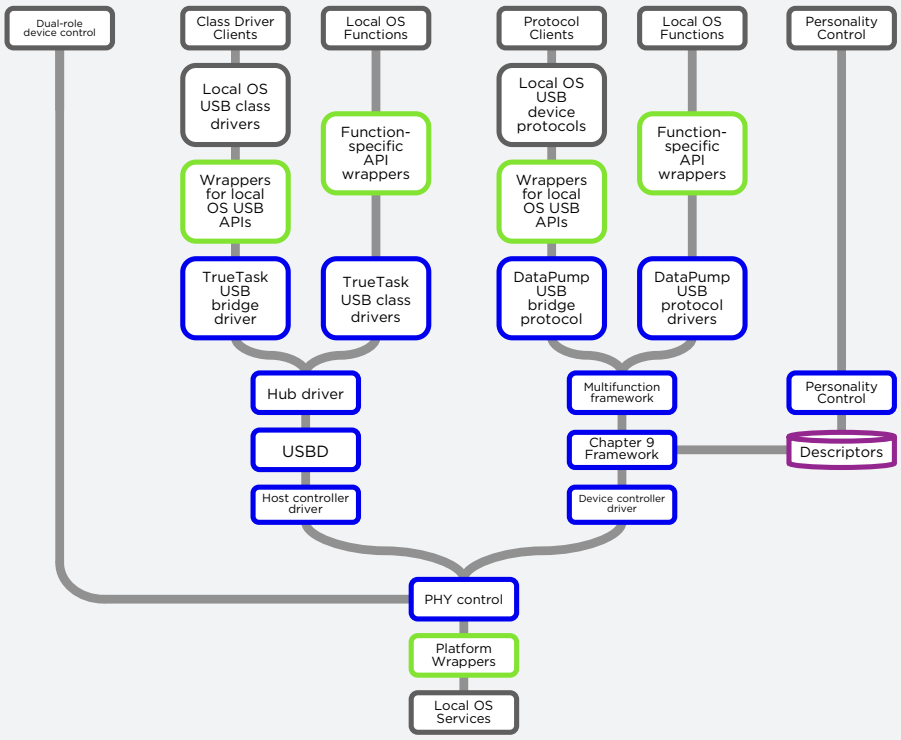


Figure 1 The MCCI TrueTask USB stack and device block diagram.

In particular with the Raspberry Pi 2, this is the first time Windows will be available to use with the platform, and it took a bit of work to get there. Windows hasn't historically been compatible with Raspberry Pi's Broadcom chip and Synopsys's DesignWare USB IP block – and let's face it, USB is essential for connectivity and it'd be hard to get very far without USB functionality.

Terry Moore, CEO of MCCI (www.mcci.com), shed some light on developing the USB stack to allow Windows 10 to work with the Raspberry Pi 2 hardware in a recent conversation.

"Synopsys's DesignWare USB 2.0 IP core is great for embedded systems, but it's not software-compatible with any of the USB controllers that are used on desktop and notebook PCs."

PCs traditionally use EHCI and XHCI register models for USB host functions. MCCI has worked with both EHCI and XHCI and the Synopsys core before, and combined its knowledge of the two to create a TrueTask USB stack to work with Windows 10 IoT Core OS and the Broadcom chip on the Pi 2 (Figure 1).

"We could just swap out the XHCI code that was already running on Windows and plug in the code for the Synopsys core, and we were 90 percent of the way there," Moore says.

Even so, there were still some development challenges to get a traditionally desktop-focused operating system to work with a more embedded-focused core on the Raspberry Pi 2.

Windows class drivers assume Intel x86 cache coherence semantics, where cache coherency is managed by the hardware, and software doesn't play a big role. However, with ARM, cache coherency is managed by software. See a problem with that?

"We found some issues with class drivers that were perfectly correct from an x86 point of view, and perfectly correct if you just looked at their code, but not correct when you combine that with the actual way that the cache happens to work on this system," Moore says.

MCCI developers also had to work with modes of the Synopsys core they haven't used before in order to meet requirements, which impacted performance.

"Scatter gather can't be used, and in a virtual memory system scatter gather is pretty important," Moore says. "The Synopsys core is really intended for a situation where you have a dedicated processor, so you can make real firm statements about what the response time on interrupts is going to be. Although the Windows kernel is really pretty good, because of the architecture of the Broadcom device we're sharing our interrupt with a couple of other drivers. It turns out that the interrupt responsive time is not ideal."

This problem can be demonstrated in the case of running a USB webcam, and how what Windows expects and what you get with the Raspberry Pi 2's core don't quite match up.

"With this hardware running a webcam, we have to take an interrupt every 125 microseconds; we have a hard real-time deadline to do our work and get out – if we slip up, we lose," Moore says. "The drivers above us are the standard Microsoft drivers designed for a desktop environment where things are, relatively speaking, infinitely fast. The application using the webcam might need only 9 Mbps because they're going to run in low res, but the class driver programs the webcam to give us those 9 Mb in 24 MBps bursts. During the 24 MBps bursts, we have a fair amount of work to do on every interrupt in order to keep up. This resulted in us having to do some reorganization of the code in order to make sure it would go that fast without having to give up portability."

Developers have been working out the bugs of the Windows 10 IoT Core operating system as a whole through the Windows IoT Core Insider Preview, and as it's being developed Microsoft has posted a lot of projects to show what these dev kits are capable of when paired with Windows 10.

One project uses speech recognition for a home automation proof of concept. A USB feature supported by Windows 10 IoT Core for the Raspberry Pi 2 was USB audio support, which allows for speech recognition so the user can turn different devices attached to the Pi 2 on and off. Other projects help with getting basics set up with the Pi 2 and Windows 10 IoT Core like adding buttons, LEDs, basic speech recognition and synthesis, displays, and more to get developers familiar with using Windows 10 on their dev kits.

It'll be interesting to see how much Windows as a maker platform takes off as well as how popular it'll become for IoT applications as it gets more established. If you've worked with Windows IoT Core we'd love to hear about your experience and your projects! ECD

For makers and professional developers looking to use Windows 10 IoT Core for their projects, visit WindowsOnDevices.com for setup instructions, projects, downloads, and more.

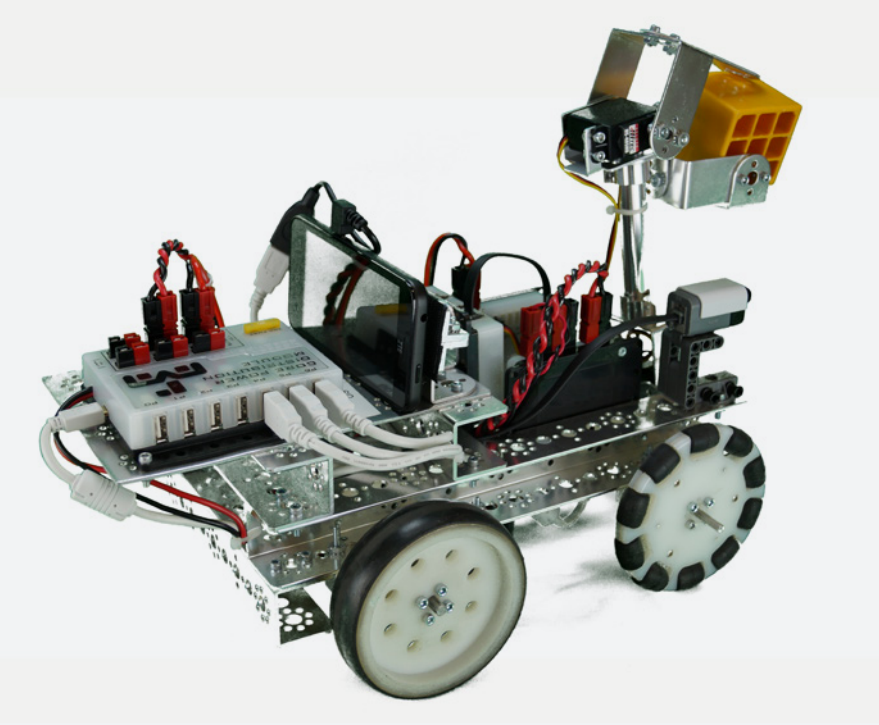
Robotics competition teaches the skills and mindset to be a great engineer

By Monique DeVoe, Managing Editor *mdevoe@opensystemsmedia.com*

Robots are a fun way to introduce practical, hands-on engineering with visible results that keep budding engineers coming back for more. And with the right platform it can be a valuable experience into tools they'll use for future engineering careers. This is exactly what the international K-12 not-for-profit organization *FIRST* (www.usfirst.org) has set out to do.

A kids' sport for the brain

FIRST (an acronym of "For Inspiration and Recognition of Science and Technology"), founded by inventor Dean Kamen, has been fostering interest and participation in STEM for more than 25 years. The organization is split into four mentor-guided programs that offer competitions for students of different ages, from kindergarten to Grade 12, with a projected 400,000 participants in all programs in the 2015/16 competition year.



FIRST, an international K-12 not-for-profit organization founded by inventor Dean Kamen, partnered with Qualcomm to offer a Java-Based Android Platform for students competing in FIRST Tech Challenge. Students will use the platform to program their robots this fall for the annual FIRST Tech Challenge competition season. For more information or to start a FIRST Tech Challenge team, visit: www.usfirst.org/roboticsprograms/ftc/start-a-team

Participants compete in teams where they develop a strategy and build a robot using engineering principles including rapid prototyping and iterative design. They're also taught to respect, teach, and learn from fellow teammates and mentors, but still compete with a passion in a "Coopertition" environment.

Competitions are held in various arenas all over the U.S. and the world where teams complete in challenges to score points. For example, *FIRST* recently had a recycling theme for *FIRST* Robotics Competition, its high school program. Teams scored points by stacking items, capping the stacks with recycling containers, and disposing of objects representing litter.

Incorporating the latest technology

Many of the younger divisions use LEGO-based platforms and moving parts to construct their projects. But, for the first time in the 2015/16 season, the *FIRST* Tech Challenge teams (Grades 7-12) will use a Java-based Android platform and robot and driver-station controls powered by the Qualcomm (www.qualcomm.com) Snapdragon 410 processor (www.qualcomm.com/products/snapdragon/processors/410). Kamen believes the new Android and Java platforms will be powerful tools for *FIRST* participants.

"Android devices are everywhere," Kamen says. "The Android operating system dominates the mobile/tablet computing market with over 51 percent of the market. The Android platform, powered by Qualcomm Snapdragon processors, offers mobile computing power well above just about any other mobile platform out there. Most people use it for cellular communication only. *FIRST* Tech Challenge is tapping into that computing power for robotics."

Qualcomm's Director of Engineering Chad Sweet says using a common smartphone technology makes for a much lower barrier to entry than other educational robotics platforms.

"Most students have access to a smartphone these days, which makes the Snapdragon 410 processor-based Android platform ideal – students can simply take their phones out of their pockets and start programming for robots," Sweet says (See what devices use the processor here: www.qualcomm.com/products/snapdragon/devices). "In addition, this new system uses the Java programming language at its core, which is the same language high school students are using to learn programming, making it that much easier for students to start developing."

From kits to competition

FIRST Tech Challenge teams have a variety of options for building their robots. Rookie teams receive metal building sets that come with motors, gears, metal, batteries, and sensors, Kamen says. They also get two Android phone handsets and electronic interface modules that allow Android devices to talk to the sensors and motors. The switch in platform won't interrupt teams' building and progress, Kamen says, as they can use their existing equipment and simply replace the former communication technology with the new Snapdragon-powered devices and interface modules.

RoBowties is a well-performing five-student team based out of San Diego participating in the *FIRST* Tech Challenge. They've topped multiple competitions and competed in the 2014/15 World Championship in St. Louis as well as the Asia Pacific Invitational in Sydney, Australia. This globetrotting team's captain, Isabelle Ho, has been competing for four years now and is eager to try out the new Snapdragon-based system.

"I'm excited to try out the new technology," Ho says. "It will be a big change from what we've had, but it seems like once everybody gets the hang of it we won't have as many problems moving forward."

Through her time with *FIRST* and the *FIRST* Tech Challenge program, she's learned many things about engineering, from teamwork to screwing together parts to programming autonomous systems.

"The great thing about *FIRST* in general is the community we build through Gracious Professionalism," Ho says. "Every team is competing and learning. We all want to learn about each other's robots and we all speak the same language."

Industry support for future engineers

FIRST is supported by about 200 of the Fortune 500 companies as well as educational and professional institutions, foundations, and individuals who provide funding, mentorship, equipment, and volunteers. Colleges, universities, professional associations, and corporations also offer college scholarships to *FIRST* participants in Grade 12 ranging from $500 to covering a full four-year tuition.

Qualcomm has sponsored *FIRST* for more than eight years and finds *FIRST* to be a valuable program for developing engineers.

"We value *FIRST* and the work they are doing because they are devoted to helping young people discover and develop a passion for science, engineering, technology, and math," Sweet says. "As a technology company, we believe that by supporting STEM learning, and especially encouraging young people to pursue STEM in school and in their careers, we will empower a new generation of great innovators and inventors. Robotics is an exciting technology and *FIRST* is a great way to inspire new engineers and technology leaders." ECD

Low-cost embedded development kits accelerate design cycles

By Pushek Madaan and Gagan Luthra

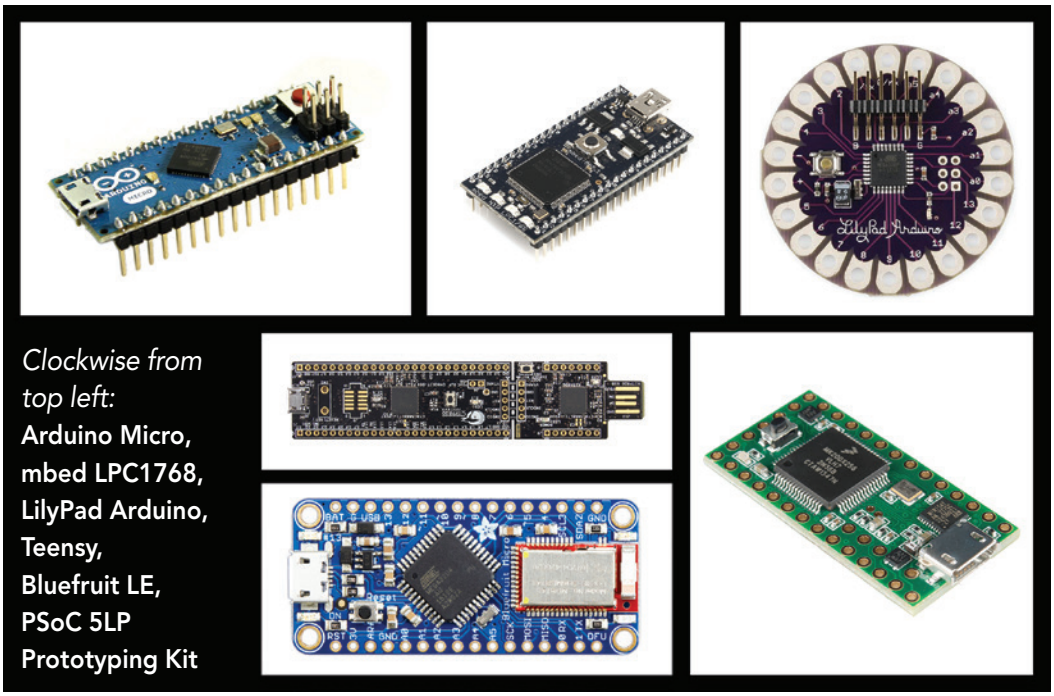
Increasing time-to-market pressures have led to the availability of full-featured microcontrollers, sometimes called system-on-chip (SoC) processors. These feature-rich microcontrollers or SoCs speed design by integrating a part of or, in some cases, all of the hardware required to implement an embedded design. Developers can then focus their design efforts on application-level functionality.

Greater integration, however, also leads to greater processor complexity. Depending on the microcontroller, it can be challenging for developers to quickly evaluate it for a particular application. To ensure that developers can evaluate the intended functionality of their microcontrollers, semiconductor manufacturers offer peripheral boards and basic development boards.

In the past, semiconductor manufacturers typically used a proprietary interface for their development boards, which required them to develop not only the basic development board for a microcontroller, but peripheral boards as well. As a result, these proprietary interfaces often limited how quickly a manufacturer could support new microcontrollers.

With the rise of the hobbyist market and various embedded developer communities, semiconductor manufacturers can now utilize popular standard interfaces such as Arduino, Raspberry Pi, and mbed for their development boards. These standard interfaces enable customers to easily switch between one development board and another without having to worry about changing peripheral boards. They also enable the embedded solution suppliers to provide a wider range of peripheral boards that can work with different development boards based on the same standard interface.

Today, developers can use these standard-based development boards for their initial evaluation of a microcontroller. However, when it is time to prototype their design, customers have to build their own hardware modules, as the form factor and cost of these development boards often restrict their usage in a prototyping environment.



Clockwise from top left: **Arduino Micro, mbed LPC1768, LilyPad Arduino, Teensy, Bluefruit LE, PSoC 5LP Prototyping Kit**

To ensure developers can evaluate their silicon in a prototyping platform, silicon manufacturers and other embedded solution suppliers have begun to offer low-cost development boards in an easy-to-use form factor based on standard interfaces. This enables the engineers to

not only evaluate the microcontroller but also prototype their embedded designs using the same development board.

These development boards often come with a generic breadboard-compatible interface, enabling the engineers to design these boards into their desired prototyping platform. These development boards not only allow developers to prototype their designs more easily, but they also enable hobbyists to create their next project without having to worry about creating a new platform for their application.

There are a variety of such prototyping-ready boards available on the market today. Each offers a unique set of features including smaller form factor, a breadboard-compatible interface, and an onboard programmer and debugger (Table 1).

Arduino Micro

The Arduino Micro (www.arduino.cc/en/Main/arduinoBoardMicro) is a development board developed by Arduino and Adafruit. This tiny development board is based on the ATmega32U4 microcontroller and comes in a breadboard-compatible form factor, which makes it easy to interface with. It also exposes 20 digital input/output (I/O) pins and features a built-in USB to communicate with a PC on its own.

The Arduino Micro comes with a pre-programmed bootloader firmware, enabling users to download new firmware without having to connect an external programmer/debugger.

Bluefruit LE

The Bluefruit LE (www.adafruit.com/products/2661) is a prototyping board developed by Adafruit. This board features an ATmega32u4 interfaced with a Nordic Bluetooth Low Energy radio (nRF51822) over a serial peripheral interface (SPI). This breadboard-compatible development board exposes 20+ general-purpose I/O (GPIO) pins for application development. These GPIO pins support I2C, SPI, a universal asynchronous receiver/transmitter (UART), and 6 analog inputs.

Pre-loaded with a bootloader project, the Bluefruit LE lets users program their applications without the need of an external programmer. The Bluefruit LE board also features an iOS and Android app that can be used to communicate with the Nordic Bluetooth Low Energy module.

LilyPad Arduino

LilyPad (lilypadarduino.org) is a unique development board designed by Arduino for wearable applications. This board can be sewn on clothes using conductive threads. This board is based on the ATmega168 microcontroller and exposes a total of 20 I/Os (14 digital I/Os and 6 analog I/Os), which can be used to interface with a variety of digital or analog sensors.

Similar to any other Arduino platform, LilyPad Arduino comes pre-programmed with a bootloader. Users can download new firmware via Arduino software.

mbed LPC1768

The mbed LPC1768 development board (developer.mbed.org/platforms/mbed-LPC1768) is based on an ARM Cortex-M3 processor, which has a rich set of peripherals including USB, Ethernet, CAN, and serial protocols. This development board also comes in a breadboard-compatible form factor and exposes 32 I/Os.

The mbed platform supports programming of the target device via standard CMSIS-DAP or the mass-storage programmer mode, where users can simply drag and drop a binary file to program the device.

The mbed is a web-based platform that allows users to write programs in C++, providing access to a wide variety of mbed libraries.

PSoC 5LP Prototyping Kit

PSoC 5LP Prototyping Kit (www.cypress.com/documentation/development-kitsboards/cy8ckit-059-psoc-5lp-prototyping-kit) is a low-cost platform by Cypress that enables users to

Features	Arduino Micro	Bluefruit LE	LilyPad	mbed LPC1768	PSoC 5LP Prototyping Kit	Teensy
Processor	ATmega32U4 8-bit AVR 16 MHz 32 KB Flash 2.5 KB SRAM 10-bit ADC	ATmega32U4 8-bit AVR 16 MHz 32 KB Flash 2.5 KB SRAM 10-bit ADC nRF51822 32-bit ARM Cortex-M0 16 MHz 256 KB Flash 32 KB SRAM Bluetooth Low Energy Radio 10-bit ADC	ATmega168V 8-bit AVR 20 MHz 16 KB Flash 1 KB SRAM 10-bit ADC	NXP LPC 1768 32-bit ARM Cortex-M3 100 MHz 512 KB Flash 64 KB SRAM Ethernet, USB, CAN, DMA 12-bit ADC, 10-bit DAC	CY8C5888LTI-LP097 32-bit ARM Cortex-M3 80 MHz 256 KB Flash 32 KB SRAM 24-bit DFB, 24x UDB, CAN, DMA, 4x Opamp 20-bit DelSig ADC, 12-bit SAR ADC, 4x Comparator CY8C5868LTI-LP039 32-bit ARM Cortex-M3 67 MHz 256 KB Flash 32 KB SRAM 24-bit DFB, 24x UDB, CAN, DMA, 4x Opamp 20-bit DelSig ADC, 12-bit SAR ADC, 4x Comparator	Freescale MK20DX256 32-bit ARM Cortex-M4 72 MHz 256 KB Flash 32 KB SRAM CAN, DMA 16-bit ADC, 12-bit DAC
Programming	Bootloader	Bootloader	Bootloader	Programming and debugging	Onboard programming and debug	Bootloader
Interface	Breadboard compatible	Breadboard compatible	Custom interface	Breadboard compatible	Breadboard compatible	Breadboard compatible
Number of I/Os	20	20	20	32	48	24
Development platform	Arduino	Arduino	Arduino	mbed web IDE	PSoC Creator	Teensyduino
Language	C	C	C	C++	C, Verilog	C
Cost	$ 24.90	$ 26.95	$ 19.95	$ 54.95	$ 10.00	$ 19.80

Table 1 A prototyping board comparison.

prototype designs with the ARM Cortex-M3 PSoC 5LP device. The PSoC 5LP device features a 24-bit hardware digital filter block (DFB) coprocessor, CPLD-based Universal Digital Blocks, high-performance direct memory access (DMA) controller, high-precision 20-bit analog front ends (AFE) with Programmable Analog Blocks including opamps, programmable gain amplifiers (PGA), filters, comparators, successive approximation register (SAR), and Delta-Sigma analog-to-digital converters (ADC), and the CapSense capacitive touch-sensing solution.

This prototyping kit comes in a breadboard-compatible form factor and exposes all of its 48 I/Os for application development. This board features a unique snap-away onboard programmer and debugger, based out of another PSoC 5LP device. This onboard programmer and debugger can be used to program and debug the target PSoC 5LP device and also supports serial communication protocols. The onboard programmer and debugger can be bootloaded to implement any custom applications.

Teensy

Teensy 3.1 (www.pjrc.com/teensy) is a breadboard-compatible prototyping-ready development board that exposes 24 I/Os of the onboard microcontroller unit (MCU). It features a Freescale ARM Cortex-M4 MCU device that supports 256K flash and peripherals such as 12-bit DAC, dual ADC, and CAN.

This development board comes pre-programmed with a USB bootloader and supports application development via Teensyduino, an add-on for Arduino IDE.

Faster, cheaper development

Today's silicon manufacturers and embedded suppliers provide low-cost breadboard-compatible development boards to enable developers to both evaluate and prototype their designs using the same hardware. With some of these development boards costing as little as $10, the decision whether to design a custom development board or use an off-the-shelf, breadboard-compatible, ready-made development board is no question. The convenience of being able to have several prototyping-ready boards available in the development workspace simply makes design the next disruptive electronics product much easier.

Pushek Madaan is Senior Product Marketing Engineer at Cypress Semiconductor.

Gagan Luthra is Product Marketing Manager at Cypress Semiconductor.

Cypress Semiconductor
www.cypress.com
@CypressSemi
www.linkedin.com/company/cypress-semiconductor
www.youtube.com/user/cypresssemi

Software is key differentiator for IoT dev kits

By Mark Rootz

Off-the-shelf development kits have become a keystone for many Internet of Things (IoT) developers, as their integrated hardware, software, and connectivity serve as the platform for engineers looking to design connected products quickly and inexpensively. However, though these kits are intended to provide a head start that allows designers to focus on value-added features, considering the long-term consequences of open source software, software licensing, and product differentiation when selecting a development kit is critical for successful IoT products, with implications that span from initial development through product launch, market adoption, and life cycle maintenance.

The Internet of Things (IoT) market offers unprecedented opportunity. The numbers alone are staggering. Analysts at Gartner Inc. estimate that 4.9 billion connected devices will be in use by the end of 2015, up 30 percent from 2014. Five years from now, they expect that number to increase to 25 billion. By then, Gartner analysts expect 10 billion connected devices (excluding PCs, smartphones, and tablets) will ship each year into a market that researchers at IDC forecast to be worth over $7 trillion.

However, many of those devices will be based on previously unconnected product designs (home appliances, building automation equipment, etc.). This will present a challenge for the engineering teams designing them as connected devices, as they'll be utilizing a variety of technologies (wired and wireless connectivity, security, cloud, etc.) that will increase the overall complexity of the design. What's more, many of the embedded developers building IoT devices don't have experience with these technologies. Adding the necessary expertise via additions to the team's headcount or spending the time necessary to train existing engineering resources is not an option available to most design teams. And yet these new products must be developed quickly and cost effectively if they are to be successful in the market.

To shorten time to market, embedded hardware vendors that supply microcontrollers, sensors, analog acquisition, and low-power wireless solutions have created new development kits for their customers. These kits usually bundle the target hardware along with software (RTOS, stacks, and middleware), often sourced from multiple vendors. These kits have become instrumental tools to aid in the development of embedded solutions (Figure 1).

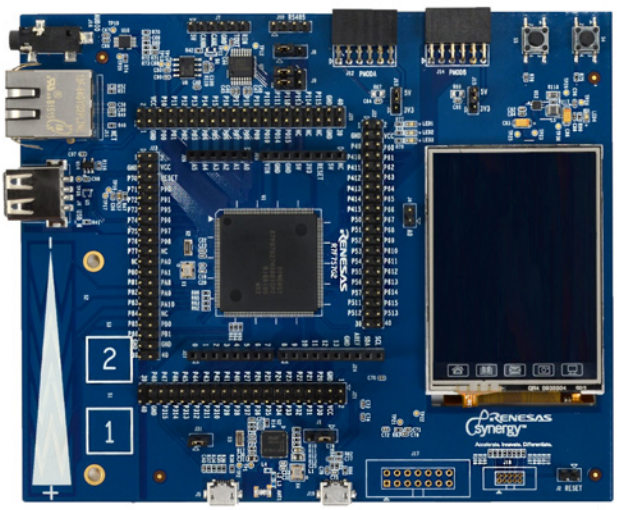


Figure 1 Development kits with ample hardware support for connectivity and user interfaces to enable the use and development of a complete software bundle bring significant value as a tool for embedded design work. (Source: Renesas Electronics America Inc.)

Three software setbacks of the traditional dev kit

How essential are development kits and design examples to the electronics design and production process? According to a recent worldwide survey of electrical engineers by element14 Pty Ltd, four out of five respondents believe that development kits have become a crucial tool to take designs to the end product stage (see page 30). Of those, most use all or part of their kits in their final production design. Moreover, three out of four survey respondents believe that kits play a critical role in driving innovation. Yet, the traditional development kit model is not a good fit for the IoT market for several reasons.

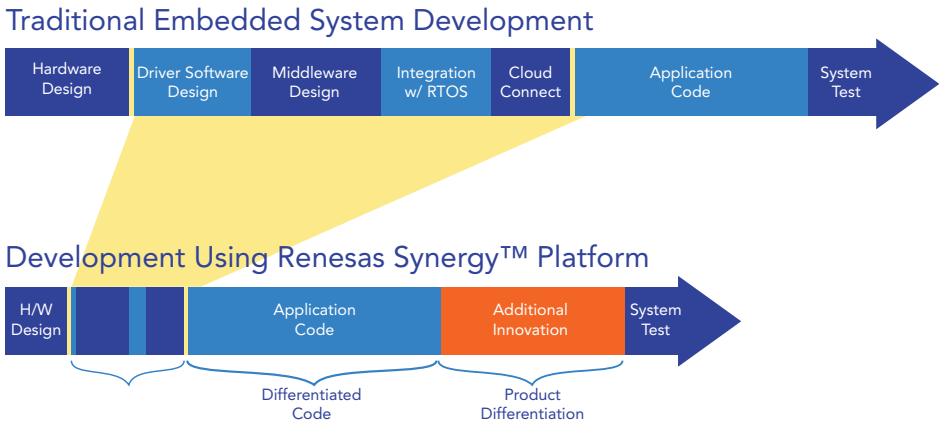


Figure 2 Leveraging kits and a platform-based approach like Renesas Synergy allows designers to focus their resources on their own end-product differentiators with confidence that the underlying essential system code is solid. (Source: Renesas Electronics America Inc.)

First, the software bundled with most development kits is usually packaged as a free or low-cost extra. And while this may save cost initially, in the long run it can actually cost more in terms of lost design time and reduced reliability. Bundled software included in a development kit is likely to have gone through minimal compatibility testing and is often not eligible for access to ongoing upgrades or bug fixes. This could lead to trouble during development if bugs or conflicts occur. Furthermore, support capabilities can vary greatly between different software vendors, and inconsistencies in quality of products and documentation can cause unacceptable delays in product development. In the event of compatibility issues between various software components and/or hardware, oftentimes it is unclear who is responsible for fixing the bugs – vendor A or vendor B – and precious time is wasted trying to determine who has the responsibility of fixing the problem. Additionally, many bugs don't appear until a new product is in the field, oftentimes months or years after its deployment. If that happens, will the vendor responsible still be in business and capable of providing a fix? What if a bug affects multiple customers and the vendor's support resources become overwhelmed?

Second is the issue of product differentiation. Many embedded software platforms offer a variety of features (connectivity, user interface, graphics, etc.) and support various software protocol stacks and middleware. These features are available to every other design team using the platform, so by themselves they don't provide any unique value add that a design team can leverage to differentiate their product from the competition.

Licensing is a third consideration. Software bundled with a development kit is typically licensing free when building a prototype, but when that prototype goes into production, software licensing fees become necessary and often add up to a substantial amount of investment both upfront and over the product lifetime. Some developers may argue that the answer to this problem is to avoid licensing fees altogether and explore open source software alternatives. This is a solution, but open source software often has hidden costs. For example, bugs or compatibility issues with open source software still require fixes, and most vendors won't be able to wait for the open source community to solve the problem. They'll either need to purchase support from a third party or develop a fix on their own. In the long run, paying up front for a software license (and access to technical support from an established vendor) may be the less costly alternative.

The path to productization

Looking at the design challenges described above, it becomes clear that much of the work around software and hardware configuration, debugging, and testing will need to be done up front by vendors. This will allow IoT product designers to focus less time on simply getting a device to function (to send that first Ethernet packet or display that first animated widget on a color display) and more time on end-product differentiation, which is ultimately what will make their product successful in the market. The way to do this is through the use of development kits that fully integrate the IoT platform's software and hardware, have been fully tested and qualified to written specifications for operation and compatibility, provide ongoing access to software updates and bug fixes, and offer detailed but easy-to-navigate technical documentation (Figure 2). Perhaps most importantly, the kit's vendor should serve as the sole point of contact for customers when it comes to technical support, software updates, and software licenses.

Mark Rootz is Marketing Director, IoT Business Unit at Renesas Electronics.

Renesas Electronics Corporation
- am.renesas.com
- @RenesasAmerica
- www.facebook.com/RenesasAmerica
- plus.google.com/+renesas/posts
- www.youtube.com/user/RenesasPresents

Flexible framework architecture and IoT

By Curt Schwaderer, Editorial Director *cschwaderer@opensystemsmedia.com*

Traditional networked embedded systems architectures are being challenged as they begin to be used within IoT applications. These embedded systems are required to interoperate with sensors, devices, gateways, and cloud applications. This article provides an overview of IoT flexible framework architecture as discussed in a recent panel.

Architecture and IoT

Internet of Things (IoT) applications build on a previous generation of networked embedded systems by adding sensors, Internet gateways, and cloud applications. IoT increases the flexibility and power of networked embedded systems by providing Internet on-ramps and end-to-end access.

As more IoT applications emerge, architectures need to be flexible enough to adapt to varying requirements and feature sets where the functionality may be located anywhere along the IoT components. Can an IoT architecture hold up? Should IoT systems ignore architecture in favor of higher flexibility? Should they stick to an architecture and live with the limitations? Is there room for a mix?

These questions were posed to a panel consisting of IoT experts from Atollic, Freescale, Kontron, ThingWorx, and Wind River. Each had some unique insights, examples, and use cases that compare, contrast, and incorporate flexible framework architectures.

IoT framework, process, tools

Stephen Martin, Vice President and General Manager at Atollic (www.atollic.com) has a unique perspective on IoT flexible framework and the role development processes and tools play. Software architecture plays an important role, but where it lives should be flexible to fit the application.

"The software defines the framework you'll be living with for a while," Martin said. "IoT should start with a defined software development process."

Choosing a tools platform is also a critical step. It's important for IoT systems to have stable tools and advanced debugging capabilities. Martin mentioned that mixing open source software with in-house development can be an effective way to increase development cycles while adding differentiation to the IoT system.

Martin also warned to not forget that IoT applications involve powerful servers and storage. These elements can be used for development as well as for the application itself. IoT environments are uniquely suited for development, staging, and deployment, and taking advantage of this can effectively utilize resources and increase quality.

ARM TrustZones and Wi-Fi

Nick Sargologos, Senior Manager in the Digital Networking Group at Freescale (www.freescale.com) said that the

silicon plays a significant role in IoT systems from devices and gateways. The ARM TrustZone feature can provide partitioning for privileged applications or users. Intelligent monitoring and component management are also important features to consider within the IoT architecture. Sargologos referenced Wi-Fi as an IoT application in itself. End equipment interacting and communicating with access points, which in turn provide Internet connectivity, can indeed be considered an IoT application. Sargologos also discussed important initiatives where Freescale is aggressively partnering with open source initiatives involving IoT in order to provide a more integrated solution for IoT applications.

Identify guiding philosophy and key tenants

Prem Kumar is Vice President and Head of Global Technology at Kontron (www.kontron.com). Kumar had a variety of observations. First was to identify some guiding philosophies with the IoT application being developed. Then prototype with a purpose – understand what the use case is and put together some key functional components to demonstrate the concept. Analytics is an important component of an IoT system as well. Transforming data into actions is a common theme of an IoT application – gathering sensor information, feeding devices, and communicating through gateways to a cloud application might be one implementation. Analytics processing may also be valuable closer to the edge. Flexible frameworks enable designers to make the right design decisions for the use case. Kumar also discussed security, manageability, and agility as three key tenants to IoT flexible frameworks.

IoT down on the farm

John Canosa is the Chief Strategist for connected products at ThingWorx (www.thingworx.com). Canosa had some unorthodox use cases to share in order to drive home important points about IoT flexible frameworks. John asserted architecture and flexibility is not an either/or – in an ever-changing environment it's critical to lean on an architecture while having flexibility to adapt the solution to multiple use cases. Canosa used coal mining equipment to the concept of the "smart cow" to illustrate his points. Cattle monitoring is one case where many sensors are used to understand how cattle movement, environment, and feeding habits can provide more effective raising of cattle. He also mentioned chicken farms where coops are monitored for a variety of factors and the data are fed back to cloud applications to identify ideal conditions for laying and chicken health. In an era of high tech, Canosa brought an interesting and different use case for IoT (For more on embedded tech on the farm see opsy.st/FoodProduction).

Corruption and rollbacks

Jeffrey Fortin is a Director at Wind River. His take on architecture was a pragmatic one. Fortin recommends developing an architecture and then validating that architecture. Of course flexibility is a must, but if designers utilize resources to prove out the architecture, developing applications and validating use cases can happen more quickly. Fortin described a home patient monitoring system that describes an overall architecture where individual patient health and monitor sensors could be applied, implemented, measured, and validated. If the architecture holds, additional features and capabilities can be added to continue to prove out the flexibility and usefulness of the architecture.

A flexible IoT

Flexible framework architectures are a key element to the successful development and deployment of Internet of Things applications. As the experts illustrate, there are a lot of moving parts in this brave new world. Development of the architecture, building in flexibility, choosing the right tools and development processes, security, and proving out the architecture iteratively are all considerations for a successful IoT deployment. From smart cows to medical monitoring, flexibility and architecture must be carefully considered. ECD

Listen to the full discussion with Stephen Martin, Nick Sargologos, John Canosa, and Jeffrey Fortin at ecast.opensystemsmedia.com/560.

A survey of open networking standards

By Alex Henthorn-Iwane

Open source standards hold a lot of potential for networking. Already, initiatives such as OpenStack and OpenDaylight have gained a fair amount of momentum due to the contributions of major vendors such as Intel who recently joining OpenDaylight as a platinum member. That momentum is even spurring telecoms like AT&T to start their own open source efforts like the Open Networking Lab.

Various open networking standards are impacting telecom businesses, IT strategy, and network engineering. At a meta level, it's fair to say that due to the influence of open source development, networking teams are becoming more concerned with APIs, DevOps automation and the need to move away from manual toward automated ways of engineering and operating networks. Of course, each standard brings something different to the table.

OpenDaylight: The open SDN control plane project

The Linux Foundation manages OpenDaylight, the goal of which is to accelerate the development of software-defined networking (SDN) and network functions virtualization (NFV). Though not even two years old, OpenDaylight is already one of the largest open source projects in the world, with 266 developers contributing in the past year or so, and the project blowing past 1 million lines of code in early 2014.

What does OpenDaylight specifically address? It is perhaps best understood as offering an open control plane solution to enable multiple layers of SDN technology:

- The data plane layer consists of the physical and virtual devices in the network that speak various protocols northbound to the control layer. This includes but isn't restricted to OpenFlow.
- Above that is the OpenDaylight controller platform with northbound APIs to the application/orchestration layer, plus a multitude of southbound protocols and APIs exposed to the underlying data plane layer hardware
- At the top is orchestration and the application layer. This layer could look like a plug-in for OpenStack Neutron, for example.

OpenDaylight is trying to function as a sort of "Switzerland" of controllers, allowing compatibility with any type of switching hardware via many different protocols and interfaces such as OpenFlow, Path Computation Element Protocol (PCEP), NetConf, etc. The OpenDaylight controller runs on its own Java Virtual Machine and supports the Open Service Gateway initiative (OSGi).

Vendors and the future of OpenDaylight

A ton of networking vendors have piled onto OpenDaylight to such an extent that detractors are arguing that the project, despite its open source moniker, may end up actually preserving the position of networking incumbents. Cisco and IBM founded OpenDaylight and were followed as members by Brocade, Juniper, Microsoft, Intel, and others. A huge portion of OpenDaylight's code contributions are made by large networking vendors.

With OpenDaylight in particular, the contributions of Cisco have often been scrutinized due to Cisco's unique take on SDN and what type of controller should be used. Network World's Jim Duffy referred to Cisco OpFlex as an "OpenFlow killer" last spring, which may be an exaggeration, but it's clear that Cisco has something other than OpenFlow in mind for the future of OpenDaylight's core. OpFlex could be a key part of the Lithium release that will succeed OpenDaylight Helium. However, Guru Parulkar, director of the Open Networking Lab, cited OpFlex's exposure of device specifics to applications as a needless complication and one of the reasons for creating an alternative to OpenDaylight.

Of course, it's a difficult balance for vendors to achieve between innovating in their own product and service lines and participating in the open source community. Projects like OpenDaylight demonstrate the promise of open source software in enabling SDN orchestration, the "Wild West" nature of openness, and the risks of having incumbent vendors become the de facto new sheriffs in that Wild West landscape due to sheer muscle. A middle path is clearly needed.

"The vendor's role in open source networking projects should be a symbiotic one with other members of the community," wrote Mike Cohen for TechCrunch. "This includes contributing to open

source efforts for the good of the community while continuing to innovate and build differentiated products that better meet user needs. Vendors should also contribute some of those innovations back to the community to help promote new standards that will help the market continue to grow."

The takeaway

OpenDaylight, initiated by Cisco and IBM in 2013 and now hosted by the Linux Foundation, is not only a significant open networking standard but one of the most prominent open source software movements. Period. It defines an open controller with a wide set of southbound protocol APIs for different types of infrastructure and northbound integrations. The shape that vendor contributions to and relationships with OpenDaylight take will be critical both to its rapid progress in delivering features, but also to its future as an inclusive effort to improve SDN and NFV orchestration and network DevOps.

Open vSwitch: Open virtual switching

Open vSwitch is the virtual switch initiative under the Apache 2.0 license. Virtual switching isn't a new concept. VMware pioneered server virtualization, which allowed virtual switches to replace physical ones, meaning that a software stack running on a server that hosts VMs can run switching functionality that connects to virtual or logical Ethernet ports. The major difference between virtual switches such as Open vSwitch (OVS) and legacy virtual L2 bridges such as those included in Linux is that they are designed to handle highly dynamic cloud environments where network state (both configured and real time) may need to move between hosts using vSwitch instances.

While there are proprietary virtual switch solutions such as the Cisco Nexus 1000V and the VMware vSphere Distributed Switch (vDS), Open vSwitch is both the most prominent open-source alternative and an increasingly important fixture of other open-source networking projects. OVS works on Linux hypervisors such as Xen and KVM, is the default in Xen Cloud Platform and XenServer 6.0, and is integrated into OpenStack, which we'll look at in a subsequent part of this series. The kernel datapath module is also now included directly in Linux.

Designed to enable network control via OpenFlow and management via the Open vSwitch Database protocol, OVS functions as a soft switch and is also able to offload dataplane processing to switching silicon on network interface controllers (NICs) or external hardware switches. Just a few of its important features include:

- Support for tunneling protocols such as VXLAN and IPsec
- OpenFlow compatibility, including numerous extensions for virtualization
- Traffic policing for each VM interface.
- Use of the Link Aggregation Control Protocol (LACP) for link aggregation
- Compatibility with IPv6

The design of Open vSwitch

Open vSwitch is comprised primarily of a number of control plane components that live in the user space, plus the kernel module, which handles the actual data plane functions.

- **Ovs-vswitchd –** The most important component is Ovs-vswitchd, which runs the switch. It talks directly with the OVS kernel module via the netlink protocol. If an outbound packet handled by the kernel doesn't have a cache entry determining how it should be forwarded, the kernel messages the Ovs-vswitchd, which does a look up in the database to find a flow table entry that matches the packet in question. The forwarding instructions are messaged back to the kernel, which establishes a cache entry. Ovs-vswitchd can also communicate with OpenFlow controllers.
- **Ovs-dbserver –** This server supports the management plane functionality to Ovs-vswitchd, storing all configuration changes typically using OVS' OVSDB schema. It provides the JSON-RPC-based OVSDB protocol to external OVS clients that are used to configure the switch.

Unlike the Cisco Nexus 1000V or VMware's solution, OVS does not have a native SDN controller. It is designed instead to work with third-party controllers and cloud orchestrators, so OpenDaylight or the OpenStack Neutron OpenFlow and OVSDB plugins could be used.

Open vSwitch and SDN

OVS has been instrumental in the progress of SDN innovation, since it is both open and high performance.

"Open vSwitch is the most popular network back-end for OpenStack deployments and widely accepted as the de facto standard OpenFlow implementation," explained OVS contributors Justin Pettit, Ben Pfaff, and Ethan Jackson in a post for Network Heresy. "For Open vSwitch to be successful, it not only must be highly programmable and general, it must also be blazingly fast. For the past several years, our development efforts have focused on precisely this tension – building a software switch that does not compromise on either generality or speed."

OVS performance has dramatically improved over the past several releases. For example, features such as kernel cache megaflow support (based on field wildcarding) that allow the kernel to send fewer exceptions to ovs-vswitchd, and classifier improvements in ovs-vswitchd such as Priority Sorting, Staged Lookup, and Prefix Tracking, dramatically reduce the number of megaflows pushed into the kernel from millions to dozens. OVS 2.0 also made ovs-vswitchd a multithreaded process, and enabled better separation of real-time and management-plane tasks.

The takeaway

Open vSwitch is a prominent open-source project for Linux-based virtual switches. An alternative to Cisco and VMware solutions, it is a primary innovation platform and the recognized standard open switch for use with OpenFlow. Key features include its utilization of the OpenFlow control plane, its highly flexible OVSDB protocol for the management plane, its ability to offload dataplane handling to hardware, and its high-performance design and advancement over time.

OpenStack: Cloud computing software

Originally conceived by NASA and RackSpace, OpenStack is cloud computing software that uses logical abstractions so that users can get what they want from their infrastructure.

Included in OpenStack are various components that cover tasks such as compute, block storage, object storage, telemetry, networking, and bare metal provisioning. Although OpenStack hasn't historically been known for its ease of use (especially compared to alternatives in the public cloud such as Amazon Web Services (AWS) or Microsoft Azure), it has prominent backers and has been integral to deployments from firms such as Walmart, who shared at the recent Vancouver OpenStack Summit how they successfully deployed OpenStack at massive scale – 2,500 compute nodes run to support their 2014 holiday season. Quite an endorsement.

OpenStack overview: How the project became a huge presence in the cloud ecosystem

OpenStack is roughly five years old and is overseen by the OpenStack Foundation. As of December 2014, that body contained 111 official stakeholders at varying levels of membership. The top tier of membership – Platinum – includes heavyweights like AT&T, HP, Canonical, Red Hat, and co-founder RackSpace.

As its name suggests, OpenStack is open-source software. Its general functionality is perhaps best understood as that of an operating system for cloud computing. More specifically, users can provision compute, storage, and networking resources via a web interface. Using the OpenStack dashboard (the component called Horizon), developers can create virtual machines (VMs), configure networks, and manage volumes, all from a single location.

Red Hat executive Alan Ho nicely summed up OpenStack's appeal in July 2014 when he pointed to benefits such as:

- **Open-source collaboration –** The size of the OpenStack community allows for extensive knowledge sharing.
- **No lock-in –** OpenStack users can avoid becoming beholden to a particular vendor or proprietary solution.
- **Low costs and great agility –** Ideally, OpenStack minimizes the impediments to effective IT infrastructure automation, since it can be run on commodity equipment and be tweaked to the user's needs. In the Walmart OpenStack Summit talk, team members shared that by moving to generic server hardware running OpenStack their new servers are 50 percent cheaper and total operating cost has decreased by 300 percent compared to their previous modus operandi.

Of course, there are some drawbacks to OpenStack. Setup and migration have been traditional pain points. Organizations could avoid these bottlenecks by choosing a more straightforward public cloud option, but picking the right cloud infrastructure isn't always strictly a matter of convenience (or price, for that matter).

A recent report from 451 Research has estimated that OpenStack-related revenue could top $3 billion by 2018, expanding at a 40 percent compound annual growth rate (CAGR) between 2014 and 2018. Most of the revenue will likely go to OpenStack service providers, with smaller chunks going toward OpenStack distributions, DevOps orchestration tools, cloud management solutions, and platform-as-a-service (PaaS) on OpenStack. It's possible that there will be continued consolidation in the OpenStack community (i.e., in the mold of Cisco's acquisition of Metacloud) as big vendors look for a larger slice of this revenue.

OpenStack components and architecture

OpenStack has accumulated many new APIs and components as a result of rapid development over the years. The component list includes:

- Horizon: Dashboard
- Nova: Compute
- Cinder: Block storage
- Swift: Object storage
- Neutron: Networking
- Keystone: Identity
- Ceilometer: Telemetry
- Ironic: Bare metal provisioning
- Sahara: Elastic map reduce
- Heat: Orchestration
- Glance: Images
- Zaqar: Multi-tenant cloud messaging

OpenStack is a vast and evolving ecosystem. It faces significant competition from public cloud, as well as from arguably more mature open-source solutions such as CloudStack, but its image, branding, and scope mean that it is likely to be an important part of the cloud conversation for years to come.

The takeaway

OpenStack is an open-source operating system for cloud computing. Created by NASA and RackSpace, it provides logical abstractions for managing and orchestrating a network. It contains a variety of components that address functionality such as compute, storage, and identity. The size of the OpenStack community, which includes many prominent service providers and equipment vendors, makes OpenStack one of the most important open-source projects in the world.

Additional open networking standards

Read about a fourth open networking standard, OpenFlow, at opsy.st/OpenFlowStandard

Alex Henthorn-Iwane is VP of Marketing at QualiSystems.

QualiSystems Ltd.
www.qualisystems.com
@QualiSystems
www.linkedin.com/company/qualisystems
www.youtube.com/user/QualiSystems

HOW OPEN-SOURCE COLLABORATION IS TRANSFORMING IVI AND THE AUTO INDUSTRY

Dan Cauchy
General Manager of Automotive Grade Linux
Linux Foundation

Traditionally a highly secretive and competitive market, the automotive industry and its long design cycles are now confronted with consumer demands for smartphone functionality in the car. Dan Cauchy, General Manager of Automotive Grade Linux project for the Linux Foundation, discusses how the shrinking time-to-market expectations for in-vehicle infotainment (IVI) systems and features are compelling members of the automotive industry to collaborate on software standards for all aspects of the automobile.

Q Can you give us some background on Automotive Grade Linux?

Our marketing tagline is "Collaborating to build the car of the future through rapid innovation," and it's more than just a marketing tagline. The reason Automotive Grade Linux (AGL) exists is that the car manufacturers realized that what they were doing was not working. An infotainment system production cycle on average, according to some industry data, is up to 39 months from start to shipping in a car. In that 39-month period, three versions, maybe four versions of iPhone and Android phones have come out. So the car manufacturers realized that the way we're doing it now with black boxes and paying our suppliers to provide us something to our spec is just not working, and they needed to adopt a new way of doing things. This is essentially why AGL was created.

Basically, we're an open-source collaborative project. It's about enabling rapid innovation. Some people often think that open source is about saving money and it's about free, but actually that's not the motivation here. The motivation is more about keeping up and providing features for the end user that are at least on par, if not better, than the mobile phone experience. It's clear because you pay $1,000 or more for an infotainment system in your car, but let's face it, right now it's nowhere near the functionality of the mobile phone in your pocket. So that's what we're really trying to address here: rapid innovation, new features. We're building a reference platform that is 70–80 percent of the starting point for a production system, but our goal is not to make it a production system. We're not going to be in the business of testing this and fixing every bug. That's for a commercial entity to take the platform and bring it to production, either the car manufacturer themselves or they pay one of their suppliers to take the platform, customize it, add the user interface (UI) that they want, the look and feel, add the applications that they want, and then productize it and make it available to the end user.

That 70–80 percent we provide is all the common bits: the common operating system (OS), the common middleware, the common frameworks that everybody needs in a car – everybody needs audio support, everybody needs graphics, all these things. Why not just have all these companies share into the collaboration on this single platform and focus on the bits that matter, which is the differentiating part? So that's what AGL is all about.

The industry is looking for this because there's been a mixture of OSs and platforms out there. There was Windows for a while that wasn't super successful and is kind of fading away; QNX was the incumbent but is rapidly fading away because of the acquisition by Research In Motion (RIM). So people are looking for one standard platform, and if you're going to do that Linux absolutely makes sense because the rest of the consumer electronics industry is using Linux – TVs, phones, Blu-ray players, etc. So, that's why AGL is in the position to become the industry standard for all of automotive. And with the backing of Toyota,

the largest car manufacturer in the world, that helps a lot. It will help this get standardized.

Another thing in our charter is requirements specifications, so we have built a big specification document, and another big part of what we do is work upstream with other projects. For example, we don't reinvent the wheel, so if there's a Wi-Fi stack or a Bluetooth stack that exists out there we just reuse it, and if we make automotive-specific changes to it, we just push those changes back upstream to that project. That open-source development methodology is a big part of what we do.

Q Can you explain how AGL plays with GENIVI right now?

So, just so you know where I'm coming from, I used to be on the Board of Directors of GENIVI for three-and-a-half years, and I also was the Founder and Chairman of the GENIVI Compliance Program. The two organizations, first of all, are not competitive, and that's a misconception that some people sometimes have. GENIVI was founded under the premise that they would develop this compliance specification, and then anyone can bring their own platform and be compliant to the spec, but there is no one open source platform that people are working on. What people do is get a commercial platform from a Wind River or a Mentor Graphics, and then that platform goes through the certification process and then becomes compliant to the GENIVI spec. But it's always been a bring-your-own platform kind of approach.

AGL was started with an opposite kind of methodology, which is we are building the platform, which is the one you start with. You don't go off and build your own, you take AGL as the starting point, and it's open source and we're not for profit, so here it is, you take it, and you go build upon it. So we're a code-first organization, whereas GENIVI is a specification-first organization.

What has happened is that GENIVI has been building a couple of very good and interesting middleware modules, but they're not part of a big distribution. So what we do, and this is how we play with GENIVI, is we port those middleware modules onto AGL, and someone who needs those can use them no problem. We talk to [GENIVI] every week. Their Chief Architect attends our meetings and gives us feedback, and we're designing our stuff to make sure we're collaborating with GENIVI and that we don't make decisions that alienate GENIVI. So we're working very closely together.

Q Let's get into the recently released AGL Requirements Specification 1.0. What can you tell us about that?

To our knowledge, this is the first open in-vehicle infotainment (IVI) specification available. It's open to anyone, can be downloaded from our website, and anyone can go ahead and give us feedback on it. It defines a highly integrated, Linux-based reference platform that has all of the services you would expect like Wi-Fi, Bluetooth, multimedia, location-based services (LBSs), a windowing system, layer management, life cycle, and audio management (Figure 1). It defines support for both native Linux apps and HTML5 apps, which makes us unique because we're going to be supporting both as we have members that want HTML5 and we have members that want native, so we're going to make them coexist (Figure 2). We're building the reference platform on multiple types of hardware, so x86 of course and various versions of ARM chips. Right now we have a Renesas R-Car reference board and we're planning to add a Texas Instruments (TI) and a Freescale board also. So, basically whoever is willing to step up and help us on a given hardware platform, we're willing to do it.

Then, one of the differences between the spec that we have and something like the GENIVI spec is that we're using it primarily as gaps analysis to figure out

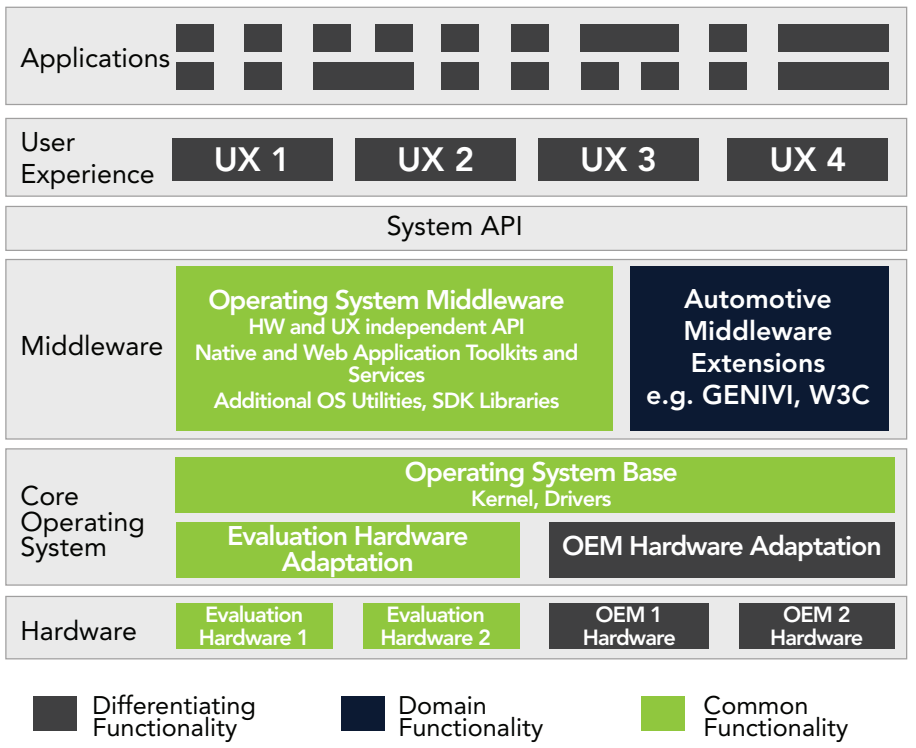


Figure 1 The Automotive Grade Linux reference architecture.

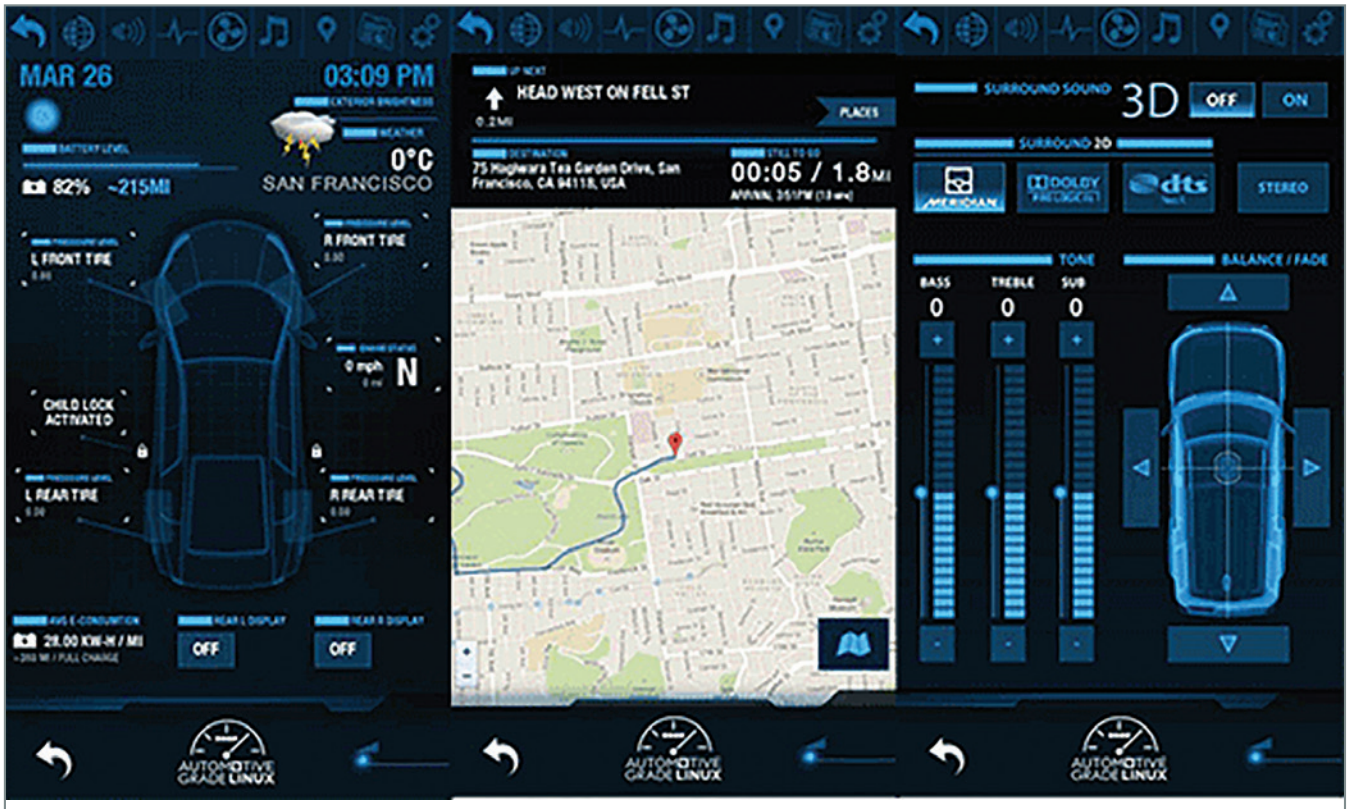


Figure 2 HTML5 demo apps built on the AGL reference platform.

what the desired features are and how they should work versus what's in the code. We're a code-first organization and will always be, so we're not planning to use the spec as a compliance document. It's more about the carmakers saying, "Here's what we need. Here's what we want." And we're analyzing, "What are the gaps between the spec and the code? Let's go write the code that's missing." That's essentially how we're using it. It's more like a design document than it is a specification or compliance document.

But to me the key takeaway is that this is the first time the automakers have really sat down together with their suppliers and worked collaboratively on a spec, because they tend to be very competitive. It's a clear indication of the shift that's happening in terms of them adopting an open-source development methodology. So they're doing things in the open, in a collaborative way, and to me that's the real message here because that type of thing wasn't done in the past. There was secrecy and all of these things. They've broken barriers to make this happen, and that's really good news.

Q So, why was Tizen chosen as the foundation for the reference platform?

We wanted to make rapid progress and we wanted to show traction and attract new members, and Tizen was a good platform. It's being supported, there are literally 1,000 developers working on Tizen from Samsung, Intel, and others, and it was a good base for us to do our prototyping and release a version 1.

Having said that, we're moving away from Tizen now, which was always our plan. We are developing our very own AGL distribution, and the codename right now is Unified Codebase. What we're trying to do is bring the best of Tizen, AGL, and GENIVI together into one unified codebase. We've announced that we've started the work, and basically we're going to have different layers: a layer that brings all the common parts together, a layer that brings the GENIVI stuff, a layer that brings the AGL stuff, and then two app frameworks, one for native and one for HTML5, all in one platform.

Right now we're really focused on rearchitecting this Unified Codebase, so we're going to be doing a lot of work on yocto layers, which are basically meta layers that define where to get the code and tell the compilers what to compile. So it's not actually writing code, it's more about rearchitecting so that we have the layers that we need for long-term maintenance. We're getting maintainers for each layer, so that work is ongoing right now.

Q What plans, if any, do you have beyond infotainment?

We plan to address everything in the car, and I always say that if Linux is in the car, we want it to be based on AGL. Part of our charter is to address the instrument cluster (which is increasingly LCD displays); heads-up display (HUD) units; telematics, which is basically the "connected car" and all of the connectivity to the cloud and all of these Internet of Things-type (IoT-type) features; and eventually control systems, which require certain safety-critical certifications that we're actually starting to work on right now.

We started where we did primarily because infotainment is the biggest pain point for the automakers right now. They just have not been able to keep up with the smartphone, and the lack of a single platform and the lack of a standardized application framework has been a real pain point. So this was the one with the most bang for your buck, and what we're going to do is take the base platform and have different profiles. So it will be the same base platform, but when you build it the compilers are going to pick up different modules for the cluster or HUD, but we're going to try to reuse as much as possible of the base platform.

Q More long term, how much do you think smartphones are going to continue to play a role in in-vehicle infotainment (IVI) platforms for automotive? What do you think the role of the smartphone will be 5-10 years from now?

That's an interesting question because there are two trains of thought, two camps in automotive right now. There's the camp that says, "Let's just use projection mode. Use the phone, project it to the display, and there you have it, we're all done." But there's a problem with that. One is that you still need an OS and you still need a lot of features in that head unit because you still need an air conditioning control and all the functions of a car still need to be displayed in that head unit, which means you need a full-blown OS – you're not going to ship a car without a radio, you're not going to ship a car without Bluetooth. So suddenly you've got two things: you've got a full OS in the car and you've got a full projection display in your hand, and you have redundant systems there.

That's one problem, but the other, bigger problem I think is that some of the car manufacturers have said, "We don't want to give away our customers' data and the relationship we have with our customers to Google and Apple." So there's that philosophy of, "Wait a minute, it's our car, it's our system. We're going to have this excellent experience based on our brand, not someone else's." So the portal that the customer goes to is going to be branded by the car manufacturer, not by Google and Apple. So those kinds of philosophical debates on who owns the data and who's going to make money off the data – for example through LBSs and things like that – a lot of car manufacturers are saying, "We're not going to go that way. We're keeping our customers."

Dan Cauchy is General Manager of Automotive for the Linux Foundation.

Automotive Grade Linux
www.automotivelinux.org
info@automotivelinux.org

FOSTERING COMMUNITIES AROUND DEVELOPMENT KITS

Dianne Kibbey
Global Head of Community
element14

One element that makes the maker/development board space great is the communities. There is always someone who can help you with an issue, a plethora of information about a kit, and endless projects for inspiration. But these communities don't just appear out of thin air after a new kit is introduced. Dianne Kibbey, Global Head of Community at element14 talks features and strategies that make for effective and thriving communities.

Q What are the essential elements of a good dev kit community?

Every strong dev kit community needs in-depth content that members can engage with. This is a mix of technical specifications and user experiences with the dev kit. For example, our element14 Design Center combines technical data with a wide array of user-generated material. In addition, coding examples, polls, and 360-degree videos and images are all great ways to generate conversation.

Another strategy we use to spark conversation in the element14 community is asking specific members to develop projects using the new kits and write blog posts about their experience. This then fosters great comments and questions as members share their experiences. We've also featured polls asking questions designed to drive conversation such as, "What are the three elements every basic dev kit needs?" All of this content combines to create a lively community in which members are willing to bounce ideas off of each other. The kit itself also plays a big part in establishing the community since that is what the conversation is centered around.

Q What are the challenges of managing a dev kit community?

Development kit technology evolves so rapidly that you are always faced with the challenge of keeping up-to-date technical data, projects, and coding examples for your community members. You may have produced content focusing on a specific dev kit, but once a revision is introduced you may need to retest it and generate new content. These fast-changing technologies require community managers to regularly monitor whether or not the content and discussion within a community is drawing interest from members and developers. If not, fresh content should be produced in order to foster conversation.

Q What are the best ways to foster developer participation/contribution?

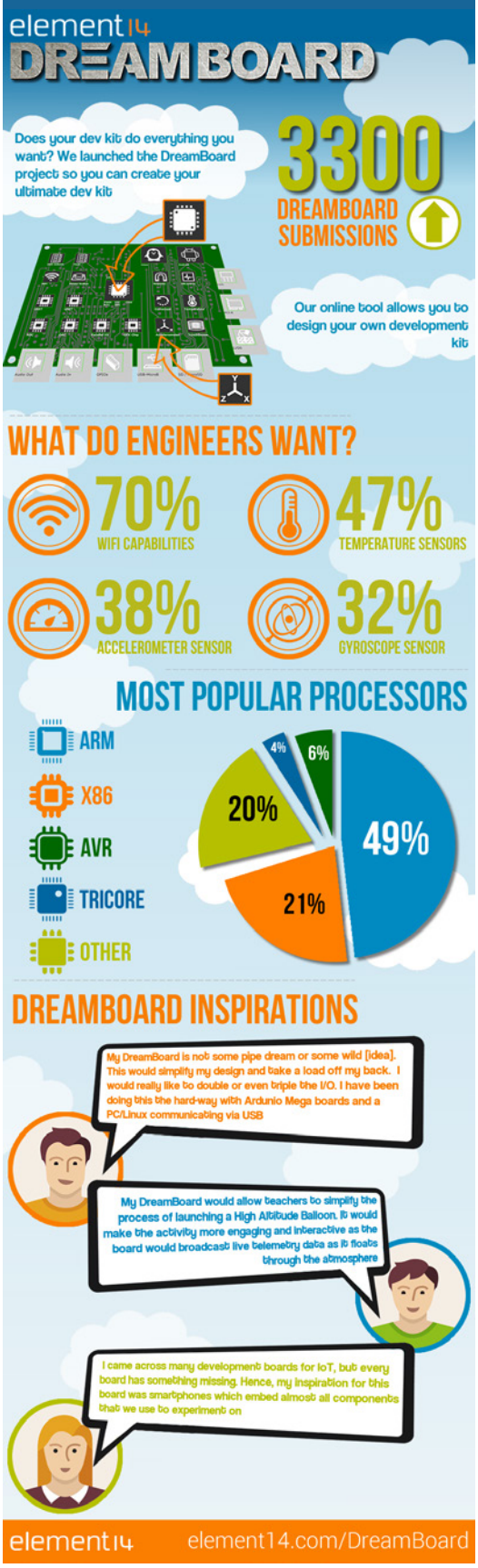
Giving more control to community members is a great way to build credibility and encourage more participation. When you enable members to take charge, they have the opportunity to help each other more effectively and mentor less-experienced members who might be new to the community or to engineering in general. This encouragement welcomes new members and gives them a way to quickly become familiar with different topics being discussed.

Turning over the community to members also results in more credible content and discussion. Engineers, both professional and self-taught, want to hear from their peers when it comes to dev kits and other board-level technologies. Empowering members to engage with the community on their own terms is crucial to creating genuine conversation and increased participation among members who otherwise might not have voiced their opinions.

Q What do professionals need in a community compared to hobbyists?

Professionals typically need more in-depth technical specs and this extends to areas such as legislative and compliance data, something many hobbyists might not need to worry about. Another important thing professionals might be concerned with is figuring out how to put their ideas into production. The Business of Engineering section of the element14 community helps professionals develop the final iteration of their original prototype and bring their product to market once they perfect the design. For professionals there is much more of an interest in real world applications,

element14's DreamBoard project asked members to submit features of their ideal development boards. Here's how the community responded.



price, and availability of dev kits – three huge factors that impact whether or not a product is successfully launched.

Q New development kits are coming out all the time. What are the challenges and strategies to creating a community around a dev board/kit?

One of the biggest challenges to creating a dev kit community consists of bringing new products to market and figuring out a way to overcome the fact that engineers can be slow to adopt emerging technologies. Not many community members are going to rush out to buy a new development kit until it is put through its paces. We want to give engineers the chance to try out new development kits so that by the time they launch, every possible question about them has been asked and answered. The key is identifying influencers within your network and allowing members to provide an honest assessment of the dev kit before making it more widely available.

When we launch a new development kit in the element14 community we give engineers an opportunity to try it out first. Our RoadTests enable our community members to test out new products and provide valuable feedback that ultimately help a supplier improve their final products. We also host educational webinars and videos that cover a wide range of engineering topics, including the newest technologies and latest design trends.

Q How do you expand a dev kit's community? Is size important?

Growing a dev kit community is all about promotion. You can't simply build the community and expect it to flourish on its own. One way in which you can raise awareness of your community is through social media. We often find that our strongest community members at element14 are also the most active on social media. Sharing content across multiple channels will help to attract members and grow the community over time.

While size is something you should consider, interesting content and engagement from all members is what will end up creating quality discussion. If you have 500,000 members, but only five of them regularly contribute, you don't have much of a community.

Q Do strong communities make or break a dev board/kit?

They definitely can. With more than 350,000 members, we've seen the element14 community have an impact on the success or failure of a new dev board. If a product comes to market and a community finds issues, there is a possibility that the dev board may not recover from member criticism. It's similar to social media in that once a review of the dev board is made public there is not much that can be done to remove it. There is a balance you have to strike by making sure to include a respectable product in addition to forming a community that enables members to test that product and honestly discuss the positive and negative aspects of it. If you don't have a quality product in a community, you're opening yourself up to a degree of risk.

Q What is the result of a good community?

The best communities are those that offer members the opportunity to shape the future of engineering and technology. It can be easy to focus on a specific dev kit, but the content and discussion between members should dive deeper into how these technologies impact the world around us. For example, element14's Engineering a Connected World report went outside of the community to survey 3,500 consumers on a wide range of innovations, such as wearable technology, autonomous cars, and the IoT. This report not only inspired a series of new design competitions centered around solving real-world problems, like vertical farming, but it gave our members the chance to step back and examine questions and topics that extend beyond a single dev kit. That's where the real power of a good community lies.

element14

- www.element14.com
- @element14
- www.youtube.com/user/element14

For more DIY news, articles, products, and more, visit embedded-computing.com/topics/diy

Dev Kit Dilemmas

How engineers are using development kits in the design and production process

element14

77% of engineers regularly check to see if a development kit is available for a component they are evaluating

58% of engineers who use development kits, use them in the majority of their designs

Where is open source used?

- 60% App Software
- 52% Operating System
- 72% SBC Design
- 59% Hardware Functions

80% of designs using a development kit move through to production in 12 months or less

76% of engineers say development kits are critical to encouraging innovation in modern technology and design

Top requirements for a development kit

Physical interface and connectivity - 47%
Number of useful on-board features - 43%
OS and language support - 36%
Processor data width and speed - 36%
Features (wireless, status LEDs, battery management) - 28%
Core architecture - 26%
Memory - 24%
On-board sensors - 21%

44% of engineers would be unable to do their job without using a development kit

Source: http://www.element14.com/community/docs/DOC-70598/l/dev-kit-dilemmas-report

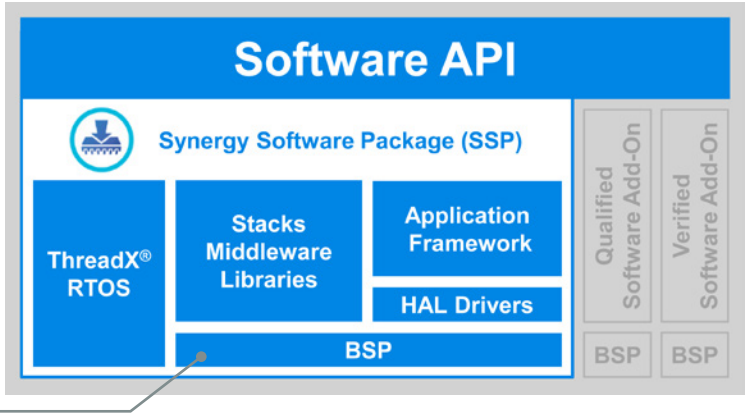
8-bit MCU advancements from IoT to safety critical

Microchip's PIC16F18877 and 1579 MCUs integrate core-independent peripherals (CIPs) including ADC computation and independent time-based PWMs that provide additional performance with determinism and reliability. Development is also made easy with the Curiosity Development Board and downloadable MPLAB code configurator for easy bring-up of applications from wearables to safety critical systems.

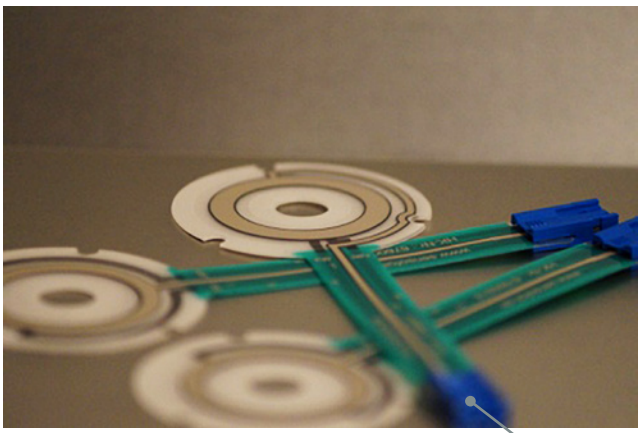
Microchip Technology Inc. | www.microchip.com
embedded-computing.com/p372911

Synergy Platform and X-Ware team for powerful IoT platform development

The Renesas Synergy Platform combined with the Express Logic X-Ware software provide an integrated IoT development platform that enables developers to focus on application development with a variety of software modules for connectivity and multimedia at their disposal. The Synergy Platform provides compatibility across all MCUs while providing pre-integrated software. The X-Ware software environment features the ThreadX RTOS, TCP/IP, and USB connectivity, a DOS-compatible file system, graphical user interface library, and event analysis tools.



Renesas Electronics and Express Logic
am.renesas.com | www.rtos.com
embedded-computing.com/372912



Ink and foil based sensors for IoT, IIoT applications

Hoffmann + Krippner demonstrates unique SensoInk and Sensofoil technologies for process automation, position tracking, and real-time supply chain support. SensoInk uses polymer thick film technology to increase flexibility and reliability with no electromagnetic effects. Sensofoil is a thin-film membrane position sensor and is available in ultra-thin, long lifetime, or hybrid versions with extended lifetime and temperature ranges.

Hoffmann + Krippner | www.hoffman-krippner.com
embedded-computing.com/p372913

Annapolis Micro Systems

The FPGA Systems Performance Leader

WILDSTAR OpenVPX Ecosystem

FPGA Processing Boards
1 to 3
Altera Stratix V or
Xilinx Virtex 6 or 7
FPGAs per Slot

Input/Output Modules Include:
Quad 130 MSps thru Quad 550 MSps A/D
1.5 GSps thru 5.0 GSps A/D
Quad 600 MSps D/A
Dual 1.5 GSps thru 4.0 GSps D/A

1 to 40 Gbit Ethernet
SDR to FDR Infiniband

Open VPX Storage
Up to 8 TBytes Per Slot
4 - 8 GBytes Per Second

GEOINT, Ground Stations, SDR, Radar, Sigint, COMINT, ELINT, DSP, Network Analysis, Encryption, Image Processing, Pattern Matching, Oil & Gas Exploration, Financial and Genomic Algorithms,

Open VPX Switch
1 to 40 Gbit Ethernet
SDR to FDR Infiniband

Chassis
4, 6 or 12 Slot
Up to 14G

Annapolis
Micro Systems

High Performance Signal and Data Processing in Scalable COTS FPGA Computing Fabric

190 Admiral Cochrane Drive, Suite 130, Annapolis, Maryland USA 21401
wfinfo@annapmicro.com USA (410) 841-2514 www.annapmicro.com