12

COMPUTER SYSTEMS LABORATORY

DEPARTMENTS OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
STANFORD UNIVERSITY · STANFORD, CA 94305

Research in VLSI Systems

Final Technical Report for

March 1981 - June 1983

Computer Systems Laboratory
Information Systems Laboratory

This work was supported by the Defense Advanced Research Projects Agency, contract MDA 903-79-C-0680.

Research in VLSI Systems

Final Report for March 1981 - June 1983

June 1983

Research in VLSI Systems
DARPA Contract No. MDA903-79-C-0680
DARPA Order No. 3773

Principal Investigators: John Hennessy and Thomas Kailath
Computer Systems Laboratory and Information Systems Laboratory
Monitored by P. Losleben

Department of Electrical Engineering
Stanford University
Stanford, California 94305

Abstract

This report summarizes progress in the DARPA funded VLSI Systems Research Projects from March 1981 to June 1983, inclusive. The major areas under investigation have included: analysis and synthesis design aids, applications of VLSI, special purpose chip design, VLSI computer architectures, signal processing algorithms and architectures, reliability studies, hardware specification and verification, and VLSI theory. The major research problems are introduced and progress is discussed.

Key Words and Phrases: VLSI, design automation, computer-aided design, special purpose chips, VLSI computer architecture, signal processing, routing, layout, memory reliability, VLSI theory.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER MDA-903-79-C-0680(Final)	2. GOVT ACCESSION NO. AD-A136 413	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Research in VLSI Systems		5. TYPE OF REPORT & PERIOD COVERED Final Report March 1981 - June 1983
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John Hennessy & Thomas Kailath		8. CONTRACT OR GRANT NUMBER(s) MDA 903-79-C-0680
9. PERFORMING ORGANIZATION NAME AND ADDRESS Stanford University Computer Systems Lab & Information Systems Lab Stanford, CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects 1400 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE December, 1983 / 13. NO. OF PAGES 33
		15. SECURITY CLASS. (of this report) Unclassified
14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office) Office of Naval Research (Stanford Branch) Stanford University Stanford, CA 94305		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this report)

Unclassified; approved for general distribution.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)

18. SUPPLEMENTARY NOTES

This contract is awarded under Basic Agreement MDA 903-79-C-0680, issued by Defense Supply Service-Washington. This research is sponsored by Defense Advanced Research Projects Agency (DARPA).

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

VLSI, Design Automation, Computer-Aided Design, Special Purpose Chips, VLSI Computer Architecture, Signal Processing, Routing, Layout, Memory Reliability, VLSI Theory

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report summarizes progress in the DARPA funded VLSI Systems Research Project from March 1981 to June 1983, inclusive. The major areas under investigation have included: analysis and synthesis design aids, applications of VLSI, special purpose chip design, VLSI computer architectures, signal processing algorithms and architectures, reliability studies, hardware specification and verification, and VLSI theory. The major research problems are introduced and progress is discussed.

Table of Contents

Executive Summary

The major progress of note during this contract is summarized below:

1. *MIPS: A VLSI Processor.* MIPS (Microprocessor without Interlock between Pipe Stages) is a project to develop a high speed (> 1 MIP) single chip 32-bit microprocessor. During this contract the processor design was completed and submitted to a both a MOSIS run and a run in Stanford's Fast Turnaround Laboratory. In addition, we developed compilers for C, Pascal, and Fortran. We also implemented a code reorganizer (which is instrumental in efficient scheduling of the pipe stages), an assembler and a simulator.

2. *SUN Workstations* We completed the design and internal fabrication of the final SUN workstation design. This design was transmitted to a number of companies for fabrication on both a board basis and as a completed design. Extensive software developments were completed for the SUN; the most important was leaf, which provides page file access over the network.

3. *Geometry Engine* The Geometry Engine is a high-performance, floating point computing engine for geometric operations in 2D and 3D computer graphics. Multiple copies of the Geometry Engine provide a parallel computing system with very high- performance. (5-10 million floating point operations per second.) Enough copies of the custom VLSI design have been implemented to construct a complete prototype (see IRIS).

4. *IRIS Workstation* The goal of the IRIS workstation project was to design a high-resolution, color, extra high-performance graphics workstation that utilized all of the features of the Geometry Engine and was software-compatible with the SUN 68000 processor (excluding graphics software compatibility). The system was implemented successfully and has been copied for various other labs.

5. *Modular Model of Event-Based Concurrent Systems.* We have constructed a model that can be used in specifying and verifying concurrent systems. The formal model has two major components: a structural algebra for describing module interconnection structures, and a behavioral semantics that defines the function computed by a network of modules. We have shown that the model has a number of attractive mathematical properties and have used it to analyze and verify several composite modules, including a memoryy cell, shift register, and a flip-flop synthesized from gates.

6. *Computer Suypport for a Fast Turnaround Laboratory.* We completed the planning and exploratory stages of a project to provide extensive automation and computer support for the Fast Turnaround Laboratory. This ambitious interdisciplinary project (involving researchers from Computer Systems, Integrated Circuits, and Solid State Laboratories) will provide control, documentation, training, portability, repeatability, and efficiency in the area of IC fabrication processes.

7. *TV: An nMOS Timing Analyzer.* TV and IA are timing analysis programs for nMOS VLSI designs. Based on the circuit obtained from existing circuit extractors, TV determines the minimum clock duty and cycle times. TV was developed during this contract period and was used heavily in the MIPS project.

8. *Logic Simulation System* We developed a logic simulation system which was used to support the development of the MIPS chip.

9. *SILT, Stanford Intermediate Language for Topology* Completed the design and implementation of SILT, a graphic-based language langauge for describing designs that support relative layout, hierarchical naming structures, and cell parameterization. SILT compiles directly to CIF and is now used to store our design format for all tools producing geometrical level descriptions.

10. *SLIM, a PLA generation and simulation language* Extensions were made to SLIM, our language for generating and simulating PLA's. These extensions include: allowing arbitrary boolean expressions and if-then-else constructs in the FSM descriptions, performing a large set of compile and simulation time checks on the design, and interfacing to a new version of SPAM (a PLA logic equation minimizer).

11. *YALE - Yet Another Layout Editor* YALE is a layout editor interface to SILT running on the SUN workstation. It was designed and implemented during this contract.

12. *LAVA, An Electrically Based Layout System* A rewritten version of Lava is running test cases, including the 10,000-transistor serial memory. It seems stable enough to support further investigations, *e.g.*, composition of cells and logic-to-sticks conversion. We have developed related mathematical optimization techniques that we believe are efficient enough for 100,000-transistor designs.

13. *DUMBO, A Logic-to-Sticks Conversion* Dumbo produced a totally automatic layout with reasonable area efficiency, using force-directed placement. Large cells still incur large area penalties, however.

14. *PLUNDER, a Control-Description Language* Plunder is a new control-description language that we have investigated as an alternative front end to our control synthesis systems. Since the Plunder language is essentially the control portion of the C programming language, the designer can write in familiar programming-language control structures (rather than requiring FSM state diagrams to describe his control sequence.

15. *Polygon Package and Design Rule Checker.* We have a high-quality design-rule checker based on our polygon package which now has support for buried contacts in nMOS. We have also implemented a design-rule checker based on the JPS bulk CMOS rules and distributed it to MIT and JPL. It was used to check pads, PLAs, and a counter submitted for fabrication; it has also checked

16. *Sticks Compaction.* Supercompaction is a set of techniques to improve the predictablility of 1-D sticks compactors. These techniques analyze a partially compacted cell and selectively move components or introduce jogs to break the critical path, thereby driving the compaction toward minimal pitch for the cell. Results so far indicate that Supercompaction applied to naively-drawn stick diagrams reduces the cell by 5-20% over straightforward 1-D compaction.

17. *Cell Library.* The 4-micron nMOS cell library is now available as a full-color book from Addison-Wesley.

18. *Routing* We analyzed the performance of our original custom-layout router and discovered that the major problems were in global routing dynamics and in the poor match of channel-based routing to the problem. We have explored using quadratic programming to control the placement, and initial results have been promising. We have also devised a new area router based on pseudo-polar coordinates, and we have proposed new metrics for characterizing area-routing problems.

19. *Circuit Extraction.* We integrated the MIT circuit extrctor with our CLL/CIF processing software, resulting in an order-of-magnitude improvement in extraction speed.

20. *Clocking Discipline.* We have developed a 2-phase clocking notation and an associated clocking discipline. The objective is to provide formal concepts for thinking about clocking in 2-phase systems, and to delineate a circuit syntax guaranteeing consistent clocking. As a result of this work, the frequency of occurance of unexplained simulation or testing problems in our classes has decreased dramatically.

21. *The MEDIUM tester chip set.* The MEDIUM tester chip set has been designed, laid out, and submitted for fabrication, along with some test chips. One of the two main chips has been partially tested, and it appears to be correct. layouts from MIT and Lincoln Labs.

22. *ICTEST and Practical Testing.* The ICTEST testing language has been updated to include the clocking notation. The Tektronix S-3260 tester has been integrated into the testing system and used to test a number of designs at speed. The MEDIUM tester chip set can also be driven by ICTEST. The ICTEST system is now in use both in classes and by many research designers.

23. *Defect Tolerance in Array Architectures.* We have developed a new body of theory treating the effect that defects have on yield of array architectures. The theory addresses such isues as whether it is possible to find chains or arrays of working elements embedded in a large array and what reconfiguration capabilities must be available for the yield of the reconfiguration process to be non-zero. The theoretical results allow the prediction of yields of reconfigurable 1-dimensional and 2-dimensional arrays. We have also developed capacity theorems and practical block codes for error correction in memory systems (eg. a code for correcting 2 hard errors and 1 soft error per word).

24. *Wiring Area of Gate Arrays.* By applying statistical modeling techniques, we have developed a body of theory that predicts how to realize a given function in a gate array with smallest overall die size.

Technical Progress

1 Design Description, Analysis, and Synthesis

1.1 SILT: A Language for Relative Layout Specification

SILT is a language that plays somewhat the same role in VLSI design as a relocatable assembler does in software development. It is not as ambitious as a "silicon compiler" would be, but it is much easier to implement and forms a valuable component of a "silicon compilation" system. SILT is more powerful than existing "silicon assemblers" in a number of ways.

SILT is a relative layout language which means that every structure's position is relative to some other structure except for the origin of the entire layout. Changing the distance between a structure and the structure to which it is relative will thus alter the shape of the cell. This makes it easy to design flexible library cells and to delay some design decisions by allowing for some stretch in "finished" parts of the cell design. The "relativity" is hierarchical in nature, which assists in structured design.

Cell descriptions may include parameters so that multiple cells differing only in some power requirement or pull-up ratio can be described by the same code.

SILT is completely descriptive. It does not attempt to put a lot of intelligence into the program, but rather depends on the good sense of a human designer. The descriptive nature of the language allows it to be efficiently translated into CIF or other mask-level descriptions.

SILT is designed primarily to be used in conjunction with a graphical front-end. Circuit descriptions could be written out by hand, but this is anticipated to be the exception rather than the rule. Although writing SILT is probably most easily done by a graphical front-end, SILT text is designed to be easily human readable to aid in debugging. Since SILT will be used as a standard intermediate form for cell designs, its readability and uniformity are important.

Finally, special care is taken to control names properly. Names for features deep down in the hierarchy are hidden from higher levels unless they are important enough to be specifically "exported" up the hierarchy.

The SILT "compiler" was completed, and was used on the MIPS project. The SILT language is translated into CIF by executing the SILT compiler. The main features of SILT are:

- A hierarchical naming structure for naming symbols(cells) and instances of symbols, connection points for symbols, parameters, simple variables, and geometric points within a symbol. This includes an "export" mechanism to prevent "name explosion" at outer levels of a design (names within instances of inner symbols must explicitly be exported to be visible from the outer symbols). Vectored names are allowed.

- Symbols are stretchable. Stretchable geometry must be rectangular, but non-rectangular geometry is allowed. The edges of rectangular geometry are attached to either horizontal or vertical *reference lines*, which may in turn be located relative to other reference lines in a hiearchical structure along the top and side boundaries of the symbol. Attaching one boundary of piece of geometry to one reference line and the other boundary to another reference line makes the geometry stretch by moving the ref. lines. Symbols stretch similarly.

- *Parameters* may control the placement of reference lines. General computational expressions are allowed on the parameter values to determine the placement of the reference lines.

- Local or global *Constraints* may be placed on the relative positions of the reference lines. Constraints are expressed in the form of inequalities.

- *Connections* between cells may be checked by naming geometric points within each cell to be connected and specifying that the named points must geometrically correspond. This allows a simple form of electrical connectivity to be verified.

Documentation for SILT is available as a preliminary edition of a user's guide/technical report. The SILT translator is implemented in PASCAL, and currently runs under both TOPS-20 and UNIX operating systems.

SILT produces enough information for a linking loader to combine a SILT file with CIF files produced by ICARUS (or other layout tools). This makes it easy to glue together primitive cells produced using an interactive layout editor.

Staff: T. Davis, J. Clark, J. Hennessy

Related Efforts: DPL (MIT)

References: [5]

1.2 YALE

Yale (Yet Another Layout Editor) is a graphical layout editor that will run on the SUN workstations. YALE makes the capabilities of SILT available in a graphics front-end. YALE has been implemented on a combination of the SUN workstation and the VAX. It uses the SUN as an intelligent graphics workstation (no disc required). YALE keeps all of its intermediate files in SILT. This work was carried out in collaboration with the Network Graphics project at Stanford. YALE is primarily a graphics interface to SILT, allowing the placement of *reference lines* graphically. It also allows textual or graphical specification of *constraints* and textual specification of expressions for computation of reference line placement.

A subset of SILT was selected for the initial implementation, and a parser for that subset has been written. Hierarchical cell descriptions, rectangles, and reference points are currently working. The cell being edited can be viewed simultaneously through a number of different windows on the screen (the magnification and region of the cell viewed can be independently set in each window). The windows can be stretched or shrunk, moved rigidly on the screen, and the view in each window can be zoomed in or out, or panned. Most of the commands are invoked by means of mouse-button clicks and pop-up menus.

Staff: J. Clark, T. Davis

Related Efforts: Daedalus and the Data Path Generator (MIT), Caesar (UCB).

References: [4]

1.3 SLIM

SLIM, Stanford language for Implementing Microcode, was initially implemented during an earlier contract and presented at the 1981 Caltech VLSI Conference. The goals of SLIM are to describe on-chip control as microcode, to simulate that microcode using a functional description of the chip components, and to generate a PLA implementation of the microcode. The initial SLIM implementation has been working since the end of 1980.

A prototype optimizer, which saves an average of 15% of the minterms, has been developed. It needs further work to characterize its theortical properties and to make it more efficient on large PLA's. A complete normalizer was added that allows arbitrary boolean expressions, default next states, and don't care equations was implemented. A state assignment optimizer was developed.

Staff: J. Hennessy, L. Adams

Related Efforts: MacPitts (Lincoln Labs), SLANG (UCB) and SLAP (Brown University).

References: [14]

1.4 TV - An nMOS Timing Analyzer

TV and IA are timing analysis programs for nMOS VLSI designs. Based on the circuit obtained from existing circuit extractors, TV determines the minimum clock duty and cycle times. It calculates the direction of signal flow through all transistors before the timing analysis is performed, in contrast to combinations of designer-assisted and dynamic determination of signal flow, as in Crystal, being done at Berkeley. The timing analysis is breadth-first (block-oriented) and pattern independent, using only the values *stable*, *rise*, *fall*, as well as information about clock qualification. Its running time is linear in the number of nodes and transistors, and can analyze 4,000 transistors per minute of VAX 11/780 CPU time.

IA (TV's Interactive Advisor) allows the user to quickly experiment with ways to increase circuit performance. With the IA, the user can resize pull-ups and pull-downs or insert super buffers, and find out the effects of these changes on chip-wide performance interactively. By using information already computed by TV, it is able to propagate the effects of changes through 1,000 transistors per second of VAX 11/780 CPU time.

TV was heavily used in the MIPS project. When TV was run on the first version of MIPS, it predicted a cycle time four times longer than our original design goal. By making extensive modifications to the design we were able to reduce the MIPS cycle time to half the original prediction.

Accuracies within 20% for most critical paths compared to circuit simulation and fabricated chips have been achieved.

Staff: N. Jouppi

Related Efforts: Crystal (Berkeley)

References: [20]

1.5 LAVA, An Electrically Based Layout System

Lava is an electrically based, general-purpose layout language. Our principal objectives are topological, rather than geometric, layout description and guaranteed design-rule correctness of layouts. Lava's major components are a sticks compactor, cell stretching and abutment mechanisms, a router, and a framework to link them together.

We have rewritten Lava to stabilize it and to incorporate some of the hooks that will be necessary for further investigations. We have concentrated on a clean implementation of the aspects that we understand well, removing some of the more ill-conceived mechanisms in the previous implementation. One major improvement is that much of the technology-specific information is now centralized; while Lava is not intended to be technology-independent, this technology file makes it possible to change easily parameters of the nMOS target process.

The result is a sufficiently stable platform for further investigations, for example, a well-conceived composition level and logic-to-sticks conversion. The rewritten Lava now successfully compiles a large number of test cases, including the serial memory chip described in a previous report.

Staff: C. Burns, D. Chapiro, P. Eichenberger, R. Mathews, J. Newkirk, D. Perkins, T. Saxe

Related Efforts: EARL (CalTech), CABBAGE (UCB)

References: [23]

1.6 Routing

We have developed a new, 2-dimensional area router, the loop routing scheme (LRS). LRS handles both rectangular- and doughnut- shaped routing areas. LRS is a promising box router for the custom routing problem because, like the dogleg channel router, it indicates how much expansion of the routing area is necessary to complete a given routing.

The difficulty of channel routing problems, and the performance of channel routers, may be measured by the number of wiring tracks required to complete the routing. Previously, no similar measures existed for comparing area routers. Such a measure must describe how difficult a given, fixed, area-routing problem is, since there is no well-defined way to expand the routing area to guarantee completion of the route.

We have developed an appropriate measure of problem difficulty, the Manhattan Area Measure. By using it to assess the difficulty of routing problems generated using Monte Carlo techniques, we have compared the performance of LRS to the classic Lee area-routing algorithm. The LRS has vastly superior performance to the Lee, successfully routing problems that are twice as dense as those that the Lee will complete successfully.

Staff: T. Saxe, L. Smith

Related Efforts: PI project (MIT)

References: [26]

1.7 DUMBO, Logic-to-sticks Conversion

Dumbo is a program aimed at directly laying out random logic from logic diagrams. It targets its output to stick diagrams for compaction by our sticks compactor, Lava. The motivation for a tool of this sort is to ease the layout of miscellaneous logic, especially control logic, in a design. Much logic of this sort is not area-critical, but its design and layout can consume a lot of time using standard techniques.

We have now refined Dumbo to the point where it produces layouts feasible for some miscellaneous logic. For small cells (under 25 components), Dumbo's initial layout will be at most about 2 times larger than one derived from hand-drawn sticks; with hints, this penalty can easily be reduced further. For larger cells, the penalty can become much larger, but is yet easier to reduce using hints.

However, Dumbo still experiences considerable area inefficiency due to the sensitivity of sticks compactors to the vagaries of a particular stick diagrams that it produces. Rather than trying to improve the quality of stick diagrams that Dumbo produces, we have turned our attention to improving the quality of the sticks compactor itself (see below).

Staff: W. Wolf, R. Mathews

References: [27]

1.8 Sticks Compaction

One-dimensional sticks compactors are sensitive to the details of the stick diagrams that they are given to compact. Two topologicallly equivalent stick diagrams can produce very different compacted cells. The essential reason for this behavior is that the compaction algorithms cannot properly exploit the degrees of freedom present in the stick diagram to prevent components from locking against each other during compaction.

The problem that we are investigating is how to compact a stick diagram to achieve a pitch specification in a specified direction. In the majority of a layout, the designer is typically not trying to minimize cell area per se; rather, he is trying to minimize cell area subject to meeting a particular pitch specification in one dimension. Therefore, we are seeking techniques to guide the compactor toward a solution with the minimum pitch in a specified direction, increasing the predicability of the results of compaction by forcing the compactor toward the same solution irrespective of the details of the initial stick diagram. The resulting compaction scheme is called Supercompaction.

To date we have investigated two principal Supercompaction techniques: moving components apart to break constraints between them, and introducing jogs into the stick diagram. Both optimization techniques work by analyzing the critical path in a partially compacted cell and rearranging components or introducing jogs to break the critical path. Naturally, these manipulations cause the cell to grow in the direction perpendicular to the preferred directionas well as reducing the pitch in the preferred direction.

We have investigated a few variants of these techniques by comparing compaction results for cells drawn from a variety of different sources and compacted using the standard Lava compactor and the Supercompactor. Our early results indicate that while supercompaction performs no better (and sometimes worse) than simple compaction for carefully optimized stick diagrams, it easily achieves 5-20% pitch reductions over simple compaction for naively drawn stick diagrams. Over a test suite of 10 cells, 5 were smaller when Supercompacted.

These initial results are in keeping with our goal of developing a predictable compactor. Initial results for jog introduction are even more promising, and we are continuing to investigate supercompaction techniques.

Staff: W. Wolf, R. Mathews, D. Perkins

Related Efforts: CABBAGE (UCB), other 1-D and 2-D compactors

References: [28, 21]

1.9 PLUNDER, A Control Description Language

Plunder is a new control-description language that we have investigated as an alternative front end to our control synthesis systems. The Plunder language is essentially the control portion of the C programming language. Thus, the designer does not need to describe his control sequences as FSM state diagrams; rather, he can write in familiar programming-language control structures. On the other hand, he sacrifices the fine control over the structure of the state machine that a language such as SLIM provides.

Plunder has been used by students in the Stanford design classes. Their experiences suggest that while most of the software control notions carry over to hardware, there are important differences that the ultimate language of this sort must cater to. In particular, the designer often must know precisely what actions are occuring on which clock cycle. Also, description of concurrent activities is an immediately pressing problem for a hardware control language. Nevertheless, Plunder enforces a structuring on control descriptions that generally eases that portion of the IC-design task.

The following is the cononical control-description example, the traffic-light controller:

```
#define green 0
#define yellow 1
#define red 2

input ts, tl, cars;
output restart, hl[2], fl[2];

fsm traffic ()
{
        restart;
        do hl=green; fl=red; while (~tl | ~cars);
        restart;
        do hl=yellow; fl=red; while (~ts);
        restart;
        do hl=red; fl=green; while (~tl & cars );
        restart;
        do hl=red; fl=yellow; while (~ts);
}
```

Staff: D. Perkins

Related Efforts: SLIM(SU), MacPitts(LL)

1.10 Polygon Package and Design-Rule Checker

For some time now, we have made available a high-quality design-rule checker based on our polygon package. It derives circuit connectivity information to prevent reporting of false separation errors between electrically connected components. This checker is used by our design classes and for our research and has been heavily tested by 80+ designers. We have recently added support for buried contacts in nMOS.

We have developed and tested an analogous checker for the JPL bulk CMOS rules. It has checked the designs submitted by Stanford, MIT, and Lincoln Labs for the bulk CMOS run. We have distributed the CMOS checker to JPL and MIT.

Staff: D. Noice

Related Efforts: Lyra (Berkeley), DRC (MIT)

1.11 Circuit Extractor

We have integrated the MIT circuit extractor with our CLL/CIF processing software, resulting in an order-of-magnitude improvement in extraction speed. Previously, the 10,000-transistor serial memory required several hours to extract; extraction now requires approximately 10 minutes. We will distribute the extractor if there is sufficient interest; however, prospective users should be aware that our CIF processing system is restricted to Manhattan-only, rectangle-only designs.

Staff: J. Newkirk, T. Saxe, S. Taylor

Related Efforts: Mextra (Berkeley), Xtract (MIT)

2 VLSI Processor Architecture

2.1 MIPS - A High-Speed Single-Chip VLSI Processor

MIPS (Microprocessor without Interlock between Pipe Stages) is a project to develop a high speed (> 1 MIP) single-chip 32-bit microprocessor. Like the RISC project at Berkeley, MIPS uses a simplified instruction set and is a load-store architecture.

The two major aims in the MIPS design are

- to attempt to shift the complexity from the architecture to the compiler and code generator for the processor, and
- to provide instructions that allow 100% utilization of the components of the micromachine architecture.

The micromachine architecture is a six stage pipeline that holds three active instructions at any one time. The three pipe stages loosely correspond to instruction fetch and decode, operand decode and fetch, and execution. The major resources that are distributed to the pipe stages are register access, PC access and increment, ALU functions, and memory access. Thus, each instruction has a chance to use the instruction memory once, the data memory once, and the ALU twice (once for operand addressing and once for execution). In each pipe stage a potential instruction can utilize all the resources associated with that stage. This leads to a design where the major resources have a duty cycle close to 100% (including the memory).

In a pipelined machine, a large segment of the hardware is associated with pipeline interlocks. Additionally, pipeline interlock hardware is extremely difficult to accommodate in VLSI, since it usually requires a complex interconnection to many elements of the data path. MIPS does not have pipeline interlock hardware; this function must be provided by the compiler. Because the MIPS pipeline is simple, it is not difficult to provide the necessary functionality. The load/store design makes the pipeline interlock checks straightforward and also simplifies the support for branching, interrupts, and page fault handling.

The MIPS instruction set consists of several instruction classes: load/store instructions, ALU instructions (which are all register-register), branches and calls, and miscellaneous instructions. All instructions are 32 bits (one word). There are 16 general purpose, symmetric, 32-bit registers. The instruction classes are orthogonal, although a single instruction word may contain two unconnected instructions. For example, a single instruction word can contain both a load instruction and an ALU instruction (which may use the value loaded in the load instruction). The instructions are summarized below:

- Load/Store:

  - Load/Store absolute
  - Load/Store based - with varying offset sizes
  - Load Immediate
- ALU Instructions:
  - Two and three register forms; instructions combine with Load/Store instructions or with another ALU instruction.
  - Operations include: Add, Subtract, Multiply and Divide Steps, Shift, Complement, And, and OR. Noncommunative instructions have a reverse form.
  - A short unsigned constant may replace one of the source registers.
- Branches and Calls:
  - Branches and Calls can be PC-relative or absolute, and may also be indexed.
  - There are conditional branches that test two registers (or a register and a constant).
  - A delayed branch is used (the two instructions following the branch are executed).
- Interrupts and Page Faults:
  - These are handled as a special class of instructions.
  - Instructions prior to the interrupting or faulting instructions are executed.
  - The status of the processor is correctly restored by the return.

Prior to implementing the full MIPS chip, a series of test chips that would provide a complete test of each major component of the chip individually were developed.

The six test chips contain all the parts needed to implement the complete MIPS processor. Each test chip also contains additional testing and pin multiplexing hardware. By fully testing the components before fabricating a complete design, the probability of success on the first run of the full MIPS design is much higher. This approach also allowed us to characterize the indivdual components and make performance adjustments before the final fabrication. By designing a single reusable test frame, the individual test chips were constructed from the exact pieces of the complete chip with a minimal amount of effort. The final assembly process consisted of merely composing the individual test components to form the complete processor. Lastly, this process offered an ideal opportunity to test the concept of fast-turnaround foundries. Because progress on the project depends on receiving and testing the chips prior to completing the final design, reasonable quality, fast-turnaround fabriction is essential.

The six test chips and their status is as follows:

1. Register File Test Chip - submitted and tested at both 3 and 4 microns. The 4 micron chips were functional, although the yield was <10% (i.e. all parts of the chip worked over 10 die, but no single die was completely functional). The 3μ fabrication produced no working parts.

2. Instruction Decode Test Chip -

3. Barrel Shifter Test Chip -

4. ALU Test Chip -

5. Program Counter Test Chip -

6. Master Pipeline Control Test Chip -

After we received and tested the MIPS' test chips, minor changes were made in the design. We then completed the entire design, spent approximately 3 months in design and performance verification and submitted the first complete version of the processor to MOSIS. The project experience is chronicled in [16].

Staff: F. Baskett, J. Burnett, J. Gill, K. Gopinath, T. Gross, J. Hennessy, N. Jouppi, W. Park, S. Przybylski, C. Rowen, A. Strong.

Related Efforts: RISC (UCB), IBM 801 (IBM Yorktown), Cray-II (Cray Research).

References: [19, 9, 15, 16, 11, 10]

2.2 Geometry Engine

The Geometry Engine is a high-performance, floating-point computing engine for geometric operations in 2D and 3D computer graphics. Multiple copies of the Geometry Engine provide a parallel computing system with very high-performance. (5-10 million floating-point operations per second.)

Enough chips were fabricated to build a geometry system (10 chips) and a complete prototype. This prototype system, called the IRIS, is discussed in the next section. The Geometry Engine design is completely functional, although the performance is less than originally predicted.

The current design is 9mm by 8.7mm. Most of this space is governed by metal pitch minimum spacing and lack of buried contacts. We have obtained metal-spacing design rules from AMI and have learned that there are several metal spacing constraints in the Mead/Conway rules that are unneccsary according to these rules. The most interesting observations are that the minimum pitch is 10 microns, rather than 12, and that the extra 4-lambda flash around all contact cuts is necessary only for the non-metal layers. We have also obtained AMI buried-contact rules, the most useful of which is that metal may cross them.

Taking these two things into consideration, the principal bit-slice of the Geometry Engine can be reduced in area by about 40%. Because this bit-slice consumes approximately 60% of the total area of the chip, the total area reduction should be about 25%. According to projected yields given by AMI, this amounts to a 5-fold increase in yield.

Staff: J. Clark, M. Hannah

References: [3]

2.3 IRIS Workstation

The goal of the IRIS workstation project was to design a high-resolution, color, extra high-performance graphics workstation that utilized all of the features of the Geometry Engine and was software-compatible with the SUN 68000 processor (excluding graphics software compatibility). The system consists of

- A SUN-compatible processor/memory board.
- A Geometry Engine board (10 Geometry Engines).
- A Raster Generation Subsystem.
- A Raster Update Subsystem.

The IRIS allows the user program to generate display programs that provide for real-time motion of 2D and 3D environments, multi-window displays and color lookup table manipulation. To provide for motion simulation, the system is dynamically configurable to provide either double or single-buffer images. The system has been in operation since August, 1982, and procedures are underway to replicate copies of the system for future research at various Stanford Laboratories.

Staff: K. Akeley, J. Clark, M. Grossman, C. Rhodes

3 Testing

3.1 The ICTEST System and the MEDIUM Tester

The ICTEST system is a unified system for functional simulation and testing. The test is written in ICTEST, a superset of C extended to include testing primitives, data formatting, and mechanisms for specifying parallelism and pipelining. The test may then be targeted to run against a simulator (ESIM or TSIM) or a tester (MINIMAL, MEDIUM, or TEK S-3260). The MEDIUM tester is the testing workhorse; the TEK tester is intended primarily for performance measurement and functional testing at speed.

ICTEST itself has remained relatively stable. We continue to use it to test our designs, including the MIPS test chips. Support for the clocking discipline is now substantially debugged, although we need to rethink our approach to qualified clocks and decide how they might be supported on the TEK (if that is indeed possible).

We are reducing the MEDIUM tester to a chip set. It will connect to a standard DEC DMA interface, and we shall distribute it to the community when it becomes available. The chip set that we have designed comprises 2 chip types, and a 64 pin tester will require a total of 3 chips. The tester control and pin electronic chips have both been designed and submitted for fabrication. Additionally, we have designed test chips for some new circuits that we need, including pads capable of driving 30mA loads. We have received the pin electronics chip and the pad test chip and partially tested them. They both seem to work correctly.

Staff: D. Boyle, D. Marple, R. Mathews, J. Newkirk, I. Watson

Related Work: FIFI Project (CalTech)

References: [Mathews82], [Watson82]

3.2 Clocking Discipline

We have developed a 2-phase clocking notation and an associated clocking discipline. The objective is to provide appropriate formal concepts for thinking about clocking in 2-phase systems, and to delineate a circuit syntax guaranteeing consistent clocking. The clocking discipline can also be co-opted to guarantee other forms of correctness, *e.g.*, freedom from charge sharing. The auditing tool *clockck* checks circuits extracted by the ESIM extractor for conformance to the discipline.

Staff: R. Mathews, J. Newkirk, D. Noice

Related Efforts: Glasser's work (MIT)

References: [Noice82]

3.3 Practical Testing

We have tested 30 more copies of a 10,000-transistor serial memory based on a 3-transistor RAM cell. The memory was originally intended as a step toward a serial signal processing system, but has actually proved to be test of our testing system and our understanding of the technology.

Of the 30 new parts from run M1DV, 40% are defect-free. Sixteen percent show anomalously low (less than 100-microsecond) storage times of the sort we reported previously. The remainder have failures that may be explained in terms of fabrication defects, with the exception of 2 chips that have the mysterious property that while every bit in the memory plane seems to be functioning correctly, when we apply error correction to this perfect data, errors result!

As a result of our frustration with short storage times, we have designed and tested a canary circuit that monitors storage times directly. The storage-time oscillator is a 3-stage ring oscillator, one stage of which is a storage node that is charged and allowed to relax toward ground.

We have tested 2 chips so far, one with very short (1 microsecond) storage times. The storage oscillator successfully indicates that the short-storage time chip is defective and that its companion is acceptable. Using optical injection to vary storage times over a large range, we have found that the storage oscillator on each chip predicts the storage time very precisely. The design seems to be insensitive to power supply variations, but we will need to test more chips before we are completely confident.

As a simple experiment in performance measurement, we have designed and tested an instrumented family of 7 PLAs with different loading characteristics. We have measured the performance of each path through each chip and computed regression lines to fit the observed data with delays predicted by τ models. The observed fits are very good, with correlation coefficients around .8 and derived τ's ranging from .25 to .57 nanoseconds. However, the intercepts of the regression line are non-zero, indicating systematic measurement errors specific to each member of the family.

Staff: G. Eckert, R. Mathews, J. Newkirk, T. Saxe, L. Shwetz, I. Watson

Related Efforts: FIFI (Caltech)

References: [Saxe 82]

4 Theoretical Investigations

4.1 Modular Model of Event-based Concurrent Systems

We have constructed a model that can be used in specifying and verifying concurrent systems. The formal model we have been developing has two major components: a structural algebra for describing module interconnection structures, and a behavioral semantics that defines the function computed by a network of modules. We have shown that the model has a number of attractive mathematical properties, which are summarized below. In addition, we have used it to analyze and verify several composite modules, including a memory cell, shift register, and a flip-flop synthesized from gates.

The structural algebra defines a set of operators for synthesizing networks of interconnected modules. The basic building blocks are primitive modules. Each primitive module is associated with a set of named ports for input/output, but it has no internal structure. Compound modules have both internal structure and ports for communication. There are three operators for defining compound modules: port renaming, composition, and the feedback loop. Port renaming leaves structure unchanged but applies new names to ports; it is useful because the other operators make connections based on port names. Composition of two modules matches output ports of the first module with input ports of the same name in the second module. Looping matches output ports of a module with input ports of the same module, wherever the names match.

The algebra defined in this way exactly captures the set of proper nets, which are essentially nets with distinct port names. That is, any expression in the algebra is a proper net, and any proper net corresponds to an expression in the algebra. Also, any expression can has an equivalent normal form as a composition of feedback loops of primitive modules, possibly with renaming of ports. This fact makes it possible to test for equivalence of nets. An axiomatic system for manipulating net expressions is provided.

The structural algebra describes static structure; the dynamic aspects of a system are described by its behavioral semantics. This semantics associates with each module:

- a functional mapping between partially ordered events at input and output ports,
- a domain constraint, specifying that certain output events must precede certain input events, and
- a functional constraint, specifying that certain input events must precede certain output events

The domain constraint is essentially a statement of the conditions under which the module can be expected to work correctly. For example, it might require that no new input events arrive until after all outputs for the current input values have been produced. If the domain constraint is violated, the behavior becomes unpredicatable. The functional constraint, on the other hand, contains information about when a module will produce new output events. Thus the domain constraint tells what the module requires of its environment, and the functional constraint tells what it guarantees.

We have used Scott's least-fixed-point semantics to derive the semantics of compound modules created by the operators of the structural algebra. As a result of this derivation, we can prove that the semantics of a compound net are computable if its components' semantics are computable.

The main problem remaining to be worked out concerns the conditions under which one module can be substituted for another without affecting the properties of a compound system. This is most important when we consider module specifications. In that case we are asking when a module implementation satisfies its specification, and can be used wherever a module of that specification is required. We also expect to expand the range of examples that we have analyzed.

Our work has been particularly concerned with the problems of module substitution and the semantics of non-deterministic systems. The module substitution issue arises because we often wish to substitute one module for another in a network and need to know when this can be done without affecting the properties of the network. A simple criterion for such substitution is *semantic equivalence*. If two modules have the same functional mapping, domain constraint, and functional constraint, then one may replace the other without any change in the network's behavior.

In some cases, however, we need a more flexible criterion. We would like to be able to make a substitution so long as it allows the network to continue working correctly and produce the same output. This may be possible even with modules that are not identical. For example, suppose we have a system containing a module that can perform correctly as long as it is asked to buffer no more than three input elements at a time. (This would be expressed in the domain constraint.) If we replace this module with one that is identical except for the fact that it can buffer more items, then the new network should continue to work correctly. In general, we can always replace a module by one with a *weaker* domain constraint. If the environment of the original module guaranteed that the stronger domain constraint was satisfied, then the weaker one will necessarily be satisfied, and the composed system will continue to perform correctly. Likewise, we can always substitute a module with a *stronger* functional constraint, because if the original module operated in a way that satisfied the domain constraints of other parts of the network, then the (more constrained) new module must do so too. Thus we can perform such a substitution if the new module has an identical functional mapping, weaker (or identical) domain constraints, and stronger (or identical) functional constraints. The new network may not be equivalent to the old, but it will operate correctly in any environment where the old one does. This sort of substitution arises very naturally when the original system is viewed as a *specification* and the substitution represents an *implementation* of the specification.

The second problem we have considered is extending the semantics to non-deterministic systems. Non-determinism is a property of many concurrent systems. It may arise even in networks where all the primitive modules are deterministic; this is because the relative timing of events at different modules is unpredictable, and different timings may cause the system to produce different outputs. A non-deterministic module can be described by altering the functional mapping to give, for each input, a *set* of possible outputs. There are several technical problems that must be resolved in this sort of definition. The most significant is being able to guarantee that loop-feedback (the operation that sends some of a network's output to its own input ports) is always well-defined. By modifying the approach of Plotkin and Smyth, which deals with non-determinism in state-oriented rather than event-oriented models, we have been able to solve these problems and develop a mathematically sound semantics for non-deterministic networks.

Staff: S. Owicki and N. Yamanouchi

References: [22]

4.2 Defect Tolerance in Array Architectures

We have developed a new body of theory treating the effect that defects have on yield of array architectures. The theory addresses such issues as whether it is possible to find chains or arrays of working elements embedded in a large array and what reconfiguration capabilities must be available for the yield of the reconfiguration process to be non-zero.

For the problem of finding a connected chain of working elements in a square array, we have developed a new algorithm that requires time linear in the number of elements to be chained. We have also made progress on the problem of finding an array of working elements embedded within a larger array by tightening the bounds determining when such reconfiguration is possible. We are beginning some new work investigating the effects of defects in the interconnect itself.

Staff: A. El Gamal, J. Greene

Related Efforts: Leiserson & Leighton (MIT)

References: [8]

4.3 Wiring Area for Gate Arrays

By applying statistical modeling techniques, we have developed a body of theory that predicts how to realize a given *function in a gate array with* smallest overall die size. The central result is that it is preferable to have a smaller gate array with a larger number of tracks in between blocks, thereby permitting higher overall use of the array elements, rather than sparsely using a larger array. These theoretical results are borne out by a large body of empirical data collected by IBM.

Staff: A. El Gamal

References: [6]

5 Other Projects

5.1 The SUN Workstation

The SUN workstation is a project begun during a previous contract. During this contract, the prototypes were completed and the ideas were successfully transferred to commercial vendors.

A set of workstation boards was manufactured early December, 1980. (The SUN workstation consists of a Ethernet Interface board, a 68000 processor board, and a graphics system).

The SUN project represents a successful transfer of technology to industry. This transfer was implemented at three levels:

1. By publishing a detailled paper on the SUN workstation.
2. By offering PC boards, wirewrap tapes, and other manufacturing documentation that will allow others to build the hardware in-house at low cost.
3. By licensing industrial companies to manufacture the design, to make assembled board-products and eventually complete workstations commercially available. Again our goal is to get the widest possible base by planning to grant only non-exclusive licenses to as many companies as practical.

SUN workstations are now available from a number of commercial vendors.

There are numerous configurations for the SUN workstation:

- Smart Graphics Terminal to the Ethernet (Processor/Memory, Ethernet Interface, and Display controller, plus keyboard and mouse)
- Standalone Workstation with Disc (Add a disc controller board and disc)
- Ethertip - terminal concentrator to allow normal terminals access to Ethernet. (Graphics Board not needed)
- Gateway between Ethernets - Two Ethernet boards, one Processor/Memory board.

5.1.1 The SUN Workstation: File System Development

The SUN represents a radical departure from the customary workstation design in that it does not have a local disk. A typical SUN workstation at Stanford has 256K bytes of memory, a frame buffer with some additional memory to hold the raster image, and a high-performance Ethernet link. In order to use the SUN as a workstation for VLSI design or other applications, it must be possible to read and write file storage from software resident in the SUN.

Our initial approach, which permits us to run simple software on the SUN, was to implement a page-at-a-time file server called a Leaf Server, which ran on a supporting computer (usually a Vax) and provided disk page access in response to request packets. This server and its development were reported in a previous report, and we have done little work to it since. It is worth mentioning that the Unix-based Leaf Server that we wrote was made available to other Arpa-supported users of Xerox 1100 Lisp Workstations, and (after suitable modifications at ISI to make it compatible with the Xerox equipment) it provides a valuable disk support facility to AI programmers using the 1100's.

Our experience with the Leaf Server approach to Remote File Access demonstrated conclusively that a single-host file server was not an adequate level of file support for a network machine participating in a distributed system. We experimented for a few months with changes that could be made to the Unix file system or to the behavior of the Unix Leaf Servers that would make a more reasonable distributed file system available to SUN users. We abandoned this approach for three reasons:

- The Unix file system does not map neatly to a distributed environment. At the design level, it assumes that there is at most one copy of any file, and that the entire file system is tree-structured. It is difficult to modify the Unix kernel to think that parts of its file systems are on other machines, though at the 1981 SIGOPS conference some Bell Labs researchers reported having accomplished it (with a severe degradation in performance.) At the implementation level, its locking mechanisms are unreliable and the fixed I/O table sizes in the kernel provide unreasonable fixed bounds on the total number of files that can be accessed simultaneously on a given server host. We thought that we could get by with a combination of a few kludged Unix file systems for the first generation of SUN software, but the problems overwhelmed us.

- Leaf-Server access to files on a time-shared Unix system was a second class citizen. We frequently found the need to log a job on to the host Unix, probe around the file system, then log off and resume a stopped SUN job that was having file problems. This problem could be bandaged by providing a set of file utility programs resident on the SUN, but the essence of the problem is that a Unix file system is not very suitable for a non-Unix-like operating system; we do not intend to run Unix on our SUNs.

- Even with just 4 server hosts available (3 Unix and one Tenex), the amount of context that needed to be maintained in a user's head was overwhelming. The lack of automatic location, migration, or replication facilities made it particularly difficult to find files whose precise location was not known.

We therefore, reluctantly, concluded that we were going to have to design and implement our own file system to our own specifications. This file system would be network-wide, provide a uniform set of access mechanisms and management tools, and be implemented on a variety of file computers.

We have settled on a multi-level design based around a central archival file system and distributed cached copies. We are implementing the design "from the inside out", starting with the reliable archival part and working towards the fast cache servers; the reasoning behind that decision being that it is better to have a slow reliable file system than a fast unreliable file system during the development phase. The initial implementation is taking place on our time-shared VAX, but we hope to move to a dedicated machine with a larger disk as soon as it becomes available.

Staff: J. Mogul, B. Reid

References: [1]

5.2 Computer support for a Fast Turnaround Laboratory

We completed the planning and exploratory stages of a project to provide extensive automation and computer support for the Fast Turnaround Laboratory. This ambitious interdisciplinary project (involving researchers from Computer Systems, Integrated Circuits, and Solid State laboratories) will provide control, documentation, training, portability, repeatability, and efficiency in the area of IC fabrication processes.

As a result of this exploration, we have isolated the following goals for this project:

- Automatic control of the IC fabrication processing equipment.
- Integration of fabrication control with simulation control, to run simulation and fabrication in parallel.
- A transportable, repeatable means for recording a fabrication process.
- The ability to repeat an arbitrary process on demand, for demand production of parts.
- The ability to manage and schedule a single IC fabrication line for multiple purposes.
- Computer aids for training and documentation.
- General data processing and database support for analytic work in the Laboratories.

We recognize that particular technical achievements will contribute to the realization of several of these goals. The most important of these is the development of a language for the representation of fabrication processes. When this language exists, it can be used as input to the control system, as input to the simulation system, as the contents of an archive for demand production of parts, and as a basis for training and documentation aids. Furthermore, the nature of this language will color most of the other work.

We have therefore spent several months on the preliminary design of such a language, which we now call FABLE. This language resembles Ada semantically, but contains built-in type support for non-numeric types pertaining to manufacturing, and contains extension and package mechanisms suitable for the description of typed objects whose physical existence is outside the controlling computer, e.g. furnaces. We quickly found that two languages were necessary, one to describe the manufacturing process and another to describe the fabrication line itself. This second language corresponds very much to the microcode found on conventional computers, and it is used to implement a somewhat abstract instruction set, into which the FABLE is compiled. A compiled FABLE program can operate an automated fabrication line with the appropriate microcode, or it can operate a simulation system (one component of which would be programs like SUPREM) with a different set of microcode. Other microcode would be written to permit experimentation and testing of new processes before actually turning them loose on the fabrication equipment.

Only by implementing all pieces of this system and actually using it to manufacture parts can we satisfy ourselves that it is complete; we would therefore want to do an automating implementation even if that were not one of the goals of the project.

Having completed the first level of implementation, we intend to write software that amounts to a distributed time-shared operating system for the fabrication line; this will permit multiple independent fabrication processes, or laboratory experimentations, to be run on the same fabrication line simultaneously just as a time-shared computer is now capable of running independent programs simultaneously. This operating system will also take responsibility for the long-term scheduling and priority realization.

We consider that a system like this will be a superb testbed for explorations in knowledge-based systems for training and diagnosis, and for applications of interactive graphics, computer aided instruction, and reliable models of computation.

Staff: B. Reid, W. Ossher

Publications: [7]

5.3 Cell Library

The Cell Library is finally available as a book from Addison-Wesley and in machine readable form from Addison-Wesley (or via ARPAnet for DARPA VLSI research groups). The book is in full color, and contains considerable additions beyond the July '81 version of the Cell Library. Many thanks to the people who contributed cells, who helped with testing them, who participated in the massive job of documenting them, and who encouraged us along the way.

Staff: R. Mathews, J. Newkirk, C. Burns, and everyone else on the project.

References: [25]

5.4 Design Classes

The Stanford Mead/Conway design classes have continued to evolve. Starting in the fall of '82, they became a 3-quarter sequence. The first quarter is the introductory class, but with a paper-only design project. This format allows TAs and graders to handle most of the routine work, important since 125 students took the class in Fall Quarter '82, 60 took it in Spring Quarter '83, and 125 are taking it this Fall Quarter. The second and third quarters are the design and testing laboratories, with a more manageable enrollment of about 50 students.

In the '82 sequence, because the students now had an entire quarter to do a design and because they had the prior experience of a first paper design, we allowed the projects to grow as large as their designers desired. That was a mistake, but the results were dramatic \- half of the projects contained over 10,000 transistors. We provided no significant new tools except a simple channel router and ICDEBUG. The end-quarter rush of design-rule checking and simulation overwhelmed our VAX, so checking was no more thorough than in previous years, and the testing results were very similar.

A small number of teams undertook bulk CMOS projects. We provided plotting, design-rule checking, RSIM, and a small library of I/O pads and pieces of a precharged PLA. Since the Stanford CMOS process is n-well and the MOSIS CMOS process is p-well, all designs were described in psuedo-twin-well design rules. (These rules are fully symetric, with explicit well and shorting layers of both types.) The resulting designs were then mapped to each target process and submitted for fabrication.

The last four years of design classes at Stanford are summarized in [24]. This technical report begins with a short paper describing the class from the instructor's perspective, but it is mostly a picture book of abstracts and plots displaying almost all of the class designs carried through since the second design class in the spring of 1980.

Staff: J. Newkirk, R. Mathews, T. Saxe, S. Taylor

References: [24]

List of Publications and References

1. Bechtolsheim, A., Baskett, F., and Pratt, V. The SUN Workstation Architecture. Technical Report 229, Computer Systems Laboratory, Stanford University, March, 1982.

2. Baskett, F., Clark, J.H., Hennessy, J.L., Owicki, S.S. and Reid, B.K. Research in VLSI Systems Design and Architecture. Tech. Rept. 201, Computer Systems Laboratory, Stanford University, April, 1981.

3. Clark, J.H. "A VLSI Geometry Processor for Graphics." *Computer Graphics 13*, 7 (July 1980), 59-68.

4. Davis, T. and Clark, J. YALE User's Guide: A SILT-Based VLSI Layout Editor. Technical Report 223, Computer Systems Laboratory, Stanford University, October, 1982.

5. Davis, T. and Clark, J. SILT: Stanford Intermediate Language for Topology. Technical Report 226, Computer Systems Laboratory, Stanford University, November, 1981.

6. El Gamal, A. Notes on Gate Array Wiring Area Prediction. ISL VLSI File #102282

7. H. L. Ossher and B. K. Reid. FABLE: A Programming-language Solution to IC Process Automation Problems. Tech. Rept. 248, Computer Systems Laboratory, Stanford University, September, 1983.

8. Greene, J. and El Gamal, A. Area and Time Penalties for Restructurable VLSI Arrays. submitted to the Third Caltech Conference on VLSI, March 1983

9. Gross, T.R. and Hennessy, J.L. Optmizing Delayed Branches. Proceedings of Micro-15, IEEE, October, 1982.

10. Gross, T. Code Optimization Techniques for Pipelined Architectures. Proc. Compcon Spring 83, San Francisco, March, 1983, pp. 278 - 285.

11. Hennessy, J.L. and Gross, T.R. "Postpass Optimization of Code in the Presence of Pipeline Constraints." *ACM Trans. Prog. Lang. and Sys.* (1983). accepted for publication

12. Hennessy, J.L. and Gross, T.R. Code Generation and Reorganization in the Presence of Pipeline Constraints. Proc. Ninth POPL Conference, ACM, January, 1982.

13. Hennessy,J.L.. "SLIM: A Simulation and Implementation Language for VLSI Microcode." *Lambda Second Quarter* (1981).

14. Hennessy, J.L. A Language for Microcode Description and Simulation in VLSI. Proc. of the Second Caltech Conference on VLSI, Caltech, January, 1981.

15. Hennessy, J.L., Jouppi, N., Baskett, F., and Gill,J. MIPS: A VLSI Processor Architecture. Proc. CMU Conference on VLSI Systems and Computations, October, 1981.

16. Hennessy, J., Jouppi, N., Przybylski, S., Rowen, C. and Gross, T. Design of a High Performance VLSI Processor. Proc. of the Third Caltech Conf. on VLSI, Calif Institute of Technology, Pasadena, Ca., March, 1983.

17. Hennessy, J.L., Jouppi, N., Baskett, F., Gross, T.R., and Gill, J. Hardware/Software Tradeoffs for Increased Performance. Sym. on Architectural Support for Programming Languages and Operating Systems, ACM, March, 1982.

18. Hennessy, J.L., Jouppi, N., Gill, J., Baskett, F., Strong, A., Gross, T.R., Rowen, C., Leonard, J. The MIPS Machine. Proc. Compcon, IEEE, San Francisco, February, 1982, pp. 2 - 7.

19. Hennessy, J.,Jouppi, N., Przybylski, S., Rowen, C., Gross, T., Baskett, F., and Gill, J. MIPS: A Microprocessor Architecture. Proceedings of Micro-15, IEEE, October, 1982, pp. 17-22.

20. Jouppi, N. TV: An nMOS Timing Analyzer. Proceedings 3rd CalTech Conference on VLSI, California Institute of Technology, March, 1983.

21. Mathews, R, Newkirk, J, Eichenberger, P. A Target Language for Silicon Compilers. , Proceedings of COMPCON, Spring, 1982, pp. 349-353.

22. Malachi, Y. and Owicki, S.S. Temporal Specifications of Self-Timed Systems. CMU Conference on VLSI Systems and Computations, Rockville, Md., October, 1981, pp. 203-212.

23. Mathews, R., Newkirk, J. and Eichenberger, P. A Target Language for Silicon Compilers. Proc. Spring Compcon, IEEE, February, 1982, pp. 349-354.

24. R. Mathews, J. Newkirk. Introduction to VLSI Systems at Stanford. Information Systems Lab, EE Dept., Stanford University, September, 1983.

25. J. Newkirk, R. Mathews. *The VLSI Designer's Cell Library.* Addison-Wesley, Reading MA, 1983.

26. Smith, L., Saxe, T., Newkirk, J., Mathews, R. A New Area Router, the LRS Algorithm. Proc. ICCC, New York, September, 1982, pp. 256-259.

27. W. Wolf, J. Newkirk, R. Mathews, R. Dutton. Dumbo, a Schematic-to-Layout Compiler. Proc. 3rd CalTech Conf. on VLSI, CalTech, March, 1983.

28. W. Wolf, R. Mathews, J. Newkirk, R. Dutton. Two-Dimensional Compaction Strategies. Proc. International Conf. on Computer-aided Design, IEEE, September, 1983.