FPGA and Structured ASIC JOURNAL
TRENDS TECHNOLOGY TOOLS TECHNIQUES
www.fpgajournal.com

# Assemble All Ye IP

Using Simulink for DSP Design

There are two levels of DSP design. First, there's the conceptual level, where hard-core algorithm development rules the day. Your big concern here is the numerical correctness of your algorithm, but there's no timing information or data typing to fret about. This is the comfort zone for the traditional DSP designer. You're dealing with a problem from a purely mathematical point of view, using a procedural language like "M" in the MathWorks' MATLAB, which is suited for un-timed algorithms with mathematically friendly data types to fine-tune your formula.

Then there's the implementation level, where you take that shiny new algorithm and implement it in either software or (queue ominous music) hardware. Hailing from the software side of town, most DSP designers have no trouble creating an application to run on a traditional DSP processor. They might need to consult with a specialist for a tweak or two, but it's all still software. Trouble is, that trusty-old DSP processor may not have the horsepower to handle your high-performance design requirements any more, at least not on its own.

This is a place where FPGAs have been taking hold in a big way, becoming the platform of choice for high-performance DSP implementation, either replacing several DSP processors or augmenting one for the heavy lifting. FPGAs offer serious benefits for cost, performance, and power consumption because of their ability to do complex computations in parallel rather than sequentially like a DSP processor. The key to exploiting the performance benefit, however, is being able to access this capability without resorting to complex VHDL- or Verilog-based custom hardware design. If you're working in C, you could try to take your code and retarget it running on a DSP for an FPGA, but you would need to do C-to-hardware synthesis using an advanced tool to get anywhere. It's not a straightforward process because C code that was targeted to run on a processor was almost certainly optimized for a sequential processing machine, and would probably need significant modification in order to take advantage of the parallelism available in an FPGA architecture. Getting to where you need to go is possible, but it may take more time, money, and complex tool expertise than you have to spare.

So, what’s a DSP designer to do? How can you accelerate your design into hardware if you’re not a hardware expert?

The path to hardware implementation doesn’t have to be fraught with peril (or bleeding-edge, super-expensive tools). In fact, you can leverage a MathWorks Simulink environment that may already be familiar to you to complete your entire design using an IP assembly approach. Simulink costs quite a bit more than its modestly priced MATLAB counterpart, but the two environments are fully integrated, so you still have full access to the algorithm development tools you’re used to using for data visualization and analysis. A big difference between the two tools is that MATLAB has no inherent notion of time. Simulink does. It provides you with a bridge to the next step in your implementation process.

Think of Simulink as a hub for your DSP implementation. You can use it to graphically capture your design based on IP blocks, and then simulate, analyze, implement, and test your system. You can also create an IP-based design flow within Simulink by adding third-party tools and IP from FPGA vendors like Altera, Lattice, and Xilinx, or from EDA vendors like Synplicity. More on that in a bit. You can model your system using a selection of more than 1,000 blocks that implement common functions. You can choose from algorithmic blocks, like sum, product, or LUT; structural blocks such as mux or switch; or continuous and discrete dynamic blocks such as unit delay. In addition to the basic blocks, Simulink has features that allow you to customize the blocks or create your own, and includes additional blocksets for specific applications.

This is an important point to note. Simulink is a multi-domain environment. It’s used not only for DSP design, but also for things like control system design and general system modeling. Therefore, it’s critical to understand how to use Simulink in “DSP mode.” For example, as we were just discussing, you’ll need to choose the Simulink signal processing blockset to give you the correct IP and sufficient signal processing capability for your design. You’ll need to employ frame-based processing as opposed to sample-by-sample processing, which would be more appropriate for control applications. If you think of the application possibilities as a continuum, at one end you have low-data-rate, deterministic control applications where you care about absolute determinism and low latency. At the other end, you need high throughput, but may be more flexible on latency and real-time determinism. Simulink can handle both extremes, but you really need to know how to set it up to operate efficiently for your application. In other words: A little bit of time spent learning the system up front will save lots of time later on.

You may be thinking, “Sure, this sounds fine for some people, but my algorithm is extraordinarily complicated, chock full of linear algebraic functions, and I don’t see how I could easily recreate it using pre-defined blocks.” In this case, you may consider bypassing Simulink and going with a DSP synthesis solution from a company like AccelChip. AccelChip specializes in taking challenging algorithmic IP straight from MATLAB to RTL, essentially skipping these steps in the Simulink flow. AccelChip’s DSP synthesis tool automates your floating- to fixed-point conversion, enables design exploration, and generates your HDL. The cool thing if you’re targeting FPGAs, though, is that AccelChip also has a downstream connection back into the Simulink flow through Xilinx.

Assuming for the sake of this discussion that you don’t have the world’s toughest algorithm and that you are, in fact, using an IP assembly approach for your DSP implementation, your next step is to do a floating- to fixed-point conversion, or quantization. Quantization allows you to represent your input signal with a finite number of values, helping you to limit the bandwidth of your transmitted signal. Using fixed-point data reduces the memory requirements and increases the speed of code generated from your model. In software (as in a DSP processor) your registers and datapaths are already built, and have a fixed width. When you move your algorithm to hardware, however, you can usually trim off bunches of bits, saving huge amounts of hardware, and improving performance at the same time. In MATLAB, you would access the fixed-point toolbox to convert basic math and logical operations to fixed point. Once you declare the variable, the algorithm doesn’t have to change. In Simulink, you would be working with a tool called Simulink Fixed Point to accomplish the same task.

Now it’s time to decide what type of device you want to target, and what Simulink subflow would be best for you. You might choose an FPGA-vendor specific flow from Xilinx, Altera, or Lattice if you’re sure you want to use one of their FPGAs. If you’re not sure, or if you’re using your FPGAs for ASIC prototyping, you might go the route of using a vendor-neutral flow with a tool like Synplicity’s Synplify DSP. That’s not to say, by the way, that you couldn’t choose an FPGA vendor flow for that scenario, too, but you’d have an extra step to deal with. When you got ready to move from FPGA to ASIC, you would have to go back to your original reference model and hand code the HDL for the ASIC. There’s no free lunch, or in this case, automatic retargeting.

If you have decided to go with an Altera flow, you will be working with a tool called DSP Builder, which sits on top of Simulink and provides you with Altera libraries that have hardware-optimized models behind them. As of last week, you can also add in your own libraries. Altera has just announced this enhancement, included in Version 5.1 of DSP Builder. This new functionality expands your capabilities because you're not just relying on the IP from Altera. You may have an optimized algorithm that was written in VHDL or Verilog, and now you can import that and simulate it with the rest of the DSP Builder library and Simulink functions. They also offer a hardware-in-the-loop feature to accelerate system-level co-simulation and debugging.

The Xilinx path to Simulink is through their System Generator for DSP. System Generator gives you the ability to incorporate different languages in the design up front. This could come in handy if your team has a bunch of legacy HDL that they want to add, and they don't want to bolt it in at the end. You can insert your HDL at the front end of the process, during the system-level modeling, and verify your whole design as part of the system. Xilinx also has hardware-in-the-loop functionality for their flow, and, as mentioned earlier, they also offer integration with AccelChip. Remember that MATLAB algorithm that bypassed Simulink and went through AccelChip's DSP synthesis tool? Well, you can export the fruits of your labor in AccelChip directly into the Xilinx System Generator for DSP as a new block. What this means is that from Simulink, you can press a button and generate the HDL for the whole system (including MATLAB blocks, HDL blocks, and Simulink primitives), as well as the bit stream for your FPGA.

Xilinx also recently announced the availability of system development kits to provide application-specific help and IP to speed you through your process.

Lattice Semiconductor is taking more of a "bring it to the masses" approach by providing a Simulink blockset as a standard part of their FPGA design tools. Their DSP generator tool is available as a standard part of the Lattice toolset. As more of a fringe player in this potentially huge market, their goal is to increase the adoption rate of FPGAs for DSP by making it easy to check out their tool and use it with Simulink.

What if you're looking for technology independence in your Simulink subflow? You've looked at the vendor-specific offerings, but perhaps you have a plan to use your Simulink source in multiple FPGA architectures. Synplicity's Synplify DSP automates the implementation of RTL directly from a Simulink specification, and lets you target any FPGA hardware from a single Simulink model. Using Synplify DSP, you can optimize your Simulink design at the system level, prior to RTL, to maximize your results for timing and area. You can also take advantage of automatic testbench generation for HDL simulators.

You may find yourself in a situation where you have DSP engineers creating models in Simulink, and FPGA engineers writing RTL, and you need to make sure that the RTL matches what you intend to build. The MathWorks has a bi-directional co-simulation interface to Mentor Graphics' ModelSim HDL simulator that integrates MATLAB and Simulink into the HDL flow so you can verify and co-simulate ModelSim RTL-level models from inside your Simulink environment. The "bidirectional" part of the interface is important if you happen to be the FPGA engineer on the team, because you can take MATLAB or Simulink models and run them as a component in ModelSim. Maybe you have a testbench that was written in Simulink. You don't have to rewrite that in RTL. You can just import the test stimulus and the test harness from the MATLAB code or the Simulink model and run it directly in ModelSim.

Based on all of these assertions about "automatic generation of HDL" being bandied about, you may be wondering if it's possible to actually make it all the way through the Simulink flow and into an FPGA without writing any HDL. The short answer is yes, it can be done. You can hop on The MathWorks site and check out success stories on the topic. But before you start doing a jig in celebration, remember that you're still going to be best served by having the ability to exploit all levels of your design, and if you have a spare hardware guy hanging around to lend some expertise, all the better.

*Amy Malagamba , FPGA and Structured ASIC Journal*

*November 15, 2005*