# Configurable Computing

Jamil Khatib

October 16, 1998

## 1 Introduction

System performance is the most important factor that designers must consider while designing their systems. As much as dedicated optimized circuits for specific problem are considered, the system performance increases. The maximum performance can be achieved when the circuits are optimized for only single problem. This is OK, but what if a new problem is to be considered even with minor changes to the old one, the whole system must be redesigned and optimized according to it. As a result, the flexibility of changing any parameter of the system is lost. This approach is usually accepted is ASIC designs. General purpose systems can be built to accommodate this problem and to increase system flexibility by means of software. This reduces the system performance because the hardware has no dedicated resources that can be used by specific algorithms. So we need new technique that can be optimized for special circuits while keeping system flexibility. The *configurable computing* can achieve this. Dedicated circuits can be built on FPGAs and these circuits can be easily modified by reprogramming the chips again. *Run-Time Reconfiguration* that is an advanced technique based on configurable computing can increase system performance by using highly optimized circuits that are loaded and unloaded dynamically during the operation of the system. In this way system flexibility is maintained and functional density is increased.

## 2 Configurable systems

### 2.1 Introduction

FPGA systems are used mainly in systems "ASICs" prototyping because of their reconfigurability and the ease and low cost of their development process. They can be used to perform multiple operations on different phases of system operations, for example and FPGA can be configured to perform system diagnostics at power-up [16] then it can be reconfigured to do something else during the system running. FPGA systems are divided into three main categories:

- FPGA Emulators.
- Custom Computing Machines "CCM".
- Run-time reconfigurable "RTR" system.

### 2.2 FPGA Emulators

These systems are built with no prior knowledge of the characteristics of the application that is going to use the system. The system is composed of FPGA chips connected together through *Field Programmable Inter-Connect* chips FPICs [12]. One of the approaches is to mount the FPGAs and FPICs on one MCM substrate. The main advantage of this model is that larger systems can be implemented but it suffers from high cost and low clock performance due to the lack of system optimization and due to the use of programmable interconnections between FPGAs.

### 2.3 Custom Computing Machines "CCM"

The most commonly used FPGA systems are the Custom Computing Machines. These machines are built from and array or matrix of FPGA ICs that are connected directly to each others. These FPGAs are located on a single board "usually computer based boards". These boards have built in memory modules to store the FPGA configurations and the intermediate results. These machines are generally built for specific applications so the inter-connections between FPGAs are optimized to get the best performance while reducing some of the system flexibility [12]. Some of the well know CCMs are the Splash and DecPerle machines. Some commercial CCMs are available from many companies such as the Virtual Computing Company "VCC".

### 2.4 Run-time Reconfigurable "RTR" Logic

Run-time reconfigurable systems are used to get best performance with maximum hardware utilization. The circuits are loaded into the hardware and unloaded from it dynamically during the operation of the system. So the area used by such systems will be less than static systems due to unloading idle circuits. RTR systems can be built on single FPGA chip or on multi-IC systems like CCMs. The flow of operation of RTR systems is different from the traditional flow of the general computing systems, where as the circuits configurations are loaded into the system's memory instead of the traditional instructions. These configurations are loaded and unloaded according to the application needs. Refer to figure 1 for a small comparison between different systems.

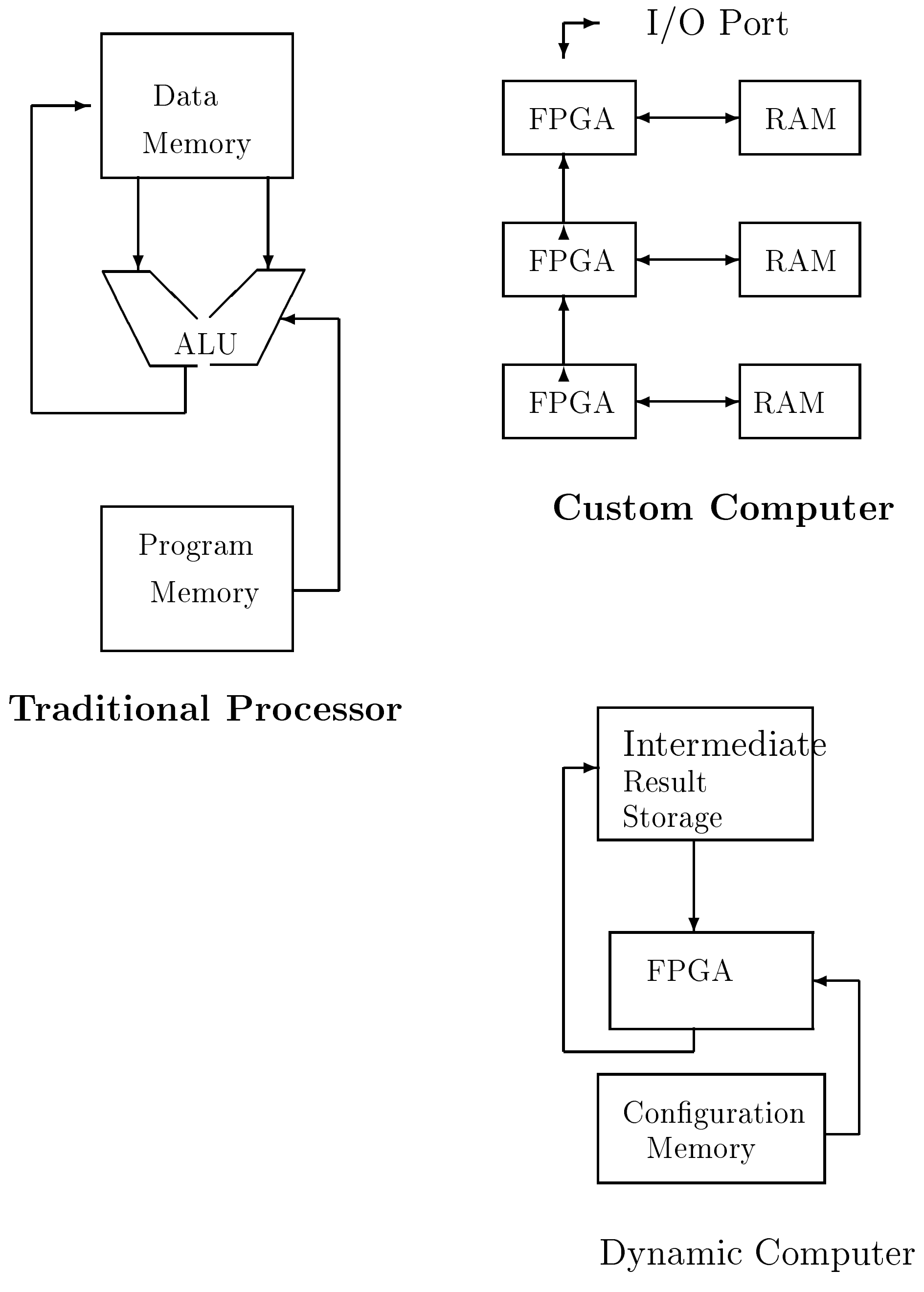


Figure 1: Systems comparison

## 3 Advantages of Configurable Computing

- **Performance:** Configurable computing can be highly optimized for specific data set or specific applications [12].

- **Cost Effectiveness:** Configurable computing can be used to reduce system costs through *hardware reuse* [12]. Any new features or can be easily updated to the system and due to the field programmability, any problems with the design can be corrected without any changes to the on board resources.
- **System prototyping:** Large systems (either ASICs or boards) can be built on a single or multiple FPGAs that makes system testing and debugging more easier than testing the real system and even cheaper.
- **Custom I/O:** FPGAs provides flexible set of programmable I/O signals. That gives the designer the opportunity to reuse existing hardware and to add new changes to it easily [12]. New FPGAs supports Different types of IOs levels and standards. On important use of this feature is matching different families.
- **System density:** Configurable computing "specially Run-time configurable logic" increases the system density and can deliver functionality of the device many times more than its size [13] by dynamically reconfiguring the system and increase resource utilization through loading and unloading different system modules.
- **Fault-Tolerance:** System reliability can be increased through redundancy by duplication or building some testing circuits. Dynamic reconfiguration technique can build destroy old circuits and build new ones after detecting any error.

# 4 Partial reconfiguration

There are two main implementation approaches for RTR applications, total or partial reconfiguration. In the first method, all FPGA resources are reconfigured or deleted between different configurations. While in the partial reconfiguration approach only the differences between the configurations are modified [8].

## 4.1 Total vs. Partial reconfiguration

One of the features of the partially reconfigured FPGAs, is that the size of the bit-stream is proportional to the size of the reconfigured resources. This means that as the size of the bit-stream decreases, the speed of the reconfiguration increases. As a result, the configuration of the partially reconfigured systems is faster than the total reconfigurable systems.
The algorithm is divided into time-exclusive segments "time independent partitions" in the totally reconfigured systems. While when the partial reconfiguration is used the it is divided into functional groups that can be

loaded or removed independently [10]. The first method of division requires that the partitions must be of equal sizes because each partition will occupy the whole resources of the FPGA. This may increase the difficulty of the partitioning process unless there is well defined partitions in the algorithm as in the RRANN project where the system is divided into three independent logical partitions that are loaded sequentially to the system [7, 16]. This means that the density of the partially reconfigured systems is increased due to the extra utilization of the idle circuitry.
The state of the system and the intermediate results between different configurations must be considered between loading and unloading these configurations. The total reconfiguration scheme uses external memory to save these information. The other scheme needs less or no memory to store these results because most of their driving circuits remain on the hardware.
One of the most advantages of partially reconfigured systems is that the partitions can be located on the FPGA resources during the run-time to provide extra performance and to increase the utilization of the resources that may be reduced due to multiple loading and unloading of hardware modules. The concept or hardware relocation complicates the software design, refer to section 5.3 on page 7.

## 4.2 Partial reconfigurations considerations

Algorithm partitioning is one of the important issues that must be considered while designing an RTR application. In partially reconfigured applications the location of the hardware module "partition" is unknown at the design time so this may cause some problems that may reduce the performance of the system. Surrounding circuitry is not known before placing the modules which increases the difficulty of the placing algorithms must be used at run-time [7, 10].
Inter-communications and interfacing between hardware modules is more problematic in the partially reconfigured systems than in the totally reconfigured ones. The interfacing between circuits is unknown at the design time and also the sub-circuits need to interact to different sub-circuits dynamically.
Special kind of architectures must be introduced to support the partial reconfiguration technique. For example there must be some predictable circuits timing to ensure that the circuits will behave according to the specifications wherever they are located. Also there must an easy interface to the whole internal resources of the FPGA. Atmel, National and Xilinx have there own versions of partial reconfigurable FPGAs. The XC6200 family from xilinx uses the $FastMap^{TM}$ interface, which is a dedicated interface to connect the FPGA directly to the host processor without consuming FPGA resources. This interface maps the FPGA resources into the memory address space of the processor, so the configuration of each individual re-

sourced can be read or written by the host processor. Also the CX6200 has a hierarchal routing method that provides predictable routing delays [1, 2].

### 4.3 Partial reconfiguration methods

To get the best performance of fully dynamic partial reconfigurable systems, the partitions must be so fine grained so as to reduce the configuration over head and to increase the system density. It is difficult to partition the system to such small circuits. Instead semi-static circuits can be used to achieve good performance while maximizing system utilization [8]. By the term *semi-static* I mean that the circuit is almost used as is but only small changes can be applied to it. The following shows some examples of semi-static circuits:

- Constant values: circuits that are optimized to operate for certain values, new constant values can be applied to these circuits without interrupting the whole design.
- Precision: Arithmetic circuits generally have the same structure but they differ with the number of duplicate circuits that yields to different precision.
- Almost any circuits that share the same functionality but have small differences can use this technique.

### 4.4 Partially reconfigured applications

Most of the new researches focus on the partially reconfigured systems. For example the RRANN II project [9, 7, 16], the DISC project [17, 18, 16] and many other applications depends on this concept of reconfiguration, refer to section 6 on page 9

## 5 Reconfigurable computing software tools

### 5.1 Introduction

Considerable efforts have been done toward improving configurable computing CAD and software tools. The main difference between reconfigurable computing tools and general hardware design tools, is that they must manage hardware resources through time as well as space [12, 11].

The general approach for reconfigurable computing software is to use a compiler that manages the hardware resources through software general techniques. The application developer has to describe his algorithm “application” by using a high level software language and then the compiler should do the rest, like hardware partitioning and circuits placement.

## 5.2 Software requirements

These kinds of compilers has to satisfy many conditions so as to be considerable for both CMM and RTR applications.

- It must be a single language describing both hardware and software, so that the project integration can be easier and the whole design flow can be automated.
- It is preferable to use a widely available language and compiler, so that the designers have not to learn new languages and they can take the advantages of the available commercial compilers [4].
- The compiler must have the ability to optimize general operations before mapping them into hardware [15, 3].
- It must support both synthesis and simulation (debugging) with minimum modifications to the code [4].
- The description must be hardware independent and the compiler should map this description into the target hardware.
- the compiler should support both partial reconfiguration and module relocation.

## 5.3 General idea

RTR software is much like the general computing Operating Systems. The OS manages the system memory and schedules the programs in the multitasking environments, as well as RTR software that allocates hardware modules and pages them in and out of the FPGA according to the schedule and the flow of the program. These modules can be cached and relocated in the FPGA to get the optimal performance with minimum area. This is one of the advantages of the partial reconfiguration.

The relocation problem is a commonly known problem faced by the OS designers. They need to manage memory fragments and paging memory modules in and out of the physical memory to increase the size of the memory that can be used by the running applications beyond the available real memory. Module removal from the cache and relocation algorithm used in RTR software are adopted from the OS's algorithms[1], but the only difference between them is that the hardware modules are two dimensional modules while the memory modules are one dimensional, which makes the RTR software more complicated and less efficient [16]. Some solutions were discussed in the DISC project [17, 18].

[1]LRU, FIFO and optimal algorithms

### 5.4 Software Approaches

According to the current researches there are two main approaches:

- The first approach uses predesigned and preoptimized hardware models that are combined with the software models. Software model interacts with the runtime environment and manages the hardware resources by loading and unloading hardware modules from the FPGA according to the algorithm requirements.

  Object Oriented Programming languages are the most suitable languages that can be used for this approach. A hardware module is encapsulated in a single class that inherits the characteristics of its parent (the hardware Object) where most of the common functions are defined like place and route algorithms, load/unload, scheduling and more.

  Load and unload problems take the advantage of the built in constructors and destructors in the Object Oriented Programming style. These functions are called when new objects are created or destroyed. They perform the proper initialization and finalization to the object by allocating and deallocating memory. This is exactly what is needed for the hardware objects when they need to be loaded or removed from the FPGA resources.

  In general, the application developer must divide his algorithm into hardware objects and software control procedures. TSI Telsys implemented this approach using the Java language for their ACECards where the Java run time environment runs the application on either the ACECard's embedded processor, or the host workstation's processor.[6]

  Virtual Computer Corporation "VCC" uses the same idea for its RC systems, but instead of using OO language, it uses the C language and header files to encapsulate the hardware objects. The header files are created by a tool that converts the bit-stream file into C like file [5].

- The PRISM project took different approach, where the C language is used to describe the whole algorithm and the compiler creates two images, software and hardware images form the pure C code. So it does not depend on predesigned hardware modules because it is the responsibility of the compiler to synthesis the description code and create the hardware image for the critical parts of the code (or as declared by the code writer).

  The PRISM project uses the GNU C compiler "gcc" as the parser and preoptimizer compiler, and the VHDL Designer and X-BLOX for code synthesis. The gcc produces RTL list and VHDL representation of the code while the synthesis tool generates the hardware image that is placed and routed by the hardware vendor tools. After all, the software image that runs on the host processor controls the program and hardware images downloading flow. [2]

Several programs have been tested on this system and showed good speed up over the pure software procedures. For more information refer to [15] and [3]

- In contrast, a special HDL dedicated to the reconfigurable systems and CCM can be used to develop reconfigurable computing applications. The JHDL "Just another HDL" from BYU uses the Java language to describe the hardware using built in classes for combinational, sequential circuits and wires. It also supports partial reconfiguration through the use of sockets where hardware modules can be plugged or removed.[4]

### 5.5 Future enhancements

Up to now, the designers must check for the appropriateness of the circuits of reconfigurable logic, so most of the design is done by hand. CAD tools must be designed to do automatic partitioning and and maps the hardware modules taking into considerations the surrounding modules and the proper circuit alignment.
Dynamic circuit generation software is one of the important issues that must be discussed [16]. Another important topic is module relocation and hardware removement algorithms in the partial reconfigurable logic.
I think enhanced place and route algorithms should be developed by hardware implementation either by building the algorithms on the hardware or by creating special designs as in the DISC project [17, 18].

## 6 RTR applications

Although RTR applications provides much enhancements over the static designs, but they are still in the research area. RTR is suitable for many applications specially those need much computations and depend on parallel methods such as DSP applications, image processing, neural networks, encryption, compression, pattern matching, motion and target tracking and even general purpose processors.

- **General computing Processor "DISC"**
  Dynamic Instruction Set Computer is a new general purpose processor

[2]

developed at BYU. The special about this processor is that it can support infinite number of custom instructions with limited hardware resources by building custom circuits for each instruction and loading them into the FPGA.

The hardware of DISC processor is divided into two parts: general "weak" processor core that has small number of instructions and addressing modes, and custom instructions area where the new instructions are loaded. The FPGA hardware space is formed of rows, each custom instruction circuit must occupy integer number of rows to speed up hardware relocation. The communication lines are extended horizontally and reach all rows which ease the communication between each instruction and the processor core without interfering with other instructions.Although the idea of rows has good advantages, it reduces the resources utilization, refer to figure 2.

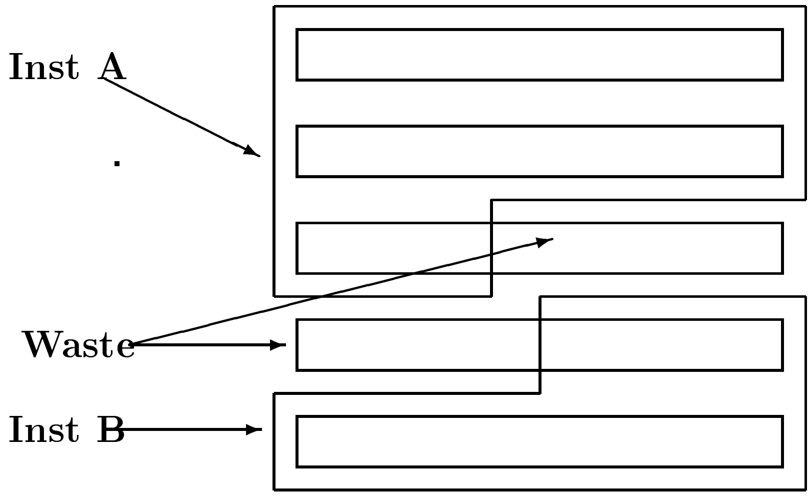


Figure 2: Concept of rows in DISC project

The DISC processor uses the idea of instruction caching to reduce the configuration overhead by preloading instructions. When new instruction is requested, the core checks if it is already loaded,then no actions are needed, else it reconfigures the hardware and loads the new instruction. The efficiency of the caching depends on the flow of the program and the hardware module removal algorithms. For more information about the DISC project refer to [17, 18, 16].

- **Automatic Target Recognition "ATR"**
  The Defense Advanced Research Projects Agency (DARPA) supports some universities (like the University of California, Los Angeles "UCLA")

to build and optimize ATR systems.[13][3]

The challenge to the ATR is the rapid comparison of an input image to thousands of templates with large number of pixels per image [14]. General purpose computing devices are used to increase the flexibility of the system and enables multiple templates to be checked, but with low performance. In contrast, dedicated hardware for special templates can be used to provide higher performance but without the flexibility of changing the templates. RTR logic can be used to combine between the advantages of the both approaches by building custom circuits optimized to each template and reconfigured on the fly to different templates.
Tree adders are used for each pixel to perform the correlation between the input image and the template. To further improve the performance, correlation between the templates can be calculated during the design time to get the common pixels between them to use the same adder tree for the correlated templates' pixels. So it is important to do templates partitioning during the design time and group the correlated pixels in sets that use the same circuits, in this way the use of the hardware is minimized with improved performance while keeping the flexibility of the circuits. For more details refer to [13].

- **Data encryption** The Data Encryption Standard "DES" algorithm showed good performance when implemented by RTR logic. When the receiver gets the key of the sender, the system reconfigure the hardware to the new key and decodes the incoming messages. At the end, the system removes this key waiting for a new key. In this case, we can get maximum performance of the system for any key [14].

## 6.1 Industrial applications

Reconfigurable logic still have limited commercial applications. Image processing and DSP applications are the most dedicated applications to evolve first as plug-in cards to the computers. Also, FPGA Coprocessors can be used in video cards to increase the performance of the system. This is because these applications need bit level calculations with high efficiency and low cost [14, 11]. FPGA systems will not replace microprocessors for general purpose computing tasks in the near future [14], instead it will be good approach to use them as coprocessors for special tasks and they may appear in mother boards of the computers[4]. Stand alone systems still need much more time to be developed.

[3] and DARPA ATR proposal

[4] There is a good discussion about this topic in e-mail from Jan Gray posted to news groups comp.arch.fpga on 6, may 96 with title "On FPGAs as PC coprocessors"

## 6.2 Reconfigurable logic limitations

There are three major factors that limits the evolution of the RTR applications:

- FPGA gate capacity is some how small compared to the ASICs [11]. This increases the complexity of the system because designers have to use more than one FPGA and they have to partition their designs between the on-board FPGAs. Interfacing problem arises when multiple FPGAs are used which increases the over all system delay.
  These days the number of gates per FPGA is increasing, for example the new FPGA family from Xilinx "Virtex" will have 1 million gates per FPGA.

- Configuration speed is the most important factor that limits the number of RTR applications. New FPGA architectures must be developed to reduce the configuration overhead. Several architectures have been proposed such as the DPGA that uses multiple configuration memory syntaxes[5].

- RTR applications need highly skilled developers who can deal with the low level design, so these applications are developed in laboratories under the supervision of qualified designers [12]. The only solution to this problem is to develop CAD tools that can reduce the design time.

- Benchmarks for RTR applications is one of the important issues that must be introduced to enable designers to discuss the performance of their designs versus the performance of other designs techniques. M. Wirthlin suggested the functional density as a performance measure for RTR applications [16]. It is a measure of the execution time with respect to the circuit area taking into consideration the configuration overhead.

Casselman the president of the Virtual Computer Corporation "VCC" wrote in [5] the issues that must be considered in RC applications:

---

5

**The General purpose RC system must:**

- install easily into current computer systems.
- be compatible with existing standards & transparent to the End-User.
- be accessible from within common application program software.
- be programmable by software engineers unfamiliar with digital design.

**Three fundamental issues for the successful insertion of RC into marketplace:**

- How do we get the large software engineering community to grasp, develop and utilize reconfigurable computing in their applications?
- How do we introduce the reconfigurable computing component into the current and future computing environment effectively, from both the technological and users side?
- How do we establish a fundamental foothold in this evolving market without redesigning or producing the entire computing platform?

So his idea is to use the existing computer systems in conjunction with the RC system to get efficient low cost usable system. The application design should be done using general computing software style without going into the details of the underlying architecture.

## 6.3 Future enhancements

Extra features must be added to the FPGAs to increase their usability, such as:

- On-chip memory must be increased.[6]
- Special-purpose blocks such as Arithmetic units [14] are needed to increase FPGA usability in the arithmetic computation instead of converting these operations to bit operations that consumes look-up tables "LUT".
- Memory and micro-processors interfacing support must be included in the FPGA chips to reduce the overhead needed to interface to these devices[7] [11]. Xilinx XC6200 FPGA family has parallel programming facility where each internal cell can be seen by the microprocessor as if it is a memory cell.

[6]Spartan family from Xilinx has good on-chip memory.

[7]Jan Gray discussed this issue in his e-mail to comp.arch.fpga news group

- Enhanced partial reconfigurable FPGA must be built to reduce the configuration time. Xilinx XC6200 family supports the partial reconfiguration with reconfiguration bandwidth 33.3 MB/sec, but this family is optimized for configuration speed and not logic density. Both Atmel and national semiconductors have their own partial reconfigurable FPGA families.

To conclude, reconfigurable logic will become more considerable for real world applications after high speed configuration FPGA architectures are developed with intelligent software that ease both the system use and design.

## References


[1] Introducing the xc6200 fpga architecture. *Xcell*, (18):22–23, 1995.

[2] *XC6200 Field Programmable Gate Arrays data book*, 1997.

[3] P. M. Athanas and H. F. Silverman. Processor reconfiguration through instruction-set metamorphosis. *IEEE Computer*, 26(3):11–18, March 1993.

[4] P. Bellows and B. Hutchings. Jhdl -an hdl for reconfigurable systems.

[5] Steve Casselman. Run-time programming method for reconfigurable-computer a tool for general purpose reconfigurable computing.

[6] D. Davis, M. Barr, T. Bennett, S. Edwards, J. Harris, I. Miller, and C. Schanck. A java decelopment and runtime environment for reconfigurable computing.

[7] J. G. Eldredge and B. L. Hutchings. Density enhancement of a neural network using FPGAs and run-time reconfiguration. In D. A. Buell and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 180–188, Napa, CA, April 1994.

[8] J. D. Hadley and B. L. Hutchings. Design methodologies for partially reconfigured systems. In P. Athanas and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 78–84, Napa, CA, April 1995.

[9] James D. Hadley. The performance enhancement of a run-time reconfigurable FPGA system through parital reconfiguration. Master's thesis, Brigham Young University, Provo, UT, November 1995.

[10] B. L. Hutchings and M. J. Wirthlin. Implementation approaches for reconfigurable logic applications. In W. Moore and W. Luk, editors, *Field-Programmable Logic and Applications*, pages 419–428, Oxford, England, August 1995. Springer.

[11] W. Mangione-Smith, B. Hutchings, D. Andrews, A. DeHone, C. Ebeling, R. Hartenstein, O. Mencer, J. Morris, V. Prasanna, and H. Spaanenburg. Seeking solutions in configurable computing. *IEEE Computer*, pages 38–43, November 1997.

[12] William Mangione-Smith and Brad Hutchings. Configurable computing: The road ahead. In Reiner Hartenstein and Viktor Prasanna, editors, *Reconfigurable Architectures: High Performance by Configware*, Microsystems Engineering Series, pages 81–96, Chicago, 1997. IT Press. Proceedings of the Reconfigurable Architectures Workshop (RAW '97).

[13] J. Villasenor, B. Schoner, K. Chia, and C. Zapata. Configurable computing solutions for automatic target recognition. In J. Arnold and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 70–79, Napa, CA, April 1996.

[14] John Villasenor and William H. Mangione-Smith. Configurable computing. *Scientific American*, pages 66–71, June 1997.

[15] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, and S. Ghosh. PRISM-II compiler and architecture. In D. A. Buell and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 9–16, Napa, California, April 1993.

[16] M. J. Wirthlin. *Improving functional density through run-time circuit reconfiguration.* PhD thesis, Brigham Young University, November 1997.

[17] M. J. Wirthlin and B. L. Hutchings. A dynamic instruction set computer. In P. Athanas and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 99–107, Napa, CA, April 1995.

[18] M. J. Wirthlin and B. L. Hutchings. Sequencing run-time reconfigured hardware with software. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 122–128, Monterey, CA, February 1996.