What Color is Your Abstraction? FPGA Design Starts with You

by Kevin Morris January 21, 2014 (Reiner's version)

In the early FPGA days, we worked with schematics. FPGAs were small, and you could stitch together a little gate-level schematic pretty easily. Then, it was just a matter of running the FPGA tool flow, and a few seconds later you had a **bitstream** ready to program your cute little programmable logic device. It was all pretty easy, and - with schematics being the universal language of EE, there wasn't a lot of special skill required.

About fifteen years ago, FPGAs outgrew **gate-level** schematic entry. We **moved on to** hardware description languages like **VHDL** and Verilog and began fighting with logic synthesis tools to try to get our designs to behave. This actually narrowed the field of FPGA designers somewhat. The universe of **"people who knew HDL and synthesis"** was quite a bit smaller than "people who could draw a decent schematic." **The FPGA companies didn't care**. They were chasing the lucrative communications and networking infrastructure market, and the folks writing the big checks had plenty of HDL experts.

Fast forward a few decades and a couple million logic elements, and things have gotten substantially more complicated. FPGAs have pretty much maxed out the telecom market, and, if the pie is to grow, FPGA companies **must expand into more large markets.** Unfortunately, **each** of those expansion markets **speaks a different language** when it comes to design creation. **Most of them aren't all that keen to spend the time and effort to make themselves into "FPGA experts" - they're too busy being expert in their own respective application domains**.

In order to win in these new markets, the **FPGA design flow has to speak the language of the new customer**, rather than try to teach all the new customers the language of FPGA. That means that **relying on plain-old VHDL** and Verilog for design entry **just won't cut it**. **We need some new abstractions** that can feed smoothly into the FPGA design flow, **enabling engineers in new markets to take advantage of the power of modern FPGAs.**

With today's FPGAs excelling in high-speed connectivity, high-performance signal processing, and **computational acceleration,** it's challenging to know where to even start on design abstractions. Should we be looking at ways **to go from Matlab algorithmic abstractions into FPGAs?** Or, should we be taking system-level block diagrams down into hardware implementations? Should we be taking C, C++, **or other "software" languages** and compiling them directly into optimized hardware abstractions? Or, should we be looking at **parallel programming languages** such as OpenCL or CUDA to **help the supercomputing crowd supercompute with FPGAs?**

Perhaps, the correct answer is**, "all of the above."**

Last week, we sat down with **Tom Feist**, Sr., Director of Design Methodology Marketing **at Xilinx**, to talk about **the range of design abstractions enabled by today's tool suites** - and specifically Xilinx's new state-of-the-art **Vivado suite**. Feist explained that Xilinx has spent a lot of time and energy understanding the various markets served (and potentially served) by today's FPGA technology (Xilinx prefers the term "All Programmable" to FPGA - as they see their devices **having programmability in many aspects such as software, IO, and analog**, in addition to the traditional FPGA LUT fabric programmability.)

Feist described five major target application areas that Xilinx is focused on serving - **networking (as usual), data centers, intelligent vision, factory automation, and energy**. Each of these application domains has specific challenges that lend themselves well to the capabilities of FPGAs. In particular, the performance of FPGAs in terms of computation per power unit is **unrivaled** by just about any other technology. Competing with multi-core DSPs in signal processing, for example, an FPGA can generate **many times the performance - on a tenth of the power** that the DSP would use. Unfortunately, however, **the complexity** of getting an FPGA to execute your code **is unrivaled as well.** That's where Xilinx's efforts in **supporting a new range of design abstractions become critical.**

In looking at the abstraction picture, Xilinx started with the trend in system engineering, and that is that design teams these days tend to have **4x-10x more software engineers than hardware engineers**. If they want to get more people putting critical functions into programmable devices**, the people to persuade are the software types**. Depending on the application, those software engineers are **using languages like C, C++, and OpenCL**. The higher-level system engineers are using languages like **SystemC and** abstractions like high-level diagrams connecting up complex IP blocks. The hard-core algorithm developers are **using abstractions such as Matlab and Simulink** - particularly in the signal-processing domain, and industrial instrumentation and control people often gravitate toward **National Instruments' "LabVIEW" approach**.

Today, all of these are valid approaches to FPGA design, and **all of them are supported at some level in Xilinx's Vivado suite,** with more robust support for many due in 2014.

On the partnership side, the company has long-standing relationships with Mathworks (Matlab and Simulink) and National Instruments (LabVIEW and RIO) to work cooperatively on design flows that go smoothly from those companies' respective front-end abstractions into the FPGA implementation flow.

The company has also put significant engineering into creating a smooth IP-based flow, **allowing third-party IP blocks** to be easily stitched together to quickly create a working FPGA implementation. The company's **"IP Integrator"** has been an early hit with designers who like the correct-by-construction approach to stitching IP blocks together. When there isn't an IP block that does exactly what you need, you might consider **the company's high-level synthesis capabilities**, which allow you to **synthesize C-language algorithmic blocks into optimized hardware datapaths.**

The most challenging group to address is that of **software engineers wanting to accelerate algorithms into heterogeneous** computing systems, including hardware accelerators. By the second half of this year, the company **plans to support C/C++ and OpenCL implementation flows with Vivado. OpenCL support** may be a bit of a concession to archrival Altera - who has been touting OpenCL support for the past couple of years. **OpenCL is popular** for writing software-targeting parallel computing platforms such as GPUs. In many applications, **FPGAs can deliver superior performance to GPUs, with a fraction of the power consumption.** The key to achieving that, however, is to **generate an optimized FPGA implementation from an OpenCL description, without requiring** the software developer **to become an expert in FPGA hardware architecture and design methods**.

Feist said that these alternative abstractions could deliver on the order of **15x the productivity of a straight RTL-based flow** for designers in these application domains. They also work to **eliminate the "over the wall"** handoff of algorithms from the "algorithm guy" to the "FPGA expert" - which is a step **that often involves complicated communication** between engineering disciplines. Simply the **elimination of the requirement to understand RTL design** in order to prototype an algorithm using **FPGAs is a major door-opener** for many engineering teams.

It isn't clear at this point **which of these abstractions might generate the "home run"** that propels FPGAs into widespread adoption in a new application area. Certainly the programming **languages like OpenCL** that promise to allow software engineers to target FPGAs just like they would target a conventional processor or a GPU **have the potential to unlock** enormous new audiences for FPGAs. However, widespread adoption of any technology depends on a number of funky factors - providing the right tools and hardware platforms are certainly necessary ingredients in that recipe, but they don't provide any guarantee of success. **We'll all just have to watch and wait to see what takes off and what doesn't.**

Channels EDA. FPGA.

An excellent article about how to cope with the FPL marketing paradox !!