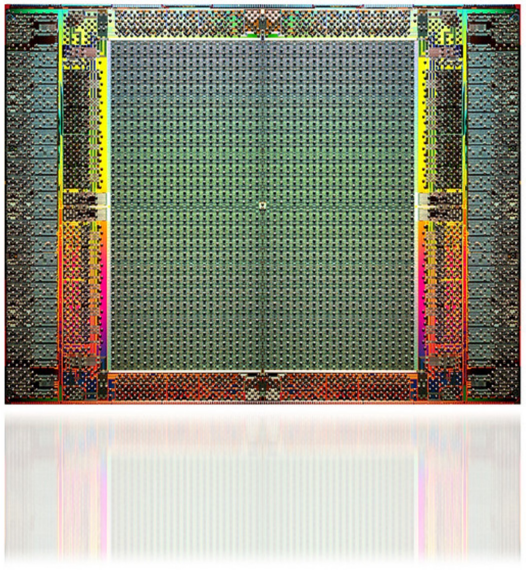

Reconfigurable Logic in 2032

Challenges and Opportunities in the Next 20 Years

Bob Blainey
IBM Software Group

Weasel words

- My opinions are informed mostly through experience with FPGA as an accelerator employed in server-side systems.
  - NOTE: The contents of this presentation are my own opinions and should not be interpreted as official statements from IBM
- My experience and focus is on software architecture and optimization, not hardware design. However, I will make a number of assumptions about the progress of hardware.
- Predicting anything 20 years in the future in this industry is foolhardy but I am in good company. ☺
  - My predictions err on the side of caution so don't expect any science fiction.
- My prediction in a nutshell is that FPGAs will become an integral part of every system and become opaque to users, developers and system administrators

Three Predictions for 2032

- **Integration**
  Reconfigurable logic in some form will be tightly integrated into virtually every system – server, communications, mobile and embedded. It will be driven by different factors for different applications – time to market, cost reduction, high performance, low latency and/or power efficiency.

- **Programmability**
  The reconfigurable logic elements will be programmed by software engineers in a manner consistent with other elements of the system. Explicitly parallel languages and tools will be in common use but so will automated code generation from higher level programming models.

- **Virtualization**
  Reconfigurable logic will be securely and reliably managed as a dynamically shared resource across many virtual machines and tenants. It will be a commonly utilized resource in cloud infrastructure.

Integration – embedded and mobile

- The drivers will be principally time to market and power efficiency.
- Integrated CPU/FPGA SoC designs will be commonplace and reduce or eliminate the need for ASICs, providing single chip solutions for most mobile devices.
- Mobile will drive the most diverse new requirements for FPGA due to evolving user interface technologies (natural language, haptics, 3D displays, geo-spatial reasoning, other AI) and the continuing evolution of wireless networks.
- FPGA usage in embedded systems will expand, driven by greater system integration but also by improved programmability. Automotive systems, for example, will exploit FPGAs extensively for telematics, entertainment and driver communications. Of course cars should also be flying by 2032 ☺

Integration – servers and storage

- The driver will be principally price-performance.
- Servers will have thoroughly adopted 3D integration by 2032 and integration of FPGA along with memory and conventional processors in 3D packages will be commonplace. Indeed, single (3D) socket servers (memory + compute + I/O) will be the norm, interconnected optically and with optical integration at the package and chip level.
- FPGAs will be co-located (again, probably using 3D packaging) with dense, storage-class memory (e.g. phase-change memory) and provide a flexible processor-in-memory solution.
- FPGAs in servers are likely to be homogeneous in design (few custom blocks) since they can be packaged at high bandwidth with more custom logic. However, fast on-chip cores may be useful to execute control logic and to provide high speed communications with host processors to implement, for example, memory bus coherency protocols.
- FPGAs will excel at pattern recognition tasks, making them invaluable to analyzing large and ambiguous data sets, including video and sensor data. They will likely play an important role in implementing neuromorphic computers and other dynamic learning-based systems.

Integration – communications and networking

- The drivers will be principally price-performance and low latency.
- FPGAs will handle very large volumes of traffic in routing and switching applications, perhaps making ASICs and custom network processors obsolete (depending on how far we can continue to push network bandwidth)
- FPGAs will provide higher level functions at wire speed such as cyber-security defense and programmable mediations (e.g. filters, transforms, cryptography) for intra-enterprise and cross-enterprise communications. Wire speed functions will be programmed in a high level language (see next section).
- FPGAs will be commonly employed in sensor networks and in sensor devices to provide first level analysis and filtering of massive real-world streaming data. In these applications, SoC integration with general-purpose cores may be needed (depending on the device sophistication) to reduce device cost and power consumption.

Programmability – explicit

- HDL design will be as common in 2032 as assembly language programming is in 2012
- Multiple programming languages will be effective for programming FPGAs, with no need to understand the underlying hardware or even to know it is present
  - Existing languages, and new ones that don't yet exist, will incorporate standard extensions for parallelism allowing the programmer to build algorithms which can be effectively mapped to FPGA
  - Optimizing compilers producing HDL code or even lower level FPGA configurations will be mature by 2032, allowing the programmer to confidently ignore low level details
  - It will continue to be a challenge to achieve portability of FPGA configurations (not HDL) across vendors and even across multiple FPGA parts from the same vendor. "Fat binaries" will be needed unless such portability can be achieved (e.g. through some kind of template architecture)
- Large libraries of portable, optimized, FPGA-accelerated functions will be widely available and usable from multiple languages
  - Linear algebra, cryptography, data compression, data transforms, statistical analyses, string handling, etc.
  - Portability either by hand coding or, more likely, by using one of the languages capable of explicitly encoding device-independent parallelism

Programmability - implicit

- Compilers will be capable of extracting parallelism automatically from existing code, targeting both conventional multiprocessors and FPGAs
  - Automatic parallelization will continue to be challenging for classical imperative languages but will be much more powerful for functional and concurrent languages, which will increase in popularity in the next 20 years
  - Compilers will be able to parallelize and target FPGAs both statically and dynamically (concurrent with program execution). Even binary translation systems will be able to automatically exploit FPGA, accelerating legacy software execution. Dynamic systems will be hindered by continued challenges in high speed synthesis so should benefit from template-based architectures.
  - Runtime systems will be advanced enough to make good decisions about the mapping of code to different system components, accounting for communications and data movement overhead
- Commonly employed "middleware" (higher level programming models and runtime systems) will be designed to exploit FPGA "out of the box"
  - Programming models with substantial implied parallelism such as rule-based systems, process flows, streaming data, relational query, map-reduce, and web services will all be implemented with runtime systems and compilers which automatically exploit FPGA

Programmability - debugging

- Debugging at a high level language source level will improve but continue to be a challenge in 2032
  - Advances in both simulation and chip-level monitoring will enable incremental improvements in HDL-level and signal-level debugging
  - However, mapping of generated FPGA configurations to the source level will be a large challenge, making the visualization of execution in a high level language debugger incomplete and error-prone.
  - The history of failures in debuggers for optimized code from conventional compilers is a good guide to the future with FPGAs programmed at a high level
- Performance analysis will be solved in part by the ability to automatically construct and execute accurate models
  - Components of the application which have been selected by the compiler or programmer to execute on FPGA will be simulated in a performance-accurate manner and execute at full speed on general-purpose parallel systems (developer laptops will have hundreds of cores available in this timeframe, assuming they aren't all working in the cloud by this time)
  - Performance testing and analysis across multiple, diverse devices can be achieved
  - Feedback from performance models will provide additional information for compilers when used in an iterative, feedback-directed fashion

Virtualization

- FPGAs will be commonly employed in cloud infrastructure
  - Dependent on high level, device-independent programming models becoming prevalent
  - Dependent on low cost integration
- FPGA resources will be dynamically sharable across multiple virtual machines (in IaaS) and tenants (in SaaS)
  - FPGAs will need to support fast & flexible partial reconfiguration
  - "system level" FPGA resources must be protected from "user level" logic running either on the host processors or on the FPGA - akin to supervisor mode on general purpose processors – i.e. it is never possible to "crash" the system components by loading new user logic
  - Fast swapping of state must be enabled, essentially providing virtual memory management for "user level" FPGA logic
  - It must be possible to map logic efficiently across multiple FPGAs, integrated closely (e.g. in a 3D package), to avoid inefficiencies in mapping resources to dynamically changing workload
  - "user level" resources must be isolated from other, potentially malicious logic running on the same chip or cluster – akin to process-level isolation in current operating systems
  - FPGA management systems ("hypervisor" for reconfigurable logic) must implement a set of policies designed to ensure fairness, reliable execution and quality of service optimization. It must not be possible for specific users to monopolize resources either intentionally (DoS) or unintentionally.
  - FPGA hypervisors must support both batch and time-sliced scheduling. Hardware support may be warranted for time-slicing.