Submicron Systems Architecture

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

5202:TR:85

September 1985

Reporting Period:	16 March 1985 to 15 September 1985
Principal Investigator:	Charles L Seitz
Faculty Investigators:	James T Kajiya
	Alain J Martin
	Robert J McEliece
	Martin Rem
	Charles L Seitz

Sponsored by the
Defense Advanced Research Projects Agency
ARPA Order Number 3771

Monitored by the
Office of Naval Research
Contract Number N00014-79-0597

SUBMICRON SYSTEMS ARCHITECTURE

Department of Computer Science
California Institute of Technology

1. Overview and Summary

1.1 Scope of this Report

This document is a summary of the research activities and results for the six month period 16 March 1985 to 15 September 1985 under the Defense Advanced Research Project Agency (ARPA) Submicron Systems Architecture Project. Technical reports covering parts of the project in detail are listed following these summaries, and can be ordered from the Caltech Computer Science Library.

1.2 Objectives

The central theme of this research is the architecture and design of VLSI systems appropriate to a microcircuit technology scaled to submicron feature sizes, and includes related efforts in concurrent computation and VLSI design. Additional background information can be found in previous semiannual technical reports [5052:TR:82], [5078:TR:83], [5103:TR:83], [5122:TR:84], [5160:TR:84], [5178:TR:85].

1.3 Highlights

Some highlights of the previous 6 months are:

- Cosmic cube host software completed (2.1.2) and available for distribution (2.1.3).
- Robust Mosaic RAMs (2.2).
- Experimental Concurrent Language developments (3.1).
- New method for deadlock-free routing on arbitrary networks (3.5).
- Torus Routing Chip completed (4.1).

1.4 Acknowledgement

Please let us take this opportunity — the semiannual report for what Paul Losleben claims is his last VLSI contractor's meeting before leaving DARPA — to thank and to congratulate Paul Losleben for his efforts as manager of the DARPA VLSI program.

2. Architectural Experiments

2.1 Cosmic Cube Project

W C Athas, Reese Faucette, Michael Lichter, Wen-King Su, Chuck Seitz

The Cosmic Cube's host environment was substantially refined in the past six months. Features such as "space-sharing" that were mentioned as plans in our previous report are now operational and in routine use. The entire programming system is stable, and most of our current work is involved with using the system. Typical applications include (1) concurrent algorithm experiments, (2) concurrent programming language experiments, (3) computer-aided design and analysis tools, and (4) simulations of other concurrent architectures.

In addition, the project group has recently received an Intel iPSC d6, scheduled to be upgraded to an iPSC d7 in October. The iPSC d7, with 128 nodes and a total of 64M bytes of primary storage, is a very high performance system for the applications listed above. The system software of the iPSC is not as mature as that of the Cosmic Cube, however we are in the process of improving this software.

In the interests of encouraging our colleagues in the DARPA community who may wish to use these systems, a copy of the latest "C Programmer's Guide to the Cosmic Cube" [5203:TR:85] is reproduced at the end of this report. Our 3-cube and 6-cube machines, and Intel iPSC are ARPAnet hosts. Also, the "cosmic environment" software – a runtime system and library that supports the same multiple-process message-passing environment on Unix 4.2bsd computers – is available for distribution. This system is described in Chapter 7 of the "C Programmer's Guide to the Cosmic Cube".

2.1.1 Node Operating System

The work on the Cosmic Cube's node operating system, the Cosmic Kernel, has progressed from development to maintainance and tuning. Very few changes have been made in the inner kernel. Several refinements have been accomplished in the two outer kernel processes, SPY and SPAWN. For example, the necessity for the `NEVERRUN` process state was eliminated, and a Unix influenced "fork" operation has been included that creates new copies of running processes was added.

A logic analyser was used to monitor the different subsystems of the Cosmic Kernel. The logic analyzer showed that the low level send/receive subsystem was not set-up for maximum overlap. Minor improvements that we are planning are to improve the synchronization between read and write operations and a more robust deadlock avoidance mechanism. However, a recent benchmark on the 6-cube showed message passing speed to be close to the 2Mbit/s hardware and interrupt routine limit.

We have a plan to create an alternate inner kernel by converting the present inner kernel code from 8086 assembly language into C. This effort is a first step in producing a portable kernel for possible future machines of this type. It is also intended to make it easier to do experiments in which we can run user programs under kernels that implement the message system differently.

2.1.2 Host Software

The host programming system of the Cosmic Cube, the "cosmic environment", has been substantially rewritten over the past six months. Although this version 6 is a major internal revision, it has involved very few changes in user programs from the version 3 system of 6 months ago. What changes the user has seen have been in the nature of "cleaning up" and simplifying some of the definitions and conventions, and have required only recompiling source programs.

The version 3 system had a number of restrictions based on the total number of socket connections that could be opened to the cube dæmon. It was possible for a user program to run out of socket connections depending on the number of others who were using the same cube dæmon. Also, the user was at the mercy of the limited queueing supplied by the 4.2bsd socket implementation. The versions 5 and 6 implementations have moved the socket connections and flow control into the `getcube` process of each process group. The version 6 system also includes a network-based update mechanism, described in the following section.

The C compiler used for cube process code was also rewritten in this period.

2.1.3 "Cosmic Environment" software distribution

The *cosmic environment* software described in the "C Programmer's Guide to the Cosmic Cube" is available for internal use to most university, government, and non-profit research organizations under the arrangements outlined below. This software may be of interest to other contractors both for concurrent computing experiments and for use in courses. Software distributions are available to other organizations only through a license with Caltech. It is expected that another version of this software will be available through Intel Scientific Computers.

The cosmic environment is package of programs, C library routines, and dæmon processes that supports on one or on multiple network-connected Unix systems the same message-passing concurrent process programming environment that runs on the Cosmic Cube. Process code is compiled with a set of definitions and library routines and run as Unix processes. The interprocess communication mechanism used by this system is internet sockets. Processes may accordingly be run all on one host or distributed across a number of network-connected hosts. This software can also be used as the host environment for Cosmic Cube programs, or in the future, for Intel iPSC programs. (See the "C Guide" for additional details.)

These programs were written to run under Berkeley Unix 4.2bsd, and have been installed and used on SUNs, Berkeley Unix VAXes, and Ultrix VAXes. We do not know whether the system runs correctly or can be made to work on other Unix systems. A minimal necessity to be able to adapt these programs to other Unix systems is a socket implementation.

This software also comes with a network-based update mechanism that is recommended for ARPA internet hosts.

In order to get a copy, send a letter to Professor Charles L Seitz, Computer Science 256-80, California Institute of Technology, Pasadena CA 91125, that includes statements:

1. of your intended use of this software (brief).
2. that you agree not to redistribute this software outside of your organization, and in no case outside of the United States, and will keep the files protected on your systems.
3. that you agree that this software is for your internal use, and that you will not sell access to or otherwise permit use of these programs to people outside your own organization.
4. that you understand that this is experimental, unsupported software that may or may not work as described, and that you hold Caltech harmless from any loss or damage that may result from its installation or use.

We are able to answer questions about this software as our time allows, and are happy to have bug reports, but only by electronic mail to "`cube@cit-sol.arpa`". It is most helpful if a bug report includes the smallest example of code that you can devise that demonstrates the bug.

If you are installing this software on a machine that is not an ARPAnet host, please include with your letter a blank 600' 1/2 inch tape labeled with your postage address. We will mail the C sources to you in Unix `tar` format. The files can be placed in any source directory (we recommend a `cube` login directory). The installation instructions are included in the `make` file.

If you are installing this software on an ARPAnet host, you do not need to send a tape. We hope you don't. Just send us your ARPAnet mail address and the name of the host on which you intend to maintain your source directory. We will then enter the name of the host on which you maintain the sources into a permission table, and will send you a message that includes a C program that you can compile and run to bring the sources over the ARPAnet to your machine. This program makes a socket connection to the cube dæmon that runs on `cit-sol.arpa`, and brings the files across the network. This same program can be used later for automatic updates, which are selective by timestamp in order to minimize the volume of messages sent through the network.

Requests to use the Cosmic Cubes or Intel iPSC d7 operated by the Caltech Computer Science project group should be addressed to "`chuck@cit-vlsi.arpa`".

2.1.4 Hardware Status

Our 6-cube system finally exhibited a hard failure after running continuously without a failure for 14 months. The two previous hard failures since the machine was put into service 23 months ago were both in dRAM chips. The failure observed this time was in an SSI communication channel driver chip. The bad board was swapped into the 3-cube for testing and diagnosis, and downtime was only a few minutes after the problem was first noticed. We also took advantage of this opportunity to experiment informally with using space sharing to keep the system in service in spite of bad or absent node boards. Users have been able to utilize subcubes while a portion of the cube has not been functioning or was running test programs.

The 6-cube alone now has logged 16,500 hours of operation. Total node-hours accumulated on all machines are now over 1,500,000, with 3 hard failures. The calculated MTBF of the nodes of 100,000 hours reported before this machine was constructed can now be regarded as conservative at the 99% confidence level, and an MTBF of 200,000 hours can be stated at a 95% confidence level.

Both machines continue to be heavily used.

2.1.5 Application programs

We continue to make the Cosmic Cube available to selected guest researchers for problems that appear to be particularly interesting or difficult. For example, several researchers from the Aeronautics group at Caltech report very good results on a very difficult jet plume computation. In addition, the following small sample of programs from the research group are suggestive of our more interesting applications of the Cosmic Cube.

The concurrent circuit simulator CONCISE was translated last May from Pascal to C, using the HCR Pascal to C translator. It now runs correctly as a concurrent program under the "cosmic environment" system. It continues to be developed and refined by Sven Mattisson in a joint project between Caltech and the University of Lund, Sweden.

A concurrent circuit extraction algorithm and program were developed as a student project, and will be developed into a useable form.

Reese Faucette, author of the cosmic kernel, wrote a program that uses the 6-cube to produce a high resolution graphic image based on a simple mathematical model taken from Mandelbrot's fractal geometry. Another of Reese's programs, virus, starts up two-sided games of 3-dimensional tic-tac-toe throughout a cube, with "mutations" occuring in the playing parameters of losing players, so that the overall playing level of a cube of processes rises with time.

2.2 Mosaic Project

Steve Rabin, Don Speck, Chuck Seitz

A 16-inch stack of SPICE simulation listings and several layout modifications to the Mosaic dynamic RAM module have finally converged to a version that is reasonably tolerant of process, timing, and power supply variations. The most recent chips received from MOSIS had all bits working, could tolerate clocks of 3.5 to 7.5 volts, and ran as fast as the tester. The yield on these recent runs was poor in contrast to our previous experience with chips with a similar active area, number of transistors and contacts, etc. Checking the defective chips under a microscope revealed missing contacts, shorts, and a few large-area poly defects. Thus we attribute the poor yield to the fabrication rather than to the design.

The Mosaic RAM seems to prefer a less "hot" clock than we had envisioned. This may allow us to eliminate the 7-volt power supply and remove some of our power dissipation worries.

Now that the RAM design looks usable, we can safely return to full Mosaic element assembly. The floorplan and schematics are done, leaving us with the layout of the ROM and routing still to do.

The floorplan is designed to keep the die *width* under five millimeters to allow packaging the chips in the same type of 18-pin large well DIPs that are used for commercial dRAMs. Far from being a hindrance, this constraint resulted in an elegant structure, with the data bus running the length of the chip and processor, ROM, and RAM modules lined up on both sides, and pads at each end. We chose to limit the length to 7 millimeters, to avoid pushing our luck on the yield. This 5mm$\times$7mm allows us 4 RAM modules (1K words total) per node.

A preliminary design of the top-assembly of the Mosaic has been completed. The Mosaic elements will by this design be packaged with 64 elements on each of 16 fairly small PCBs that can be plugged into different backplanes for either (Ultracomputer style) shuffle or mesh networks. Some recent results on routing on meshes (see section 3.5) have made the mesh connection more attractive than it formerly appeared to be, and we could very much simplify the top assembly by using a simple toroidal mesh structure.

A simulator and an assembler for the version B Mosaic architecture have been completed, and the floating point arithmetic code, and other short programs have been extensively simulated using this simulator and assembler. The datapath artwork for the version B architecture has been completed; the control and I/O sections have not.

2.3 Future Architecture Experiments

The research group has decided to defer building a fine grain SIMD message-passing machine along the lines of the Supermesh design previously reported. Having been set back in our schedule on the Mosaic project by problems with the Mosaic RAM, we expect to see the first working Mosaic systems early in 1986. We appear to be able to build new experimental systems at intervals of about 18 months. Our work with fine- and medium-grain MIMD message-passing machines such as the Cosmic Cube and Mosaic currently has a great deal of momentum, and we have decided that we would do better to concentrate in that area for the immediate future.

In our experience with distributed operating systems, in "pushing the envelope" of applications of these machines, in some of the recent results described in the following section, and in the appearance of much higher performance single-chip processors, we see an opportunity to contribute to a second generation of these MIMD message-passing machines. These machines will exhibit about 10 times better processor performance and 100 times better communication performance – particularly with short messages –, with the principal objective of extending the application span and programmability. Some of our ideas in these areas are described more completely in the following section.

What we are planning, as described in previous reports under the heading of "advanced technology homogeneous machines", is to start construction in the summer 1986 of an experimental system based on the DEC microVAX II processor and floating point coprocessor, 1MB of primary storage, and a custom communication chip. The first version of this communication chip, which was designed in the new MOSIS SCMOS technology, is described in sections 3.5 and 4.1, and it has already been submitted for fabrication. The system will employ a toroidal mesh connection rather than a binary n-cube. The size of the system is expected initially to be 256 nodes, a 16 by 16 mesh, and of course will be expandable. DEC has agreed verbally to contribute the microVAX II chips and Ultrix source licenses. The operating system kernel will be derived from the Ultrix (Berkeley Unix) kernel (VMUNIX). All aspects of this plan are subject to change depending on new developments.

3. Concurrent Computation

3.1 Experimental Concurrent Language

Bill Athas, Chuck Seitz

A highly concurrent object-oriented language, tentatively named ECL for "Experimental Concurrent Language", has been designed and exists now as a compiler, software engine, and technical report [TR:85:5196]. The unit of computation for ECL is the "concurrent object" which can interact with other such objects only by message passing. ECL is based on the computational model of the Cosmic Cube experiment, including the weak synchronization properties of the message system, but abstracts away many of the details currently necessary to program the Cosmic Cube. These details, such as the management of message locks and the automatic setting-up of message descriptors are handled by the compiler through extensive program flow analysis.

The "process abstraction" used in existing Cosmic Cube programs is captured by ECL's concurrent objects. To soldify both the semantics and syntax of the language, a test suite of four programming examples were chosen as a basis for resolving design issues about the language. These examples were a Hamming decoder, solution of the 8-queens problem, Gaussian elimination, and the generation of primes by sieving. The concurrent formulations and their solutions written in ECL have been included in the technical report. The solutions represent a wide range of concurrent formulations and have been compiled and the results checked using the first software engine.

The software engine is a microcosm of the cosmic environment and compiled ECL programs could readily be ported to such an environment, provided that a special resource allocator or "real estate" process be included. The current plan, however, is to perform static analysis of the code generated by the compiler so that resource allocation may be conducted at load time rather than at runtime, whenever possible.

3.2 The Sneptree

Pey-yun Peggy Li, Alain J Martin

Compared with other augmented binary trees, the Sneptree is shown to be the only network that can simulate an arbitrary size binary tree well, and also has regular and symmetric topology, thus is suitable for VLSI implementation.

An improved routing algorithm for the leaf nodes of the Cyclic Sneptree has been developed. The routing algorithm always finds the shortest path within the minimal subtree containing the source and the destination nodes. The routing result is a good approximation to the optimal result, yet still can be found in $O(n)$ time, where n is the height of the Sneptree. In some specific communication patterns, such as shifting by 2^k, the average routing result is almost optimal. Besides, the traffic at the upper-level nodes is reduced to about a half of the traffic in a pure binary tree.

Many interesting results of the Sneptree can be found in the technical report [5194:TR:85].

3.3 The Sync Model for Parallel Prolog Processing

Pey-yun Peggy Li, Alain J Martin

The research of the machine model for parallel Prolog processing has continued in last six months. Our Sync Model combines data flow concepts with a special synchronization mechanism to realize AND parallelism. The dynamic data flow diagram of sibling AND processes is constructed during the time the AND processes are invoked. The change of the data flow diagram caused by binding a variable to a partially instantiated term is detected by a simple type checking, and is realized by adding extra dynamic links between AND processes. The above method is more efficient than the previous data flow models. An efficient merge algorithm has been designed to merge the multiple input streams in an AND process or the partial solution streams in an OR process. The merge algorithm is capable of handling a set of finite input streams with arbitrary combinations of Sync signals. The correctness proof of the merge algorithm as well as the Sync Model is being developed.

Different mapping algorithms for mapping our Sync Model to the Sneptree have been investigated. The one we selected is to map our Sync Model to a binary tree by adding extra padding nodes, and then map the binary tree onto the Sneptree. This algorithm turns out to have the maximal parallelism among other mapping algorithms.

3.4 Concurrent Computational Geometry

John Ngai, Chuck Seitz

The appearance of parallel computers has motivated the development of parallel algorithms and distributed data structures for solving problems by exploiting the concurrency of these machines. Until recently, very little work has been done in developing parallel algorithms and distributed data structures to solve geometric problems. Computational Geometry is the systematic study of algorithms and complexity of fundamental problems such as inclusion, intersection, hull finding, and proximity searching, all of which are geometric in nature. Such problems arise in many conventional applications, such as CAD, Robotics, Data Base Retrival, and Operations Research, as well as in many military systems. Traditionally, such problems are solved by uniprocessor systems employing complicated data structures.

Our research focuses on the investigation of intrinsic geometric properties of such problems to reveal concurrencies that can be exploited in the concurrent formulation of their solutions. Proceeding in this direction, we have been able to develop fast sublinear parallel algorithms for the determination of the convex hull of point sets in the Euclidean two- and three-space. The complexity of our algorithm for the two-dimensional case matches the best known in published literature, for example, $O(\log^2 N)$ on the binary n-cube. To the best of our knowledge, no corresponding parallel algorithm for the three-dimensional convex hull has been published. By exploiting the criteria derived from the normal vector representations, we are able to generalize and obtain sublinear algorithm for spatial point sets. A detail report of these algorithms is being prepared and will be submitted for publication.

Current work on concurrent computational geometry is concentrating on convex decomposition and geometric searching. Parallel algorithms for multiple point searching in planar subdivisions have been developed. Investigations to generalize them to other objects and to higher dimensions are in progress.

3.5 Deadlock Free Routing

William Dally, Chuck Seitz

We have developed a method for constructing deadlock-free routing algorithms in arbitrary strongly-connected interconnection networks. To construct a deadlock-free routing algorithm, we consider the routing function to be of the form $C \times N \rightarrow C$ (the current channel and the destination node determine the next channel) rather than the conventional $N \times N \rightarrow C$. A routing function from channel to channel describes a channel dependency graph, D. We have proved that a routing function is deadlock free if and only if D is acyclic.

To make a routing function that describes a cyclic D deadlock-free, we have developed the concept of virtual channels. Several virtual channels, each with its own queue, can be multiplexed over a single physical channel. It is the interaction of queues that determines the channel dependencies and thus the deadlock properties of the network. A routing function that has a dependency graph with a single cycle can by made deadlock-free by splitting each physical channel around the cycle into two virtual channels and specifying the routing on the virtual channels so that their dependency graph describes a helix. Routing functions with several cycles can be treated by splitting each cycle in turn.

This technique, described in full in a preprint reproduced at the end of this report, has proved to be a key idea for our plans for advanced technology homogeneous machines. The Torus Routing Chip described in section 4.1 is a demonstration of this technique.

3.6 Concurrent Graph Algorithms

William Dally, Chuck Seitz

We have developed two new concurrent algorithms for the max-flow problem. The concurrent augmenting digraph (CAD) algorithm works by propagating an augmenting directed graph rather than a single augmenting path as in the Ford-Fulkerson algorithm. The concurrent vertex flow (CVF) algorithm, on the other hand, works by pushing flow from one vertex to another. It is similar in spirit to Karzanov's algorithm and achieves the same worst case bound of $O(V^3)$.

We have developed a new concurrent heuristic algorithm for graph partitioning. Based on the Kernighan and Lin algorithm, our algorithm works by iterative improvement. Unlike K & L, however, the new algorithm may move several vertices at once. A two-pass attraction algorithm is used to prevent multiple moves from interfering with one another. An embedded tree is used to assure that the partition remains balanced.

3.7 Generalizations of Message Routing Mechanisms

Craig Steele, Chuck Seitz

This investigation is seeking generalizations and extensions of the current message routing mechanisms for message-passing concurrent computers. Communications deadlock must be avoided in such systems, but can occur when a message system cannot deliver any message already in the system due to competition from other messages in the system. An efficient method – "distance queues" – for constructing deadlock-free store-and-forward message systems has been found for networks of arbitrary connectivity. We can easily find minimal cost routings for powerful but complex interconnections such as the shuffle-exchange-shift network. In addition, the ability to automatically generate deadlock-free routes for irregular networks allows a distributed processor to tolerate failure of some nodes or communications links. If process placement is performed by the previously reported technique of simulated annealing, automatic reconfiguration and restart of a computation is possible even in the event of serious hardware faults.

For toroidal mesh networks it is possible to generalize the e-cube deadlock-free routing scheme used in the Cosmic Cube. These networks allow a trade-off to be made between communications performance and the cost of implementing the connections. With dedicated virtual circuits, it is possible to emulate toroidal meshes on an arbitrary physical network. The cost of such emulation is minimized by using simulated annealing to find a good embedding. This approach to the problem of deadlock-free routing allows us to decrease undesired competitive interaction of messages by increasing the buffering resources allocated.

Even a deadlock-free message system can be overwhelmed and deadlocked by unconstrained use of its buffering capabilities. Currently, Cosmic Cube programs written without explicit synchronization depend on process placement and the particular hardware characteristics for correct operation. Current research is examining extensions of the present Cosmic Kernel mechanisms to allow such programs, written in a convenient "pipelined" or "data flow" style, to run reliably on arbitrary networks.

3.8 Semantics of Infinite Lists

Young-il Choo, Jim Kajiya

To define rigorously the meaning of higher order infinite lists (lists which can contain infinite lists as elements recursively), we were led to topological idea of limits. The domain of infinite lists constructed by using the inverse limit method provides a formal basis for dealing with infinite objects in an implicit manner with lazy evaluation as the essential programming language construct. Full discussion of the technique and its applications in programming language sementics can be found in the Technical Report *An Inverse Limit Construction of a Domain of Infinite Lists* [5188:TR:85].

3.9 On Representing Petri Net Behavior

Young-il Choo, Jim Kajiya

Petri nets are mathematical constructs for modeling concurrent activity. Their structure is given by a bipartite directed graph; their behavior is determined by the movement of tokens around the graph. The behavior is usually represented as the set of strings generated by the firings of the transitions.

Concurrency Algebra is an algebraic framework for reasoning about the dynamic behavior of Petri nets. Based on a simple algebra containing concatenation, choice, and shuffle of strings, concurrency algebra uses a substitution schema determined by the structure of the given Petri net to generate new terms that denote its behavior. For non-terminating Petri nets, the behavior becomes the smallest set of strings satisfying a recursive equation over sets of strings.

Full details can be found in the Technical Report *Concurrency Algebra and Petri Nets* [5190:TR:85].

4. VLSI Design

4.1 The Torus Routing Chip

William Dally, Chuck Seitz

As a demonstration of the use of virtual channels to perform deadlock-free routing we have designed a custom VLSI integrated circuit: the torus routing chip (TRC). The TRC performs packet routing in arbitrary k-ary n-cube or torus networks. Each chip provides a single node's communications needs in two dimensions. The chips are designed so they can be cascaded to construct networks of higher dimension. The TRC uses virtual channels to break the channel dependency cycles inherent in this class of networks.

In addition to virtual channels, the chip incorporates several other innovative features. First, it is completely self-timed. Each of the two virtual channels in each dimension and the processor channel, as well as the internals of the chip, operate completely asynchronously. Arbiters in the chip's crossbar switch perform all required synchronization. Second, the chip uses incremental routing. A header byte specifies the distance to be routed in each dimension. After routing in one dimension is complete, the byte is stripped off and the message is passed to the next dimension. Third, the chip uses through-node or "wormhole" routing. Rather than storing the entire message in a node and then forwarding it, the message is routed directly to its destination. This technique makes message latency almost independent of distance, and avoids consuming storage bandwidth in routing nodes. Finally, the chip is intended to be fast. It uses byte-wide communication channels, and τ-model calculations indicate that each channel should operate in excess of 20MHz (160Mbits/s).

The torus routing chip has been laid out in SCMOS technology using the Magic system, simulated using MOSSIM, and submitted to MOSIS for fabrication.

4.2 Compiling Programs into Self-timed VLSI Circuits

A. J. Martin

We have developed a method for "compiling" a high-level description of a computation (a set of communicating processes) into a self-timed VLSI circuit. Self-timed (or delay-insensitive) circuits are sequential circuits in which the sequencing is entirely enforced by communication. There is no clock, and no assumption is made on the propagation delays in operators and wires. The advantages are many: First, with the increasing size of circuits, it becomes more and more difficult to distribute safely a clock signal across a chip. Second, clocked circuits rely on worst-case assumptions on the timing behavior of the components, which decreases their performances. Third, with no restriction on the length of wires, layout is facilitated.

The compilation is systematic and essentially relies on the four-phase handshaking implementations of communication actions. The program of each process is compiled into a set of "production rules" from which all explicit sequencing has been removed. By matching these production rules to those describing the semantics of the VLSI-operators (and-gate, or-gate, C-element, arbiter, flip-flop, etc), the programs are identified with networks of operators, *ie*, self-timed circuits.

The method has been applied to a whole spectrum of problems, some of them quite difficult, like distributed mutual exclusion and fair arbitration. The results are far beyond our expectations. For most circuits, especially complex ones, the compiled circuits are superior to their "hand-designed" counterparts.

In particular, the method has allowed us to solve an open problem: it had been conjectured and even "proved" that it is impossible to construct a delay-insensitive, fair arbiter. We have been able to demonstrate that the conjecture is false by constructing such an arbiter, applying our method.

Additional information can be found in: A J Martin, "The Design of a self-timed circuit for distributed mutual exclusion", *Proceedings of the Chapel Hill Conference on VLSI*, May 1985, 245-259 (a preprint was reproduced in our previous semiannual technical report [5178:TR:85]), and in: A J Martin, "A delay-insensitive fair arbiter", [5193:TR:85].

4.3 A Circular Geometry GDR checker

Steve Rabin, Chuck Seitz

A circular geometry (lines and arcs) design rule check program based upon the ideas of Ivan Sutherland has been written to support the verification of the Caltech layouts that use the Boston Geometry style. The program is functional but incomplete at this point. As far as we know, this is the first program ever written to automate the checking of artwork containing arcs and not based on a regular grid.

4.4 Other efforts

There are several other ongoing efforts in VLSI design that are not summarized here due to the researchers being away from Caltech during the month previous to this report. Certain other activities, such as the VLSI teacher's course 22 July – 2 August 1985, are all but impossible to describe.

Attachments:

Tech report list

C-guide

Dally/Seitz Deadlock Free Routing paper