Electronic Product

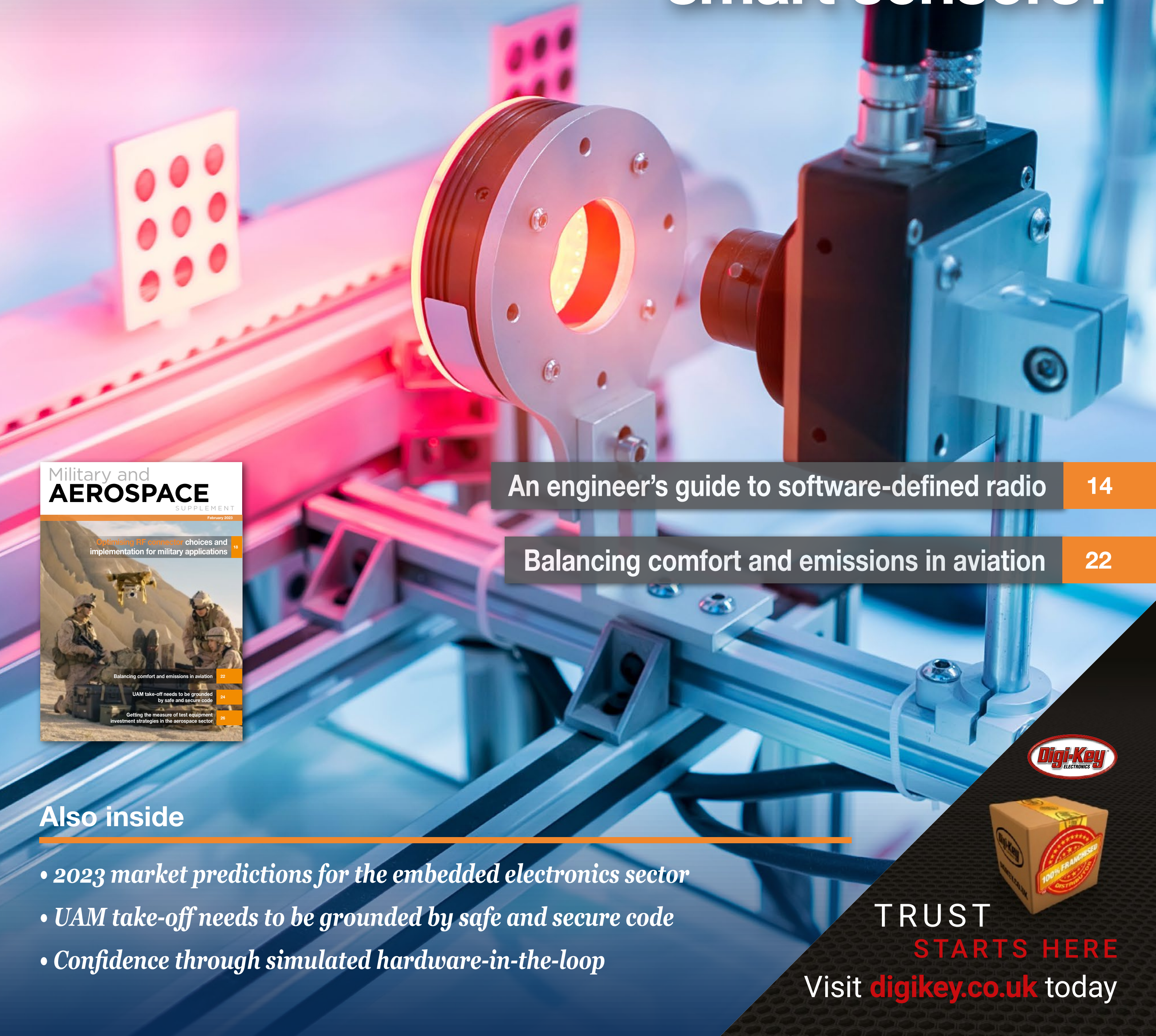
Design & Test

www.epdtonthenet.net

February 2023

ASICs, SoC or SiP

Which do you choose for smart sensors?



Military and
AEROSPACE
SUPPLEMENT
February 2023

Also inside

Digi-Key ELECTRONICS

TRUST
STARTS HERE
Visit digikey.co.uk today

Contents

www.epdtonthenet.net | February 2023

HAMMOND

Ask Hammond to customise any of our 5000+ standard stocked enclosures to your requirements

Learn more: hammfg.com/mods

uksales@hammfg.com • 01256 812812

Confidence through simulated hardware-in-the-loop

*The functional verification of a product's hardware and software working together, and how they interact with the product's intended operating environments, gives designers peace of mind that product reliability is high. Most if not all of that 'environment' can be created using benchtop instrumentation, as **Peter Wrigley, Managing Director of Bermondsey Electronics**, explains.*

OEMs serving global markets with commercial and/or industrial electronic products must compete on quality as opposed to cost if they are to ride the waves of fluctuating exchange rates. Also, just focusing on domestic markets is not particularly safe either, as global competition (from the Far East, for example), is always encroaching.

Quality is a measure of how well a product performs its functions. Reliability, robustness (physical and cyber) and the expected life of the product all contribute to quality.

To deliver on quality, OEMs of products such as IoT devices, are striving to meet the same kind of quality expectations as products intended for safety-critical applications.

Thorough verification is essential if the OEM is to have complete confidence in the product. Moreover, traceability - a record of the verification – is a must. That's how things are in the aerospace industry, for example, when verifying safety-critical avionics equipment. There, the product is connected to sensors and actuators on an 'iron bird' to prove that its internal hardware and software function as intended when interfacing with the external hardware within the bigger system. The practice is of course hardware-in-the-loop (HIL).

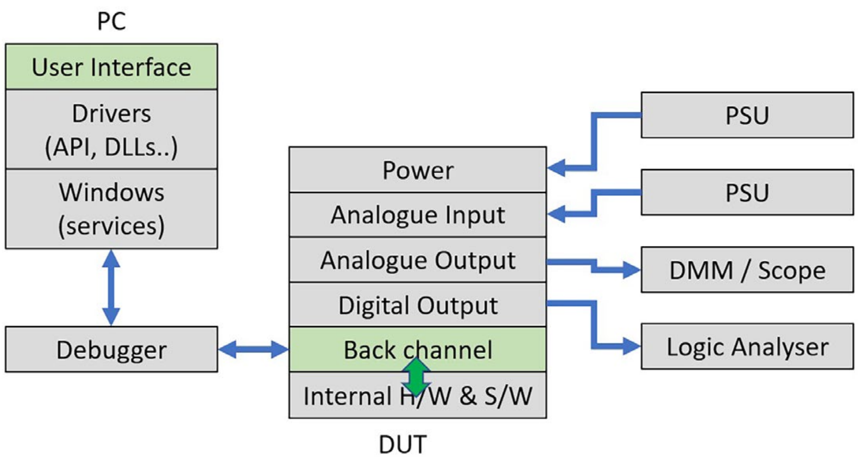


***Figure 1** – Simulating an environment for the DUT and monitoring software execution (and registers) via a debugger*

However, most mid-price commercial products are developed by SMEs. They don't have the resources of an OEM of high-value, low-volume products intended for safety-critical applications.

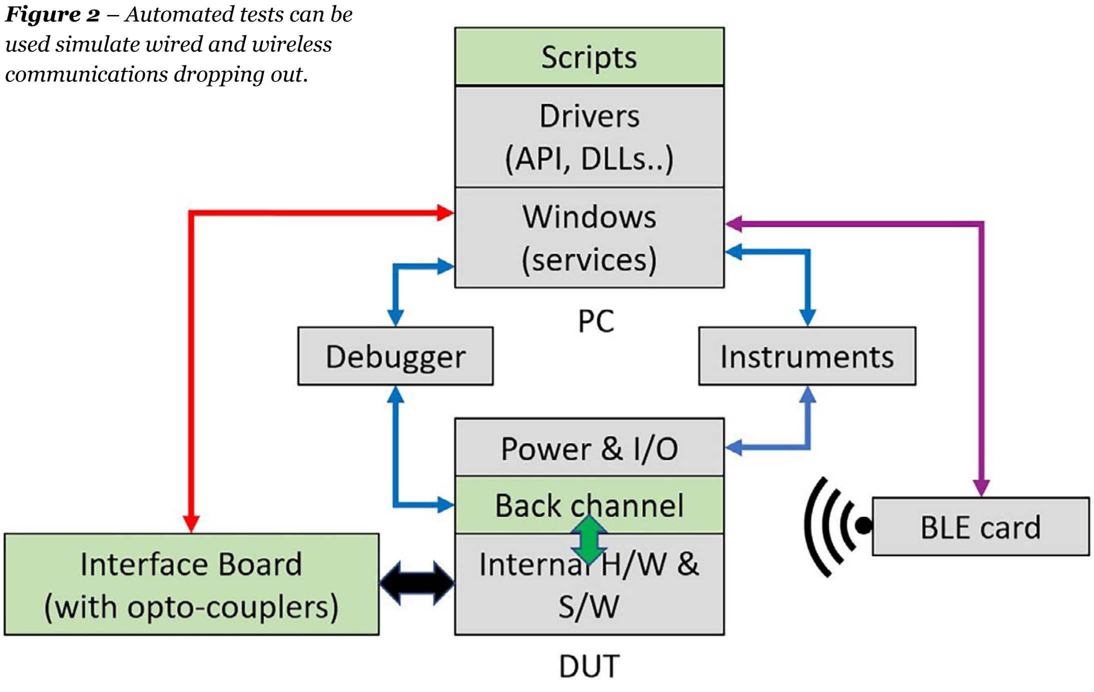
Sounds familiar?

If you're an embedded systems engineer working for an OEM of commercial or industrial products, you are likely to be in an engineering team of five to 10 people, and your product development will be an iterative process, one that is usually compressed to meet time- and volume-to-market pressures. You probably have to accommodate last-minute changes to requirements too.

Software developed on a PC will be verified using software-in-the-loop (simulations, testbenches etc.). Then, a few prototype PCBs will be made. These will be tested as thoroughly as possible, as it will be the first-time software meets

If you're an embedded systems engineer, your product development will be an iterative process, one that is usually compressed to meet time- and volume-to-market pressures

Figure 2 *– Automated tests can be used simulate wired and wireless communications dropping out.*



real hardware - i.e., the microcontroller, onboard memory, ADCs and DACs. There will undoubtedly be some real-world issues, such as latency, to fix and as much as possible will be addressed through software changes. Some fixes may require hardware changes, though, and a second batch of prototype boards will be made. Again, the hardware/software integration will be tested as thoroughly as possible.

But just how thorough is thorough? The fact of the matter is most SMEs of embedded systems for commercial and industrial applications do minimal HIL testing.

For example, consider an IoT device that is designed to interface with a wireless carrier, such as EE or Vodafone in the UK, as well as other carriers around the world. If the device is prototyped in the UK, then verification on the domestic carriers is easy. Verification against other carriers is harder.

Unless the DUT can be tested under all intended operating environments, there's just no way of knowing if there are any corner cases that will result in errors. And on the subject of errors, how do you verify that the DUT handles those correctly?

You have most of what you need

The creation of most of your DUT's operating environment can be simulated using benchtop instrumentation. For example, a benchtop PSU can be used to verify that your DUT handles under- and over-voltage scenarios. It can also be used to exercise analogue inputs too. And waveform/signal generators can be used to provide more complex inputs.

Also, by accessing its microcontroller via a debugger it is possible to interrogate the DUT to establish the voltage levels and inputs it thinks it is seeing. Your DUT's outputs can be monitored using a digital multi-meter or oscilloscope. Figure 1 shows such a combination of connections.

However, there are four problems with the scenario depicted in figure 1.

1. Operator variations. If the analogue input needs time to settle, then one person changing the voltage on the PSU might result in DUT behaviour that is different from when someone else changes the voltage (at a faster or slower rate).
2. Human error. As mentioned, traceability is needed, and that means data collection. DMM and scope readings can be misinterpreted or noted incorrectly in a spreadsheet.
3. The scenario in figure 1 is not a closed loop. Something happening on an output cannot automatically influence an input.

Also, for the collected data to have credibility, all tests must be highly repeatable and accurate. For example, there must be no doubt over how your DUT's inputs were driven. And that's where automation comes in.

Automation using scripts is the only way to standardize the execution of tests and automatically record results, thus solving problems 1 and 2. If your instrumentation is networkable, then you are in luck.

Drivers are freely available from most test and measurement equipment (TME) manufacturers, and you can write scripts to drive power supplies and collect data from instruments. Indeed, a complete closed HIL environment can probably be simulated using the instrumentation in your engineering department - thus solving problem 3.

However, your instruments are unlikely to all be from the same manufacturer, and the drivers will probably be for different languages. For example, Keysight's OpenTAP favours C# and Nordic's BLE driver is in Python. Your team likely codes in C, so, you may need to learn a variety of different programming languages, to stitch everything together – and that could take several months of billable engineering hours. It could cost the equivalent of £100,000 for some companies.

An alternative, saving a great deal time and money, is to be independent of

Unless the DUT can be tested under all intended operating environments, there's just no way of knowing if there are any corner cases that will result in errors.

Your instruments are unlikely to all be from the same manufacturer, and the drivers will probably be for different languages.

the programming languages of your instruments, and settle on a language that is close to the one you already know. In the case of BELIeVE (see box) that language is JavaScript. It is closer to C than either C# or Python, so your MCU programmer can write the tests.

Thoroughly tested

As mentioned, the inclusion of a networkable debugger makes possible three very important things:

- Mocking. It temporarily modifies the system's code to force certain behaviour, typically to make the DUT interpret inputs in a different way. For example, if the DUT is a battery management system (BMS) for an electric vehicle, it may need to monitor a DC voltage in the range 0 to 800V. Verification of that feature would normally require a very special (and expensive) PSU. However, if you place mocking software on the DUT that tells it to treat every 1VDC seen on the input as if it were 100VDC, a standard benchtop PSU can be used. It can easily be made to sweep between 0 to 8V.
- Event advancement. Many products are designed to enter a certain mode or perform a function after a set number of operations, as part of a maintenance strategy that is based on use rather than time in service, for example. However, if the number of operations is large (say 1,000) and the operations have an average cycle time of 2 hours, you won't want to wait 2,000 hours just to confirm the DUT does what it is meant to.
- Fault injection. This ideal for verifying that the DUT recognises faults and handles them correctly; goes into a fail-safe mode, for example.

As well as being able to inject fault conditions into the DUT's software, it is also possible to replicate conditions the end product is likely to experience. For instance, if your product is to have a wireless interface, such as Bluetooth Low Energy (BLE) or WiFi, it is important to verify what happens if the signal drops out. Similarly, how does the DUT react to a USB or ethernet cable being pulled?

Thankfully, these conditions can all be simulated. It is also possible to simulate user interactions, such as button pushes. Figure 2 illustrates the connectivity required for these tests.

Conclusion

As mentioned, those working in aerospace and other sectors where safety-critical equipment is used must go down the HIL route for verification purposes. It verifies the integration of the DUT's internal hardware and software, and that both interact correctly with external hardware (such as sensors) and communicate with other sub-systems. For non-safety-critical applications there is no need to go down such an expensive route.

Everything HIL sets out to achieve can be simulated. In fact, I believe more can be verified with the set up shown in figure 2, because we have the ability to advance events and inject faults.

BELIeVE

BELIeVE is a PC-based (Windows 10 or higher) application that can automate the operation of a wide variety of networkable instrumentation, such as the PSU and DMM shown above, to create an entire simulated HIL test environment. Because it is automated (including report generation) tests can not only run faster than if they were performed manually but they can also run during work breaks, overnight and over the weekend. You can download a trial (but fully functional) version of BELIeVE from https://www.bermondseyelectronics.com/believe/