Cooking up a **connected** product

Adding a connected device to your portfolio can be challenging. **Jonathan Pallant** looks at how to pick the 'perfect ingredients'

Connected devices tend to sit at the more complex end of the embedded system spectrum, irrespective of whether they gather sensor data by themselves or interface with pre-existing equipment.

Their complexity arises from the interaction between their three constituent parts: the local part, which generates the data or needs some control; the far system which records the data and/or provides instructions; and the connection between them, which typically operates over a large distance and is often observable (and even interruptible) by untrustworthy third parties.

These complexities are, however, often outweighed by the improved customer engagement, detailed business insights, and even entirely new commercial models that can be achieved.

The fine balance between complexity and function means that building the right kind of connected device is often part art, part science. In fact, it's a lot like cooking a great meal – there are practically an infinite number of ways to bring key ingredients and dishes together. Some of which can be assembled quite quickly, whereas others will take more time and effort to perfect, but which may be better suited to someone's tastes.

One approach is to break the IoT development process into several specific but fundamentally connected choices. But it isn't always easy to work out which decisions to take first, and there is always a risk that an early decision could restrict your later choices and lead to the whole system being sub-optimal.

We've alliteratively grouped these choices into six different areas to consider when planning the perfect IoT 'feast'.

Connectivity is probably the best place to start, as it's the fundamental component that the rest of the offering hangs around. The main course, if you will.

The Connectivity Course

The key connectivity technology requirements for products are defined by cost, power consumption, range, data rate, and latency. But they are ultimately tied together by the Laws of Physics and Economics. And all of today's technologies exist in their own particular sweet-spot within this multi-dimensional space.

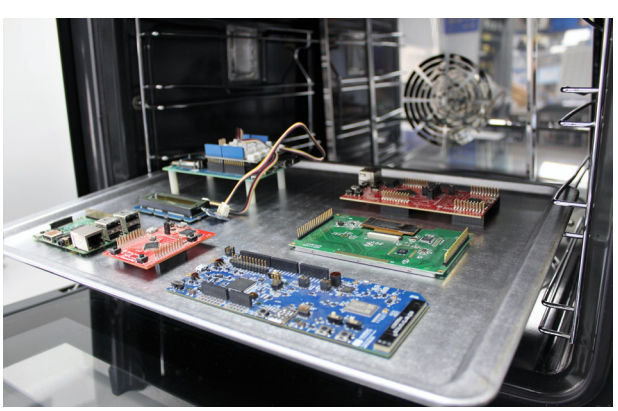
Most systems will probably need wireless connectivity to access the Internet, which sadly is typically less reliable and higher cost than using a wired connection. Almost all wireless standards involve at least two classes of device – a smaller, lower-power component (like a mobile phone) and a larger, higher-power gateway (like a Wi-Fi router). You also need to consider the ownership and reliability of those gateways, comparing for example a homeowner's Wi-Fi router versus accessing a mobile network.

Whichever way you go, it is almost always better to rely on existing connectivity standards with broad market acceptance rather than developing something from scratch.

The Chipset Course

The first decision to make is whether to opt for a chipset that integrates processor, memory and radio, or whether to combine several ingredients to get the right mix of performance, price and power consumption.

If you need low-power cellular



Below: There are few absolutes in technology. There will always be a range of potential solutions available

connectivity, either with LTE-M or NB-IoT, a device like the Nordic nRF9160 is a compelling package of CPU, RAM, Flash, GPS and Modem. But, if you can afford the PCB space, you might prefer the flexibility of a "standard" microcontroller and a separate LTE module. A similar story applies to wireless standards like Wi-Fi and Bluetooth, where there are microcontrollers available with the radio built-in, and also stand-alone modem modules.

Of course, IoT devices don't just run on basic microcontrollers. There are plenty of examples where the power and technical ability of a full Linux Kernel-based system was the right choice, even with the knock-on effects on power consumption, space and software support burden.

The Core Course

This stage involves bringing in pre-existing software (essentially "third-party IP") including: the Operating System (or Real-Time Operating System) kernel; the chipset drivers; and the board support package (BSP) which customises those drivers for your particular PCB design.

Chipset vendors often provide a free Software Development Kit (SDK) containing all of this. But it is worth remembering that this SDK may not work with anyone else's silicon, limiting your ability to change supplier in the future. If that matters, perhaps consider a vendor-neutral offering such as Amazon FreeRTOS or Microsoft's ThreadX.

Linux-based systems have similar options too – often with a free "distribution" to get you started. But you may prefer to roll-your-own or to use a more generic off-the-shelf Linux distribution, such as Ubuntu Smart Start or Fedora IoT. It all depends on your security requirements, development timescales and any system limitations on power consumption and storage space.

The Code Course

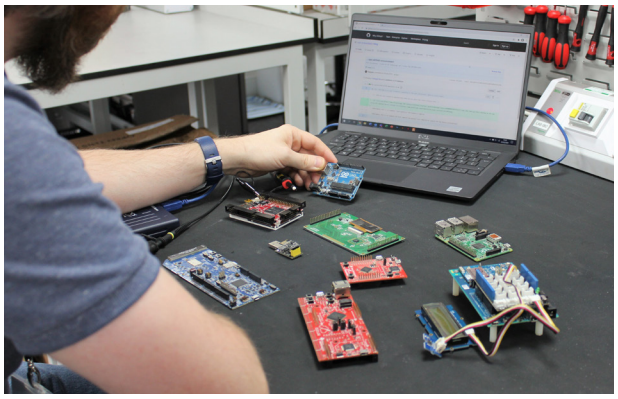
Of course, you can't just install embedded Linux on your device, or flash on an RTOS, and call it a day. You'll need to add the "secret sauce" by writing code to customise this system to your needs. That could be as simple as specifying the endpoints and keys to upload sensor readings, or as complex as running multiple bespoke protocol stacks and sophisticated AI processing at the edge for good measure. Either way, the language and tools you use will have a huge impact on your up-front development programme, as well as for on-going support and maintenance.

The C programming language has been the default for almost 40 years now, but maybe there's value in trying a memory-safe systems language, like Rust or even something like MicroPython. But if you find your Chipset or Core can't handle your preferred choice here, you may need to go back and re-think.

The Cloud Course

Most connected device developments involve the gathering of data for later analysis, and the issuing of commands to control devices "in the field". It makes sense almost universally to offload this computing requirement to one of the pre-built IoT management and data storage offerings available from the big "cloud" providers, rather than using on-premises systems.

Author details:
Jonathan Pallant is senior embedded systems consultant at 42 Technology



Below: Deciding what to prioritise is not always easy when developing new connected products

Whichever cloud platform you choose though, you'll need to balance the up-front costs of integration against on-going support and maintenance. And remember, you're going to need support from your cloud, chipset and connectivity providers for the entire lifetime of your product in the field.

And finally…the Comms Course

The default Communications protocol your Code will use to talk to your Cloud provider will usually be text-based (such as JSON or XML) running over an encrypted connection-oriented HTTP link.

Whilst this sort of stack is ubiquitous (it's how most current websites are built), its plain-text nature increases both bandwidth requirements on your connectivity and chipset resources, and the connection-oriented nature can interact badly with some high-latency wireless services. For example, if you are designing an ultra-low power monitoring system using NB-IoT, you might be better with a binary connection-less protocol like CoAP.

Also, being able to rely on a well-used, real-world-tested protocol stack (such as the one that comes with your Core software) is almost certainly going to be better than trying to spin one yourself.

It's worth remembering

There are few absolutes in technology and there will always be a range of solutions to meet your requirements. What works best will depend on your budgets, timescales and the skills you have available in-house, from an external development partner or through any existing relationships with hardware, software or network providers. But, even the best technical solution will never become a winner unless it gets to market on time and with the right combination of price, specification, and availability too.