COMPUTER SYSTEMS LABORATORY

STANFORD UNIVERSITY · STANFORD, CA 94305-2192

Research in VLSI Systems Design and Architecture

Forest Baskett
James Clark
John Hennessy
Susan Owicki
Brian Reid

Technical Report No. 201

March 1981

This work was partially supported by the Defense Advanced Research Projects Agency, contract # MDA903-79-C-0680, and by the Joint Services Electronics Program under contract # DAAG29-79-C-0047.

Research in VLSI Systems

Design and Architecture

Forest Baskett, James Clark,
John Hennessy, Susan Owicki, Brian Reid

Technical Report No. 201

March 1981

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

Abstract

The Computer Systems Laboratory has been involved in a VLSI research program for one and a half years. The major areas under investigation have included: analysis and synthesis design aids, applications of VLSI to computer graphics, !he design of a personal workstation, special purpose chip design, VLSI computer architectures, and hardware specification and verification. Progress on these research problems is discussed, and a research program for the next two years is proposed.

Key Words and Phrases: VLSI, design automation, computer-aided design, graphics, special purpose chips, VLSI computer architecture, hardware verification, personal design stations.

Table of Contents

List of Figures

1 Introduction

As fabrication technology provides more usable silicon each year, two complementary problems are presented: how do we construct robust systems that efficiently utilize the space and what design **tools** enable us to construct these systems more efficiently? This portion of the proposal addresses these two issues.

Robust integrated systems are characterized by several features. First, they should employ design practices that scale with as little change as possible. The best hope for integrating large systems is that they be constructed from subsystems that are known to be correct. Simulation is necessary in constructing the subsystems, but the ultimate test is that they have been fabricated and demonstrated **to** work. Systems constructed from working subsystems that employ scalable design practices can be expected to work. Second, they should use a consistent timing strategy. For example, large integrated systems cannot be reliable if they cannot avoid synchronization failures **[SeitzB].** Thus, any timing strategy used in a VLSI system must be consistent, scalable and not subject to failures resulting from design. These are key features of the architectures proposed here.

Designing and implementing large systems on silicon requires software design tools that simplify the design process. These tools should decrease the amount of time and expertise needed to complete a design. Although substantial time is saved by employing a good design methodology, **such** as regular geometric and temporal structures as advocated by [Mead] and **[SeitzA],** better tools **will** also decrease design time. Good. tools should also provide additional documentation of the design. Just as with large software development systems, these VLSI design systems should make the design self-documenting when possible. Finally, simulation and verification are a necessary component of all design tools; they increase the probability that the design is correct. The design aids **proposed** here are motivated by these considerations.

To provide a high quality interface to the designer, a method of graphically interacting with the designs is needed. The aim is to allow the designer to interact with a single user interface, which provides graphics capabilities and allows access to all design tools. To accomplish this a graphics design station that easily couples to other computing resources is proposed.

2 Summary of Accomplishments During the Previous Year

- The SUN workstation is a modular personal computer system designed for use in an Ethernet-type local network. A SUN workstation provides a single user with significant **local** computing power, a high resolution graphical display, graphical input, and network communication [Bechtolsheim]. The SUN station capabilities can be realized at **a** cost of approximately **$10,000** per station, using commercially available VLSI components. Its configuration **and** capabilities make **it highly** suitable for use as a **personal VLSI** design station. At the present we have constructed a prototype SUN **workstation and interfaced** it **to** UNIX as a virtual terminal with bit-mapped image graphics capability.

- Geometry Engine component of SUN terminal design completed and being prepared for fabrication.

- **Smart** Image Memory Processor for high-performance image memory in SUN terminal designed and being fabricated.

- SLIM (Stanford Language for Implementing Microcode) is a language and processing system for specifying, simulating, and implementing microcode in VLSI. SLIM microcode **is** written as a finite state machine that is simulated using an environment description to specify the functional units controlled by the machine. A SLIM program can be compiled into a PLA implementation and automatically laid out. The SLIM microprogram simulation **and** PLA generation system is operational and has been **used in designing** and debugging the control section of the Geometry Engine and the Image Memory Processor.

3 VLSI Design Aid Systems

There are two important classes of design aids:

- *Analysis aids* analyze an existing design or design component and attempt to reveal potential errors They may also provide additional design documentation.

- *Synthesis aids* assist the designer in the development of a workable, well-documented design. They are usually more ambitious than analysis aids both in concept and implementation.

This proposal addresses the investigation and implementation of both classes of aids. Our philosophy in approaching these problems is influenced by several considerations. First, our experience has shown that design aids inspired by **a** design in progress are most useful and responsive to the needs of the design community. This approach also ensures that the tools will be used in a real design environment. Thus, we shall continue to employ the practice of design-driven aids. Second, the design aids must be user-oriented and compatible. This **means** that the design aids **are** all usable in a consistent and uniform manner, store similar input/output in a compatible format, **and,** are accessible with a single user interface. Third, the design aids are hierarchically oriented. Various aids may be used at various stages of the design process to address problems that arise at different levels. Fourth, simulation is crucial at almost every level of the synthesis and analysis process and must be supported by the-design tools.

3.1 Analysis Aids

Analysis aids fill a major need in a design system: they enable the designer to check his design for potential errors and violations of his design specifications. We propose to investigate three types of analysis aids:

- fast circuit extraction and switch level simulation from a layout-level description with the goal of extracting information useful for a timing-based simulation,
- logic-level extraction and logic equivalence with the goal of ensuring the consistency of a layout and a logical design,
- hierarchical timing analysis of a VLSI system.

Circuit extraction from a layout is a particularly useful tool. It provides both static and dynamic checks. The static checks report the existence of components of the layout that are unconnected or not logically structured. The circuit extractor's output can be used to drive a switch-level simulation. Moreover, if the extractor provides the right geometric information, a simple RC timing model of the circuit can provide very useful event-driven timing information for the system as laid out. Our experience with measurements made on working chips (the Geometry Engine piece fabricated with MPC79 [Conway]) has shown that a simple RC timing model, which properly takes into account the measured bottom-wall and side-wall capacitance and resistance parameters of a process, gives excellent estimates of the temporal characteristics of a circuit. In the designs that we have thus far completed, this timing information was found the most lacking.

Logic extraction from a layout provides a basis for comparing a top-down, logic-level design with the actual layout. This process of checking the consistency of two levels that should be equivalent is particularly important until we are able to employ a totally synthesized design approach. The logic-level equivalence of two designs can be established by a program.

Timing verification is especially important in systems design. In many digital systems local timing of some portion of the design critical, or the final overall timing of the system is important. Hence, a method of specifying timing constraints is needed. Because of the complexity of digital systems, timing constraints are most useful if they can be automatically checked. Mechanical checking of timing constraints requires a method to specify the timing of components within the design. A timing verifier provides for the specification of timing data and timing constraints on components and supports the mechanical checking of the constraints. For such a process to be computationally feasible and logically consistent, it must be done in a strict hierarchical fashion; this requires that the design have hierarchical structuring properties.

3.1 .1 Current Status

For circuit extraction and simulation, we currently employ programs written by Baker and Terman at MIT [Baker, Terman]. These programs are useful in their present state, but they fail to provide information that could be used to estimate the temporal behavior of a circuit layout. The circuit extractor also runs very slowly, since it operates on an expanded CIF file.

We are currently designing a program to perform the task of logic extraction from a CIF description or switch level description (using the circuit extractor as a first step). We are investigating an approach to the logic equivalence problem based on a notion of labelling common nodes in the two logic diagrams.

The S-1 design system was constructed to support the development of the S-1 processor (supported by the Office of Naval Research) [McWilliams]. The S-l design system consists of

- SCALD (Structured Computer Aided Logic Design) System — a hierarchically structured logic design system.
- SUDS (Stanford University Drawing System) — a graphics system, used to edit, input, and output to/from SCALD.
- A timing verifier — based on SCALD [McWilliamsB] and designed to analyze and verify timing specifications and constraints in a digital system.
- The physical design system — consists of both placement tools (chip placement) and wirewrap technology tools.

The timing verifier is oriented towards ECL rather than nMOS IC technolcgy. It has been used extensively in the design of the S-l and was helpful in eliminating large numbers of timing errors and providing confirmation of important timing characteristics.

3.1.2 Proposed Research

We propose to construct a circuit extraction program that provides information from which the temporal characteristics of a circuit can be estimated. The program will extract area and perimeter of nodes, from which the capacitance of the nodes can be calculated using the process parameters, and the length of polygonal conductors from which resistance calculations can be made.

The difficult part of this work is accurately estimating the resistance of **a** conductor, given its geometry and the process parameters. In general, this is a very difficult problem requiring **finite-**element analysis to accurately compute point-to-point resistances for arbitrary geometries. For exact resistance measurements, this calculation is necessary. However, to be able to *estimate* the worst. case timing paths in a complete layout, such calculations are too costly, and for the most part would be unnecessary. Thus, this work will focus on devising a simple -mechanism for computing the **resistance,** given some simplified resistance models for the most common conductor geometries and **the process** parameters. Then a simple RC estimation scheme will give reasonable *comparative* worst-case timing paths for a *system* as laid out, thereby providing an extremely valuable system **timing tool.**

We propose to investigate the design and construction of a logic-level extraction system. The **extractor** will use the output of the circuit-level extractor. A logic equivalence program to determine if two **labelled** logic graphs are equivalent will also be designed and implemented. While such a **program** falls in the category of (possibly) short- lived analysis aids, it will be very useful in verifying that a design as laid out corresponds to the logic-level design. All such aids will hopefully become **less** useful as better synthesis aids come into existence.

The timing verifier **[McWilliams]** will be implemented in a VLSI System design context. This will **involve** integrating the timing-verification ideas in that work with the circuit extraction and timing estimation plans discussed above.

3.2 Synthesis Aids

Many of the concepts in a synthesis system obey a compiler-like paradigm: the design is usually **expressed** in a language at various high levels and translated to a low-level design. In the case of **VLSI,** the high-level design is a functional specification in some programming language, and the **low-level** design is a set of rectangles describing the layout. Our research objective is to simplify the **design** problem by automating it where possible and providing, through simulation and verification, **consistency** checks between the various levels of the design process.

Our design system, called Pegasus, is based upon the concepts in the SCALD hierarchical design system **[McWilliams],** suitably augmented for VLSI implementation to provide timing simulation at all **levels of** the design process. Using PEGASUS, the compilation process of a design involves design at different functional levels and level-to-level translation. The designer specifies the design at a variety of levels increasing in detail. The bottom-most level consists of logical cells chosen from a standard cell library; these cells accomplish the most basic functions (inverter, simple gates, pass gate, etc.). The representation of these cells might be either CIF layouts or some topology description such as STICKS. The higher levels define functional components in terms of lower levels using concepts such as replication and parameterization. Figure 1 illustrates the major facets of the system.

At the highest level the design is functionally oriented. It consists of a high-level specification of the functional units that compose the design (e.g. alu's, registers, etc.) and the control specification. The control specification is automatically compiled to create a PLA or ROM implementation. This control function can be thought of as microcode. Using that view, the system consists of functional units and an implementation of the microcode control program. Viewing the microcode control as a program (e.g. written in SLIM) leads to two advantages: the control function can be thought of as a programming activity with the potential of eased development, and it becomes trivial to alter the control function. These are goals that are normally associated with a high level language. The functional units are implemented using a hierarchical logic design system.

A fundamental problem with this approach is consistency between levels in the design hierarchy. For example, a functional unit *alu* may be specified at one level and elaborated in successive lower levels. Although the design system provides assistance in defining the functional expansion, the actual correctness of the functional implementation of the *ah* must be ensured by the designer. To resolve this problem requires an automated method of performing the checks for level-to-level consistency. Because of the number of levels involved (functional specification to high-level logic design to low-level logic design to cells) and the potential complexity of each level, this check is extremely important. There are two methods of performing such checks: verification and multi-level simulation. The application of software specification and verification to VLSI hardware is discussed in Section 1.3. Multi-level simulation is useful, if a method for specifying the relationship of different levels in the hierarchy can be found. Because timing analysis (see analysis section) is a hierarchical tool dependent on timing specifications at the different levels of the hierarchy, it is important to establish a method for specifying this information during the design synthesis.

Since the design is hierarchically oriented and different versions of objects with the same logical function can exist at different levels of the hierarchy, a major version problem exists. There are two classes of version problems. The first class of problems results when functional units at different levels of the hierarchy are replaced (usually by better implementations of the same function). Since these versions may not be exactly compatible from the designer's viewpoint, he may desire that the design system preserve the consistency of his design. This is similar to the software versioning problem that plagues any design under active development. Another class of the same problem arises

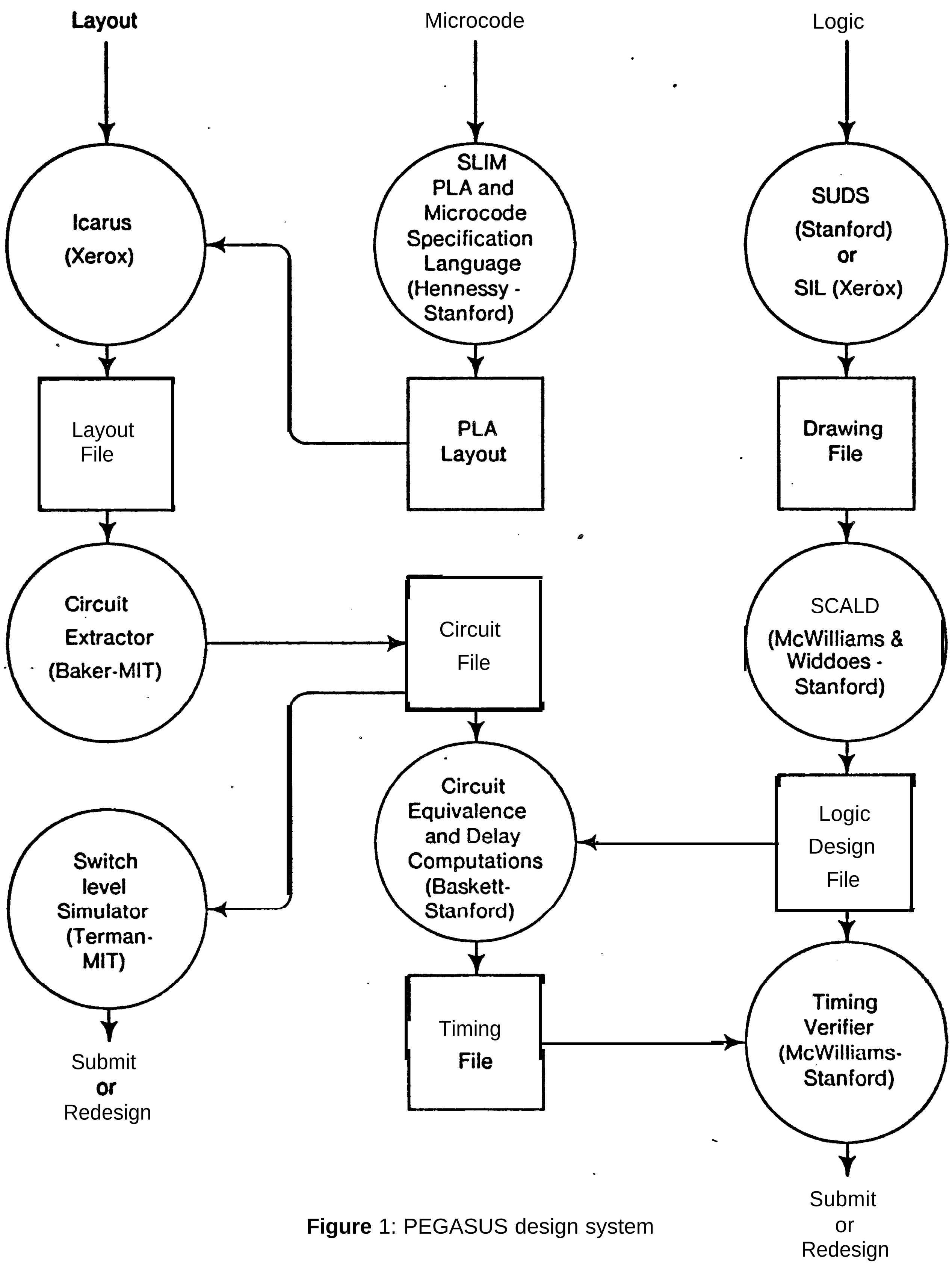
Layout
Microcode
Logic
Icarus (Xerox)
SLIM PLA and Microcode Specification Language (Hennessy - Stanford)
SUDS (Stanford) or SIL (Xerox)
Layout File
PLA Layout
Drawing File
Circuit Extractor (Baker-MIT)
Circuit File
SCALD (McWilliams & Widdoes - Stanford)
Circuit Equivalence and Delay Computations (Baskett-Stanford)
Logic Design File
Switch level Simulator (Terman-MIT)
Timing File
Timing Verifier (McWilliams-Stanford)
Submit or Redesign
Submit or Redesign

Figure 1: PEGASUS design system

when the designer has multiple choices for a functional unit. We would like to choose among multiple implementations in an "intelligent" manner based on general indications of the designer's goal. For example, the design system should be able to make intelligent choices among implementations with different power, size, and speed tradeoffs. We should be able to acquire such information **during** the design specification. .

3.2.1 Current Status

Research <in synthesis design aids, suitable for VLSI environments, at Stanford has concentrated **on** two major projects: the ***SCALD*** (Structured Computer Aided Logic Design) System and ***SLIM*** (Simulation Language for Implementing Microcode) [Hennessy]. We propose to construct a system, Cascade, for the design of functional units in VLSI. Cascade would be based on the Scald design system, but would be oriented towards the needs of VLSI and would integrate with SLIM to form a hierarchical synthesis design system.

3.2.2 Structural Description

SCALD is a hierarchical system for designing and compiling logic designs. It has been used to specify and implement the design of the S-l **[McWilliamsB],** a high speed 5500 chip **ECL-10k processor.** The design experience using SCALD has been favorable; the entire S-1 processor was designed and implemented with approximately two man-years of effort. SCALD is a graphically-based **design** system. The input and various stages in the hierarchical expansion of the design are displayed **to** the user in a graphical format. The actual input and output to SCALD is a well-defined language and a standard graphics system is used to display or enter this information graphically.

A design in SCALD (i.e. a SCALD input program) consists of a set of macro definitions that **represent** the hierarchical structure of the design. There is also a distinguished top-level macro that **represents** the **entire** system. The bottom level of the macro expansion is a set of primitives that are implemented directly. In the S-l these primitives are **ECL-10k** chips. The SCALD system is a retargettable compiler in the sense that any of the levels can be replaced with a different logically equivalent structure. The design can then be recompiled into an implementation using the new **primitives.**

For the macro expansion approach to succeed a suitable method for handling replication of structures (e.g. register cells) andsignals is needed. This makes it substantially easier to specify the levels in the design and the translation process. SCALD supports these concepts with the following **features:**

1. Language constructs for declaring and using signal vectors.
2. Constructs for specifying local, global, and parameter signals.
3. A method for replicating a single macro both logically and graphically.
4. A mechanism for conveniently dealing with multiple physical versions of the same logical signal.
5. A method for allowing the use of both physical polarities of a signal.

3.2.3 Control Description

The control function on a chip is viewed as software and is susceptible to a more software-oriented approach. Programming the microcode is still an error-prone task for several reasons. The programming language is extremely low level: the designer must deal with a binary machine without the benefit of a human-engineered interface. Microprograms are often large and relatively complex, without a great deal of structure. Another difficulty is the significant level of detail that must be expressed. This leads to one of two pitfalls: either the microcode description is very low level and cluttered with details, which makes it impossible to understand; or the designer uses an ad hoc higher level description of the microcode. Lastly, for microcode machines that will be implemented with PLA's, the process of laying out the PLA is both time consuming and tedious, leading to unnecessary errors. However, given a description of the PLA terms, in an equation or state table form, the process is mechanically straightforward and numerous programs exist to do this.

However, the difficult part of the task of microcoding a design is not in PLA layout, but in the description and design of a correct microprogram. SLIM (Simulation Language for Implementing Microcode) is a language and processing system for defining microcode to be implemented using PLA's. It addresses the problem of designing correct microcoded machines. This is particularly suitable in the VLSI environment as the high-level microcode specification can be debugged in software and finally directly compiled into hardware. The microcode control for a chip is written in SLIM as a high-level-language program.

A SLIM program consists of the control specification and a functional environment description. The control specification is a program written in a form that can be easily translated into a finite-state machine. The environment is a description of the functional units with which the controller interacts. It is described in a standard programming language and used to simulate the microprogram. When a SLIM program is processed, the designer may specify either PLA generation or the generation of a functionally oriented simulation program. The simulation program utilizes the environment description to describe the functional aspects of the chip.

The use of SLIM in actual chip design has proved to be very valuable. It has been heavily used in both the Geometry Engine and Image Memory Processor projects and by several other smaller projects. Without SLIM the larger projects would not have been feasible. The use of SLIM has also made incremental development and design alteration possible, as well as contributing substantially to the correctness of the final design. The main features of SLIM that make it a useful and powerful tool are:

- the simulation capability,
- the ability to abstract from the low-level control by providing the mapping between functional concepts and physical signals once, using a powerful notation,
- the ability to specify details about the PLA output signals and their occurrence that are implemented by SLIM.

3.2.4 Proposed Research

The major goal of our proposed research in synthesis aids is a consistent, compatible, hierarchical design system that proceeds from the functional description level as characterized by a SLIM program to the “Sticks” level. A SLIM program provides both control definition and a functional description of the other chip components. These components are designed using Cascade. Our goal is to have a consistent design system using both SLIM and Cascade. To make a consistent system we must explore new methods for ensuring consistency between design levels, e.g. the functional description of the design given by SLIM should define the behavior of the design in Cascade.

SCALD was designed as a general-purpose logic design system. Since Cascade will be a an SC design system, it will require a set of structure libraries that are finally implemented in VLSI technology. A structure library is a functional level in a SCALD type design. It defines a number of components and their implementation in terms of components in a lower level structure library. We will use existing designs to build our initial structure libraries. The bottom-most level of the structure hierarchy will consist of primitive cells. An investigation of this technique and the use of cell libraries for Cascade is needed.

Although this approach seems reasonably straightforward, there are a number of major problems to be solved. The most critical problem is the basic difference between SCALD logic design and Cascade IC-design. In typical logic design the set of components available for implementing a function is fairly small. However, in IC-design issues like power and speed determine the type of component needed. We intend to solve the problem in two steps. Initially, some method for indicating parameters or constraints about a particular functional block will be used in Cascade. Eventually we plan to make such choices automatically based on knowledge database for the design.

. Such an IC design system must have substantially more knowledge about the nature. of the components under design and involved in the design, which will necessitate the investigation of more sophisticated knowledge representation forms than simple macros for representing the design and its components. The primary difficulty with macros as a knowledge representation technique is that the object-management software cannot easily determine the properties of objects by simple examination of the definition. A macro, stored as an unparsed character string, is used by expanding it into the necessary code and then evaluating that code. As the complexity of the objects defined as macros increases, and as the set of limitations on their use and interaction with other objects increases, a knowledge representation must be devised that permits inspection of the objects' properties and analysis of their interaction with other objects. We expect additional benefits to accrue from the improved knowledge representation. Purely from the standpoint of software engineering, macros are not a particularly good way to build large libraries of complex objects because they are so difficult to error-check and difficult to edit. We expect that the improved knowledge representation techniques will, as a byproduct, give us a way of editing, storing, and analyzing structure definitions that is substantially superior to the existing method.

Although SCALD has been traditionally used with SUDS (Stanford University Drawing System), the input to SCALD is a well-defined language and any suitable graphics front-end could be used for SCALD or Cascade. An alternative graphics front-end like SIL, the Xerox logic design system, could also be used. Cascade will also support a programming language input. The output of a Cascade IC-design would consist of graphically represented hierarchical designs of function units, IC designs in a topological form e.g. STICKS. The output of Cascade would be fed to a layout program to produce a design containing all the geometry information, probably in CIF.

Although SLIM has proved to be an invaluable tool in the design of large VLSI systems, it can be improved in several areas. The ability to specify PLA outputs and inputs in SLIM is currently very powerful; this ability is needed to limit the amount of low-level design effort required. However, because of this ability the designer can introduce easily overlooked errors. As a first improvement to SLIM, we will develop better methods of ensuring that the descriptions of PLA outputs and inputs match the functional specification. Some of the possible methods for doing this are: to allow the user to specify constraints that will detect certain errors in the use of PLA signals, to automatically check for certain potential problems such as the incorrect use of pipelining, and to use the functional output for SLIM to drive simulations of the components that are controlled by the PLA. The overall goal of both of these improvements is increase the consistency of the functional simulation and the implementation of the control function in the PLA.

The major problem with any hierarchical design system is ensuring that the functions defined at a given level are correctly implemented using the lower levels of the hierarchy. In a standard compiler, **the levels of** translation (parse tree, intermediate form, etc.) are specified by the compiler designers and translated by the compiler code. In the hierarchical synthesis system that we propose **to** investigate, the levels are defined by the various components at each level, and each component is defined in terms of lower levels. The level translation is done by the compiler but the actual levels (i.e. the components) and inter-level translations (i.e. the implementation of the components) are defined **by** the designer. This makes the level inconsistency problem very serious.

We propose to use simulation as a tool to checkout a hierarchically structured Cascade design. Although a simulation approach can not **prove the** correctness or consistency of a design, it can substantially increase the probability that the design is correct. A hierarchical design is defined in **terms of a series of levels, L,,,...,L,,** from the highest to the lowest level. Given a simulation definition **for** the objects on level L,,...,L_{i-1}, we can derive a simulation model for level L_i. Thus we can simulate the entire design in terms of models for level L_1. Unfortunately simulating the top-level of the design in **terms** of the most basic components leads to unacceptable inefficiencies.

. **One** solution to this problem is to employ multi-level simulation. Each level of the design hierarchy can **be** simulated using a model that simulates only the necessary level of detail. This leads to the requirement to show that multiple levels of a simulation are consistent and correct, i.e. a high-level simulation model of level L_i should be functionally equivalent to a derived model that is obtained by construction from levels $L_1,...,L_{i-1}$. In general this is **a** program verification problem, but it is reasonable to attempt to insure the consistency of the high-level and derived models by simulation.

We say that a high-level simulation model for level L_i is consistent if levels $L_2,...,L_{i-1}$ are consistent **and** there exists a mapping between the high-level simulation and derived simulation for level L_i, such **that they** are functionally equivalent. Since the base level, L,, is provided once and has no derived simulation it must be correct. Once a high-level simulation has been shown to be consistent it can be used instead of the derived simulation for that level. This allows efficient simulation of large designs **that are** hierarchically structured. Once interlevel consistency is shown for the entire design we can conclude that an entire design is correct, in simulation terms, by examining only the high-level **simulation. How is the** derived simulation constructed, what the starting properties for the lowest level are, and how the consistency check occurs are the major issues that require investigation. We will explore only functional simulation for this initial period; the timing verifier will be used for timing **analysis. We expect the improved** knowledge representation in Cascade to be of substantial help in **this work.**

3.3 Specification and Verification of Concurrent Hardware Systems

As improvements in VLSI technology lead to more and more compact circuits, the complexity of the systems that are implemented in a single chip is growing exponentially. Mastering this complexity is essential, so that the time and expense involved in the design phase do not grow at the same rate. Software systems of comparable complexity have existed for some time, and many of the techniques of "software engineering," such as modularization, can be profitably applied to VLSI design.

We propose to investigate the application of methods developed for describing and reasoning about software modules to hardware problems. Availability of similar techniques in the VLSI environment should assist the design process in several ways. Precise specifications can be used for communication betweendesigners, and can provide the basis for formal checks of the correctness of the design. Moreover, work on program specification and verification has led to a better understanding of programming languages, and hence to their improvement. We expect a similar payoff in improved construction methods and tools for VLSI systems.

Our aim is to develop a useful set of abstractions for reasoning about VLSI systems. To this end there are three questions that must be addressed. First, what properties do we need to be able to specify and verify? It is clearly important to be able to describe functional behavior (the relationship between input and output values), and various aspects of timing behavior. But exactly. what do we **need** to say in these areas, and what other properties are important? Second, how can we formally express the properties we identify as important? We will consider various specification styles, and assess their suitability. Finally, how can we reason about the systems in order to convince ourselves that an implementation meets its specifications? Program verification is based on a set of rules for deriving logical properties of programs from their code, and we will seek to develop similar rules for VLSI designs.

A major difficulty with VLSI systems is that they typically involve a very high degree of concurrent activity, and timing errors are a serious problem. Thus techniques for coping with the complexity of concurrency in software should be particularly relevant to reasoning about VLSI systems. An important principle in concurrent programming is speed-independent design: the system is designed to work correctly regardless of the relative speeds of its component processes. This makes the system easier to understand (since timing details can be ignored) and more robust (since changes which effect the speeds of parts of the system do not interfere with other parts). There has been considerable work on reasoning about this sort of concurrent software. Self-timed logic [SeitzA] is based on the same principle of speed-independent design, so many of the techniques for concurrent software apply to self-timed systems as well. For this reason, we plan to concentrate initially on self-timed systems. Our work is complementary to the work of Seitz and Mead; we are attempting to develop the-logical foundations for reasoning about the sorts of designs they are developing.

Modularization is another technique that has proved valuable in developing and reasoning about software, and we plan to make use of modular specification and verification techniques in analyzing VLSI systems. A modular approach requires composition rules for combining small modules into larger ones, as well as abstraction mechanisms that suppress irrelevant details of a complex module. Some principles for composition and abstraction have been identified, such as Seitz's rules for self-timed modules [SeitzA], but much remains to be done. This class of modularization and verification techniques should also be extremely useful in ensuring that functional consistency is maintained between levels.

3.3.1 Current Status

We first review the status of specification and verification techniques for software systems, and then discuss our initial experiences in applying software techniques to hardware.

Sequential programs are typically characterized by their functional behavior, and there are well known techniques for verifying such input-output relations for sequential programs. Functional behavior is also important in concurrent programs, but other properties are important as well. Many concurrent systems, such as operating systems, are intended to operate for an indefinite time, providing a continuously available service. These systems can be specified by a combination of safety properties, which assert that the system never enters an undesirable state, and *liveness* properties, which assert that some desirable state is eventually reached. Thus, one might specify a message system by saying that all messages delivered correspond to messages that were sent (safety) and that each message sent will eventually be delivered (liveness).

Methods for verifying safety properties have been known for some time; for example see [Keller, Lamport, OwickiA & B]. Suitable methods for proving liveness have been developed more recently [OwickiD]. They are based on the use of temporal logic, an extension of ordinary logic to include operators for reasoning about the future of a program computation. The basic temporal operators are □ (pronounced "henceforth") and 0 (pronounced "eventually"). The formula "□P" is true of a computation if P is true of every state in the computation, while "OP" is true if P is true of one or more states in the computation. Safety properties can be expressed using Cl, and liveness properties using ◇. In addition, more complex formulas using the temporal operators are valuable for specifying other properties, such as the behavior of synchronizing primitives. Temporal logic provides the first method for rigorously proving liveness properties of concurrent programs.

Other relevant work concerns modular specification and verification techniques for concurrent programs [Francez, Good, HailpernA & B, Howard, OwickiC]. Safety properties are typically defined by invariants. Each module in the system is specified by an invariant, and there are composition rules for deriving the system invariant from global invariants. Liveness is considered in [HailpernA, HailpernB], where it is defined by temporal logic formulas called *commitments.* Commitments can be defined at the module level, and module commitments can be combined to derive systems commitments.

We plan to build upon this work in our analysis of VLSI systems. Some simple self-timed modules have already been studied under this grant [Bochman], We have been able to write precise specifications of the timing behavior of a number of self-timed circuit components, including combinational circuits, memory elements, and the Muller c-element. In addition, we have verified the implementation of an arbiter [SeitzB] based on logical reasoning from the specifications of its components. Specification of these circuits requires an additional temporal operator, a generalization of 0: "P □ Q" is true of a computation if Q is true as long as P is. This form of relation is essential for self-timed logic specifications, although it has not been important in reasoning about programs. For example, self-timed combination& modules have the property that no output becomes defined before at least one input is defined; this can be expressed as:

all inputs undefined □ all outputs undefined.

3.3.2 Proposed Research

Our initial goal is the development of a specification method that allows precise statement of the behavioral properties of a module, and rules for verifying the correctness of module implementations. **At a** later stage, the understanding gained in this process will be applied to identify construction rules that exploit the advantages to be gained from modular design.

Research into specification and verification methods will proceed along two lines The first is the analysis of modules that are more complex than those already studied, although still small enough to be tractable. The purpose will be to identify problem areas that can not be adequately handled by existing specification and verification techniques. During the coming year, we plan to investigate examples on the order of complexity of a self-timed bus and a simple processor architecture. The result of this study will be a partial specification of the relevant behavior, and an identification of those problems that make complete specification impossible.

The second line of attack will be expansion of the specification and verification techniques at our **disposal.** One requirement is an extension of temporal logic reasoning to cover the sort of timing relationships that frequently are required between hardware modules, but are not common in software. For example, concurrent software processes often interact by the exchange of discrete **messages.** Self-timed hardware modules, on the other hand, interact through signals that must be maintained long enough to be detected by the other party. Specification of this sort of property requires the generalized $\square$ operator discussed above, and we need considerably more experience in using that operator in specifications and verification.

in addition, we find that the most common way of describing hardware systems is in terms of events (changes of values), while software systems are more frequently analyzed using reasoning based on states. There is obviously a close relationship between state-based and event-based models, since **either** one can be defined in terms of the other. We plan to investigate the suitability of state-based reasoning for self-timed systems, and attempt to develop an automatic mapping between the two kinds of specifications.

4 Systems Architecture

The purpose of design tools is to help the designer manage the complexity associated with designing very large integrated systems on a single piece of silicon. Just as important as the need for good tools is the need to structure these systems so that they make the most efficient use of the design medium. We propose to investigate and design highly parallel special-purpose and general-purpose systems architectures that are suitable for large-scale integration on silicon.

These systems architecture projects provide an excellent driving force for design aid development. As a result of our design-driven philosophy, we have generated two novel LSI designs, the Geometry Engine [Clark] and the Image Memory Processor [Clark&Hannah], and have developed a very useful design tool in SLIM [Hennessy]. We propose to continue this practice, as outlined in the following sections.

Initially, we have focused on building **a** good design workstation, the SUN terminal. Consequently, the principal architecture projects were to enhance the graphics capability of the workstation. We **plan** to continue this work, with a few extensions as described in Section **4.1.** We also propose ways that these projects are suitable vehicles for investigating very large system integration issues. In addition, we propose to design more general-purpose architectures, as outlined in Section 4.2.

4.1 Special-purpose Architectures

Quality, low-cost graphics is important to VLSI design. No design environment is complete without a **fast** mechanism for displaying integrated circuit diagrams and layouts. High-performance 3D capability can be added to the basic 2D capability because it can be had without much added complexity or cost **via** the Geometry Engine.

The graphics architecture projects made up of the Geometry Engine and the Image Memory Processor push the limits of VLSI. The Geometry Engine, for example, has about 60,000 transistor complexity, and since the design scales trivially, the complete Geometry Subsystem would produce a system with almost 3/4 million transistors on one chip with no substantive change to the design. Likewise, depending upon the configuration, the Smart Image Memory array could require a vast number of transistors. Redundant schema for placing the entire subsystems on single pieces of silicon should be investigated.

We present the status of the graphics and workstation projects and propose some extensions to the architectures that will yield significant high-performance, low-cost imaging systems that are useful **in** contexts other than IC design, such as topographic mapping and flight simulation. The system described in the following three sections can easily handle performance demands of 2D and 3D design problems such as IC layout and circuit drawing, real-time display of 3D geometric objects and real-time, arbitrary-font character graphics on a bit-mapped display. It is based on the general-**purpose** SUN workstation, the Geometry Engine, and the Image Memory Processor. Figure 2 is an illustration of the system.

4.1 .1 SUN Workstation

SUN is an acronym for Stanford University Network, an ethernet-based local network being implemented throughout the Stanford University Campus. This effort (Gorin, 1980) is designed to connect systems that span the spectrum for computing needs from large timesharing systems to personal computing systems. The current ethernet connects several major computers within the Computer Science building, the Center for Integrated Systems annex and the Electronic Research Laboratory, with plans to connect all major resources in the next few years.

The SUN workstation is a modular personal computer system designed for this network providing a single user with significant local computing **power, a** high resolution graphical display, graphical input, and network communication [Bechtolsheim]. The SUN station capabilities can be realized at a cost of approximately $10,000 per station, using commercially available VLSI components.

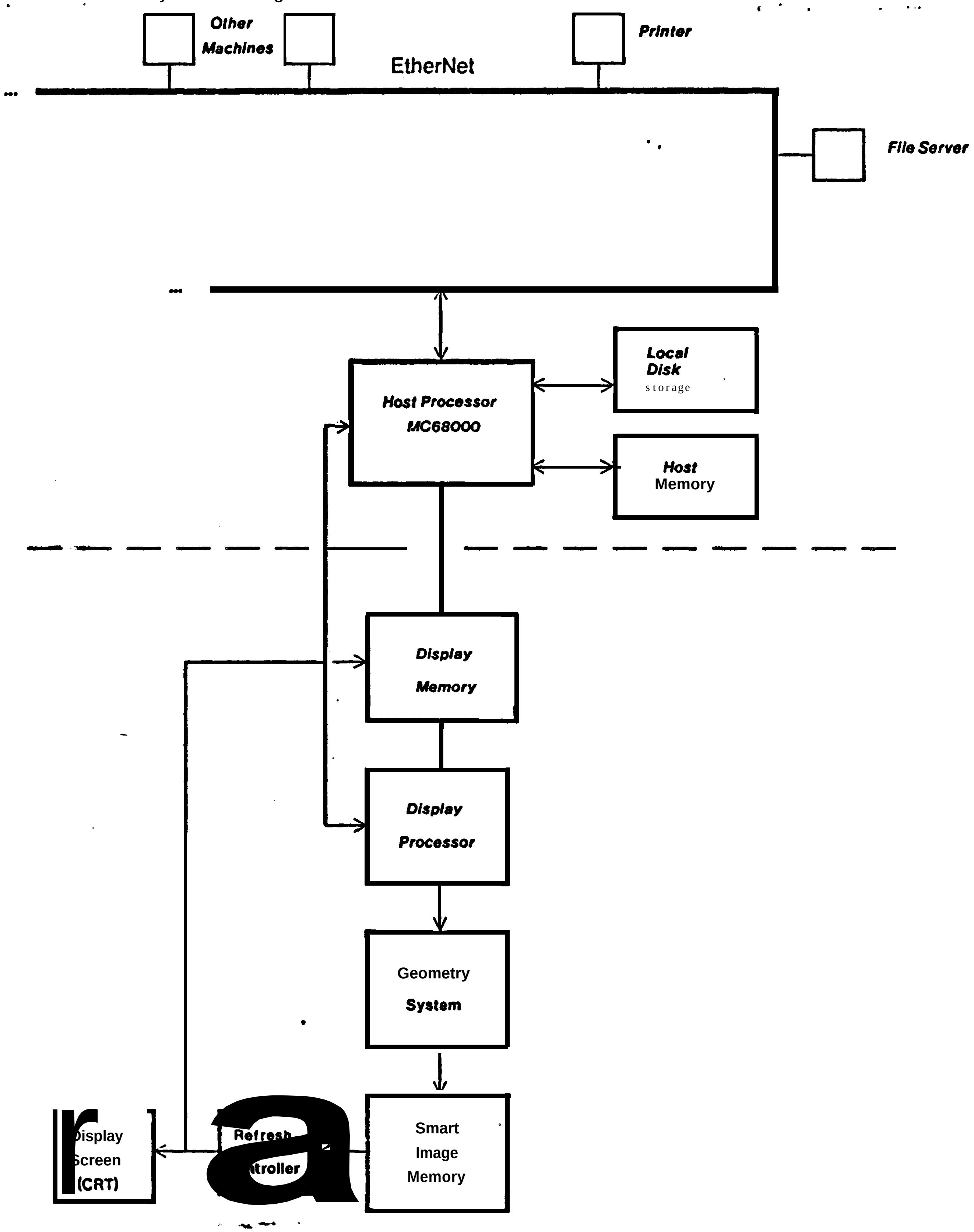

Figure 2: The SUN Workstation

The highly interactive environment provided by these workstations can be used both to facilitate actual circuit designs, and to develop more powerful design tools. The ability to simulate multiple contexts simultaneously using standard communication protocols allows' an easy transition to a distributed processing system.

4.1.2 Current Status

We currently have one prototype frame buffer and processor module in operation. Production versions of the frame buffer and processor modules are currently in the PC board layout stage. A fairly general-purpose ethernet interface module has also been designed. Automated design procedures are being used for all modules, aiding in design, implementation, and debugging.

We have developed network software for the VAX/Unix operating system to allow virtual terminal access, high-quality document printing and file transfer via the ethernet. A terminal emulation program has been written in the C language, which is cross-compiled on a VAX for the MC68000. This program allows the dynamic creation of multiple windows, each with an entirely separate operating system context. It was written as an initial use of the SUN prototype to debug the software and provide a good user interface to the VAX. Larger, more general programs will be developed when the **new** processor/memory-management board are available.

4.1.3 Proposed Research

We plan to complete the design of production frame buffers, processor modules, and ethernet interfaces, and build 10 personal workstations over the next year. The modular nature of the system allows many customizations, like the use of color or black and white, or interlaced or non-interlaced monitors.

The use of portable languages like C means that much useful software, such as the implementat on of Internet protocols being done at MIT, can be used. We plan to use Pascal on the MC68000, so many of the design tools currently written can be transported. Initially graphics operations will be done in software, with minimal hardware assists for time-consuming shifting and masking operations. This initial system will eventually be augmented by the powerful image processing hardware described in the next sections.

4.1.4 Geometry System

The geometry system is composed of 12 copies of the Geometry Engine arranged in a parallel processing pipeline. The Geometry Engine is a four-component vector function unit that can perform the most common repetitive geometric computations of graphics. It is microprogrammed to do bloating-point matrix transformations of 4-component vectors, clipping of lines, polygons and characters to the user's floating-point viewing-space window, and scaling to the integer-space viewport of the destination screen. The structure of the system is described in [Clark], with a few significant enhancements. Figure 2 is an illustration of the system in its new form.

4.1.5 Current Status

The microcode for performing the standard graphics functions has been completed and simulated using the SLIM microprogram simulator that was developed in parallel with the design of the Geometry Engine [Hennessy]. The chip is in the final stages of layout; the PLA control structures are being wired to the function units and the whole circuit will soon be extracted and simulated. This microprogramming phase has taken about 75% of the total design time.

A structural change has been made to the system that makes it scale much more nicely. In contrast to the original organization, the matrix multiplier chips and the scaler chips are now strictly in a pipeline rather than in the "parallel" organization given in [Clark]. Figure 3 illustrates the new organization. This required a small change to the microcode, did not change the performance of the system, and, coupled with another change, eliminated the need for pins to tell each chip its type. In other words, the system is parameterized so that after initialization, the first commands to the chips tell them their types. After each chip receives its type, it enters its normal input-wait state. Because chip-type is now "soft," rather than "hard," wired and because of the simpler pipelined organization, any number of the 12 Geometry Engines can be distributed on any number of die for packaging.

Another modification to the Geometry Engine enables it to trivially generate 3D stereo images. This was done by recognizing that a stereo transformation involves modulation of the horizontal displacement of points with the depth of the point being displaced. In the original organization of the scaler chips, one of them was half idle; that is, two of its function units were not used in the scaling phase. With the stereo modification, these two units now do the same type of operation as do the other two, one computing for the left eye and the other for the right eye of the viewer; in 3D stereo mode, two x coordinates are output by the Geometry Subsystem. Figure 4 illustrates the stereo modifications.

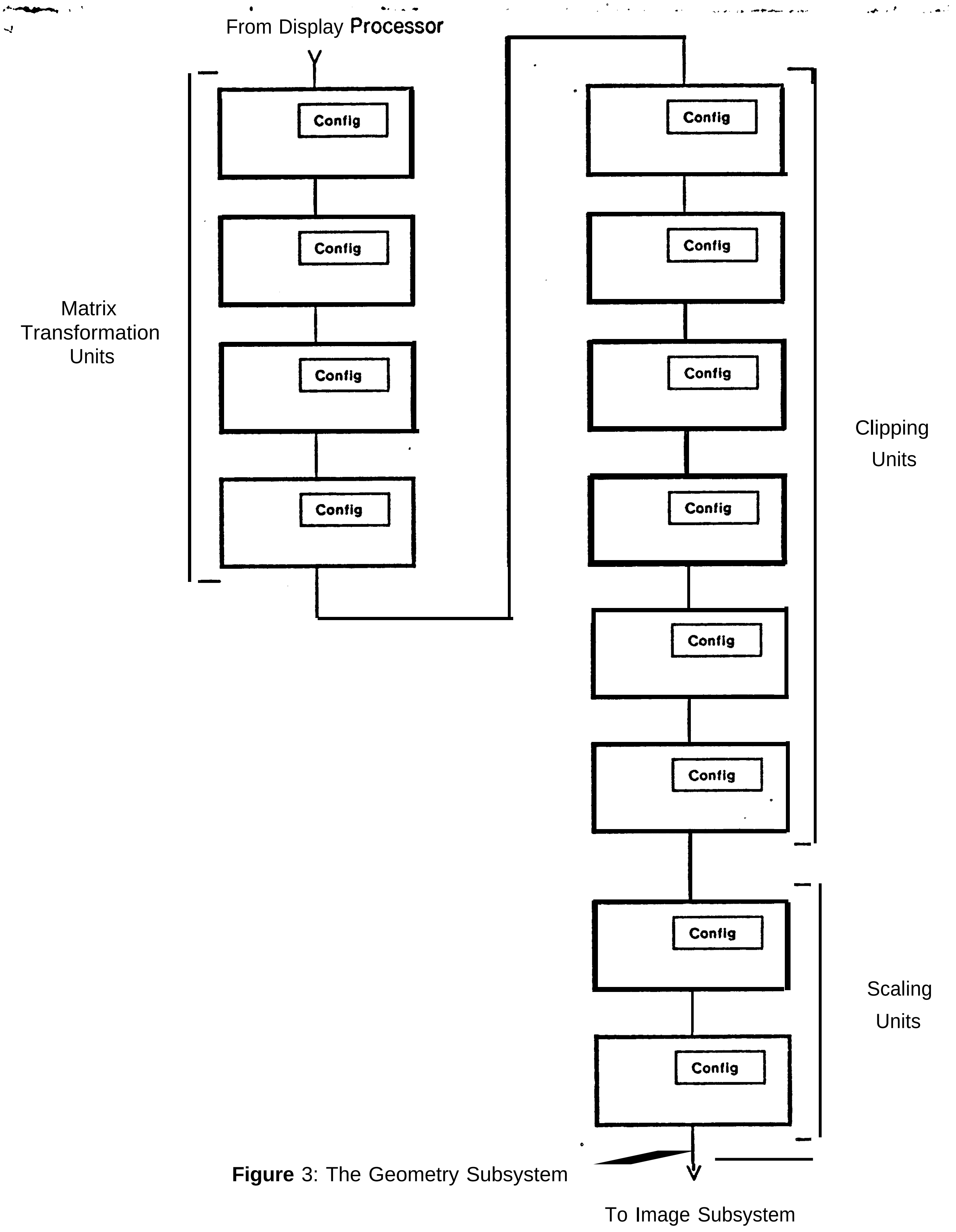

Figure 3: The Geometry Subsystem

... can be shown that the stereo transformation can be expressed as

$$x' = (x/w)*Vsx + Vcx + A(z/w)$$
$$x'' = (x/w)*Vsx + Vcx - A(z/w)$$
$$y' = (y/w)*Vsy + vcy$$
and $$z' = (z/w)*Vsz + vcz,$$

where x' is the x coordinate for the right eye of the viewer. x'' is that for the left eye, and A is a constant proportional to the separation between the eyes of the observer. y' and z' are the y and z coordinates for each eye. The only difference between the two eyes is in their x coordinates, and this difference is a constant times the perspective z coordinates.

Each scaling unit can compute two things of the form $(q/r)*Vs + Vc$. Therefore, arranging two units in pipeline as shown below will yield the stereo transformation.

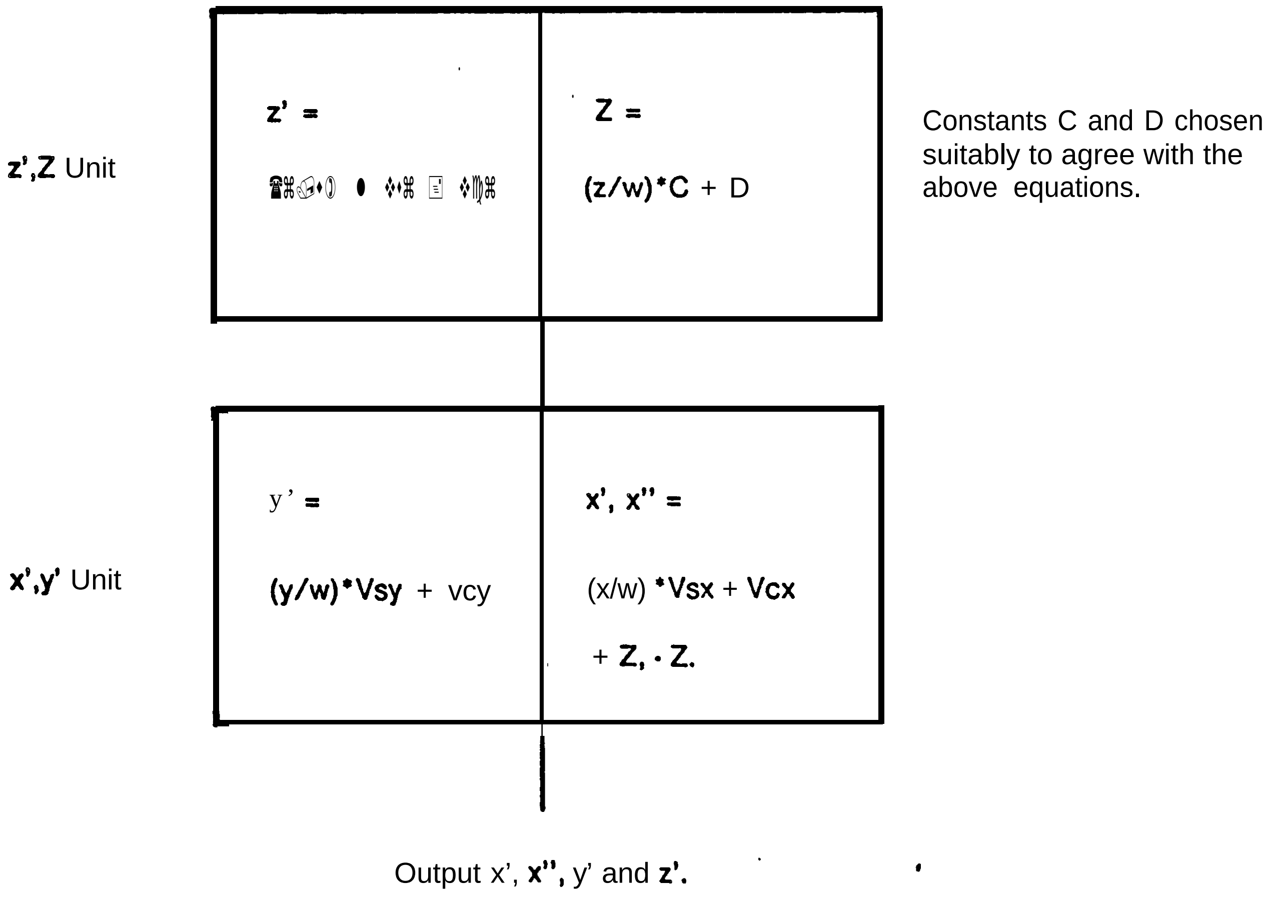

Figure 4: Stereo Transformations of Scaler Chips

4.1.6 Proposed Research

The current purely pipelined organization of the Geometry System provides a useful architecture for investigating the integration of large pipelined systems on a single piece of silicon. We propose to investigate redundancy techniques for pipelined architectures. This activity will involve integrating the entire system, probably on a wafer scale, such that pathways exist for including/excluding a given pipelined function unit on the basis of tests that are run on the units after fabrication. We expect these tests to be "soft" or adaptive in that they might be changed dynamically, as opposed to "hard" or determined by some hard-wired means. We shall direct the work in such a way that the schema are applicable to any pipelined architecture.

The proposed mechanism for initial implementation is to incorporate starting frames for each die that provide interconnections between copies of the Pipelined Element (PE), as shown in Figure 5. A single-wire, shift-register selection mechanism is associated with each copy of the PE to allow it to be conditionally selected or bypassed. The selection scheme is unary encoded to allow any combination of PE's to be included/excluded. Clearly, for this scheme to work, the shift-register selection mechanism must be free of fabrication-related defects, but it does not in itself require much area, so there is a good possibility that the mechanism will yield fruitful results.

4.1.7 Smart Image Memory

The performance limitations of conventional image memories are in the time required to do the arithmetic of scan-conversion, i.e., determining which pixels to alter, and in the memory bandwidth. Normal organizations, such as the lower-cost version of the SUN terminal described above, have one I/O port to the host processor and one output port to the CRT refresh controller. The host is often a general-purpose processor with the image memory part of its normal memory address space. Thus, scan-conversion is done by the host. The process is limited by the computation rate of the processor in combination with the bandwidth of the memory.

4.1.8 Current Status

The Smart Image Memory System described briefly below and in [Clark&Hannah] scan-converts polygons, lines and characters using a custom nMOS Image Memory Processor (IMP). Multiple copies of the IMP in combination with either 16k or 64k memory chips will yield a low-cost Smart Image Memory System that should provide the following improvements over existing systems:

- Vector-drawing rates comparable to those of random-deflection CRTs (40 nsec/pixel).
- Previously unattainable, arbitrary-font character drawing rates (90,000 characters/second).

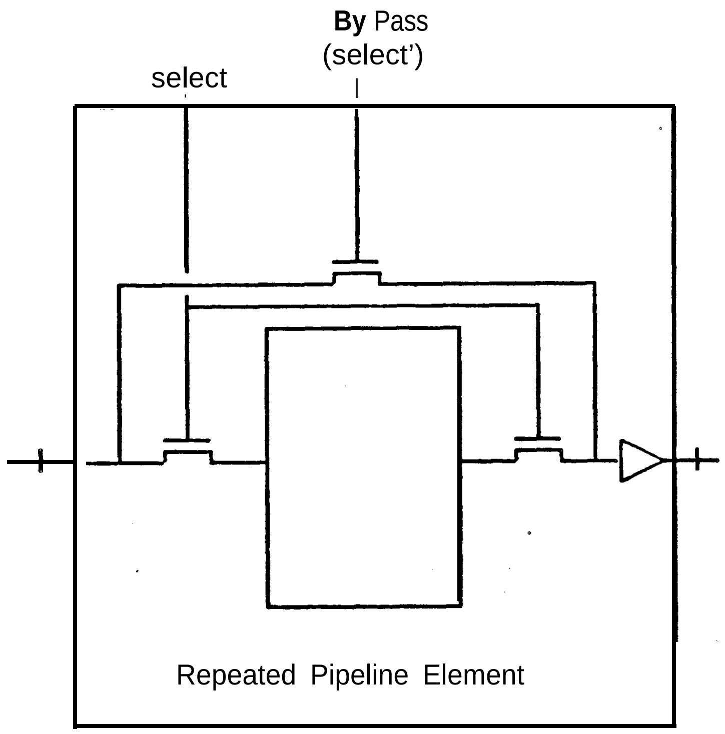

Figure 5: Pipelined Element Selection Mechanism

- Polygon drawing rates comparable to those of very expensive, dedicated **polygon** systems (10 **nsec/pixel).**

4.1.9 System Organization and Function

The Image Memory System for a 1024x1024 organization using **16k** RAM chips is shown in Figure 6. **The** particular size of the memory chips and resolution shown are independent of the memory system organization; the choice is to illustrate the organization. The two-level, hierarchical, interleaved-processor busing structure maps onto the screen in such a way that, for any contiguous 8x8 set of pixels on the screen, each pixel is controlled by one processor and each processor **controls** one pixel. All data passes through the Parent Processor on its way to the memory. This **processor** is not a custom chip but is a microprogrammed processor that can send information asynchronously to the 8 Column **IMPs** (C-IMPs). Its primary purpose is to prepare the geometric primitives for scan conversion. Character codes are looked up in the font memory by the Parent Processor., Each column is sent to the appropriate C-IMP, which in turn sends the code to its R-IMPs, **who** select the appropriate bits to alter in their memory chips. Lines and polygons are scan-converted using linear difference equations computed by the Parent Processor. Horizontal scan-conversion is done by the C-IMPs; vertical scan-conversion and actual memory accesses are done by the R-IMPs.

The IMP has been designed, thoroughly simulated, laid out and submitted for fabrication to the Stanford IC Laboratory.

4.1.10 Proposed Research

The low cost of **nMOS** memory chips makes the memory system described here economical, but the optimum way of building the memory is to integrate the IMP with the memory on the same chip. **This** approach has a number of advantages. It reduces the pin count of the resulting package; it allows the memory to be partitioned differently, thereby giving still higher bandwidth; and it reduces the total number of packages. Moreover, integrating the two leads to thoughts of larger-scale integration on a wafer.

The IMP currently requires a **64-pin** package. Although pin count is high partly to assist in testing **the** prototype, and some of the pins can be eliminated after this phase, the current lo-bits/pixel limitation on the IMP arises from the need to keep this count down.

Ten bits/pixel is satisfactory for many applications, but there are interesting applications for which **it is not** enough. The realistic representation of full-color, fully-shaded objects is one application requiring a minimum of about 6 bits per color component, or 18 bits/pixel. Creating these realistic

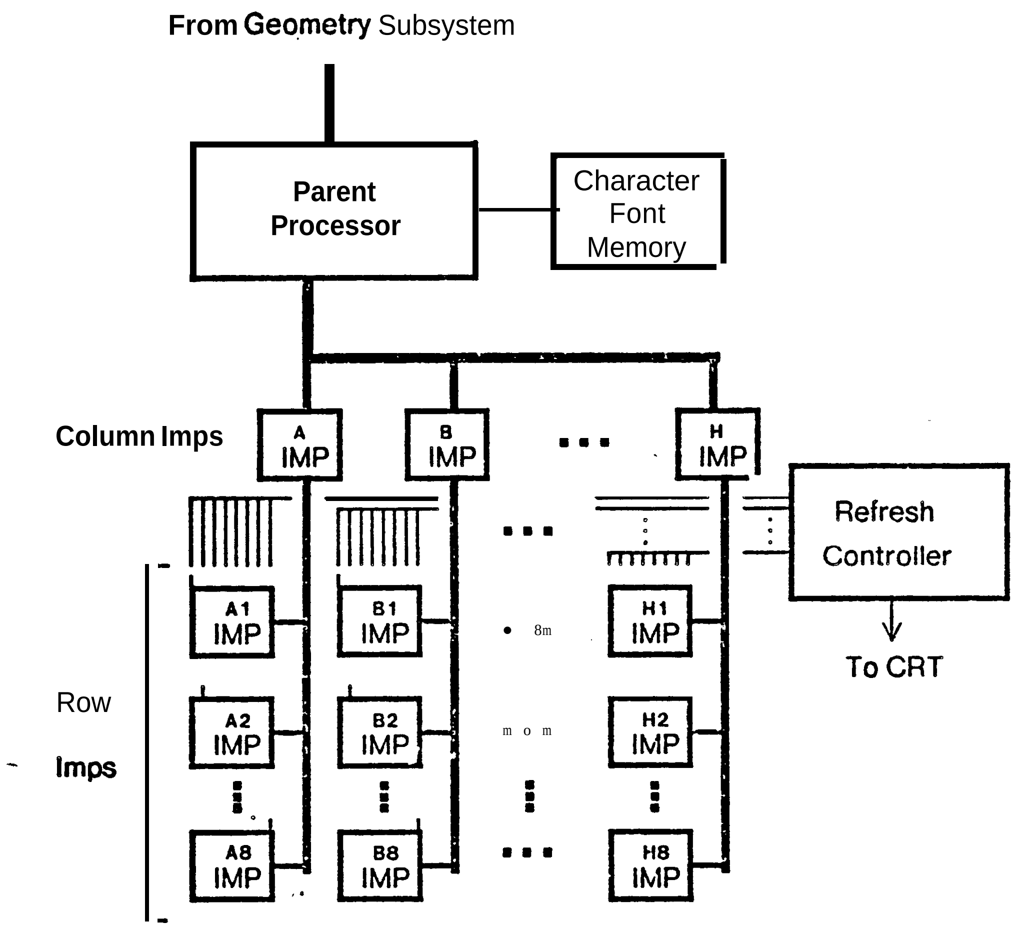

Figure 6: The Image Memory System

images usually requires some form of hidden-surface removal. Adding more bits per pixel to represent depth in what is commonly called a z-buffer [Catmull] enables the IMP to do this hidden-surface removal in the most straightforward manner: by having a depth value for each pixel on the image. In scan-converting the polygons making up the object, the IMP scan-converts not only the horizontal **and** vertical dimensions of the polygon but also its depth (z), shade and color characteristics, each of **which can be** expressed in the linear difference form that the IMP can iterate. Before writing each **pixel, the** IMP would trivially compare the depth value stored in the z-buffer with that which it is about **to write,** and write the new value only if it is in front of the present value.

Although this scheme uses a large amount of memory, it is much cheaper, comparable in speed, **and** much simpler than any existing, real-time, graphics system with hidden-surface capabilities. **Building a** system based on these ideas will:

- Drastically reduce the cost of high-performance 3D systems for flight simulation, 3D geometric design, topographic map-making and other applications in which 3D objects **must be** quickly viewed with clarity.
- Provide a useful architecture to further investigate smart memory structures.
- Provide a design to use in exploring ways of integrating very large scale systems on a **single piece of** silicon.

The problems to be solved include:

- Integrating IMP and Memory on the same piece of silicon.
- Further extending the hierarchy in the memory to improve bandwidth even more than is **presentlypossible.**

4.2 General-Purpose Processor Architectures

The architecture of today's microprocessors is often governed by a number of economic considerations that are more related to compatibility with previously existing products than to a carefully analyzed systems view of the problem. We propose to design a microprocessor that **embodies** systems concepts, such **as self-timed subsystems,** that have resulted from investigations both here at Stanford in connection with previous architecture work and at other places where VLSI considerations have been given priority. We plan to let language structure help determine instruction set design, which in turn will determine the basic architecture of the micromachine. We intend to implement a suitable microarchitecture as a vehicle for implementing a variety of target machines with different instruction sets, suitable for efficiently executing Algol/Pascal based languages.

4.2.1 Instruction Set Design

Classical instruction set design is relatively independent of the. host language. As a result current architectures tend to be large and baroque; even the most modern’architectures (e.g. VAX) have **large** and rather cumbersome instruction sets that even the best compilers cannot fully utilize. Large instruction sets cause major difficulties for VLSI implementations. First, it is difficult to implement **a large** instruction set on a single chip. Second, larger instruction sets tend to require larger instruction words. This poses a problem because of pin limitations on the chip and because of the need for wide **bus** structures.

There are several solutions to these problems:

- Use a multiple-chip implementation. This is unattractive largely because of the significant **speed** degradation encountered when going off chip.
- **Multiplex** the internal data paths and pins. For example, a single set of pins is used for both address and data lines from the processor. However, this solution usually results in a stower processor.
- Simplify the instruction set to support the language in the most economical way possible. We propose to investigate this approach, using a language oriented architecture.

Thus the aim of this research is to investigate the design of a language-oriented VLSI architecture **that** will be fast‘compared with current architectures. The Reduced Instruction Set Computer project at Berkeley has similar goals but utilizes a different approach. They employ a semi-standard, machine-level architecturewith a small instruction set. In contrast, the approach proposed here uses **a** small but language oriented instruction set with potentially powerful (but compact) instructions This design is based on language structures as opposed to the machine level structures typically generated by a compiler.

Because the architecture is language-oriented, it should be designed in such a way that the instruction set is easily altered. This will allow the easy development of processors, based on a single microprogrammable organization, for a wide variety of languages. Using SLIM, the microcode to PLA translator and microcode simulator, this is easily done. The goal is to have a general functional architecture that is easily microprogrammable into a large number of efficient language-oriented architectures.

4.2.2 Current Status

Research in building language-oriented architectures has been done both here at Stanford and elsewhere. At Stanford, Flynn's work on DELs (Directly Executed Languages) serves as a basis for beginning our study of the problem [Flynn]. They have shown size and speed improvements for executing a program using a DEL architecture to be as high as 500%. We plan to examine their results and incorporate them where possible.

The directly-executed-language approach has also been used by Xerox as a basis for Mesa [McCreight]. Mesa byte-code is directly executed on Altos by the microcode. The resulting density of Alto code vs. conventional machine code is dramatic. We have compared the size of programs in Mesa on the Altos and C on the VAX (which is claimed to be a compact instruction set). The Mesa byte-code was significantly smaller.

Western Digital has explored direct execution with their P-code chip set. However, their results are disappointing probably due to a combination of two problems. First, the implementation uses multiple chips, thus resulting in a slower implementation. Second, P-code is not a suitable representation. It was designed as a compiler intermediate form and never intended for direct execution. Optimizing language for direct execution is a different problem. The failure to do so results in significant performance degradation.

4.2.3 Proposed Research

We propose to approach the problem in two parts: design of a suitable language-oriented execution format and design of an architecture able to execute that language format and other similar direct execution languages. Eventually, we will implement both a language translator for at least one high level language (most likely Pascal based) and a processor to execute the translator output. The basic architecture of the machine should then be suitable for other language translators, such as Ada; only the microcode for the new language need be implemented.

To effectively study the issue of designing a language oriented instruction set, we need to be able to measure the effectiveness of certain designs. This measurement must come from two viewpoints: how effectively can the compiler use this instruction set, and how efficiently can it be implemented? The first question is as important as the second and has frequently been overlooked as an important part of the instruction set design process.

Thus, we propose to utilize portable compiler front-ends that we have developed at Stanford and to build a test-bed code generation system that can be retargetted to a wide variety of language-oriented instructions sets. We are considering the use of the Carnagie Mellon retargettable **backend** from the PQCC project for the basis of this test-bed. This test-bed can then to be used to determine the effectiveness of certain instruction sets. It should be possible to build-a reasonable test-bed system, **because** of the goal of a language-oriented architecture. Such a test-bed system would include instrumentation for measuring the instruction effectiveness both statically and dynamically. Given that information and cost-oriented information (in terms of instruction size and timings), we plan to optimize the instruction set design.

4.2.4 Microprocessor Architecture

Most currently available microarchitectures have been forced to be compatible with busing and **timing** structures of previous products and are consequently more complex than they might otherwise be. Current microarchitectures also use timing methodologies that do not properly address the problems of synchronization failure between processors. In addition, their designers have typically considered the memory interface a separate problem with the result that many more support chips are required for this interface. We propose to design and implement a high-performance microarchitecture from a systems viewpoint.

4.2.5 Current Status

We have designed very high-speed ALU's and barrel shifting units that will be useful in the processor. Also, the memory interface of the IMP can be used with little modification as the memory interface for this general-purpose processor. Numerous other existing parts of the-graphics projects will **be** useful in this context, such as the microprogram counter and self-timed clock.

4.2.6 Proposed Research

The main goals in the design of our microarchitecture will be simplicity, performance, and robustness. The simplicity will result from the language considerations of the previous section. The simplicity will also to a large extent determine the performance, since it seems clear from the Geometry Engine measurements that performance limitations are largely in the control PLA propagation time, and a simpler instruction set means smaller, and hence faster, control. Robustness will come from both the simpler control and data paths and the self-timed system timing methodology.

Our philosophy differs from that of the UC Berkeley RISC activity in that we propose architectures **taylored** to a language, rather than a simple instruction set onto which all languages might map. We feel that the benefits will be in performance.

The basic features of the initial micro-architecture will be independent of the language interpreter imbeded in the control. Thus, our initial micro-machine will have a data path like that of the OM2, i.e. barrel-shifter, register-file and ALU. Variations on these basic blocks will be pipelining of micro-instruction fetch and execution in straight-forward ways. Additions to the micro-architecture wilt Include:

- Simple data paths with pipelining where'necessary to decrease the microcycle time.
- A self-timed timing methodology that avoids synchronization failure, as advocated by Seitz and as used in the Geometry and Image Memory Systems.
- On-chip memory and bus interfaces.
- Small on-chip associative instruction and data cache to decrease off-chip memory delays.

The key point being addressed by this proposed work is that in order to seriously address integration of very large systems, we must have building blocks. Already, we have felt the need for a general-purpose micro-engine configured as a pipelined computing engine in the Parent Processor position of the Image Memory System. A significant portion of the cost of that system could be eliminated if we presently had such a processor designed. The problem would be reduced to one of writing the micro-code, just as in current TTL processor implementations based on the ADM2900 series. (We are now adopting a policy that all micro-control store specifications in our system-building work use the SLIM system. A post-processor will then generate either EPROM programming input for discrete-component designs or PLA layout. Much of the control specification for the Parent Processor will then be directly portable to the micro-architecture work proposed here.)

At Stanford, we disagree with the view expressed by some that the University community need not get involved in the design of general-purpose processors and memories. We feel that by embarking on ambitious tasks such as these, we will have a useful forcing function for design tool development. Moreover, much of the insight in improving ways of building systems comes from building them and making them work.

Publications

[Clark] J. H. Clark, “A VLSI Geometry Processor for Graphics,,’ Computer, Vol 13, No. 7, July, 1980.

[Clark] J. H. Clark, “Structuring a VLSI System Architecture,” Lambda, 2nd Quarter, 1980.

[Clark&Hannah] J. H. Clark and M. R. Hannah, “Distributed Processing in a High-Performance Smart Image Memory,” Lambda, 4th Quarter, 1980.

[Hennessy] J. Hennessy, “SLIM: A Language for Microcode Description and Simulation in VLSI,,, CSL Tech. Rep. # 193, Computer Systems Laboratory, Stanford University, July 1980, also Second Caltech Conference on VLSI.

[Clark, Hennessy&Hannah]
J. H. Clark, J. L. Hennessy and M. R. Hannah, “VLSI System Control Structures, " CSL Tech. Rep., in Preparation, Stanford University.

[Clark] J. H. Clark, “Some VLSI System Timing Conventions”, CSL Tech. Rep., In Preparation, Stanford University.

References

[Baker] C. Baker, “Analysis Tools for VLSI,” Masters Thesis, Dept. of Electrical Engineering, MIT 1980.

[Bechtolsheim] A. Bechtolsheim and F. Baskett, “A Low-cost Raster-graphics Terminal,” Proceedings SIGGRAPH 80, July, 1980.

[Bochman] G. V. Bochman, “Hardware Specification with Temporal Logic: an Example”, Computer Systems Laboratory, Stanford University, unpublished.

[Catmull] E. Catmull, “A Subdivision Algorithm for Computer Display of Curved Surfaces,” Univ. of Utah Computer Science Dept. Tech Rep., UTEC-CSc-74-133, Dec. 1974.

[Clark] J. H. Clark, “A VLSI Geometry Processor for Graphics,” Computer, Vol. 13, No. 7, July, 1980.

[Clark&Hannah] J. H. Clark and M. R. Hannah, “Distributed Processing in a High-performance Smart Image Memory,” Lambda, 4th Quarter, 1980.

[Conway] Conway, L.A., Bell, “A., and Newell, M. E., MPC79: A Large Scale Demonstration of a New Way to Create Systems in Silicon,” Lambda, 1st Quarter, 1980.

[Flynn] M. J. Flynn, “Directions and Issues in Architecture and Language,” Computer, Vol. 13, NO. 10, October, 1980.

[Francez] N. Francez and A. Pnueli, “The analysis of cyclic programs”, Acta Informatica 9 (1978) 133-157.

[Good] D. I. Good and R. M. Cohen, “Principles of proving concurrent programs in Gypsy”, Proc. Principles of Programming Languages (1979) 42-52.

[HailpernA] B. T. Hailpern and S. Owicki. “Verifying Network Protocols Using Temporal Logic”, Proceedings of Trends and Applications, 1980: Computer Network Protocols, NBS, 1980.

[HailpernB] B. T. Hailpern. Verifying Concurrent Processes Using Temporal Logic. PhD Thesis, Computer Science Department, Stanford University, 1980.

[Hennessy] J. Hennessy, “SLIM: A Language for Microcode Description and Simulation in VLSI,” CSL Tech. Rep. # 193, Computer Systems Laboratory, Stanford University, July 1980, also Second Caltech Conference on VLSI.

[Howard] J. Howard, “Proving monitors”, CACM 19:5 (May, 1976) 273-278.

[Keller] R. M. Keller, “Formal verification of parallel programs”, CACM 19:7 (July, 1976), 371-384.

[Lamport]	**L. Lamport,** "Proving the correctness of multiprocess programs", IEEE Transactions on Software Engineering **3:2** (1977) **125-143.**
[McWilliams]	W.R. **Bryson,** P.M. Farmwald, T. M. McWilliams and B. **Rubin,** ***The*** S-l Project, Lawrence Livermore Labs Tech. Report UCID- **18619, 1979** Annual Report.
[McWilliamsB]	**T.** M. McWilliams, "Verification of Timing Constraints on Large Digital Systems," **PhD** Thesis, Computer Science, Stanford University, May 1980.
[Mead]	**C. A.** Mead and L. A. Conway. *Introduction* ***to VLSI*** *Systems.* Addison-Wesley, **1980.**
[Owicki A]	S. Owicki, and **D.** Gries, "An axiomatic proof technique for parallel programs", **Acta** Informatica 6 **(1976), 319-339.**
[OwickiB]	S. Qwicki and **D.** Gries, "Verifying properties of parallel programs: an axiomatic approach", CACM **19:5** (May, 1976) 279-284.
[OwickiC]	**S.** Qwicki "Specifications and Proofs for Abstract Data Types in Concurrent Programs", in F. Bauer (eci.), Program Construction, Springer-Verlag,, 1979.
[OwickiD]	S. Owicki and L. **Lamport** "Proving Liveness Properties of Concurrent Programs'" submitted for publication.
[SeitzA]	C. **Seitz,** "System Timing'" Chapter 7 in C. Mead and **L.** Conway, *Introduction to* ***VLSI*** *Systems,* Addison-Wesley, 1980.
[SeitzB]	C. **Seitz,** "Ideas about Arbiters," Lambda, First Quarter, 1980, pp. **10-14.**
[Sutherland]	I.E. Sutherland, C. Molnar, **C.** Mudge, R. Sproull, "The **TriMosBus,"** Proceedings **of CalTech** Conference on VLSI, Computer Science, 1979.
[Terman]	**Terman,** MIT Switch Level Simulator.

Research Staff

Faculty

Forest Baskett	Principal Investigator, SUN workstation, analysis design aids.
Jim Clark	Associate Investigator, Geometry Engine, Image Memory Processor, design aids.
John Hennessy	Associate Investigator, SLIM, design aids.
Susan Owicki	verification and specification of concurrent VLSI systems.
Brian Reid	knowledge representation and library support for Cascade.
John Wakerly	SUN workstation.

Research Associate

Tom Davis	development of graphics front-end for Cascade/SCALD, relative layout description language, constraint satisfaction.

Students

Andy Bechtolsheim	SUN workstation.
Mark Hannah	(unsupported) Image Memory Processor, PLA layout program for SLIM.
Thomas Gross	(unsupported) CIF support software (e.g. Check plot software).
Cary Kornfeld	(unsupported) worked on retargettable code generator, to be used in instruction set studies.
Andrew Schneider	SUN workstation.
Michael Spreitzer	logic extraction and equivalence.
Stephan Demetrescu	working on test environment.
N. Yamanouchi	concurrent hardware specification and verification.

Technical Staff .

Jeff Mogul	programming support for the workstation and VAX.
Mark Grossman	research and development engineer support for the SUN workstation hardware development, and the integration of the Geometry Engine and Image Memory Processor.