

A focus on reducing power consumption in DSPs

Octasic's Opus asynchronous architecture addresses power consumption for multicore DSP applications. **By Robert Cravotta, Technical Editor**

“The high end of the application space allows us to get our hands around all parts of the product.”

DOUG MORRISSEY
OCTASIC

The Vocollo multicore media-gateway DSP is the first product employing the Opus core. *EDN* recently sat down and spoke with Tom Awad, Opus team leader; Doug Morrissey, chief technology officer and vice president; and Alain Legault, vice president of engineering, about the Opus engineering team.

A portion of that interview follows.

What were the issues you were trying to solve when you started putting together the Opus architecture?

A key driving factor was that we wanted to achieve a performance and power ratio that was three times better than the current offerings. Another key factor was that we did not want to accomplish this ratio at the expense of the programming model. This was a multidisciplinary team

effort where each part of the team had to work with and understand the issues of the other parts of the team. For example, the software guy had the responsibility to make sure that whatever we did maintained a standard [development] model to the customer in terms of how it ran. With a chip able to do low-channel versus high-density applications, we also wanted to introduce the concept of “pay as you grow,” where we can enable additional codecs to be turned on in the future as the customer increases requirements.

What kind of design and implementation issues did you face, and how did you address them?

The nature of the problem we were trying to solve required input from all of the disciplines on the team. [Because it is asynchronous architecture], we had to develop some of our own tools to enable us to address the gate-level and silicon

Octasic's Opus engineering team included (left to right) Doug Morrissey, Tom Awad, and Alain Legault.





part of the design. In terms of integrating that with the architecture and into the software, we had to have our own compiler development, tools development, and debug environment to deal with the core. Developing a compiler and assembly language that both could implement C code efficiently and be implementable in an asynchronous architecture took quite a bit of interaction and a lot of cross-education so that both sides knew what needed to be done. I think that was a real team effort to make that happen.

On top of that, ... while this is a great technology, a technology that cannot do anything is not worth too much, so we have a whole application that has been developed. Our first product offering includes [technology] all the way up to an entire application to work on a multicore DSP, which is the Vocallo gateway product. We are also working on other applications to go onto this and leverage the entire product platform.

We used our application team to validate the architecture as we went. The application team was developing the application even before everything was set in silicon. It was playing with the various paths we went down. We actually explored several paths as a team. Part of the decision of which was the right way to go was driven by schedule: When do you have to be complete? At some point, you have to be able to say it is good enough and move on. Each division of the team brought its own matters into the fray. In the end, everyone understood the other person's point of view and position, and we made a decision that this is the right time to freeze this portion of the design and move on.

Supporting the pay-as-you-grow concept requires a team effort because the way that is implemented is through software licensing. It involves everything from having nonvolatile RAM on the devices that needs to be configured during the manufacturing process all the way through the ordering system. So that what is delivered to the customer is actually a combination of silicon, a license file, and code licensing that is tailored to what the customer needs and can be upgraded over time.

There is quite a large software application just in that order process that manages the database and manages the configuration and needs to know everything about the software application and the

hardware, and they all had to be designed with this model in mind. We have a guy on the team doing Web business-to-business programming, so when a customer places an order on the Web, it goes down to cutting an encrypted key to the DSP nonvolatile memory.

What was the processing sweet spot you were trying for with your architecture?

We are a multicore product, and, as a small company, we look at the high end of the market place as an easier place for us to differentiate ourselves. As you move down the performance-application curve, it becomes much more of an integration effort. The high end of the application space allows us to get our hands around all parts of the product—be that at the gateway-application level and all the way down to the silicon. So, from that perspective, we're multicore. One of the values we bring is that of being low-power. One of the limiting conditions of multicore is: How many cores can you stick on the die because of the power consumption?

One of the requirements that we had was that a single core must have a high enough level of processing performance to be useful to the traditional [programming] paradigm. There are a lot of multicore people out there that are looking at really tiny cores that you have to rethink your whole application, your whole problem, in order to use that device. So, our single core must have a minimum threshold of performance. There is an argument where that number lies, but it is significant.

The other aspect is: How many cores can I stick on a device that will optimally position the product for the widest application or end-product space that we can? We tried to target from a cost or die-size perspective something that would allow us to address the wide range of end-application space cost-effectively. We ended up with 15 cores [in our first product] that allowed us to address the high-end applications and still be cost-effective to go down to what would be considered the lower end of performance. We felt that the [number of cores] was the right mix to address the range of products we were looking to address. In the gateway product, that [approach] allows us to go from as low as eight to 16 channels to as high

as 350 channels. If you look at the current products on the market, that range of performance would be divided into multiple silicon offerings, simply because the architecture does not lend itself to be cost-effective at the 16-channel space.

How do you preserve the traditional programming model with your multicore offering?

The biggest challenge with the asynchronous architecture and the compiler and getting all to work well together is branching. Any kind of branching in the code that changes the execution order, however that occurs, via a jump or some sort of test, causes challenges in asynchronous architectures. You are breaking the anticipated flow, and a normal assumption when you are architecting a core is that you are going to go from Address 1 to Address 2 to Address 3 and so on. Anytime you have a branch, for whatever reason, you're not going to execute Address 4 next; you're going to have to go somewhere else and start executing from there. In an asynchronous architecture, that is a synchronous event, so to speak, with the code execution. The instructions themselves, as long as you're doing them in order, [give] you no feedback that you have to react instantaneously to. It's a problem in synchronous designs, as well, but it produces some other, unique challenges in the asynchronous world.

As the team examined the implications to traditional compiler technologies and traditional assembly instructions, we were able to develop some unique execution-control-flow instructions or methodologies that the compiler was able to incorporate and allow us to avoid unnecessary branching in a way that people had not come across before.

We have a full ANSI C compiler. There are pragmas to help with optimization choices for the compiler. One of the interesting things is that the assembly is highly C structured. The assembly is inline with the C, but it is not just inline assembly; the C constructs flow into the assembly language. You can use any stack reference in the assembly instructions, as well as in the C instructions, and can flow back and forth between the two languages. The assembly supports all of the C typing. We have one semaphore to support the ability for cores to poke into the local register set of another core. ■