



## IMPULSE: "C" TO FPGA PROGRAMMING SOLUTIONS

1 October 2008: "C" to FPGA Programming Solutions

INTERVIEWEE. RALPH BODENNER,  
DIRECTOR OF PRODUCT DEVELOPMENT  
TEL. (425) 605-9543  
EMAIL. RALPH.BODENNER@IMPULSEC.COM  
COMPANY. IMPULSE ACCELERATED TECHNOLOGIES, INC.  
WEB. <http://www.ImpulseC.com/>

**Q. First of all, tell us a little bit about yourself and your responsibilities at Impulse.**

A. I've been leading the product development team at Impulse for the past five years, and through three major releases. I have a number of roles at Impulse, but my primary responsibility is prioritizing and managing new releases, as well as working with our partners to extend the CoDeveloper product to new FPGA-based platforms. Our company is geographically distributed, with developers in Washington, Colorado and Vermont. I'm personally located in the Boulder area.

**Q. One of the great yet somewhat unfulfilled promises of FPGA is to take software problems and "accelerate" them by migrating them to hardware. How does Impulse help developers put into practice this FPGA potentiality?**

A. We believe that FPGA acceleration has moved well beyond potential and has been proven to be both practical and cost-effective. There are significant commercial and defense deployments already in image processing – for example in robotics, medical imaging and UAVs – as well as an increasing number of successes in high performance computing domains such as life sciences and financial feed handling. What is really changing now, though, is the robustness and availability of tools and platforms. Modern tools and libraries make it practical for software developers to target FPGAs. These tools don't make FPGA programming trivial, but they do make it far more productive and practical than lower-level methods based on hardware description languages. It's also important that software programmers have access to a wide range of FPGA-based platforms and systems appropriate for specific application domains. This is why ongoing partnerships with platform vendors are so important to us, and to our customers.

**Q. More specifically, you mentioned in our talks beforehand, that Impulse has some new ideas to share on the frequent problem of migrating microprocessor code that is largely "serial" to the "parallel" programming environment of FPGAs. How so?**

A. As an organization that produces both design tools as well as providing design services, we have developed a pretty good understanding of what software programmers need to know when retargeting their existing software applications to FPGA-based platforms. Although many software algorithms can be ported at the subroutine level to an FPGA, for the highest performance and best hardware utilization some level of algorithm refactoring will usually be required. The most important insight – the "aha" moment if you will – comes when a software programmer begins to think in terms of data moving serially through a parallel, pipelined system. In an FPGA you have the possibility of pipelining at the process-level, at the level of loops, and at the level of individual, multi-cycle operators as well. A C-to-hardware compiler can help to automatically detect and generate such pipelines, but it can't do everything. A software programmer can therefore get much better results by thinking of ways to increase pipelining and

parallelism at a system level, and provide guidance to the automated compiler tools. Once a programmer understands these concepts, designing a highly efficient algorithm for an FPGA turns out to be not so different than designing a processor-based application involving multiple threads or processes. It's just a subtle change in the programming model. The Impulse C compiler features and associated API functions make such applications relatively easy to describe, and easy to debug as well using standard C tools.

**Q. This brings up the whole transition to multicore and parallel processing. Experts say that the software is the current stumbling block. So is it fair to put Impulse on the “cutting edge” of helping us transition to multicore, at least in terms of FPGA-based multicore?**

A. Software tools have certainly been a barrier to the increased use of parallel processing and massively multicore systems. In an FPGA-based platform, the opportunities and challenges of multicore processing become more extreme than in perhaps any other platform technology. Consider, for example, a large FPGA-accelerated high performance computing (HPC) platform. In such a platform, the application programmer may have access to multiple commodity processors, each of which is closely coupled to one or more large-capacity FPGA devices. And each FPGA device in turn may include an embedded cluster of hard and soft processor cores, and embedded or adjacent blocks of shared memory. These FPGA-embedded cores may or may not be running some embedded operating system, such as Linux, that provides scheduling for multiple threads or processes, which themselves are communicating with customized FPGA hardware accelerators.

Is anyone creating such a hybrid-processing platform today? Absolutely, in fact you can buy such a system from SGI or Cray already, or you can assemble it yourself using off-the-shelf components. Does Impulse C make the programming of such a platform easy? Not really. But we can make it *practical* for software programmers to do the job, or at least 90% of the job, without the detailed hardware-level knowledge that would otherwise be required.

**Q. Many software developers know little about hardware, yet want to take advantage of FPGAs. How little “hardware” knowledge might be required of a developer, who is fluent in “C” yet works in an FPGA design team?**

A. We have introduced FPGA programming to many users who have never experienced programming at the hardware level, have never used VHDL or Verilog, and who had no prior knowledge with digital design. How much knowledge such a person needs depends in large part on what kind of FPGA-based platform they are targeting, and their requirements for non-standard I/O. For someone who is targeting a well-supported HPC platform (a platform for which we offer a complete platform support package), a minimum understanding of the I/O functions that we provide, along with a reference example, will normally be enough to get them started. For someone creating a custom hardware/software system involving direct input of data from a network interface, an A-to-D converter, a camera or some other input, then more hardware knowledge is clearly going to be needed to create bridging interfaces.

Although we promote higher-level methods of design, we recommend that anyone wanting to get the most out of an FPGA should spend a day or two learning the fundamentals of HDL-based programming, perhaps by stepping through a few HDL tutorials provided by the FPGA vendors. This helps not because we want our users to be dropping into HDL, but so they can think more intuitively about FPGA parallelism, and so they can have a better understanding of the messages and reports that are generated by FPGA synthesis and place-and-route tools. HDL fundamentals are always a part of our Impulse C user training programs, for exactly those reasons.

**Q. What applications are particularly helped by your approach? Are they largely compute-intensive like image or DSP?**

- A. Image processing is by far the biggest application market we are serving today. We think this is true for a number of reasons. Image processing algorithms are extraordinarily compute-intensive, with a high degree of parallelizing possible. Image processing algorithms are also inherently pipelined, making them ideal for the flexible fabric of an FPGA and for the Impulse C programming model. And perhaps just as important, image-processing algorithms are diverse, complex and fast changing. While many DSP applications can be built up quickly using DSP filter libraries (and using toolkits based on MATLAB and Simulink), image-processing applications often require hand-optimized, highly customized code that is unique to a particular application.

Other domains share these same characteristics, including many scientific algorithms; data compression and encryption, financial feed handling and real-time analytics. We also believe there are significant opportunities in areas such as computational fluid dynamics, particle simulations and bioinformatics.

**Q. What is the best way to “get started” for someone who is not familiar with Impulse? Do you offer free tutorials, webinars, or demo’s? Can you point us to some of the web URL’s where an interested developer could learn more before committing to a purchase?**

- A. The best way to get started with Impulse C is to obtain a low-cost FPGA platform for which we have ready-to-run examples and tutorials. These examples and tutorials guide you through the process of creating and partitioning an application, understanding and using FPGA parallelism, and generating hardware. The tutorials also walk you through the FPGA synthesis and place-and-route process, which is an aspect of FPGAs that most software programmers have never seen. Our goal in these examples is to demystify the FPGA development process. Not trivialize it – FPGA design is not as simple as programming for your PC – but at least make it an environment in which a software programmer can be productive.

Even if a software programmer doesn’t yet have access to an FPGA device, our evaluation software and training materials (including the Prentice Hall title *Practical FPGA Programming in C* authored by David Pellerin and Scott Thibault) can help speed the learning process and lead to better-informed decisions regarding FPGA platform selection.

The evaluation version of Impulse CoDeveloper can be obtained on request from <http://www.ImpulseC.com/eval>.

I should also mention that we hold weekly live webinars in which we cover FPGA programming methods and provide demonstrations of the entire software-to-hardware tool flow. Information about these webinars can be found at <http://www.ImpulseC.com/webinar>.

**Q. Thank you for this interview.**