Semiannual Technical Summary

Restructurable VLSI Program

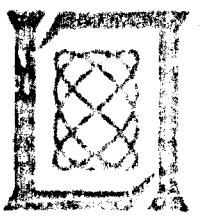
31 March 1982

Prepared for the Defense Advanced Research Projects Agency
under Electronic Systems Division Contract F19628-80-C-0002 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LEXINGTON, MASSACHUSETTS

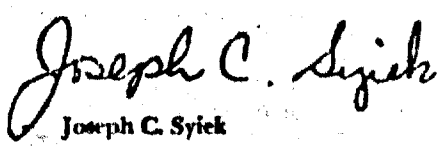


The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Defense Advanced Research Projects Agency under Air Force Contract F19628-80-C-0002 (ARPA Order 3797).

The Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER



Joseph C. Syiek
Acting ESD Lincoln Laboratory Project Officer

ESD-TR-82-059

AD-A120 471

ERRATA SHEET
for
SEMIANNUAL TECHNICAL SUMMARY:
RESTRUCTURABLE VLSI PROGRAM (31 MARCH 1982)

The ESD-TR number printed on the cover and report documentation page of MIT/LL Semiannual Technical Summary, "Restructurable VLSI Program," dated 31 March 1982 is incorrect. Please change the number to ESD-TR-82-059.

29 October 1982

Publications
M.I.T. Lincoln Laboratory
P.O. Box 73
Lexington, MA 02173-0073

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

RESTRUCTURABLE VLSI PROGRAM

SEMIANNUAL TECHNICAL SUMMARY REPORT TO THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

1 OCTOBER 1981 — 31 MARCH 1982

ISSUED 16 AUGUST 1982

LEXINGTON MASSACHUSETTS

ABSTRACT

This report describes work on the Restructurable VLSI Research Program sponsored by the Information Processing Techniques Office of the Defense Advanced Research Projects Agency during the semiannual period 1 October 1981 through 31 March 1982.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

I. PROGRAM OVERVIEW AND SUMMARY

A. OVERVIEW

The main objective of the Lincoln Restructurable VLSI Program (RVLSI) is to develop methodologies and architectures for implementing wafer-scale systems with complexities approaching a million gates. In our approach, we envisage a modular style of architecture comprising an array of cells embedded in a regular interconnection matrix. Ideally, the cells should consist of only a few basic types. The interconnection matrix is a fixed pattern of metal lines augmented by a complement of programmable switches or links. Conceptually, the links could be either volatile or nonvolatile. They could be of an electronic nature such as a transistor switch, or could be permanently programmed through some mechanism such as a laser. The RVLSI Program is currently focusing on laser-formed interconnect.

The link concept offers the potential for a highly flexible, restructurable type of interconnect technology that could be exploited in a variety of ways. For example, logical cells or subsystems found to be faulty at wafer-probe time could be permanently excised from the rest of the wafer. The flexible interconnect could also be used to "jump around" faulty logic and tie-in redundant cells judiciously scattered around the wafer for this purpose. Also, the interconnect could be tailored to a specific application in order to minimize electrical degradations and performance penalties caused by unused wiring.

Further, the testing of a particular logical subsystem buried deep within a complex wafer-scale system poses a very difficult problem. A properly designed restructurable interconnect matrix could be temporarily configured to render internal cells both controllable and observable from the wafer periphery. In this way, each component cell or a tractable cluster of cells could be tested in a straightforward manner using standard techniques.

With an electronic linking mechanism it is possible to think in terms of a dynamically reconfigurable system. Such a feature could be used to alter

the functional mode of a system subject to changes in the operating scenario, or it could be used to support some degree of fault tolerance if the system architecture was suitably designed.

Several major areas of research have been identified in the context of the RVLSI concept:

(a) System architectures and partitionings for whole-wafer implementations.
(b) Placement and routing strategies for optimal utilization of redundant resources and efficient interconnect.
(c) Assignment and linking algorithms to exploit redundancy and flexible interconnect.
(d) Methods for expediting cell design with emphasis on functional level descriptions, enhanced testability, and fault tolerance.
(e) Methods for testing complex, multiple-cell, whole-wafer systems.

Complementary work on the development of various link and interconnect technologies as well as fabrication/processing technology is being supported by the Lincoln Air Force Line Program, and results are reported under the Lincoln Laboratory Advanced Electronic Technology Quarterly Technical Summary.

B. SUMMARY OF PROGRESS

Work for this period is reported in six sections: Design Aids for RVLSI (Sec. II), Restructurable System Studies (Sec. III), RVLSI Technology (Sec. IV), Testing (Sec. V), and Applications (Sec. VI). The following sections summarize progress to date.

1. Design Aids

Progress on the MACPITTS silicon compiler continues at a satisfactory pace; automatic power bus sizing, automatic organelle input ratio selection, and Weinberger gate array layout features were installed. Also, the boolean flags layout and automatic placement/interconnect mechanisms were completed. Several test designs have been submitted to MOSIS for fabrication. The PAL 81 system was modified to satisfy the latest requirements of the laser zapping process, which included a reordering of the zap sequence to accommodate human intervention and testing. A flexible, VAX-based laser controller subsystem, compatible with human intervention and testing, is now operational. A basic framework for PAL 82, a general assignment/linking system suitable for research and production applications, has been defined with a basic set of primitives. Major subsystems include a constructor, planner, and executor.

Work has begun on a VAX-based design aid package for CMOS. Based on the same technology-independence philosophy as the MDRC mask design rule checker, the package will comprise a node extractor, a static electrical rules checker, and possibly a switch-level simulator in addition to the MDRC itself. The MDRC proper is now operational on the VAX for both MOSIS NMOS and Lincoln bulk CMOS processes.

2. Restructurable System Studies

Fixed segmentation interconnect structures have been reevaluated in the light of recent processing experience, the requirements of dynamic restructurability, and the implications of mapping mesh-connected topologies into defective arrays. The behavior of fixed segmentation requirements for defect avoidance in 1-D/nearest-neighbor topologies has been examined. Also, some theoretical analyses have been performed relating circuit complexity to pin-out requirements. This is very useful in formulating partitioning strategies for restructurable systems.

3. Testing

An incremental constructive self-test strategy for 1-D/nearest-neighbor topologies has been analyzed. A simple, effective technique has been devised to monitor the state of the wafer-scale interconnect matrix during laser programming. Experimental verification of a novel approach for applying optical probing techniques to NMOS is underway. Also, a Tektronix S3260 integrated circuit tester has been obtained and is being used on packaged devices. A wafer-probing module will soon be interfaced.

4. RVLSI Technology

Initial results on the MOSIS second-level metal experiment have been obtained. Vias in the 6-8 m size range can be reliably fabricated using AlSiCu alloy on the second level and a hydrofluoric acid glint. No appreciable problems with metal-to-metal shorts have been encountered. Also, two new approaches are being evaluated for providing a simple, one-level metal laser-link capability to the MOSIS community. The approaches differ in the type of insulating material used in the gap between two adjacent metal segments.

5. Applications

Some Phase I digital integrator cells have been successfully operated, validating the basic logic design. Work continues in designing the wafer-scale interconnect. A flexible multiply/add cell has been defined with an architecture for a new digital-signal processing building block. The building block can be used for correlation or convolutional filtering and can be readily customized to various throughput and impulse response length requirements by personalizing the intercell connectivity. Other, less obvious uses for the cell have been discovered in the radar processing area.

II. DESIGN AIDS FOR RVLSI

A. MACPITTS

The major components of the MACPITTS software system as reported previously and their interrelations are depicted in Fig. 1. The present implementation status of these components is given below. "Tested" in this section means that a given module has been integrated in the latest version of the MACPITTS compiler, and that MACPITTS programs exercising that module have passed checks including simulation and layout rules checking. The simulation test of a layout module implies a switch-level simulation, not a functional simulation.

Technology-Independent Feature Extraction includes a code to extract the data-path, sequencer, control, and flag information from a MACPITTS program. Because the user provides only an algorithm, these items are not explicitly specified. These modules have been designed, coded, and tested.

Data-Path Layout is designed, coded, and tested.

Sequencer Layout is designed.

Control Layout, also referred to as the Weinberger gate array, is designed, coded, and tested.

Flag Layout is designed, coded, and tested.

Pin Layout is designed and coded, but not yet fully tested.

Placement and Routing of signals other than to and from I/O pins has been designed, coded, and tested. Placement and routing of the I/O pads and their signals have been designed and coded, but not fully tested.

Organelle Library is designed, coded, and tested, although it is continually being upgraded.

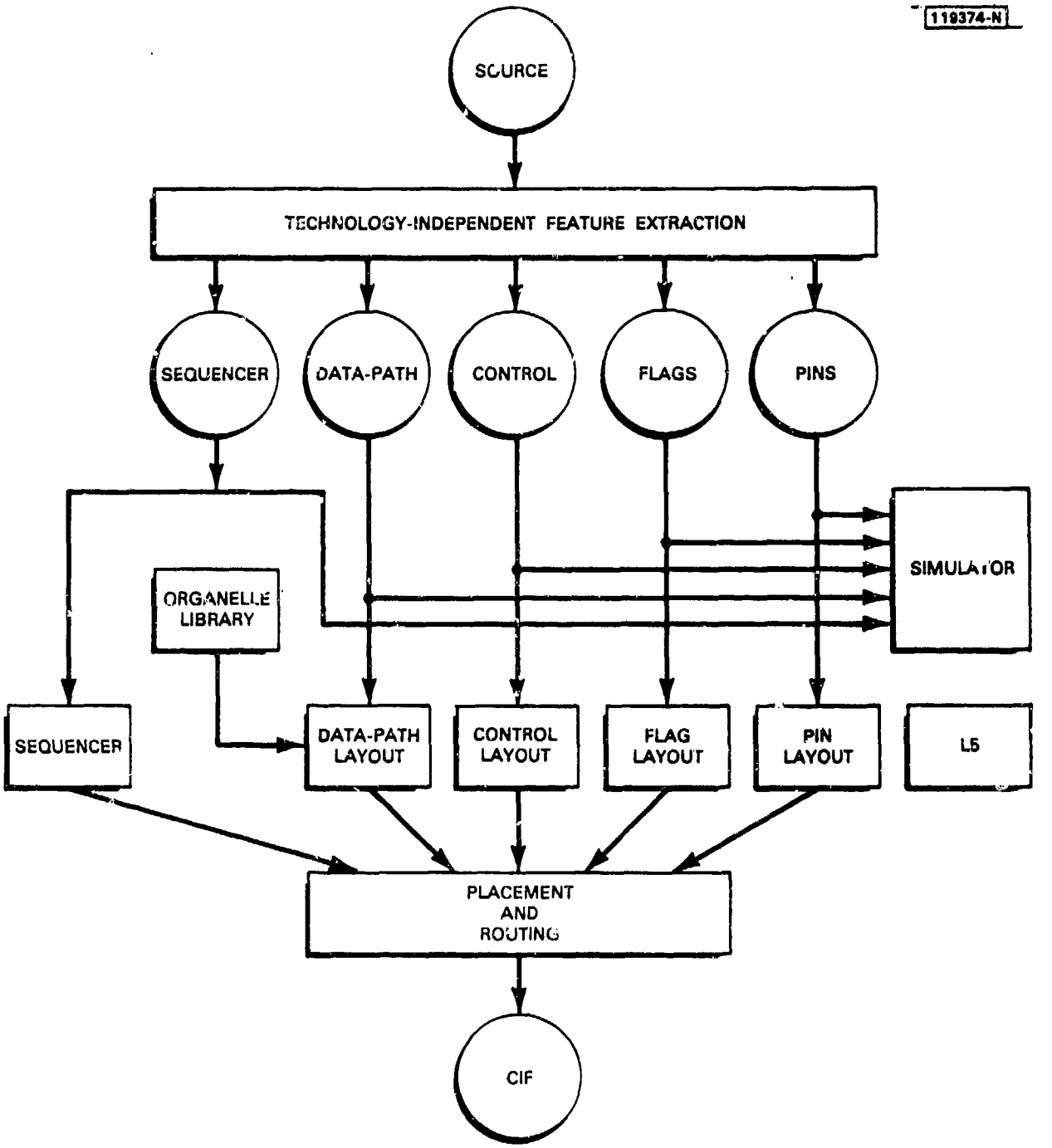


Fig. 1. MACPITTS structure.

- L5 is the Lincoln Laboratory LISP-based Layout Language. Like the organelle library, an adequate version is designed, coded, and tested, but is continually being refined and upgraded.
- Simulator - A version of the simulator is designed, coded, and tested, but undergoing occasional revision.

Five basic designs have been used to check the MACPITTS system and are described below. The MACPITTS code for each is provided in Appendix A.

- abs is a chip which accepts in a twos complement number and outputs its absolute value. The logic is extremely simple, and there is no state information.
- addr is a chip which is a programmable address controller/matcher. An address range can be programmed. In addition, the "match" output signal can be programmed to be active high or low. While there is no process state, a flag is utilized.
- counter is a relatively inefficient version of a variable module counter. However, this particular form of the design utilizes a sequencer with stacking capability.
- taxi is a simple finite-state machine-based controller element similar in spirit to the Mead-Conway traffic light controller example.
- toc was first described in the March 1981 Quarterly Technical Summary and is a cascadable tester slice for dynamic circuits. It was the toc design effort that motivated the development of MACPITTS.

Several changes have been made to the MACPITTS ideas presented in the September 1981 Semiannual Reports. These are described below.

Sequencer Layout

Because of problems in integrating the initial sequencer layout module with the rest of compiler, the module was completely rewritten.

Data-Path Extraction

A major flaw was detected in the algorithm previously used to extract the intermediate level code from the source code, necessitating a major rewrite of the technology-independent portion of the compiler using new algorithms. These changes enhanced the clarity and readability of the code, and several improved features in the language were implemented. During this revision the compiler was inoperable and in a state of constant flux for about one month. These revisions have now been successfully completed, and those portions of the compiler which have been fully implemented at present--the intermediate code extraction and data-path layout--are operational and seem quite robust.

The problem was as follows. The MACPITTS compiler consists primarily of the two levels of routines. The upper level compiles from the MACPITTS source language into a technology-independent intermediate level code. The lower level routines use this code to construct the layout of the circuit. The intermediate level code consists of three major parts that describe (1) the sequencers used for each process, (2) the data-path, and (3) the control logic. The original algorithms for the upper level of the compiler extracted each part of the intermediate code separately: first the sequencers, then the data-path, and finally the control. This caused a basic problem in that necessary information pertaining to the allocation of particular organelles to statements in the source program was not preserved from data-path extraction time to control extraction time. This permitted the control extractor to make mistakes in choosing which copy of an organelle in the data path to allocate to a particular statement in the source program. Consider the following example:

```
(program t2 8
  (def a register)
  (def b register)
  (process corrupt 0
state0
  (setq a (1+ a))
state1
  (setq b (1+ b))
state2
  (par (setq a (1+ a)) (setq b (1+ b)))))
```

This would result in the following data-path specification:

```
((register a -1 (((internal 2))))
 (organelle + -2 (((internal 1)) ((internal 3))))
 (register b -3 (((internal 4))))
 (organelle + -4 (((internal 3)))))
```

When generating the control for the above program, "state0" would increment register "a" using unit number two. "State1" would increment "b" using unit number two as well since it is the first unit capable of incrementing register "b". "State2" would likewise allocate unit number two for incrementing "b" and then find that no remaining unit was suitable for incrementing "a" since the wrong choice was made for the allocation of the "b" incrementer in "state2".

Two solutions were considered. It was possible to backtrack over incorrect allocations and try again or make use of the information used by the data-path extractor for allocating organelles during control extraction as well. The latter alternative was chosen as the conceptually cleaner one.

The new algorithm extracts the data-path and control from a program simultaneously. The extraction routine recursively traverses the program extracting primitive data-path and control pieces from basic program constructs and, as it unwinds up the tree, merges these pieces into the complete data-path and control for the program. One of two merge routines is used depending upon the particular construct being compiled. The "merge parallel" routine combines two data-paths so that they may operate in parallel. All necessary organelles are duplicated. The "merge exclusive" routine combines two data-paths that are known never to execute together. This can occur either between two different states or between two exclusive consequents in a cond statement. In this case, duplicate units are factored out resulting in a more compact data-path.

An additional optimization is performed during parallel merging. Normally two copies of a given unit will be included in the merged data-path

if one copy appears in each of the two data-paths being merged. If, however, the unit does not have any control inputs and both copies of the unit have identical singleton multiplexers, then they will always compute the same results and only one copy is needed. This single optimization greatly reduces the size of most of the data-paths generated so far.

While rewriting the code, a new routine called "defstruct" was used. This allows the definition of new data structures and automatically generates functions for creating and manipulating those structures. Use of this package has made the MACPITTS compiler considerably more readable and easy to modify and maintain.

Integer and Boolean Types (Including Flags)

The syntax of the MACPITTS source language was simplified considerably. The old syntax distinguished among forms, formulas, and conditions. The new syntax considers all program constructs to be forms which may return values of various types. Conditions are now forms of type boolean, formulas are forms of type integer, and forms that do not return a value are termed void. If a construct expects a form of a given type such as boolean but an integer form is given, the form may be coerced or converted to the proper type. This coercion mechanism has not yet been fully implemented but the necessary access mechanics have been installed.

The concept of distinguishing integer and boolean types allowed two expansions of MACPITTS. First, the language now includes flags that store boolean values just as registers store integer values. It is no longer necessary to construct a new process for each bit of boolean storage. Second, functions and tests can accept boolean as well as integer arguments. This allows the design of such units as shifters where the bit to be shifted in is dependent upon some boolean condition. This feature promises to be useful in the design of arithmetic operations such as multiplication and division.

B. PLACEMENT/ASSIGNMENT/LINKING

1. Refinements to PAL 81

The PAL 81 system described in the last semiannual report was modified to satisfy the latest requirements of the laser zapping process. These requirements necessitated reordering the laser zap process to accommodate possible human intervention and testing. Intervention and testing is required because of the inaccuracies in the present laser table and uncertainties of some processing elements and not because of assigning and linking algorithm inadequacies.

Because of the wide range of observed cell yields, it became necessary to use a different assignment algorithm than previously contemplated. The algorithm currently used is described in the September 1981 Semiannual Technical Summary, Sec. III-C. The salient point of this algorithm is its backtracking capability, which allows much greater flexibility during the assignment process. The linking algorithm was modified also so that logical (from the human interpretation viewpoint) groups of nets are linked in identifiable clumps.

Control of the laser table and laser zapping process has mostly been moved from the Apple laser controller to the VAX computer. This is in line with the requirement for human interaction since the data base containing human readable information is too big for the Apple. Keeping the Apple code simple also allows considerably more development on the time-sharing VAX system rather than tying up the Apple/laser system. Section B.4. discusses the VAX/Apple/Laser Table Interface in more detail.

2. Review of the PAL 81 System

In developing the PAL 81 system, a number of lessons have been learned from experimental results. Before reconsidering the PAL 82 framework, assumptions in the original conception of PAL 81 that affect PAL 82 and have since proven inappropriate are as follows.

- Wafer defects are concentrated in the cells. Not only is this false, but information about the "goodness" of interconnect is subject to rapid change as lines are cut, fuses zapped, etc.
- The desired logical system is fixed for a given wafer. Possibly false for technical as well as practical reasons (e.g., customization and testing).
- Wafer I/O pads are equivalent to cell pins for PAL purposes. In reality, a wafer I/O pad may be connected directly to a long interconnect line, as shown in Fig. 2(a), as opposed to Fig. 2(b).

These initial assumptions made the PAL 81 system operationally unwieldy for the digital integrator application as its design ultimately evolved. Too many operations needed tedious human specification for truly practical operation.

3. PAL 82

PAL 82 is currently envisaged as a planner-actor system. The planner section takes an initial world description and a goal world, and returns an

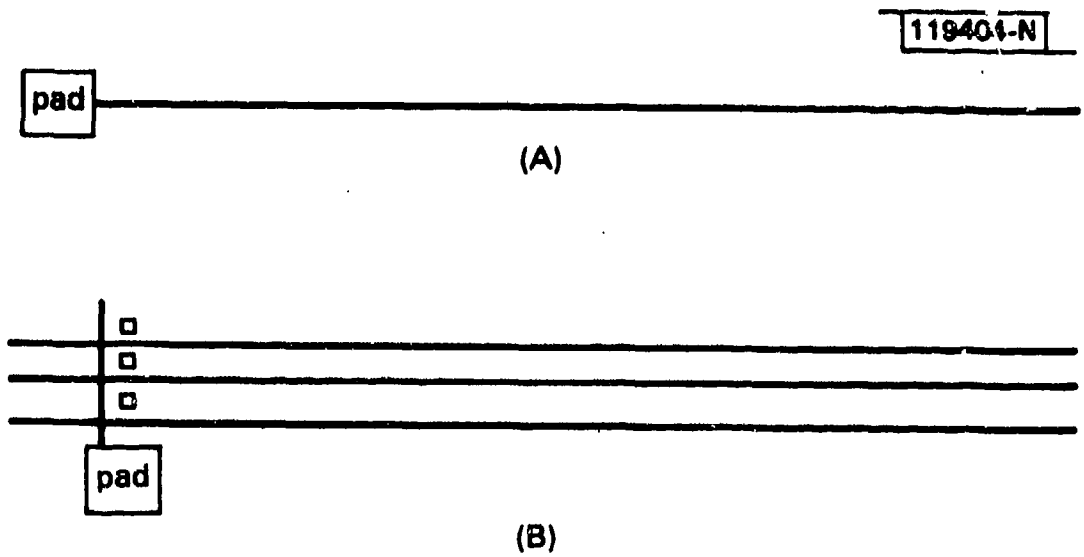


Fig. 2. Examples of pad placement.

action sequence which, if executed, would achieve the goal. The action sequence may include tests and opportunity for human interaction. As the sequence is executed, the world description is updated.

Test failures and appropriate interactive commands will cause an inconsistent world description. In such a case the remaining plan is discarded, the world description is reintegrated to correspond to reality, and the planner section is reentered. An extension to the planner would allow an initial helping plan or the remainder of a failed plan to be used heuristically.

It is quite possible that the planner will be unable to find an action sequence for the goal, in which case the designer may provide alternate goals which can be tried. For example, an integrator-type wafer may not have enough cells to form a 4 x 4 array (the primary goal), but a 4 x 3 arrangement may be of some use. Interactive goal specification would also be possible at this point.

The PAL 82 system (Figure 3) consists of:

- A constructor program by which a designer initially specifies a world description. The constructor program will specify both a physical wafer and a logical system, so it may be helpful to consider these two functions separately.
- A planner program as discussed above. A planner program might consist of assigner and linker routines with algorithms similar to those presented in previous reports. One modification to previous routines is that the planner must be able to restart with modified world descriptions, not merely the initial one.
- An executor program that impacts the real world. This program would command the laser table, receive and interpret test results, etc.
- The PAL 82 framework which comprises functions and structures that the constructor, planner, and executor programs should use for efficient implementation. The PAL 82 framework is presented in more detail below.

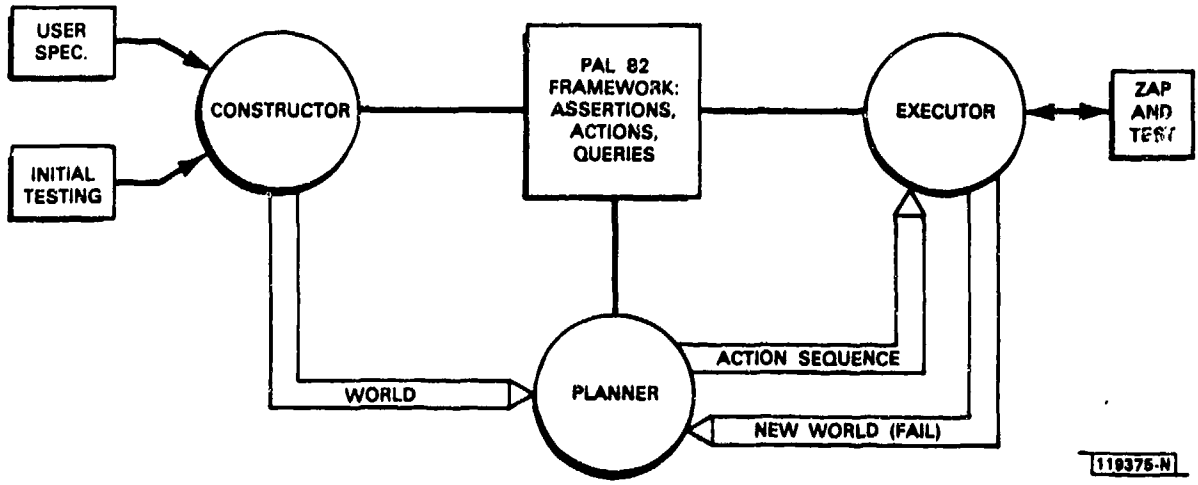


Fig. 3. PAL 82 system.

The PAL 82 framework consists of:

- A world description including the physical wafer description and a logical system description. Goals are part of the logical system description, but so may be assertions about goals being completed. For example, if through some actions one of several logical nets is instantiated, the world description must recognize that fact, especially if a later test is failed and replanning is necessary. Many world descriptions may be extant simultaneously, especially while the planner is operating.
- Assertions that cause modification of the world description. These would include functions to generate the description.
- Actions that cause real world as well as world description modifications. Every action has a simulated action counterpart which modifies a world description and creates a planned "real" action sequence. Simulated actions are of use during planning.
- Queries that interrogate a world description for information.

Appendix B contains a partial list of suggested framework functions. Aspects of the world description data format are more subtle depending on performance and size tradeoffs. Since designers' programs use the framework functions, the exact format description is hidden. Therefore, its exact specification has been postponed.

4. VAX/Laser Table Interface

A system has been put in place for driving the laser table directly from the VAX. It consists of two cooperating pieces of software, one residing on the Apple laser controller, the other on the VAX. The two halves communicate with each other over a serial line, in an interlocked fashion. The VAX transmits all information redundantly with an error recovery scheme.

An operator using a VAX terminal can completely control the wafer alignment process and automatic flow of commands. Linking and other automatic functions are performed by sequencing command streams in files to the driver. The operator can interrupt automatic processing at any point and intervene with commands of his/her own.

There has been established a notion of "selected" position and "current" position. The table is always physically at the current position. The selected position is the last place the table was moved to using a position selecting command. It is like a cursor that can be left at a particular point. For each of the commands the effect on the current and selected position has been defined. Alignment is performed by selecting a position, moving the current table position to where that selected point really is, and then issuing an alignment command.

A powerful feature of this interface is that it supports a general stacking scheme for switching among many input sources. An operator can request that input be taken from a file, then suspend automatic operation by typing an interrupt. Commands can then be entered from the keyboard. A new input stream can be nested from this point, or the interrupted stream can be removed from the stack of input streams, or that stream can be resumed from the point of interruption. It is possible to stack as many input streams as

there are allowable simultaneously open files. Using this feature it is easy to write macros to do alignment and ask the operator to inject commands. The These macros can be called by the linking files, the operator, or other macros. A summary of commands is provided in Appendix C as well as descriptions of the VAX/Apple serial interface and low-level VAX drivers.

C. DESIGN RULE CHECKING AND NODE EXTRACTION

The MDRC (Mask Design Rule Checking) system is now operational on the VAX. The MOSIS NMOS rules were coded, and to demonstrate operation, three designs were checked. Two of these originated on the VAX (via LICL) and one originated on the Calma. No false alarms were reported and run times were satisfactory, one and one-half to three times faster than the MIT checker. At the request of MIT, a copy of the NMOS MDRC has been provided for use on the Real Time Systems Group VAX.

The strawman set of CMOS bulk geometrical design rules was obtained from JPL. These rules were easily coded in the MDRC system. This exercise demonstrates the flexibility of the MDRC system in adapting to process changes.

A node extractor for CMOS is currently in progress and is nearly complete. This is a valuable addition to the current package of design aids, providing the input for static electrical rules checking as well as functional simulation.

III. RESTRUCTURABLE SYSTEMS STUDIES

A. YIELD MODELING AND COMPLEXITY THEORY

In formulating the advanced yield model introduced in earlier reports, Rent's Rule was mentioned. Because of certain observed anomalies and to better understand the pin-to-transistor relationship for VLSI, further investigations were conducted.

When a digital logic circuit is partitioned into subcircuits, there exists a relationship between P, the number of pins of the circuit; b, the average number of pins per subcircuit; and N, the number of subcircuits. This relationship is traditionally expected to be of the form $P = bN^R$. Experimental evidence has been introduced supporting this model and suggesting a value for R of $0.57 < R < 0.75$. This range is customarily cited for interconnect-capacity requirement calculations.

If the partitioning is done hierarchically obeying certain rules of "self-similarity" and balance, Rent's Rule can be derived analytically. The expression for $\overline{R}$, the average exponent over all hierarchies, is dependent on the number of pins at the system level and the number of transistors.

Figure 4 shows two hierarchy levels of a one-dimensional channel-type system. This figure is only for intuition; our calculations do not depend on one-dimensional channels and are equally valid for any layout. Self-similarity means that except for explicitly varying statistical properties, every level looks like all the other levels. Some definitions we will use are:

P_I = number of pins for each cell at level I
C = cluster factor of the hierarchy (i.e., 2)
b = number of pins at lowest level (3 for transistors)

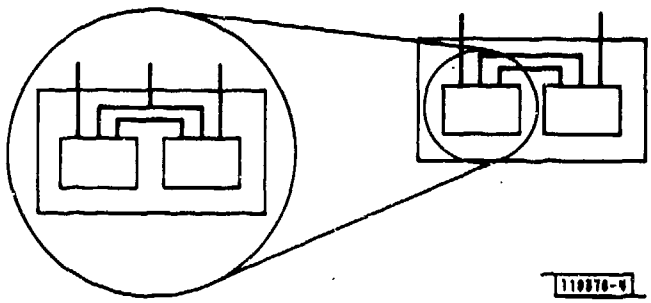


Fig. 4. Self-similar hierarchy.

$$A_I = \frac{P_I}{P_{I-1}}$$

$$R_I = \log_c A_I$$

where I and I-1 are hierarchy-level numbers. The cluster factor C is somewhat artificial. On the one hand, most designs do not have a constant hierarchical clustering factor. On the other hand, the A_I's are arbitrary, which means that the system could be artificially partitioned with any arbitrary clustering factor without violating the assumptions. In any case, no explicit dependency on C appears in the final result.

Finding the number of transistors (M) at any given level of the hierarchy (I) is relatively simple:

$$M = C^I \tag{1}$$

The total number of transistors (N) and number of levels (K) in a system are related by $N = C^K$.

Finding the pin count of a given level of the hierarchy implies solving the recurrence equations:

$$P_I = A_I P_{I-1} \tag{2}$$

$$P_o = b$$

Evaluating this expression for level K(K > 1) we find:

$$P_K = b \prod_{I=1}^{K} A_I$$

$$= bC^{\sum_{I=1}^{K} R_I}$$

$$= bC^{K\bar{R}} \tag{3}$$

where $\bar{R}$ is the average of the R_Is: $\bar{R} = \frac{1}{K} \sum_{I=1}^{K} R_I$. Equations (1) and (3) can be combined to yield:

$$P = bN^{\bar{R}} \tag{4}$$

Inspecting Eq. (4), we note that P, the system pin-out; b, the number of leads on a transistor; and N, the number of transistors of the system, are all observables. This means that $\bar{R}$ can be calculated for any design. The assumption of full self-similarity would imply that all $R_I = \bar{R}$; however, our result shows that even when this assumption is relaxed, the average Rent exponent of the system can be found.

Performing the calculation for several systems spanning the MSI to LSI range of complexity we found that $0.2 < R < 0.4$ is the norm for commercial and custom chips. For example, the $\overline{R}$ for the Phase 0 Packet Radio Integrator chip is $\log_{7000} \frac{24}{3} = 0.2$. The $\overline{R}$ for an FET-16 wafer being built for radar-signal processing applications is $\log_{1500} \frac{64}{3} = 0.27$.

B. FIXED VS. ARBITRARY INTERCONNECT SEGMENTATION

Recent developments have necessitated a reassessment of the value of arbitrary interconnect segmentation in restructurable wafer-scale systems.

Preliminary statistics, based on the first fabricated wafers, show that while the operations of segmenting lines and fusing links by laser are highly reliable, continuity in wafer-long segments may be a problem.

The linear next-neighbor topology may be considerably more important than previously realized for the purposes of testing. If testing is considered during the design stage, the intermodule connection topologies for the test phase and the operation phase could be different. Specifically, the test topology may be configured as linear next-neighbor without loss of operational generality.

In some recent work at MIT,* a little known formula for the distribution of maximum skip lengths has been applied in considering the problem of mapping mesh-connected topologies into defective arrays. While the initial application was for assignment purposes, there is an obvious extension to fixed segmentation requirements. Further, it has always been recognized that dynamic restructurability would necessitate fixed segmentation, because no known electrically alterable link technology can support arbitrary segmentation.

*C. Leiserson, T. Leighton, private communication.

C. DEFECT AVOIDANCE IN LINEAR NEXT-NEIGHBOR TOPOLOGIES

In a linear next-neighbor topology the ability to skip n cells is equivalent to the ability to "survive" a sequence of n cell failures. The ability to skip n is also directly proportional to a required channel width in a fixed segmentation environment. Thus, there is considerable interest in the exact statistics of system size and yield based on n.

The statistics of n can be computed based on cell yield and total number of cells in a given system. What is really of interest, however, are the statistics of k--the number of good cells encountered before a sequence of n failures. This parameter k can mean the maximum implementable system size given a maximum skip capability of n. We should be able to show that the characteristics of k are dependent on:

n: the maximum skip capability,
p: the cell yield, and
N: the number of cells.

To normalize the probability curves, k or average k ($\bar{k}$) is divided by A = pN. A is the average number of good cells and can be thought of as the maximum implementable system given $n = \infty$. Because A is the expected system size given fully adequate interconnect, $\bar{k}/A$ can be interpreted as a size-reduction factor caused by inadequate interconnect for redundancy.

Simulations of wafers with random defects based on typical cell yields were measured for their k vs. n vs. p behavior (Fig. 5). It is seen, for example, that a skip capability of 6 will yield a size reduction factor of 0.4 at 40% cell yield but a 0.9 reduction factor at 60% cell yield.

Another interesting aspect of the distribution of k was discovered. Although for a skip capability of 5 the average size factor was roughly 0.5, the actual distribution of k's was found to be bimodal (Fig. 6). This means that a significant percentage (15+%) of wafers have $k \approx pN$ or, equivalently, a size reduction factor of ≈1. Since the system yield is really the number of wafers for which k exceeds the desired logical system size, this may be a significant factor in picking n.

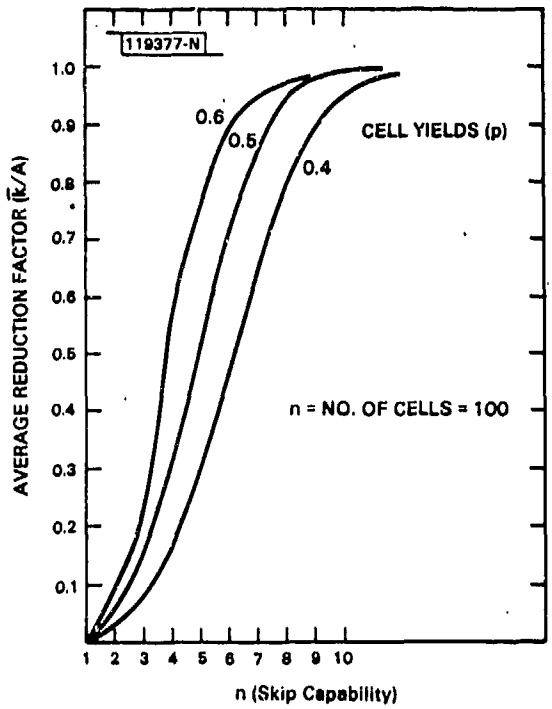


Fig. 5. Varying cell yield and maximum skip.

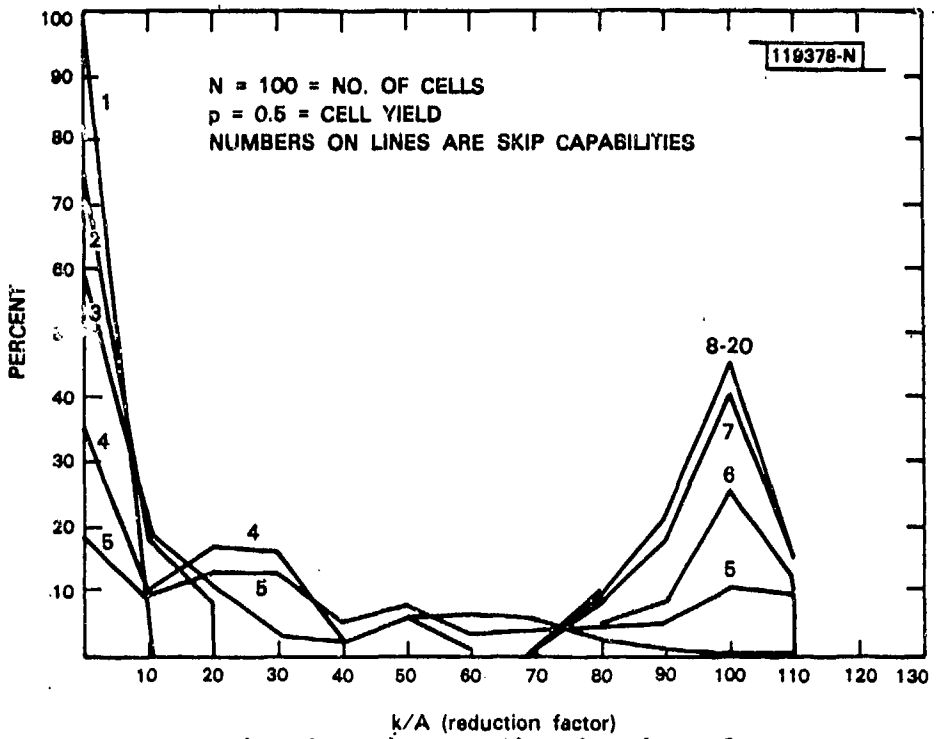


Fig. 6. Bimodal distribution of k.

IV. RVLSI TECHNOLOGY

A. RESTRUCTURABLE WEINBERGER ARRAYS

A Weinberger array can be considered a structured generalization of a PLA and is useful for implementing random logic. In the NMOS version of both, all gates are nor gates. A PLA contains two nor planes oriented at 90° to each other where outputs from the first plane become inputs to the second. The Weinberger array has no planes per se, rather all nor gates are oriented the same way, each gate running the width of the array, and outputs of some gates are connected to inputs of others by lines running lengthwise in the array (Fig. 7).

The Weinberger array has the advantage of allowing indefinite levels of gate logic, whereas the PLA allows only two. This gives the Weinberger array considerable flexibility in implementing a boolean expression, and therefore results in fewer transistors and usually less area than the equivalent PLA.

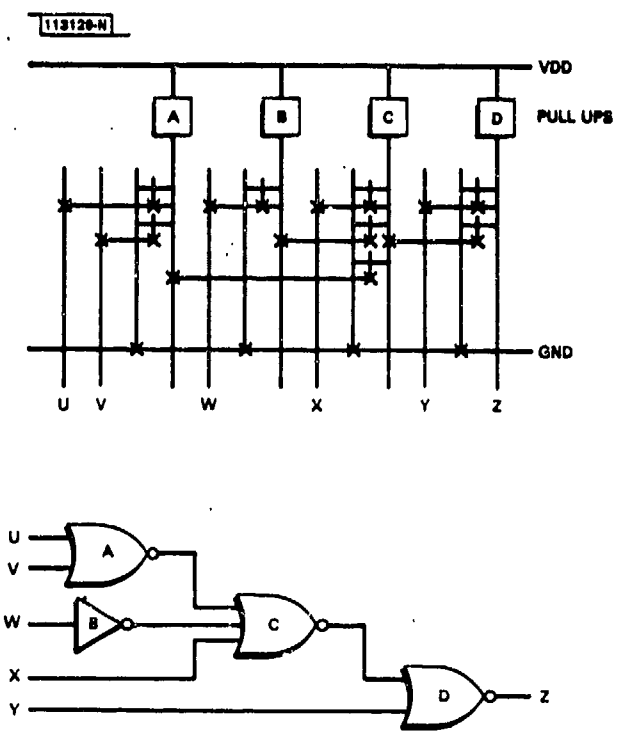


Fig. 7. Weinberger array.

Looked at another way, however, "levels of logic" are equivalent to "gate delays," so care must be taken in the proliferation of the number of levels. Actually, because the PLA requires inverters on the inputs to generate the true and complement levels for input signals, it really requires three gate delays.

The PLAs are adaptable to various types of programming; that is, because of their strict regularity, they can be personalized very much like ROMs. The programming can be done in the field by end users, at design time through automatic mask generation techniques, or by post-fabrication laser trimming. The latter technique is of interest because incorrectly programmed or faulty PLAs might be repaired through laser surgery. Also, if several-times-programmable links were provided, the testing of complex PLA structures could be simplified by selective removal and reactivation of logic terms in boolean expressions. It is also possible to consider a PLA-based master cell that could be replicated throughout a wafer. These could be personalized using a laser, providing post-fabrication design flexibility.

It is of interest to consider whether laser programmability can be extended to Weinberger arrays. Figure 8 depicts a scheme whereby this may be accomplished. The restructurable version is an array of cells, each of which consists of a pull-up, a pull-down, and several cut and weld points. A two-input nor gate could be formed out of the two left-hand cells by the following actions:

(1) Define inputs at points "A" and "B".
(2) Cut I/O lines at "C" to enable further utilization of the horizontal lines. If "A" and "B" inputs are to be used elsewhere, this would not be done. However, for this example the "A" line will be used for output.
(3) Cut at point "D". This isolates the adjacent pull-transistor, since this example uses the upper pull-up.
(4) Fuse link "E". This connects the horizontal line to the pull-up, making it the output line. Note that one end of each pull-down is already connected to the vertical line for proper operation.

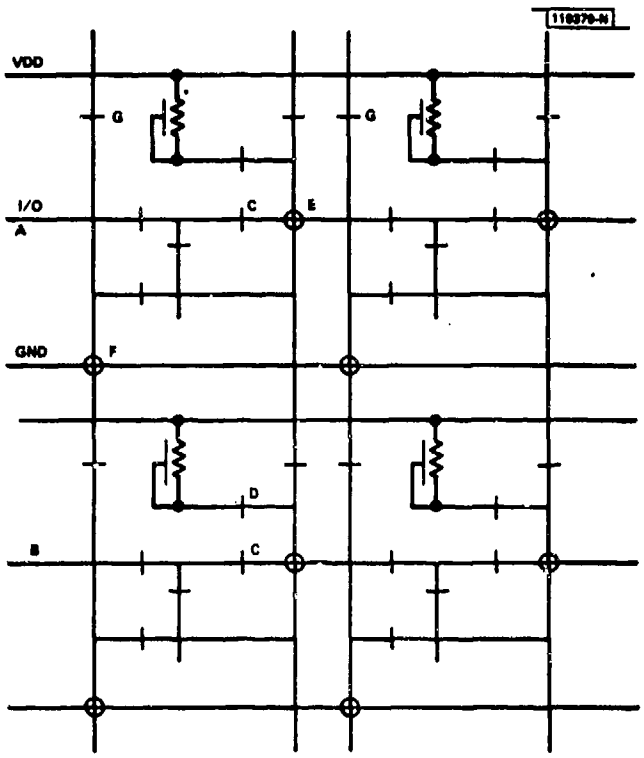


Fig. 8. Restructurable Weinberger array.

(5) Fuse link "F". This connects the other end of each pull-down to ground, as necessary.

(6) Finally, cut all points "G" to isolate the internal signal paths of this gate from interference with the rest of the array.

It remains to work through several practical examples to obtain a feel for the balance achieved between active area, interconnect overhead, and restructurability overhead. Further analytical studies may be useful to develop design guidelines for identifying situations where restructurability at this elemental level is possible.

B. SECOND-LEVEL METAL FOR MOSIS

The present standard MOSIS NMOS process does not offer a two-level metal capability. We are in the process of defining a mechanism for accepting wafers from MOSIS that have been processed through first-level metal, and

adding insulating material and second-level metal according to the designer's specification.

To evaluate the feasibility of this mixed mode of processing, a simple, metal-only experiment is in progress. A mask set comprising the appropriate set of test patterns was laid out on our Calma system. Using our Mann-CIF conversion software, the artwork was transferred to the VAX. The resulting CIF was then shipped to MOSIS via the ARPANET. At present, second-level masks and ten wafers have been received from MOSIS with first-level metal etched.

Four of the ten wafers were further processed using various combinations of second-level metal alloy and glint. The results are summarized in Table I. For 12 m vias all combinations were seen to be possible. However, for vias in the 6-8 m range, only the combination involving AlSiCu alloy on the second level and a hydrofluoric (HF) acid glint provided satisfactory performance. Fortunately, this is consistent with the standard MOSIS processing. It was also found that first- to second-level metal shorting through the polyimide was not a problem.

The remaining six wafers are currently being processed with the AlSiCu and HF glint to reconfirm these initial results.

C. LASER-PROGRAMMABLE LINKS FOR MOSIS

Two techniques are currently under investigation for providing the MOSIS community with laser-programmable links and requiring only a small amount of extra processing. Both approaches require only a single level of metalization. The approach is to use the laser beam to close a gap between adjacent metal runs (hence the term "lateral link"). Two types of gap insulators are being investigated, one exhibiting about 10 ohm resistance after laser forming and the other about 1K ohm. Initial results are quite encouraging and further experiments are under way.

TABLE I

MOSIS SECOND-LEVEL METAL EXPERIMENT
SUMMARY OF RESULTS FOR
MOSIS TEST WAFERS (After Sinter)

Wafer	2nd Level Metal	Glint	~Via Chain Yield (60 to 160 vias)
1	Al	Chromic-phosphoric	12 μm - 98% 8 μm - 30%
2	Al	Buffered HF	12 μm - 100% 8 μm - 50%
3	Al/Si/Cu	Chromic-phosphoric	12 μm - 100% 8 μm - 50%
4	Al/Si/Cu	Buffered HF	6, 8, 12 μm - 100% 3, 4 μm - 50%

Notes: 1. Al was used on first-level metal for all 4 wafers.

2. First- to second-level isolation yield for all 4 wafers is >95%.

V. TESTING

A. IN-PLANE SELF-TESTING

Linear next-neighbor systems are interesting candidates for fabrication time-incremental constructive testing or, equivalently, in-plane self-testing. Because the system being constructed need only be the test system, not the ultimate operational system, a next-neighbor test system topology generally does not restrict the logical operating design. It is preferable to avoid laser link zapping during testing as much as possible since there will be many tentative cell coupling and decouplings, and the fundamental once-programmable nature of laser links/zaps as well as the slow mechanical steps of the technique are ill-matched to this requirement. In addition, dependence on defect-free, wafer-long lines may not be a sufficiently conservative strategy for an initial cell test procedure. This makes the fixed segmentation statistics important, since they provide a measure of how much extra interconnect will be necessary to support the test scheme. Figure 9 shows the derivation of the number of channels required for a linear skip of n cells:

number of channels for maximum skip of n

$$= \sum_{1}^{n+1} i = \frac{(n+1)(n+2)}{2} = \frac{n^2 + 3n + 2}{2} \quad .$$

For a skip of 5, it is seen that:

$$\text{number of channels} = \frac{n^2 + 3n + 2}{2} = \frac{25 + 15 + 2}{2} = 21 \quad .$$

For a skip of 6, the number of channels must be 28. These channel requirements are too large for this method to be practical.

Further aspects of the test mechanism for linear next-neighbor topologies have been considered, and one more critical objection remains to

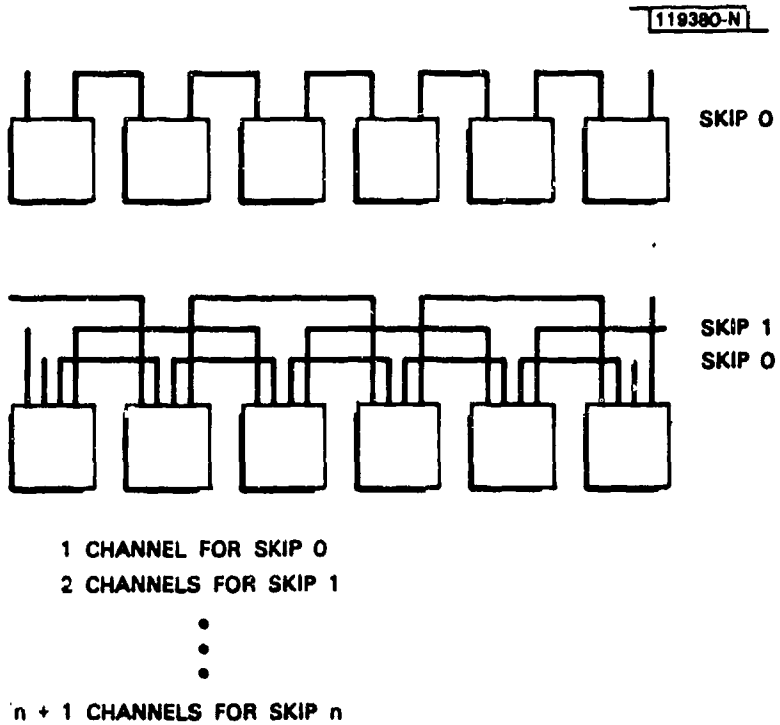


Fig. 9. Examples of channel requirements for linear skip strategy.

be overcome: in order to be tested a cell must be powered up. A cell failure resulting in a short from V_{dd} to ground would so severely impact the wh[illegible]afer that nothing else could be tested. This difficulty is avoided by the wafer-probe test method, since such cells are identified before they are connected to the wafer power lines. Some design techniques, such as the use of precharging instead of pull-up depletion mode transistors, might alleviate this problem.

B. OPTICAL PROBING

A scheme for applying optical probing techniques to NMOS has been proposed that should eliminate the parasitic effects of back-gate bias modulation on depletion loads. In this approach the laser beam is used to stimulate photo current in the drain and subsequently the source junctions of

the device under test. Laser power is adjusted to develop equal photo currents for each beam position. A combination of current measurements and some simple calculations yields the desired result. Experimental verification of this technique is now being attempted.

C. INTERCONNECT TESTING

A method for testing wafer-long interconnect lines prior to any restructuring was documented in the March 1981 Semiannual Report. Experience with actual wafers, however, shows that interconnect defects are liable to be created as lines are cut, links fused, etc. Therefore, a method has been developed for incremental testing of the interconnect concurrently with the laser restructuring process. This process provides vital feedback to the VAX assignment/linking software as laser zapping is underway. As a net* is constructed it is tested for internal continuity and short circuits to previously instantiated nets. The technique is completely independent of the logical connectivity requirements of the system.

The requirements for concurrent test and zap are automatic continuity testing hardware and two tracks per channel overhead. The procedure first involves building two nets of two combs each such that each net occupies one track in every channel (both horizontal and vertical). This method is compatible with the previously reported strategy for testing the interconnect before any restructuring has been done. One track per channel can be left behind on each side of the test combs as they are used during this test. Then the four combs can be joined to form two nets of two combs each, termed the "builder" and "accumulator," respectively.

Once the builder and accumulator have been created, linking of the system can begin. An example for a two-pin net is illustrated in Fig. 10.

*A net is a group of electrically connected track segments.

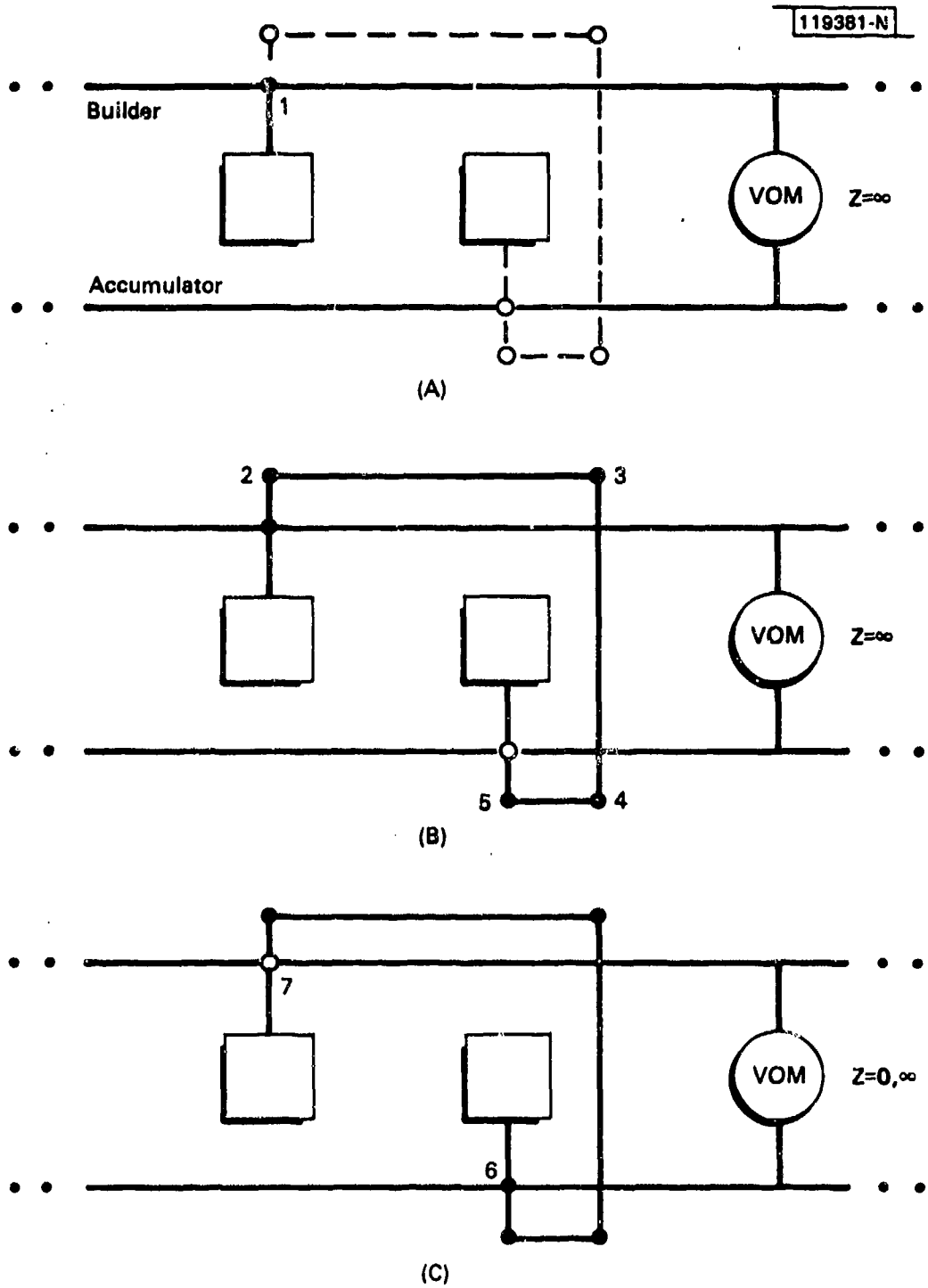


Fig. 10. Concurrent zap/test technique.

The net is constructed as follows:

(1) The first pin of the net is attached to the builder [Fig. 10(a), operation 1].

(2) Continuity between the builder and accumulator (z = 0 on the VOM) indicates a short circuit between the net and the accumulator.

(3) Each subsequent operation used to build the net is followed by a continuity check. If a short is detected, its location is known to be somewhere on the segment connected by the last link [Fig. 10(b), operations 2 through 5.

(4) After the net has been completed, without detection of a short, the pin at the other end is attached to the accumulator. Continuity via the net should now be observed between builder and accumulator [Fig. 10(c), operation 6].

(5) The net is then cut away from the builder, leaving it on the accumulator [Fig. 10(c), operation 7]. This is how each test in step 3 can detect shorting to all previously constructed nets.

(6) Repeat steps 1 through 5 for the remaining nets.

This technique can be extended to nets with more than two pins. An arbitrary pin is selected for connection to the builder. When the net is completed, some other arbitrary pin is connected to the accumulator. After checking for continuity, the builder is cut away from the net and reattached to some other pin on the net. This last step is repeated until all pins on the net have been checked after which the net is severed from the builder forever.

VI. APPLICATIONS

A. DIGITAL INTEGRATORS

The phase 1 integrator cell comprising four 10-bit buffered counters and data input and read-select shift registers has been fabricated and verified by successful operation of a number of die. All three wafers processed had over-etched poly gates that caused high leakage from the resulting short channel transistors. Of the two wafers with high, but not catastrophic leakage, yields of approximately 20% were obtained even though a blemish spoiled one of the three cells on the stepped reticle. With a clean reticle and elimination of the poly-etching problem, the desired cell yield of about 50% is not unreasonable. Simulation studies are underway to aid in designing the wafer-scale interconnect matrix assuming 200% cell redundancy.

B. SIGNAL PROCESSING STRUCTURES

A whole wafer FFT-16 has been designed to exploit restructurability for defect avoidance. The same wafer could be wired up to perform several other signal processing functions such as second-order sections, FIR filters, or correlation. Some minor extensions to the originally proposed multiply/add cell will greatly extend its versatility.

The current cell suffers from several limitations that restrict its versatility:

- It is not possible to use the adder if the multiplier is defective. As the multiplier is the largest single unit in the cell and therefore the most likely site of a defect, it is highly likely that there will be a significant number of cells on a wafer that are operational except for the multiplier. Thus, applications requiring more adders than multipliers cannot use the partially functioning cells.

- There is no way to store variable coefficients on the wafer. The current cell accepts only coefficients that are dynamically input or laser zapped. This prevents construction of, for instance, a FIR filter with changeable coefficients.
- All of the delays in the cell are fixed at 38 clock cycles (the transmit time of the cell, hereafter to be referred to as a "sync" delay). A number of algorithms require delays of one word length (a "word" delay) in addition to sync delays. This limitation forces a word length of 38 bits in a number of algorithms. In applications requiring a shorter word, this can reduce the throughput of the wafer to a fraction of that which the arithmetic portions are capable.
- The multiplier causes a greater than necessary propagation delay. The current multiplier design emphasizes simplicity of implementation rather than minimization of the propagation delay. In feed-forward applications, this delay increases the pipeline delay but does not reduce the throughput. In feed-back applications, the delay reduces the throughput to one word per sync delay. If the desired word size is less than the sync delay, the system throughput is reduced.
- There are an insufficient number of delay elements in each cell. A number of algorithms require more delay elements than are available in the cells in which the arithmetic operations are performed. The arithmetic portions of other cells must be wasted and interconnect must be used to provide the needed delays. Cells with defective arithmetic units may be recruited for their delays.

An augmented cell has been designed (Fig. 11) to correct some of the deficiencies of the original multiply-add cell used on the FFT wafer. This cell is identical to the original (at the block diagram level) with the following additions:

- A laser steerable multiplexer between the multiplier and the final adder allows isolation of the final adder when the multiplier is inoperable.

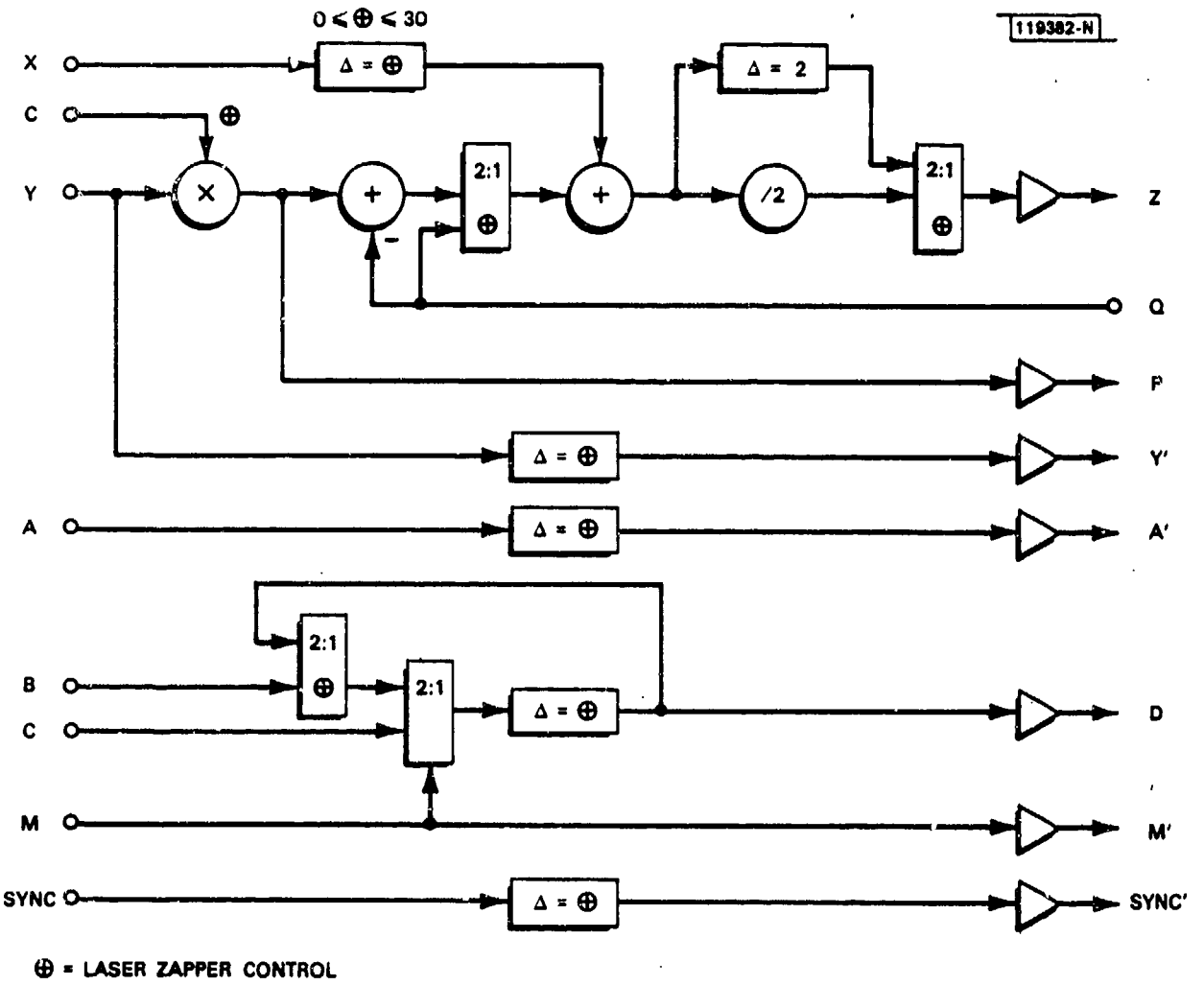


Fig. 11. Proposed extended MAD cell.

- An isolated delay line.
- An isolated delay line with two input multiplexers, one laser steerable and one electrically steerable. This delay line may be used to store a coefficient in recirculate mode and load the coefficient in nonrecirculate mode.
- All of the major delay lines have laser-programmable delays adjustable between one cycle and a sync delay.
- A possible addition not shown is laser control of the adder such that the adder could also be set to subtract. These changes require no new technology over that required to construct the FFT wafer, but will require an increased cell area (~30%) and may slow the maximum clock rate because of the capacitance of the lazer zapping pads. They will increase the versatility of the wafer and, with the increased cell capability, reduce the interconnect required to wire up some systems.

A wafer containing an array of the augmented cells will be capable of performing all of the same functions as the original wafers. The filter and correlator functions will now allow run-time loadable coefficients. The coefficient load scenario might be stop the operation, load the coefficients, and finally restart the operation. Word lengths will be controlled by the designer rather than forced by the hardware.

A new configuration, a series-parallel organization of the transversal (FIR) filter, has been found (Fig. 12) which provides a speed-up in throughput over a direct implementation. The current FFT wafer is expected to implement a direct form FIR filter at a 1 MHz word rate for a 16-bit word at the expected clock rate of 16 MHz. The I/O structure contains S/P and P/S converters, which have a word-serial, bit-parallel data format on the parallel side and a word-parallel, bit-serial data format on the serial side. These structures can be used to provide M-parallel serial-bit data streams to and from the filter structure where M is the desired speed-up factor. If the filter is N taps long (where N is an integral multiple of M), MN multiply-add

$$y_i = \sum_{j=0}^{N-1} x_{i-j} \; h_j$$

119405-N

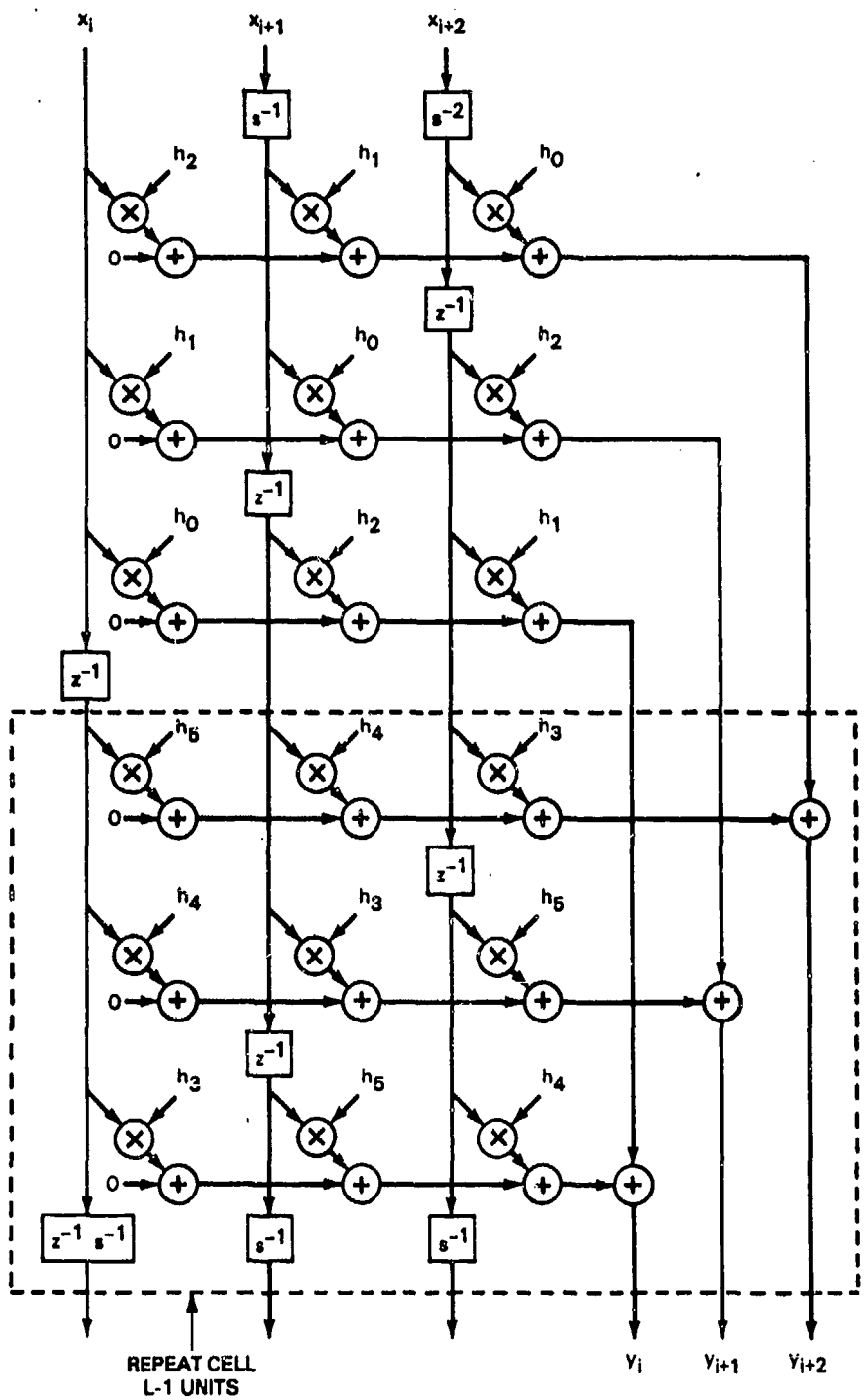


Fig. 12. Series-parallel filter implementation with MAD cells.

cells and N-M adder cells are required. The structure uses both word and sync delays. The repeat cell can be replicated as many times as required to achieve the desired filter length. The serial data streams (with the associated synchronization lines) can cross wafer boundaries to increase the available number of taps for large filters.

An assignment and linking strategy for this filter has been designed. This strategy assumes the cell array in linear, which can be achieved by a serpentine ordering of the cells in a two-dimensional pattern. The functional cells of each rectangular "super cell" of the filter are also placed in a linear array. The assignment strategy involves simply assigning the next functional cell to the next available cell on the wafer. This requires a maximum of 3M+5 interconnect channels for fixed (laser-zapped) coefficients and a maximum of 3M+10 interconnect channels for variable (run-time loadable) coefficients. It is expected that exploitation of the two-dimensional structure of the filter would significantly reduce the interconnect requirements for large M, but because estimation of the interconnect requirements is nontrivial, the linear layout is presented as an upper bound.

The cells on this wafer should be capable of implementing a number of signal-processing operations that require minimal amounts of control. For example, a useful algorithm composed of such operations is the Widrow-Hopf LMS adaptive filter. This algorithm may be implementable entirely upon one wafer if the filter length is moderate.

The concept of using a repeated cell on a large wafer represents a "gate-array" approach to signal-processing algorithm implementation. Its versatility is limited by the nature of the repeated cell. As a result, its position in the performance hierarchy at any time is bounded by state-of-the-art custom implementations at one end and microprocessors at the other. Microprocessors are more versatile, and a custom implementation can be at least as fast as the restructurable wafer system. The primary advantages of the restructurable wafer with a repeated cell result from the amenability of its systolic structure to VLSI implementation. The speed and cost of implementation for intermediate throughput applications look potentially attractive.

APPENDIX A

MACPITTS CODE FOR TEST CASES

```
(program addresser 8
 (def 23 power)
 (def 1 ground)
 (def 2 phia)
 (def 3 phib)
 (def 4 phic)
 (def 5 reset)
 (def addr0 register)
 (def mask0 register)
 (def active0 flag)
 (def myloc-addr0 constant 255)
 (def myloc-mask0 constant 254)
 (def myloc-active0 constant 253)
 (def data port input (6 7 8 9 10 11 12 13))
 (def addr port input (14 15 16 17 18 19 20 21))
 (def cs signal output 22)
 (always (cond ((= addr myloc-addr0)
                (setq addr0 data)
                (setq cs (not active0)))
               ((= addr myloc-mask0)
                (setq mask0 data)
                (setq cs (not active0)))
               ((= addr myloc-active0)
                (setq active0 (bit 0 data))
                (setq cs (not active0)))
               ((= addr0 (word-and addr mask0))
                (setq cs active0))
               (t (setq cs (not active0))))))
```

```
(program counter 4
 (def 16 power)
 (def 1 ground)
 (def 2 phia)
 (def 3 phib)
 (def 4 phic)
 (def 5 reset)
 (def count register)
 (def up signal input 6)
 (def load-zero signal input 7)
 (def limit port input (8 9 10 11))
 (def output port tri-state (12 13 14 15))
 (def zero constant 0)
 (always (cond ((or load-zero (and up (= count limit))) (setq count zero))
               (up (setq count (1+ count))))
         (setq output count)))
```

```
(program taxi 8
 (def 17 power)
 (def 1 ground)
 (def 2 phia)
 (def 3 phib)
 (def 4 phic)
 (def 5 reset)
 (def timer register)
 (def fare register)
 (def time-on signal input 6)
 (def hire signal input 7)
 (def mile-mark signal input 8)
 (def display port tri-state (9 10 11 12 13 14 15 16))
 (def maximum-time constant 100)
 (def base-fare constant 190)
 (def cost-per-mile constant 50)
 (def cost-per-time constant 10)
 (process time-clock 0
  off
      (cond (time-on (setq timer 0) (go on))
            (t (go off)))
  on
      (cond (time-on (cond ((= timer maximum-time)
                            (setq timer 0)
                            (signal charge-time))
                           (t (setq timer (1+ timer))))
                     (go on))
            (t (setq timer 0) (go off))))
 (process fare-clock 0
  for-hire
      (cond (hire (setq fare base-fare) (go hired))
            (t (go for-hire)))
  hired
      (par (cond ((not hire) (go for-hire))
                 ((and charge-time mile-mark)
                  (setq fare (+ (+ fare cost-per-mile) cost-per-time))
                  (go hired))
                 (charge-time
                  (setq fare (+ fare cost-per-time))
                  (go hired))
                 (mile-mark
                  (setq fare (+ fare cost-per-mile))
                  (go hired))
                 (t (go hired)))
           (setq display fare))))
```

```
(program toc 12
 (def tpa register)
 (def tbhs register)
 (def tehs register)
 (def bpa register)
 (def bbhs register)
 (def vector register)
 (def word register)
 (def transmit register)
 (def receive register)
 (def count register)

 (def error0 flag)
 (def error1 flag)
 (def character-received flag)
 (def transmitter-ready flag)
 (def swap flag)
 (def bank flag)

 (def 40 power)
 (def 1 ground)
 (def 2 phia)
 (def 3 phib)
 (def 4 phic)
 (def 5 reset)
 (def address port tri-state (6 7 8 9 10 11 12 13 14 15 16 17))
 (def data port i/o (18 19 20 21 22 23 24 25 26 27 28 29))
 (def read signal output 30)
 (def write signal output 31)
 (def cascade-in signal input 32)
 (def cascade-out signal output 33)
 (def serial signal i/o 34)
 (def drive signal output 35)
 (def dut0 signal i/o 36)
 (def dut1 signal i/o 37)
 (def dut2 signal i/o 38)
 (def dut3 signal i/o 39)

 (def bit1 constant 2)
 (def bit4 constant 16)
 (def bit7 constant 128)
 (def bit10 constant 1024)

 (always (setq receive transmit))

 (always (setq cascade-out (=0 count)))

 (always (cond ((eq vector 4 ( 2  1 0)) (setq dut0 f)))
         (cond ((eq vector 5 ( 2  1 0)) (setq dut0 t)))
         (cond ((eq vector 4 ( 5  4 3)) (setq dut1 f)))
         (cond ((eq vector 5 ( 5  4 3)) (setq dut1 t)))
         (cond ((eq vector 4 ( 8  7 6)) (setq dut2 f)))
         (cond ((eq vector 5 ( 8  7 6)) (setq dut2 t)))
         (cond ((eq vector 4 (11 10 9)) (setq dut3 f)))
         (cond ((eq vector 5 (11 10 9)) (setq dut3 t))))
```

```
(process tester 0
loop
  (setq address tpa)
  (par (setq address tpa) (setq read t))
  (par (setq address tpa)
       (setq read t)
       (setq vector data))
  (par (cond ((or (and (eq vector 0 (11 10 9)) dut3)
                  (and (eq vector 1 (11 10 9)) (not dut3)))
              (setq vector (word-or vector bit10))
              (setq error t)))
       (setq cycle-steal-ok t))
  (par (cond ((or (and (eq vector 0 ( 8  7 6)) dut2)
                  (and (eq vector 1 ( 8  7 6)) (not dut2)))
              (setq vector (word-or vector bit7))
              (setq error t)))
       (setq cycle-steal-ok t))
  (cond ((or (and (eq vector 0 ( 5  4 3)) dut1)
             (and (eq vector 1 ( 5  4 3)) (not dut1)))
         (setq vector (word-or vector bit4))
         (setq error t)))
  (cond ((or (and (eq vector 0 ( 2  1 0)) dut0)
             (and (eq vector 1 ( 2  1 0)) (not dut0)))
         (setq vector (word-or vector bit1))
         (setq error t)))
  (par (setq address tpa) (setq data vector))
  (par (setq address tpa) (setq data vector) (setq write t))
  (par (setq address tpa)
       (setq data vector)
       (cond ((= tpa tehs)
              (cond (swap (cond (bank (setq tpa 0) (setq bpa 2048))
                                (t    (setq bpa 0) (setq tpa 2048)))
                          (setq tbhs bbhs)
                          (setq tehs bpa)
                          (setq swap f))
                    (t    (setq tpa tbhs))))
             (t (setq tpa (1+ tpa))))
       (go loop)))
```

```
(process command 0
  (par (setq character-received f) (setq transmitter-ready t) (setq bank f))
wait-for-character
  (cond ((not character-received) (go wait-for-character))
        (t (setq character-received f)
           (cond ((eq receive "r") (go reset))
                 ((eq receive "v") (go vector))
                 ((eq receive "h") (go hold))
                 ((eq receive "g") (go go))
                 ((eq receive "x") (go examine))
                 ((octal receive) (go octal))
                 (t (go wait-for-character)))))
reset
  (par (cond (bank (setq error0 f) (setq bpa 0))
             (t    (setq error1 f) (setq bpa 2048)))
       (setq count 3)
       (setq word 0)
       (go wait-for-character))
vector
  (cond ((not cycle-steal-ok) (go vector))
        (t (setq address bpa) (setq data word)))
  (par (setq address bpa) (setq data word) (setq write t))
  (par (setq address bpa)
       (setq data word)
       (setq count 3)
       (setq word 0)
       (setq bpa (1+ bpa))
       (go wait-for-character))
hold
  (cond ((not cycle-steal-ok) (go hold))
        (t (setq address bpa) (setq data word) (setq bhbs bpa)))
  (par (setq address bpa) (setq data word) (setq write t))
  (par (setq address bpa)
       (setq data word)
       (setq count 3)
       (setq word 0)
       (setq bpa (1+ bpa))
       (go wait-for-character))
go
  (par (setq swap t) (setq count 3))
wait-for-swap
  (cond ((or swap (not transmitter-ready) (not cascade-in))
         (go wait-for-swap))
        (t (cond ((if bank error0 error1)
                  (setq transmit "y"))
                 (t (setq transmit "n")))
           (setq transmitter-ready f)))
wait-for-completion-of-query
  (cond ((not transmitter-ready) (go wait-for-completion-of-query))
        (t (setq count 0) (setq word 0) (go wait-for-character)))
examine
  (cond ((not cycle-steal-ok) (go examine))
        (t (setq address bpa)))
  (par (setq address bpa) (setq read t))
  (par (setq address bpa)
       (setq read t)
       (setq word data)
       (setq bpa (1+ bpa))
       (setq count 3))
wait-for-cascade
  (cond ((or (not cascade-in) (not transmitter-ready)) (go wait-for-cascade))
        (t (setq transmit (+ "0" (word-and word 8)))
           (setq word (3>> word))
```

```
            (setq transmitter-ready f)
            (cond ((not (= count 1))
                   (setq count (1- count)) (go wait-for-cascade))
                  (t (go wait-for-completion-of-query)))))
octal
  (par (cond ((and cascade-in (<>0 count))
              (setq word (word-or (<<3 word) (word-and receive 8)))
              (setq count (1- count))))
       (go wait-for-character))))
```

APPENDIX B

PAL 82 FRAMEWORK FUNCTIONS

APPENDIX B

PAL 82 FRAMEWORK FUNCTIONS

I. ASSERTIONS

A. CONSTRUCTOR ASSERTIONS

1. Logical

net (net[<pin-name>|(<instance-name><pin-name>)]*)

The net function defines a logical net. The pin names must be of the i^{th} or i-1st level.

structure

(structure <structure-name> (<component-desc>*) (<pin-name>*)
(<net>*))

The first <structure-name> is the type name of the structure being defined. The following <component-desc> list is composed of type and instance names of the substructures used in this level. The <pin-name>s are the names of the pins of the defined structure, although the <net>s may reference either these pins or pins of the substructures. A cell (i.e., a bottom-level structure) is defined by a null substructures list and null net list.

remove

(remove<net>)

The remove command is primarily for use in the test-assign-link-test iteration loop. <net>s that have been instantiated to fixed cells (by the input map) cannot be reassigned. The <net> items must be of the format (<full-instance-name> <pin-name>), where <full-instance-name> of a cell is described below.

make-top-level

(make-top-level<structure-name>)

This allows a designer to use a precanned set of structures and designate which sub-tree is to be used.

make-cell

(make-cell<structure-name>)

This allows a designer to use a precanned set of structures in which the trees may run deeper than the actual cell level. A designer could use the make-cell/cancel-cell functions to experiment with the effects of larger or smaller cells.

2. Physical

track

(track<point><point>

This returns an item that defines a track from point to point. A point is a two-tuple :(xy).

link (link<point><link type>)

This returns an item that defines a link of a specific type at a point. The link-type definition will be described later.

pin (pin<name><point>)

This returns an item that defines a pin with a name at a point.

The functions above define primitive items that can be grouped together, named, and put into the database.

track-group

(track-group<name><track-expr>)

This creates a type definition named <name> that is described <form>.

<track-expr> is an expression that evaluates to a list of track, track-width, and contain items. Contain items are described later.

link-group

(link-group<name><link-expr>)

This groups together link definitions. It is similar to track-group.

cell-type

(cell-type<name><pin-expr>)

This groups together pin definitions. The logical and physical descriptions may eventually be unified.

cell-group

(cell-group<name><cell-type-expr>)

This groups together cell-types. The cell-types should be declared first. It would be inconvenient to have to digitize every track, link, and

pin over the entire wafer. The contain function allows groups to be used in definitions of other groups.

contain

(contain<previous-group-name><point>)

Previously defined groups can be put at a specific point in the definition of the current group. <point> in this case defines where the (0,0) of the sub-group is to be offset in the new group. Declaring types or groups does not describe the wafer. To do this, a copy of the type or group must be put "onto" the wafer. That copy is called an instance.

put-tracks,put-links,put-cell,put-cells

(put-tracks<type-name><instance-name><point>)

This function puts an instance of a track group. The other puts commands are similar in nature.

bad-cell

(bad-cell<physical-cell-name>)

bad-segment

(bad-segment<seg>)

II. QUERIES

A. LOGICAL

cells?

(cells?)

The cells? query returns the list of the <full-instance-name> of cells involved in the logical description. The <full-instance-name> of a cell utilizes the <instance-name>s of all the structures above it. The last structure command is implicitly the system logical description.

net?(net?<full-instance-name><pin-name>)

This returns a list of the <net> form (as described in remove). The <net> returned contains the query pin name in it.

nets?

(nets?<full-instance-name>) This returns a list of all <net>s touching the input cell. Queries about the physical world can refer to information that requires only connectivity, requires an order relationship, or requires actual physical relations.

B. PHYSICAL CONNECTIVITY

connectable?

(connectable?<pin>|<seg>)

Returns a list of <seg>s that are connectable to the <pin> or <seg> by one link. The format of a <seg> is not described here since <seg> paths are probably started from <pins>.

allocated?

(allocated?<seg>)

Returns "t" if the segment is allocated.

C. PHYSICAL ORDER

connectable?

(connectable?<pin>|<seg>[<range>])

Returns an ordered list of <seg>s connectable to <pin> or <seg> within the <range>. The <range> is composed of two <seg>s of which the first is < the second by the ordering criterion.

Physical Relationship

connectable?

(connectable?<pin>|<seg>[<range>])

Returns as other connectables?s. The <range> in this case consists of physical coordinates.

loc? (loc?<seg>|<pin>)

Returns <range>(physical range) of item.

seg?(seg?<range>)

Returns a list of <seg>s within the physical <range>.

III. ACTIONS

connect

(connect<seg><seg>)

(sim-connect<seg><seg>)

Link the <seg>s.

disconnect

(disconnect<seg><seg>)

(sim-disconnect<seg><seg>)

Inverse of connect command.

allocate

(allocate<seg>[<range>])

(sim-allocate<seg>[<range>])

The segment is allocated. If range is included (either order range or physical range), the segment is also cut.

IV. DEFINITIONS

Some undefined terms are used liberally.

<piu-name> <structure-name> <name> <previous-group-name> <instance-name> <type-name> Names are any character string. Each name may have to fulfill certain requirements, such as being associated with some defined data-type, but this should be clear from the function descriptions.

<component-desc>:= (<structure-name><instance-name>)

The <structure-name> must have appeared in a previous structure command. The <instance-name> must be unique in the current structure command. See I.A.1. structure.

<full-instance-name>

This is the name of a cell that fully describes it. A cell is generally invoked in structures which are invoked in higher structures, and so on. All <instance-name>s of the structures above the cell are included in the <full-instance-name>.

<point>

A point is an x,y coordinate pair that is in terms of actual physical coordinates (i.e., numbers).

<seg>

The format for <seg> is now undefined because <seg> formats will be generated from the connectable? function.

<pin>

<pin>:= (<instance-name><pin-name>)

The <instance-name> is specified in a put-cell or put-cells function. The <pin-name> comes from a pin function <name> which has been collected into the cell-type function whose name is used for the put-cell or put-cells function.

<range>

Range means different things for different functions and modes. A physical range of a <seg> consists of two <points>, the first of which is the second. The physical range of a <pin> consist of a single <point>.

APPENDIX C

LASER CONTROLLER COMMANDS

APPENDIX C

LASER CONTROLLER COMMANDS

COMMANDS

E <i>, <j><pin name>[<track>]
Moves the current and selected positions to the point identified as the pin name pin on cell i, j. The track is only specified if the table is moving from a different channel.

H <channel><track>
Moves the current and selected positions to the point whose vertical coordinate is the specified track, and whose horizontal coordinate is the same as the current position.

V <channel><track>
Moves the current and selected positions to the point whose horizontal coordinate is the specified track, and whose vertical coordinate is the same as the current position.

. Moves the current position back to the selected position.

@ Causes a zap at the current position.

< Cuts a first level metal vertical track.

> Cuts a second level metal vertical track.

! Cuts a first level metal horizontal track.

^ Cuts a second level metal horizontal track.

M <x>, <y>
Moves the current position to the selected position, offset by x and y, in microns.

m <x>, <y>
Moves the current position to the current position, offset by x and y, in microns.

A <x>, <y>
Moves the current and selected positions to the absolute coordinate, x, y, in microns.

D <message>\n*
Must be terminated by a \n. The message is printed on the VAX terminal.

---

* /n is the location for new line.

s An alignment command. It tells the routines that translate from microns into table steps that the selected point on the wafer is really the current position. The first two times this command is used, only the translational offset is adjusted. The third and subsequent times cause the full linear transformation, from microns into table steps, to be recalculated.

a Like the "s" command, but only updates the translational offset.

h Jogs the current position 1.27 microns (1 table step) to the left.

j Jogs the current position 1.27 microns (1 table step) downward.

k Jogs the current position 1.27 microns (1 table step) upward.

l Jogs the current position 1.27 microns (1 table step) to the right.

r <index>
Sets the index, in microns, for the following four commands.

y Indexes the current position to the left.

u Indexes the current position downward.

i Indexes the current position upward.

o Indexes the current position to the right.

f <mode>
Sets the mode of operation of the intermediate level driver. This can do local mechanical optimization for automatic linking. See the intermediate level driver description for more detail.

F <distance>
Sets the radius of the circle, in microns, in which the laser table does not need to be refocused.

x <file name>
Input from the current stream is suspended, the named file is opened, and pushed onto the input stream stack. It is much like a subroutine call to the file; when the end of the file is reached, the file is closed, the stream is popped off the stack, and input continues from the calling stream.

P The next on-stack stream is closed and removed, and the stack compresses to fill the space. When a stream is interrupted and will not be resumed, this command aborts it. An example would be to abort an automatic linking.

rub-out
: Interrupts a stream, for example, to realign the wafer. When the currently executing command finishes, a new version of the standard input stream (the terminal) is pushed onto the input stream stack. The user can then type whatever commands is desired.

q
: Just like an end of file. It causes the current stream to be popped and the next on-stack stream to be resumed. If the next on-stack stream was interrupted, as opposed to having issued an "x" command, then the current and selected positions are automatically restored to what they were at the time of the interrupt. This may cause table movement, if necessary.

VAX-TO-APPLE SERIAL INTERFACE

The Apple side of the interface is simple. Currently, only three commands are used:

Mx,y\n
: Causes the Apple to index the table stepper motors by x and y steps.

F\n
: The Apple will move the z-axis of the table to focus the video image from the microscope.

Z\n
: The Apple will activate the laser shutter to cause a "zap."

The communications sequence for each command is as follows:

(1) The Apple sends a "G\n", indicating that it is in its tight read loop and can receive characters at 1200 baud.

(2) The VAX sends one of the above commands.

(3) The Apple sends another "G\n."

(4) The VAX repeats the command.

(5) The Apple checks that the two command strings are identical.
If they are, it executes the appropriate function and, when it is finished, transmits the acknowledgement, "OK\n." This process continues at step 1.
If they are not identical, the Apple transmits the message "BAD\n"; transmission of the command restarts at step 1.

LOW-LEVEL DRIVER FOR THE VAX

At the lowest level on the VAX there are a set of functions, written in C, that correspond exactly to the commands that the Apple can execute:

index-by(x,y) long x,y;
Corresponds to Mx,y.

focus()
Corresponds to F.

zap()
Corresponds to Z.

These functions return after the acknowledgement is received from the Apple.

INTERMEDIATE-LEVEL DRIVER FOR THE VAX

Written on top of the low level routines, is a set of functions that provide a linear transformation from a virtual coordinate system, typically in microns, to table steps. They allow initialization and maintenance of the alignment of the coordinate system. Also provided are several modes of operation to meet the needs of both automatic and manual operation. The intermediate functions include:

set-position (x,y mode) double x,y; int mode;
Identifies the current table position with the virtual coordinates x and y according to mode:

If mode is 0, the coordinates become one of the last three remembered, with the oldest one thrown away. Then, if at least three coordinates have been given, the coefficients for the linear transformation are recomputed. The first two times the routine is called, the coordinates are remembered, but only the offset coefficients are computed.

If mode is 1, the coordinates are not remembered, and the offset coefficients are recomputed. This allows easy recovery from events that cause translational misalignment.

move-to(x,y) double x,y;
This causes the table to move to the virtual position specified. Depending on the operational mode, this may also cause a focus operation.

zap()
This is like its counterpart in the low-level driver, except that depending on the operational mode, it may cause the physical movement to the last specified coordinates, possibly followed by a focus operation, before the zap.

set-focus-mode (mode) int mode;

This modifies the operational mode to accomodate both automatic and manual control. The mode is a superposition of three independent bits. If all are zero, focusing will never occur. If the 1 bit is set, focusing will be performed every time the table moves outside the circle, of radius focus distance, about the point where focusing last occurred. If the 2 bit is set, focusing will occur when the above condition is met, but only before a zap. If the 4 bit is set, all physical motion is suppressed until just before a zap. This allows local mechanical optimization suitable for automatic operation.

set-focus-distance (distance) long distance;

This sets the previously mentioned focusing radius.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-~~81-153~~ 82-059	2. GOVT ACCESSION NO. AD-A120 471	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Restructurable VLSI Program		5. TYPE OF REPORT & PERIOD COVERED Semiannual Technical Summary 1 October 1981 — 31 March 1982 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Peter E. Blankenship		8. CONTRACT OR GRANT NUMBER*(s)* F19628-80-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173-0073		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order 3797 Program Element No. 61101E Project No. 2D30
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE 31 March 1982 13. NUMBER OF PAGES 76
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Electronic Systems Division Hanscom AFB, MA 01731		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

None

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

VLSI	customization	routing
Restructurable VLSI (RVLSI)	hardware description language	systolic array
programmable interconnect	placement	integrator
defect avoidance		

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

This report describes work on the Restructurable VLSI Research Program sponsored by the Information Processing Techniques Office of the Defense Advanced Research Projects Agency during the semiannual period 1 October 1981 through 31 March 1982.

DD FORM 1 Jan 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*