

The Future of Reconfigurable Systems

Scott Hauck

Department of Electrical and Computer Engineering

Northwestern University

Evanston, IL 60208-3118 USA

hauck@ece.nwu.edu

Abstract

In the decade since FPGAs were invented they have created many new opportunities. Perhaps the most exciting is reconfigurable computing. In this paper we present a vision of the next decade of reconfigurable computing. FPGA architectures will support millions of gates on a chip, requiring the integration of FPGAs with CPUs, DSPs, and SRAMs. Reconfigurable computing specific architectural features will be developed, though their place in the commercial marketplace must be carefully considered. CAD support for reconfigurable system, and commercial applications of reconfigurable computing, will be major keys to deploying reconfigurable computing devices into the mainstream. Finally, we can consider applying the lessons of Run-Time-Reconfiguration within ASIC designs. Hopefully, these technologies can help create a commercial market for reconfigurable computing, making it a truly general-purpose paradigm.

Introduction

Reconfigurable computing, the harnessing of FPGA logic for computation tasks (also often called “Custom” or “Adaptive” Computing) has demonstrated significant potential for the acceleration of general-purpose computing [Hauck98b, Hauck98c, Smith97a, Smith97b, Villasenor97, Vuillemin96]. Unfortunately, this is where it currently remains: a potential computing paradigm. In order to create a truly viable reconfigurable computing market it is necessary to improve the hardware, software, and applications of this technology. In this paper we explore the issues to be faced in the second decade of reconfigurable computing, and point to promising avenues of future work.

Multi-million gate FPGA Architectures

As with all of electronics and high-performance computing, the most critical force in the development of new systems is the ever-increasing performance and logic capacities of new fabrication technologies. For FPGAs, this is both an opportunity and a danger. Obviously, as the process technologies improves, FPGAs become faster, higher capacity, and cheaper per gate for the end-user. This allows FPGAs to take over portions of the design space formerly restricted to Mask-Programmable Gate Array (MPGA) implementations. However, other technologies gain from the underlying process technology as well. That means that while FPGAs eat into the MPGA market, the low-end FPGA market itself is being eroded by PLDs and embedded processors. This is particularly important because, for most circuit designs, if a PLD or embedded processor can handle the same problem as an FPGA, the user will choose not to use FPGAs for this task because of cost and design complexity issues. Thus, process technology improvements demand that FPGAs continue to get larger and faster. Unfortunately for the FPGA manufacturers (though fortunate for the FPGA research community!) these higher density processes present numerous challenges to FPGA design, and may significantly change the architectures of future devices.

Currently, FPGAs are capable of supporting approximately 100,000 gate designs on a chip [Xilinx96, Roelandts97]. Based on the National Technology Roadmap for Semiconductors [Semiconductor94], we can conservatively expect a capacity of almost 4 million gates on a single FPGA by 2010. Harnessing this capacity effectively is a major challenge, and it is clear that current FPGA architectures cannot simply be tiled ever larger and still achieve reasonable quality. The most obvious problem is the one that has always plagued FPGAs: the routing interconnect on the FPGA is the bottleneck. As the capacity of an FPGA increases, Rent's rule demands that the interconnect increase as well. However, maintaining the flat routing topology of commercial FPGAs may not be practical. Even worse, the passive interconnect on some devices clearly will not suffice for these future, high-capacity devices [Rose97], where the quadratic growth of the routing delays will demand the insertion of some active buffering to the interconnect. It is likely that the solution to these problems will be multiple, separate regions of FPGA logic similar to current architectures, connected by a higher-level interconnect structure. Thus, subcircuits can be mapped to the individual logic arrays, and the interconnect between these devices will be handled by a faster, longer-distance communication structure. One good candidate for such an interconnect structure is the HFPGA work on hierarchical interconnects [Aggarwal94, Chan96]. Note that such a move towards a multi-array structure within an FPGA is

particularly ironic given the apparent move in the reconfigurable computing community away from multi-FPGA systems to single chip systems, since the fixed interconnect and related partitioning requirements of multi-FPGA systems will now be found within the FPGA devices themselves.

Supporting multi-million gate designs in a field-programmable device is more than just providing vast arrays of random logic. If one looks at the composition of multi-million gate designs, one quickly realizes that these systems are not simply unstructured computations. Typically, these systems are dominated by datapaths with standard arithmetic and logic computations, sequencing logic to step through complex sequential operations, and large memory requirements. In current circuits this demands the inclusion of microprocessor, DSP, and SRAM resources. In future reconfigurable circuit implementations there is no reason to believe that these requirements will radically change. While these standard computations could be mapped into FPGA logic, the significant performance and density penalties for doing so rule out such an approach for most systems. Microprocessors, DSPs, and SRAM arrays are highly optimized structures, and for computations that fit within their target model they are radically more efficient than an FPGA implementation. For example, a standard estimate is that custom logic (such as processor and memory resources) is an order of magnitude faster, and up to two orders of magnitude smaller, than an equivalent FPGA implementation.

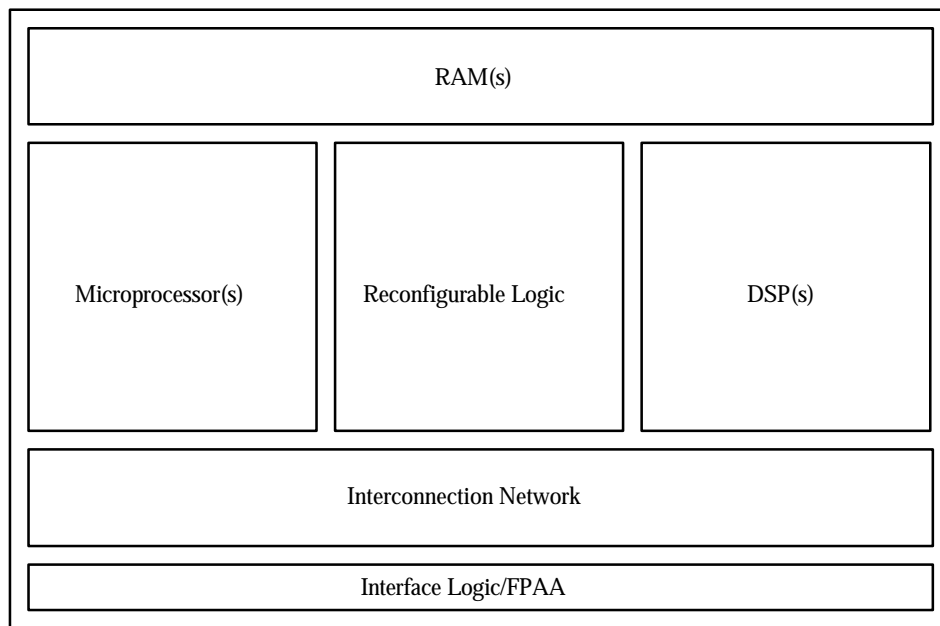


Figure 1. Schematic of a future Field Programmable System-on-a-Chip (FPSC). Microprocessor, FPGA, and DSP resources are provided for computation, RAMs are included for storage, and special-purpose interface logic or Field Programmable Analog Arrays handle I/O communication (note that Keutzer's FUP-SOC is similar [Keutzer97]).

It seems obvious that as FPGAs reach and surpass capacities of a million gates, the circuits they need to support will demand that standard components be integrated into the FPGA die itself. I believe that we will see microprocessors, DSPs, and SRAMs imbedded into the chip, building a hybrid reconfigurable device that provides the benefits of all of these resources in a single package. However, these future Field Programmable Systems-on-a-Chip (FPSC) present challenges both to the chip architectures, as well as the companies that sell them.

While I believe it is clear that the future of reconfigurable systems involves the merging of FPGA, DSP, and microprocessor resources, it is not nearly as clear which companies will be delivering these devices. Microprocessor manufacturers seem uninterested in the FPGA market, with Intel abandoning its early FPGA efforts. Major FPGA vendors do not seem to have the resources for competing in the microprocessor or DSP marketplace, though this may change with time. One possibility is for companies such as National Semiconductor and Motorola, currently minor players in the FPGA marketplace but with expertise in many other semiconductor design areas, to exploit this new opportunity. In fact, these companies are already experimenting with the merging of reconfigurable logic with more traditional computing paradigms [Rupp98].

Regardless of what company delivers these future products, a major issue in such systems will be the heterogeneity of these chips. While most multi-million gate designs need SRAM and processor resources, the amount of these resources required can vary widely between designs. Some applications may require multiple DSP processors on a single chip and relatively little SRAM, while others may be memory and microprocessor dominated. Also, the appropriate FPGA architectures may differ for each application, with some efficiently fitting a coarse-grained architecture, while others requiring more fine-grained elements. Given the range of resource mixes, it is easy to just abandon the concept as unworkable, since no company can afford to custom design the range of chips required for every type of application, and some applications may not have a large enough customer base to justify their own architectures.

There are two solutions to this problem of excessive heterogeneity. The first is to combine similar customer bases together in a single chip, with the superset of the required resources included in the device. This is attractive if the resources not shared between user groups represents a negligible portion of the chip area. For example, the configuration management hardware required to support Run-Time Reconfiguration may be small enough to be merged into standard general-purpose FPGAs, allowing companies such as Xilinx to support reconfigurable computing within their main-line FPGA architectures, and thus not have to support two independent FPGA families.

An alternative method for coping with heterogeneity is to develop common component architectures that can be easily combined to create different resource mixes. As mentioned earlier, I believe that future FPGAs will need to go to a hierarchical routing structure, with local FPGA-like arrays interconnected with a higher-level communication medium. However, other types of resources (DSP, CPU, SRAM) could also be designed to interface to this inter-array routing structure, allowing some FPGA arrays to be replaced with these components. Such a common structure gives rise to an FPGA family [Betz95], with different members having different resource mixes, just as the tileability of current FPGA architectures allows for an FPGA family with differing logic capacities. While fabrication NREs and other concerns will still limit the number of family members, such a methodology allows for the heterogeneity I believe will be essential in future FPGAs without requiring an excessive design effort. However, as always the major problem with all new architectures is developing the software mapping tools to efficiently support the architecture, and a heterogeneous architecture opens many challenges to CAD for FPGAs.

FPGA Architectures for Reconfigurable Computing

Perhaps the domain for FPGAs that should see the most change in the future is that of FPGAs for reconfigurable computing [Hauck98b, Hauck98c, Smith97a, Smith97b, Villasenor97, Vuillemin96]. Not only must these systems evolve in the face of future semiconductor processes, but they also are starting from current devices that are not well optimized for their tasks. Reconfigurable computing today is largely based upon general-purpose FPGAs, architectures that are inefficient for the demands of this domain.

One of the most important features of reconfigurable computing, and one that is not well supported in current architectures, is the need for Run-Time-Reconfiguration. In many FPGA-based computations it is necessary to change the configuration of the device rapidly in order to compute different portions of the overall algorithm. However, changing the configuration of an FPGA for a new operation can require more than a megabit of data. Obviously, this is a much greater demand than that found in standard processors, where only a few bytes of data must be loaded at a time, yet in reconfigurable computing even the concept of adding a cache to an FPGA has not been adequately investigated (for example, an efficient cache replacement policy for FPGA configurations has yet to be developed). This is especially important since some current applications of reconfigurable computing waste from 25% [Withlin96] to over 70% [Wirthlin95, Villasenor96] of their cycles stalled while the FPGAs are reconfigured.

Future FPGA-based computers will need integrated hardware and software systems for configuration management. Multiple levels of caches will efficiently store the data, parallel communication buses will move the data from level to level, and run-time support will be evolved to ensure that the proper data is available as quickly as possible. Initial work has already been done on several techniques for configuration management: Multi-context FPGAs (a form of on-chip cache for configurations) to maintain data in the FPGA [Bolotski94, Trimberger97]; Compression [Dehon96, Hauck98d, Hauck98e] and partial Run-Time Reconfiguration [Lysaght94a, Churcher95, Hutchings95, Wirthlin95, Rupp98] techniques to reduce the amount of data that must be transferred; Prefetching algorithms to preload the next needed configuration and thus overlap computation with reconfiguration [Hauck98a]; Parallel configuration buses to increase reconfiguration bandwidth [Churcher95]; While these techniques form a promising initial set of optimizations for the reconfiguration problem, there is still much left to do.

While reducing reconfiguration times is a priority for reconfigurable computing, other considerations also require changes to the chip architecture. For example, standard FPGAs can have “forbidden configurations” [Hauck97], settings of programming bits that destroy the device from short-circuits. For standard applications this is a reasonable design decision, since the mapping tools can ensure that such states are never entered. In reconfigurable computing devices, which may involve multiple rapidly changing configurations, the danger of damage to the device from intermediate partial configurations requires that there be no such forbidden configurations. However, creating an FPGA without forbidden configurations requires an overhaul of the entire routing structure.

When we use an FPGA as a computing resource, paging computations into the FPGA, we see the need to fit the next circuit element into the free space on the device instead of having a fixed placement. However, to achieve this we need to have relocatable mappings, and the architecture must be symmetric (i.e. the resources in the FPGA are the same across the entire FPGA, though some local heterogeneity is possible). Keeping this symmetry throughout the array is difficult because of the need to have I/O pins, carry chains, and other resources. One likely solution is to require symmetry in only one dimension [Ebeling96, Hauck97, Schmit97]. Mappings might take up the entire width of the device, but only a subset of the rows. In such an architecture the individual cells in a row can be different, while each row is the same as all others. Note that this approach will also simplify the run-time system, since we will not need to consider two degrees of freedom (X and Y position) when deciding where to load the next mapping.

Providing for relocatable mappings requires not just symmetry in the architecture, but also a method to distribute I/O signals to arbitrary points in the array. The most promising solution is to have an internal distribution bus to handle communications with the logic [Churher95, Hauck97, Schmit97]. With such a bus, and some way to specify the row that the target logic occupies, signals can be delivered to the logic regardless of where it is mapped. Note that an architecture which can relocate mappings in only one direction (i.e. mappings occupy complete rows, though rows can be placed into any contiguous section of the array) can simplify this bus structure, since the logic using a signal will always remain in the same column.

The final complexity is in the granularity of the architecture. Current general-purpose FPGAs have logic blocks ranging in flexibility from 2-input function generators to multiple 4-input LUTs. While such an architecture can support random logic well, standard datapath elements such as multiplication and barrel shift operations, as well as more general floating-point computations, cannot be efficiently computed. To deal with this many groups have considered more coarse-grained structures, where the basic logic elements are multi-bit ALUs [Ebeling96, Mirsky96, Moritz98]. These structures can support more complex operations, but may give up the ability to compute non-standard functions, which can be a major strength of reconfigurable computing. What the proper grain size is for these systems is unclear at the moment, and will require significant study. One potential solution is a “multi-grain” architecture, either a standard LUT-based FPGA with some distributed coarse-grained units, or an ALU-based architecture with some LUT computations added into the datapath. Alternatively, a multi-million gate FPGA could be constructed with logic arrays of differing granularities, with FPGA-like regions for complete configurability, and ALU-based regions for high-performance datapath computations.

While it is clear that the optimum architecture for reconfigurable computing will have many features not found in standard FPGAs, there currently is not a large enough market to support a custom architecture (in fact, it can be argued that currently there is NO market for such devices outside of research organizations, though there is hope that this will quickly change). For example, while Xilinx has produced an FPGA architecture for reconfigurable computing (the Xilinx 6200 series [Churher95]) which has many desirable features, it is unlikely that there will be new reconfigurable computing-only architectures from Xilinx in the near term. Thus, the reconfigurable computing community can use standard FPGAs, and make due with clearly sub-optimal devices, or create research prototype devices, which cannot keep pace with commercial development. The most promising alternative is to create optimizations for reconfigurable computing that are either small enough to fit into standard architectures with minimal area and performance overheads, or perhaps even demonstrate capabilities which are also useful for more mainstream FPGA users. In this way, commercial FPGA developers can be convinced to add reconfigurable computing features to general-purpose architectures, serving the research community with mainstream commercial devices. Of course, the ultimate solution is to create a reconfigurable computing market lucrative enough to justify custom architectures, though doing so requires advances in the software and applications of these systems.

CAD Support for Reconfigurable Systems

Perhaps the most challenging problem for reconfigurable systems, and one that is the least well supported, is the software needed to create the FPGA realizations. Like a compiler for a general-purpose processor, CAD software for a reconfigurable system is responsible for efficiently harnessing the resources to solve the user’s problems.

The level of support appropriate for reconfigurable computing is an area of intense debate. One option is to require the user to make almost all mapping decisions, from circuit design to exact placement and routing of logic components. This is in fact the method that has been used for most successful applications of reconfigurable logic. Alternatively, we can seek to raise the level of abstraction, developing software techniques that can map from high-level languages directly to the FPGA resources.

My personal belief is that if reconfigurable computing is to be successful it must create a methodology to automatically map from standard programming languages to the hardware system. Just as general-purpose computing has moved away from assembly language programming, and large ASIC designs are increasingly dependent on hardware description languages, future users of reconfigurable computing will either insist on such a mapping tool suite, or will abandon this technique. The complexities of the hand mapping process, and the required hardware and software sophistication of the designer, are too much for the typical user of these systems to handle.

Creating this mapping toolsuite is a very challenging problem, and one that is only beginning to be adequately addressed [Hauck98c]. These tools must be able to map to future multi-million gate devices quickly and efficiently, making use of the flexibility of the hardware substrate, supporting run-time-reconfiguration, and potentially allowing run-time compilation for problem-specific circuitry [Smith97a]. Some of these tasks are similar to those faced in ASIC designs: algorithms specified in a high-level language must be transformed into a circuit implementation. Just as important, hardware/software codesign issues must be solved, determining which portions of an overall algorithm should be mapped into the reconfigurable logic, and what can be retained on a general-purpose processor.

There are two major concerns for reconfigurable computing that are not found in standard ASIC designs: Run-Time-Reconfiguration, and fast mapping tools. In order to best make use of the limited logic capacity of FPGAs (even future multi-million gate FPGAs may be too small to handle complete applications) the circuitry must be split into separate configurations and paged into the FPGA as needed. In order to efficiently perform this splitting it is necessary to group together functions that will be used in sequence in order to reduce the amount of reconfiguration necessary, while also grouping together logic that can share subcircuits in order to reduce logic size. Although scheduling and allocation heuristics from logic synthesis might be applicable to this problem, the unique constraints of this domain will require alterations to these base techniques, or perhaps even completely new approaches.

A second major concern is the time it takes to compile an application. For ASIC designs, where it takes multiple weeks to have a circuit fabricated, it is okay if the mapping tools take hours to days to complete. In reconfigurable computing, where the system is ready to go immediately after the circuit is generated, these mapping times become a significant bottleneck. Perhaps the most dramatic demonstration of this need is in the area of problem-specific circuits. FPGA-based implementations can be highly specialized to the needs of a given problem, or even a problem invocation, and the greater the amount of information given to the mapping tools on the data being operated on the better the resulting circuitry. For example, researchers at Princeton and NEC have demonstrated a highly efficient solution to the Boolean Satisfiability problem by converting the exact Boolean function to be operated on into a circuit implementation [Zhong98]. While this approach achieves significant performance improvements during operation, it requires that a completely new circuit be generated for each input dataset. Thus, unless the mapping tools can create this circuit quickly, the speedups in operation will be overwhelmed by the circuit creation times. Reconfigurable computing in general achieves significant benefits from such data-specific optimizations, and run-time circuit generation will become an ever more important component to reconfigurable systems. Unfortunately, this issue has not yet been adequately addressed, though efforts are ongoing. This will require a complete reexamination of the complete mapping process, and all algorithms adopted for each of the stages must be mapping-time efficient, even if this requires some loss in mapping quality.

Applications of Reconfigurable Systems

While hardware and software for reconfigurable computing are important components to this new computing paradigm, these efforts are futile unless compelling applications of this technology can be developed. While some impressive research applications have been developed [Hauck98b, Villasenor96, Vuillemin96], there has yet to be any major commercial success for this technology. However, there is some hope that this may soon change. It is clear that image processing tasks can very efficiently be supported in FPGAs. With the development of reasonably cheap and available PCI boards for standard PCs [Annapolis97, VCC97], and the demonstration of how these systems can easily be used with existing software via plug-in mechanisms [Singh98], it is now possible to provide a useable, useful system to the end-user. My hope is that the research community will step forward and develop a set of plug-ins for other standard programs, such as DES encryption for email and data compression/decompression for

Netscape and Internet news programs. In so doing the benefits of reconfigurable computing can be delivered to the end-user, helping to bootstrap reconfigurable computing commercial viability. Whether this happens or not is yet to be seen, but it is clear that without such a cheap, accessible demonstration of the power of reconfigurable computing it is unlikely this technology will succeed.

FPGAs vs. Reconfigurable Systems

In most of this paper we have concentrated on FPGA technology, virtually assuming that reconfigurable computing and reconfigurable systems are synonymous with FPGAs. However, this is obviously not the case, and I believe there are very compelling arguments for investigating other technologies. To place this in context, I put forward the following assertion:

The Law of FPGAs vs. ASICs: For any FPGA-based application, there is an ASIC implementation of the system that is AT LEAST as fast, dense, power-efficient, and cheap in (very) high volumes as the FPGA-based solution.

The proof of this assertion is simple: For the ASIC solution we start with an ASIC implementation of the FPGA used in the application, since an FPGA architecture is simply just another ASIC. We can then remove those features of the array that are not needed in the application, potentially increasing the density, power-efficiency, and speed. Thus, for an FPGA-based implementation to be the best solution, other optimization goals (such as cost at low/medium volumes, time-to-market, ease of use) must be important to the application.

This argument can be viewed in two ways: The most obvious is that this puts a limit on the applications that should be considered for FPGA-based implementations, and the reconfigurable computing community has as a whole done a good job of exploring this issue. The converse however, that Run-Time Reconfiguration (RTR) and other reconfigurable computing techniques can be applied to ASIC implementations has, as far as I know, not been considered at all. This observation opens up a compelling new research area: determining how to harness RTR on ASIC. Techniques such as custom generating Systems-on-a-Chip with hardware, software, and Run-Time Reconfigurable components may be able to radically improve power and area efficiency of many future systems. Alternatively, one can consider starting with a general-purpose FPGA (or FPGA generator) and developing techniques for reoptimizing this substrate for the applications for which it will be used, throwing away useless features in the system.

Conclusions

Although reconfigurable computing has been an active research area for at least a decade, the technology is still in its infancy. In the years to come I expect we will see radical changes in all aspects of this design space. FPGA architectures must cope with ever more powerful fabrication technologies, altering the interconnect and resource mixes to best support multi-million gate designs. Reconfigurable computing places its own demands on the chip architectures, but the community must find a way to make their desired features cost-effective for the commercial chip vendors, either by making them valuable (or at worst an insignificant cost) to be inserted into general-purpose architectures, or by growing a commercial market for reconfigurable computing-specific chips. Finally, the techniques of reconfigurable computing need to consider devices beyond just FPGAs, including future ALU-based reconfigurable devices, as well as custom ASICs designed with Run-Time-Reconfigurable features embedded inside them.

References

- [Aggarwal94] A. A. Aggarwal, D. M. Lewis, "Routing Architectures for Hierarchical Field Programmable Gate Arrays", *International Conference on Computer Design*, pp. 475-478, 1994.
- [Annapolis97] "Firefly", Annapolis, MD: Annapolis Micro Systems, 1997.
- [Betz95] V. Betz, J. Rose, "Using Architectural "Families" to Increase FPGA Speed and Density", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 10-16, 1995.
- [Bolotski94] M. Bolotski, A. DeHon, T. F. Knight, "Unifying FPGAs and SIMD Arrays", *ACM/SIGDA Workshop on Field Programmable Gate Arrays*, 1994.
- [Chan96] V. C. Chan, D. M. Lewis, "Area-Speed Tradeoffs for Hierarchical Field-Programmable Gate Arrays", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 51-57, 1996.
- [Churcher95] S. Churcher, T. Kean, B. Wilkie, "The XC6200 FastMapTM Processor Interface", *Field-Programmable Logic and Applications*, pp. 36-43, 1995.
- [DeHon96] A. DeHon, "Entropy, Counting, and Programmable Interconnect", *ACM/SIGDA Symposium on Field Programmable Gate Arrays*, pp.73-79, 1996.

- [Ebeling96] C. Ebeling, D. C. Green, P. Franklin, "RaPiD – Reconfigurable Pipelined Datapath", *International Workshop on Field-Programmable Logic and Applications*, pp. 126-135, 1996.
- [Hauck97] S. Hauck, T. W. Fry, M. M. Hosler, J. P. Kao, "The Chimaera Reconfigurable Functional Unit", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 87-96, 1997.
- [Hauck98a] S. Hauck, "Configuration Prefetch for Single Context Reconfigurable Coprocessors", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 65-74, 1998.
- [Hauck98b] S. Hauck, "The Roles of FPGAs in Reprogrammable Systems", *Proceedings of the IEEE*, Vol. 86, No. 4, pp. 615-639, April 1998.
- [Hauck98c] S. Hauck, A. Agarwal, "Software Technologies for Reconfigurable Systems", submitted to *IEEE Transactions on VLSI Systems*, 1998.
- [Hauck98d] S. Hauck, Z. Li, "Don't Care Discovery for FPGA Configuration Compression", submitted to *International Conference on Computer-Aided Design*, 1998.
- [Hauck98e] S. Hauck, Z. Li, E. Schwabe, "Configuration Compression for the Xilinx XC6200 FPGA", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1998.
- [Hutchings95] B. L. Hutchings, M. J. Wirthlin, "Implementation Approaches for Reconfigurable Logic Applications", in W. Moore, W. Luk, Eds., *Lecture Notes in Computer Science 975 - Field-Programmable Logic and Applications*, London: Springer, pp. 419-428, 1995.
- [Keutzer97] K. Keutzer, "Challenges in CAD for the One Million Gate FPGA", *ACM/SIGDA Symposium on Field Programmable Gate Arrays*, pp. 133-134, 1997.
- [Lysaght94a] P. Lysaght, J. Dunlop, "Dynamic Reconfiguration of FPGAs", in W. R. Moore, W. Luk, Eds., *More FPGAs*, Oxford, England: Abingdon EE&CS Books, pp. 82-94, 1994.
- [Mirsky96] E. Mirsky, A. DeHon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 157-166, 1996.
- [Moritz98] C. A. Moritz, D. Yeung, A. Agarwal, "Exploring Optimal Cost-Performance Designs for RAW Microprocessors", *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1998.
- [Roelandts97] W. Roelandts, *The Future of FPGAs*, San Jose, CA: Xilinx, Inc., 1997.
- [Rose97] J. Rose, D. Hill, "Architectural and Physical Design Challenges for One-Million Gate FPGAs and Beyond", *ACM/SIGDA Symposium on Field Programmable Gate Arrays*, pp. 129-132, 1997.
- [Rupp98] C. Rupp, M. Landguth, T. Gaverick, E. Gomersall, M. Holt, J. Arnold, M. Gokhale, "The NAPA Adaptive Processing Architecture", *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1998.
- [Schmit97] H. Schmit, "Incremental Reconfiguration for Pipelined Applications", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp.47-55, 1997.
- [Semicon94] *The National Technology Roadmap for Semiconductors*, Semiconductor Industry Association, 1994.
- [Singh98] S. Singh, R. Sloys, "Accelerating Adobe Photoshop Using the XC6200 FPGA", *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1998.
- [Smith97a] W. H. Mangione-Smith, B. L. Hutchings, "Configurable Computing: The Road Ahead", *Reconfigurable Architectures Workshop*, pp. 81-96, 1997.
- [Smith97b] W. H. Mangione-Smith, B. Hutchings, D. Andrews, A. DeHon, C. Ebeling, R. Hartenstein, O. Mencer, J. Morris, K. Palem, V. K. Prasanna, H. A. E. Spaanenburg, "Seeking Solutions in Configurable Computing", *Computer*, Vol. 30, No. 12, pp. 38-43, December 1997.
- [Trimberger97] S. Trimberger, D. Carberry, A. Johnson, J. Wong, "A Time-Multiplexed FPGA", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 22-28, 1997.
- [VCC97] "H.O.T.Works™ The Hardware Object Technology™ Development System", Reseda, CA: Virtual Computer Corporation, 1997.
- [Villasenor96] J. Villasenor, B. Schoner, K.-N. Chia, C. Zapata, H. J. Kim, C. Jones, S. Lansing, B. Mangione-Smith, "Configurable Computing Solutions for Automatic Target Recognition", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 70-79, 1996.
- [Villasenor97] J. Villasenor, W. H. Mangione-Smith, "Configurable Computing", *Scientific American*, pp. 66-71, June 1997.
- [Vuillemin96] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, P. Boucard, "Programmable Active Memories: Reconfigurable Systems Come of Age", *IEEE Transactions on VLSI Systems*, Vol. 4, No. 1, pp. 56-69, March, 1996.
- [Wirthlin95] M. J. Wirthlin, B. L. Hutchings, "A Dynamic Instruction Set Computer", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 99-107, 1995.
- [Wirthlin95] M. J. Wirthlin, B. L. Hutchings, "A Dynamic Instruction Set Computer", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.
- [Wirthlin96] M. J. Wirthlin, B. L. Hutchings, "Sequencing Run-Time Reconfigured Hardware with Software", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 122-128, 1996.
- [Xilinx96] *The Programmable Logic Data Book*, San Jose, CA: Xilinx, Inc., 1996.
- [Zhong98] P. Zhong, M. Martonosi, P. Ashar, S. Malik, "Accelerating Boolean Satisfiability with Configurable Hardware", *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1998.