



CHRISTOPHER SMITH

Christopher Smith explains how to capture real-time bugs when they occur

# DELIVERING ON DEBUG

Twenty years ago as a young engineer I witnessed a revolution in electronic engineering that brought unprecedented power but also great challenges to the electronic design community. The competition to produce ever more capable CPUs for office computers kicked off in earnest in 1984, as the newly introduced IBM personal computer squared up to Apple's imaginative Macintosh. The shockwave of the resulting processor power struggle reached embedded developers in the form of an unrelenting wave on which engineers are still riding to this day. Need more performance? By the time you are in production it will be here.

But harnessing that performance was always another matter. Increasing complexity, bus width and CPU speed all made software debugging more difficult. ROM monitors provided very basic assistance. Proprietary development systems from the chip manufacturers offered more – but at a considerable cost – and eventually spawned the in-circuit emulator (ICE), a device that allowed the user to 'view' what was happening inside the processor as it executed instructions in real time. When other equipment makers entered the market, producing ICEs that could be used with standard PCs, costs fell and it became feasible to equip almost every workbench with an ICE.

The 32bit and 64bit processor wars, therefore, have also provided the catalyst for the evolution of embedded development tools. The dominant trend has been to move significant debug capabilities closer to – and ultimately on-board – the target MCU.

But rapidly increasing processor speeds have also overtaken the capabilities of the traditional ICE. Transmission line effects between the processor socket and the ICE itself ensured the signal timing errors alone make debugging impossible. After all, the cable from the target board to the ICE can only be shortened by so much before it becomes impractical.

Another limitation of the ICE is that engineers have always wanted to debug their application program under conditions that imitate the actual setup of their system. The ICE requires the processor to be removed, or at least replaced with a bond-out chip. But with increasing processing speeds, modern complex SoCs and larger user applications, this limitation is an even greater hindrance today than in the past. Today, in-circuit debugging, the capability to debug a user program in an actual target system, is a basic necessity.

The arrival of on-chip debug modes (e.g. BDM and JTAG) provided the next significant step forward. The silicon ven-

dors themselves made this possible by implementing debug hardware on the target MCU, bringing the debug data out to external pins that could be routed to a pin header on the PCB. Some vendors standardised the on-chip debug interface across all MCUs and also defined a standard hardware connector to connect the target system to the host PC via an interface pod.

Since on-chip debug is supported by hardware integrated on the silicon, rather than embedded firmware running a monitor ROM, for instance, it does not consume CPU resources like ROM, RAM and interrupt handling. But even though on-chip debug allows the designer to perform fundamental debug activities such as single-stepping code at the source or assembly level, or manipulating CPU and peripheral registers, these are fairly low level capabilities; particularly in the context of today's most complex systems. And although many debug commands are available a significant proportion can only be used when the MCU is not running a user program.

Despite these limitations, on-chip debug has demonstrated that, as hardware debug facilities evolve to provide more functionality closer to the CPU registers, the possibility exists to debug systems running higher processor speeds. But consider this further set of constants: the most difficult bugs happen infrequently, and usually only when executing in real-time; they rarely show up when stepping through the code; they show no obvious symptoms until failure and have effects that seem unrelated to the cause. And they frequently fail in catastrophic ways that cover their tracks.

For these reasons, it is preferable to be able to trace program execution from the beginning. Recently, on-chip debug hardware has taken another step toward this ideal by providing a trace port on-chip, which makes it possible to connect a high speed debug device capable of tracing program execution. Some of this silicon-based hardware only tracks code execution, but there are more sophisticated implementations that capture full address, data and processor specific information rather like the old ICE.

By collecting the data presented at the trace port, a new generation of powerful debuggers will allow engineers to capture and analyse the system behaviour as if it were a live system. The first processors to implement a trace port include MIPS EJTAG 2.0, the PowerPC405 and PowerPC440 and the Nexus IEEE-ISTO 5001 Class 3 MCUs, such as the MPC5554. The ARM Embedded Trace Macrocell (ETM) also makes trace capability available for ARM-based SoCs. In the

future, even more embedded MCUs and embedded processor IP will become available with trace port capability.

Some trace implementations use a very small on-chip buffer of maybe around 16 KB for storage of the trace information and use regular on-chip debug commands to retrieve it without the need of additional pins. But directly routing the trace data off-chip, through the trace port pins to a larger memory - at least 1-2MB - unleashes more of the power of trace. So now we are back where we started: we need advanced hardware debug probes to maximise the power of trace debug.

But this is not quite back to the future. In fact, we are talking about is a mutation; a super probe packing hundreds of megabytes or more of memory and capable of logging address, data and processor specific data in real time to allow reconstruction of the complete real time behaviour of a system over an entire period. A device such as the Green Hills SuperTrace Probe is capable of supporting port speeds over 300MHz and has 1Gb trace memory capable of storing a continuous execution history of some two billion cycles.

Moreover, new and sophisticated debug tools are also emerging to deliver a new debug model that allows the cap-

tured trace to be used for everyday application debugging as well as long-term statistical analysis and locating bugs that show no symptoms or happen only once. Using tools such as the Green Hills TimeMachine debugger, in conjunction with SuperTrace Probe and the on-chip trace port, engineers can now “travel back in time” to examine real-time bugs when they occurred. They can, for example, install a hardware breakpoint on the address of a variable and run the program backward to the source of a memory corruption problem (instead of trying to reproduce the problem in a next run), and manipulate the system in many other powerful ways.

So what has really changed is that by leveraging these new silicon features we can now begin to deliver the capabilities that have existed on every developer’s “Wouldn’t it be nice if...” list from the very beginning. All of which is aided by readily available high performance desktop workstations, the reduced cost of adding trace modules to the silicon and the emergence of large capacity hardware trace probes. **GDG**

*Christopher Smith is vice president of marketing , Green Hills Software*