

Got OCP? The Role of the OCP in Multicore Designs

by Clive "Max" Maxfield

When I began my career way back in the mists of time things were so much simpler than they are today. As a simple example, microprocessors were understandable by almost anyone who cared to experiment with them. Ah, the joy of playing with an 8-bit data bus, a 16-bit address bus, and a handful of general-purpose registers.

By comparison, today's high-end System-on-Chip (SoC) designs boggle the mind and defy description. Comprising tens or hundreds of millions of logic gates, these beasts typically feature multiple processor cores performing different tasks, tiered memory structures with multi-level memory caching, and multi-layer bus structures (and don't get me talking about super-pipelining and super-scaling).

The problem is that the vast majority of these bits-and-pieces – like the processor cores and other functions – come in the form of third-party IP blocks. The process of building an SoC involves gathering processor, peripheral, and memory cores from a wide variety of sources. Furthermore, there is a temporal aspect, because these cores may have originated at different times.

The function of a core dictates its interface to the outside world (in the form of the rest of the chip). There's no way we can expect that everyone who created cores at different times in different places for different purposes all made the same decisions. Thus, the designers of a modern SoC are in the position of constructing the chip out of functional units that they did not themselves devise, and that they almost certainly do not understand at some level. (If designers did need to fully understand the internals of each and every core, they might as well design them all from the ground up, which would result in a new chip reaching the market once every hundred years or so.)

Obviously, some way to address these multicore problems is required. There is a solution in the form of the OCP. In order to wrap our brains around this we need to ask (and answer) three fundamental questions:

- What is a multicore design?
- What is a socket?
- What is the OCP?

What is a multicore design?

Let's take this step-by-step, starting with the concept of multiple processors. In reality, we've been using multi-processor systems in "Computer Space" for a long time. Consider a simple Personal Computer (PC) circa 1995, for example. In addition to the main microprocessor on the motherboard, there would also be a smaller processor in the hard disk drive, and even a tiny processor in the keyboard.

In fact, even the simplest of today's PCs tend to have processors scattered around like confetti. However, this type of multi-processor implementation is relatively simple, because all of the processors are independent. The main, general-purpose processor handles the bulk of the work, while the smaller, special-purpose processors handle a limited number of specialized tasks. Furthermore, each processor is the "master of its own domain" in the sense that it has absolute control over its own resources, such as its local memory sub-system.

Another multi-processor scenario involves a number of identical processors sharing common resources; for example, a group of processors sharing the same memory space. Consider the "Big Iron" machines of the 1970s, such as the IBM mainframes and Crays, for example.

All of which leads us to the concept of "multicore," which refers to multiple processor, peripheral, and memory cores on the same silicon chip. One example of this would be the multicore processors from Intel

and AMD, each containing two, four, or more identical (homogeneous) processor cores sharing common resources.

But off-the-shelf microprocessors are not what we're interested in here. For the purposes of this article, we are pondering the creation of high-end SoC devices. In this case, while it is true that homogeneous implementations involving multiple copies of the same core are more general-purpose and more flexible, heterogeneous realizations comprising a variety of specialized cores assigned to specific tasks are more efficient.

And, in reality, we can have a mixture of homogeneous and heterogeneous cores on the same chip. The ARM Cortex-A9, for example, is available with two, three or four homogeneous cores (designers can lay down multiple such clusters on the chip if they so-desire). These cores can then be augmented with specialized cores from other vendors.

The end result is like having a supercomputer on a chip, with incredibly complex combinations of processors scattered across the die. The challenge is how to deal with the heterogeneous nature of all of this...

What is a socket?

The term "socket" originated in the context of network systems, where a socket defines the boundary of responsibilities between two disparate elements in the system. More specifically, a socket completely specifies the boundary electrically, logically, and at the protocol level, and it fully defines communications behavior with the outside world.

Now, consider the case where one is architecting an SoC using multiple cores from disparate sources. Let's take a really simple example showing only four cores communicating via a common bus as illustrated in Figure 1 (we'll return to ponder things like memory subsystems in a little while).

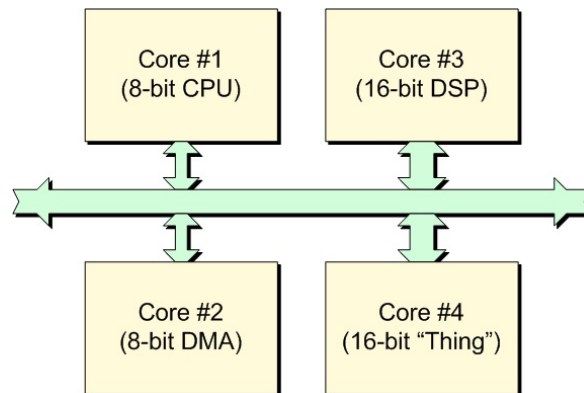


Figure 1. A very simple multicore example.

Purely for the sake of this discussion, let's assume that Core #1 is a legacy processor with an 8-bit data bus and featuring a certain set of control signals. Similarly, let's assume that Core #2 is a legacy DMA engine with an 8-bit data bus. Meanwhile, Core #3 is a modern 16-bit DSP engine.

Now, suppose that we want the Core #1 CPU to use the Core #2 DMA, but that they have incompatible control signals. Furthermore, let's assume that this system uses DRAM, but our 8-bit CPU and DMA cores were created before anyone had to worry about DRAM bursts, and... you can come up with myriad such examples.

Apart from the obvious complexities, the situation is further confused by the fact that – as we previously discussed – the designers of a modern SoC are in the position of constructing the chip out of functional units that they did not themselves devise, and that they almost certainly do not understand at some level.

The solution is to isolate the various cores with sockets, which take the cores' internal control and data signals and interface them into a common domain as illustrated in Figure 2.

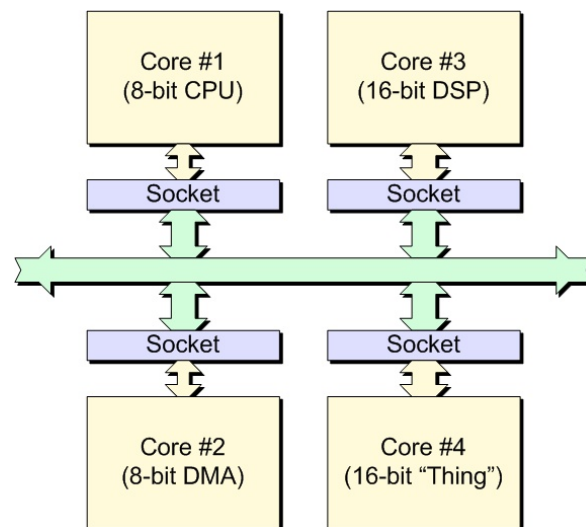


Figure 2. Using sockets to interface the cores to the rest of the system.

Like many things, this appears obvious if you say it quickly and wave your hands around a lot, but this isn't as simple as it seems. The problem with a heterogeneous system is that, if we're not careful, we're going to end up with a completely different socket for each core, which would bring us back to square one...

What is the OCP?

In December 2001, the Open Core Protocol International Partnership Association (OCP-IP) leapt onto the scene. The OCP is the first fully-supported, openly-licensed, comprehensive interface socket for semiconductor intellectual property (IP) cores. Meanwhile, the OCP-IP is an independent, non-profit semiconductor industry consortium formed to administer the support, promotion and enhancement of the OCP. This consortium, which has grown to include more than 200 member companies, is generally making the world a much better place for SoC designers. For example, I'm informed that nearly half a billion "gizmos" have shipped with the OCP inside.

As creators of SoCs themselves, the folks who founded the OCP-IP were fully aware of the problem of integrating multiple cores from disparate sources and times. Knowing this, from its inception the OCP was developed and maintained with multicore designs in mind. The OCP's creators also understood that it was unrealistic to try to force the people creating cores to implement their interfaces in a specific way. In a nutshell, they realized that the solution was to be found in the form of a *configurable* socket.

Having said this, however, it's important to understand that the OCP is not a configurable socket per se; instead, it's a configurable socket *specification* that provides a consistent framework for the identification of all aspects of on-chip data, test, and control flows to or from a core.

In fact, the OCP specification is actually a superset of all the ways in which the designers of different cores may wish their functions to communicate. Suppose you have created a core for which you wish to create an OCP socket. You look at your core's control, data, and test signals, and the communications

features it must have like required burst sizes and specific addressing modes, and then you peruse the OCP specification to identify equivalent constructs that you use to create a corresponding socket.

Let's suppose you want your core to talk to "Max's Top Secret Core". We may have created our internal interfaces a little differently, but if I also provide you with an OCP socket that does the same thing you're doing with your socket, then you really don't need to know (or care) what I'm doing or how I implemented things inside my core.

So now we've gotten into a place where we've disposed of semantic challenges and we are communicating with a consistent vocabulary. Using the OCP, intellectual property (IP) designers can make their cores independent of specific bus protocols, and hence of any particular design implementation. Quite apart from facilitating your current SoC design, this allows easier reuse of OCP-compliant cores across multiple SoC designs. Also, the extreme configurability of the OCP allows you to take your existing cores with other interfaces and make them OCP-compliant more quickly than any other approach – without losing any performance!

We should also note that the OCP embraces more than a protocol, but also the parameters that control the protocol, which are captured in a metadata file that describes the selected parameter settings for a particular socket. This allows automatic tooling to look at the metafiles for different sockets that are talking to each other to make sure they are talking the same language (and also the same dialect).

Last but not least, because the OCP specification is so comprehensive, it's important to show new OCP users where to start. Thus, the OCP offering includes guidelines and sets of profiles, which are OCP configurations that work well for common tasks; for example, there's a profile focused on giving advice for DMA cores.

OCP TNG

Today's SoCs are the convergence point for functions that used to be implemented on separate devices. These functions range from simple legacy cores to complex modern processors. The OCP is efficient for both rudimentary cores with simple communications protocols, for high-end processors with advanced communications needs, and for the whole spectrum in-between; with regard to this level of scalability the OCP is unique in the industry.

And what for the future? Well, the current OCP release is 2.0, but OCP 3.0 – which I prefer to think of as OCP TNG ("The Next Generation") – includes all sorts of enhancements. One hot topic, for example is that of cache coherence. Not surprisingly, it is harder to create software for systems formed from special-purpose heterogeneous cores than for systems formed from their general-purpose homogeneous counterparts.

Multicore processors like those from ARM and MIPS are, of course, internally cache-coherent, but we're talking about coherence outside of these local clusters – "super-cache-coherence" if you will. In order to address this issue, the creators of IP cores are adding support in the hardware for system-wide cache coherence. Thus, OCP 3.0 has been augmented to include cache coherence signaling.

Power consumption is another hot-topic (pun intended). It is now common practice to power-down functions when they are not currently in use. In order to address this issue, the OCP 3.0 has been enhanced with the ability to add signaling to the sockets to allow one side of the interface to be turned off. This involves extra hand-shaking protocols that make it easier for one side of the interface to disconnect from the system and to turn off/on the clock and/or power in a safe manner.

Rather than listen to me, you should bounce over to the OCP-IP website (www.ocpip.org) to hear it from the experts and discover all of the different things they are doing (I didn't even get to mention Verification IP (VIP) for example). Until next time, have a good one!

About the Author

Clive (Max) Maxfield is the president of TechBites (www.TechBites.com), a marketing consultancy specializing in high-technology. Max is also the editor of the EE Times daughter site, Programmable Logic DesignLine (www.PLDesignLine.com).

Over the years, Max has designed everything from silicon chips to printed circuit boards. Max's articles have appeared in technical magazines around the world and he is the author and co-author of a number of books, including "Bebop to the Boolean Boogie (An Unconventional Guide to Electronics)", "The Design Warrior's Guide to FPGAs (Devices, Tools, and Flows)", and "How Computers Do Math".