There are parallel problems. There are obscenely parallel problems. And then there is the bioinformatics edit distance problem: given a newly discovered DNA or protein sequence of length n and a massive database of size m, find all sequences in the database that relate to the newly discovered sequence.

The accepted algorithm, Smith-Waterman, is O(nm), which, given the size of the database and genetic sequences, requires a large running time. Naturally, there is a desire to discover all the relationships between all the components of the database: O(K^2) executions of an O(nm) algorithm, on databases which today are now terabytes!

Hoang’s paper demonstrated the absolutely massive speedup potential present in FPGA acceleration for DNA sequence matching, showing that a single Splash 2 board (consisting of 17 Xilinx 4010 FPGAs) should be two orders of magnitude faster than a MasPar MP-1, a SIMD supercomputer uniquely suited to this problem. A full 16-board Splash 2 configuration would be sixteen times faster. Comparisons with a general-purpose computer are even more favorable, showing an expected four orders of magnitude improvement with a 16 board Splash 2 compared with a typical SPARC. This paper helped create a thriving industry producing custom FPGA solutions for bioinformatics, an industry that continues to demonstrate substantial benefits in compute per dollar and massive benefits in compute per Watt.

Two decades later, it is critical to reflect on why Hoang showed such amazing success. Hoang’s systolic structure turns an O(nm) time problem into an O(m) problem using O(n) parallel computational resources. This one-dimensional solution allowed his implementation to scale linearly in the number of FPGAs, while a greater interconnect requirement would have prevented the scaling to multiple chips and multiple boards. The problem also scales effectively without limit, since if one string fails to consume the available resources, there is always room to compare multiple strings simultaneously.

There are also two features that make FPGAs uniquely suited to this problem beyond the problem’s inherent parallelizability. The narrow bit-width design efficiently utilizes FPGA resources on the datapath, while the limited branching simplifies the control logic. Together this creates a very compact systolic cell, utilizing only 28 CLBs per cell. This compact nature ensures that the decedents of Hoang’s system not only remain fast, but that they are often the most efficient of any alternatives.

Nicholas Weaver

DOI: http://dx.doi.org/10.1109/FPGA.1993.279464
Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators
Jonathan Babb, Russell Tessier, and Anant Agarwal

Year of publication: 1993
Area: Architecture and Technology

In the early 1990s, one of the major uses of FPGAs was ASIC emulation. The emulator system could include dozens or hundreds of FPGAs, but each FPGA might be only 10% occupied with logic. The low utilization was a result of the limited pins available for inter-FPGA communication and by the impracticality of re-partitioning and re-routing all those FPGAs for each modification of the ASIC design as it was debugged.

The Virtual Wires work was directed at addressing the FPGA pin limitation with a simple, powerful idea: time-multiplexing the pins, serializing data onto an output pin and de-serializing it at the input. This innovation employed unused logic in the FPGA fabric to implement the serializing and de-serializing. Since logic emulation ran with clock speeds well below the FPGA’s capability and since the pin limitation left the FPGAs with excess unused logic, no delay or capacity penalty resulted from Virtual Wires. In fact, emulator systems employing Virtual Wires achieved improved speed and capacity because of their more efficient use of the FPGA logic and pins. To commercialize the idea, the authors founded Virtual Machine Works which was later acquired by IKOS. Understandably, the Virtual Wires concept was quickly adopted by major logic emulator companies.

The paper presents the Virtual Wires concept, describing not only chip-to-chip communication, but the full application in emulators in different topologies. The paper includes the circuit diagram, timing diagram and software flowchart for transforming a large netlist into a virtual wires netlist. The authors further support the idea with a formal analysis of the tradeoffs of deeper multiplexing and a derivation of an optimal partition size.

This paper demonstrated large-scale insertion of custom logic into an FPGA netlist to perform computations in soft logic that the FPGA hardware did not support. It was an “Aha!” moment for many.

The Virtual Wires concept quickly made its way into FPGA hardware. Xilinx implemented hardware assistance for serializing data onto output pins in 1995. Serializers and deserializers have since become standard components of FPGA I/O blocks, and high-speed SERDES blocks of today can trace a lineage back to Virtual Wires and this paper in 1993.

Steve Trimberger

DOI: http://dx.doi.org/10.1109/FPGA.1993.279469
The year 1995 was early in the history of reconfigurable computing research. Commercial FPGAs had existed for only a decade, and the idea of partial run-time reconfiguration was even newer. The DISC project, and this paper in particular, brought together a number of concepts that are pervasive throughout reconfigurable computing research, and did so using a fully-implemented system.

One of the key contributions of this paper is the idea of treating the two-dimensional FPGA area as a one-dimensional array of custom instructions, which the paper refers to as linear hardware space. Each custom instruction occupies the full width of the array but can occupy different heights to allow for instructions of varying complexity. Instructions also have a common interface that includes pass-thrus which connect to the previous and next instructions in the array. This communication interface greatly simplifies the problem of relocating a custom instruction to a free area, allowing other instructions to potentially remain in-place. The communication paths for an instruction are always at the same relative positions, and run-time CAD operations (and their corresponding overheads) are completely avoided. Thus, the linear hardware space reduces the overheads and simplifies the process of run-time reconfiguration of custom processor instructions.

Beyond the linear hardware model, the DISC system is notable because it takes the idea of custom processor instructions to the extreme: all instructions are custom instructions, swapped in and out of hardware by a global controller located within the processor. Even instructions that might be considered “standard”, such as add/subtract, are swapped in and out of the hardware as needed. Although this particular approach is not often taken by more modern research, it was motivated by a drive to explore the capabilities (and effects) of run-time reconfiguration.

As part of this exploration, the DISC paper presented one of the first studies of run-time reconfiguration overheads, concluding that the overhead was approximately 71%, but could be reduced to 16% with “maximized” configuration speeds. Although this demonstrates that the overhead of treating all instructions as custom reconfigurable instructions is perhaps too high, these results, coupled with the idea of the linear hardware model, provided a simple conceptual model and a powerful motivating force for future reconfigurable computing research.

Katherine Morrow

DOI: http://dx.doi.org/10.1109/FPGA.1995.477415
In the context of an Automatic Target Recognition application this paper explores computing structures that uniquely exploit FPGA characteristics under the assumption that the FPGA can be rapidly reconfigured. In doing so, this paper was highly influential in raising awareness to the potential of runtime reconfiguration, as evidenced by the citations it has received in the ensuing years.

To address the application, the work described in the paper develops an application-specific CAD system to generate a sequence of optimized circuits that can be sequentially overlaid on the available FPGA resource via rapid reconfiguration. By swapping configuration overlays, the authors overcome the classic performance cliff often encountered in application mapping to FPGA where, beyond a certain size, a circuit no longer fits in the available device and a larger device must be used or the circuit must be radically rearchitected. They also introduced an application-specific common subexpression mapping optimization (see figure) that increased the number of patterns they could pack into one configuration by an order of magnitude.

FPGAs in 1996 were much smaller than they are today and some of the detailed mapping issues discussed in the paper may seem quaint from a modern perspective, but in truth, the fact that implementation details like automatically mapping shift registers to on device memory resources are now handled automatically by FPGA vendor synthesis systems is in part due to pioneering work such as is described in this paper. Today the scale has changed, for instance multipliers are now primitive elements, but at a system level the issues of application mapping and scalability, particularly in real-time image and video processing, are not so different and the contributions of the paper remain relevant.

This paper showed the potential of rapid reconfiguration, but, in the scope of the work it describes, it anticipated the difficulty of bringing such solutions to market. FPGA vendors have been slow to react, but increasingly we are seeing them offering devices and the necessary CAD support that allow applications such as this paper envisioned to enter the mainstream.

Mark Shand

DOI: http://dx.doi.org/10.1109/FPGA.1996.564749
MATRX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources
Ethan Mirsky, André DeHon

**Year of publication:** 1996

**Area:** Architecture and Technology

This paper describes a coarse-grain, reconfigurable computing array that has been highly cited by subsequent work using the coarse-grain approach.

The novelty of this work is that the fundamental array element can be configured as control logic, memory, or datapath logic, providing the flexibility to implement the functionalities where they are needed. To illustrate the flexibility of the array, it is shown how a simple convolution can be implemented using several architectures: systolic, microcoded, VLIW and a hybrid multi-SIMD/VLIW.

More significantly, the work is representative of an era when there was significant activity devoted to proposing new architectures for reconfigurable computing. VLSI technology was still simple enough that universities could contemplate and prototype significant designs with graduate student labor within the timeframe of a graduate degree. In this paper, the layout of one of the array elements is done using a 0.5µm technology, which then provides reliable information about the size of the block and a reasonable estimate of performance. This is important because it is information that can be presented to venture capitalists as validation that it is worthy to fund a new company. Many startups followed this model. Few survived. The work in this paper led to the founding of Silicon Spice, Inc., which may be the most successful reconfigurable computing company if measured by its peak value. Silicon Spice was acquired by Broadcom Corp. in August 2000 for $1.2 billion during the height of the Dot Com bubble at the turn of the century.

As an architecture paper, this paper lacks any quantitative comparisons to alternative architectures that would validate the benefits of this architecture. It only provides a qualitative evaluation using a simple application. In today’s world, this paper would also need some discussion of the software needed to map to this architecture.

Paul Chow

DOI: [http://dx.doi.org/10.1109/FPGA.1996.564808](http://dx.doi.org/10.1109/FPGA.1996.564808)
OneChip: An FPGA Processor With Reconfigurable Logic
Ralph D. Wittig and Paul Chow

Year of publication: 1996
Area: Architecture and Technology

As FPGAs rapidly scaled beyond their initial role of replacing digital glue logic, the idea of using a single chip in a system for everything was very compelling. However, researchers in the reconfigurable computing community continued to find that for some applications a processor was hard to beat. So why not join them?

Ralph Wittig and Paul Chow proposed exactly that in their 1996 OneChip paper. Some prior work in the field (e.g. PAM) had loosely coupled FPGAs and host processors, often limiting performance to achievable communication speed. Alternatively, it had been shown that a small amount of reconfigurable logic could be integrated into a CPU (e.g. PRISC), although the benefits of this customization were small and constrained to mostly simple combinational functions. In contrast, OneChip proposed a tight coupling of a large amount of reconfigurable logic with a microprocessor fabric. In fact, their design showed so much reconfigurable logic in comparison to the CPU core (see figure above) it more accurately could be described as a processor embedded within reconfigurable logic, rather than the inverse.

OneChip not only contained a CPU core within reconfigurable logic, but the reconfigurable logic also appears as a function unit, called a PFU, directly in line with the CPU execute pipeline. Programmable logic placed between the CPU and the chip I/Os enabled a broad range of embedded applications, while tight coupling enabled an order of magnitude speedup or more to be maintained with even a small computation grain size.

This paper not only describes an innovative architecture, but also an innovative prototyping environment that consisted of a multi-FPGA system that leverages both a programmable interconnect chip and pin multiplexing. The architecture also contains an optimized memory interface and register file size to reduce the cost of spilling, showing a speedup of nearly 50 for some cases versus an optimized design. From their architectural design insights, the authors correctly predicted that run-time operating systems that handle context switching would become a major research challenge for the field of reconfigurable computing.

Today, integrated processor cores come standard in available FPGAs and have been one of the major contributing factors in expanding the application space of reconfigurable devices to a large and diverse market space.

Jonathan Babb

DOI: http://dx.doi.org/10.1109/FPGA.1996.564773
A Time-Multiplexed FPGA
Steve Trimberger, Dean Carberry, Anders Johnson, Jennifer Wong

Year of publication: 1997
Area: Architecture and Technology

In the early 1990’s, it was challenging to fit designs onto the small FPGAs of the day, and several research groups [(Bhat 1993), (DeHon 1995), (Tau 1995)] were exploring rapid reconfiguration capability. The reconfigurable mesh model, which was proposed in the 1980’s offered a theoretical view, but no architectural mechanisms to support it. While the community was debating dynamic reconfiguration and the potential value of such a mechanism for application developers, this paper offered a feasible approach to realize such a model in hardware, illustrating potential benefits and some applications of such a feature. The design offered a small number of configurations to be stored on the device and provided architectural support for the device to be rapidly reconfigured by switching between these configurations. The configuration memory itself could be used as on chip storage, predating the current Block RAM (BRAM) idea. This paper pushed the boundary (at that time) by showing programmable devices that could do more than one task rapidly.

The key concept is to enable dynamic re-use of the hardware. The paper proposed to store eight configurations in distributed SRAM on the device. These configurations can be used to configure the logic as well as the interconnects. The authors proposed three modes of operation. In logic engine mode, a single large design can be emulated using multiple smaller configurations. In the time-share mode, designs with multiple communicating FPGAs can be emulated. Finally, they provide a static mode in which the logic is resident on the device all the time and is not reconfigured. The architecture also provided the capability to mix these modes to improve application performance. The configuration memory could also be written from on-chip logic, a feature that later showed up in Xilinx Virtex FPGAs with the ICAP interface. The paper provided detailed designs to enable these capabilities.

The paper stimulated thinking both from the architecture community to realize dynamic reconfiguration in hardware and from the applications community to contemplate novel uses of FPGAs if rapid reconfiguration was feasible. The exposition is crisp, and it makes a good reading in FPGA architectures as well as application acceleration courses. However, the design did not lead to time-multiplexed FPGAs from Xilinx---Moore’s Law scaling remained their preferred path to increasing device capacities.

Viktor K Prasanna

DOI: http://dx.doi.org/10.1109/FPGA.1997.624601
Defect Tolerance on the Teramac Custom Computer
Bruce Culbertson, Rick Amerson, Richard J. Carter, Philip Kuekes, Greg Snider

Year of publication: 1997
Area: Architecture and Technology

This paper was groundbreaking in its use of component-specific mapping for defect tolerance. They built a full system with individual resources (logic elements, MCM traces, and board-to-board cable connections) that failed in the few percent range (see table) and developed the tools to make it all work!

Defect tolerance in regular structures such as RAM arrays was well established in the 1980s. Altera PLDs and FPGAs extended RAM-like techniques for tolerating defects. However, they both used an approach that tried to repair the IC behind the scenes to make it look like a perfect component to the user and tools. These techniques work reasonably well when the defect rate is low, but have prohibitive costs at the level of defects suggested above.

In contrast, the Teramac design embraces the reconfigurability of the base FPGA-like substrate and exposes the defects in each particular chip, board, and connector to the placement and routing tools. With this knowledge, place-and-route can avoid the defective components during mapping, reasonably tolerating defects that are orders of magnitude higher than typical for RAMs. The resulting strategy is closer to how operating systems help tolerate defective sectors on hard drives, by mapping logical blocks to working physical sectors and avoiding the bad sectors. In addition to an over-provisioned interconnect and defect-aware-place-and-route, the paper describes novel group testing techniques to quickly identify the defects in a particular component or system and in-system delay testing to determine the speed of components.

At the time, these techniques allowed HP to aggressively use technologies that would not otherwise have been possible (large IC dies, large and complex MCMs, myriads of inexpensive ribbon cables) and build a system of a scale that was not otherwise feasible (440K 6-LUTs spread over 864 FPGAs on 8 boards). However, the real value in this work is that it provided a roadmap and demonstration that this size of system and level of defects could be conquered. The ideas developed here have been enabling for bottom up molecular logic designs, both from HP and from researchers around the world, and may be essential for all near-atomic-scale technologies including highly-scaled lithography.

André DeHon

DOI: http://dx.doi.org/10.1109/FPGA.1997.624611
**Garp: A MIPS Processor with a Reconfigurable Coprocessor**

John R. Hauser, John Wawrzynek

**Year of publication:** 1997  
**Area:** Architecture and Technology

This paper introduced a novel hybrid architecture that tightly coupled a MIPS processor with a customized reconfigurable array. A programming environment was described that allows a programmer to jointly develop the reconfigurable and sequential portions of an application.

Although there were several good architectural examples of coupled processor/reconfigurable arrays developed prior to this work, Garp provided convincing, quantitative evidence that such an architecture is both technically feasible and offers performance advantages. This paper provides one of the best examples of an FCCM paper that introduces a novel architecture, provides a programming environment, estimates area and delay, and demonstrates the benefits of the architecture and programming environment with speed-up on application examples. Few FCCM papers provide such a convincing demonstration on such a broad set of work.

For good reason, this paper is among the most widely cited papers from FCCM. The concepts introduced in this paper in 1997 address the many of the same challenges facing reconfigurable computing systems today. Some of these concepts are still of interest to modern research in reconfigurable systems. Custom processor instructions were introduced to manage the configuration of the reconfigurable array and data transfer between the processor and the array. A shared memory architecture allowed for high-bandwidth, coherent communication between memory and the processor/array pair. Caching of configuration data allowed for high-speed configuration context switching. A customized reconfigurable fabric facilitated efficient processing of 32-bit data.

This paper inspired many follow-on efforts in both reconfigurable architecture and compilation tools. A fascinating effort was pursued to automatically generate reconfigurable hardware accelerators from traditional software. The tight integration of processor and reconfigurable array within the Garp architecture motivated the need for a unified development environment where all functionality is defined in traditional sequential code. This paper will continue to impact the field of reconfigurable computing for the coming decade.

Mike Wirthlin

DOI: [http://dx.doi.org/10.1109/FPGA.1997.624600](http://dx.doi.org/10.1109/FPGA.1997.624600)
Incremental Reconfiguration for Pipelined Applications
Herman Schmit

Year of publication: 1997
Area: Run-time Systems and Run-Time Reconfiguration

By the mid-1990s, FPGAs were being used for signal processing and computing. However, using FPGAs for computational tasks was hard. The FPGAs were small. An application developer had to be acutely aware of the FPGA capacity and massage the design to fit. The chip capacity created a performance cliff for designs. Furthermore, when a new, larger FPGA came along, it was necessary to redesign the application to exploit the new logic capacity. This was particularly unattractive to developers long accustomed to microprocessors, where you did not have to be aware of the size of your computation in order to get it working. Furthermore, once you had a design working, you could reasonably expect newer microprocessors to run the design faster without further development.

At the same time, FPGA users, vendors, and researchers were experimenting with runtime reconfiguration to create the illusion of additional logic capacity. While early runtime reconfiguration applications looked promising, they demanded more design effort and did not address the issue of scaling.

The signal processing and cryptography kernels that were showing good performance on FPGAs often obtained their performance benefits by exploiting pipeline parallelism---building a deep spatial pipeline for the computation. Schmit observed that pipelined computation could be used as an abstract model for these applications, and this model could be supported with a novel reconfiguration architecture to address the problem of design fit and scalability.

In particular, the pipeline provided a basis for loading only a small fraction of the configuration per cycle---the configuration for a single stage of the pipeline. It also served as a key unit of temporal locality---the same configuration could be reused on the next cycle to compute the next set of data flowing through the pipeline. The configuration could, itself, be pipelined through the computational fabric to spatially adjacent pipeline stages. This allowed (1) the reconfigurable array to be compact, holding a single configuration, (2) the configurations to live in large, dense memories outside of the array, and (3) the array to productively use limited bandwidth to the external configuration memory. The architecture could scale by adding physical pipeline stages.

This paper was the first of a series of papers about the architecture that would eventually be known as PipeRench. It identified the challenge and the basic solution, used simple analysis to show the potential benefits of the scheme, provided preliminary VLSI implementation characteristics, and illustrated support for a couple of applications. The PipeRench design later became a key part of the CMU Q-Machine and was briefly commercialized by Rapport, Inc.

André DeHon

DOI: http://dx.doi.org/10.1109/FPGA.1997.624604
The Chimaera Reconfigurable Functional Unit
Scott Hauck, Thomas Fry, Matthew Hosler, Jeffrey Kao

Year of publication: 1997
Area: Architecture and Technology

This paper introduced a reconfigurable functional unit that could accelerate critical portions of applications without negatively impacting RISC processor core clock speed or memory bandwidth.

Almost from the introduction of commercial field-programmable gate arrays, it was recognized that reconfigurable logic could be used in conjunction with a standard RISC microprocessor to accelerate certain types of computation. Early FPGA-based coprocessors were generally located on the processor bus, some distance from the processor. These units were often configured for coarse-grained computation due to memory bandwidth constraints between the RISC microprocessor and the coprocessor. The integration of reconfigurable logic as a functional unit inside the RISC processor core met with limited success due to the delay difference associated with reconfigurable logic versus fixed logic. The inclusion of FPGA logic in the processor core limited the clock frequency of the RISC processor, negatively impacting all instructions.

This paper took a fresh approach to addressing both the memory latency and processor performance issues. The Chimaera reconfigurable functional unit (RFU) is located close to the RISC microprocessor, but outside of the primary RISC execution path. The RFU has access to the same registers as the processor via a shadow register file that includes a collection of dedicated, multi-operand ports. This supplemental register file provides fast access to the same data that is available to the processor without restricting the RISC processor operating frequency. The paper describes the possibility of speculatively executing certain reconfigurable operations while the remainder of the RISC code proceeds in a standard fashion. Additionally, unlike earlier FPGA coprocessor offerings, the paper provides a customized LUT-based datapath configuration for the RFU which is optimized for word-based operation. Similar configurations are later seen in other reconfigurable architectures, such as PipeRench.

This paper provided a significant step forward in understanding how to integrate RISC microprocessors and reconfigurable compute units together. Although contemporary commercial efforts in this space (e.g. Stretch) met with limited success, some of the elements of this design style are becoming more popular as compiler technology matures. The ability of a compiler to identify functions that can take advantage of the reconfigurable resource and configure it appropriately is still an active area of research in reconfigurable computing.

Russell Tessier

DOI: http://dx.doi.org/10.1109/FPGA.1997.624608
Configuration Compression for the Xilinx XC6200 FPGA
Scott Hauck, Zhiyuan Li, Eric Schwabe

Year of publication: 1998
Area: Tools

This paper shows how to automate bitstream compression by exploiting the wildcard addressing scheme in the XC6200. It is an excellent example of identifying a new optimization problem that arises with novel uses of FPGAs, formulating the problem carefully, and providing a sophisticated solution based on a non-trivial reduction of the problem to powerful, known optimization techniques.

Bitstream load time was a small concern for load-once applications. However, as engineers started to consider run-time reconfiguration applications that exploited the fact that FPGA configurations could be changed dynamically during operation, it became a bigger concern—one that could severely limit the cases where run-time reconfiguration was a benefit. Since bitstream load is typically I/O bandwidth limited, one way to reduce bitstream load time is to compress the bitstream; this can be particularly effective for regular applications where many resources are identically configurable. The XC6200 was a short-lived FPGA family aimed at regular computing applications that had an open bitstream format that supported random access into the bitstream. In addition to random access, it allowed wildcard row and column addressing so that the same configuration could be simultaneously written into a set of configurable cells. While it was somewhat clear how to use this for carefully hand-designed logic and layout, it came with no CAD tools to automate the use of the wildcard addressing or automatically identify how to use the wildcards for arbitrary designs.

This paper showed that the problem of determining the minimum set of wildcard writes to configure the FPGA could be reduced to the same set-covering problem that is at the core of two-level logic minimization. Furthermore, it shows how to formulate the ability of the FPGA to overwrite cells as don’t-cares in the two-level optimization problem and how to leverage the espresso two-level optimization tool to perform the core covering operation. The result is an elegant optimization formulation and solution for this new challenge that provides a practical tool that can compress regular, hand-crafted bitstreams to about one-fifth their size and CAD mapped designs to about half their size.

The formulation and development of this optimization technique also tells us about the value of the wildcard addressing architecture. Without the algorithm, it was not possible to quantify the actual savings this architectural feature offered. The paper quantifies the benefit on a set of applications as well as introducing the tool needed to perform further evaluations.

André DeHon

DOI: http://dx.doi.org/10.1109/FPGA.1998.707891
FCCM20 Endorsement

**Accelerating Boolean Satisfiability with Configurable Hardware**
Peixin Zhong, Margaret Martonosi, Pranav Ashar, Sharad Malik

**Year of publication:** 1998  
**Area:** Applications

This paper addresses accelerating Boolean satisfiability, a problem of great interest in a number of computer-aided design (CAD) applications and beyond. Satisfiability is the problem of assigning the variables of a Boolean expression in such a way that the expression evaluates to true. If there is no such assignment, a counter example is produced. Satisfiability is used in many automated proof tools to prove the validity of a statement. It is also used for automatic test pattern generation and routing, among other applications.

Boolean satisfiability was the first problem to be shown to be NP-complete. Many optimization problems are cast into Boolean satisfiability, allowing them all to leverage advances in satisfiability solvers. Using parallelism, such as the parallelism available in FPGA fabrics, helps to accelerate specific instances of the Boolean satisfiability problem.

This paper represents one of the early, successful approaches to implementing a SAT solver on FPGAs and is an early example of the advantages of specializing a circuit for a particular problem instance. It has several excellent features, including software that translates the Boolean satisfiability problem to FPGAs. An interesting discussion of compile times versus run times in the paper is still relevant to current problems in heterogeneous acceleration. In addition, the FPGA implementation ran on state-of-the-art hardware for the time, a DEC Pamette board and an IKOS logic emulator designed for emulating processors. SAT solving speedups of several hundred times were measured for problems of up to four hundred variables and one thousand clauses.

This paper represents the basis of much of the research that followed in this hugely important application area. SAT solving continues to be an important application today, and FPGA implementations of SAT continue to be researched and published. Researchers are interested in increased speed, but also in the ability to solve much larger problems with orders of magnitude more variables and clauses.

This work also had enormous impact on pure software satisfiability solvers. The huge speedups compared to software motivated the developers to understand where time was going in their software solvers. The resulting understanding, initiated a series of optimizations to the software solvers, starting with Chaff, that provided orders of magnitude speedup without FPGA hardware.

Miriam Leeser

**DOI:** [http://dx.doi.org/10.1109/FPGA.1998.707896](http://dx.doi.org/10.1109/FPGA.1998.707896)
FCCM20 Endorsement

A CAD Suite for High-Performance FPGA Design
Brad Hutchings, Peter Bellows, Joseph Hawkins, Scott Hemmert, Brent Nelson, Mike Rytting

Year of publication: 1999
Area: Languages and Compute Models

It's funny how languages and tools get established, spread and become standard. For decades now, digital hardware in ASICs and FPGAs has been designed in languages originally defined for documentation (VHDL) and simulation (Verilog). ASIC engineers put up with these languages and their synthesis foibles despite the fact neither was conceived or defined to be a design language. We still use the ASIC model and languages for FPGAs in FCCMs, even though FPGAs don't even have logic gates, and a "silicon turn" takes minutes. Out of a number of efforts to escape from this and design FPGAs in FPGA terms, arguably the most successful was JHDL, a landmark in FPGA design.

JHDL started as a design tool for reconfigurable systems that change over time, supporting runtime and partial reconfiguration, as originally reported in an excellent FCCM 1998 paper (with "JHDL" in the title, that paper is often cited for this paper's work). JHDL's major impact came with further development into the unified, graphical FPGA tool suite reported in this paper.

You design in JHDL by writing a program in a well-known language, Java, using library objects for hardware components. JHDL is constructive and parametric. Higher-level design and instance specialization comes naturally from object-oriented programming. You construct your design as a hierarchy of objects instantiated with constructor parameters. What you write is what you get, avoiding pitfalls and uncertainty of synthesis, and providing fine control over all levels of the FPGA implementation, including placement. This is particularly important for datapaths and computing applications where automated tools often fail to find the mappings the designer knows the FPGA can realize. Executing your Java program generates its netlist or runs its simulation. Since JHDL design and simulation are unified in a simple graphical toolset, shifting between editing and testing is instantaneous. It happens so fast it's addictive and fun. JHDL also makes it easy to switch between software simulation and hardware execution in the same tool.

BYU released JHDL in open-source form, so it came into wide use by FCCM researchers and students. BYU faculty and students provided strong support through several FPGA generations. A substantial number of reconfigurable computing projects were developed in JHDL, and it had a strong influence on high-level FPGA and FCCM design tools to follow. This put it in the top tier of FCCM citations. JHDL is one of the most important and effective design methodologies to come out of the first twenty years of FCCM.

Mike Butts

DOI: http://dx.doi.org/10.1109/FPGA.1999.803663
Parallelizing Applications into Silicon
Jonathan Babb, Martin Rinard, Csaba Andras Moritz, Walter Lee, Matthew Frank, Rajeev Barua, Saman Amarasinghe

Year of publication: 1999
Area: Tools

In 1999 FPGA architecture was at the inflection point where the performance of the core computing logic in FPGAs had improved to the point where the communication was fast becoming the blocking factor for performance improvement. A high-level synthesis approach that assumed that wires were for free was bound to fail, and the insights in this paper represent a quick response to the changes in the economics of the underlying architecture. Various prior efforts had proposed compilation from sequential programming languages, but none had really tackled issues of large-scale distributed memories and control.

This paper shows how sequential programs written in C and FORTRAN can be mapped to digital logic while taking care to model the memory organization of an application to try and improve the degree of parallelism that can be automatically extracted. Specifically, the authors attempt to transform one large slow memory into many smaller fast memories by analyzing the memory access patterns in each application. This transformation process enables fine-grained, highly parallel computation with short communication paths and enables a higher aggregate memory bandwidth than is possible with a monolithic memory model. The authors highlight an important realization about the economics of high performance computing systems: the cost of communication is the bottleneck rather than the cost of computation.

The compilation process exploits basic pointer analysis methods to help decompose the heap into regions that can be accessed independently, yielding impressive performance. Experiments in the paper reported a 6x speedup of an MPEG kernel compared to the same algorithm executed on a MIPS R2000 processor. The authors also reported comparisons with the Raw machine using 16 processors which produced an 8x speedup (a hand wired version ran at 32x).

Today engineers expect modern synthesis systems convert C or behavioral HDL code into efficient circuits with judicious use of memory blocks and the ability to process nested loop computations over arrays into parallel fine-grained computations. However, this 1999 paper was well ahead of its time, identifying and solving several issues that lead to a promising synthesis flow.

Satnam Singh

DOI: http://dx.doi.org/10.1109/FPGA.1999.803669
Stream-Oriented FPGA Computing in the Streams-C High Level Language
Maya Gokhale, Jan Stone, Jeff Arnold, Mirek Kalinowski

**Year of publication:** 2000  
**Area:** Languages and Compute Models

An important step in popularizing the field-programmable custom computing machine approach is to provide design tools that enhance designer productivity by enabling application development using high-level programming languages, rather than hardware description languages such as VHDL or Verilog. However, such productivity enhancement must not sacrifice the performance benefit of the FCCM implementation. This issue was particularly acute in the 1990’s when field-programmable gate arrays had a limited amount of resources. If application implementations did not meet timing requirements or did not fit within the target devices, the high-level programming language solution would not be usable.

This paper demonstrates a pioneering effort to meet the challenge of providing design tools for field-programmable custom computing machines that support both designer productivity and design efficiency. There are three reasons for its significance.

First, the proposed approach focuses on stream-oriented computing which has an abundance of data parallelism that can be effectively mapped onto resources of field-programmable gate arrays. This focus allows the tools to cover applications with inherent concurrency without limiting them to a single narrow application domain. Second, the proposed tools support a subset of the C language with annotations to direct a compiler to generate both hardware circuits targeting one or more field-programmable devices, and a multi-threaded control program for a host processor that communicates with the field-programmable devices. The use of annotations has been adopted by various hardware compilers (e.g., the AutoPilot system from AutoESL). Third, the paper addresses the difficulty issue of productivity evaluation. For an image processing application involving contrast enhancement, it reports that designer productivity was improved by 10 times, at the expense of a speed reduction by half and a three times increase in area, when compared with a hand-coded version in VHDL. Just like advances in software compilers, the efficiency gap between implementations from hardware compilers and from hand-coded designs would increasingly narrow.

It is a tribute to this paper that many of its ideas continue to live on in the commercial Impulse-C compiler, which has been advertised for a range of applications from scientific computing to secure communications.

Wayne Luk

**DOI:** http://dx.doi.org/10.1109/FPGA.2000.903392
FCCM20 Endorsement

**Configuration Caching Management Techniques for Reconfigurable Computing**

Zhiyuan Li, Katherine Compton, Scott Hauck

**Year of publication:** 2000

**Area:** Run-Time Systems and Run-Time Configuration

Anyone who has ever been on a program committee knows that there is one debate that almost always occurs, and that debate concerns the relative value of novelty and thoroughness. Most clever innovative papers are experimentally weak. The papers that have flawless experimental sections are usually incremental. I was not at the FCCM meeting in 2000, but this paper would not have created that discussion. It is both novel and thorough.

The idea is simple. Configuration time is so large that run time reconfiguration might not be able to effectively improve the performance of reconfigurable computing systems. In order to minimize the impact of configuration, this paper proposes and examines the possibility of configuration caching. To evaluate this, the authors have to construct a model for three different kinds of FPGA configuration structures: single context FPGAs, run-time reconfigurable FPGAs, and multi-context FPGAs. This was non-trivial because these models did not all exist commercially, and those that did exist were not comparable to each other. In addition, the paper evaluates both off-line and on-line optimizations of the configuration placement. Finally, the authors recognized the issue of configuration fragmentation, as illustrated above, and its impact on the ability to reconfigure. The paper is an exemplar of thorough evaluation.

The influence of this paper is an interesting subject. Twelve years after publication, there are no commercial systems that utilize cached configurations, at least not in the manner envisioned in this paper. However, neither are there commonly accepted run-time reconfiguration execution models that could benefit from these solutions. The multi-context FPGA that does exist is based on a different computational model. This paper is highly referenced by others who have attempted variations of the techniques presented here with different applications and algorithms. The less acknowledged value of research is to explore ideas that do not seem to pay off. Then again, perhaps this paper is still just ahead of its time. If we do eventually define suitable, standard execution models, this work provides a good first set of hints about architectures and algorithms to support them.

Herman Schmit

**DOI:** [http://dx.doi.org/10.1109/FPGA.2000.903390](http://dx.doi.org/10.1109/FPGA.2000.903390)
A MATLAB Compiler For Distributed, Heterogeneous, Reconfigurable, Computing Systems
Prithviraj Banerjee, U. Nagaraj Shenoy, Alok Choudhary, Scott Hauck, Christopher Bachmann, Malay Haldar, Pramod Joisha, Alex Jones, Abhay Kanhare, Anshuman Nayak, Suresh Periyacheri, Michael Walkden, David Zaretsky

Year of publication: 2000
Area: Languages and Compute Models

As reconfigurable computing headed into the 21st century, it was widely recognized that compilation technology would be a limiting issue in the advancement of the field. For the full power of reconfigurable computing to be utilized, compiler technologies that support computer languages used by large numbers of programmers and scientists were needed. Although the earlier development of C compilers for field-programmable systems had somewhat addressed this concern, a gap still existed in reconfigurable computing support for applications expressed in MATLAB, a widely popular language used for signal processing and scientific computing.

This paper provides a high-level view of a MATLAB compiler targeted at systems that include a diverse set of computing resources. The compiler first converted MATLAB code into an abstract syntax graph consisting of a collection of smaller subgraphs. Each subgraph can then be targeted to multiple FPGAs, a digital signal processor (DSP), or a Unix-based processor system. The FPGA implementations included a simple message passing protocol to simplify data exchange across devices. For complicated applications, the compilation environment allowed for the use of program directives to guide the compiler towards better mapping decisions. The authors demonstrated the effectiveness of their compiler system by using it to compile several applications to all three computing substrates. For all the included benchmarks, the FPGA implementation provided the most parallelism and the largest speedup.

The work described in this paper provided several steps forward in compilation technology for reconfigurable systems. First, it drew attention to the need for reconfigurable system compilers to not only target FPGAs, but also a heterogeneous system of compute components. Second, and more importantly, it provided a first step in the development of the MATLAB compilation tools in Xilinx System Generator that are used by thousands of engineers annually.

Russell Tessier

DOI: [http://dx.doi.org/10.1109/FPGA.2000.903391](http://dx.doi.org/10.1109/FPGA.2000.903391)
Fast Regular Expression Matching Using FPGAs
Reetinder Sidhu, Viktor K. Prasanna

Year of publication: 2001
Area: Applications

Creating a matcher for a specific regular expression is typically a two-step process. First an NFA is derived from the grammar representing the regular expression. Second, the NFA is converted into a corresponding DFA. The DFA is then typically implemented in software and used to identify strings in an I/O stream.

In this paper, the goal of the authors was to accelerate regular-expression matching by implementing regular-expression matchers in reconfigurable hardware. Their approach was quite novel; they completely skipped the DFA conversion step and directly implemented the NFA in hardware. This reduced the amount of time it took to implement the matcher in hardware. It also avoided a step that, in the worst case, may require exponential time to complete. Their NFA-based matchers require $O(n)$ time and $O(n^2)$ space where $n$ is the length of the regular expression. The NFA-based hardware matcher processes one character per clock cycle.

This is an excellent example of an application paper that demonstrates how reconfiguration not only enables an application, but also provides a new way to look at a problem. Because they are typically developed in the context of general-purpose processors, NFAs are usually just a step on the path toward implementing a DFA-based matcher. However, as demonstrated by this paper and others that followed it, NFAs can be an effective way to implement regular-expression matchers on a reconfigurable fabric. The authors showed that NFA netlists can be directly implemented on FPGAs as one-hot state machines, allowing the state machine to directly represent the set of states in which the NFA might currently be, and they provided a simple algorithm that converts an NFA graph into a netlist that can be placed and routed by vendor place and route tools.

Though the NFA-based compilation process has fewer steps than a DFA-based process, it is still hampered by the need to place and route the NFA-based netlists, a process that can take minutes or longer. As such, place and route time will typically represent the lion’s share of total processing time required to convert a regular expression to a hardware matcher. The author’s solution to this problem was to introduce a theoretical architecture (SRGA) that made constructive placement simple, but there remains no practical solution to this problem. Still, the results from this paper had immediate practical implications for applications that can cope with place and route overhead and helped to enable the work reported in another paper included in this volume: “Assisting Network Intrusion Detection with Reconfigurable Hardware.”

Brad Hutchings

DOI: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1420919
Pilchard – A Reconfigurable Computing Platform with Memory Slot Interface
Philip H. W. Leong, Monk-Ping Leong, Ocean Y. H. Cheung, Tung Tung, Chung-Man Kwok, Ming-Yee Wong, Kin-Hong Lee

Year of publication: 2001
Area: Architecture and Technology

This is one of the earliest papers to report a low cost, high performance reconfigurable computer board that was made available to the field-programmable custom computing community. As the name ‘Pilchard’ suggests, this board provides a small, cheap and easily accessible platform to feed the custom computing community.

The uniqueness of this platform is its use of the 133 MHz memory bus interface instead of the then much slower PCI interface found in other platforms. The authors did not only design the hardware platform, but also provided a development environment based on the Linux operating system with easy-to-use interfaces between the host computer and the Pilchard board. Through the efforts of this team, research groups around the world working on reconfigurable computing were able to explore this technology without having to design their own platform. The paper also demonstrated the usefulness and advantages in using memory interface with an implementation a fully parallel, pipelined data encryption standard (DES) core.

There have been many papers published about reconfigurable computing platform. This paper stands out as one that is both technically excellent and promotes the ideal of sharing within the community.

Peter Cheung

DOI: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1420913
FCCM20 Endorsement

**Assisting Network Intrusion Detection with Reconfigurable Hardware**

Brad L. Hutchings, Rob Franklin, Daniel Carver

**Year of publication: 2002**

**Area: Applications**

This paper was pioneering in that it introduced an integrated domain-specific tool flow for generating efficient FPGA circuitry for carrying out the regular expression matching required in practical network intrusion detection (NID).

The work built upon the prior work of Sidhu and Prasanna (FCCM 2001), who showed how regular expressions could be mapped to a modular FPGA implementation of Nondeterministic Finite Automata (NFA) without intermediate Deterministic Finite Automata representations.

The central contributions of the paper were threefold. First, the NID module generator was driven by pattern matching rules used by a standard software system, specifically the open-source SNORT system. This showed how an FPGA-neutral, domain-specific language could automatically be mapped to an efficient FPGA implementation and set a trend for future researchers, who adopted the SNORT rule database as a standard benchmark. The use of SNORT rules meant adding support for extended regular expression features beyond the basics of concatenation, choice, and Kleene star.

Second, the paper made good practical use of the JHDL hardware design tool kit (FCCM 1998) to underpin the NID module generator, by providing a convenient means to build modular implementations of the NFA required to perform regular expression matching.

Third, the paper showed how the module generator provided a complete flow from the domain-specific NID rules through to optimized FPGA implementations, and illustrated this with performance results that showed the scalable benefits of the FPGA-based approach, compared with the limitations of a standard software-based approach.

Through these contributions, this paper exemplifies what is desirable in an FCCM applications paper. It was not just a direct implementation of some application on an FPGA. Rather, it showed a reusable tool and methodology in a specific domain. Although NID was the motivating application, the work is more generally applicable to situations where regular expression matching is required. While the actual FPGA resource counts and performance data are now somewhat dated, and other researchers progressively improved upon the results, the approach used in this work has stood the test of time.

Gordon Brebner

**DOI:** [http://dx.doi.org/10.1109/FPGA.2002.1106666](http://dx.doi.org/10.1109/FPGA.2002.1106666)
This paper presents a performance study of double precision, floating-point, dense matrix operations implemented on several conventional microprocessor and FPGA platforms, predicting that FPGA peak performance on these operations will outstrip commodity CPU’s and offer the promise of significantly higher future performance. To support these predictions, issues of memory bandwidth and on-chip/off-chip memory size are examined in detail.

The dense matrix operations implemented in this paper are members of the Basic Linear Algebra Subroutines (BLAS) library, an industry standard library defined by Jack Dongarra et al. in the late 1970’s. BLAS, in turn, is closely tied to the LINPACK benchmark that has become the standard to gauge the performance of high performance scientific computers. Many microprocessor companies, such as Intel and AMD, have spent a great deal of time and money optimizing their versions of these libraries to give the highest performance on their products.

The contribution of this paper is the recognition of the emergence of high-performance, double precision, floating-point FPGA arithmetic that challenged the long held position of the commodity microprocessor in scientific computing. This was particularly significant since floating-point computations were not generally considered viable on FPGAs just a few years prior to this paper. This seminal paper also sparked a flurry of research in FPGA applications that require double precision floating-point arithmetic.

Since the publication of this paper, the rapid rise of scientific computing on Graphics Processor Units (GPU’s) has currently eclipsed both conventional microprocessors and FPGAs alike in many, if not most, floating-point applications.
FCCM20 Endorsement

Reconfigurable Molecular Dynamics Simulator
Navid Azizi, Ian Kuon, Aaron Egier, Ahmad Darabiha, Paul Chow

Year of publication: 2004
Area: Applications

Molecular dynamics (MD) involves the computer simulation of Newtonian mechanics as applied to a system of atoms or molecules. Since its discovery in 1957, it has become one of the most widely used tools for studying molecular behaviour and, due to its large computational requirements, one of the most demanding problems for supercomputers. MD simulations have practical applications in materials design, thermodynamics, protein folding/unfolding and drug discovery. This paper was the first of many using MD to demonstrate the feasibility of applying FPGA-based field-programmable custom computing machines to large-scale problems.

Several techniques were employed to maximise performance: spatial parallelism, pipelining, input/output optimisation, and precision optimisation. Spatial parallelism was applied so that operations without dependencies are processed in parallel computational units. Pipelining allowed the system to execute stages in parallel, resulting in higher clock rates. Memory accesses were carefully optimised to ensure that the maximum available bandwidth was utilised, and the data were kept on the acceleration hardware to minimise host to accelerator communications. Previous application-specific integrated circuit implementations of MD simulations used floating-point arithmetic. This was avoided here since its implementation is both resource hungry and slow. Instead, computations were done in fixed-point arithmetic and a careful precision analysis was made to ensure accuracy.

The ultimate bottleneck in the described Transmogrifier 3 implementation was memory bandwidth. Performance could be further increased with an improved memory system and the paper also detailed how this could be achieved.

The employed techniques can be applied to many applications beyond molecular dynamics and this work represents a good example of a state-of-the-art reconfigurable computing application in 2004.

Philip H. W. Leong

DOI: http://dx.doi.org/10.1109/FCCM.2004.48
This paper provided a study of two different high-level approaches to interconnecting complex processing elements on an FPGA, a compile-time-scheduled, time-multiplexed interconnect and a packet-switched interconnect with communication patterns determined at run-time.

One of the distinguishing features of field-programmable gate arrays is the user's ability to customize FPGA resources to exactly the functionality that is required. Since the introduction of the first commercial FPGAs, on-chip interconnect, rather than logic, has been the limiting resource in most devices. However, until the mid-2000s, most reconfigurable computing research projects focused on the best ways to customize computation for available logic resources, while leaving its interconnection as an afterthought.

The advent of the network-on-chip era of inter-processor communication brought greater focus onto the interconnection of FPGA processing elements, such as processor cores or multi-LUT functional blocks. This paper contrasts two distinctly different approaches to providing high-level on-chip interconnection between FPGA processing elements. One approach involves a compile-time analysis of all inter-element communication. A static communication schedule for each inter-element router is then determined. A second approach assumes that a router has sufficient knowledge to dynamically determine a data packet's route as the application executes. Although this second approach often requires more router hardware than the first, in some cases when a large communication schedule must be stored, the hardware requirements for the statically-scheduled approach may be greater than those of the packet-switched. The paper fully evaluates the tradeoffs for a series of graph-based applications.

The paper provides a thought-provoking analysis of high-level FPGA interconnection models. Since its publication numerous follow-on papers have examined both improved FPGA network-on-chip architectures and custom interconnect architectures for statically-scheduled on-chip communication.

Russell Tessier

DOI: http://dx.doi.org/10.1109/FCCM.2006.55
FCCM20 Endorsement

A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing
Michael Butts, Anthony Mark Jones, Paul Wasson

Year of publication: 2007
Area: Languages and Compute Models

This paper is notable for the design, implementation, fabrication and fielding of a production-level, single-chip, coarse-grained reconfigurable computer “Ambric” and associated programming environment (language, compiler, simulator, debugger) based on a formal parallel computation model. The Kahn Process Network model describes a collection of communicating sequential processes in which messages are transmitted over unbounded buffered channels.

Like other coarse-grained architectures, this design foregoes the bit-level flexibility of traditional FPGAs for the energy efficiency of 32-bit word-level operations. However, unlike some coarse-grained designs, the compute module is a full CPU running a clock-cycle-oblivious software program. This design choice solves the application development hurdle of FPGAs and some other ALU-based coarse-grained processor arrays: the applications are written as communicating sequential programs in a Java derivative, and compilation is truly a software compile process — i.e. fast. The architecture supports a reconfigurable, asynchronous communication network. All communication, CPU-to-CPU as well as CPU-to-memory-block, occurs via messages. Communication is expressed graphically in the Eclipse programming environment or in an equivalent textual language, solving the schematic vs. text controversy with the answer “yes.” The communication is routed using FPGA-like place and route tools, but since there are a couple of orders of magnitude fewer structured objects than primitive FPGA resources, P&R time is on the order of minutes. The architecture also supports run time reconfiguration.

The ease-of-use dimension is pushed even further with a seamless simulator/debugger and real time visualization of the application running in the simulator or in hardware. In our experience, the visualization environment was invaluable in debugging performance as well as correctness: mismatches in rates between producer and consumer causing performance loss due to stall propagation were clearly visible waves in the data flow displayed in the visualization.

The Ambric chip fabricated in the 2007 timeframe at 130nm was capable of 1 tera-ops/sec of 8-bit operations at 333MHz, consuming less than 15W, an attractive power/performance combination for embedded DSP applications. Unluckily, the economic downturn caught the company, and they did not survive the rough economic roller coaster of the next few years. Yet their design and implementation remain one of the best examples of reconfigurable compute arrays solving the programmability weakness of FPGAs while maintaining high performance.

Maya B. Gokhale

DOI: http://dx.doi.org/10.1109/FCCM.2007.7