By Mark Honman, Sundance Multiprocessor Technology, Ltd.

www.sundance.com mark.h@sundance.com

If wishes were fishes – An accidental computer scientist goes embedded

Can't we just TALK?

There is a well-worn cliché in computing, "Standards are great – there are so many to choose from!" In computing this is perhaps less true than it used to be, thanks to the Internet and commoditization of end-user computing.

However, in the world of embedded systems there has not been the same kind of convergence in interface standards. While the DSP and FPGA vendor platforms have become evermore powerful and applicable to a broader range of applications, there seems to be little attention given to combining different types of processing elements in a single hardware module.

Before I go further, is it even still relevant to discuss the topic of integrating DSP and FPGA devices in the same hardware? After all, DSP vendors are now offering data acquisition interfaces that are supposed to make FPGAs obsolete, and FPGA vendors are offering software compilation to RTL that is supposed to make DSPs obsolete. Not to mention that everyone's space is being invaded by ARM and Linux-wielding refugees like myself from the "real computing" domain.

Perhaps the best reasons are productivity and time-to-market. After two decades of refinement, DSP architectures and surrounding tools and support libraries are wonderfully mature, and FPGA tools are similarly all-encompassing. The problem, as I see it, is that the overlap between DSP and FPGA capabilities only goes so far. There will continue to be projects that require processing for which no DSP has a suitable accelerator or interface, and others where there is a large computationally intensive code base.

Since it is easy to become so familiar with one vendor's platform that everything else appears alien, there is a trap waiting for all of us: "When all you have is a hammer, everything looks like a nail." We can generally sense when it is becoming more difficult – or risky – to implement functionality within the platform chosen for the current project. The bad news is that while adding a DSP or FPGA may save time by reducing the number of workarounds, compromises in the interface often mean that the cure is worse than the disease.

The success of the ARM-enhanced DSPs and FPGAs is with good reason – so much time can be saved by offloading non-critical functionality to ARM cores running Linux. And, at least where this code does not touch vendor-specific communication libraries, it is possible to reuse the software in a subsequent project based on another vendor's processor.

So, wouldn't our lives be simpler – and more interesting – if there was a simple, standardized, low-latency interface that made it possible to harness the best offerings of multiple vendors?

... WOULDN'T OUR LIVES BE SIMPLER – AND MORE INTERESTING – IF THERE WAS A SIMPLE, STANDARDIZED, LOW-LATENCY INTERFACE THAT MADE IT POSSIBLE TO HARNESS THE BEST OFFERINGS OF MULTIPLE VENDORS?

Now, it could be that that there is a solution that already fits the bill, but for the moment let's assume that is not the case. Other than low latency, there are two characteristics that are essential for a successful interface: simplicity and decoupling.

Without a powerful customer insisting that there is a common standard, it only makes sense to work together if using the standard saves us time and money compared to the effort we put into its development. A simple standard keeps the effort low and makes it relatively easy to adapt some new platform or bridge to a proprietary on-chip communication system.

Decoupling is a gift to the developers of the software or firmware on either side of the interface, allowing development of a communicating module that focuses on what it is supposed to do rather than being shaped by the quirks of its communication partners. When processes communicate there will always be some form of coupling, but much pain can be avoided by making these interactions predictable and consistent in their behavior. By making it easier to reason about process state, debugging is simpler and, even better, less likely to be needed.

At this point I am half hoping that someone will step forward and say "this has already been done," for surely after 25 years of embedded parallel processing this is a problem that has been solved many times over? If that is you, please write and put me out of my misery!

Next I will consider some of the options that could be used in the world of real project timescales and budgets, though from time to time you will also find me dreaming ... SPD