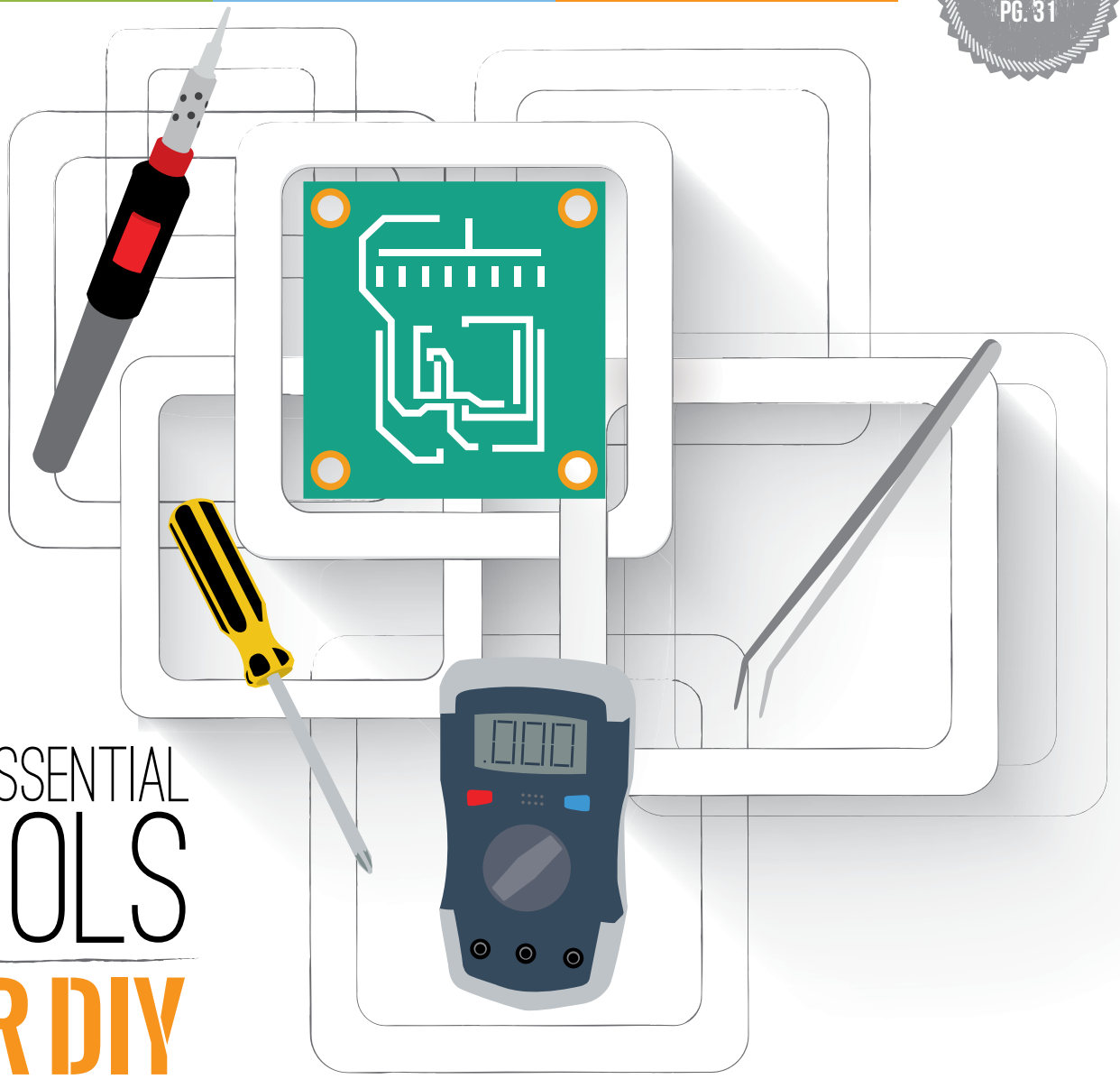
SEPTEMBER 2014
VOLUME 12
#6
EMBEDDED-COMPUTING.COM

Embedded COMPUTING DESIGN

Connecting Silicon, Software, and Strategies for Intelligent Systems

DIY ISSUE
PG. 31



THE ESSENTIAL TOOLS FOR DIY

+ PLUS

30 MINUTES OF 4K VIDEO CONSUMES OVER 10GB OF MOBILE MEMORY.

Get high-performance storage for an Ultra HD future.

With Ultra HD on the rise, smartphones and tablets will need highly responsive storage to keep up. That's why for over 25 years, SanDisk has been expanding the possibilities of storage. The result is more than just incredible consumer experiences. It's enabling the next generation of mobile devices. **sandisk.com/storage**

SanDisk®

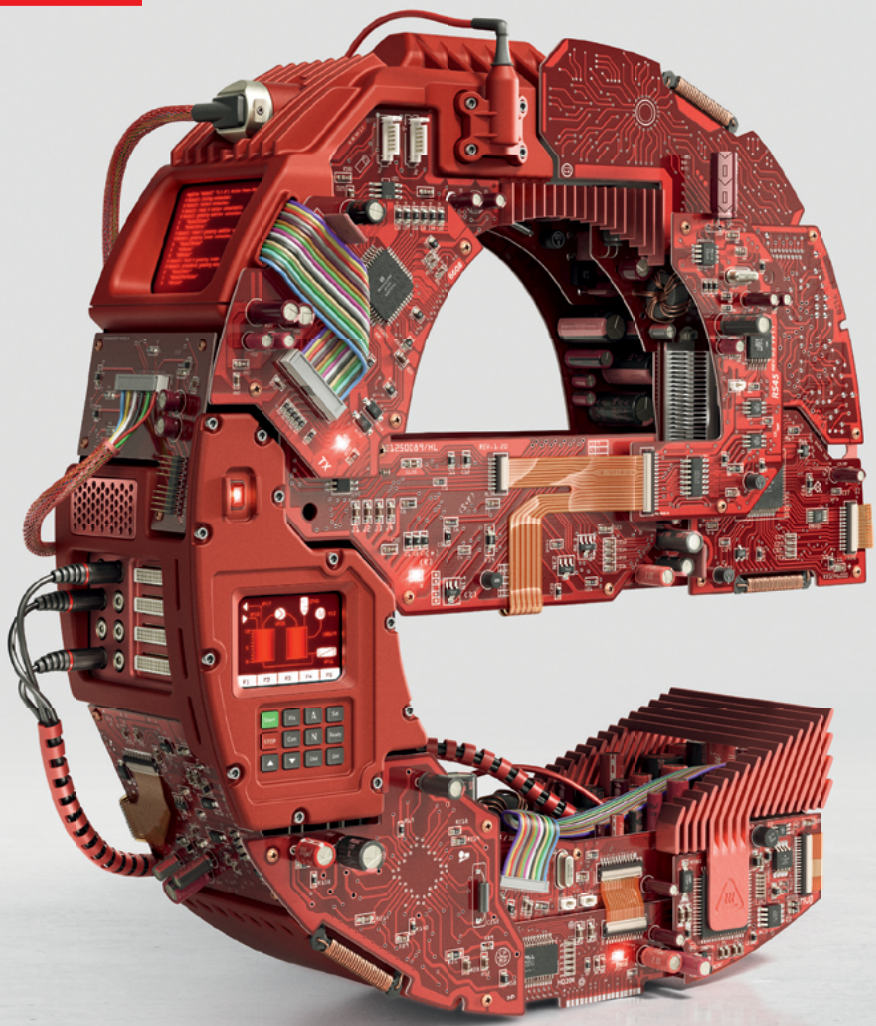
Messe München International

Connecting Global Competence

Welcome to Planet e.

The entire embedded universe at a single location!

Tickets & Registration
www.electronica.de/en/tickets



26th International Trade Fair for
Electronic Components, Systems
and Applications
Messe München
November 11–14, 2014
www.electronica.de

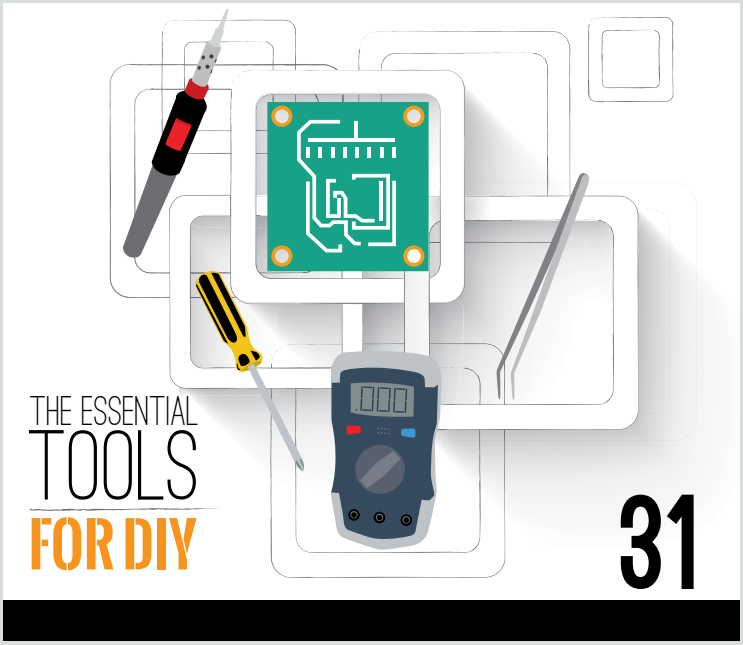
50 years
electronica

electronica 2014
inside tomorrow

Embedded COMPUTING DESIGN

SEPTEMBER 2014
VOLUME 12
#6

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



Departments

Silicon

Software

Strategies

Special Features

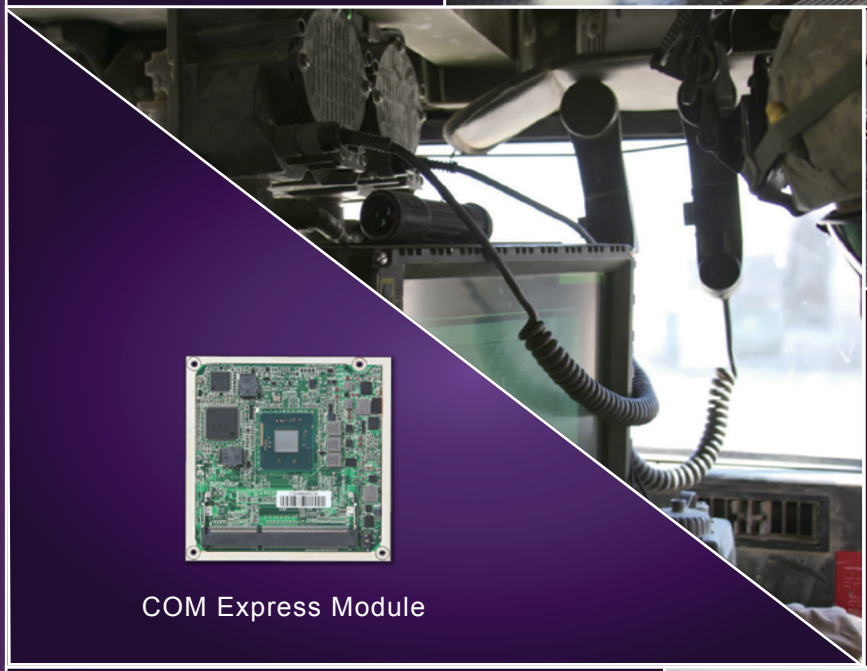
APP EXCLUSIVE CONTENT

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

Portwell Empowers Intelligent Solutions

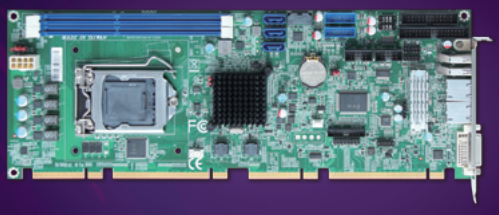
Mini-ITX

Small Form Factor System



COM Express Module

Network Security Appliance



PICMG SBC

www.portwell.com
info@portwell.com
1-877-278-8899

Portwell

intel IoT Solutions Alliance Premier

ADVERTISER INDEX

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

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

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 the cloud and embedded connectivity to aircraft black boxes

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

Such is the worldwide notoriety of recent infamous aviation tragedies, they need no introduction. For the most recent tragedy, MH17, the flight data recorder or "black box" was missing in action when I started drafting this article, though by completion is now thankfully in the hands of the correct authorities. However, with its time spent in unscrupulous hands, potential evidence tampering is a significant concern, an aspect I'll later detail.

The MH270 catastrophe and search still baffles the very latest technology and the most brilliant minds throughout the globe. Given the square mileage involved, searching "after the event" is a monumental task; given that almost all of this is deep ocean, "problematic" is an understatement. Detection equipment, such as radar and satellite imagery, evidently only goes so far. What is needed is embedded intelligence on the aircraft itself.

Given that early passenger jets had no event recording equipment at all, the black box (actually orange for visibility) we utilize today is a huge step forward. Modern passenger jets in fact utilize two black boxes, a 25-hour flight time statistical recording alongside a rolling two-hour recording of cockpit conversations. The boxes are built to survive crashes, fire, and deep-sea immersion, emitting a tracking frequency to facilitate locating the device for up to a month after, though recent events prove this isn't nearly enough.

The predominant issues with these now-archaic solutions are thus: the ease of locating them, the restrictive depth of information they record, the investigative delay of having to physically retrieve them, and the opportunity for these to fall either permanently into the wrong hands or the questionable validity of that data stored therein if returned (if they still contain usable data at all!).

You'd be safe to assume that the majority of passengers on both of these planes possessed smartphones, all of which you may be surprised to learn are many multitudes more powerful than the evidence-collecting black box. It's easy to retrospectively postulate that if even one MH270 passenger had GPS active and transmitting coordinates to a relevant authority, billions of dollars, and, far more importantly, lives could have been saved by immediately tracking their location to within a few meters – this is not the passenger's role of course, but should be the black box's.

The existing machinery does record altitude, airspeed, and bearing, but combining this existing information with GPS technology would provide the information necessary to recreate an exact, time-accurate 3D model of the entire flight from start to ill-fated finish, easily layered over Google Earth to show exact locations.

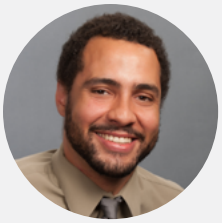
Coming back to my earlier mentioned investigative delays and potential evidence tampering, we only have to look at the scandalous behavior of those guarding the MH17 crash site to see that not everyone is keen on authorities establishing the truth. It seems obvious to me that both are easily addressed by utilizing the power of the cloud.

With modern mass storage technologies, live recording of the data of approximately 5,000 planes in the air simultaneously is more than feasible. Not only would live storage of this data on the cloud allow instantaneous access when any incidents do occur, it would immediately remove even the potential for any unscrupulous evidence tampering in one fell swoop.

I'm happy to concede that we're still not achieving constant 3G access, even in developed countries – this isn't a problem. I'm not arguing for the removal of the local black box from the equation; the data should persist to be stored locally and updated whenever available signal permits to the cloud. Remember, we're not talking about a lot of data here, all this data is merely numbers.

Maybe there's a staggered approach – the most critical data (GPS coordinates and time-stamps) are transmitted by "always on" but low data throughput methods that aren't signal dependent. The future could even offer live video recording, as many passenger cars now utilize for insurance purposes.

Yes, there are difficulties in retrofitting, but recent events have shown this cannot wait until the next range of passenger jets – when these things happen we need answers, immediately. ECD



Hardware commoditization and the IoT service model

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

Earlier this year, economist Jeremy Rifkin released the book "The Zero Marginal Cost Society: The Internet of Things, the Collaborative Commons, and the Eclipse of Capitalism." In it, Rifkin argues that the Internet of Things (IoT), which he defines as a unison of the Communications, Logistics, and Energy Internets, will converge with the competitive capitalist market to usher in a period of extreme economic productivity in which "the cost of actually producing each additional unit – if fixed costs are not counted – becomes essentially zero, making the product nearly free." As a result, capitalism as we know it today will be slowly replaced by the distributive economic model of the Collaborative Commons.

While this notion may be objectionable to those of you in the Western world, there's no denying that the cost of compute and connectivity are in a sustained decline. Moore's Law continues (at least for now) to eat away at the margins of hardware vendors, and Google Fiber is currently providing free 5 Mbps Internet in Austin, Texas, Kansas City, Missouri, and Provo, Utah, with 1 Gbps speeds available for $70 per month. Trends like these have led to a lot of business model rethinks in the tech sector, with many companies turning to the cloud for answers.

The cloud space has become a crowded one to say the least over the past couple of years, partially because of the "services" model it offers businesses. Today cloud service models range from Software-as-a-Service to Platform-as-a-Service to Infrastructure-as-a-Service (SaaS, PaaS, and IaaS, respectively), with the newly coined Everything-as-a-Service (XaaS) entering the fold as well. These service platforms deliver everything from storage and security to full-blown end-user applications, which can each be neatly packaged as line items on a monthly statement.

So why is the cloud important for embedded developers? Hardware commoditization.

Hardware commoditization and the IoT-as-a-Service

As the dust settles around IoT standardization, open, modular architectures with an emphasis on software development and app enablement will take precedence over custom or application-specific hardware designs (look at the success of "maker" boards like the Raspberry Pi). Does your next system require wireless connectivity? Order a Wi-Fi module from Shanghai. Do you also need analog sensors? Browse the capes on Adafruit's website. If Rifkin's predictions hold true, specialized hardware will only be sustainable in a very narrow set of fringe applications, so the majority of system developers will have to find other ways to create value.

Take, for example, ConnectSense, a company based out of Naperville, Illinois that produces a line of Wi-Fi sensors for home and building automation (www.connectsense.com). ConnectSense sensors range from temperature and humidity to water, motion, light, and dry contacts, but the target market demanded a cost-conscious approach across the product line. Therefore, the company organized the portfolio around a base platform consisting of a repurposed ARM7 SoC that was developed in-house, a TI MSP430 MCU, and a low-cost, low-power Wi-Fi module from partner Shanghai High-Flying Electronics Technology Co., Ltd. This approach allows multiple sensors to be manufactured quickly and easily with only few modifications to the common platform.

What makes an architecture like ConnectSense unique, however, is that it's also powered by a proprietary cloud platform that handles most of the heavy lifting of software and application development, so additional hardware resources aren't required on the physical sensors themselves. For novice users, the ConnectSense cloud provides an if/then rules engine that can be used to set up alerts via email, text message, phone call, webhook, or tweet in a plug-and-play fashion, while more advanced developers can take advantage of a full REST API (Figure 1). Today the ConnectSense system is being leveraged in applications such as datacenter monitoring and agricultural observation.

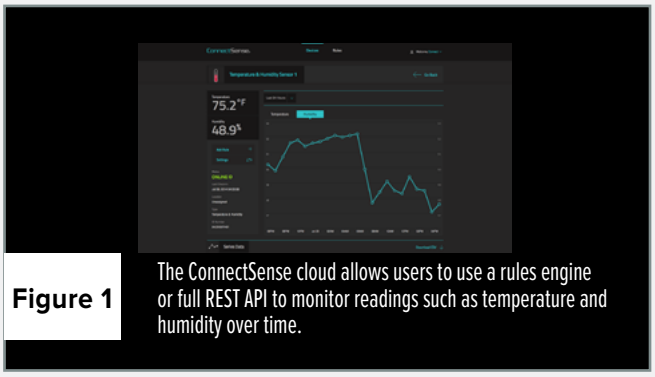


Figure 1 The ConnectSense cloud allows users to use a rules engine or full REST API to monitor readings such as temperature and humidity over time.

ConnectSense sensors currently start at around $150 a piece, with the cloud platform included at no extra cost. But what's really intriguing about this architecture is that if hardware margins erode substantially enough, the company could conceivably flip the current model on its head and monetize their cloud services while providing ConnectSense sensors free of charge (sort of like your cable provider). I guess you could call this "IoT-as-a-Service." ECD

Making with Ben Heck

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

Benjamin J. Heckendorn, also known as Ben Heck in the maker community, is a prolific maker and modder always working on something cool. Though tinkering wasn't always his day job. As a kid he was into electronics kits, and then got into video games and movie making. When he got a day job as a graphic artist, he used to look for fun projects to do over the winter.

"I liked Atari game consoles as a kid, but never had one because it cost more than $3," Ben says. "But as an adult I wanted to hack and mod an old Atari. I tried to make it into a portable version of itself."

He did just that about 10 years ago, and wrote a blog post about it (and later a book: "Hacking Video Game Consoles: Turn your old video game systems into awesome new portables"), and the interest started pouring in. "I had no idea people were into the modding community. This was before there was a real hacker/maker community."

For several years he did console mods, but then branched out into accessibility equipment and things that were useful instead of just "fancy toys and gussied up gaming consoles." Now his projects run a huge range of applications.

As a podcaster, his goal is to demystify things so people realize they can make it themselves. And he appreciates how much work people do in the maker community to make things accessible. Parts are low cost, kits can make projects easier, and there's a ton of information on how to make just about anything.

"People are scared sometimes," Ben says. "You might not know how to do something, but you know how to do this [other thing] – maybe they can do programming, but don't know how to solder, but with a kit they can do it."

Ben has quite the workshop (though he says it looks bigger on the show) with a wide variety of tools, but there are certain tools he says all makers should have. His favorite, most useful, can't-live-without is his tweezers. After that he lists nice screwdrivers (though a nearly 30-year-old one is his favorite) and a decent soldering iron with multiple temperatures. Optional: automatic wire strippers. ("Sometimes I use those instead of my teeth," he says. Ben is good, but I don't think DIY dental work would be a smart idea. Stick with the tools!)

3D printers are also a tool he holds in high regard, especially the up-and-coming delta robot 3D printers, one of which he

brought back from the Detroit Maker Faire in July. Most 3D printers use an XY Cartesian system, but the delta has a 3-axis delta head for greater printing speed. That seems to be the new trend for maker 3D printers, he says.

With sensors, servos, and microcontrollers already widely available and affordable, Ben is waiting for someone to put the desktop-scale 3D printing concept in PCB manufacturing to bring quick, safe, and affordable circuit boards to hobbyists. The current methods of breadboarding and sending them off to be made elsewhere leads to expensive, slow, or iffy quality parts.

"That's the next big thing, and whoever can do it will have a million dollar Kickstarter on their hands. I'd throw my credit card at that person. It's the holy grail." ECD

www.benheck.com | www.element14.com/tbhs

Memorable project

Robot Luggage: opsy.st/BenHeckRobotLuggage

"We made carry-on luggage that had wheels, but you push down the handle into a unit and a third leg pops out like R2D2 and follows you. The reason for that was if you're at the airport and you get food and you have to hold a bag, laptop, food, drink, and whatever else, and you're trying to carry it all – what if your luggage could just follow you?" Yes, please!

If you could make anything...

If you could build anything you wanted without limitations, what would it be?

Ben Heck says: "Some sort of car I would think. There are already aluminum electric cars, but I would build some sort of electric car. If I had lots of time and space and money I'd try to build a custom car…or maybe a solar powered boat…with solar panels on the top and sides because light reflects on water and you can collect light from that...Or a boat car!" ECD

Making: The gateway to an engineering career

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

Making and DIY platforms are now impacting university engineering programs across the country. MIT has even begun accepting maker portfolios as part of their admission process! Arizona State University is now using Arduino and Raspberry Pi in coursework and co-locating classroom facilities in a TechShop makerspace for greater access to prototyping facilities, transforming the way the university conducts engineering education.

Makers and Arizona State University Fulton Schools of Engineering Assistant Professors Shawn Jordan and Micah Lande are two ASU Polytechnic campus faculty integrating DIY into their programs. Jordan, also a Rube Goldberg Machine past Guinness World Record holder, has a background in electrical engineering, and teaches embedded systems classes with a special focus on real-world applications. Lande has a background in mechanical engineering design, and his coursework is focused on human-centered design, manufacturing, and engineering innovation. Lande also founded the first Maker Corps (makered.org/makercorps) site in Arizona. They use DIY projects as learning platforms in the classroom, and study the maker community and its relation to the everyday practice of engineering.

"From our first-hand experience, we both believe that engaging in making has significant value in connecting theory to practice and exciting students about what it means to be an engineer," Lande says. "Most engineering programs do this at the capstone level; our program does it all four years, and our research explores how to further bring making into engineering education."

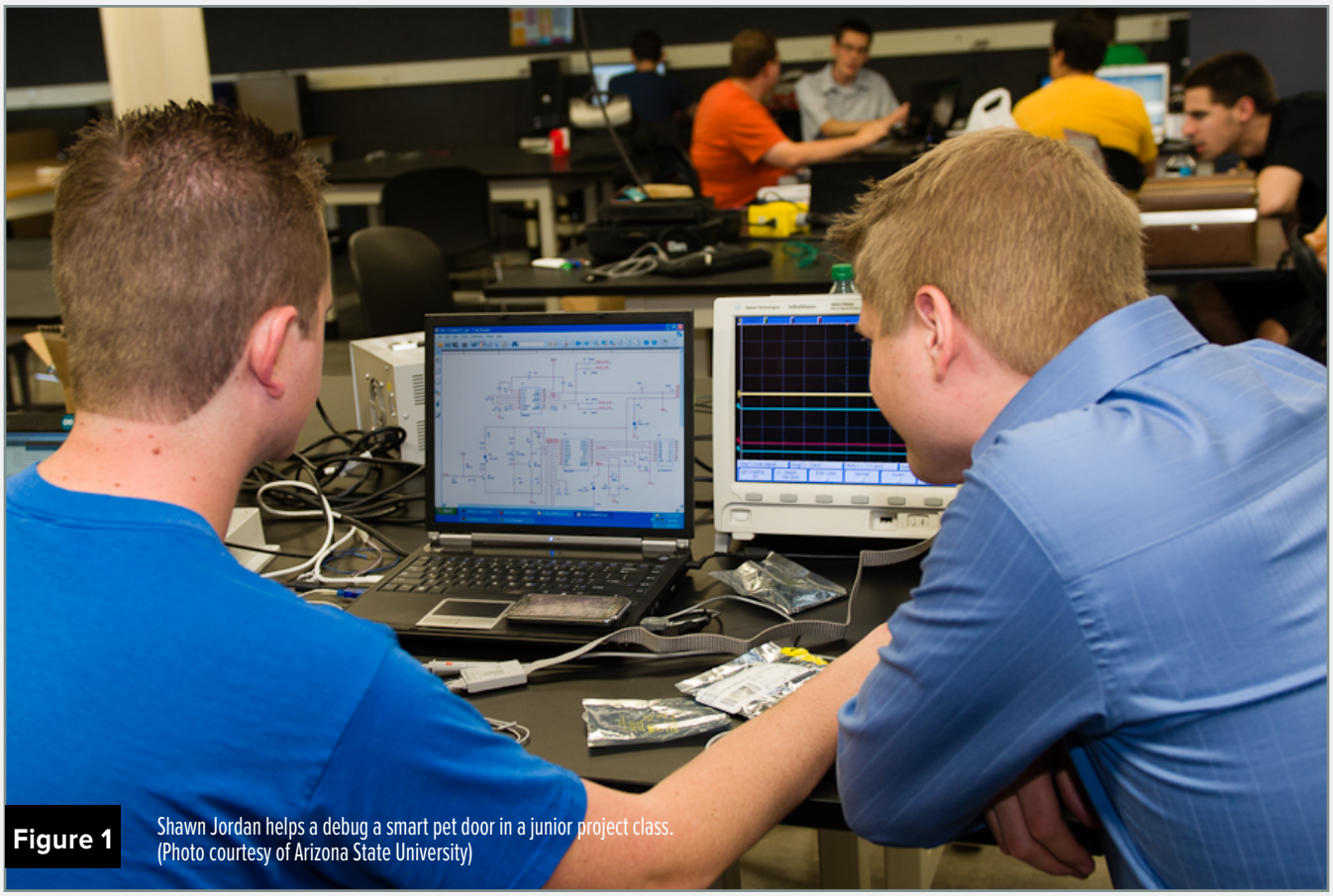


Figure 1 Shawn Jordan helps a debug a smart pet door in a junior project class. (Photo courtesy of Arizona State University)

Jordan and Lande say DIY platforms like the Arduino and Raspberry Pi are extremely popular in robotics and maker clubs at the university, and the platforms are used throughout students' engineering academic career. As freshmen, their students build projects like elevators with Arduinos. Sophomores create science and engineering exhibits with the Arizona Science Center based on Arduinos. Jordan's junior-level embedded systems project courses have students generating their own project ideas within a particular context, such as inventions with business plans, wearable electronics, and Internet of Things (IoT) devices, using the Raspberry Pi and Cypress's Programmable System-on-Chip (PSoC) platform – an ARM core and programmable analog and digital components in a single FPGA (Figure 1). Industry- and community-sponsored capstone projects also sometimes use Arduinos.

"This format engages students to make their ideas a reality, and to become lifelong learners by determining what they need to know and motivating them to learn beyond the minimum requirements of the course," Jordan says.

Both professors are also involved in research on what making means to people and how it can foster interest and success in regular engineering. Jordan and Lande conducted nearly 50 interviews at Maker Faires in the U.S. to find "educational pathways in and around engineering, and the knowledge, skills, and attitudes developed during their making activities."

They found that making can encompass the act of creating and building just about anything, and it should be what they call "additive innovation" – contributing to the open maker community of sharing and learning.

"Citizenship in the maker community means additively building on the world of others and freely sharing knowledge and processes back with the community," Jordan says. "The making community is an additive innovation network, both as a social and intellectual community."

Playful invention and creativity are hallmarks of the maker community and their projects, driven by makers' personal interest. It's also a community where failures are celebrated as learning experiences. All of this bodes well for making's influence on the National Academy of Engineering's vision for "The Engineer of 2020."

"Makers demonstrate lifelong learning, practical ingenuity, and creativity by the action of making itself, and their contribution to additive innovation networks," Lande says.

Though their research initially focused on adult makers, Jordan and Lande have expanded it to include young makers and their educational and career ambitions. In future research they plan to explore making and STEM (science, technology, engineering, and math) pathways and inform engineering education efforts. In addition, parents of young makers and their role is their next target of study.

"For young makers, parents are additional, active members of the additive innovation network, supporting their children financially, technically, logistically, and emotionally," Jordan says. "We plan to expand our interview strategy to include parents, as they have strong opinions about the benefit of making for their kids."

Overall, they see the maker movement as a positive influence on engineering education and the future of engineering.

"The democratization and availability of building tools make such things more accessible and available, in the classroom and more broadly," Lande says. "More and more, we have students arriving at college with significant experience creating and building complex systems – that allows for us to dive deeper into not just what can they make, but what should they make. We can teach closer to more authentic engineering practice." ECD

Rube Goldberg, engineering, and making

Shawn Jordan is a Rube Goldberg Machine champion, founding one of the most successful collegiate Rube Goldberg Machine Contest teams at Purdue University, leading teams to two national championships, and influencing future success with his design process and strategies. He has held a Guinness World Record for the world's largest Rube Goldberg Machine (125 steps), and one of his machines was even featured in a movie. Jordan has made television appearances on Modern Marvels on The History Channel, Jimmy Kimmel Live, and served as a behind-the-scenes engineer for PBS's Design Squad engineering design reality TV show. Aside from being fun to build and watch, Jordan sees Rube Goldberg Machines as similar to embedded systems and making.

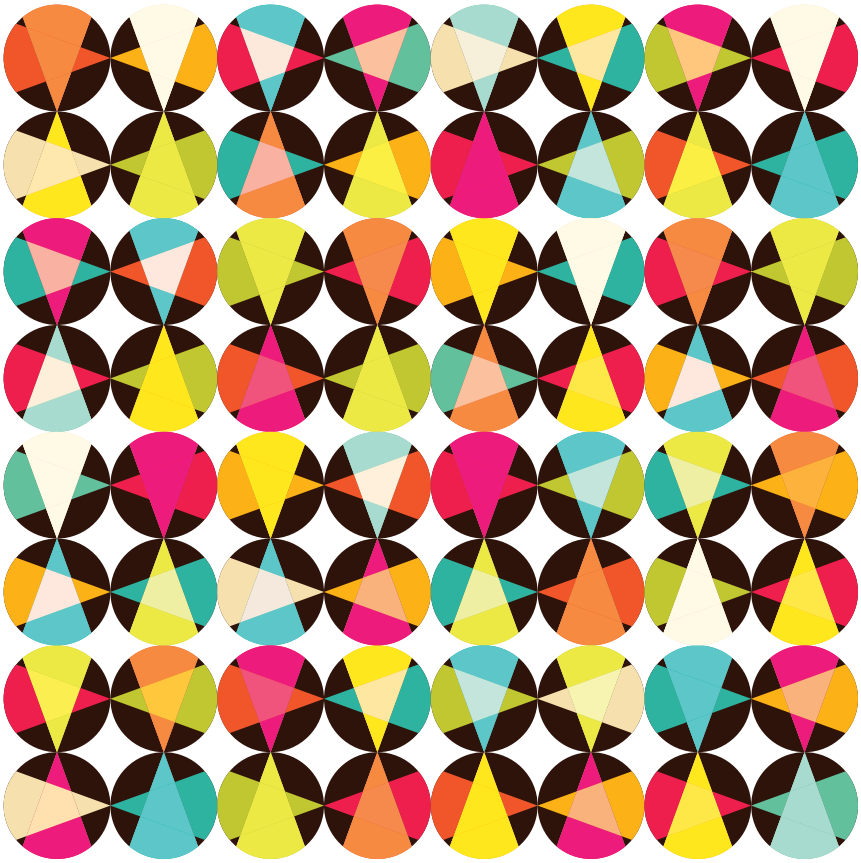
"I see Rube Goldberg Machines and embedded systems as being very similar, particularly when using platforms like Arduino and Raspberry Pi," Jordan says. "Both are systems of systems. Rube Goldberg Machines are typically designed in modules that link together into a single chain reaction, where each module senses something (electrically or mechanically) and triggers an action based on the input. Embedded systems are similar in that they are made up of sensors, actuators, and processing elements. When building a Rube Goldberg Machine, you are both a subsystem designer and a systems integrator. Products designed by makers similarly require systems integration."

See more about how Jordan is using Rube Goldberg Machines and building creative projects to teach engineering skills in the Community Outreach section on page 22.

Passive IR sensor front-end design

By Nidhin Mulangattil Sudhakaran and Kendall Castor-Perry

Conventional passive infrared (PIR) sensor application circuits contain numerous bulky passive components. Shrinking the electronics down to the size of the sensor element requires some new design thinking. This article describes an approach using digital signal processing and closed-loop control of operating points to achieve DC blocking and filtering that matches the extremely low cutoff frequency of conventional analog circuits. It can be conveniently built using a programmable system-on-chip (PSoC) that's small, low-cost, and which can enable a wide range of differentiating features for smart sensor design.



Low-cost pyroelectric sensors find use in occupant detection and remote monitoring of heat sources. They consist of a ceramic crystal that's locally buffered by a JFET preamp that gets its drain current bias through an external load resistor.

The change in output voltage with incident IR is very small (~3.6 mV peak-to-peak, for the Murata sensor used to prototype the system) compared to the sensor's inherent offset voltage (more than 1,800 mV for a 5 V excitation). The offset voltage is highly sensitive to temperature. Therefore the interface circuitry has to differentiate between the variations in this offset voltage and the actual sensor output corresponding to a motion. The interfacing circuitry has its work cut out for it, as the signal corresponding to a motion is often less than 0.2 percent of the offset voltage.

The datasheets of PIR sensors usually show various distinct application circuits for different modes of operation. These circuits require significant numbers of bulky external passive and active components to filter out the low frequency offset voltage. The circuits provide a broad bandpass response to the AC signal coming from the buffered output of the sensor element. DC and extremely low frequency signals are suppressed by the highpass portion of the response, with cutoff frequencies often below 0.01 Hz, resulting in settling and recovery times measured in minutes! At the upper end, frequencies above about 50 Hz are suppressed, generally with a two-pole response (not by enough to cope with large amounts of AC line frequency modulation, though). The mid-band gain applied to the signals that do get through is anything from 10,000 to 40,000!

The PSoC system in Figure 1 uses a hybrid (analog-plus-digital) servo loop to remove the DC offset, resulting in a low-cost single chip solution that consumes far less area and volume, and can be integrated within a small sensor unit. This integrated sensor-plus-SoC package, the same size as a standalone sensor, can give high analog output levels, digital outputs (including LED drivers), and host connectivity options such as UART, I2C, SPI, etc. Since many functions are performed in the digital domain, using firmware or programmable hardware, the SoC solution is highly flexible. The integrated SoC can be re-configured in the field to meet the requirements of the application. In addition, the parameters of the servo loop can be re-tuned using a host controller during runtime, to adapt to the changes in environment.

Figure 2 shows the implementation of the servo loop using the PSoC's programmable analog and digital blocks. The start of conversion signal for the ADC is generated from the PWM output, using a frequency divider. This arrangement synchronizes the ADC conversion to the PWM servo feedback and reduces this noise component in the ADC output. The PWM runs at 5.8 kHz. The ADC sample rate is approximately 293 samples per second. In the servo loop circuit, Rpwm_1, Rpwm_2, and Cpwm form a T filter for the PWM output. Rg1 and Rg2 provide a DC bias that is close to the offset voltage of the PIR sensor. The PWM-filter circuit combination acts as a cost-effective, high-resolution 12-bit DAC.

The equivalent resistance from the inverting input of the Opamp to ground is close to 60 kΩ. Therefore, the gain of the non-inverting amplifier is approximately 113.

Mid- to high-frequency noise is controlled by a modest-sized capacitor connected across the feedback resistor to give a bandwidth of several hundred Hertz. This single gain stage is all that's required, because the rest of the system sensitivity will come from the ADC stage that follows this. Using a single amplifier saves power, reduces cost, and improves the overload behaviour.

The non-inverting input to the amplifier has a capacitor to ground and a resistor to an I/O pin that is driven by a 12-bit PWM DAC whose input word comes

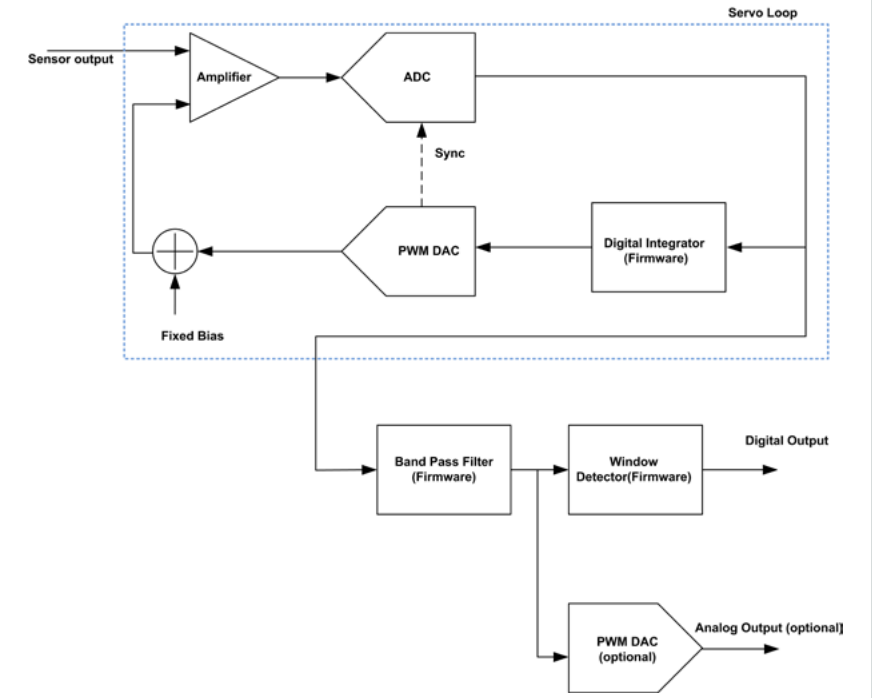


Figure 1 Cypress Semiconductor PSoC 4200 solution for the PIR interface circuitry.

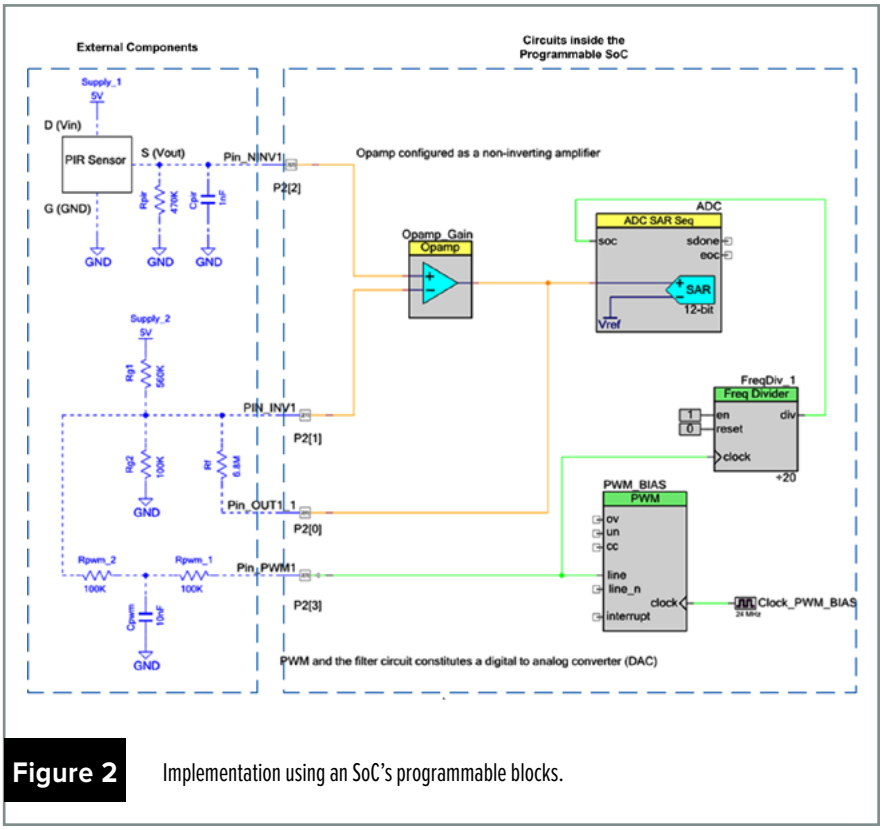


Figure 2 Implementation using an SoC's programmable blocks.

from a digital integrator, which could be implemented either in firmware or in programmable digital hardware.

The Opamp output drives the inverting input of the SAR, whose non-inverting input goes to ground. This connection inverts the sense of the converter output. The SAR is converting at a rate of around 293 sps. The ADC's aperture time can be left at the default setting, because the capacitor in the feedback loop rolls the closed loop gain off to unity well below the equivalent aperture frequency of the ADC.

The ADC output format is set to signed, so that an input voltage of 1.024 V (the reference voltage value) will result in a code of all zeroes at the ADC. The ADC drives the integrator, and when the loop is closed, the integrator output will move around to keep the mean value of the ADC code equal to zero. In other words, this process acts both as a highpass filter and as a working point stabilizer. It replaces the large capacitor value that would otherwise be needed to block the DC bias from the sensor. Because it's implemented in code, it can easily be paused, reset, or sped up, meaning that the system can stabilize very quickly after power-up. This is a significant advantage over passive circuits relying on a large RC time constant.

As well as feeding the integrator, the ADC's output data is applied to a first order IIR highpass filter at 0.005 Hz to 0.01 Hz, which eliminates any DC and low frequency information not completely suppressed by the servo loop. Following that, a suitable low pass filter blocks off whatever higher frequencies are considered undesirable, and extends the resolution of this stage well below the normal 12-bit resolution floor set by the ADC used here. At the sample rate involved, these filters can be implemented in the CPU. The overall response looks like a bandpass filter, obviously far more configurable in this digital implementation.

> "The amplifier, ADC, digital integrator, and the PWM DAC constitute a closed servo loop that removes the DC offset from the sensor output."

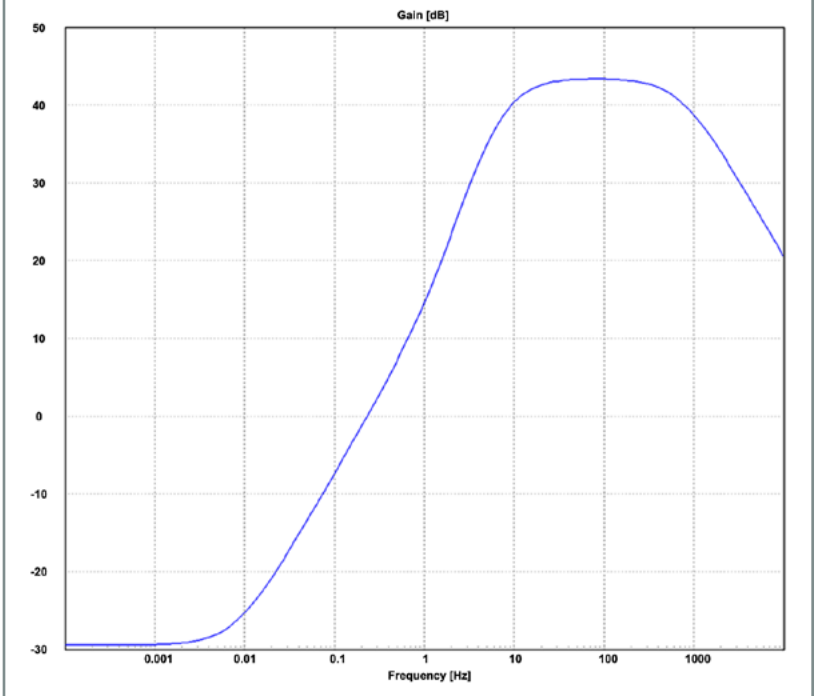


Figure 3 Frequency response of the servo loop.

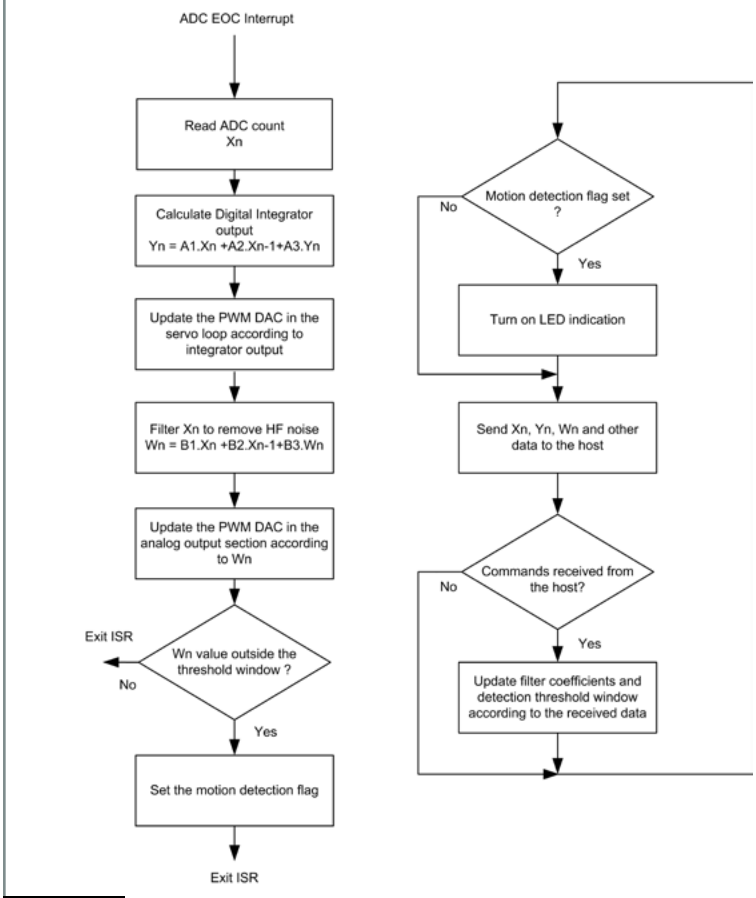


Figure 4 Firmware flowchart.

The amplifier, ADC, digital integrator, and the PWM DAC constitute a closed servo loop that removes the DC offset from the sensor output. When the sensor offset changes, the integrator accumulates the error signal and changes the amplifier bias via the PWM DAC, in such a way that the Opamp output (and the ADC count) is maintained at zero. Therefore, any change in offset is nullified. However, the change in sensor output corresponding to a motion will not be nullified because the servo loop is not fast enough to respond to sudden changes in sensor output. Typical frequency response of the servo loop is shown in Figure 3.

The filtered ADC value can be used in several ways. It can be compared against a threshold, and an alarm raised whenever the threshold is exceeded. It can be applied to timing circuits so that an alarm is only triggered when the level is exceeded by a preset time, and the alarm can then be stretched so that it lasts for a preset duration even when the stimulus disappears. Separate thresholds can be set for rising and falling values of signal. This allows the PIR sensor to discriminate between approaching and receding heat sources.

The digital output can also be converted back to a high-amplitude analog signal if required. The filtered ADC count data is appropriately scaled and applied to the input of another PWM block. The PWM signal is applied to a pin whose output is lowpass filtered by an RC filter and buffered using the other available Opamp in the SoC device chosen. This output can be passed to external circuitry as a replica of the input infrared variation irradiating the sensor.

The firmware flow of the project is shown in Figure 4. The firmware filters and integrators need to operate in real-time for the servo loop to work. Therefore, the firmware integrator and filters are included in the ADC interrupt service routine. Non-critical functions are in the outer loop in main.c.

In addition to running the servo loop and output routines, the firmware can send the data to a host controller for real-time monitoring, using the wide variety of communication blocks available in this modern programmable SoC, such as UART, I2C, SPI etc. The host can also re-tune the loop parameters during the runtime by sending commands back to the SoC.

Figure 5 shows a typical output of the system. The large variations in Wn indicate the detection of a motion. The filter coefficients and the threshold window can be adjusted during

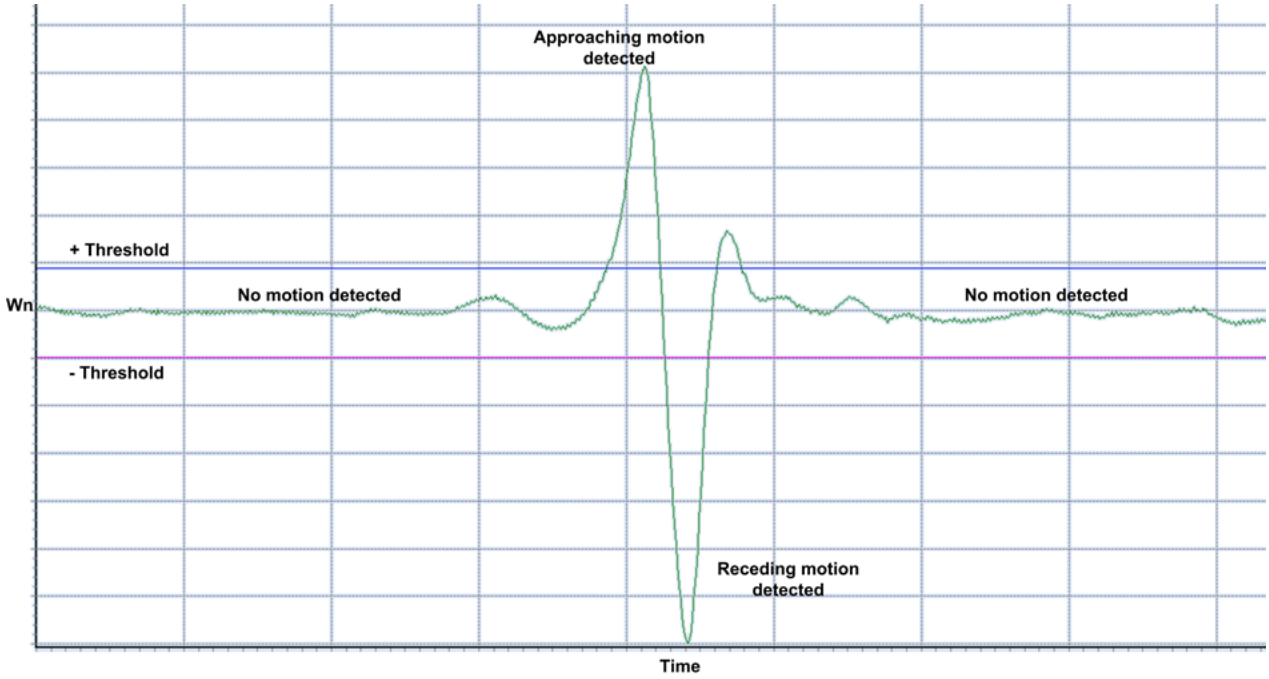


Figure 5 Typical output of the system.

run-time. This is useful for not only characterizing and optimizing the design during development, but also adapting it during operation. SoCs with integrated wireless connectivity such as Bluetooth are becoming available, ushering in a new no-wires approach to industrial and environmental sensing.

This simple application shows how the highly integrated nature of a modern programmable SoC, combined with the flexibility of the design tools, can result in fresh, contemporary solutions to established circuit and system design issues.

Nidhin Mulangattil Sudhakaran is an Applications Engineer at Cypress Semiconductor.

Kendall Castor-Perry is a Member of Technical Staff with Cypress Semiconductor.

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

Android beyond tablets and smartphones for embedded

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

Android has been championed by Google for quite some time in the smartphone market. Android is a Linux-based environment and Linux is a very popular software platform for embedded systems. The growth of Internet of Things (IoT) and machine-to-machine (M2M) applications and their associated graphics and communications requirements are causing engineers to explore Android for embedded applications beyond the smartphone.

I got the opportunity to talk with Sébastien Ronsse, Solutions Architect and seven-year veteran of embedded systems development with Adeneo (www.adeneo-embedded.com) about Android, embedded systems, and its challenges and opportunities. Ronsse works with companies who have questions about the best technologies to choose for their embedded applications.

Android overview

Android can be best thought of as a software environment that sits on top of Linux. Android has grown from its "API layers" start into an entire ecosystem. Android can be thought of loosely as its own distribution of Linux. The goa of Android is to allow the developer to easily create an entire product with all the packages you need for specific applications. Ronsse mentioned that there are an increasing number of customers coming to Adeneo and asking about Linux versus Android for their application. Ronsse mentioned choosing Android makes sense in some areas, but isn't always recommended.

"If you have a simple product with a small set of fixed functions, Linux makes more sense," Ronsse says. "Embedded environments that want to host multiple applications that need a nice shell with lots of eye candy and ready-to-use telephony, Wi-Fi, Bluetooth, and GPS are where the Android sweet spot is. These days, even embedded devices in industrial, medical, and automotive environments are expected to be interacted with in a similar manner to the smart phone. We all respond to paradigms we know and the smartphone has become a significant influence in our device interaction preferences."

The new Android version requires higher horsepower processors, so the ARM Cortex-A9 is usually preferred. Google selects a particular kernel version of Linux for Android. The Android Open Source Project (AOSP) becomes the starting point for all other Android distributions.

Think of Android's distribution as providing layers on top of the Linux baseline. The first layer consists of the native C/C++ libraries, SQLite database, Webkit (the browser rendering engine), along with OpenGL/ES for graphics rendering. The layer also includes the Android Runtime (ART) – Google's version of the Java Virtual Machine (CM).

"ART is what allows Android apps to run – the heart of Android – it's a virtual machine," Ronsse says. "Google developed a framework on top of Java which can be leveraged by developers – a location manager, a window manager, a resource manager, etc. Then on top of that there are some stock applications from Google – a default shell, contacts, and browser for example. Then of course with this run-time you have all the downloadable Android applications at your disposal."

Android is geared toward tablets and smartphones. Additional development advantages are that the application is portable and you can literally use your smartphone or tablet as your target machine for development. As more people develop for Android, familiarity with the APIs become more commonplace resulting in faster development.

Android for industrial

Android platform development is like any other porting of an operating system – it's not necessarily straightforward. Custom hardware designs require a port of Android to that platform. One option is to start with a platform design that runs Android, then adapt the platform and do an incremental change, but it depends on the embedded hardware needed.

Industrial, medical, and auto embedded systems need connectivity, I/O, or interfaces that probably don't have standard support in Android. In this case extending Android is required. If you're familiar with Linux, accessing hardware is typically done by opening the right driver node and communicating through the filesystem. But in Android you can't do that. The middleware portion of Android must be extended so you can make sure the application can talk with the hardware through the proper middleware/OS stack. Unfortunately, this is not straightforward or well documented.

"Five years ago when we started working on Android, it was hard to find where to start. We struggled to understand the architecture and there wasn't much documentation. Everything that was available was application development oriented, not the internals and architecture," Ronsse says.

Years ago this required looking through the internals to understand how it worked. Ronsse referenced the "Embedded Android" book by Karim Yaghmour, a good reference for Android platform development.

Silicon vendors are now providing porting guides for Android making embedded development easier. This provides more information, but still requires a learning curve, support, maintenance, and access to embedded systems experts to complete the work.

Specific to industrial control, the big factor is connectivity for M2M. Users are expecting to arrive in the industrial environment with their smartphone or tablet and securely connect to the industrial systems to get status via Bluetooth or Wi-Fi. These specific services are provided by Android, which makes the environment a good choice.

Headless embedded systems

Ronsse mentioned that you rarely see Android without a screen. Headless devices aren't popular for Android and the environment is not designed to work without a screen. However, there are projects out there that have customized a version of Android that will operate in a headless environment. Gary Bisson, also working at Adeneo Embedded, has successfully demonstrated this scenario at the Android Builders Summit.

Medical devices are becoming increasingly multi-function. These devices need multiple applications to record notes for patients, sensor recording, and patient vitals measurements. In this kind of environment where one device runs multiple applications, Android is a good choice with its shell and ability to load and run a variety of applications. You don't have to be an embedded expert to create an application that can be attractive in the embedded systems environment.

Traditionally, Java was the language you wrote in for Android. For embedded development, there may be a need for C/C++ to accommodate legacy code. The Android NDK supports C/C++ so that makes porting legacy code easier.

Embedded Android example

Many Android embedded projects tend to be adaptations. One Adeneo project involved a tablet design for restaurants where you can browse the menu, select a meal, play games while you wait, pay with a credit or debit card, and print a receipt on the tablet device at the table. Much of the tablet leveraged pre-existing capabilities and applications, but adaptations needed to be done for the card swipe, point of sale transaction, and receipt printing.

The porting starts with the creation of a custom boot loader. This is the initial program run by the processor. Then drivers needed to be developed for the card swipe and printer mechanism. Once these low level drivers are completed, the middleware layer of Android needs to be extended to expose these new services to the application layer. This requires the creation of a special Android SDK for the platform.

While the Android adaptation work was being done, the company could develop their restaurant application. APIs were defined for the credit card calls and printing so the application could implement them and simply stub out those calls until the platform was ready.

Processors, development, and debug

Processor support for Android is not as sparse as you might think. Of course the companies that support ARM cores (such as Freescale, TI, NVIDIA) support Android. Ronsse mentioned that Altera and Xilinx also have integrated ARM/FPGA chips that run Android. We are also starting to see x86 solutions that Intel is now actively involved in.

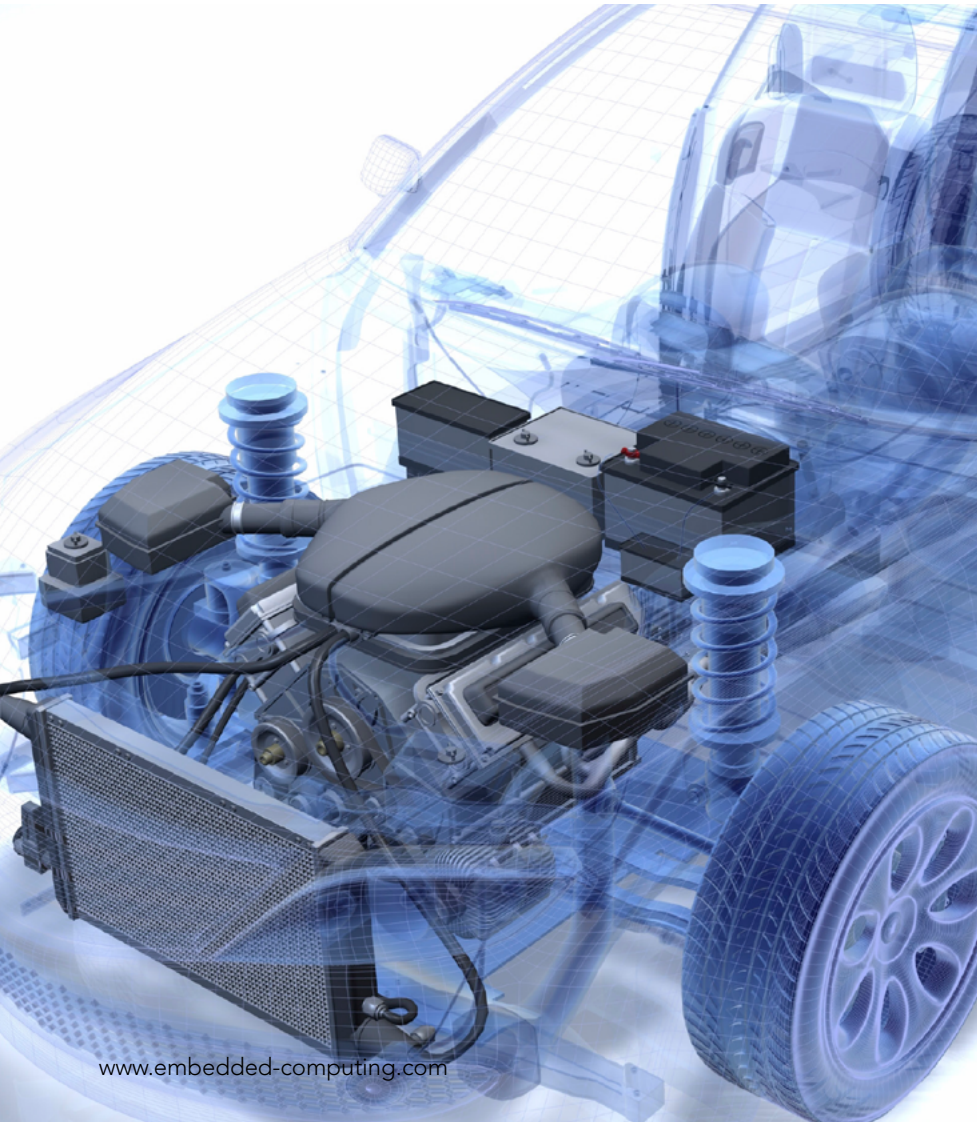
As you might expect, Linux-based development is the preferred environment for the lower Android layers while Eclipse and other Java-oriented IDEs can be used for application development on Windows, Mac OS X, and Linux. Application debugging is done via USB and ADB (the Android Debugger). When you enable Android debugging, you get access to a number of tools for application development. System tracing is also available that allows insight into the various threads of execution with HTML output providing a nice summary of the tasks for diagnosing races or problem areas.

Will Android sweep the embedded community like Linux has? For applications needing wireless connectivity, Bluetooth, and a familiar user interface paradigm, Android may fit. However, for applications that don't use these kinds of graphics and connectivity extensions that also have higher performance requirements, it may not make sense. ECD

Why your car should act more like your phone

By Alex Agizim

Making a call is probably the least important thing your phone does these days. We use our smartphones as cameras, game consoles, cash registers, social media portals, and more. Banking apps enable us to deposit checks simply by taking a photo of them, and medical apps enable us (and our doctors) to remotely monitor and analyze our health. We can even use our phones to operate our home appliances, like TVs and thermostats. However, the one thing our phones cannot do is fully integrate with our cars.



The disconnected car

While Bluetooth is a step in the right direction toward device integration, it is still a flawed technology that often cannot keep up with the devices it's trying to connect. Even Doron Elliott, Ford's Bluetooth global lead, admits, "Automotive works at one cycle and mobile devices work at another cycle."[1] While most new smartphones are developed within a year, a new vehicle's electronics were probably developed 4-5 years ago due to stringent certification processes. It's simply not cost-efficient for automotive manufacturers to be continuously re-testing and re-certifying their systems in order to introduce frequent software updates.

In addition to extended development lifecycles, the very infrastructure of a vehicle prevents automotive manufacturers from attaining the levels of innovation seen in other technology industries. Since all of a vehicle's functions are managed by computers that are physically located within the vehicle, there is a limit to how many features can be implemented. Each vehicle computer must also be rigorously tested, making it very difficult to reuse software across multiple machines.

Furthermore, a vehicle's computers must be manually accessed in order to analyze or repair software issues. Can you imagine bringing your iPhone to the Apple store every time you needed to update your operating system (OS) or install a software patch?

As the Internet of Things (IoT) becomes the new norm, with all of a consumer's devices connecting and integrating with each other, the automotive industry needs to take a page from the Apple and Google playbooks. Consumers are no longer satisfied with solo-functioning devices, whether it's a phone or a tablet or even a car. Of course, the million-dollar question is, "how does the automotive industry boost its innovation potential while remaining compliant with stringent regulations?" The answer is actually fairly simple: virtualization.

Why virtualization?

For those unfamiliar with the technology, virtualization leverages computer software, such as a hypervisor, to run multiple virtual computers (even those with different OSs) on a single piece of hardware. From Amazon's Infrastructure-as-a-Service (IaaS) offering to the U.S. Department of Defense's operational infrastructure, virtualization has become the go-to solution for secure, scalable computing.[2, 3] Furthermore, today's vehicle hardware finally has the capacity to support virtualization with minimal impact to performance or security.

As you can imagine, virtualization would open up a new realm of possibilities for automotive manufacturers and software developers alike. As mentioned earlier, vehicles are currently limited to a finite set of features based on the physical footprint of its multiple computers. Even with recent improvements in vehicle CPU processing power, there are only so many programs these computers can run. Creating virtual machines (VMs) that run off a single computer would enable manufacturers to offer an infinite number of features and passenger customization opportunities. For example, each seat in a vehicle could have its own unique VM, enabling passengers to customize their vehicle environments, from chair settings to radio stations. Backseat passengers could even watch separate movies on their LCD screens (a boon to family road trips everywhere).

Beyond passenger convenience, virtualization offers many time and cost benefits to stakeholders across the automotive landscape. Maintenance becomes more efficient since software repairs and upgrades can be done remotely rather than having to manually access the vehicle. Similarly, a virtualized environment significantly accelerates software development since everything runs on the same piece of hardware and therefore only needs to be tested once. Furthermore, since each VM is completely sandboxed, manufacturers can leverage different operating systems within a single vehicle (for example, QNX or AUTOSAR to manage mission-critical features and Android, Tizen, or Linux to manage in-vehicle infotainment (IVI) and introduce cutting-edge features).

Implementing a high-level OS (HLOS) like Android in a virtualized environment would also enable consumers to customize and "connect" with their vehicles through downloadable automotive apps. Imagine if your vehicle could send you a text message whenever its oil level or tire pressure was low. Imagine if you could start or even track your vehicle directly from your phone. Imagine if your garage door automatically opened when you pulled into the driveway.

In fact, some of these options are already available in the market today. The problem is that these services require consumers to buy third-party hardware that either they or their vehicle dealership must install. With virtualization, consumers would be able to simply download an app from an automotive-grade market. Just as you currently download apps to make your smartphone a more powerful tool for your lifestyle, so too could automotive apps expand the utility of your vehicle (Figure 1).

Integration challenges (and solutions)

As with most things, virtualizing a vehicle's software system is easier said than done. Part of the reason the automotive industry is so slow to innovate is its incredibly stringent safety and certification standards. Although HLOSs like Android, Tizen, and Linux would provide automobile manufacturers with innovative new machine-to-machine (M2M) and customization opportunities, they are inherently less stable than the QNX and AUTOSAR OSs that are used in vehicles today. This is why an intelligent system architecture is crucial to successful virtualization.

A sample diagram that illustrates how virtualization could be implemented in the automotive industry is shown in Figure 2. As you can see, a Xen Type 1 (in other words, baremetal) hypervisor sits on top of the vehicle's physical computer and manages various VMs. Related software systems are grouped together in function-specific machines that are completely sandboxed from each other. This means that even if your Android-operated IVI system crashes, your mission-critical drivers – which are sitting in a separate machine and running on the ultra-reliable QNX or AUTOSAR OS – will not be affected.

Figure 1 Proof-of-concept for an automotive-grade application that turns a consumer's smartphone into a key fob.

Although a hypervisor like Xen offers the greatest functionality (and reliability) for running multiple VMs, it's difficult to certify because it's open source software. With a global community of developers continuously developing and improving the hypervisor (and using different processes and technologies to do so), it cannot pass ISO 26262 certification. While one solution is to use a private hypervisor, it cannot offer the same level of innovation (nor time-to-market) that stems from tapping into a global community of developers. Private hypervisors also typically come with hefty per-unit royalties. The challenge, then, is to align the dynamism of the open source community with the stability of the automotive industry's standards.

To address this challenge, GlobalLogic, Inc. is currently developing a unique procedure for certifying an open source hypervisor. After performing a detailed

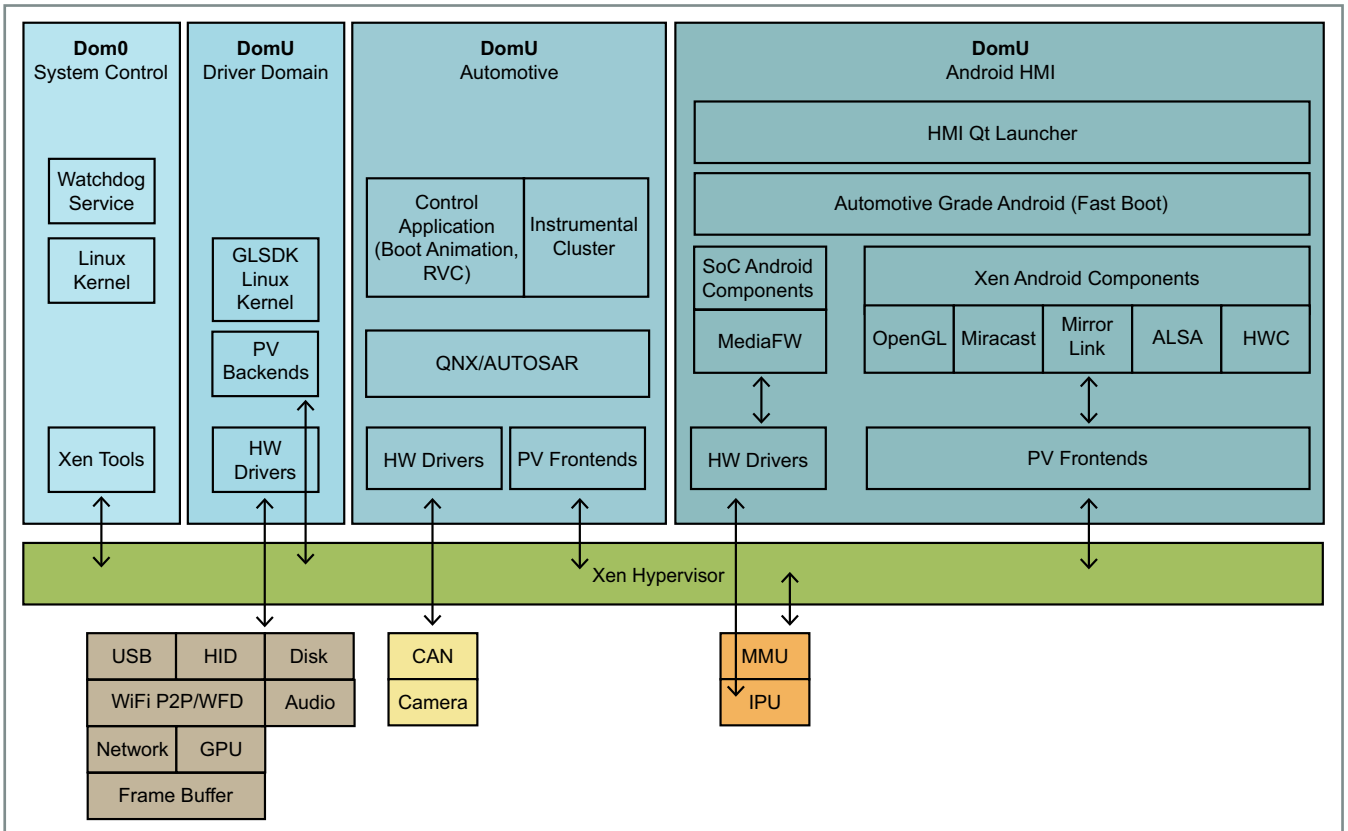


Figure 2 Example of a virtualized automotive software environment.

risk and gap analysis for a specific version of the Xen hypervisor, the hypervisor is then modified and verified to meet ISO 26262 standards. Over time, the Xen version can also be re-analyzed and re-certified as the open source community adds new features.

Of course, the industry will soon need to create a whole new set of standards to screen potential automotive-grade applications. The whole point of implementing a HLOS within a virtualized environment is to enable consumers to customize their vehicles the way they do with their other smart devices. This means creating an app market like iTunes or Google Play that is specifically focused on the automotive industry. Although it's possible that some manufacturers may opt to develop all their customization apps in-house, this approach would be very time-consuming and cost-prohibitive – not to mention a major roadblock to innovation.

On the other hand, adopting a free market approach to automotive apps poses some inherent risks, such as bugs hitching a ride into a vehicle's IVI or instrumental cluster system. Although mission-critical drivers would not be harmed due to the sandboxed architecture, a virus could still wreak havoc on the affected module. To this extent, a major challenge to automotive manufacturers in the future is to create a strong system for screening app submissions. Manufacturers must also architect their virtualized systems in such a way that consumers cannot "jailbreak" their vehicles. While jailbreaking a smartphone rarely results in more than potentially lost revenue for service providers, jailbreaking an automotive system could result in user safety concerns.

A truly "connected" car

Although reliability, security, and certification issues may deter the automotive industry from being completely in-sync with faster-paced industries like mobile, a virtualized approach to automotive software systems can get it remarkably close. By leveraging an ISO 26262 certified hypervisor to run a system of sandboxed VMs on both industry-standard and HLOSs, automotive manufacturers can design vehicles that are both safe and cutting-edge. Because in an age where a phone is more than just a phone, shouldn't a car be more than just a car?

Alex Agizim is Chief Technology Officer of Embedded, GlobalLogic Inc.

GlobalLogic, Inc.
www.globallogic.com
@GlobalLogic
linkedin.com/company/globallogic

References:

1. http://www.edmunds.com/car-technology/bluetooth-in-your-car-still-indispensable-still-imperfect.html
2. http://openfoo.org/blog/amazon_ec2_underlying_architecture.html
3. http://www.vmware.com/files/pdf/vmw-DH-DW-desktop-solution-DoD-case-study.pdf

COMMUNITY OUTREACH

Bringing creative engineering to students

By Monique DeVoe, Managing Editor

Shawn Jordan, Assistant Professor at Arizona State University's Fulton Schools of Engineering, has combined his Rube Goldberg proficiency and passion for making and teaching into the STEAM Labs™ program (steamlabs.asu.edu), which challenges middle school and high school students to apply the engineering design process to create and build Rube Goldberg-like chain reaction machines.

"It teaches engineering skills, systems thinking, and collaboration, and integrates the arts with the STEM fields of science, technology, engineering and math," Jordan says. Adding arts to the traditional STEM acronym transforms it to STEAM.

STEAM Labs™ brings deeper opportunities for creativity not often found in engineering outreach program activities.

"Rube Goldberg Machines engage students on multiple levels to design both the problems they want to solve and the solutions for those problems (similar to the maker movement)," Jordan says. "This is different than many of the standard engineering outreach activities, where students are given a specific problem to solve. This environment creates an opportunity for creativity, imagination, and making dreams of inventions a reality."

Scholars in engineering and gifted education have developed the program over the past seven years, and it has been deployed to more than 2,500 middle and high school students in the U.S. and Trinidad and Tobago. Students work in face-to-face and virtual teams at camps to build chain-reaction STEAM Machines™ in a project-based, cooperative learning environment with online collaboration tools.

Engineering design will be a requirement in science classes beginning in fall 2015 as part of the Next Generation Science Standards for K-12 education in the U.S. STEAM Labs™ is designed to help students better understand engineering career possibilities in addition to learning real-world engineering skills.

"The program challenges students to not only ask 'why?' but also 'why not?' – a question that I think is all too often lost in today's youth," Jordan says. "This in turn helps students understand that you can be creative and be successful in engineering – an important message, given pop culture's less-than-flattering messages about engineering." ECD

Open source software everywhere?

By Robert B.K. Dewar

These days, the use of open source software is amazingly ubiquitous. The planes we fly on every day use safety-critical software developed with open source tools, as well as open source library code. In Europe the air traffic control systems have been developed using open source software. Everything from kitchen appliances to advanced weapons systems is powered by open source software. For some embedded systems, such as those using ARM technology, often the only available development tools are open source, and you can for example get a complete Ada language environment for the ARM chip, including tools for safety-critical certification and formal proof techniques.

If for some reason you decided that you did not trust open source software and wanted to avoid it completely, you would be pretty much stuck. You couldn't fly, you couldn't turn on your computer (even highly proprietary systems like Windows incorporate some open source software), you couldn't drive your car, you couldn't use the ATM machine ... All these systems incorporate some open source components, or are built using open source tools. At this stage, it would probably be easier to survive using only open source software than to avoid it completely.

So what are we to make of this? Is there some amazing software revolution in progress? Before we try to answer that question, let's understand a bit better what the term "open source" means. In fact it's a rather confused term, since it involves two related but different concepts. First we have the notion of licenses that provide much more freedom to users. Richard Stallman coined the term "Free Software" to capture this idea, where by "Free" he referred to freedom, not a $0 charge (languages like French which distinguish the words "libre" and "gratuit" avoid this English confusion). He prefers to avoid the term "open source" because it obscures this important notion of freedom, and to reserve the term to refer to the development methodology where software sources are freely available and where many different people – from high-school student hobbyists, to top professional programmers paid by major companies – work on the development.

In practice the two aspects of open source – licensing and methodology – tend to go together, because a necessary component of the open source software notion is a licensing approach free enough to permit this kind of open development, and a key concept behind Free Software is making the sources available, which makes open development possible. We like to use the term "FLOSS" (freely licensed open source software) to capture both notions.

What does this ubiquitous availability of FLOSS tools and code mean to a developer? If you are a student or a hobbyist, it means you can find freely downloadable software on the web (free in both sense of the term), and develop interesting additions to the FLOSS infrastructure. For example, the Ada-based development system for the (bare-board) ARM chip is freely downloadable, and we know of one university where students are developing interesting new drone software technology using this system.

But what does it mean for the developer of "serious" software? We put "serious" in quotes here because FLOSS software developed by students can very definitely be serious. For instance, the CubeSat developed at Vermont Technical College using freely available Ada technology, launched with other CubeSats last November as part of a NASA-sponsored effort, is now orbiting the earth and is one of the few CubeSats from that launch still operational, and it is sending beautiful pictures back to Earth. So let's narrow this down to the development of commercial systems, such as critical avionics software.

From one point of view, the availability of FLOSS tools and code really doesn't make any difference. How software is developed has little to do with its quality or suitability, and when it comes to licensing, freer license terms can only help and not hurt. So check out available software, evaluate whether it is suitable for your use from a reliability/quality/legal point of view; whether or not it is FLOSS is really quite irrelevant.

First, in the minds of many, including sometimes those who make important policy decisions at large companies, "open source" means downloading free (in the $0 sense) software from the Internet and using it without proper qualification and vetting. The notion of hundreds of software developers furiously downloading stuff from the Internet in a completely uncontrolled manner seems like a nightmare to those trying to control things. Are they right to be concerned? Absolutely, there are many problems in just pulling stuff off the Internet. First, you have no idea if it is licensed correctly (you absolutely cannot rely on copyright notices in files that you download, they have no legal significance). Second, you often end up with software that is totally unsupported, and that can be a major problem in a critical development environment. Third, you have no idea if the quality meets your requirements.

It really isn't FLOSS per se that is the problem here. If a company allowed programmers to freely spend money on any proprietary tools they wanted, they would be in an equal or perhaps even worse mess. But the $0 price tag means that this can go on without the normal control by management and purchasing departments. So indeed this has to be brought under control, and all companies need policies on software acquisition that make clear what is and what is not allowed.

Is an appropriate policy reaction to forbid the use of FLOSS entirely? No. In the first place there are many situations where you would be completely stuck, e.g. embedded chips where the only available compiler technologies are open source GCC based. Second, such a policy makes no sense. If we compare FLOSS and proprietary software, we find that there is great/horrible software in both categories, great/horrible support in both categories, and appropriate/inappropriate licensing terms in both categories.

What you need are policies that only permit the use of appropriate software; whether it is FLOSS or proprietary is really a secondary consideration. First you need to check licensing conditions carefully, and educate your programmers on proper usage in this regard. Just because something is FLOSS does not mean you can use it freely without any limitations. Although the concern that somehow you can lose all your intellectual property (IP) rights by using FLOSS is bogus nonsense spread by large companies developing proprietary alternatives, it definitely is the case that, just as with proprietary software, you can find yourself in copyright violation situations if the licensing is inappropriate, and that's bad enough!

Further, you need to carefully evaluate any potential tools or libraries that you consider using, and this needs to be done with care, considering quality and reliability of the software, suitability to your needs, quality of the support, track record in developing useful new capabilities, etc. This is a difficult task indeed, and FLOSS does not somehow make it magically easier, but it also does not make it more difficult!

We live in a software ecosystem where FLOSS plays an increasingly critical part. Learning how to take maximum advantage of everything that is available in this ecosystem is a critical part of effective software development, and every manager and programmer needs to be aware of how to make best use of all available tools and components, including most certainly those that come from the FLOSS world.

Robert B.K. Dewar is President of AdaCore.

AdaCore
www.adacore.com
@AdaCoreCompany
linkedin.com/company/adacore

Rapid deployment through open platform customization

By Scott Wilken

Basing a product on a proven open-source platform has great advantages in development time reduction, especially for products that do not change in hardware configuration once deployed. However, open-source offerings as they exist today are largely insufficient for deploying products that require field extensibility.

There are many sources of both existing hardware and software that can be incorporated into new designs available from the open source community. Commercially viable open-source artifacts range from operating systems software such as Linux to physical hardware devices such as BeagleBoard. While basic building blocks of this sort can generally be used without charge, they are still subject to the terms under which the original creator released the design, whether that be the GNU Public License, Apache, or similar.

The main disadvantage in extensibility is that the traditional paradigm is to build one board support package (BSP) for the underlying hardware platform and then build all applications on top of the tested software platform that has abstracted the hardware from the application developer. If the underlying hardware eventually needs to change in the field, then in most instances, housings need to be opened, circuit boards replaced, and a new BSP must be built, tested, and installed.

The first step to overcome this disadvantage is to tightly define a set of lowest-level hardware interfaces that can be exposed through detailed electrical and mechanical documentation and supported at the lowest layers by locked-down driver code that becomes part of a published BSP. This needs to be planned for and accommodated in a product's industrial design. In most cases, this implies more than simply selecting a ubiquitous USB connector and assuming extensibility has been accounted for. USB can pose practical environmental issues in many markets, such as lack of suitability to high-vibration usage for an example. There is also a potential negative impact to the aesthetics of the product when a previously unknown USB device is attached as a new protrusion from the core device.

The approach of defining a common electrical and mechanical expansion interface has been used successfully in the past by the personal computer industry. The earliest models of PC machines shared ISA bus commonality between IBM's offerings and the many clones that followed. Unfortunately, current PC expansion slot strategies do not translate well into most embedded applications for a variety of physical reasons, and embedded designs lack the commonality of a PC BIOS to make disparate hardware look relatively uniform to a software developer. Accordingly, deployment of software support for field-added hardware can be more difficult for embedded products.

The PC model for deploying driver software is also still generally better than what is industry standard in embedded products. Modern PC operating systems including Microsoft-based and various Linux distributions automatically read device information as new hardware enumerates and retrieve the associated software drivers from a central repository automatically. To an end user, this is usually a very satisfying experience that needs to be recreated if field extensible embedded devices are to be successful.

The combination of easy to develop hardware coupled with software support that is easy to deploy with minimal impact to a field installer has been lacking from the embedded space on top of which most end products are actually developed. With a well-thought-out design approach that considers both platform independence of hardware

expansion options as well as a centralized mechanism to deploy software, it is possible to replicate the PC approach in the embedded space within a family of products.

Although this approach to creating customizable devices from standardized platforms can be applied to many devices that need field configurability, it is particularly important in devices centered on connectivity for the Internet of Things (IoT) space. When considering a generic monitoring and control application, it is easy to design a communications gateway that implements a few standard interfaces, but it is far more difficult to be able to supply one that communicates with every sort of sensor and actuator in existence. The standard approach has been to pick a market and utilize the interfaces that are most common within that market. With field extensible platforms, the OEM no longer has to pick a particular market to pursue. If additional sensors or actuators need to be supported, then a new expansion card can be created to augment the original system and any new enabling software can be quickly deployed – the underlying physical interfaces are already natively supported at the lowest level with the API for the low-level drivers having already been made available freely.

Figure 1 shows an open IoT gateway product that is extensible only through GPIO, USB, or other interfaces that are fixed at the time of manufacture. As a traditional embedded device, it has a customized Linux BSP in order to make the most of the flexibility offered by the hardware and does not have significant post-release expansion capability. USB offers a route for adding features, but, as discussed previously, imposes limitations for practical field deployment.

Taking a new approach for a second-generation product, expansion ports have been created so that in addition to a traditional USB port that could be used in those instances where it is mechanically practical, there are also many other interfaces available to be exposed on expansion cards including USB itself, UART, SDIO, I2C, Wi-Fi, and others. This helps to ensure that the gateway architecture will enjoy a reasonable degree of longevity in the platform design by being able to accept new expansion port cards as they become available. As an open strategy, generally the OEM will want to document and publish the definition of the accessory port as well as providing the interface software support freely so that external sources both including end-customers and also other OEMs can develop additional hardware that the original hardware vendor had not thought of at product inception.

The next step is to identify reasonable ways to leverage this standardized hardware interface while providing a level of abstraction that allows other software developers to create applications without having to develop low-level drivers or build a new BSP from the ground up.

For a less computationally intensive device, ARM has a great answer with their mbed approach, which has been created to support their Cortex-M series of devices. The SDK is provided to the community by ARM under the Apache 2.0 open source license and feels very familiar to those who have worked with OS and non-OS embedded software development in the past. With mbed, software support for hardware can be placed in a central repository for any potential user to retrieve and build with their application. ARM themselves provide a level of platform independence at the processor I/O level through mbed, so it is possible to migrate code from one supported mbed device to another, even when the devices come from different microprocessor vendors. User-created software with the right platform support can then be built easily with ARM's tools, or with other third party development environments that are available for sale commercially. ARM's native deployment mechanism for deploying mbed code is as easy as drag-and-drop over a USB interface but the OEM may choose to extend deployment in a way more suitable to their own need if the devices tend to be deployed remotely and get field upgrades.

Since computationally intensive applications may not run well on a Cortex-M processor, an alternate approach may also need to be considered. Linux is one obvious choice as most

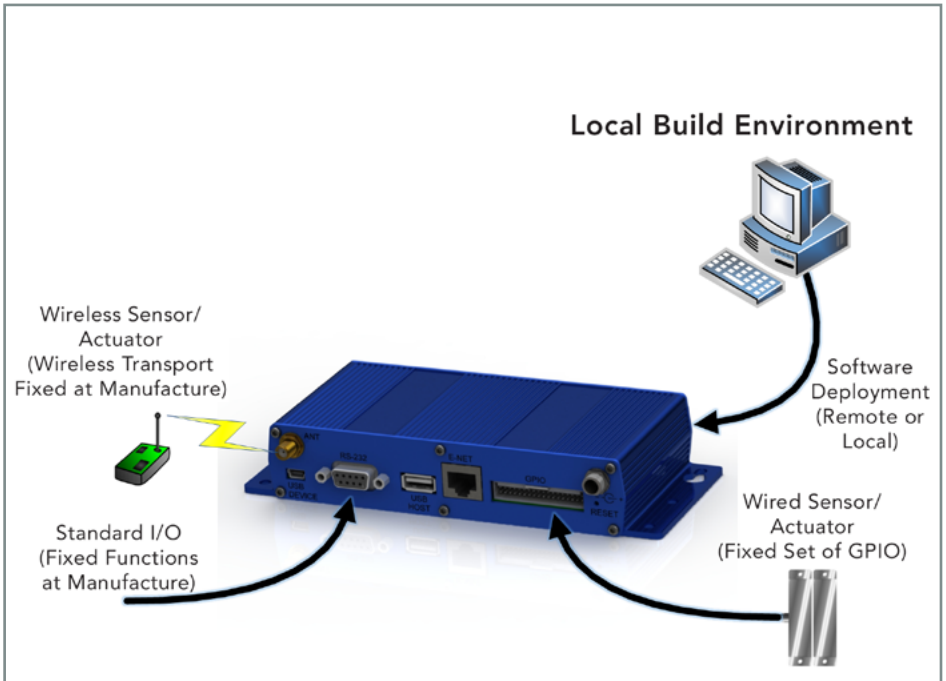


Figure 1 Traditional embedded approach using hardware fixed at release.

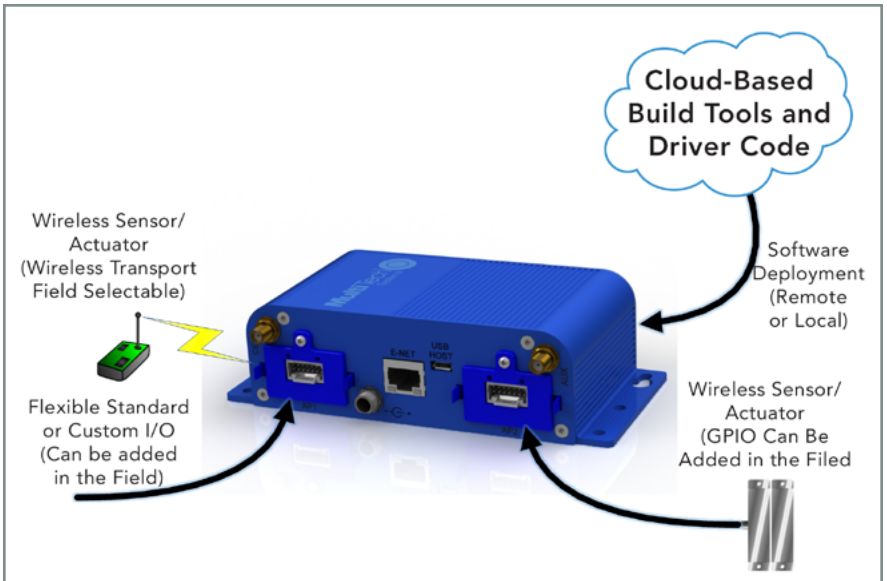


Figure 2 Open platform with flexible hardware expansion and freely-provided interface driver support.

providers of higher-end embedded processors tend to release an initial BSP for their own development kits into the community and often alongside open source hardware made available at initial release. The challenge in this approach is in recognizing that either an embedded Linux savvy software engineer will be needed for any customizations or an abstraction layer will need to be built. A hybrid model is to support both enablement of the more limited set of embedded Linux developers who are comfortable with building a full platform on their Linux development system as well as engineers who operate a level above that in Java, Python, Perl or other languages.

Figure 2 shows a MultiTech gateway that is one of a family available in late 2014 that will have openly documented hardware expansion slots. The expansion interface is common to multiple gateway products, and all supporting low-level driver code will be open and provided for both mbed and Linux. This allows both OEM and third parties to create new expansion boards that are easy to bring up and deploy into a much wider array of applications.

The expansion cards shown in Figure 2 are defined with a mechanism for personality so that the platform knows what driver resources are required and if any additional software needs to be retrieved from a cloud-based repository. The same cards work in any platform within the product line, allowing the end developer to select what level of processing capability is most appropriate to the individual application. In both cases, a general embedded software engineer would feel comfortable developing for these as the model follows the traditional PC paradigm of expandable standard hardware interfaces with software that exists on the main device natively or is download automatically from a central repository. The key advantage is that this approach creates platforms that are easy to understand by a wide range of software engineers and are based in open source artifacts that already have widespread community support. This results in products that can be rapidly customized both before and after initial field deployment in a vastly reduced amount of time. Although the examples provided center on IoT applications, the general approach maps well to any application where field configurability and support of a wide variety of known and unknown features needs to be considered.

Scott Wilken is Chief Technology Officer at MultiTech Systems.

MultiTech Systems
www.multitech.com
@MultiTechSys
linkedin.com/company/multi-tech-systems

MAKING DIY PROJECT SOFTWARE

You've got the hardware to make your own DIY projects, but what about the software to make it do what you want it to do? Though some makers can design any system from scratch, others could use a little help getting started or with more complex projects. MathWorks offers MATLAB and Simulink geared toward making, and offers more resources at its MakerZone. *Embedded Computing Design* interviewed Paul Kassebaum, Manager, Maker Community Relations at MathWorks on what the company is doing in the maker space.

PAUL KASSEBAUM
MathWorks

Q How did MathWorks get involved in the maker/DIY space?

MathWorks has always had a mission to engage students with engineering and science at formal institutions like colleges and universities and more informal playful spaces like our sponsorship of student competitions. MathWorks first expanded its support of education and innovation into the maker movement by sponsoring one of the world's largest makerspaces, Artisan's Asylum near Boston, and also jointly organizing an Autonomous Robot Design Challenge as part of the 2013 and 2014 Cambridge Science Festivals. Since then, we've gone on to sponsor other makerspaces such as Makespace and the Stockholm Makerspace. We've also been taking part in Maker Faires around the world including the Paris Maker Faire, where we organized an autonomous Mars rover competition. We are currently preparing to exhibit at the World Maker Faire New York in September 2014.

Q What are some of the typical challenges people run into during board bring up and development with open source/DIY hardware, and how do MATLAB and Simulink help alleviate some of those challenges?

Most of the time spent on hardware projects is consumed by iterative testing of algorithms on the hardware. MATLAB provides instantaneous manipulation and analysis of input and output to rapidly prototype algorithms. Simulink is at once a simulation and development environment for your algorithms allowing you to closely couple mathematical models of your system with the algorithms that are meant to run on your hardware by automatically generating code based on high-level visual representations of your system. Combined, these two features reduce the painful process of iterative testing.

Additionally, most DIY hardware is supported through MATLAB and Simulink Hardware Support Packages, easing the installation process for makers and getting them up and running sooner.

Q Are the tools accessible to novice hobbyists, students, and younger developers, or geared toward professional engineers? What types of DIY projects are MATLAB and Simulink best suited for?

MATLAB and Simulink are available for students and hobbyists on our website through two offerings: MATLAB Student and MATLAB Home.

MATLAB is best suited for projects that involve logging and analyzing data, such as a DIY weather station (Figure 1). Simulink is best suited for controlling systems such as a line following robot.

Q What is MakerZone and how is it unique from other DIY communities?

MakerZone is a website to learn how to get started using MATLAB and Simulink to design and program hardware projects. Many other DIY communities focus on physical design and low-level programming, based around tools that become cumbersome as the complexity of your projects grow. MakerZone highlights projects and programming methods that will carry you from getting a LED to blink to designing sophisticated systems. Our support also extends to MATLAB Central and File Exchange, which offer thousands of useful downloads contributed by our community members.

Q There are ports available for MATLAB and Simulink on Arduino, Raspberry Pi, and LEGO MINDSTORMS. Why these three platforms? What are the costs and features of the MATLAB and Simulink platforms used? Are there any plans for other platforms?

The Arduino highlights control projects, the Raspberry Pi highlights data analysis projects, and LEGO MINDSTORMS provide the shallowest learning curve for those new to electronics, letting them focus on programming.

Used by permission, © 2014 The LEGO Group

Low-cost hardware has helped to make engineering more accessible to people of all ages and backgrounds. We want to support this accessibility so we're offering these additional packages built on top of MATLAB and Simulink at no extra charge. MATLAB and Simulink Student and Home empower students and hobbyists with all the features used by professionals.

MATLAB Student and MATLAB Home can already interface with an extensive list of hardware far beyond the three platforms we highlight on MakerZone. MathWorks is always listening to the market to add other hardware to our offering.

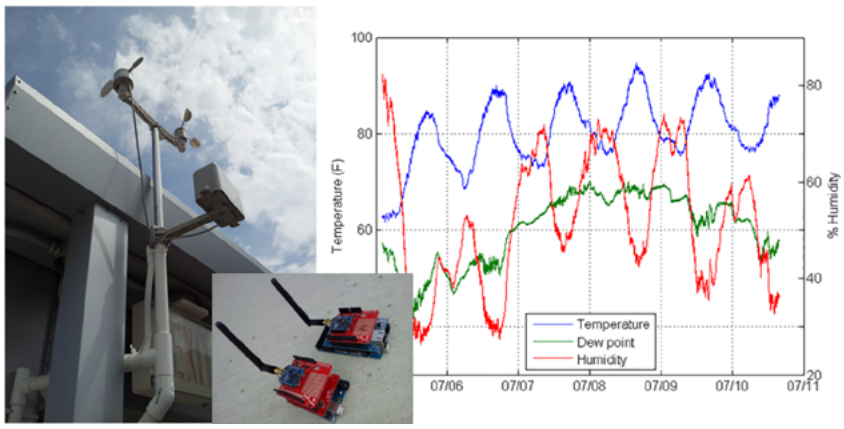


Figure 1 A weather station data analysis project using an Arduino. Image courtesy of MathWorks.

Q What is the MATLAB and Simulink Student Design Challenge and how can students get involved? What should students know before getting started?

The MATLAB and Simulink Student Design Challenge is open to college students at the undergraduate and graduate levels to submit their capstone or thesis projects that utilize our tools in interesting and innovating ways for a chance to win cash prizes.

MathWorks

- www.mathworks.com
- makerzone.mathworks.com
- @MATLAB
- facebook.com/MATLAB
- plus.google.com/+matlab

DIY Developers Form A Sophisticated, Diverse Group That Drives Technology

By Colin Cureton, GizmoSphere Spokesperson and Embedded Product Manager at AMD

Home-based tinkerers know what they want and aren't afraid to dive into big, complex projects. Gone are the days of purchasing a do-it-yourself (DIY) board just to program it to toggle an LED on and off. Today's DIY developers want a versatile development board they can incorporate into a whole system to drive sophisticated applications, everything from pothole detection to home automation.

These developers are savvy in that they're fluent in at least one area of development, whether it's hardware design, code writing, firmware optimization or something else, and whatever expertise they're missing they can hunt for and find online. They are experts at systems integration, willing to pair their specific proficiency with general knowledge and research to get the results they're looking for.

While many tinker for the fun of it, others show an entrepreneurial streak. To be sure, they enjoy development enough to do it in their spare time whether or not there's financial gain to be had. But if their tinkering results in a potentially marketable idea, they'll pursue it with the goal of creating a side business, or even a full-time venture.

Whether a one-person show or a team coordinator, a code developer or a circuits expert, a technology aficionado or a business tycoon, the DIY developer prefers to work with a board that's open, accessible, and powerful enough for advanced applications in network control, robotics, cloud computing, digital signage, and other areas.

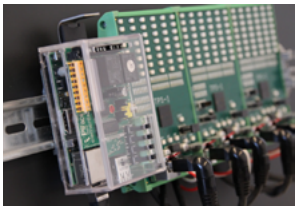
That's why GizmoSphere's x86 development board, the Gizmo, is a good choice for today's developers. It is powerful enough to drive high-level applications while offering tremendous flexibility and choice in operating systems (OS) and hardware integration. It provides a variety of bus interfaces plus both programmable and dedicated connections.

What's more, the Gizmo DIY board is completely open source, offering schematics, software downloads, and even a bootloader. Anyone who wants to duplicate the board for production can do so. The board's open-source status encourages dialogue between developers around the world, leading to discussions that foster innovation and drive technology into new territory.

DIY PRODUCT SPOTLIGHT

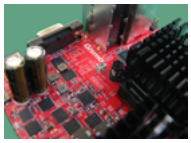
DIY Product Spotlight IEC 61131 Starter Kit for Raspberry Pi

- KW-Software provides full featured, low cost software platform for the Raspberry Pi that supports the following:
  - ❍ A simple and cost effective introduction into the world of IEC 61131 Programming
  - ❍ MULTIPROG programming system for the development of logic applications
  - ❍ ProConOS embedded CLR (eCLR) PLC runtime system
  - ❍ PROFINET Controller Stack and PROFINET Configurator
  - ❍ Quick start guide featuring an example application and I/O driver for the PiFace



Gizmo Development and Evaluation Board

- Versatile DIY development board powered by the AMD Embedded G-Series APU, which includes on-chip discrete-level graphics that support multi-screen HD displays.
- First x86 development board to be fully open source end-to-end – hardware, software and even bootloader – with design files and key FW/SW downloads available at GizmoSphere.org.
- Low-power, high-processing standalone board performs at a robust 52.8 GFLOPs.
- Compatible with Linux plus Windows 8 and previous versions (including Windows 7 and XP), RTOS, Android, and other OS.
- Developer-created Gizmo applications include automatic pothole detector, self-managed cloud infrastructure, and home automation system.
- Makes the most of lightning-fast 64-bit processing power.application and I/O driver for the PiFace



Koala EVM

- STM32F4xx based EVM with COM8 TI WL8 boards or WMI wireless modules
- Full Bluetooth, BLE and WiFi protocol stack software available
- Choice of RTOS – ThreadX or uC/OS-III, (on roadmap Nucleus, FreeRTOS)
- Ethernet, USB-OTG, USB-UART bridge, optional LCD, 1Mpixel camera
- Easy debug with STLink JTAG connection (IAR, Keil or GCC)
- Clarinox PC based debugger and protocol analyser

Take your DIY projects to the next level with UDOO

- Low cost, low power consumption tiny single-board computer
- Double processor ARM i.MX6 CPU Quad/Dual core 1GHz + ARM SAM3X8E Atmel
- Open source and open hardware system
- Multi development platform solution for Android, Linux, Arduino and Google ADK 2012
- Compatible with Arduino IDE and shields
- Dedicated OS Distribution: Lubuntu 12.04 LTS armHF, Android 4.3 Jelly Bean, XBMC, Debian Wheezy armHF, Yocto, OpenMediaVault, Volumio, Archlinux

www.mscembedded.com

MSC Q7-TI8168

The innovation of the MSC Q7-TI8168 is the combination of an ARM® Cortex™ A8 RISC MPU with TI's C674x VLIW DSP core offering up to 8000 MMACS. This powerful product is especially suited for Medical/Industrial Vision Systems, High-end Test & Measurement, Tracking and Control as well as Medical/Biological Imaging.

The MSC Q7-TI8168 MPU Module incorporates a high performance C6A8168 MPU @ 1.2 GHz with DDR3 memory, GbE and industrial interfaces including Flash memory. The module fully supports the Qseven Standard Rev. 1.2 interface, allowing simple integration with our range of Qseven baseboards or custom developments.

For evaluation and design-in of the MSC Q7-TI8168 module, MSC provides a number of different development platforms.

FEATURES

› Texas Instruments TMS320DM8168:
- ARM Cortex™ A8 CPU up to 1.2 GHz
- DSP Subsystem C674x up to 1.0 GHz
- 1GB DDR3 SDRAM
- 2GB Flash SSD soldered on board
- Gigabit Ethernet
- 1x PCI Express x1 port
- HDMI/DVI up to 1920x1080 resolution
- Single Channel LVDS 24 bit up to 1280x720 resolution
- Dual Independent Display support
- Two SATA-II interfaces
- Six USB 2.0, HD Audio

MSC Embedded Inc. | 650-616-4068

Contact: info@mscembedded.com
embedded-computing.com/p9915999

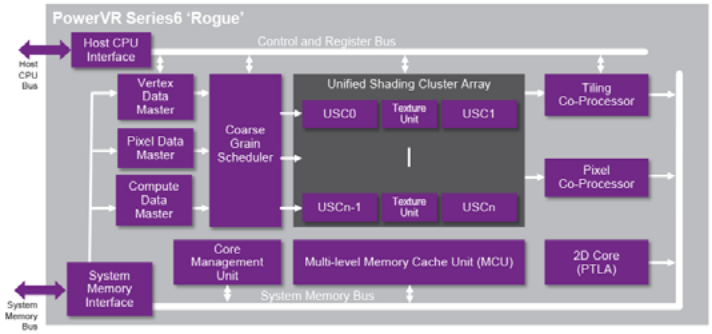
Video and image analytics DSP, evaluation board

The Texas Instruments C5517 DSP provides 200 MHz performance at less than 0.45 mW of standby power to provide a foundation for signal processing in analytics, medical, mission critical, and ratio applications. The DSP also includes peripheral options for McSPI, McBSP, MMC/SD, USB, ADC, and UHPI. The C5517 EVM is an evaluation module image that includes hardware and software for evaluation of image and video analytics applications. Software development is made easier with the Code Composer Studio v5 – an Eclipse-based development and debug environment with plug-ins that support TI microcontrollers, ARM, and/or DSP processors.

Texas Instruments | www.ti.com
embedded-computing.com/p372352

Advanced GPU capabilities for OpenGL ES 3.0, OpenCL 1.1 EP

The Imagination Technologies PowerVR Series6 is a GPU IP core family for high performance, low power graphics from mobile and tablet to high end gaming and computing. The PowerVR Rogue architecture is based on a scalable number of compute clusters – arrays of programmable computing elements designed to increase performance and minimize power and bandwidth requirements. The IP also incorporates a patented tiling technology (Shader-Based TBDR) that enables on-chip processing of hidden surface removal and pixel blending. This eliminates unnecessary processing and off-chip memory access for graphics that are not visible. The hardware-based tiling and culling algorithms alongside scene complexity management and compression provides efficient, scalable handling of complex scenes.



Imagination Technologies | www.imgtec.com
embedded-computing.com/p372353

Open source license management solution

Protecode offers scalable, flexible open source license management solutions that scan for vulnerabilities and compliance of your open source code base. Reporting capabilities include software bill of materials (BOM) and obligations associated with the licenses and copyrights, encryption content, and overall make-up of your software portfolio. Protecode offers three product options – a complete open source license management system for real-time, periodic, pre-launch analysis as well as post-launch correction; a compact version that is a single-seat license for small organizations; and a cloud-based solution.

Protecode | www.protecode.com
embedded-computing.com/p372354

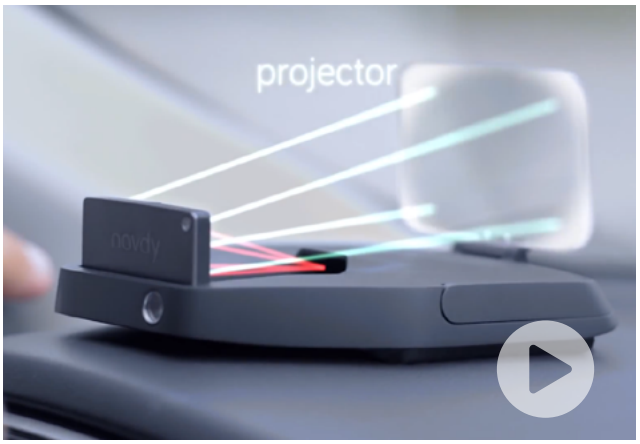
BLOG

THE CURRENT EDUCATIONAL SYSTEM ISN'T CUTTING IT

By Rich Nass, Embedded Computing Brand Director

I recently had the pleasure of being reacquainted with Jack Ganssle, who is clearly the King of Embedded. If you ever wanted to get an education of anything embedded-related, whether it's on the software side, the hardware side, or anything in between, Jack is your guy.

http://opsy.st/NassEducation



NEWS

IN-VEHICLE HUD BRINGS "GOOGLE GLASS" TO AUTO

By Brandon Lewis, Assistant Managing Editor

Pre-order campaign launches Navdy automotive head-up display (HUD) that interacts with smartphone apps to increase driver safety. Navdy combines a projection display and gesture and voice controls to allow drivers to intuitively make and receive phone calls, text message, and access apps safely within the driving experience.

http://opsy.st/ECDSept14HUD
http://opsy.st/ECDSept14Video

BLOG

THE SMARTEST CITY IN THE WORLD

By Angelo Corsaro, PrismTech

Nice, France has recently gained media attention thanks to a series of innovative projects aimed at preserving the environment and improving the quality of life through creative use of technology. In particular, Connected Boulevard target and enhanced city management capabilities.

http://opsy.st/ECDSept14PrismTech

NEWS

10 FINALISTS WILL RE-INVENT THE HAT IN ELEMENT14 DESIGN COMPETITION

By Premier Farnell Group

Participants use the Adafruit GEMMA and an accompanying accessory pack to make new wearable technology hats. The grand prize winner will receive an Oculus Rift from element14 and one complete FLORA Collection parts pack from Adafruit. The competition concludes October 31, 2014. See the list of finalists.

http://opsy.st/ECDSept14Wearable

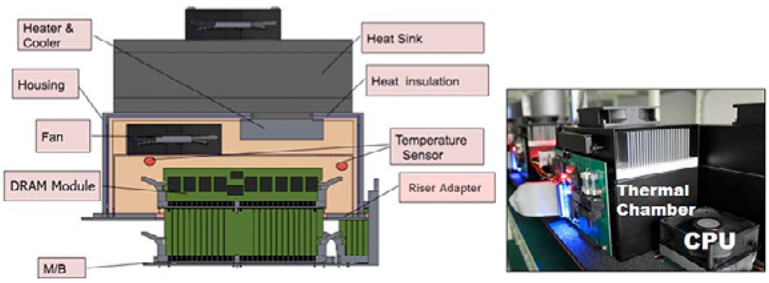
WHITE PAPER

UNDERSTANDING THE EFFECTS OF WIDE TEMPERATURES ON YOUR DRAM

By ATP Electronics

In several industrial applications, equipment is installed in a working environment with extreme temperatures. Commercial-grade DRAM can encounter problems outside their temperature ranges, but new testing technologies can help ensure they meet requirements.

http://opsy.st/ATPDRAMWP



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

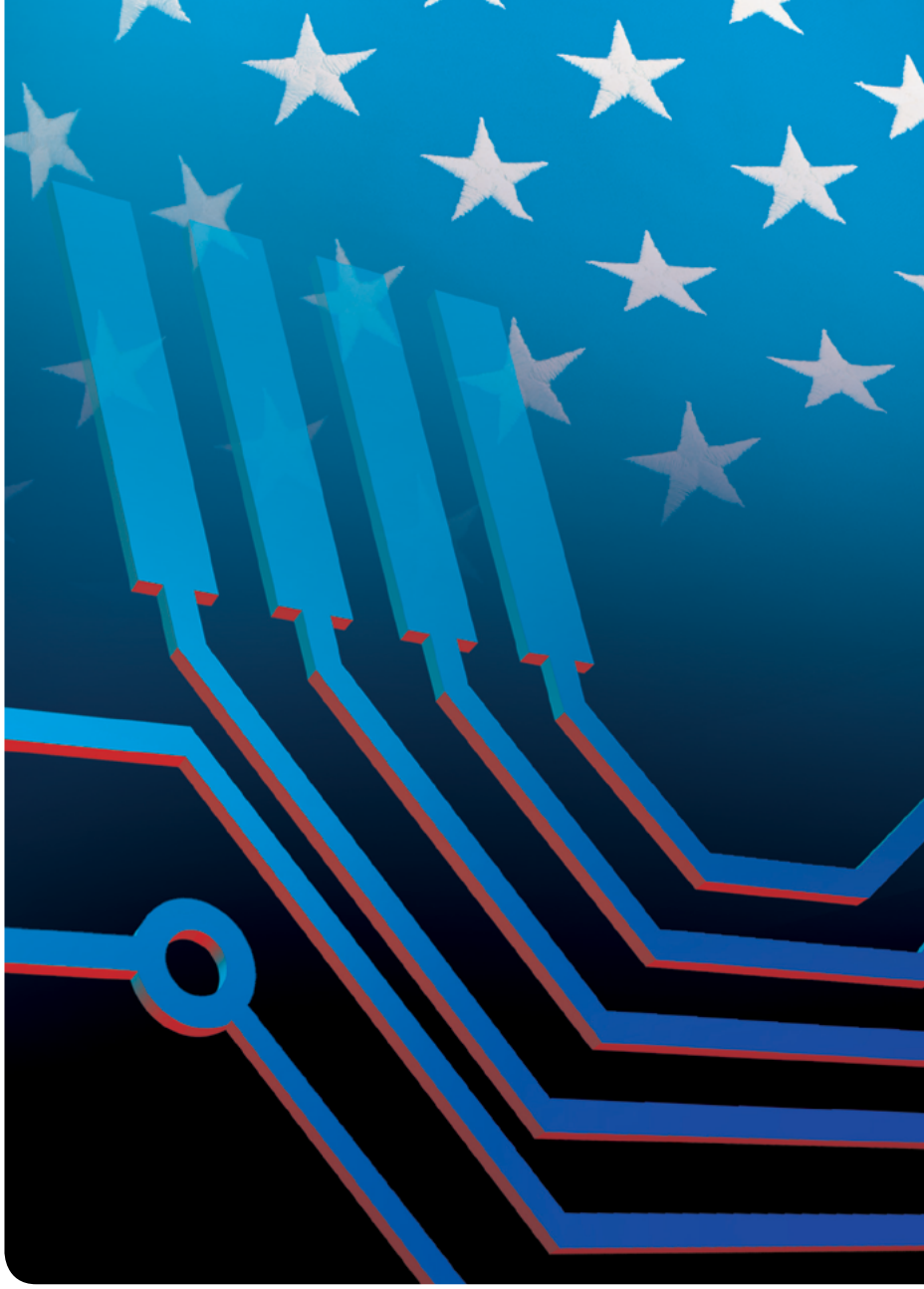
190 Admiral Cochrane Drive, Suite 130, Annapolis, Maryland USA 21401
wfinfo@annapmicro.com USA (410) 841-2514 www.annapmicro.com

Industrial DC/DC Power Solutions
Renewable Energy, UPS, and PoE
Fanless -40° to +85°C Operation

Small Form Factor Computers
Intel® Atom™ E3800 and i.MX6 CPUs
Fanless -40° to +85°C Operation

EPIC Single Board Computers
Rugged, Stackable Form Factor
Fanless -40° to +85°C Operation

Single Board Computers
COM Express Solutions
Power Supplies
I/O Modules
Panel PCs



Freedom From Support Delays and Long Lead-times

When the success of your project is on the line, you can't afford to waste time and money on poor technical support or products with long lead times. Call WinSystems and you'll speak directly with an Applications Engineer in our Arlington, Texas facility. Our Engineers are ready to guide you through product selection, customization, troubleshooting, and life-long support of our product. WinSystems keeps products in stock so we can ship out most orders within 1-2 business days.

Let us put over three decades of embedded computer experience, integrity, and service to work for you.

715 Stadium Drive | Arlington, Texas 76011
Phone: 817-274-7553 | Fax: 817-548-1358
info@winsystems.com

Call 817-274-7553 or visit www.winsystems.com.
Ask about our product evaluation!

WinSystems®
An Employee-Owned Company