to appear in *Proceedings of the IEEE*.

The Roles of FPGAs in Reprogrammable Systems

Scott Hauck
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208-3118 USA
hauck@ece.nwu.edu

Abstract

FPGA-based reprogrammable systems are revolutionizing some forms of computation and digital logic. As a logic emulation system they provide orders of magnitude speedup over software simulation. As a custom-computing machine they achieve the highest performance implementation for many types of applications. As a multi-mode system they yield significant hardware savings and provide truly generic hardware.

In this paper we discuss the promise and problems of reprogrammable systems. This includes an overview of the chip and system architectures of reprogrammable systems, as well as the applications of these systems. We also discuss the challenges and opportunities of future reprogrammable systems.

1. Introduction

In the mid 1980s a new technology for implementing digital logic was introduced, the field-programmable gate array (FPGA). These devices could either be viewed as small, slow mask programmable gate arrays (MPGAs) or large, expensive programmable logic devices (PLDs). FPGAs were capable of implementing significantly more logic than PLDs, especially because they could implement multi-level logic, while most PLDs were optimized for two-level logic. Although they did not have the capacity of MPGAs, they also did not have to be custom fabricated, greatly lowering the costs for low-volume parts, and avoiding long fabrication delays. While many of the FPGAs were configured by static RAM cells in the array (SRAM), this was generally viewed as a liability by potential customers who worried over the chip's volatility. Antifuse-based FPGAs also were developed, and for many applications were much more attractive, both because they tended to be smaller and faster due to less programming overhead, and also because there was no volatility to the configuration.

In the late 1980s and early 1990s there was a growing realization that the volatility of SRAM-based FPGAs was not a liability, but was in fact the key to many new types of applications. Since the programming of such an FPGA could be changed by a completely electrical process, much as a standard processor can be configured to run many programs, SRAM-based FPGAs have become the workhorse of many new reprogrammable applications. Some uses of reprogrammability are simple extensions of the standard logic implementation tasks for which the FPGAs were originally designed. An FPGA plus several different configurations stored in ROM could be used for multi-mode hardware, with the functions on the chip changed in reaction to the current demands. Also, boards constructed purely from FPGAs, microcontrollers, and other reprogrammable parts could be truly generic hardware, allowing a single board to be reprogrammed to serve many different applications.

Some of the most exciting new uses of FPGAs move beyond the implementation of digital logic, and instead harness large numbers of FPGAs as a general-purpose computation medium. The circuit mapped onto the FPGAs need not be standard hardware equations, but can even be operations from algorithms and general computations. While these FPGA-based custom-computing machines may not challenge the performance of microprocessors for all applications, for computations of the right form an FPGA-based machine can offer extremely high performance, surpassing any other programmable solution. Although a custom hardware implementation will be able to beat the power of any generic programmable system, and thus there must always be a faster solution than a multi-FPGA system, the fact is that few applications will ever merit the expense of creating application-specific solutions. An FPGA-based computing machine, which can be reprogrammed like a standard workstation, offers the highest realizable performance for many different applications. In a sense it is a hardware supercomputer, surpassing traditional machine architectures for certain applications. This potential has been realized by many

different research machines. The Splash system [Gokhale90] has provided performance on genetic string matching that is almost 200 times greater than all other supercomputer implementations. The DECPeRLe-1 system [Vuillemin96] has demonstrated world-record performance for many other applications, including RSA cryptography.

One of the applications of multi-FPGA systems with the greatest potential is logic emulation. The designers of a custom chip need to verify that the circuit they have designed actually behaves as desired. Software simulation and prototyping have been the traditional solution to this problem. However, as chip designs become more complex, software simulation is only able to test an ever decreasing portion of the chip's computations, and it is quite expensive in time and money to debug by repeated prototype fabrications. The solution is logic emulation, the mapping of the circuit under test onto a multi-FPGA system. Since the logic is implemented in the FPGAs in the system, the emulation can run at near real-time, yielding test cycles several orders of magnitude faster than software simulation, yet with none of the time delays and inflexibility of prototype fabrications. These benefits have led many of the advanced microprocessor manufacturers to include logic emulation in their validation process.

In this paper we discuss the different applications and types of reprogrammable systems. In section 2 we present an overview of FPGA architectures, as well as FPICs. Then, section 3 details what kinds of opportunities these devices provide for new types of systems. We then categorize the types of reprogrammable systems in section 4, including coprocessors and multi-FPGA systems. Section 5 describes in depth the different multi-FPGA systems, highlighting their important features. Finally, sections 6 and 7 conclude with an overview of the status of reprogrammable systems and how they are likely to evolve. Note that this paper is not meant to be a catalog of every existing reprogrammable architecture and application. We instead focus on some of the more important aspects of these systems in order to give an overview of the field.

2. FPGA Technology

One of the most common field-programmable elements is programmable logic devices (PLDs). PLDs concentrate primarily on two-level, sum-of-products implementations of logic functions. They have simple routing structures with predictable delays. Since they are completely prefabricated, they are ready to use in seconds, avoiding long delays for chip fabrication. Field-Programmable Gate Arrays (FPGAs) are also completely prefabricated, but instead of two-level logic they are optimized for multi-level circuits. This allows them to handle much more complex circuits on a single chip, but it often sacrifices the predictable delays of PLDs. Note that FPGAs are sometimes considered another form of PLD, often under the heading Complex Programmable Logic Device (CPLD).

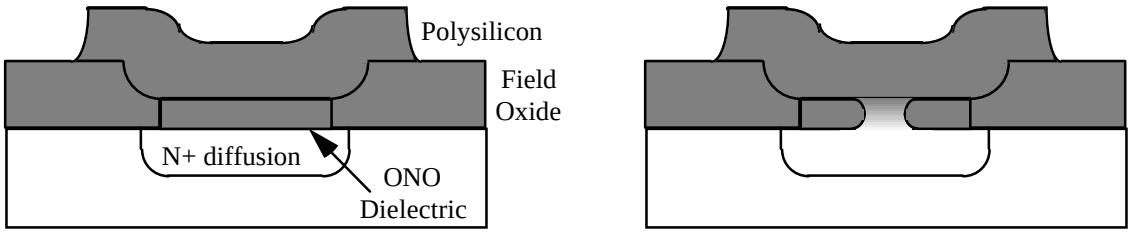

Figure 1. Actel's Programmable Low Impedance Circuit Element (PLICE). As shown at left, an unblown antifuse has an oxide-nitride-oxide (ONO) dielectric preventing current from flowing between diffusion and polysilicon. The antifuse can be blown by applying a 16 Volt pulse across the dielectric. This melts the dielectric, allowing a conducting channel to be formed (right). Current is then free to flow between the diffusion and the polysilicon [Actel94, Greene93].

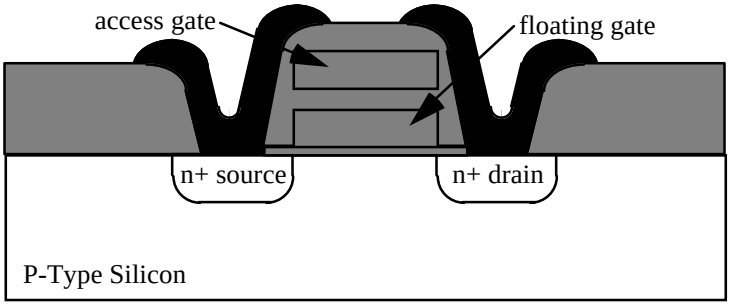

Figure 2. Floating gate structure for EPROM/EEPROM. The floating gate is completely isolated. An unprogrammed transistor, with no charge on the floating gate, operates the same as a normal n-transistor, with the access gate as the transistor's gate. To program the transistor, a high voltage on the access gate plus a lower voltage on the drain accelerates electrons from the source fast enough to travel across the gate oxide insulator to the floating gate. This negative charge then prevents the access gate from closing the source-drain connection during normal operation. To erase, EPROM uses UV light to accelerate electrons off the floating gate, while EEPROM removes electrons by a technique similar to programming, but with the opposite polarity on the access gate [Altera93, Wakerly94].

Just as in PLDs, FPGAs are completely prefabricated, and contain special features for customization. These configuration points are normally either SRAM cells, EPROM, EEPROM, or antifuses. Antifuses are one-time programmable devices (Figure 1), which when "blown" create a connection, while when "unblown" no current can flow between their terminals (thus, it is an "anti"-fuse, since its behavior is opposite to a standard fuse). Because the configuration of an antifuse is permanent, antifuse-based FPGAs are one-time programmable, while SRAM-based FPGAs are reprogrammable, even in the target system. Since SRAMs are volatile, an SRAM-based FPGA must be reprogrammed every time the system is powered up, usually from a ROM included in the circuit to hold configuration files. Note that FPGAs often have on-chip control circuitry to automatically load this configuration data. EEPROM/EPROM (Figure 2) are devices somewhere between SRAM and antifuse in their features. The programming of an EEPROM/EPROM is retained even when the power is turned off, avoiding the need to reprogram the chip at power-up, while their configuration can be changed electrically. However, the high voltages required to program the device often means that they are not reprogrammed in the target system.

SRAM cells are larger than antifuses and EEPROM/EPROM, meaning that SRAM-based FPGAs will have fewer configuration points than FPGAs using other programming technologies. However, SRAM-based FPGAs have numerous advantages. Since they are easily reprogrammable, their configurations can be changed for bug fixes or upgrades. Thus they provide an ideal prototyping medium. Also, these devices can be used in situations where they can expect to have numerous different configurations, such as multi-mode systems and reconfigurable computing machines. More details on such applications are included later in this paper. Because antifuse-based FPGAs are only one-time programmable, they are generally not used in reprogrammable systems. EEPROM/EPROM devices could potentially be reprogrammed in system, although in general this feature is not widely used. Thus, this paper will concentrate solely on SRAM-based FPGAs.

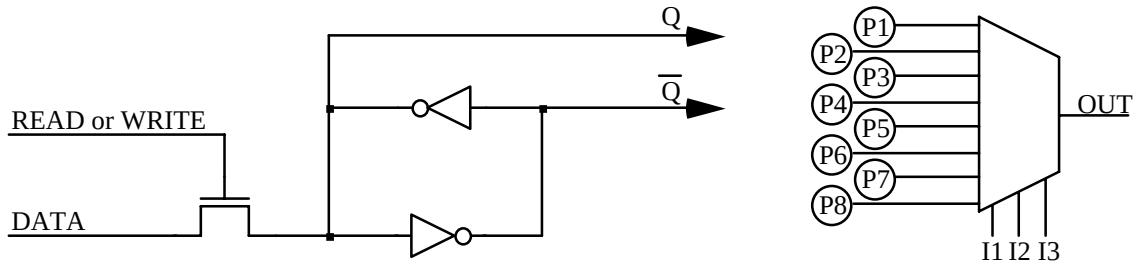

Figure 3. Programming bit for SRAM-based FPGAs (left) [Xilinx94], and a 3-input LUT (right).

There are many different types of FPGAs, with many different structures. Instead of discussing all of them here, which would be quite involved, this section will present two representative FPGAs. Details on many others can

be found elsewhere [Brown92, Rose93, Chan94, Jenkins94, Trimberger94, Oldfield95]. Note that reconfigurable systems can often employ non-FPGA reconfigurable elements; These will be described in section 5.

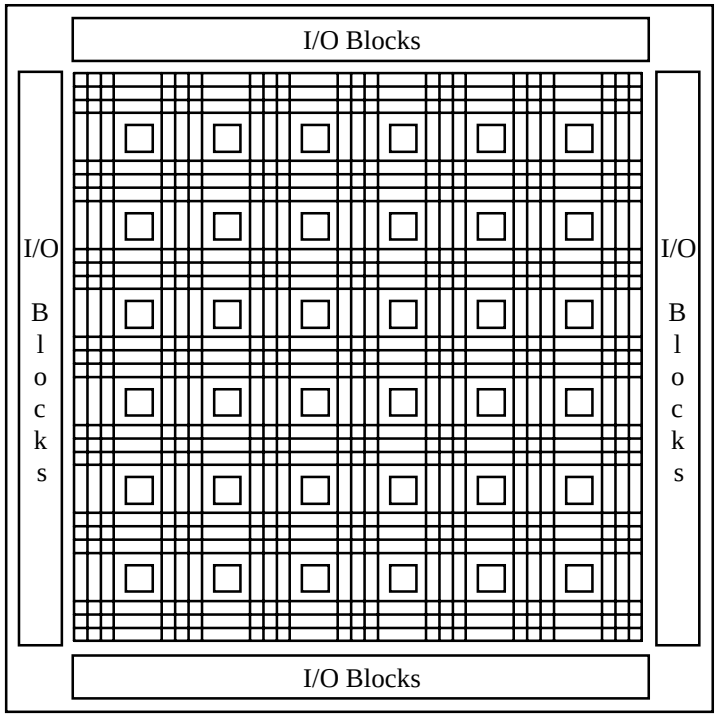

Figure 4. The Xilinx 4000 series FPGA structure [Xilinx94]. Logic blocks are surrounded by horizontal and vertical routing channels.

In SRAM-based FPGAs memory cells are scattered throughout the FPGA. As shown in Figure 3 left, a pair of cross-coupled inverters will sustain whatever value is programmed onto them. A single n-transistor gate is provided for either writing a value or reading a value back out. The ratio of sizes between the transistor and the upper inverter is set to allow values sent through the n-transistor to overpower the inverter. The readback feature is used during debugging to determine the current state of the system. The actual control of the FPGA is handled by the Q and $\overline{Q}$ outputs. One simple application of an SRAM bit is to have the Q terminal connected to the gate of an n-transistor. If a 1 is assigned to the programming bit, the transistor is closed, and values can pass between the source and drain. If a 0 is assigned, the transistor is opened, and values cannot pass. Thus, this construct operates similarly to an antifuse, though it requires much more area. One of the most useful SRAM-based structures is the lookup table (LUT). By connecting 2^N programming bits to a multiplexer (Figure 3 right), any N-input combinational Boolean function can be implemented. Although it can require a large number of programming bits for large N, LUTs of up to 5 inputs can provide a flexible, powerful function implementation medium.

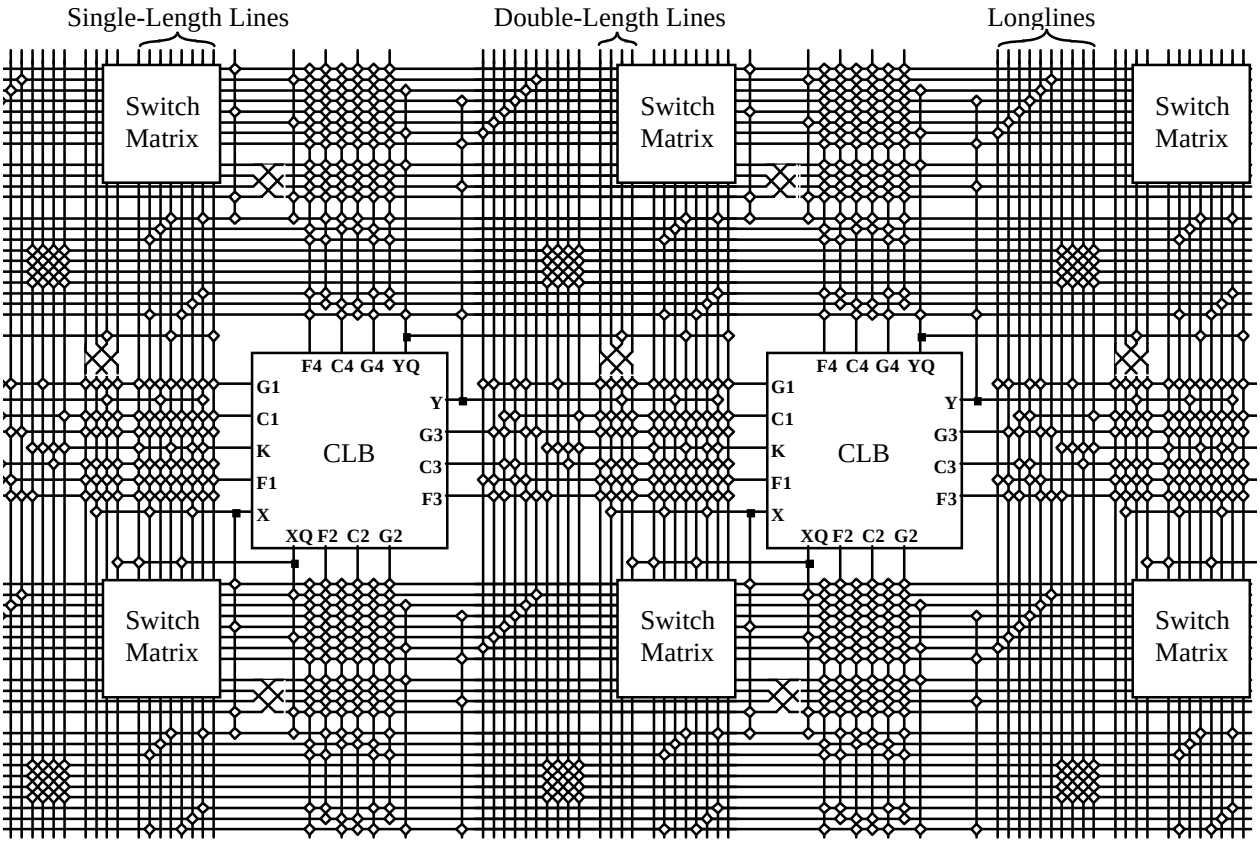

Figure 5. Details of the Xilinx 4000 series routing structure [Xilinx94]. The CLBs (Configurable Logic Blocks) are surrounded by vertical and horizontal routing channels containing Single-Length Lines, Double-Length Lines, and Longlines. Empty diamonds represent programmable connections between perpendicular signal lines (signal lines on opposite sides of the diamonds are always connected).

One of the best known FPGAs is the Xilinx Logic Cell Array (LCA) [Trimberger93, Xilinx94]. In this section we will describe their third generation FPGA, the Xilinx 4000 series. The Xilinx array is an Island-style FPGA [Trimberger94] with logic cells embedded in a general routing structure that permits arbitrary point-to-point communication (Figure 4). The only requirement for good routing in this structure is that the source and destinations be relatively close together. Details of the routing structure are shown in Figure 5. Each of the inputs of the cell (**F1**-**F4**, **G1**-**G4**, **C1**-**C4**, **K**) comes from one of a set of tracks adjacent to that cell. The outputs are similar (**X**, **XQ**, **Y**, **YQ**), except they have the choice of both horizontal and vertical tracks. The routing structure is made up of three lengths of lines. Single-length lines travel the height of a single cell, where they then enter a switch matrix (Figure 6 right). The switch matrix allows this signal to travel out vertically and/or horizontally from the switch matrix. Thus, multiple single-length lines can be cascaded together to travel longer distances. Double-length lines are similar, except that they travel the height of two cells before entering a switch matrix (notice that only half the double-length lines enter a switch matrix, and there is a twist in the middle of the line). Thus, double-length lines are useful for longer-distance routing, traversing two cell heights without the extra delay and the wasted configuration sites of an intermediate switch matrix. Finally, longlines are lines that go half the chip height, and do not enter the switch matrix. In this way, very long-distance routes can be accommodated efficiently. With this rich sea of routing resources, the Xilinx 4000 series is able to handle fairly arbitrary routing demands, though mappings emphasizing local communication will still be handled more efficiently.

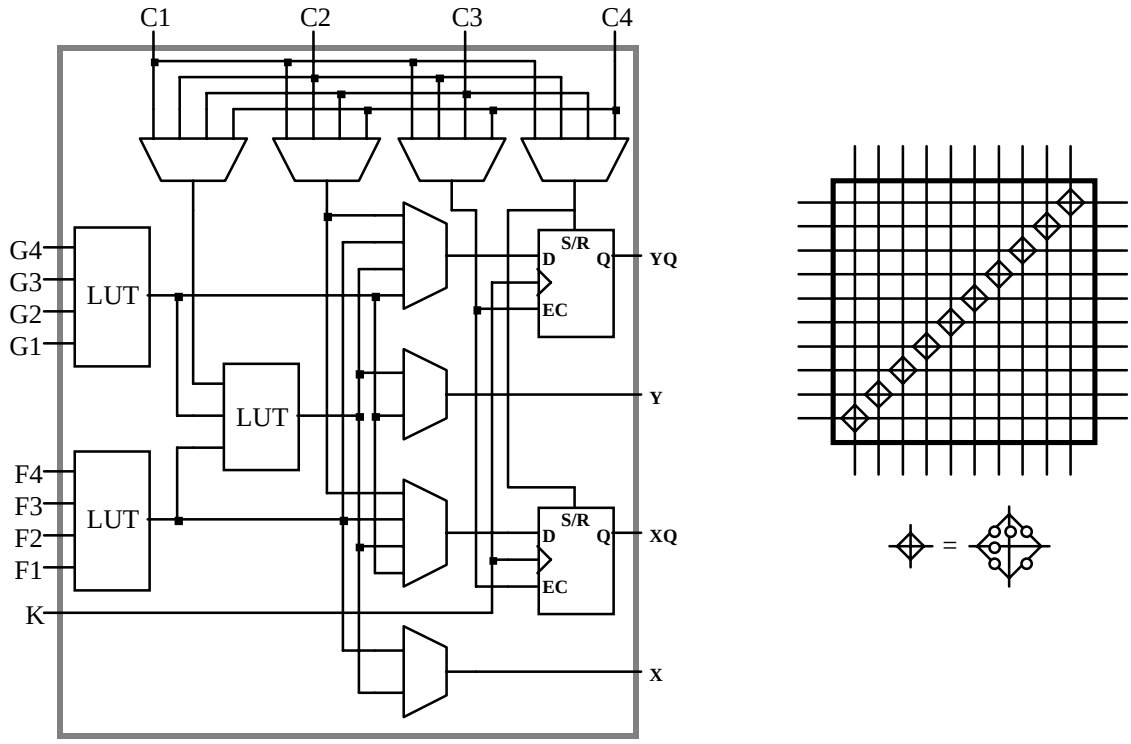

Figure 6. Details of the Xilinx CLB (left) and switchbox (top right) [Xilinx94]. The multiplexers, LUTs, and latches in the CLB are configured by SRAM bits. Diamonds in the switchbox represent six individual connections (bottom right), allowing any permutation of connections among the four signals incident to the diamond.

As shown in Figure 6 left, the Xilinx 4000 series logic cell is made up of three lookup-tables (LUTs), two programmable flip-flops, and multiple programmable multiplexers. The LUTs allow arbitrary combinational functions of its inputs to be created. Thus, the structure shown can perform any function of five inputs (using all three LUTs, with the **F** & **G** inputs identical), any two functions of four inputs (the two 4-input LUTs used independently), or some functions of up to nine inputs (using all three LUTs, with the **F** & **G** inputs different). SRAM controlled multiplexers then can route these signals out the **X** and **Y** outputs, as well as to the two flip-flops. The inputs at top (**C1-C4**) provide enable and set or reset signals to the flip-flops, a direct connection to the flip-flop inputs, and the third input to the 3-input LUT. This structure yields a very powerful method of implementing arbitrary, complex digital logic. Note that there are several additional features of the Xilinx FPGA not shown in these figures, including support for embedded memories and carry chains.

While many SRAM-based FPGAs are designed like the Xilinx architecture, with a routing structure optimized for arbitrary, long-distance communications, several other FPGAs concentrate instead on local communication. The Cellular style FPGAs [Trimberger94] feature fast, local communication resources, at the expense of more global, long-distance signals. As shown in Figure 7, the CLi FPGA [Jenkins94] has an array of cells, with a limited number of routing resources running horizontally and vertically between the cells. There is one local communication bus on each side of the cell. It runs the height of eight cells, at which point it enters a repeater. Express buses are similar to local buses, except that there are no connections between the express buses and the cells. The repeaters allow access to the express buses. These repeaters can be programmed to connect together any of the two local buses and two express buses connected to it. Thus, limited global communication can be accomplished on the local and express buses, with the local buses allowing shorter-distance communications and connections to the cells, while express buses allow longer-distance connections between local buses.

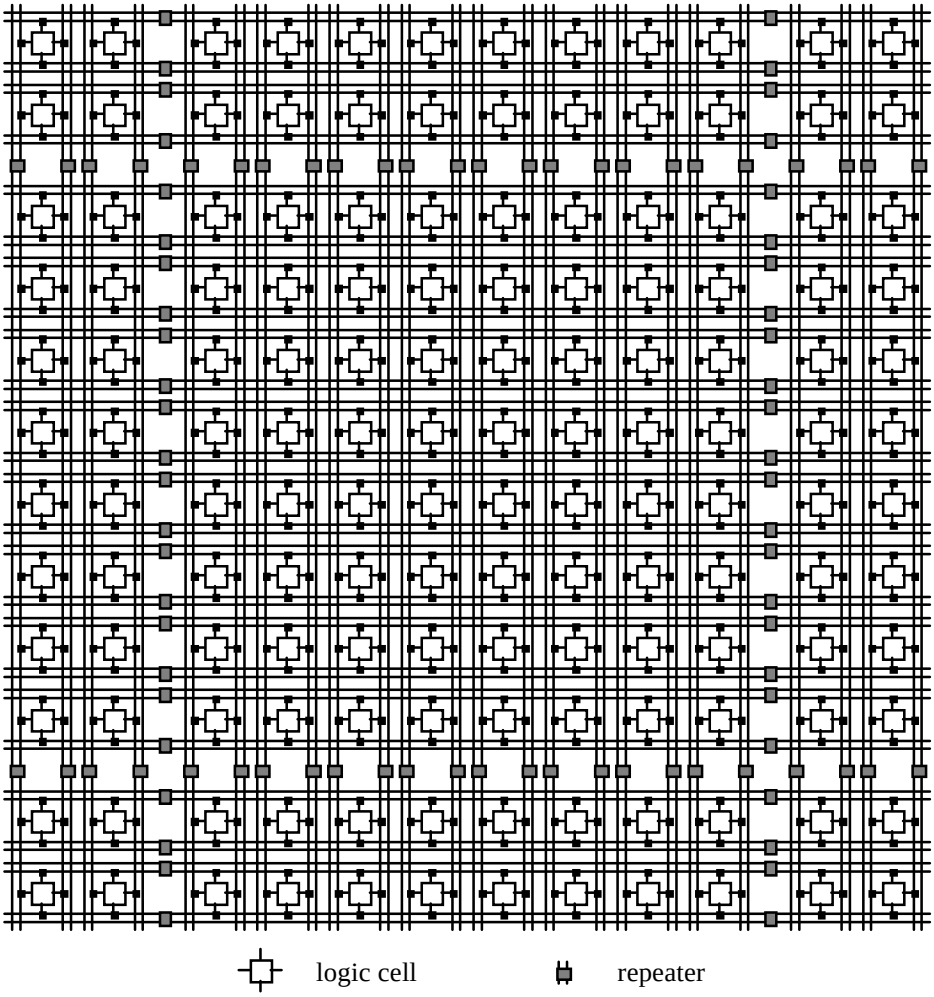

Figure 7. The CLi6000 routing architecture [Jenkins94]. One 8x8 tile, plus a few surrounding rows and columns, is shown. The full array has many of these tiles abutted horizontally and vertically.

While the local and global buses allow some of the flexibility of the Xilinx FPGA's arbitrary routing structure, there are significantly fewer buses in the CLi FPGA than are present in the Xilinx FPGA. The CLi FPGA instead features a large number of local communication resources. As shown in Figure 8, each cell receives two signals from each of its four neighbors. It then sends the same two outputs (**A** and **B**) to all of its neighbors. That is, the cell one to the north will send signals **AN** and **BN**, and the cell one to the south will send **AS** and **BS**, while both will receive the same signals **A** and **B**. The input signals become the inputs to the logic cell (Figure 9).

Instead of Xilinx's LUTs, which require many programming bits per cell, the CLi logic block is much simpler. It has multiplexers controlled by SRAM bits which select one each of the **A** and **B** outputs of the neighboring cells. These are then fed into AND and XOR gates within the cell, as well as into a flip-flop. Although the possible functions are complex, notice that there is a path leading to the **B** output that produces the NAND of the selected **A** and **B** inputs, and sending it out the **B** output. This path is enabled by setting the two 2:1 multiplexers to their constant input, and setting **B**'s output multiplexer to the 3rd input from the top. Thus, the cell is functionally complete. Also, with the **XOR** path leading to output **A**, the cell can efficiently implement a half-adder. The cell can perform a pure routing function by connecting one of the **A** inputs to the **A** output, and one of the **B** inputs to the **B** output, or vice-versa. This routing function is created by setting the two 2:1 multiplexers to their constant inputs, and setting **A**'s and **B**'s output multiplexer to either of their top two inputs. There are also provisions for bringing in or sending out a signal on one or more of the neighboring local buses (**NS1**, **NS2**, **EW1**, **EW2**). Note that since there is only a single wire connecting the bus terminals, there can only be a single signal sent to or received from the local buses. If more than one of the buses is connected to the cell, they will be coupled

together. Thus, the cell can take a signal running horizontally on an **EW** local bus, and send it vertically on a **NS** local bus, without using up the cell's logic unit. However, by bringing a signal in from the local buses, the cell can implement two 3-input functions.

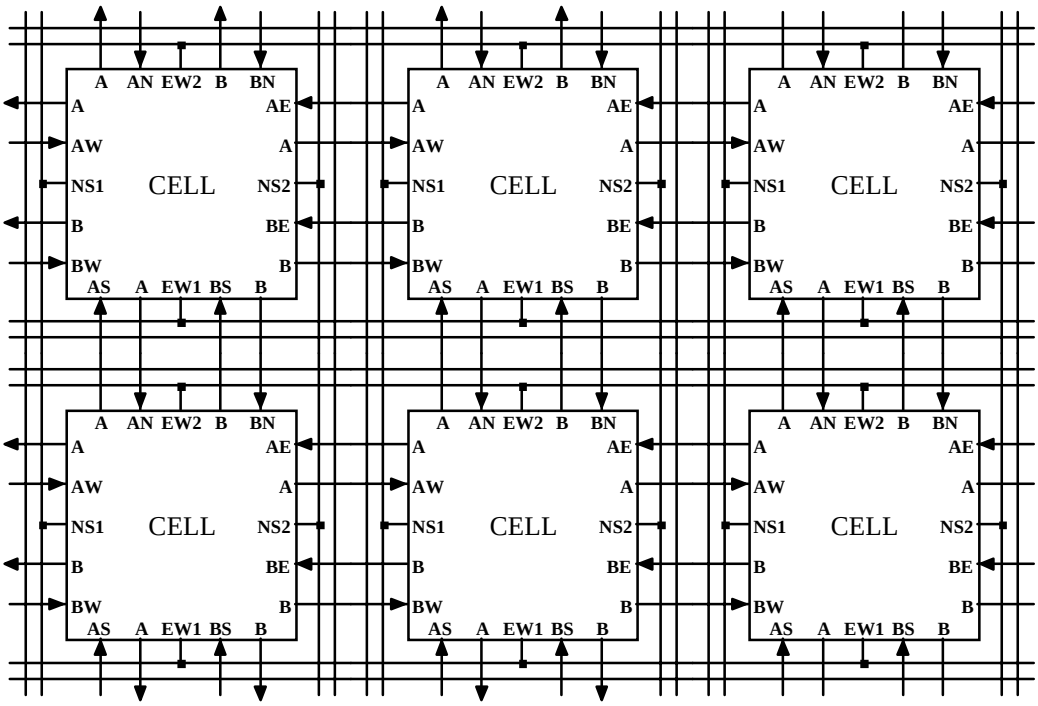

Figure 8. Details of the CLi routing architecture [Jenkins94].

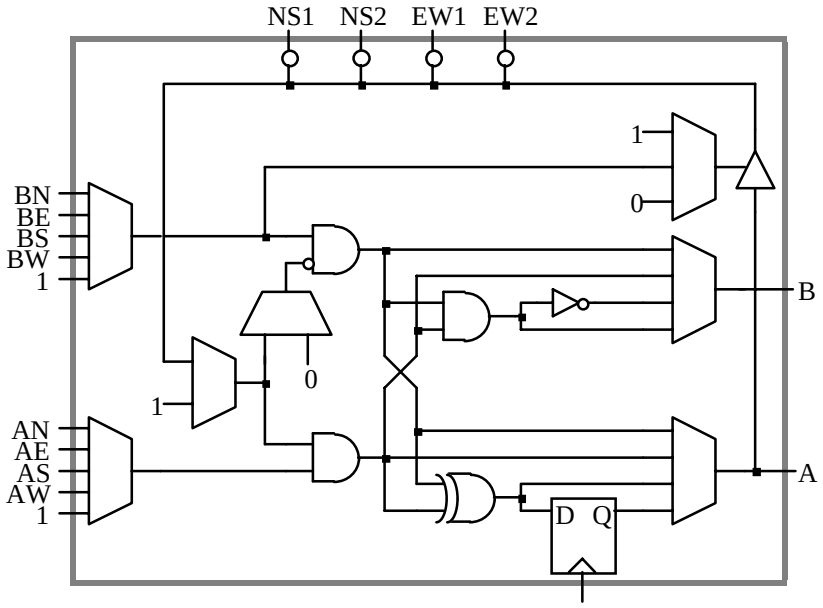

Figure 9. The CLi logic cell [Jenkins94].

The major differences between the Island style architecture of the Xilinx 4000 series and the Cellular style of the CLi FPGA is in their routing structure and cell granularity. The Xilinx 4000 series is optimized for complex, irregular random logic. It features a powerful routing structure optimized for arbitrary global routing, and large logic cells capable of providing arbitrary 4-input and 5-input functions. This provides a very flexible architecture,

though one that requires many programming bits per cell (and thus cells that take up a large portion of the chip area). In contrast, the CLi architecture is optimized for highly local, pipelined circuits such as systolic arrays and bit-serial arithmetic. Thus, it emphasizes local communication at the expense of global routing, and has simple cells. Because of the very simple logic cells there will be many more CLi cells on a chip than will be found in the Xilinx FPGA, yielding a greater logic capacity for those circuits that match the FPGA's structure. Because of the restricted routing, the CLi FPGA is much harder to automatically map to than the Xilinx 4000 series, though the simplicity of the CLi architecture makes it easier for a human designer to hand-map to the CLi's structure. Thus, in general, cellular architectures tend to appeal to designers with appropriate circuit structures who are willing to spend the effort to hand-map their circuits to the FPGA, while the Xilinx 4000 series is more appropriate for handling random-logic tasks and automatically-mapped circuits.

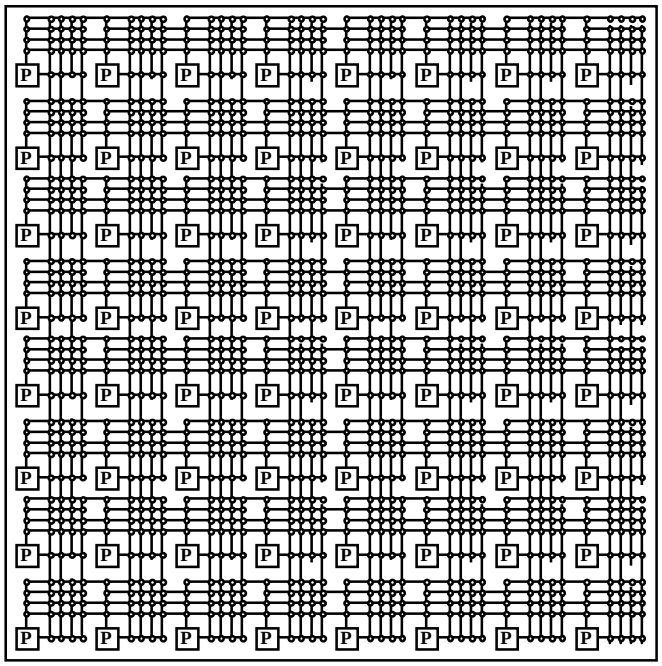

Figure 10. The Aptix FPIC architecture [Aptix93a]. The boxed P indicates an I/O pin.

Compared with technologies such as full-custom, standard cells, and MPGAs, FPGAs will in general be slower and less dense due to the configuration points, which take up significant chip area, and add extra capacitance and resistance (and thus delay) to the signal lines. Thus, the programming bits add an unavoidable overhead to the circuit, which can be reduced by limiting the configurability of the FPGA, but never totally eliminated. Also, since the metal layers in an FPGA are prefabricated, while the other technologies custom fabricate the metal layers for a given circuit, the FPGA will have less optimized routing. This again results in slower and larger circuits. However, even given these downsides, FPGAs have the advantage that they are completely prefabricated. This means that they are ready to use instantly, while mask-programmed technologies can require weeks to be customized. Also, since there is no custom fabrication involved in an FPGA, the fabrication costs can be amortized over all the users of the architecture, removing the significant NREs of other technologies. However, per-chip costs will in general be higher, making the technology better suited for low volume applications. Also, since SRAM-based FPGAs are reprogrammable, they are ideal for prototyping, since the chips are reusable after bug fixes or upgrades, where mask-programmed and antifuse versions would have to be discarded.

A technology similar to SRAM-based FPGAs is Field-Programmable Interconnect Components (FPIC) [Aptix93a] and Devices (FPID) [I-Cube94] (we will use FPIC from now on to refer to both FPIC & FPID devices). Like an SRAM-based FPGA, an FPIC is a completely prefabricated device with an SRAM-configured routing structure (Figure 10). Unlike an FPGA, an FPIC has no logic capacity. Thus, the only use for an FPIC is as a device to arbitrarily interconnect its I/O pins. While this is not generally useful for production systems, since a fixed

interconnection pattern can be achieved by the printed circuit board that holds a circuit, it can be quite useful in prototyping and reconfigurable computing (these applications are discussed later in this paper). In each of these cases, the connections between the chips in the system may need to be reprogrammable, or this connection pattern may change over time. In a reconfigurable computer, many different mappings will be loaded onto the system, and each of them may desire a different interconnection pattern. In prototyping, the connections between chips may need to be changed over time for bug fixes and logic upgrades. In either case, by routing all of the I/O pins of the logic-bearing chips to FPICs, the interconnection pattern can easily be changed over time. Thus, fixed routing patterns can be avoided, potentially increasing the performance and capacity of the prototyping or reconfigurable computing machine.

There is some question about the economic viability of FPICs. The problem is that they must provide some advantage over an FPGA with the same I/O capacity, since in general an FPGA can perform the same role as the FPIC. One possibility is providing significantly more I/O pins in an FPIC than are available in an FPGA. This can be a major advantage, since it takes many smaller I/O chips to match the communication resources of a single high-I/O chip (i.e., a chip with N I/Os requires three chips with 2/3 the I/Os to match the flexibility). However, because the packaging technology necessary for such high I/O chips is somewhat exotic, high-I/O FPICs can be expensive. Another possibility is to provide higher performance or smaller chip size with the same I/O capacity. Since there is no logic on the chip, the space and capacitance due to the logic can be removed. However, even with these possible advantages, FPICs face the significant disadvantage that they are restricted to a limited application domain. Specifically, while FPGAs can be used for prototyping, reconfigurable computing, low volume products, fast time-to-market systems, and multi-mode systems, FPICs are restricted to the interconnection portion of prototyping and reconfigurable computing solutions. Thus, FPICs may never become commodity parts, greatly increasing their unit cost.

3. Reprogrammable Logic Applications

With the development of FPGAs there are now opportunities for implementing quite different systems than were possible with other technologies. In this section we will discuss many of these new opportunities, especially those of multi-FPGA systems.

When FPGAs were first introduced they were primarily considered to be just another form of gate array. While they had lower speed and capacity, and had a higher unit cost, they did not have the large startup costs and lead times necessary for MPGAs. Thus, they could be used for implementing random logic and glue logic in low volume systems with non-aggressive speed and capacity demands. If the capacity of a single FPGA was not enough to handle the desired computation, multiple FPGAs could be included on the board, distributing the computation among these chips.

FPGAs are more than just slow, small gate arrays. The critical feature of (SRAM-based) FPGAs is their in-circuit reprogrammability. Since their programming can be changed quickly, without any rewiring or refabrication, they can be used in a much more flexible manner than standard gate arrays. One example of this is multi-mode hardware. For example, when designing a digital tape recorder with error-correcting codes, one way to implement such a system is to have separate code generation and code checking hardware built into the tape machine. However, there is no reason to have both of these functions available simultaneously, since when reading from the tape there is no need to generate new codes, and when writing to the tape the code checking hardware will be idle. Thus, we can have an FPGA in the system, and have two different configurations stored in ROM, one for reading and one for writing. In this way, a single piece of hardware handles multiple computations. There have been several multi-configuration systems built from FPGAs, including the just mentioned tape machine, generic printer and CCD camera interfaces, pivoting monitors with landscape and portrait configurations, as well as others [Xilinx92, Fawcett94, Mayrhofer94, Shand95].

While the previous uses of FPGAs still treat these chips purely as methods for implementing digital logic, there are other applications where this is not the case. A system of FPGAs can be seen as a computing substrate with somewhat different properties than standard microprocessors. The reprogrammability of the FPGAs allows one to download algorithms onto the FPGAs, and change these algorithms just as general-purpose computers can change programs. This computing substrate is different from standard processors, in that it provides a huge amount of

fine-grain parallelism, since there are many logic blocks on the chips, and the instructions are quite simple, on the order of a single five bit input, one bit output function. Also, while the instruction-stream of a microprocessor can be arbitrarily complex, with the function computed by the logic changing on a cycle by cycle basis, the programming of an FPGA is in general held constant throughout the execution of the mapping (exceptions to this include techniques of run-time reconfigurability described below). Thus, to achieve a variety of different functions in a mapping, a microprocessor does this temporally, with different functions executed during different cycles, while an FPGA-based computing machine achieves variety spatially, having different logic elements compute different functions. This means that microprocessors are superior for complex control flow and irregular computations, while an FPGA-based computing machine can be superior for data-parallel applications, where a huge quantity of data must be acted on in a very similar manner. Note that there is work being done on trying to bridge this gap, and develop FPGA-processor hybrids that can achieve both spatial and limited temporal function variation [Ling93, Bolotski94, DeHon94, Maliniak94, DeHon96, Mirsky96].

There have been several computing applications where a multi-FPGA system has delivered the highest performance implementation. An early example is genetic string matching on the Splash machine [Gokhale90]. Here, a linear array of Xilinx 3000 series FPGAs was used to implement a systolic algorithm to determine the *edit distance* between two strings. The edit distance is the minimum number of insertions and deletions necessary to transform one string into another, so the strings “flea” and “fleet” would have an edit distance of 3 (delete “a” and insert “et” to go from “flea” to “fleet”). As shown in [Lopresti91], a dynamic-programming solution to this problem can be implemented in the Splash system as a linear systolic circuit, with the strings to be compared flowing in opposite directions through the linear array. Processing can occur throughout the linear array simultaneously, with only local communication necessary, producing a huge amount of fine-grain parallelism. This is exactly the type of computation that maps well onto a multi-FPGA system. The Splash implementation was able to offer an extremely high performance solution for this application, achieving performance approximately 200 times faster than supercomputer implementations. There have been many other applications where a multi-FPGA system has offered the highest performance solution, including: mathematics applications such as long multiplication [Bertin89, Vuillemin96], modular multiplication [Cuccaro93], and RSA cryptography [Vuillemin96]; physics applications such as real-time pattern recognition in high-energy physics [Högl95], Monte Carlo algorithms for statistical physics [Monaghan93, Cowen94], second-level triggers for particle colliders [Moll95], and Heat and Laplace equation solvers [Vuillemin96]; general algorithms such as the Traveling Salesman Problem [Graham95], Monte Carlo yield modeling [Howard94b], genetic optimization algorithms [Scott95, Graham96], region detection and labeling [Rachakonda95], stereo matching for stereo vision [Vuillemin96], hidden Markov Modeling for speech recognition [Schmit95], and genetic database searches [Lopresti91, Hoang93, Lemoine95].

One of the most successful uses for FPGA-based computation is in ASIC logic emulation. The idea is that the designers of a custom ASIC need to make sure that the circuit they designed correctly implements the desired computation. Software simulation can perform these checks, but does so quite slowly. In logic emulation, the circuit to be tested is instead mapped onto a multi-FPGA system, yielding a solution several orders of magnitude faster than software simulation.

Logic emulation shares many of the advantages (and disadvantages) of both prototyping and software simulation. Like a prototype, the circuit to be evaluated is implemented in hardware so that it can achieve high performance test cycles. However, like software simulation, the emulation can easily be observed and altered to help isolate bugs. Logic emulation takes a gate-level description of a logic circuit and maps it onto a multi-FPGA system. This multi-FPGA system is a prefabricated, reprogrammable compute engine that can be configured to implement the desired circuitry in a matter of seconds. However, to transform the circuit description into a mapping suitable for this multi-FPGA system can take many hours to complete. This mapping process is usually completely automated by the emulator’s system software. Once the circuit is mapped to the multi-FPGA system, the emulator provides a complete, functional implementation of the circuit that can evaluate millions of circuit cycles per second. This is orders of magnitude faster than even simulation-accelerators, since the multi-FPGA system can implement the complete circuit in parallel, while accelerators simply provide one or more sequential logic evaluation processors.

Emulators provide a middle-ground between software simulation and prototyping. Compared to software simulation, an emulation executes much faster than a simulation. However, it can take a long time to map a circuit onto the emulator, and it is more difficult to observe and modify the behavior of the circuit. Thus, software simulation is a better choice for testing small subcircuits or small numbers of complete circuit cycles, where the software's flexibility and ease of use outweighs the performance penalties. Compared to a prototype, an emulation is much easier and faster to create, and it has much greater observability, controllability, and modifiability than a prototype. However, the emulation cannot run at the same speed as the target system. Thus, the emulator is a much better choice for providing a huge number of test cycles than a prototype when one expects to find bugs in the system, but it is no replacement for final checkout of the system via a prototype. For circuits that will execute software programs, an emulator can be used to debug this software much earlier in the design process than a physical prototype. This is because an emulation can be created from a high-level specification of the circuit, while prototyping must wait until the complete circuit has been designed. Simulation in general cannot be used for software development, since it is much too slow to execute enough cycles of the software. Also, just like a prototype, an emulation can be given to the end-user so that the circuit can be evaluated before the design is completed. In this way, the user can get a much better feeling for whether the design will fulfill the user's needs, something that is difficult with just a written specification. The emulation can be inserted into the target environment (as long as some method for reducing the performance demands on the system can be provided [Quickturn93, Hauck95a, Hauck95b]), and the system can be evaluated in a more realistic setting. This helps both to debug the circuit, and also to test the circuit interfaces and environment. For example, often a custom ASIC and the circuit board that will contain it will be developed simultaneously. An emulation of the ASIC can be inserted into this circuit board prototype, testing both the ASIC functions and the board design.

One limitation of emulation is that it retains only the functional behavior of the circuit, which means that validation of the performance and timing features of a circuit cannot be performed on a logic emulator. Once a prototype is constructed, both logic emulation and software simulation are still valuable tools [Gateley94]. When an error is found in a physical prototype, it can be difficult to isolate the exact cause in the circuit. An emulator can be used to reproduce the failure, since it can execute nearly as many cycles as the prototype, and the emulator's observability can be used to isolate the failure. Then, detailed testing can be performed by software simulation. Thus, logic emulation plays a complementary role to both software simulation and prototyping in logic validation.

An emerging application of FPGA-based computing is the training and execution of neural networks. A neural network is a powerful computational model based on the structure of neurons in the brain. These systems have proven effective for tasks such as pattern recognition and classification. One important aspect of these nets is that each of the basic elements in the network must be configured for a given problem. This configuration (or "learning") process revolves around exposing the network to situations where the correct answer is known, and adjusting the network's configuration so that it returns the correct answer. The execution of a neural network can be time consuming on a standard processor, especially for the repeated execution runs required during the training process. Thus, there is great interest in implementing neural networks in reprogrammable systems, both because of the speed benefits, as well as because the reprogrammability of the FPGAs can support the reconfiguration necessary to program a neural network.

Systems like the just mentioned neural-network implementations, as well as multi-mode systems, take advantage of an FPGA's reprogrammability by changing the chip's programming over time, much as a standard processor context-switches to a new program. However, it is possible to make more aggressive use of this ability to develop new types of applications. The FPGA can be viewed as a demand-paged hardware resource, yielding "virtual hardware" similar to virtual memory in today's computers. In such systems (usually grouped under the term *Dynamically Reconfigurable* or *Run-Time Reconfiguration*) an application will require many different types of computations, and each of these computations has a separate mapping to the reprogrammable logic. For example, an image processing application for object thinning may require separate pre-filtering and thresholding steps before running the thinning operation, each of which could be implemented in a separate FPGA mapping [Wirthlin96]. Although these mappings could be spread across multiple FPGAs, these steps must take place sequentially, and in a multi-FPGA system only one mapping would be actively computing at a time. Run-time reconfiguration saves hardware by reusing the same resource. It also relaxes the upper limit on the number of

configurations allowed, since in a multi-FPGA system the number of FPGAs available is usually fixed (either by the architecture or by the current hardware instantiation), while a run-time reconfigured system can have as many configurations as there is storage space to hold them. Because of these advantages there has been a lot of work on run-time reconfigurable systems, applications, and support tools [Lysaght91, French93, Eldredge94, Lysaght94b, Koch94a, Razdan94, Ross94, Gokhale95, Hadley95, Jones95, Schoner95, DeHon96, Luk96, Villasenor96, Wittig96]. Note that this approach can be taken even further to local run-time reconfiguration [Lysaght94a, Singh94, Hutchings95, Brebner95, Lysaght95, Wirthlin95, Clark96, Wirthlin96]. In a locally run-time reconfigurable system different phases of an algorithm can have mappings to just a portion of the FPGA. Multiple configurations can be loaded into the system, with each configuration occupying different portions of the FPGA. In this way, the FPGA becomes a hardware cache, with the set of loaded mappings varying over time based on the requirements of the algorithm. When a configuration is needed that is currently not in the FPGA, it is loaded into the FPGA, replacing other mappings that are no longer required. In this way multiple small mappings can coexist-exist in the FPGA, potentially eliminating most of the time overhead of complete FPGA reprogramming for each small mapping.

Reprogrammable systems have a great deal of potential for providing higher-performance solutions to many different applications. However, just as it is important to carefully select the type of application mapped to a multi-FPGA system, it is also crucial to carefully construct the reprogrammable system to support these applications. In section 4 we discuss the types of reprogrammable systems.

4. Reprogrammable Systems

There is a wide variety of different roles for reprogrammable logic, and these roles can require very different types of reprogrammable systems. Some are isolated, small capacity systems used for the replacement of small digital logic systems. Others have hundreds or thousands of FPGAs, with capacities into the millions of gates, which rival supercomputers for certain types of applications.

One of the most common types of reprogrammable systems is a one-FPGA or two-FPGA system used for interfacing and other standard logic tasks. With the addition of memory resources and the appropriate interface circuitry, these systems provide a much greater flexibility than traditional implementations. These systems can be prefabricated and used for multiple applications, reducing both design time and inventory risk. When a new application must be developed, the designer need only specify the logic required to handle the new functions. This logic can then be automatically mapped to the FPGAs, and the prefabricated system can be used immediately. This contrasts with traditional approaches, which have the extra time and complexity costs of custom board design and fabrication.

Reprogrammable systems need not be prefabricated. In fact, there is a great deal of interest in automatic creation of custom application-specific reprogrammable systems [Kuznar93, Woo93, Kuznar94, Chan95, Huang95]. In such an approach, the logic is again specified by the designer, and automatic tools map this circuitry into FPGA logic. However, instead of being constrained to a predefined topology, the mapping tools are instead allowed to develop a custom multi-chip solution. Because the chips are interconnected in response to the needs of a specific application, a much more highly optimized solution can be developed. This can be a significant savings, since premade systems often present the designer or the mapping tools with a very constrained system, greatly complicating the mapping process. However, there are two downsides to this approach. First, if a reprogrammable system is custom designed in response to a specific application, then obviously this system cannot be premade. Thus, the user must wait for the new board to be fabricated. Also, there is little chance of reusing this system for other applications. This increases the hardware costs since these expenses cannot be amortized over multiple designs. The second problem with this approach may be more significant: the final logic design may not be ready when the board layout must be finalized, meaning that much of the advantage of a custom designed system is lost. In some cases, the need to get the system completed is significant enough that the board design cannot wait for the FPGA's logic to be completely specified. In others, even though a mapping may have been generated for the FPGAs before the board was designed, new functions or bug fixes may mandate revisions to the FPGA mappings. In either case, the final mapping to the FPGAs must fit within the constraints of a fixed reprogrammable system, and thus must deal with most of the issues and inefficiencies of premade systems, without a premade system's benefits of reduced time and expense.

Some of the most interesting uses for reprogrammable systems are where the system is viewed not as an isolated entity, but instead as an extension of a computer system. Specifically, reprogrammable logic resources can be added to a standard workstation or personal computer, greatly increasing the processing power for some applications. By using the reprogrammable logic for tasks that work well on these devices, while leaving the rest of the computation on standard processors, the benefits of both models can be realized.

As shown in Figure 11, there are several ways in which reprogrammable logic can be added to standard computer systems: as a functional unit, as a coprocessor, as attached processing units, or as standalone processing units (A somewhat similar classification can be found in [Guccione95]). The most common methods are attached processing units, which are reprogrammable systems on computer add-on cards, and standalone processing units, which are separate reprogrammable cabinets. In these models, a complex reprogrammable system is built out of multiple FPGAs and perhaps memories and other devices. We will refer to these systems jointly as *multi-FPGA systems*. Because of their size, these multi-FPGA systems can accommodate huge logic or computation demands, allowing them to add significant capabilities to the system. However, because these systems are relatively distant from the computer's CPU, and thus there is a large communication delay from the processor to the reprogrammable system, to be effective these systems must handle large chunks of the computation. Specifically, even if a multi-FPGA system can perform the equivalent of 40 processor instructions in a single clock cycle, if it takes 100 cycles to get the data to and from the reprogrammable system the performance advantages of the multi-FPGA system could be swallowed by the communication times. However, if the multi-FPGA system can execute even 40 cycles (of 40 instruction-equivalents per cycle) before needing to communicate with the CPU, the system can achieve speedups over standard processor-only systems. Thus, successful multi-FPGA systems tend to take over large portions of the application's computation, particularly those portions that have only limited communication with other parts of the algorithm. Examples of this include multi-cycle simulation of circuit designs, complete major inner loops of software algorithms, and others. Note that multi-FPGA systems can be viewed as hardware supercomputers, providing a centralized resource for high-performance computation, but only for those applications that fit their computation model. We will discuss multi-FPGA system hardware structures in depth later in this paper.

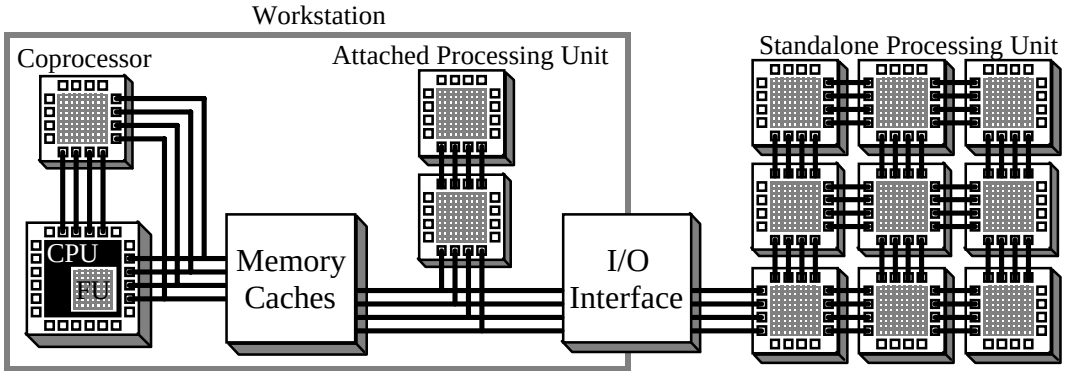

Figure 11. Types of reprogrammable systems.

One way to overcome the communication bottleneck, and potentially achieve wider applicability, is to move the reprogrammable logic closer to the CPU. The reprogrammable logic can be viewed as a coprocessor, akin to a standard FPU, connected directly to the processor. This coprocessor would be used to implement, on a per-application basis, one or more short code sequences from the application. Instead of performing these instructions on the processor, the reprogrammable coprocessor would instead perform the computation. These computations potentially could be performed on the reprogrammable coprocessor much faster than on the CPU, thus speeding up the algorithm. Although communication overheads can still be significant, by placing the reprogrammable logic closer to the CPU (at least in the bus hierarchy) these delays will be much lower, meaning that the coprocessor can handle much smaller portions of the computation than a multi-FPGA system and still achieve performance improvements. This model has already been successful with special-purpose coprocessors for floating-point calculations, graphics acceleration, and many other applications. While a reprogrammable coprocessor cannot match the performance of these application-specific coprocessors, since reprogrammability

can add significant delay to the system, the advantage of a reprogrammable coprocessor is that it can act as a "generic" coprocessor, handling the demands of many different application domains. Specifically, although an FPU can improve the performance for some scientific computations, other applications may never use floating point numbers. Thus, the FPU is wasted for these applications. A reprogrammable, generic coprocessor can have a different configuration for each application, providing functions designed to accelerate each algorithm. Thus, while an application-specific coprocessor can achieve significant performance improvement for some or most applications, a reprogrammable coprocessor can achieve some performance gain for a wider range of applications. This includes accelerating infrequently used or niche algorithms, algorithms for which it will never be cost-effective to include custom acceleration hardware in a general-purpose computer. There has already been work on such generic coprocessors [Athanas93, Cuccaro93, Filloque93, Wazlowski93, Agarwal94, Churcher95, Lawrence95, Clark96, Wirthlin96] which have shown some promising results. Note that a coprocessor need not use only a single FPGA, and many coprocessors are multi-FPGA systems.

A final alternative is to integrate the reprogrammable logic into the processor itself. This reprogrammable logic can be viewed as another functional unit, providing new functions to the processor. Like a reprogrammable coprocessor, a reprogrammable functional unit can be configured on a per-algorithm basis, providing one or more special-purpose instructions tailored to the needs of a given application. If chosen well, these special-purpose instructions can perform in one or two cycles operations that would take much longer in the processor's standard instruction set. Thus, the addition of these new instructions yields an application-specific instruction set on application-independent hardware, yielding a much faster implementation of many different applications. However, this benefit does not come without a cost. Processor real-estate is still a precious commodity, and the inclusion of reprogrammable logic into a processor means that there is less room for caches or other architectural features. Thus, a reprogrammable functional unit need not only yield some performance improvement, but must yield a larger benefit than that of the other features that could have been placed into the processor in the space taken up by the reprogrammable functional unit. This can be especially difficult because the reprogrammable logic must synchronize with the cycle period of the custom processor hardware, meaning that the overhead of reprogrammable logic forces the unit to perform less complex functions than a custom functional unit could achieve. Also, a reprogrammable functional unit makes context-switches more complex, and increases the external bandwidth requirements, since the configurations of the reprogrammable functional units must somehow be changed for different algorithms. While there has been some work done on reprogrammable functional units [French93, Albaharna94, DeHon94, Razdan94, Albaharna96, Rajamani96, Wittig96], there is still much left to do.

While there has been some work on reprogrammable coprocessors and functional units for standard computer systems, by far the majority of reprogrammable systems have been multi-FPGA systems. In section 5 we discuss many of the different multi-FPGA system architectures, highlighting their important features.

5. Multi-FPGA Systems

In previous sections we discussed the applications of multi-FPGA systems. In this section we will explore some of the existing multi-FPGA systems themselves. A large number of systems have been constructed, for many different purposes, using a wide range of structures. Note that this section is intended to illustrate only the types of systems possible, and is not meant to be an in-depth discussion of all the details of existing systems. Thus, some details of the topologies, as well as the number of wires in each link in the systems, have been omitted.

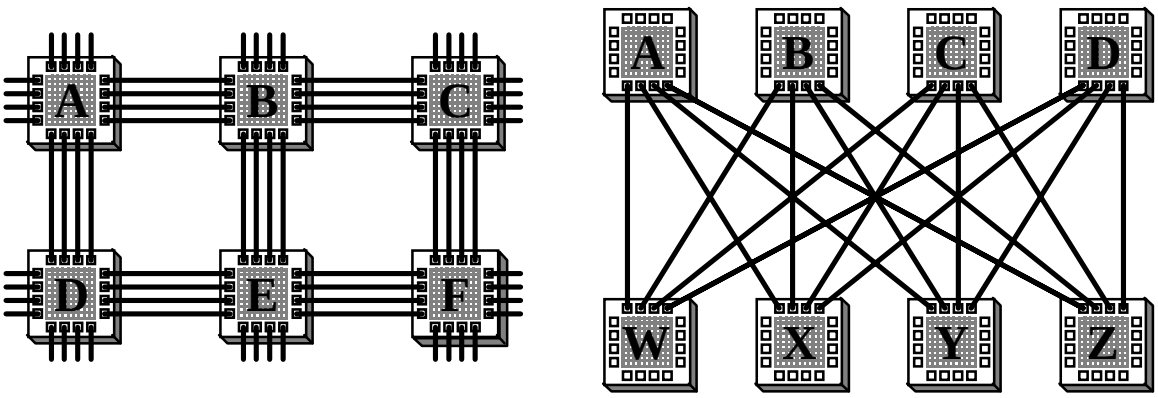

Figure 12. Mesh (left) and crossbar (right) topologies. In the crossbar, chips **A**-**D** are routing-only, while **W**-**Z** hold all the logic in the system.

The most important distinguishing characteristic among multi-FPGA systems is in the topology chosen to interconnect the chips. The most common topologies are mesh and crossbar (or bipartite graph) topologies. In a mesh, the chips in the system are connected in a nearest-neighbor pattern (Figure 12 left). These topologies have the advantage of simplicity, because of the purely local interconnection pattern, as well as easy expandability, since meshes can be grown by adding resources to the edge of the array. Numerous 2D mesh-based systems have been built [Kean92, Shaw93, Bergmann94, Blickle94, Hauck94b, Tessier94, Yamada94], as well as 3D meshes [Sample92, Quénot94]. Linear arrays, which are essentially 1-dimensional meshes, have also been made [Gokhale90, Filloque93, Raimbault93, Monaghan94]. Note that the design of a mesh topology can involve several subtle tradeoffs, with some constructs yielding significantly improved topologies [Hauck94a].

Crossbar topologies separate the elements in the system into logic-bearing and routing-only chips (Figure 12 right). The logic-bearing FPGAs contain all the logic in the system, while the routing-only chips are used purely for inter-FPGA routing. Routing-only chips are connected only to logic-bearing FPGAs, and (usually) have exactly the same number of connections to all logic-bearing FPGAs. Logic-bearing FPGAs are connected only to routing-only FPGAs. The idea behind this topology is that to route between any set of FPGAs requires routing through exactly one extra chip, and that chip is one of the routing-only chips. Because of the symmetry of the system, all routing-only chips can handle this role equally well. This gives much more predictable performance, since regardless of the locations of the source and destinations the delay is the same. In a topology like a mesh, where it might be necessary to route through several intermediate chips, there is a high variance in the delay of inter-FPGA routes. There are two negative features of the crossbar topology. First, crossbar topologies are not expandable, since all routing-only chips need to connect to all logic-bearing FPGAs, and thus the system is constrained to a specific size once the connections to any specific routing-only chip are determined. Second, the topology potentially wastes resources, since the routing-only chips are used purely for routing, while a mesh can use all of its chips for logic and routing. However, since the bottleneck in multi-FPGA systems is the inter-chip routing, this waste of resources may be more than made up for by greater logic utilization in the logic-bearing chips. Also, some of the cost of the wasted resources can be avoided by using less expensive devices for the routing-only chips. Possibilities include FPICs, crossbars, or cheaper FPGAs (either because of older technology or lower logic capacity). Several pure crossbar topologies have been constructed [Chan92, Ferrucci94, Kadi94, Weiss94, Li95].

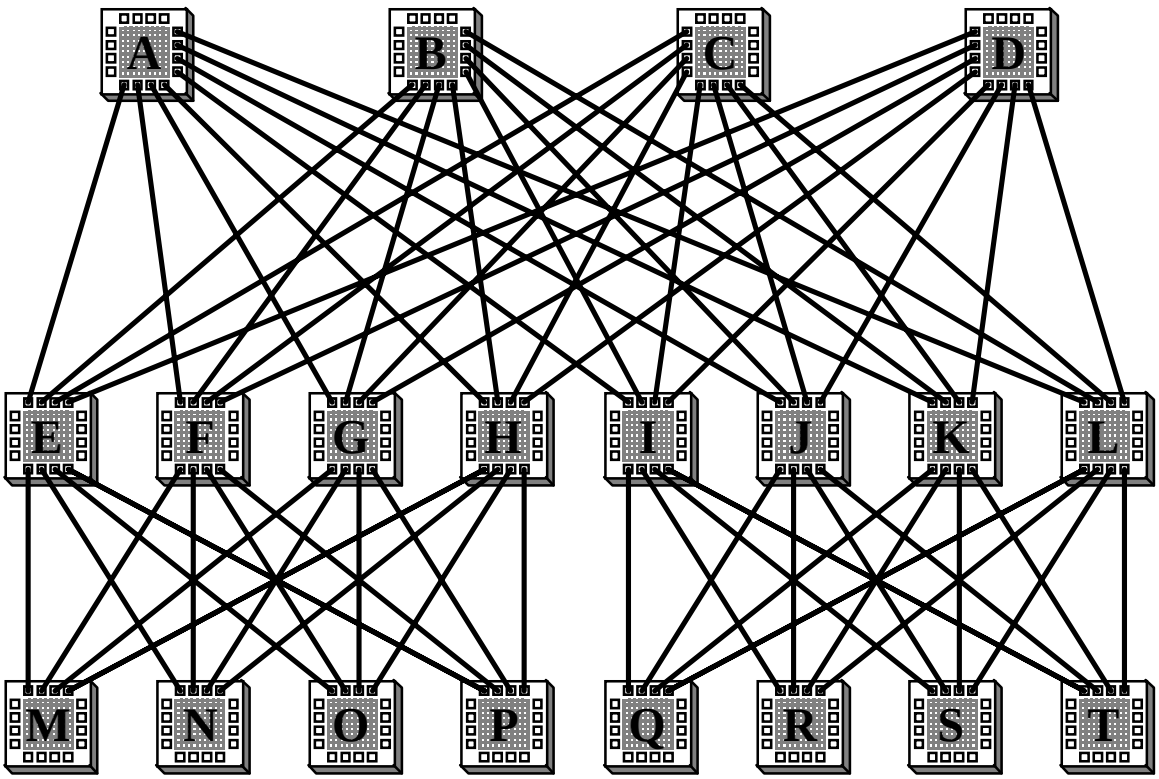

Figure 13. A hierarchy of crossbars. FPGAs **M-T** hold all the logic in the system. Chips **E-H** and **I-J** form two first-level crossbars, and chips **A-D** form a second-level crossbar.

Another topology, which combines the expandability of meshes and the simpler routing of crossbars, is hierarchical crossbars [Varghese93]. As shown in Figure 13, crossbars can be stacked together hierarchically, building up multiple levels of routing chips. There are two simple crossbars in the system, one consisting of routing-only chips **E-H** and logic-bearing FPGAs **M-P**, and a second one consisting of routing-only chips **I-L** and logic-bearing FPGAs **Q-T**. Routing chips **E-L** will be called the first-level crossbars, since they connect directly to the logic-bearing FPGAs. To build the hierarchy of crossbars, the simple crossbars in the system can be thought of as logic-bearing chips in an even larger crossbar. That is, a new crossbar is built with routing-only chips and logic-bearing elements, but in this crossbar the logic-bearing elements are complete, simple crossbars. Note that the connections within this higher-level crossbar go to the routing-only chips in the simple crossbars, so first-level and second-level routing-only chips are connected together. This hierarchy can be continued, building up other crossbars with third-level and higher routing-only chips. In an *N*-level hierarchical crossbar, the chips are arranged as above, with routing-only chips at the *Ith* level connected to chips at the *(I+1)th* and *(I-1)th* level, where the *0th* level is the logic-bearing chips. Note that in contrast to the simple crossbar topology, in a hierarchical crossbar the logic-bearing FPGAs are not connected to all the routing-only chips (even those at the first-level). Full connectivity occurs at the top (*Nth*) level, where all *Nth*-level routing chips are connected to all *(N-1)th* level routing chips.

Routing between two logic-bearing FPGAs in the system simply requires determining the level at which the source and destination share an ancestor, and then routing from the source up to one of these shared ancestors and down to the destination. The routing from the source to the shared ancestor requires routing through exactly one routing-only chip in each of the intervening levels, as does the routing from the ancestor to the destination. Because of the symmetry of the topology, any of the ancestors of the source (for the route up) or destination (for the route down) at a given level can be used to handle the routing, regardless of what other chips are part of the route.

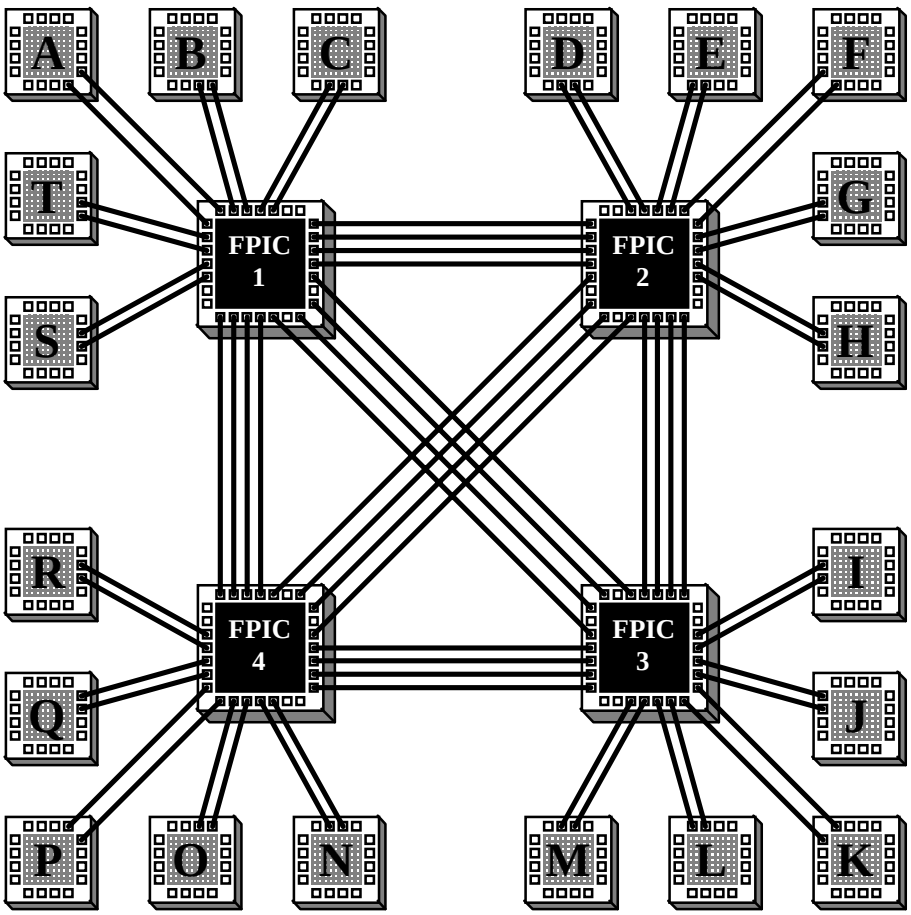

Figure 14. The Aptix AXB-AP4 topology [Aptix93b]. Logic-bearing FPGAs are connected only to routing-only FPICs, but the FPICs connect to both FPGAs and other FPICs.

As mentioned earlier, the advantage of a hierarchical crossbar topology is that it has much of the expandability of a mesh, yet has much of the simplified routing of a crossbar. Since levels of hierarchy can be added to a hierarchical crossbar, it can easily grow larger to handle bigger circuits. Levels of the hierarchy tend to map onto components of the system, such as having a complete first-level crossbar on a single board, a complete second-level crossbar contained in a cabinet, and a complete third-level crossbar formed by interconnecting cabinets [Butts91]. The routing simplicity, as shown above, demonstrates a large degree of flexibility in the routing paths, as well as significant symmetry in the system. How easy it is to route between logic-bearing FPGAs is simple to determine, since if two logic-bearing chips are within the same first-level crossbar, then they are as easy to route between as any other pair of logic-bearing chips within that crossbar. Thus, when mapping onto the topology, the system tries to keep most of the communication between chips in the same first-level crossbar, and most of the rest of the communication between chips in the same second-level crossbar, and so on.

There are two downsides to this topology. First, signals may have to go through many more routing-only chips than in a simple crossbar, since they could potentially have to go all the way up to the top level of the hierarchy to make a connection. However, the maximum routing distance is less than in a mesh, since the length in chips routed through of the maximum route in a mesh grows by $O\left(\sqrt{N}\right)$ (where N is the number of logic-bearing FPGAs in the system), while the length in a hierarchical crossbar grows by $O(\log N)$. The other problem is that the hierarchical crossbar topology requires a large number of resources to implement the routing-only chips in the system. If one could instead use the routing-only chips as logic-bearing chips, the capacity of the system might be greatly increased. However, if this extra logic capacity cannot efficiently be used, it will be of little value.

Some systems use a two-level topology, which is somewhat similar to the crossbar and hierarchical crossbar topologies. Just as in the crossbar topologies, FPGAs in the system are connected only to routing-only chips. However, unlike the pure crossbar, the routing-only chips in the system are connected to both logic-bearing and routing-only chips. That is, there is a topology of connections between the routing-only chips in the system, and the FPGAs connect to these chips. Thus, these systems are similar to multiprocessors, in that for two processing elements to communicate (the FPGAs in a two-level topology), they must send signals through a router connected to the source (an FPIC or crossbar). The signal then travels through intervening routers until it reaches the router connected to the destination, at which point it is sent to that processor (an FPGA in our case). An example of such a system is the Aptix AXB-AP4 [Aptix93b]. As shown in Figure 14, five FPGAs are connected to each FPIC, and all the FPICs are connected together. In this system, the longest route requires moving through 2 FPICs, and no intermediate FPGAs are used for routing. If instead a mesh was built out of the 20 FPGAs in this system, it potentially could require routing through many FPGAs to reach the proper destination, increasing the delay, and using up valuable FPGA I/Os. However, whether the routing flexibility of the two-level topology justifies the substantial cost of the FPICs is unclear. Other two-level systems have been constructed with a variety of topologies between the routing-only chips [Adams93, Galloway94, Njølstad94].

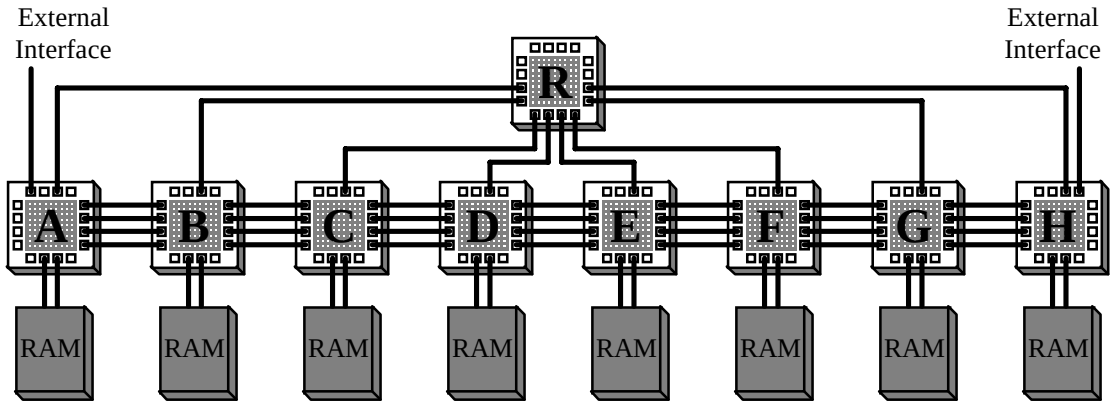

Figure 15. The Splash 2 topology [Arnold92]. The linear array of FPGAs (**A-H**) is augmented by a routing-only crossbar (**R**). Note that the real topology has 16 FPGAs in the linear array.

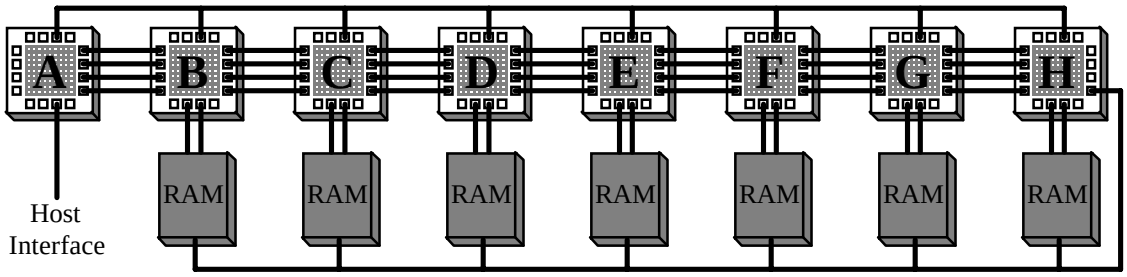

Figure 16. The Anyboard topology [Thomae91, Van den Bout92]. The linear array of FPGAs is augmented with a global bus.

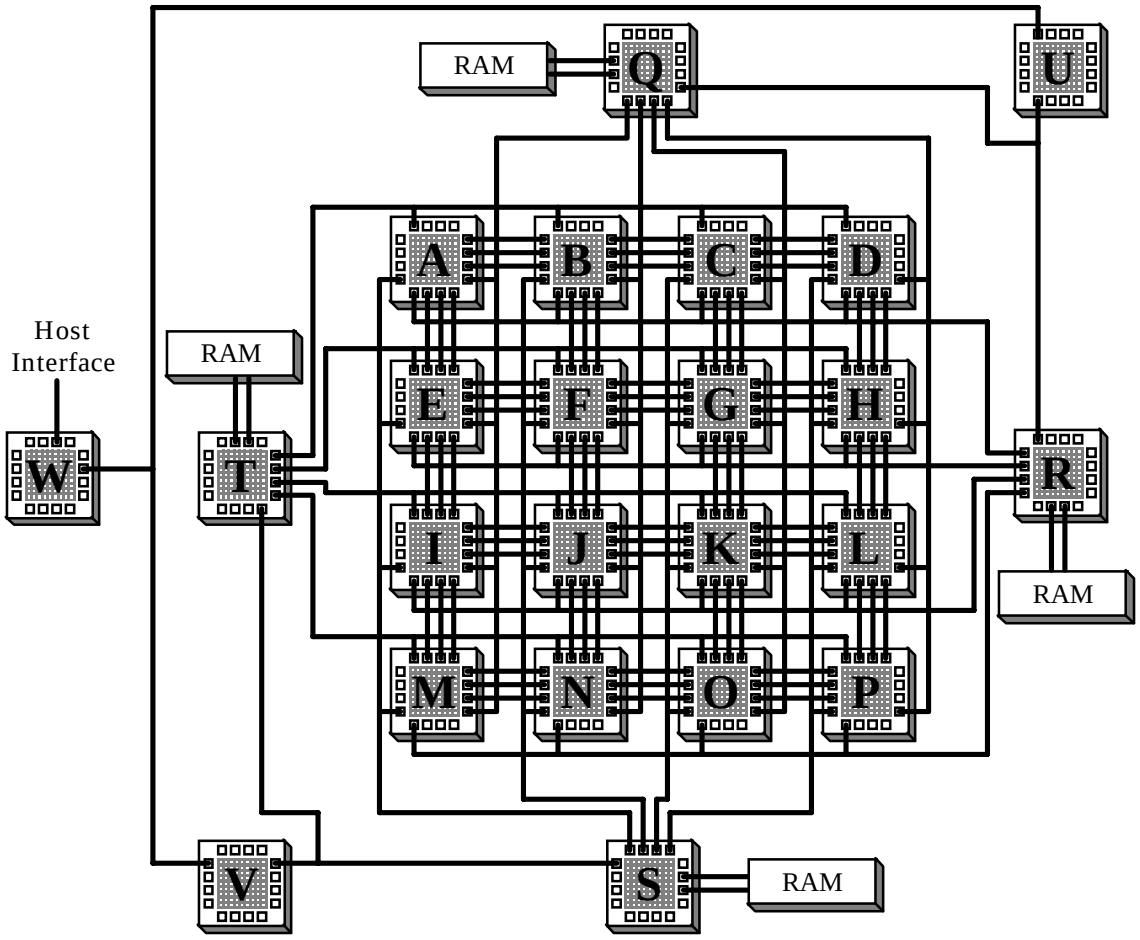

Figure 17. The DECPeRLe-1 topology [Vuillemin96]. The central 4x4 mesh (**A**-**P**) is augmented with global buses to four support FPGAs (**Q**-**T**), which feed to three other FPGAs (**U**-**W**).

Some systems are more of a hybrid between different topologies than a single topology. One example of this is the Splash 2 architecture [Arnold92]. The Splash 2 machine takes the linear array of its predecessor Splash [Gokhale90], and augments it with a crossbar interconnecting the FPGAs (Figure 15). In this way the system can still efficiently handle the systolic circuits that Splash was built to support, since it retains the nearest-neighbor interconnect, but the crossbar supports more general communication patterns, either in support of or as replacement to the direct interconnections. There are several other augmented linear arrays [Benner94, Box94, Darnauer94, Carrera95]. One example is the Anyboard [Thomae91, Van den Bout92], which has a linear array of FPGAs augmented by global buses (Figure 16). Another hybrid topology is the DECPeRLe-1 board [Vuillemin96], which has a 4x4 mesh of FPGAs augmented with shared global buses going to four support FPGAs (Figure 17). These are then connected to three other FPGAs that handle global routing, connections to memory, and the host interface. The DECPeRLe-0 board is similar [Bertin93].

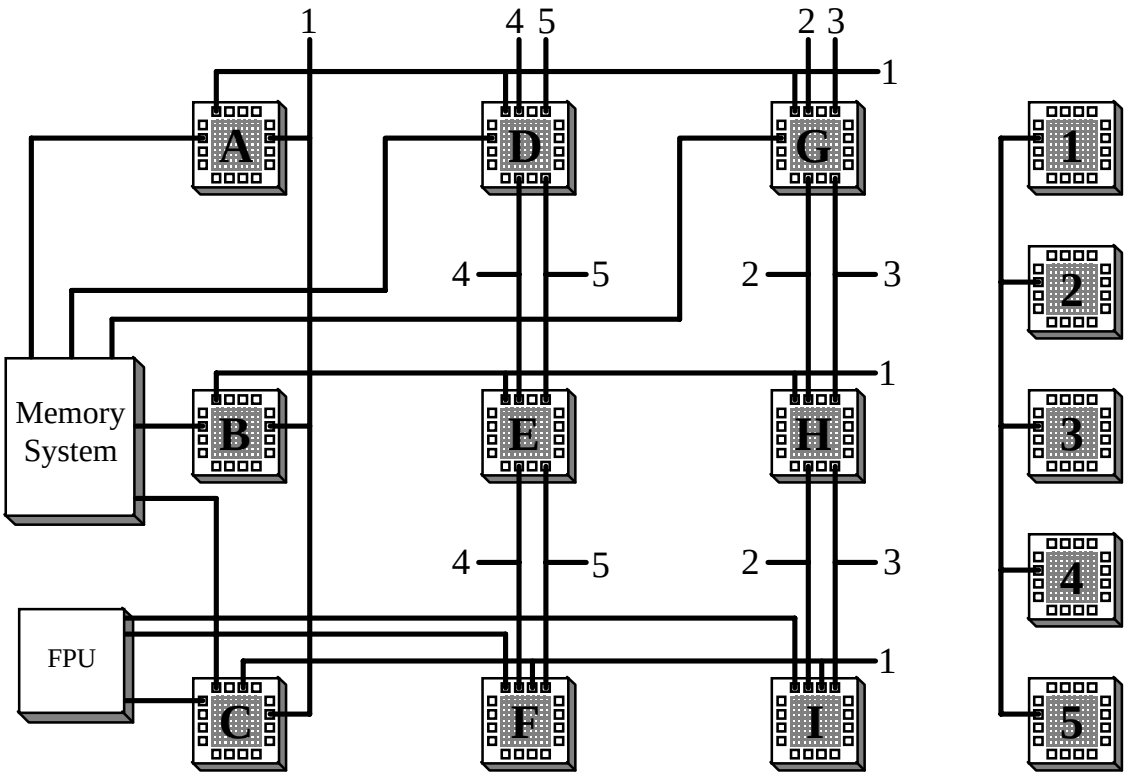

Figure 18. The Marc-1 topology [Lewis93]. The complete topology consists of two copies of the left subsystem (**A**-**I**, Memory & FPU), and one copy of the right (**1**-**5**). Numbers by themselves indicate connections to the FPGAs at right.

Some topologies are unique [Engels91, Cox92, Erdogan92, Suganuma92, Casselman93, Herpel93, Iseli93, Lewis93, Wazlowski93, Howard94a, Jantsch94, Nguyen94, Saluvere94, Dunn95, Hayashi95, Herple95, Högl95, Van den Bout95, Bakkes96, Knittel96]. These are often machines primarily built for a specific application, and their topology is optimized for the features of that application domain. For example, the Marc-1 (Figure 18) [Lewis93] is a pair of 3x3 meshes of FPGAs (**A**-**I**), where most of the mesh connections link up to a set of FPGAs intended to be used as crossbars (**1**-**5**). While the vertical links are nearest-neighbor, the horizontal links are actually buses. There is a complex memory system attached to some of the FPGAs in the system, as well as a floating point unit. This machine architecture was constructed for a specific application - circuit simulation (and other algorithms) where the program to be executed can be optimized on a per-run basis for values constant within that run, but which may vary from dataset to dataset. For example, during circuit simulation, the structure of the gates in the circuit is fixed once the program begins executing, but can be different for each run of the simulator. Thus, if the simulator can be custom compiled on a per-run basis for the structure of the circuit being simulated, there is the potential for significant speedups. Because of the restricted domain of compiled-code execution, the topology was built to contain a special-purpose processor, with the instruction unit in FPGAs **A**-**C**, and the datapath in the FPGAs **D**-**I**.

Another example of a system optimized for a specific task is the RM-nc system [Erdogan92]. As shown in Figure 19, the system has a controller chip **A**, and two linear arrays of FPGAs **B**-**E** and **F**-**I**. Each FPGA has a local memory for buffering data. The system contains three major buses, with the outside buses going to only one linear array each, while the center bus is shared by all FPGAs in the system. This system is optimized for neural network simulation, with the controller chip handling global control and sequencing, while the FPGAs in the linear array handle the individual neuron computations. A similar system for neural-network simulation, with buses going to all FPGAs in the system, and with no direct connections between neighbors, has also been built [Eldredge94].

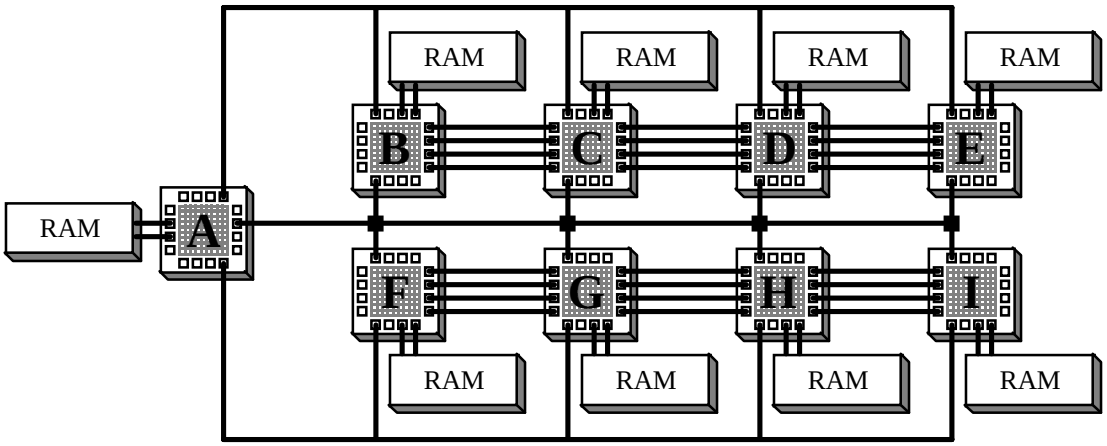

Figure 19. The RM-nc system [Erdogan92]. The linear arrays can be grown larger than the system shown.

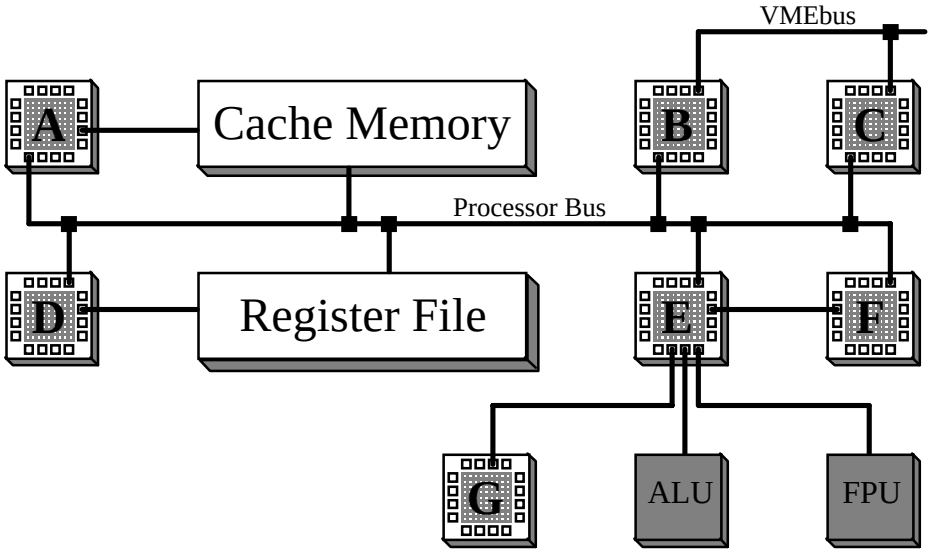

Figure 20. System similar to the Mushroom processor prototyping system [Williams91].

One of the most common domains for the development of a custom multi-FPGA system is the prototyping of computers. A good example of this is the Mushroom system [Williams91], which is a system of FPGAs, memories, and computation chips meant to be used for processor prototyping (the exact interconnection pattern is unavailable, but an approximate version is shown in Figure 20). Seven FPGAs are included in the system, and each of these has a specific role. FPGA **A** is a cache controller, **B** & **C** serve as a VMEbus interface, **D** handles register accesses, **E** is for processor control, **F** performs instruction fetches, and **G** is for tag manipulation and checking. Memories are included for caches and the register file, while an ALU and FPU chip are present for arithmetic operations. The advantage of this system is that the FPGAs can be reprogrammed to implement different functions, allowing different processor structures to be explored. The arithmetic functions and memory, features that are inefficient to implement in FPGAs and which are standard across most processors, are implemented efficiently in chips designed specifically for these applications. Other systems have adopted a similar approach, including another system intended for developing application-specific processors [Wolfe88], a workstation design with FPGAs for I/O functions and for coprocessor development [Heeb93], and a multiprocessor system with FPGA-based cache controllers for exploring different multiprocessor systems and caching policies [Öner95]. A somewhat related system is the CM-2X [Cuccaro93], a standard CM-2 supercomputer with the floating-point unit replaced with a Xilinx FPGA. This system allows custom coprocessors

to be built on a per-algorithm basis, yielding significant performance increases for some non-floating-point intensive programs.

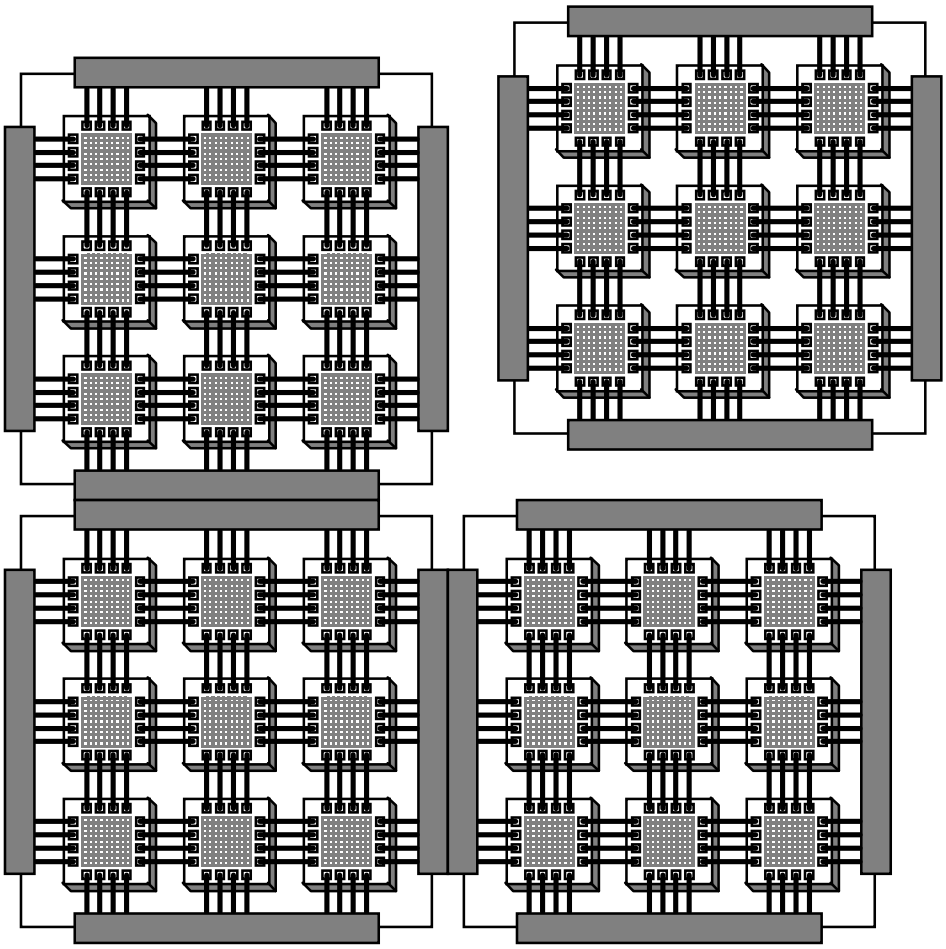

Figure 21. Expandable mesh topology similar to the Virtual Wires Emulation System [Tessier94]. Individual boards are built with edge connectors and limited logic resources, and can be interconnected to form a larger mesh.

There are important features of multi-FPGA systems beyond just the interconnect topology. One of these is the ability to increase the size of the multi-FPGA system. As described previously, systems based on hierarchical crossbars [Varghese93] can grow in size by adding extra levels of hierarchy. Other systems provide similar expandability, with special interconnection patterns for multi-board systems [Amerson95, Högl95]. In the case of meshes and linear arrays, the systems can be built of a basic tile with external connectors and limited on-board logic resources (Figure 21), and the system can be expanded by connecting together several of these boards [Thomae91, Arnold92, Van den Bout92, Filloque93, Shaw93, Hauck94b, Tessier94, Drayer95, Bakkes96]. The GERM system [Dollas94] is similar, except that the four external connectors are built to accommodate ribbon cables. In this way, fairly arbitrary topologies can be created.

Multi-FPGA systems are often composed of several different types of chips. The majority of systems are built from Xilinx 3000 or 4000 series FPGAs [Xilinx94], though most commercial FPGAs have been included in at least one multi-FPGA system. There are some system designers that have chosen to avoid commercial FPGAs, and develop their own chips. Some have optimized their FPGAs for specific applications, such as general custom-computing and logic emulation [Amerson95, Amerson96], emulation of telecommunication circuits [Hayashi95], or image processing [Quénot94]. Another system uses FPGAs optimized for inter-chip communication on an

MCM substrate [Dobbelaere92], since MCMs will be used to fabricate the system to achieve higher density circuits.

Reconfigurable systems do not need to use traditional FPGAs as their main computation unit, but can instead develop new FPGA-like architectures to serve the same role. These architectures can take advantage of architectural features from other computation domains, such as DSPs, multiprocessors, systolic arrays, and other systems to provide a better resource structure than standard commercial general-purpose FPGA architectures. Examples include the Pegasus system [Maliniak94], which uses a hybrid processor/FPGA chip with multiple configurations, and MATRIX [Mirsky96], a more coarse-grained architecture with similarities to SIMD processor systems.

Many multi-FPGA systems include non-FPGA chips. By far the most common element to be included is memory chips. These chips are usually connected to the FPGAs, and are used as temporary storage for results, as well as general-purpose memories for circuit emulation. Other systems have included integer and/or floating point ALUs [Wolfe88, Williams91, Lewis93, Benner94, Bakkes96, Knittel96], DSPs [Engels91, Bergmann94, vom Bögel94, Zycad94, Pottinger95], and general-purpose processors [Filloque93, Shaw93, Raimbault93, Benner94, Koch94b, vom Bögel94, Zycad94] to handle portions of computations where dedicated chips perform better than FPGA solutions. Another common inclusion into a multi-FPGA system is crossbars or FPICs. For example, a multi-FPGA system with a crossbar or hierarchical crossbar topology requires chips purely for routing. An FPIC or crossbar chip can handle these roles. If the FPIC or crossbar has a lower unit cost, is capable of higher performance or higher complexity routing, or has a larger number of I/O pins, then it can implement the routing functions better than a purely FPGA-based solution. There have been several systems that have used crossbar chips or FPICs in crossbar [Kadi94, Weiss94] and hierarchical crossbar topologies [Varghese93], as well as hybrid crossbar/linear array topologies [Arnold92, Darnauer94, Carrera95] and other systems [Adams93, Aptix93b, Casselman93, Galloway94, Njølstad94, Herpel95, Högl95, Van den Bout95].

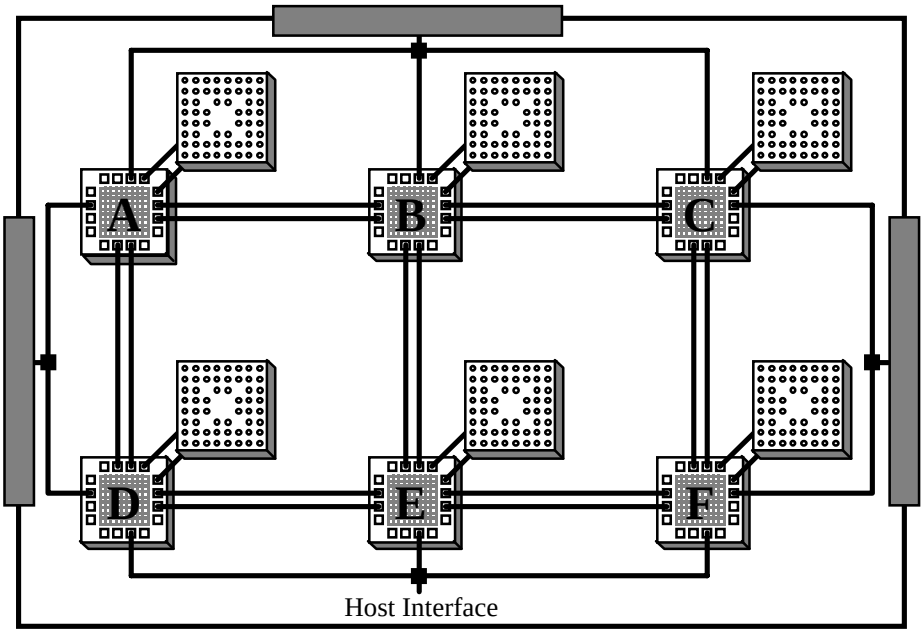

Figure 22. The MORRPH topology [Drayer95]. Sockets next to the FPGAs allow arbitrary devices to be added to the array. Buses connected to external connectors allow multiple boards to be hooked together to build an arbitrarily large system.

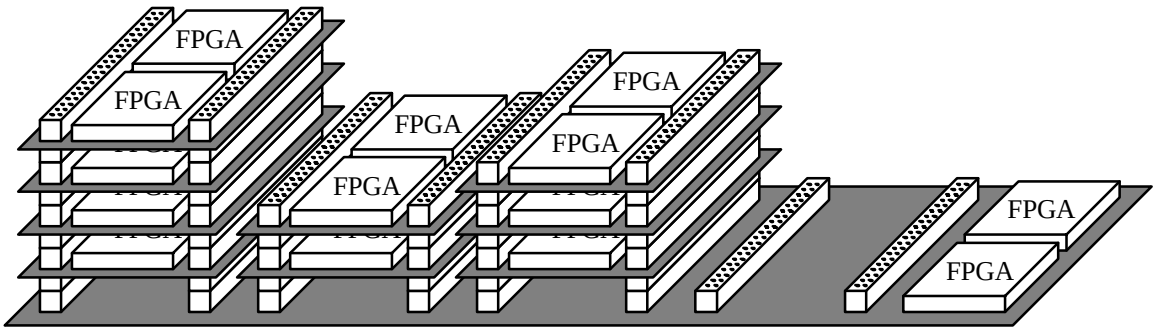

Figure 23. The G800 board [Giga95]. The base board has two FPGAs and four sockets. The socket at left holds four computing module boards (the maximum allowed in a socket), while the socket at right has none.

While adding fixed, non-FPGA resources into a multi-FPGA topology may improve quality for some types of mappings, it is possible to generate a more general-purpose solution by allowing the inclusion of arbitrary devices into the array. For example, in the MORRPH topology [Drayer95] sockets are placed next to the FPGAs so that arbitrary devices can be inserted (Figure 22). Thus, in a mapping that requires extra memory resources, memories can be plugged into the array. In other circumstances, DSPs or other fixed-function chips can be inserted to perform complex computations. In this way, the user has the flexibility to customize the array on a per-mapping basis. Other systems have adopted a similar strategy [Butts91, Sample92, Aptix93b], with a limited ability to insert arbitrary chips into the multi-FPGA system.

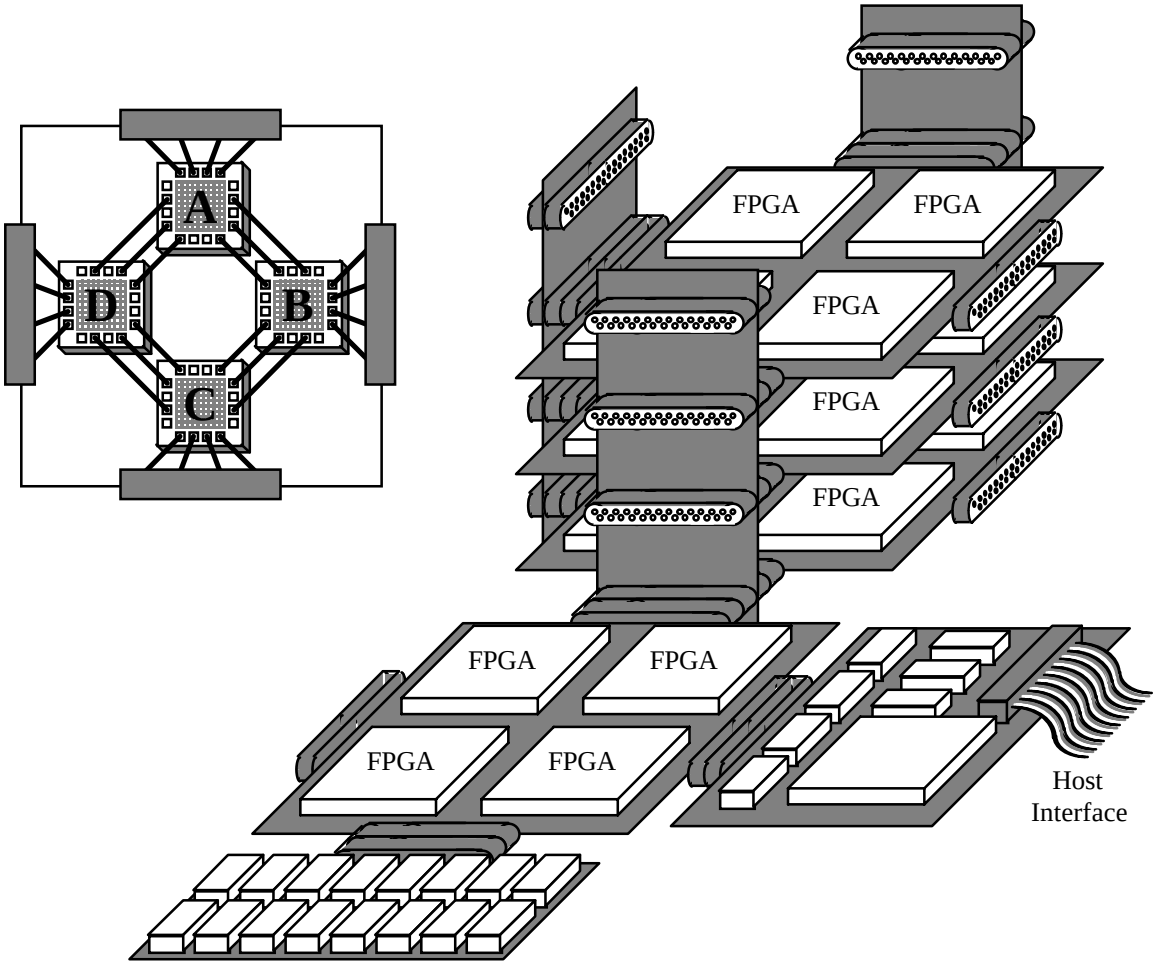

Figure 24. Base module (upper left) and example topology built in the COBRA system [Koch94b]. The mapping includes a tower of three base modules surrounded by three bus modules (top), a base module (center), a RAM module (bottom), and an I/O module (right).

Some systems are more of an infrastructure for bringing to bear the best mix of resources rather than a specific, fixed multi-FPGA system. One example of this is the G800 system [Giga95]. The board contains two FPGAs, some external interface circuitry, and four locations to plug in compute modules (Figure 23). Compute modules are cards that can be added to the system to add computation resources, and can be stacked four deep on the board. Thus, with four locations that can be stacked four deep, a total of 16 compute boards can be combined into a single system. These compute boards are connected together, and to the FPGAs on the base board, by a set of global buses. Compute modules can contain an arbitrary mix of resources. Examples include the X210MOD-00, which has two medium Xilinx FPGAs, and the X213MOD-82, which has two large FPGAs, 8MB of DRAM, and 256 KB of SRAM. The users of the system are free to combine whatever set of modules they desire, yielding a system capacity ranging from only two medium FPGAs (a single X210MOD-00), to 32 large FPGAs (16 X213MOD-82s) and a significant RAM capacity. Similar systems include DEEP [vom Bögel94] and Paradigm RP [Zycad94]. The Paradigm RP has locations to plug in up to eight boards. These boards can include FPGAs, memories, DSPs, and standard processors. In the G800 and Paradigm RP systems, cards with new functions or different types of chips can easily be accommodated.

Even more flexible systems are possible. One example is the COBRA system [Koch94b]. As shown in Figure 24, the system is made up of several types of modules. The standard *base module* has four FPGAs, each attached to an external connection at one of the board's edges. Boards can be attached together to build a larger system, expanding out in a 2D mesh. Other module types can easily be included, such as modules bearing only RAM, or a host interface, or a standard processor. These modules attach together the same way as the base module, and will

have one to four connectors. One somewhat different type of module is a bus module. This module stacks other modules vertically, connecting them together by a bus.

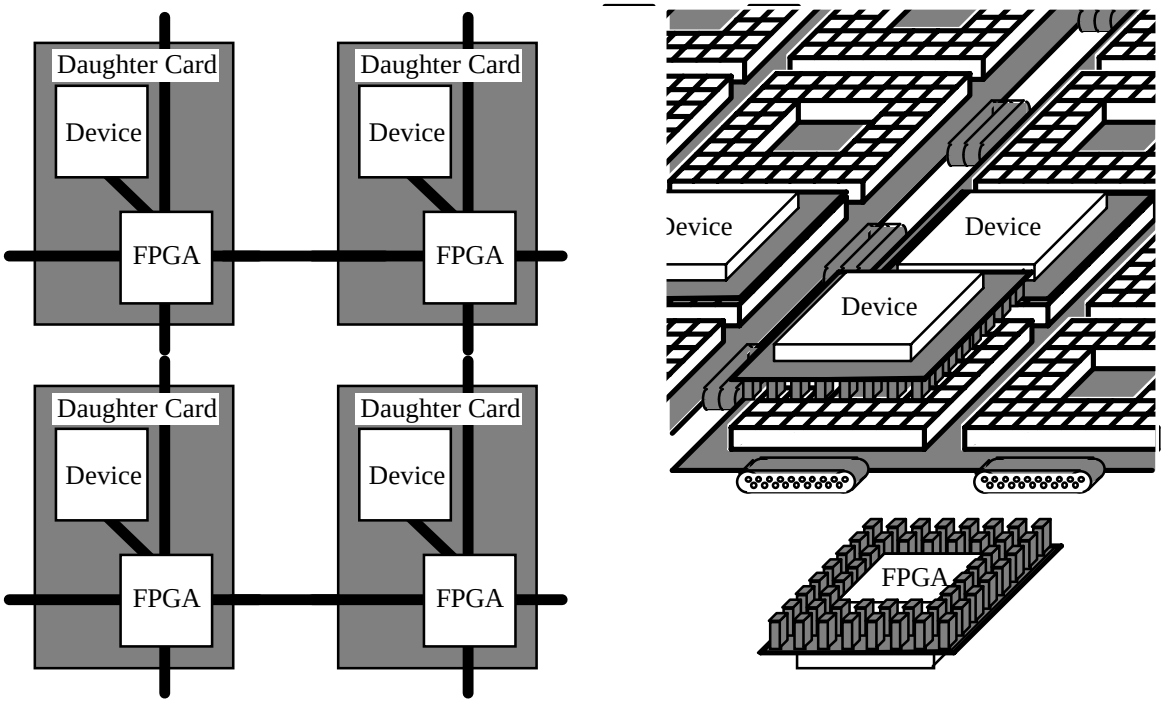

Figure 25. The Springbok interconnection pattern (left), and two connected Springbok baseplates with four daughter cards (right). The card at front is similar to the other daughter cards (with an FPGA on one side and an arbitrary device on the other), but is shown upside-down.

Systems like COBRA allow an arbitrary resource mix to be brought to bear on a problem. Thus, a custom processing system can quickly be built for a given task, with FPGAs handling the general logic, while standard processors handle complex computations and control flow. Thus, the proper resource is always used for the required computation. One of the earliest such systems, which combines COBRA's flexible topology with MORRPH's ability to add arbitrary devices, is the Springbok system [Hauck94b, Hauck95b]. Springbok is designed to support the rapid-prototyping of board-level designs, where numerous premade components must efficiently be embedded in an FPGA-based structure. The FPGAs in the system are used both for routing and rerouting of signals, as well as the implementation of random logic (Figure 25). To allow a specific circuit to be implemented in this structure, the system is comprised of a baseplate with sites for daughter cards. The daughter cards are large enough to contain both an arbitrary device on the top, as well as an FPGA on the bottom. Note that the device can be a chip, such as a processor or memory, I/O elements such as switches and LCD interfaces, or whatever else is necessary to implement the system. If necessary, daughter cards can be built that span several locations on the baseplate to handle higher space or bandwidth requirements. The daughter cards plug into the baseplate, which handles power, ground, and clock distribution, FPGA programming, and inter-daughter card routing. The baseplates include support for communicating with a host computer, both for downloading programming and for uploading data captured during prototype runs. The baseplates are constructed such that they can be connected together with each other, forming an arbitrarily large surface for placing daughter cards. In many ways, this approach is similar both to mesh-connected multiprocessors, as well as the approach suggested in [Seitz90].

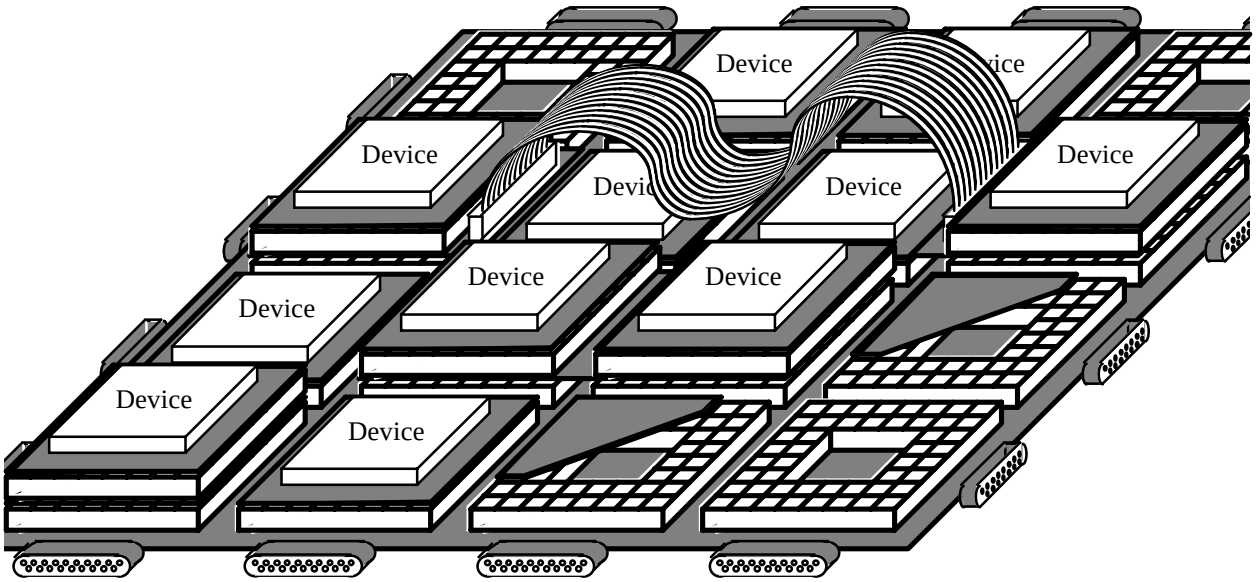

Figure 26. Non-device daughter cards and extender cards, including cards to add more FPGA logic (lower left), bandwidth (double-sized card in the second row), long-distance communication (third row), and edge bandwidth (two cards at front right). All but the edge bandwidth cards have FPGAs on the bottom.

As in most other FPGA systems, there is the potential that Springbok's simple connection scheme will not be able to accommodate all of the logic or routing assigned to a given location. However, as opposed to a fixed multi-FPGA system, Springbok can insert new extender cards between a daughter card and the baseplate to deal with these problems (Figure 26). For example, if the logic assigned to a given FPGA simply will not fit, an extender card with another FPGA can be inserted so that it can handle some of the logic. If too many signals need to be routed along a given link in the mesh structure, an extender card spanning several daughter card positions can be added, with new routing paths included on the inserted card. For signals that must go long distances in the array, sets of extender cards with ribbon cable connections can be inserted throughout the array to carry these long-distance wires. Also, at the edge of the mapping where edge effects can limit available bandwidth, dummy daughter cards which simply contain hardwired connections between their neighbors can be inserted. Thus, the Springbok approach to resource limitations is to add resources wherever necessary to map the system. In contrast, a fixed array cannot afford a failure due to resource limitations, since it would then have to redo the costly step of mapping to all of the constituent FPGAs. Thus fixed arrays must be very conservative on all resource assignments, underutilizing resources throughout the system, while Springbok simply fixes the problems locally as they arise.

While Springbok and Cobra provide an FPGA-intensive approach to the prototyping of board-level designs, another approach is possible. Instead of relying on FPGAs to handle the inter-chip routing, FPICs can be used in their place. In the systems developed by Aptix Corporation [Aptix93a, Aptix93b], the user is provided with a Field-Programmable Circuit Board (FPCB). The board is simply a sea of holes, similar to a wire-wrap board. However, instead of requiring the user to manually complete the wiring like a wire-wrap system, all the holes in an FPCB are connected to FPICs. These FPICs serve as central, programmable routing hubs. If there is more than one FPIC on the FPCB, the FPCB's holes are divided up among the FPICs, with each hole connecting up to a specific FPIC. Connections between holes going to the same FPIC are handled by that FPIC, while connections between holes going to different FPICs requires using one of a limited number of inter-FPIC wires. Thus, simply by programming the FPICs the connection patterns between chips in the prototype can be made, and later alterations are much simpler than in a non-programmable system.

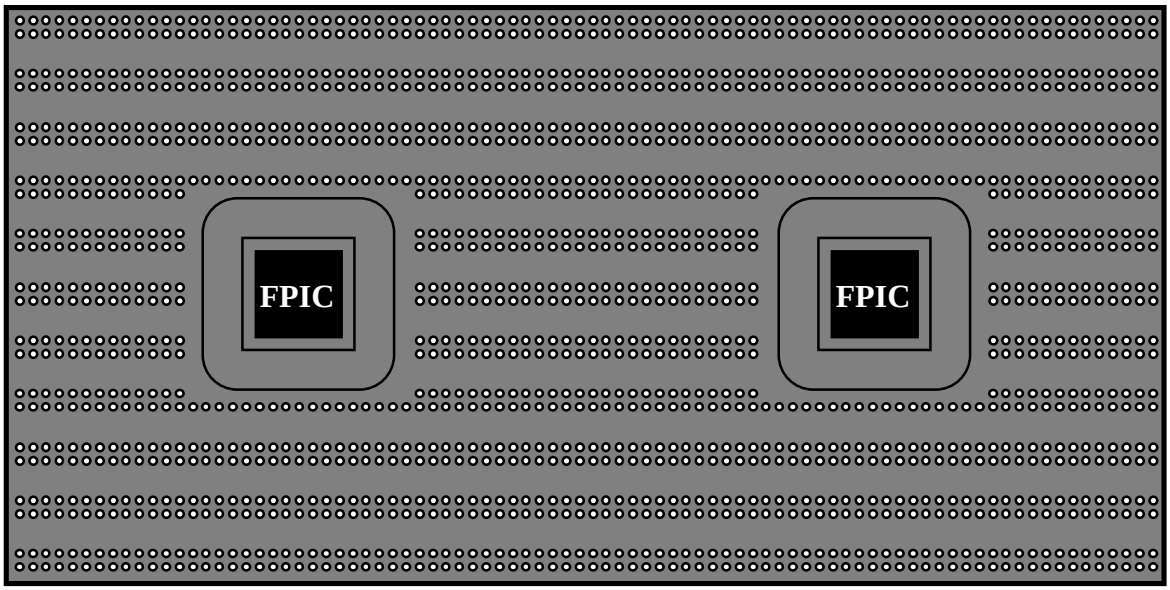

Figure 27. The Aptix FPCB. Holes in each of the two shaded regions are connected to the corresponding FPIC.

6. The Challenges of Reprogrammable Systems

In the past ten years FPGAs have grown from a simple glue-logic replacement to the basis of a huge range of reprogrammable systems. We have seen multi-FPGA systems deliver world record performance for many key applications, and logic emulation systems raise testing performance by several orders of magnitude. Multi-mode hardware has reduced inventory risk and development time for many standard logic applications, providing truly generic hardware systems. However, while these examples have shown much of the promise of reprogrammable systems, there is much yet to be done to reach their full potential.

While this paper has focused on reprogrammable hardware systems, many of the problems are actually in the software used to map circuits and algorithms onto these systems. While simple one or two-chip multi-mode systems can easily be developed by hardware designers using current tool suites, for multi-FPGA systems with hundreds or thousands of reprogrammable components the job becomes much more complex. Even worse, many of these systems are targeted towards general computation, where the users will not be skilled hardware designers, but will instead be software programmers. Thus, one of the keys to unlocking the full potential of these systems is developing truly automatic mapping tools. This software must be able to quickly and efficiently map logic descriptions into the appropriate configuration files. Not only must this result in fast and compact implementations, but it must be able to generate these mappings very quickly. Traditional mapping tools designed for custom chip implementations, where the fabrication time can be measured in weeks to months, can take hours or days to complete their tasks. For many reprogrammable systems, especially logic emulation systems where alterations to accommodate bug fixes will be common, waiting hours or days for the software to complete is unreasonable. Any performance gain from executing on a reprogrammable system will be swallowed up by the time necessary to map to the system in the first place. A survey of software for reprogrammable systems can be found elsewhere [Hauck97].

In order to drive future deployment of reconfigurable systems, commercial “killer applications” for this technology must also be developed. Although there have been many impressive demonstrations of reconfigurable systems, it is not clear that any application has already been developed which can drive wide-scale adoption of this technology. The one major exception to this is logic emulation, which has created a viable commercial market for FPGA-based compute engines, but much of the more advanced possibilities of reconfigurable computing have not yet reached the end user. Finding these applications, and creating compelling implementations that are ready for immediate commercial use, will be another key to the success of reconfigurable computing.

Another significant issue in the design of reprogrammable systems is the structure of the components used in these systems. Primarily, current reprogrammable systems are built from FPGAs that were designed to handle standard logic implementation tasks. Thus, these chips have not been optimized for the needs of reprogrammable systems. This mismatch can cause significant inefficiencies. For example, the time to program an FPGA isn't that important for standard applications, but a reprogrammable system may need to context switch often in some environments, making the reconfiguration time a critical issue. This is especially important for techniques such as run-time reconfiguration. Run-time reconfiguration views an FPGA as a logic cache, paging in only those portions of logic needed to execute at a given time, and thus the reprogramming time is important. Standard FPGAs also may lack hardware support for some common structures in reprogrammable systems. While there are FPGAs which support internal RAMs [Xilinx94, Wilton95], other constructs such as multiplication and floating point arithmetic cannot be efficiently accommodated in today's FPGAs. Finally, the I/O capacities of current FPGAs are much lower than what is demanded by reprogrammable systems. For example, current logic emulation systems often achieve only 10%-20% logic utilization because there are simply too few I/O pins available.

The temptation is to develop FPGAs specifically for reprogrammable systems, and as mentioned before, several groups have done specifically that. However, whether it is cost-effective to develop a custom FPGA, and no longer take advantage of the economies of scale provided by using standard chips, is an open question. This issue is even more crucial for FPICs. Since FPICs in general are only useful for reprogrammable systems, and may require non-standard packaging to achieve high pin counts, they are quite expensive. Whether their advantages are worth the added cost is unclear, especially when FPGA manufacturers are providing ever higher pincount packages on their standard products.

A final issue is whether some of the new directions in reprogrammable system research are viable. Specifically, there is interest in moving reprogrammable logic much closer to the processors in standard computers. This can involve generic coprocessors built from FPGAs, or even reprogrammable functional units inside the processors themselves. The argument is that by tightly coupling reprogrammable logic with the processor one can add custom instructions to the instruction set for a given application, yielding application-specific processor performance in an application-generic manner. While this model has much promise, it is unclear whether these opportunities exist in practice, whether the benefits will be worth the chip resources consumed, and whether we can even develop compilers capable of recognizing and optimizing for these opportunities.

7. Conclusions

Reprogrammable systems have provided significant performance improvements for many types of applications. Logic emulation systems are providing orders of magnitude better performance than software simulation, providing virtual prototypes without the delays of creating true prototype designs. FPGA-based custom-computing machines have achieved world-record performance for many applications, providing near application-specific performance in an application-generic system. Finally, multi-mode systems reduce the complexity and inventory risk for logic implementation, providing truly generic hardware systems

The hardware structures used to achieve these results are quite varied. Built from general-purpose FPGAs, or perhaps even custom reprogrammable devices, they differ greatly in structure and complexity. Some are one or two-chip systems providing small logic capacities. Others are capable of expanding up to a system of hundreds or thousands of FPGAs, providing a huge capacity for high-performance logic implementation. There is also great variation in the types of resources employed, with some systems including memories, DSPs, or CPUs, and others even allowing the user to include arbitrary devices into the system.

While these systems provide high-performance implementations, there are several problems with today's systems. Mapping software is critical to widespread deployment of these systems, yet current software is much slower and provides lower quality than is often required. Also, current systems primarily use off-the-shelf FPGAs, which while they provide some economies of scale they also present the system designer to significant inefficiencies.

There is much left to be done with reprogrammable systems, with much promise yet to be fulfilled. Reprogrammable functional units in standard processors may yield application-specific processors in an

application-generic manner. Reprogrammable coprocessors could become generic coprocessors, yielding custom accelerators for arbitrary applications, including niche needs for which custom hardware will never be cost-effective. In these ways FPGAs may become the cornerstone of many future computation systems.

Acknowledgements

This research was funded in part by DARPA contract DABT63-97-C-0035 and NSF grants CDA-9703228 and MIP-9616572.

List of References

[Actel94] *FPGA Data Book and Design Guide*, Sunnyvale, CA: Actel Corp, 1994.

[Adams93] J. K. Adams, H. Schmitt, D. E. Thomas, "A Model and Methodology for Hardware-Software Codesign", *International Workshop on Hardware-Software Codesign*, 1993.

[Agarwal94] L. Agarwal, M. Wazlowski, S. Ghosh, "An Asynchronous Approach to Efficient Execution of Programs on Adaptive Architectures Utilizing FPGAs", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 101-110, 1994.

[Albaharna94] O. T. Albaharna, P. Y. K. Cheung, T. J. Clarke, "Area & Time Limitations of FPGA-based Virtual Hardware", *International Conference on Computer Design*, pp. 184-189, 1994.

[Albaharna96] O. T. Albaharna, P. Y. K. Cheung, T. J. Clarke, "On the Viability of FPGA-based Integrated Coprocessors", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1996.

[Amerson95] R. Amerson, R. J. Carter, W. B. Culbertson, P. Kuekes, G. Snider, "Teramac - Configurable Custom Computing", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[Amerson96] R. Amerson, R. Carter, W. Culbertson, P. Kuekes, G. Snider, "Plasma: An FPGA for Million Gate Systems", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 10-16, 1996.

[Aptix93a] Aptix Corporation, *Data Book*, San Jose, CA, February 1993.

[Aptix93b] Aptix Corporation, *Data Book Supplement*, San Jose, CA, September 1993.

[Arnold92] J. M. Arnold, D. A. Buell, E. G. Davis, "Splash 2", *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 316-322, 1992.

[Athanas93] P. M. Athanas, H. F. Silverman, "Processor Reconfiguration Through Instruction-Set Metamorphosis", *IEEE Computer*, Vol. 26, No. 3, pp. 11-18, March, 1993

[Bakkes96] P. J. Bakkes, J. J. du Plessis, B. L. Hutchings, "Mixing Fixed and Reconfigurable Logic for Array Processing", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1996.

[Benner94] T. Benner, R. Ernst, I. Könenkamp, U. Holtmann, P. Schüler, H.-C. Schaub, N. Serafimov, "FPGA Based Prototyping for Verification and Evaluation in Hardware-Software Cosynthesis", in R. W. Hartenstein, M. Z. Servít, Eds., *Lecture Notes in Computer Science 849 - Field-Programmable Logic: Architectures, Synthesis and Applications*, Berlin: Springer-Verlag, pp. 251-258, 1994.

[Bergmann94] N. W. Bergmann, J. C. Mudge, "Comparing the Performance of FPGA-Based Custom Computers with General-Purpose Computers for DSP Applications", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 164-171, 1994.

[Bertin89] P. Bertin, D. Roncin, J. Vuillemin, "Introduction to Programmable Active Memories", in J. McCanny, J. McWhirter, E. Swartzlander Jr., Eds., *Systolic Array Processors*, Prentice Hall, pp. 300-309, 1989.

[Bertin93] P. Bertin, "Mémoires actives programmables : conception, réalisation et programmation.", *Ph.D. Thesis, Université Paris 7 UFR D'INFORMATIQUE*, 1993.

[Blickle94] T. Blickle, J. König, L. Thiele, "A Prototyping Array for Parallel Architectures", in W. R. Moore, W. Luk, Eds., *More FPGAs*, Oxford, England: Abingdon EE&CS Books, pp. 388-397, 1994.

[Bolotski94] M. Bolotski, A. DeHon, T. F. Knight Jr., "Unifying FPGAs and SIMD Arrays", *2nd International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, 1994.

[Box94] B. Box, "Field Programmable Gate Array Based Reconfigurable Preprocessor", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 40-48, 1994.

[Brebner95] G. Brebner, J. Gray, "Use of Reconfigurability in Variable-length Code Detection at Video Rates", in W. Moore, W. Luk, Eds., *Lecture Notes in Computer Science 975 - Field-Programmable Logic and Applications*, London: Springer, pp. 429-438, 1995.

[Brown92] S. D. Brown, R. J. Francis, J. Rose, Z. G. Vranesic, *Field-Programmable Gate Arrays*, Boston, Mass: Kluwer Academic Publishers, 1992.

[Butts91] M. Butts, J. Batcheller, "Method of Using Electronically Reconfigurable Logic Circuits", *U.S. Patent 5,036,473*, July 30, 1991.

[Carrera95] J. M. Carrera, E. J. Martínez, S. A. Fernández, J. M. Chaus, "Architecture of a FPGA-based Coprocessor: The PAR-1", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[Casselman93] S Casselman, "Virtual Computing and the Virtual Computer", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 43-48, 1993.

[Chan92] P. K. Chan, M. Schlag, M. Martin, "BORG: A Reconfigurable Prototyping Board Using Field-Programmable Gate Arrays", *Proceedings of the 1st International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, pp. 47-51, 1992.

[Chan94] P. K. Chan, S. Mourad, *Digital Design Using Field Programmable Gate Arrays*, Englewood Cliffs, New Jersey: PTR Prentice Halls, 1994.

[Chan95] P. K. Chan, M. D. F. Schlag, J. Y. Zien, "Spectral-Based Multi-Way FPGA Partitioning", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 133-139, 1995.

[Churcher95] S. Churcher, T. Kean, B. Wilkie, "The XC6200 FastMap Processor Interface", in W. Moore, W. Luk, Eds., *Lecture Notes in Computer Science 975 - Field-Programmable Logic and Applications*, London: Springer, pp. 36-43, 1995.

[Clark96] D. A. Clark, B. L. Hutchings, "The DISC Programming Environment", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1996.

[Cowen94] C. P. Cowen, S. Monaghan, "A Reconfigurable Monte-Carlo Clustering Processor (MCCP)", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 59-65, 1994.

[Cox92] C. E. Cox, W. E. Blanz, "GANGLION - A Fast Field-Programmable Gate Array Implementation of a Connectionist Classifier", *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 3, pp. 288-299, March, 1992.

[Cuccaro93] S. A. Cuccaro, C. F. Reese, "The CM-2X: A Hybrid CM-2 / Xilinx Prototype", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 121-130, 1993.

[Darnauer94] J. Darnauer, P. Garay, T. Isshiki, J. Ramirez, W. W.-M. Dai, "A Field Programmable Multi-chip Module (FPMCM)", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 1-10, 1994.

[DeHon94] A. DeHon, "DPGA-Coupled Microprocessors: Commodity ICs for the Early 21st Century", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 31-39, 1994.

[DeHon96] A. DeHon, "DPGA Utilization and Application", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 115-121, 1996.

[Dobbelaere92] I. Dobbelaere, A. El Gamal, D. How, B. Kleveland, "Field Programmable MCM Systems - Design of an Interconnection Frame", *First International ACM/SIGDA Workshop on Field Programmable Gate Arrays*, pp. 52-56, 1992.

[Dollas94] A. Dollas, B. Ward, J. D. S. Babcock, "FPGA Based Low Cost Generic Reusable Module for the Rapid Prototyping of Subsystems", in R. W. Hartenstein, M. Z. Servít, Eds., *Lecture Notes in Computer Science 849 - Field-Programmable Logic: Architectures, Synthesis and Applications*, Berlin: Springer-Verlag, pp. 259-270, 1994.

[Drayer95] T. H. Drayer, W. E. King IV, J. G. Tront, R. W. Conners, "MORRPH: A MOdular and Reprogrammable Real-time Processing Hardware", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[Dunn95] P. Dunn, "A Configurable Logic Processor for Machine Vision", in W. Moore, W. Luk, Eds., *Lecture Notes in Computer Science 975 - Field-Programmable Logic and Applications*, London: Springer, pp. 68-77, 1995.

[Eldredge94] J. G. Eldredge, B. L. Hutchings, "Density Enhancement of a Neural Network Using FPGAs and Run-Time Reconfiguration", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 180-188, 1994.

[Engels91] M. Engels, R. Lauwereins, J. A. Peperstraete, "Rapid Prototyping for DSP Systems with Microprocessors", *IEEE Design & Test of Computers*, pp. 52-62, June 1991.

[Erdogan92] S. S. Erdogan, A. Wahab, "Design of RM-nc: A Reconfigurable Neurocomputer for Massively Parallel-Pipelined Computations", *International Joint Conference on Neural Networks*, Vol. 2, pp. 33-38, 1992.

[Fawcett94] B. Fawcett, "Applications of Reconfigurable Logic", in W. R. Moore, W. Luk, Eds., *More FPGAs*, Oxford, England: Abingdon EE&CS Books, pp. 57-69, 1994.

[Ferrucci94] A. Ferrucci, M. Martín, T. Geocaris, M. Schlag, P. K. Chan, "EXTENDED ABSTRACT: ACME: A Field-Programmable Gate Array Implementation of a Self-Adapting and Scalable Connectionist Network", *2nd International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, 1994.

[Filloque93] J. M. Filloque, C. Beaumont, B. Pottier, "Distributed Synchronization on a Parallel Machine with a Logic Layer", Euromicro Workshop on Parallel and Distributed Computing, pp. 53-58, 1993.

[French93] P. C. French, R. W. Taylor, "A Self-Reconfiguring Processor", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 50-59, 1993.

[Galloway94] D. Galloway, D. Karchmer, P. Chow, D. Lewis, J. Rose, "The Transmogrifier: The University of Toronto Field-Programmable System", *Second Canadian Workshop on Field-Programmable Devices*, 1994.

[Gateley94] J. Gateley, "Logic Emulation Aids Design Process", *ASIC & EDA*, July, 1994.

[Giga95] Giga Operations Corp, "G800 RIC", Berkeley, CA, 1995.

[Gokhale90] M. Gokhale, B. Holmes, A. Kopser, D. Kunze, D. Lopresti, S. Lucas, R. Minnich, P. Olsen, "Splash: A Reconfigurable Linear Logic Array", *International Conference on Parallel Processing*, pp. 526-532, 1990.

[Gokhale95] M. Gokhale, A. Marks, "Automatic Synthesis of Parallel Programs Targeted to Dynamically Reconfigurable Logic Arrays", in W. Moore, W. Luk, Eds., *Lecture Notes in Computer Science 975 - Field-Programmable Logic and Applications*, London: Springer, pp. 399-408, 1995.

[Graham95] P. Graham, B. Nelson, "A Hardware Genetic Algorithm for the Traveling Salesman Problem on Splash 2", in W. Moore, W. Luk, Eds., *Lecture Notes in Computer Science 975 - Field-Programmable Logic and Applications*, London: Springer, pp. 352-361, 1995.

[Graham96] P. Graham, B. Nelson, "Genetic Algorithms In Software and In Hardware - A Performance Analysis of Workstation and Custom Computing Machine Implementations", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1996.

[Greene93] J. Greene, E. Hamdy, S. Beal, "Antifuse Field Programmable Gate Arrays", *Proceedings of the IEEE*, Vol. 81, No. 7, pp. 1042-1056, July, 1993.

[Guccione95] S. A. Guccione, M. J. Gonzalez, "Classification and Performance of Reconfigurable Architectures", in W. Moore, W. Luk, Eds., *Lecture Notes in Computer Science 975 - Field-Programmable Logic and Applications*, London: Springer, pp. 439-448, 1995.

[Hadley95] J. D. Hadley, B. L. Hutchings, "Design Methodologies for Partially Reconfigured Systems", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[Hauck94a] S. Hauck, G. Borriello, C. Ebeling, "Mesh Routing Topologies for Multi-FPGA Systems", *International Conference on Computer Design*, pp. 170-177, October, 1994.

[Hauck94b] S. Hauck, G. Borriello, C. Ebeling, "Springbok: A Rapid-Prototyping System for Board-Level Designs", *ACM/SIGDA 2nd International Workshop on Field-Programmable Gate Arrays*, February, 1994.

[Hauck95a] S. Hauck, G. Borriello, C. Ebeling, "Achieving High-Latency, Low-Bandwidth Communication: Logic Emulation Interfaces", *University of Washington, Dept. of Computer Science & Engineering Technical Report #95-04-04*, January 1995.

[Hauck95b] S. Hauck, *Multi-FPGA Systems*, Ph.D. Thesis, University of Washington, Dept. of Computer Science & Engineering, 1995.

[Hauck97] S. Hauck, A. Agarwal, "Software Technologies for Reprogrammable Systems", submitted to *Proceedings of the IEEE*, 1997.

[Hayashi95] K. Hayashi, T. Miyazaki, K. Shirakawa, K. Yamada, N. Ohta, "Reconfigurable Real-Time Signal Transport System Using Custom FPGAs", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[Heeb93] B. Heeb, C. Pfister, "Chameleon: A Workstation of a Different Color", in H. Grünbacher, R. W. Hartenstein, Eds., *Lecture Notes in Computer Science 705 - Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping*, Berlin: Springer-Verlag, pp. 152-161, 1993.

[Herpel93] H.-J. Herpel, N. Wehn, M. Gasteier, M. Glesner, "A Reconfigurable Computer for Embedded Control Applications", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 111-120, 1993.

[Herpel95] H.-J. Herpel, U. Ober, M. Glesner, "Prototype Generation of Application Specific Embedded Controllers for Microsystems", in W. Moore, W. Luk, Eds., *Lecture Notes in Computer Science 975 - Field-Programmable Logic and Applications*, London: Springer, pp. 341-351, 1995.

[Hoang93] D. T. Hoang, "Searching Genetic Databases on Splash 2", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 185-191, 1993.

[Högl95] H. Högl, A. Kugel, J. Ludvig, R. Männer, K. H. Noffz, R. Zoz, "Enable++: A Second Generation FPGA Processor", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[Howard94a] N. Howard, "Defect-Tolerant Field-Programmable Gate Arrays", *Ph.D. Thesis, University of York, Department of Electronics*, 1994.

[Howard94b] N. Howard, A. Tyrrell, N. Allinson, "FPGA Acceleration of Electronic Design Automation Tasks", in W. R. Moore, W. Luk, Eds., *More FPGAs*, Oxford, England: Abingdon EE&CS Books, pp. 337-344, 1994.

[Huang95] D. J.-H. Huang, A. B. Kahng, "Multi-Way System Partitioning into a Single Type or Multiple Types of FPGAs", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 140-145, 1995.

[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.

[I-Cube94] I-Cube, Inc., "The FPID Family Data Sheet", Santa Clara, CA, February 1994.

[Iseli93] C. Iseli, E. Sanchez, "Spyder: A Reconfigurable VLIW Processor using FPGAs", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 17-24, 1993.

[Jantsch94] A. Jantsch, P. Ellervee, J. Öberg, A. Hemani, "A Case Study on Hardware/Software Partitioning", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 111-118, 1994.

[Jenkins94] J. H. Jenkins, *Designing with FPGAs and CPLDs*, Englewood Cliffs, New Jersey: PTR Prentice Hall, 1994.

[Jones95] C. Jones, J. Oswald, B. Schoner, J. Villasenor, "Issues in Wireless Video Coding Using Run-time-reconfigurable FPGAs", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[Kadi94] M. Slimane-Kadi, D. Brasen, G. Saucier, "A Fast-FPGA Prototyping System That Uses Inexpensive High-Performance FPIC", *ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, 1994.

[Kean92] T. Kean, I. Buchanan, "The Use of FPGAs in a Novel Computing Subsystem", *First International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, pp. 60-66, 1992.

[Knittel96] G. Knittel, "A PCI-Compatible FPGA-Coprocessor for 2D/3D Image Processing", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1996.

[Koch94a] A. Koch, U. Golze, "A Universal Co-processor for Workstations", in W. R. Moore, W. Luk, Eds., *More FPGAs*, Oxford, England: Abingdon EE&CS Books, pp. 317-328, 1994.

[Koch94b] G. Koch, U. Kebschull, W. Rosenstiel, "A Prototyping Environment for Hardware/Software Codesign in the COBRA Project", *Third International Workshop on Hardware/Software Codesign*, September, 1994.

[Kuznar93] R. Kuznar, F. Brglez, K. Kozminski, "Cost Minimization of Partitions into Multiple Devices", *Design Automation Conference*, pp.315-320, 1993.

[Kuznar94] R. Kuznar, F. Brglez, B. Zajc, "Multi-way Netlist Partitioning into Heterogeneous FPGAs and Minimization of Total Device Cost and Interconnect", *Design Automation Conference*, pp. 238-243, 1994.

[Lawrence95] A. Lawrence, A. Kay, W. Luk, T. Nomura, I. Page, "Using Reconfigurable Hardware to Speed up Product Development and Performance", in W. Moore, W. Luk, Eds., *Lecture Notes in Computer Science 975 - Field-Programmable Logic and Applications*, London: Springer, pp. 111-118, 1995.

[Lemoine95] E. Lemoine, D. Merceron, "Run Time Reconfiguration of FPGA for Scanning Genomic Databases", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[Lewis93] D. M. Lewis, M. H. van Ierssel, D. H. Wong, "A Field Programmable Accelerator for Compiled-Code Applications", *ICCD '93*, pp. 491-496, 1993.

[Li95] J. Li, C.-K. Cheng, "Routability Improvement Using Dynamic Interconnect Architecture", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[Ling93] X.-P. Ling, H. Amano, "WASMII: a Data Driven Computer on a Virtual Hardware", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 33-42, 1993.

[Lopresti91] D. P. Lopresti, "Rapid Implementation of a Genetic Sequence Comparator Using Field-Programmable Logic Arrays", *Advanced Research in VLSI 1991: Santa Cruz*, pp. 139-152, 1991.

[Luk96] W. Luk, N. Shirazi, P. Y. K. Cheung, "Modelling and Optimising Run-Time Reconfigurable Systems", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1996.

[Lysaght91] P. Lysaght, "Dynamically Reconfigurable Logic in Undergraduate Projects", in W. Moore, W. Luk, Eds., *FPGAs*, Abingdon, England: Abingdon EE&CS Books, pp. 424-436, 1991.

[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.

[Lysaght94b] P. Lysaght, J. Stockwood, J. Law, D. Girma, "Artificial Neural Network Implementation on a Fine-Grained FPGA", in R. W. Hartenstein, M. Z. Servít, Eds., *Lecture Notes in Computer Science 849 - Field-Programmable Logic: Architectures, Synthesis and Applications*, Berlin: Springer-Verlag, pp. 421-431, 1994.

[Lysaght95] P. Lysaght, H. Dick, G. McGregor, D. McConnell, J. Stockwood, "Prototyping Environment for Dynamically Reconfigurable Logic", in W. Moore, W. Luk, Eds., *Lecture Notes in Computer Science 975 - Field-Programmable Logic and Applications*, London: Springer, pp. 409-418, 1995.

[Maliniak94] L. Maliniak, "Hardware Emulation Draws Speed from Innovative 3D Parallel Processing Based on Custom ICs", *Electronic Design*, May 30, 1994.

[Mayrhofer94] J. Mayrhofer, J. Strachwitz, "A Control Circuit for Electical Drives based on Transputers and FPGAs", in W. R. Moore, W. Luk, Eds., *More FPGAs*, Oxford, England: Abingdon EE&CS Books, pp. 420-427, 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*, 1996.

[Moll95] L. Moll, J. Vuillemin, P. Boucard, "High-Energy Physics on DECPeRLe-1 Programmable Active Memory", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 47-52, 1995.

[Monaghan93] S. Monaghan, C. P. Cowen, "Reconfigurable Multi-Bit Processor for DSP Applications in Statistical Physics", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 103-110, 1993.

[Monaghan94] S. Monaghan, J. E. Istiyanto, "High-level Hardware Synthesis for Large Scale Computational Physics Targetted at FPGAs", in W. R. Moore, W. Luk, Eds., *More FPGAs*, Oxford, England: Abingdon EE&CS Books, pp. 238-248, 1994.

[Nguyen94] R. Nguyen, P. Nguyen, "FPGA Based Reconfigurable Architecture for a Compact Vision System", in R. W. Hartenstein, M. Z. Servít, Eds., *Lecture Notes in Computer Science 849 - Field-Programmable Logic: Architectures, Synthesis and Applications*, Berlin: Springer-Verlag, pp. 141-143, 1994.

[Njølstad94] T. Njølstad, J. Pihl, J. Hofstad, "ZAREPTA: A Zero Lead-Time, All Reconfigurable System for Emulation, Prototyping and Testing of ASICs", in R. W. Hartenstein, M. Z. Servít, Eds., *Lecture Notes in Computer Science 849 - Field-Programmable Logic: Architectures, Synthesis and Applications*, Berlin: Springer-Verlag, pp. 230-239, 1994.

[Oldfield95] J. V. Oldfield, R. C. Dorf, *Field Programmable Gate Arrays: Reconfigurable Logic for Rapid Prototyping and Implementation of Digital Systems*, New York: John Wiley & Sons, Inc., 1995.

[Öner95] K. Öner, L. A. Barroso, S. Iman, J. Jeong, K. Ramamurthy, M. Dubois, "The Design of RPM: An FPGA-based Multiprocessor Emulator", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 60-66, 1995.

[Pottinger95] H. Pottinger, W. Eatherton, J. Kelly, T, Schiefelbein, L. R. Mullin, R. Ziegler, "Hardware Assists for High Performance Computing Using a Mathematics of Arrays", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 39-45, 1995.

[Quénot94] G. M. Quénot, I. C. Kraljic, J. Sérot, B. Zavidovique, "A Reconfigurable Compute Engine for Real-Time Vision Automata Prototyping", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 91-100, 1994.

[Quickturn93] Quickturn Design Systems, Inc., "Picasso Graphics Emulator Adapter", 1993.

[Rachakonda95] R. V. Rachakonda, "High-Speed Region Detection and Labeling Using an FPGA-based Custom Computing Platform", in W. Moore, W. Luk, Eds., *Lecture Notes in Computer Science 975 - Field-Programmable Logic and Applications*, London: Springer, pp. 86-93, 1995.

[Raimbault93] F. Raimbault, D. Lavenier, S. Rubini, B. Pottier, "Fine Grain Parallelism on a MIMD Machine Using FPGAs", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 2-8, 1993.

[Rajamani96] S. Rajamani, P. Viswanath, "A Quantitative Analysis of Processor - Programmable Logic Interface", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1996.

[Razdan94] R. Razdan, M. D. Smith, "A High-Performance Microarchitecture with Hardware-Programmable Functional Units", *International Symposium on Microarchitecture*, pp. 172-180, 1994.

[Rose93] J. Rose, A. El Gamal, A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays", *Proceedings of the IEEE*, Vol. 81, No. 7, pp. 1013-1029, July 1993.

[Ross94] D. Ross, O. Vellacotta, M. Turner, "An FPGA-based Hardware Accelerator for Image Processing", in W. R. Moore, W. Luk, Eds., *More FPGAs*, Oxford, England: Abingdon EE&CS Books, pp. 299-306, 1994.

[Saluvere94] T. Saluvere, D. Kerek, H. Tenhunen, "Direct Sequence Spread Spectrum Digital Radio DSP Prototyping Using Xilinx FPGAs", in R. W. Hartenstein, M. Z. Servít, Eds., *Lecture Notes in Computer Science 849 - Field-Programmable Logic: Architectures, Synthesis and Applications*, Berlin: Springer-Verlag, pp. 138-140, 1994.

[Sample92] S. P. Sample, M. R. D'Amour, T. S. Payne, "Apparatus for Emulation of Electronic Hardware System", *U.S. Patent 5,109,353*, April 28, 1992.

[Schmit95] H. Schmit, D. Thomas, "Implementing Hidden Markov Modelling and Fuzzy Controllers in FPGAs", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[Schoner95] B. Schoner, C. Jones, J. Villasenor, "Issues in Wireless Video Coding using Run-time-reconfigurable FPGAs", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 85-89, 1995.

[Scott95] S. D. Scott, A. Samal, S. Seth, "HGA: A Hardware-Based Genetic Algorithm", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 53-59, 1995.

[Seitz90] C. L. Seitz, "Let's Route Packets Instead of Wires", *Advanced Research in VLSI: Proceedings of the Sixth MIT Conference*, pp. 133-138, 1990.

[Shand95] M. Shand, "Flexible Image Acquisition using Reconfigurable Hardware", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[Shaw93] P. Shaw, G. Milne, "A Highly Parallel FPGA-based Machine and its Formal Verification", in H. Grünbacher, R. W. Hartenstein, Eds., *Lecture Notes in Computer Science 705 - Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping*, Berlin: Springer-Verlag, pp. 162-173, 1993.

[Singh94] S. Singh, P. Bellec, "Virtual Hardware for Graphics Applications Using FPGAs", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 49-58, 1994.

[Suganuma92] N. Suganuma, Y. Murata, S. Nakata, S. Nagata, M. Tomita, K. Hirano, "Reconfigurable Machine and Its Application to Logic Diagnosis", *International Conference on Computer-Aided Design*, pp. 373-376, 1992.

[Tessier94] R. Tessier, J. Babb, M. Dahl, S. Hanono, A. Agarwal, "The Virtual Wires Emulation System: A Gate-Efficient ASIC Prototyping Environment", *2nd International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, 1994.

[Thomae91] D. A. Thomae, T. A. Petersen, D. E. Van den Bout, "The Anyboard Rapid Prototyping Environment", *Advanced Research in VLSI 1991: Santa Cruz*, pp. 356-370, 1991.

[Trimberger93] S. Trimberger, "A Reprogrammable Gate Array and Applications", *Proceedings of the IEEE*, Vol. 81, No. 7, pp. 1030-1041, July 1993.

[Trimberger94] S. M. Trimberger, *Field-Programmable Gate Array Technology*, Boston: Kluwer Academic Publishers, 1994.

[Van den Bout92] D. E. Van den Bout, J. N. Morris, D. Thomae, S. Labrozzi, S. Wingo, D. Hallman, "Anyboard: An FPGA-Based, Reconfigurable System", *IEEE Design & Test of Computers*, pp. 21-30, September, 1992.

[Van den Bout95] D. E. Van den Bout, "The XESS RIPP Board", in J. V. Oldfield, R. C. Dorf, *Field-Programmable Gate Arrays : Reconfigurable Logic for Rapid Prototyping and Implementation of Digital Systems*, pp. 309-312, 1995.

[Varghese93] J. Varghese, M. Butts, J. Batcheller, "An Efficient Logic Emulation System", *IEEE Transactions on VLSI Systems*, Vol. 1, No. 2, pp. 171-174, June 1993.

[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.

[vom Bögel94] G. vom Bögel, Petra Nauber, J. Winkler, "A Design Environment with Emulation of Prototypes for Hardware/Software Systems Using XILINX FPGA", in R. W. Hartenstein, M. Z. Servít, Eds., *Lecture Notes in Computer Science 849 - Field-Programmable Logic: Architectures, Synthesis and Applications*, Berlin: Springer-Verlag, pp. 315-317, 1994.

[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.

[Wakerly94] J. F. Wakerly, *Digital Design: Principles and Practices*, Englewood Cliffs, New Jersey: Prentice Hall, 1994.

[Wazlowski93] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, S. Ghosh, "PRISM-II Compiler and Architecture", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 9-16, 1993.

[Weiss94] R. Weiss, "MCM+4 FPGAs = 50,000 gates, 4048 flip-flops", *EDN*, pp. 116, April 28, 1994.

[Williams91] I. Williams, "Using FPGAs to Prototype New Computer Architectures", in W. Moore, W. Luk, Eds., *FPGAs*, Abingdon, England: Abingdon EE&CS Books, pp. 373-382, 1991.

[Wilton95] S. J. E. Wilton, J. Rose, Z. G. Vranesic, "Architecture of Centralized Field-Configurable Memory", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 97-103, 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.

[Wittig96] R. Wittig, P. Chow, "OneChip: An FPGA Processor with Reconfigurable Logic", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1996.

[Wolfe88] A. Wolfe, J. P. Shen, "Flexible Processors: A Promising Application-Specific Processor Design Approach", *21st Annual Workshop on Microprogramming and Microarchitecture*, pp. 30-39, 1988.

[Woo93] N.-S. Woo, J. Kim, "An Efficient Method of Partitioning Circuits for Multiple-FPGA Implementations", *Design Automation Conference*, pp. 202-207, 1993.

[Xilinx92] *The Programmable Gate Array Data Book*, San Jose, CA: Xilinx, Inc., 1992.

[Xilinx94] *The Programmable Logic Data Book*, San Jose, CA: Xilinx, Inc., 1994.

[Yamada94] K. Yamada, H. Nakada, A. Tsutsui, N. Ohta, "High-Speed Emulation of Communication Circuits on a Multiple-FPGA System", *2nd International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, 1994.

[Zycad94] *Paradigm RP*, Fremont, CA: Zycad Corporation, 1994.