

Accelerating algorithms in FPGAs, one process at a time

By David Pellerin

FPGAs have become commonplace in embedded systems and are now beginning to appear in high-performance, server-class computing applications as well. This shift toward hardware-accelerated computing has been driven by two factors: increased FPGA device capabilities and improved software-to-hardware tool flows. Emerging tools can help demystify FPGAs, making them more accessible to software application programmers.

Programmable logic has come a long way since its humble beginnings in the mid to late 1970s. FPGAs first appeared in 1984 and now serve as highly capable platforms for high-performance embedded computing. Developers use FPGAs to integrate embedded processors with custom hardware peripherals, forming single-chip embedded systems. FPGAs also can serve as coprocessors for enterprise and scientific computing (see Figure 1).

FPGAs have proven themselves in a broad range of hardware applications, including telecommunications, aerospace and defense, medical electronics, and consumer electronics. FPGAs can be found inside set-top boxes, big-screen televisions, and car dashboards as well as on Mars in the robot brains of NASA's Mars Exploration Rovers *Spirit* and *Opportunity*.

Because they are memory technologies, FPGAs have increased in size and density annually in accordance with Moore's Law. FPGAs have grown to the point where they can now be considered for entirely new types of computing applications – applications that until recently would have been in the domain of traditional CPU architectures. As a result, the high-performance com-

puting community is beginning to experiment with and deploy these devices to accelerate algorithms in areas as diverse as financial modeling, scientific computing, and bioinformatics.

On paper, an FPGA sounds like a perfect fit for these types of applications. From a gigaflops-per-watt perspective, no other fundamental computing technology can match a modern FPGA. But there is a catch: FPGAs are notoriously difficult for software programmers to use. The two most common programming

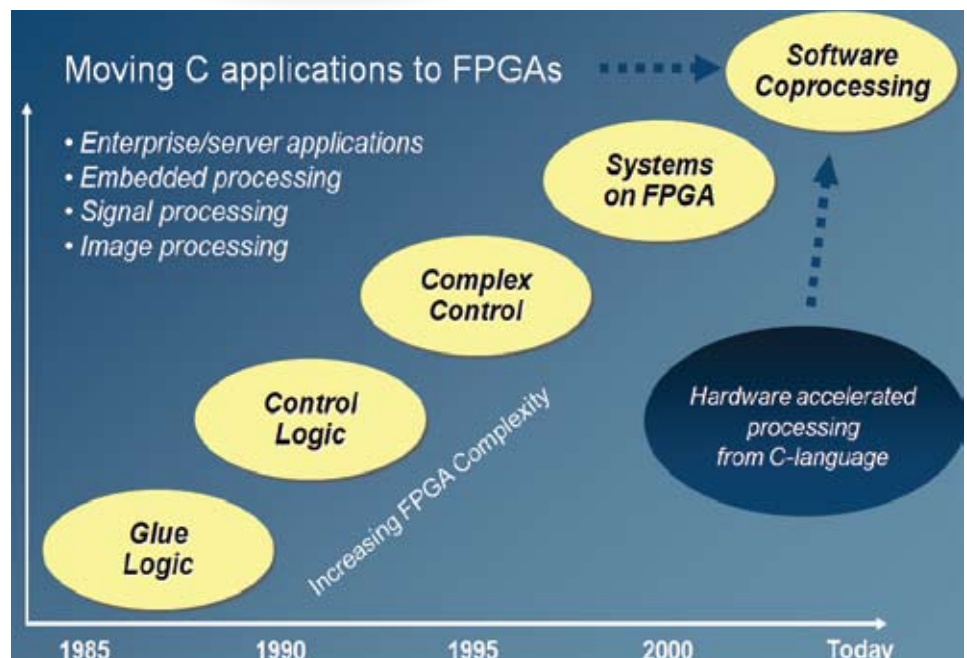


Figure 1

languages for FPGA programming, VHDL and Verilog, are inscrutable to the average software programmer, and the tool flow from those languages to working hardware can be difficult to comprehend and slow to complete.

However, software-to-FPGA compiler tools are now available and maturing every year. These tools offer significant benefits, not the least of which is design productivity. C-to-hardware tools in particular (see Figure 2) are gaining converts from both software and hardware design backgrounds. When properly applied, these tools can give software programmers access to FPGAs as computing devices.

Hurdles to overcome

At this point in the evolution of FPGAs, two near-term hurdles must be overcome before FPGAs become truly commonplace as software acceleration platforms. The first hurdle is the lack of a consistent, integrated, and portable tool flow. FPGA platform vendors provide some parts of this tool flow, most notably the placement and routing software required to implement logic within an FPGA. Third-party tool vendors provide other parts of the flow, including both hardware and software front-end design tools.

FPGA vendors recognize the value of device-independent software tools, and have historically worked to foster and promote partner tools ecosystems for higher-level FPGA design. These tool ecosystems encourage competition and innovation, but also present a bewildering set of choices to new users, with very few standards for interoperability. For FPGAs to flourish in software applications, the tools environment for FPGAs must become more consistent and software-like, while at the same time providing ways to take direct advantage of what an FPGA has to offer in terms of performance. Organizations such as OpenFPGA can play a role in improving interoperability.

The second critical hurdle to overcome is software designers' knowledge gap concerning what an FPGA is and what it can do. What does the term *field programmable* mean to a software programmer familiar with traditional processors? What is a *gate array* and how does it relate to running software algorithms? What does the term *electronic system-level design* mean to a financial or scientific application developer?

Demystifying the FPGA as a computing platform is therefore critical. Just as a good development suite for traditional processors encourages efficient programming methods for processors, a tool suite for FPGAs must encourage programming methods that allow software programmers to utilize available FPGA resources and make intelligent decisions regarding hardware/software partitioning and source code optimization.

Software-to-hardware tools break down barriers

Because FPGAs have become commonplace in the domain of embedded systems, it makes sense for enterprise and scientific

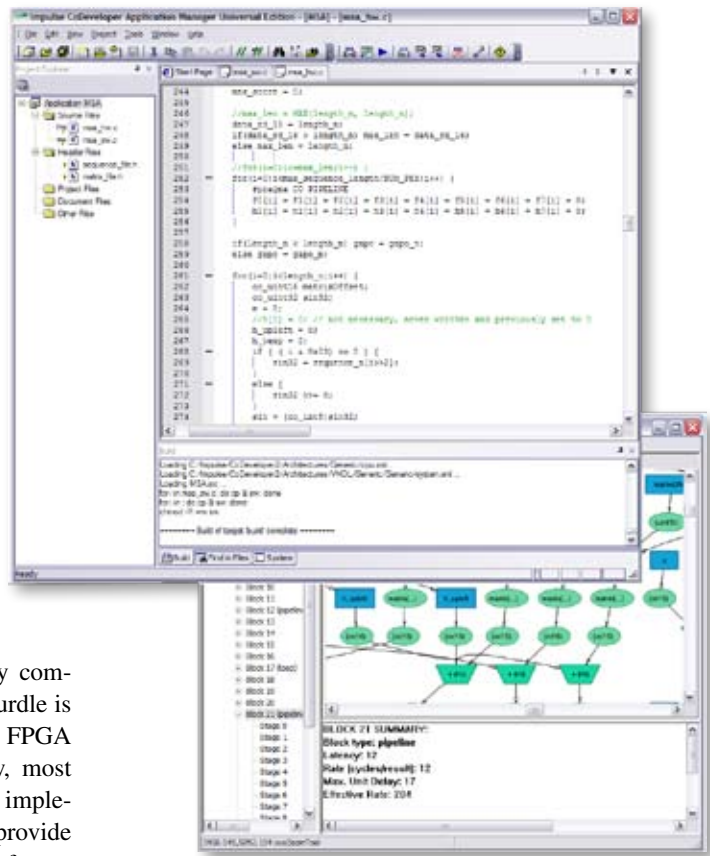


Figure 2

application developers to examine what has and has not worked for embedded systems designers.

Embedded systems design teams are often composed of hardware and software specialists. In some cases, the embedded systems designer will have the complete skill set needed to program a complex application *and* design hardware components. However, more often than not, the disciplines of hardware and software design have remained distinct. Today, software-to-hardware compiler tools are beginning to change this differentiation.

Most embedded systems design teams freeze the hardware first, then continue to work on software until later in the project cycle. Some hardware/software codevelopment may occur to ensure that the overall system achieves its objectives; nonetheless, a schedule mismatch exists between the two groups. One key appeal of programmable logic is that it allows hardware freezing to occur later in development. FPGAs encourage rapid prototyping and design flexibility and allow for future modifications without the need to completely respin the board or system. Software-to-hardware tools build on the benefits of FPGAs by reducing the time to prototype and encouraging iterative, explorative design methods.

The advantages of FPGAs and software-to-hardware tools are exactly what higher-level, enterprise-type applications need. Algorithm performance and throughput are of paramount importance to software developers. Even more so than in embedded systems, software drives the project forward, leaving the hardware platform as nothing more than a platform, replaceable at any time.

The challenge moving forward is to provide tools that best utilize on-chip processing capabilities without forcing FPGA developers to become hardware design experts.

Getting from C to FPGA hardware

For a software developer – someone not experienced with FPGAs or hardware design in general – the tool flow from software to hardware is challenging. Software teams have intellectual property and expertise developed using GCC, Visual Studio, Eclipse, or other software integrated development environments.

When moving into FPGAs, these software programmers must have a software-oriented design environment in which to work. If it is a simple application that can be represented as a single FPGA process, they may be able to wrap their legacy algorithm using FPGA-friendly API functions, recompile/cross-compile the application for an FPGA target, and go. But to achieve efficient results for larger, more complex applications, software programmers need to understand more about the tool flow and methodology. For a C-to-FPGA tool, the tool flow includes the following fundamental steps:

1. Design and debug the application using standard C tools
2. Partition the C language algorithm between hardware and software
3. Optimize the C code for an FPGA, introducing parallelism
4. Synthesize and generate a bit file via FPGA place and route tools
5. Download to an FPGA development board or system for testing

While these steps sound simple, they can represent an iteration time measured in hours or even days. Most successful FPGA projects are therefore built in steps, using partitioning techniques to reduce compile times.

The methodology for a single FPGA accelerator process is straightforward. For applications requiring multiple hardware processes, however, it takes a bit more effort. Developers must focus on one critical function at a time and use iterative partitioning and optimization methods to move incrementally more of the application into the FPGA.

There is no magic tool that will, without fail, take arbitrary C code in and generate efficient hardware. C-to-FPGA tools can help make FPGAs more accessible for software/hardware design teams but require some specialized knowledge. Modern FPGAs offer impressive sets of features, including embedded microprocessor cores. The challenge moving forward is to provide tools that best utilize on-chip processing capabilities without forcing

FPGA developers to become hardware design experts.

FPGA reconfiguration gaining a greater role

FPGAs' future for high-performance computing appears positive. Software developers are learning about these devices, and higher-level tools including Impulse C can help them start their first projects.

Looking forward, developers can expect that dynamic FPGA reconfiguration will become increasingly important. This reconfiguration will require some level of operating system support to be efficiently performed and the ability to partially reconfigure the FPGA without resetting it. Partial reconfiguration is expected to be an enabler for shorter place and route times and correspondingly quicker design iterations. Standards must emerge to allow each tool in the chain to be evaluated on its merits.✦



David Pellerin is cofounder and CTO of Impulse Accelerated Technologies in Kirkland, Washington. He is a programmable logic industry veteran and the author of five books, including Practical FPGA Programming in C, VHDL Made Easy, and Practical

Design Using Programmable Logic. David was formerly the president and founder of Accolade Design Automation, an FPGA tools and training provider. Prior to that, he spent more than a decade as staff member of Data I/O Corporation, where he was a key contributor to the development of programmable logic device and FPGA tools. David can be contacted at david.pellerin@ImpulseC.com.

Impulse Accelerated Technologies, Inc.

425-605-9543

www.ImpulseC.com