Teramac—Configurable Custom Computing

Rick Amerson, Richard J. Carter, W. Bruce Culbertson, Phil Kuekes, Greg Snider

Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto CA 94304

Abstract—The Teramac configurable hardware system can execute synchronous logic designs of up to one million gates at rates up to 1 megahertz. A fully configured Teramac includes half a gigabyte of RAM and hardware support for large multiported register files. The system has been built from custom FPGA's packaged in large multichip modules (MCMs). A large custom circuit (~1,000,000 gates) may be compiled onto the hardware in approximately 2 hours, without user intervention. The system is being used to explore the potential of custom computing machinery (CCM).

1 Teramac System Overview

Research on special purpose parallel architectures and custom computing is very much an experimental science dependent on the existence of prototypes. We have built an FPGA-based configurable custom computing engine to enable experiments on an interesting scale.

Teramac is a configurable hardware system comprising 1728 custom FPGAs and .5 gigabytes of RAM. It features:

- 1,000,000 gate capacity for synchronous logic circuits.
- up to 1 MHz clock rate.
- .5 Gbytes of memory organized into 64 independent, 32-bit-wide banks, each with independent read and write ports. Banks may be combined horizontally and vertically to form large memories.
- Fully automatic compilation.
- Checkpoint restart capability.
- Scalability. A minimum Teramac system (a single board) supports designs of up to 64K gates. Additional boards may be added to expand the capacity incrementally, up to maximum of 16 boards.

We are currently conducting experiments with an 8 board Teramac system.

2 Hardware

The Teramac system logically consists of four major components as shown in figure 1:

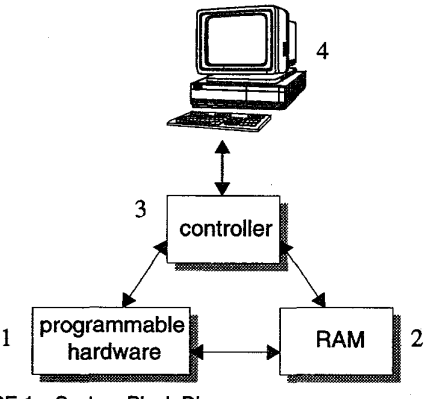

FIGURE 1. System Block Diagram

(1) programmable hardware, which is configured to functionally reproduce a user's circuit; (2) RAM, which may be incorporated into user designs requiring memory; (3) a controller, which is responsible for controlling the programmable hardware as well as exchanging configuration and state data with an external host; and (4) a host workstation which provides the center of control—user interface, compiler, and debug environment. The host connects to Teramac with a set of SCSI buses, making it easy to upgrade the host without modifying the Teramac hardware. The Teramac system is shown in figure 2.

Teramac's programmable hardware implementation is uniform: sixteen identical PC boards are interconnected with cables; each board carries four identical multichip modules (MCMs); each MCM carries 27 identical

0-8186-7086-X/95 $04.00 © 1995 IEEE

Figure 2. Teramac Hardware

FPGAs. Thus, a fully configured Teramac contains 1,728 FPGAs.

2.1 PLASMA—custom FPGA

We investigated using standard FPGA's for Teramac, but we ultimately designed our own—the PLASMA[1] chip. PLASMA is a routing-rich FPGA that consists of 6-input, 2-output lookup tables (with configurable latches and registers on their outputs), interconnected by partially populated crossbar switches. We chose a custom approach for several reasons:

Compilation Time: Placement and routing time for standard FPGA's is still much longer than is acceptable for a custom computing machine (CCM). Although the place and route times for a single chip have decreased from overnight to minutes in the last few years, this is still much greater than the 3 seconds we achieved using a custom chip. Even at 10 minutes of compilation time per chip, placing and routing 1728 off-the-shelf FPGAs would have consumed twelve days, not to mention the additional overhead of system level partitioning, placement and routing.

Pinouts: A CCM requires greater connectivity than is provided by standard FPGA's. PLASMA has 336 signal I/O pins, roughly double that which can be obtained off-the-shelf. On average, less than half of the I/O pins on any one chip are allocated by the compiler for a typical user design, but the extra pins are necessary to accommodate local "hotspots" in the design requiring greater bandwidth. In addition, placement and routing in PLASMA is insensitive to signal pin assignments.

Raw Die Availability: Building a multi-chip module requires access to tested bare die—HP's internal fab capability allowed us to contract for this. We were unable to find an external vendor who could meet our requirements for raw die and would be willing to sell them.

Custom Features: The PLASMA chip has unique features, such as support for multiported register files, which are perhaps not useful for general-purpose FPGA's. Multiported register files were effi-

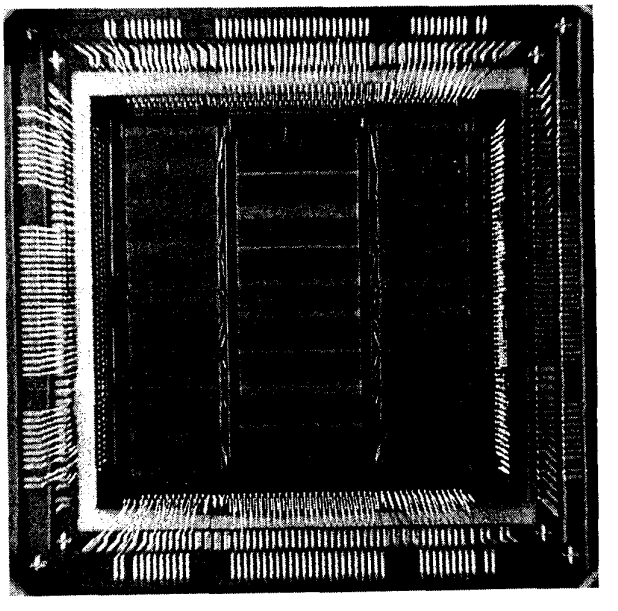

Figure 3. PLASMA FPGA

1. *PLASMA: P*rogrammable *L*ogic *A*nd *S*witch *MA*trix

ciently implemented by capitalizing on the structure of the lookup tables in our logic cells—the lookup table decoders could be reconfigured to implement read and write ports; with the addition of some register bits, we are able to configure some of the logic cells to behave as a multiported register file slice. The decode logic for a register file would have been expensive to implement in standard FPGA's.

Proprietary Configuration Formats: The vendors we considered for supplying the FPGA's had proprietary formats for internal data they were unwilling to disclose to a research project with limited volume. Thus we would have been required to use their software tools to develop a design; our users would have needed access to the same tools.

2.2 MCM Design

The Teramac MCM, a very large (6.13 by 7.4 inches) MCM-C, has 27 chips, each with 408 pads [1]. The problem we faced was one of wiring complexity in a system which would contain hundreds of 408 pad ICs. The total number of wires was such that we had to very carefully balance the costs of the MCM level of connection, the PCB level, and board to board connections. By using MCMs for the vast majority of the wires we were able to relieve the pressure on the PCBs and the board to board connections.

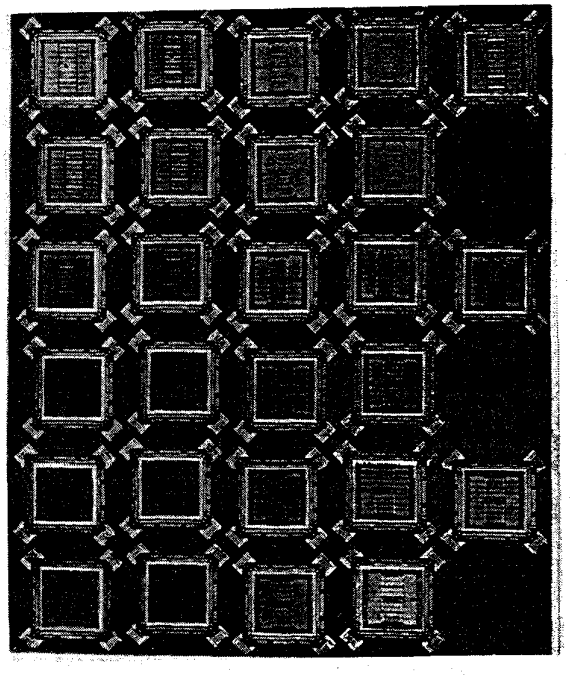

Figure 4. MCM with 27 PLASMA chips

The advantage to a very large MCM comes from Rent's Rule [2],

$$\text{I/O wires} = \text{constant} * \text{average\_pinout} * \text{chips}^{0.5}$$

By putting more chips on the MCM, a large percentage of the total wires are removed from the PCBs, which intrinsically have less capacity, being limited to under 20 layers. By using an MCM C we reduced the number of pins to be connected on the PCB from 11,016 to 3,264, a factor of 3.375 improvement compared to single chip modules (SCM's). Had we attempted to use SCM's, the area required just to mount the 27 chips on the board would have been over 5 times the area of the 27 chip MCM, assuming the board could have been routed.

The 27 chip MCM is shown in figure 4. Even though much effort went into minimizing the number of layers and vias (a major cost component), 39 layers were required (12 layers for power planes and signal spreading, 27 layers for signal routing) with 260,000 blind vias buried in the MCM. The wire length in the vias alone is over 2000 inches.

2.3 Logic Boards

Each board contains four MCMs which implement a simple network of 108 identical chips. Up to 16 boards may be interconnected via cables. Each board has 4032 signal I/O pins through cables to other boards.

2.4 Controller Boards

Each logic board connects to a daughter controller board containing four banks of 32-bit wide, 2 M deep, 2-port static RAM, and control circuitry for interfacing the board to the host computer. The RAM banks are connected into the PLASMA network to support user designs containing embedded memory. The controller circuitry: (1) relays configuration data from the host, (2) transfers state data between the memories, PLASMAs, and the host, and (3) controls clocking and breakpointing of the PLASMA network.

3 Software

3.1 Compiler

The Teramac compiler was the principle driver of the architecture. The desire for fast compilation of large designs inspired separating the problem of mapping a user circuit onto programmable hardware into smaller, independent subproblems that could be attacked with

low computational complexity algorithms. This separation guided the hardware in directions that allowed for fast compilation. As implementation problems developed or the need for choosing parameters arose, the compiler then became a tool for exploring alternatives and refinement.

The compiler operates on an input netlist in multiple passes in order to perform the mapping to the Teramac system:

Netlist Filter: The first pass of the compiler is merely a filter that transforms the user's input netlist into an internal format used by the remaining passes of the compiler. Filters have been built for Tsutsuji netlists [3], and EDIF netlists.

Merger: The second pass transforms the original circuit of logic gates into a functionally equivalent circuit of lookup tables by "greedily" packing gates into groups respecting the 6-input, 2-output limitation of the lookup tables used in PLASMA, and then computing the truth table for the resulting function (additional heuristics in this pass also unpack and repack groups and replicate gates to further reduce the number of lookup tables).

Global Partitioning: The global partitioner takes the circuit of truth tables generated by the merger and partitions them into PLASMA chips in the Teramac system. Partitioning begins by constructing a tree to represent the user's circuit. Hierarchical information in the netlist, if present, is used to build the initial tree, which is then refined through the use of ratio-cut partitioning[4]. Applying the approach of Yeh and Cheng [5], the tree guides the recursive partitioning of the circuit onto the Teramac network using min-cut partitioning algorithms [6,7]. The majority of compiler processing time is spent in this pass.

Global Placement: Based on the results of global partitioning, subcircuits are assigned to specific PLASMA chips by a global placement pass.

Global Routing: The global router routes signals between PLASMA chips. The signals to be routed are sorted in order of fan-out, and routed one at a time (highest fan-out first) through the network using a "first-fit" algorithm. The output of this pass is an assignment of signals to all of the PLASMA pins in the network.

Local Place and Route: Once the signal/pin assignments have been completed by the global router, each PLASMA chip can be individually placed and routed. Since a PLASMA chip has the same topological structure as the system, similar partitioning, placement and routing algorithms are used. The computation time depends on the complexity of the subcircuit, with 3 seconds being a typical value.

Configuration Mapper: This pass of the compiler simply maps the logical placement and routing of PLASMA chips into a configuration bitstream that can be downloaded to the Teramac hardware.

Timing Analyzer: Using a SPICE-based timing model of PLASMA and the interconnect, along with the placed and routed circuit, the timing analyzer predicts the maximum clock rate at which the compiled circuit will run in Teramac. Measured maximum clock rates for the custom computers run thus far on Teramac agree well with the timing analyzer's predictions.

The largest circuit we have compiled, containing approximately 1,000,000 gates, required about 2 hours of compilation time. Smaller circuits require much less time: a 170,000 gate circuit requires about 18 minutes, and a 60,000 gate circuit about 5 minutes (on an HP 735 workstation). After the Merger pass, compilation could be distributed across several workstations via a network to speed it up further, but we have not pursued this.

3.2 Meltdown Protection

Configurable hardware, while quite versatile, is at significant risk from user designs which may not work correctly. Multiple outputs can be accidentally connected together damaging the system, requiring expensive component replacement. To prevent this, we developed a verification program which checks every design for consistency before it is downloaded to the Teramac hardware. This program, TMID (pronounced timid), short for "Three Mile Island Defense," ensures that bugs in a user design will not physically damage the hardware. TMID verifies consistency between chips as well as within chips.

Downloading a design could potentially cause problems as configurations are being established, but the intermediate state is invalid. Chips are disabled during configuration to prevent problems with the random bit patterns which appear.

3.3 User Interface

The Teramac user interface makes it easy to focus on results, not the tool. Custom screens can be configured to display the internal state of a design in a convenient format. A filter capability allows the user to intercept the

bits before they reach the output screen and transform them to a different format, even graphics if desired. Mnemonic signal names are easily handled in user display windows. Assigning probe points on the design allows monitoring any signal in the original schematic. The user interface recreates signals that exist on the original schematic but were subsumed into lookup tables on the Teramac. Single stepping, tracing, and hardware breakpoints are built in. Breakpoints can be triggered by logic signals or configured to occur after a fixed number of cycles.

4 Experiments

Teramac has already given us some insight into the real challenges of reconfigurable computing. The following four custom engines have been implemented in Teramac. This is very much an experimental science and good debugging tools are essential when the bug in question may lie in the initial algorithm, the custom parallel design, the mapping to FPGA's by the compiler, or the prototype hardware of Teramac

Bubble Sort

One of our first experimental designs was a bubble sort engine using the multiported register files. By cascading register slices it is possible to sort very wide fields. By using the multiple read and write ports the engine could read a consecutive pair of words, swap them if needed, and rewrite them in the sorted order. Four such engines run in parallel each working on a quarter of the list to be sorted. A 20-bit wide, 64 word deep bubble sorter has been run at 1 MHz.

Graph Partitioning

The graph partitioning engine is a special purpose cellular automata which uses a random search to partition a graph into equal halves with a minimal number of edges cut between the halves. As shown in figure 5 the automata (a simple 2D mesh for this example) begins with a random assignment of cells to the partitions, labeled 0 and 1. Logic in each cell computes the total number of edge crossings, the sum of edge crossings and the absolute difference of the number of 1 cells and 0 cells to determine a "cost" of a trial partition. The configuration is randomly perturbed and the best results are kept. A "temperature" which limits the freedom to explore the configuration space is adjustable. We have run a 28 by 28 array (corresponding to a graph with 784 nodes) on Teramac. It took 651,105 clocks to find the optimal solution shown in figure 6. (This test array has an obvious optimal solution; practical designs would not). The partitioning engine has 73,000 gates and runs at 480 kHz.

Pi

Another test design built on Teramac is a systolic array for computing pi to an arbitrary number of digits. The design shown in figure 7, uses the trigonometric identity pi/4 = arctan(1/2) + arctan(1/3). All arithmetic is done in decimal. We add one cell for each digit of pi to be computed. A 120 digit pi engine has 200,000 gates and runs at 500,000 kHz.

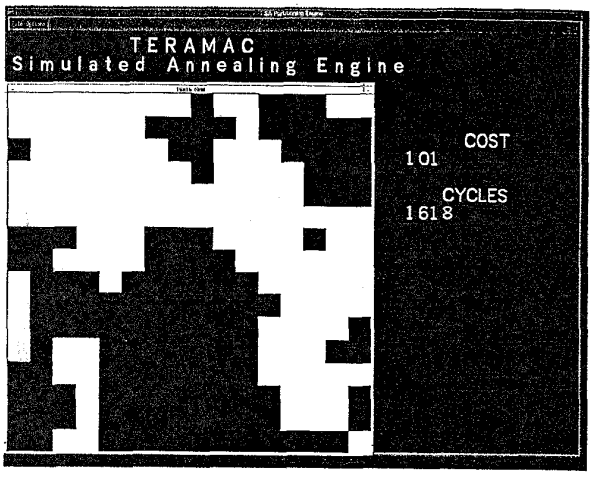

Figure 5. Initial configuration before partitioning.

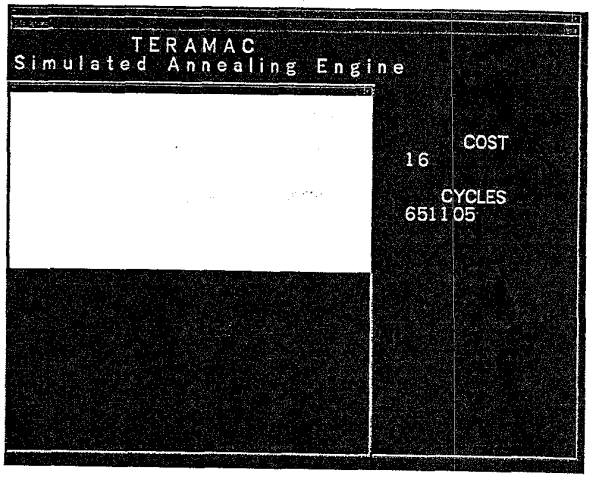

Figure 6. Configuration after partitioning.

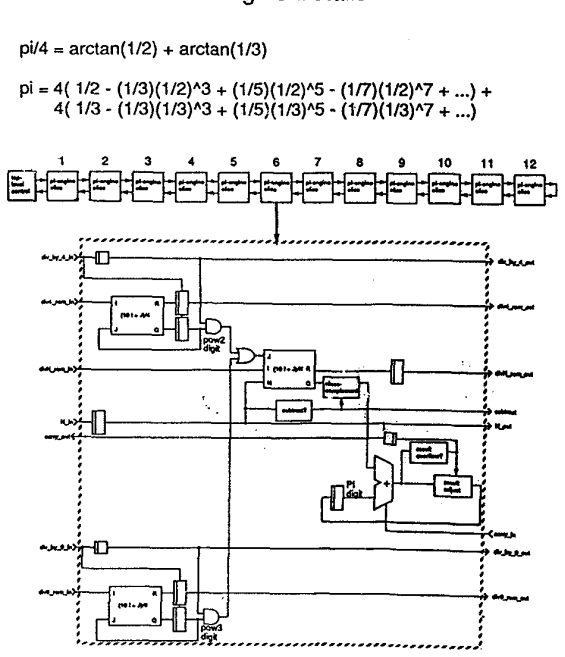

Figure 7. PI Engine

Long Integer Multiply

We have constructed some large multipliers on Teramac and believe that it holds promise for experiments with very long integer arithmetic.

DNA String Matcher

We have compiled a systolic string matcher which computes the similarity of DNA strings. It is a modified version of a family of previous designs [8,9,10]. A 5000 cell long systolic string matcher compiles to a 500,000 gate, 1.5 MHz design on an eight board Teramac.

In addition we are developing applications in the areas of cellular automata, stochastic neural nets, and multidimensional convolution. The compiler predicts that highly pipelined designs will run at close to 3 MHz.

5 Conclusions

Configurable custom computing has a promising future. Several researchers have used tens of FPGAs to create a variety of highly parallel custom machines [11-14]. Teramac will allow experiments using many hundreds of FPGAs.

The Teramac architecture provides a routing-rich environment for implementing user designs due to an investment in custom FPGA's, MCM's and PC boards. The resulting speed of compilation, tens of minutes rather than tens of hours, will allow another layer of abstraction in defining highly parallel custom computers to be used by researchers.

We have begun experiments on a variety of styles of design of custom computers. We hope that the speed of compilation and execution along with easy access to many large memory banks and multiported register files provided by Teramac will allow fruitful methods for the simple specification and synthesis of custom computers.

6 Acknowledgments

Teramac was originally conceived by Barry Shackleford and Bob Rau. Greg Snider has been the chief architect and compiler writer. Rick Amerson has provided management and contributed to the PLASMA architecture. Phil Kuekes is project manager and has contributed to both the architecture and the MCM design. Brian Jung, Sue Blockstein, Lyle Albertson, and Tom Myers designed PLASMA. Peter Maxwell developed the chip yield model for PLASMA. Dick Carter defined the user interface, created numerous custom computing designs, and contributed to the compiler. Bruce Culbertson created the system test code. Wulf Rehder developed test algorithms. Arnie Berger and Andy Blasciak designed and implemented the system hardware. Dan Kary developed the control and user interface software. Martin Guth developed the MCM wire-bonding process.

References

[1] R. Amerson and P. Kuekes, "A Twenty-Seven Chip MCM-C," Proceedings of the 1994 International Conference on Multichip Modules, pages 578-582.

[2] B. Landman and R. Russo, "On a Pin vs. Block Relationship for Partitions of Logic Graphs," IEEE Transactions on Computers, December 1971, pages 1469-1479.

[3] W. B. Culbertson, T. Osame, Y. Otsuru, J. B. Shackleford, M. Tanaka, "The HP Tsutsuji Logic Synthesis System," Hewlett-Packard Journal, August 1993, pages 38-51.

[4] Y. C. Wei and C. K. Cheng, "Toward Efficient Hierarchical Designs by Ratio Cut Partitioning," Proc. IEEE International Conference on Computer- Aided Design, 1989, pages 298-301.

[5] Ching-Wei Yeh and Chung-Kuan Cheng, "A General Purpose Multiple Way Partitioning Algorithm," Proc. 28th ACM/ IEEE Design Automation Conference, 1991, pages 421-426

[6] Balakrishnan Krishnamurthy, "An Improved Min- Cut Algorithm For Partitioning VLSI Networks," IEEE Transactions on Computers, Vol C-33, No 5, May 1984, pages 438-446.

[7] Laura A. Sanchis, "Multiple-Way Network Partitioning," IEEE Transactions on Computers, Vol. 38, No. 1, January 1989, pages 62-81.

[8] R. J. Lipton and D. P. Lopresti, "A Systolic Array for Rapid String Comparison," 1985 Chapel Hill Conference on VLSI, H. Fuchs, Ed. Computer Science Press, pages 363-376.

[9] Daniel P. Lopresti, "P-NAC: a Systolic Array for Comparing Nucleic Acid Sequences," Computer, July 1987, pages 98-99.

[10] Dzung T. Hoang, "Searching Genetic Databases on Splash 2," Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, 1993, pages 185-191.

[11] Patrice Bertin, Didier Roncin, and Jean Vuillemin, "Introduction to programmable active memories," in Systolic Array Processors, Prentice-Hall, 1989, pages 301-309.

[12] J. M. Arnold, D. A. Buell, and E. G. Davis, "Splash 2," Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, 1992, pages 316-322.

[13] J. Babb, R. Tessier, and A. Agarwal, "Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators," Proceedings, IEEE Workshop on FPGA-based Custom Computing Machines, Napa, CA, April 1993, pages 142-151.

[14] S. Casselman, "Virtual Computing, " Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, April 1993, pages 43-48.