



The U in USB currently stands for 'universal', but the rate at which USB is being integrated in to all manner of devices could justify it being replaced by 'ubiquitous'.

Its convenience is undeniable; 'plug and play' is an expression that is often used, but rarely delivered. Not so for USB, and its convenience is matched by its reliability.

Yet its convenience belies its complexity. The term 'digital' is now synonymous with 'reliable' but, as any engineer will appreciate, digital is considered reliable largely because it's predictable. But 'reliable' isn't synonymous with 'quality'.

The reason why USB is predictable, reliable, and largely of a high quality isn't because it's simple; on the contrary, it is surprisingly complex, particularly when considered against old stalwarts such as RS232, parallel and uart interfaces.

Clearly, complexity is the price paid for USB's predictable, reliable quality; there's always a trade off. Such is its complexity compared to other interface technologies that the market now offers a range of emulation devices, designed to convert USB traffic into, for instance, RS232, or to offer

What began life as a convenient interface for pc peripherals could soon be truly ubiquitous.

By **Philip Ling**.

a uart interface that gives the impression of a USB interface in legacy equipment.

The benefit here is that the USB protocol is often handled by the silicon, allowing the application to treat the interface as an existing uart or fifo, but providing the outside world with what looks like a USB port. The downside is that the driver required by the host to interface to the device isn't necessarily standard and so may not be installed. This means, of course, that it needs to be supplied by the manufacturer and, in the event that the host pc doesn't recognise the device natively, installed by the end user.

This may be acceptable for industrial systems, where the connectivity is perhaps between a data logger and a particular pc, or 'closed' systems where the device is used as a licensing dongle for a specific applica-

tion running on the host. But for standard peripherals (consumer devices, for instance) where such a union doesn't exist, this becomes untenable.

The proliferation of emulation devices could have the effect of further masking the real complexity behind USB. We're all familiar with USB I/O devices, such as a mouse and keyboard, or printers and scanners, being recognised instantly by the pc when attached.

These are tasks that have, in the past, been achieved using the original parallel and serial interfaces provided by pc manufacturers; interfaces that are virtually transparent when compared to USB. For almost as long as the pc has been around, engineers have shown extensive resourcefulness in turning these standard and transparent interfaces into portals for bespoke equipment. Coupled with a working knowledge of DOS and, more recently, Visual C, any pc can be turned into special to type test equipment, a data logger or industrial controller, by connecting to the equipment via the parallel port and reading/writing to/from the port's address.

Now that USB has taken over and

Clarity from complexity





expanded the world of peripherals, these original interfaces are becoming less available or have simply fallen out of fashion. The result is that more and more applications are turning towards the ubiquitous serial bus for connectivity.

While the various emulation devices offer a great deal of inherent USB functionality, hardwired into silicon, moving towards a fully fledged USB device impresses greater demands on the designer.

Typically, because general USB connectivity isn't controlled by a relatively simple state machine, it employs a microcontroller to manage its various layers and it is here that the intelligence shifts away from the hardware and more towards the software; residing in the firmware.

In order to be recognised by a standard driver, the USB device needs to fall into one of the predefined and approved classes, these include the well known Human Interface Device (HID), Communication Device Class (CDC) and Mass Storage classes, as well as less common Cable and Connector, Common Class and Content Security Class.

The real point here is that there is a framework within which to work, but it doesn't stop at the class. The device then needs to comply with a transfer type, so the host knows

what to expect from it and when. While this is relatively simple in a proprietary system using, say, RS232, it becomes a little more involved with USB.

There are four transfer types on offer, depending on whether the data being transferred is large or small, and time sensitive or not; specifically they are: control; interrupt; isochronous; and bulk. On top of this, there are three different transfer speeds to comply with, depending on the version of USB being implemented (low, high and full speed).

The topology of the connection is also quite different from a peer to peer connection typified by RS232. For example, being able to enumerate a USB device is inherent in the host controller but needs to be accommodated in the device itself, further complicating the firmware's tasks.

In an attempt to address what could stand in the way of growing demand for designing in USB peripheral capability, STMicroelectronics recently announced the availability of a free and complete USB firmware development kit for its ARM based STR7 and STR9 microcontrollers. According to ST, it promises painless development of firmware for all USB layers and transfer types.

Both of these microcontroller families include variants with USB ports, as do microcontrollers from other vendors. Surprisingly, software support for these ports isn't always included, so ST's move – if successful – is likely to be replicated by other leading IDMs in the near future.

The kit includes a library of drivers and

demonstrations for each transfer type. An HID driver provides an example of fast response (interrupt) transfer, as used by a mouse or keyboard device; bulk transfers are illustrated with a mass storage application, whilst isochronous transfer is supported with a voice/speaker/microphone demonstration.

Having a reference design in the form of sample source code will be invaluable for some developers, but it's worth noting that developing a USB compliant device is only half the story; in order to make it generally available, the device manufacturer will need to apply to the USB-IF for a unique Vendor ID and Product ID, data used to identify the vendor and device to the operating system. For small manufacturers or for devices being manufactured in small volumes, this could prove prohibitive.

Some IDMs are working around this requirement by extending their Vendor ID to their customers and assigning their own Product ID in the silicon. This is feasible where the vendor's driver is already embedded in the operating system, for instance.

As USB further extends its presence beyond the desktop, efforts are being made to provide host controller functions in embedded devices too, as well as the continued development of USB On The Go and Wireless USB. One thing is for sure, the ubiquity of USB is set to become truly universal. 🌀

Author profile:

Philip Ling is a freelance technical writer.