AD-A267 866

COMPUTER-AIDED DESIGN PACKAGE FOR DESIGNERS OF DIGITAL OPTICAL COMPUTERS

Miles Murdocca, David Berger, Apostolos Gerasoulis, Vipul Gupta, Saul Levy, Chun Liew, Masoud Majidi, Donald Smith, and Thomas Stone

Final Report for Grant #N00014-90-J-4018
Period Covered: 5/1/90 – 4/30/93

Miles Murdocca, Principal Investigator
Department of Computer Science
Rutgers University, Hill Center
New Brunswick, NJ 08903
(908) 932-2654
murdocca@cs.rutgers.edu

DTIC
ELECTE
AUG 13 1993
S B D

July, 1993

TABLE OF CONTENTS

1. Summary 1
2. Design Tools and Algorithms for Digital Optical Circuit Design 2
3. Fault Avoidance 21
4. Architectural Implications of Reconfigurable Optical Interconnects 24
5. Investigation and Development of Optical Interconnects 30
6. References 35
7. Publications and Presentations 36
8. Patent Disclosures 39
9. Technology Transfer 39
10. Plans for Future Work 41
11. Conclusion 42

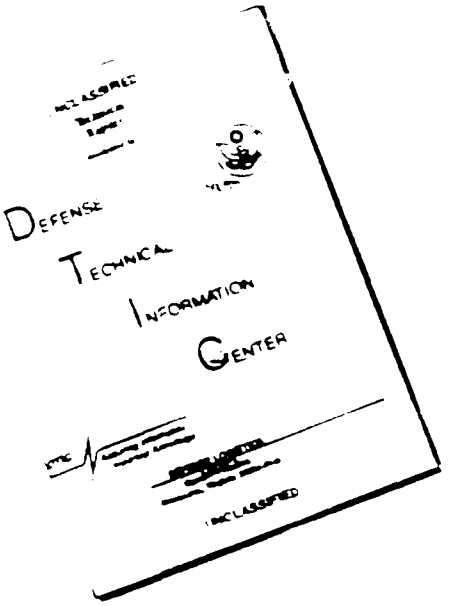
1. SUMMARY

This final report details progress made during a three-year project that is jointly supported by AFOSR and ONR, administered as a grant to Rutgers University through ONR, covering the period 5/1/90 – 4/30/93. The primary goals of the effort are to create methods for designing digital optical computers, and to investigate aspects of optical interconnection that influence the design process. The emphasis is on the Bell Labs style of architecture, in which arrays of optical logic gates are interconnected in free space with regular patterns at the gate-level. Other approaches are also explored that make use of hybrid technology (**smart pixels**), irregular interconnects, and reconfigurable interconnects.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

93-18737

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Major accomplishments for the project include: (1) the development of novel automated layout methods for regularly interconnected optical circuits; (2) the development of an interactive design tool and supporting methods for designing circuits interactively using both regular and irregular interconnects; (3) the creation of two methods of bypassing faults in optical device arrays; (4) the discovery and characterization of **functional locality** in ordinary computer programs, which is applied to hardware caching; (5) the creation of a trade-off study involving optical interconnects and architectural complexity; and (6) the development of a novel method of optical spot array generation and interconnection.

2. DESIGN TOOLS AND ALGORITHMS FOR DIGITAL OPTICAL CIRCUIT DESIGN

The effort began with the anticipation that all-optical computing, in which the gate-level switching devices have optical inputs and outputs, would become a practical approach. The goal was to use photonics at all levels in a computer, from the gate level up to the system level. During the course of the effort, we learned that an all-optical approach may be practical in a laboratory experiment (which we are in the process of demonstrating, see Technology Transfer in Section 9). For a full-scale system, however, an all-optical approach appears to introduce a cost and a performance limitation that is much greater than the potentially small additional cost of using an opto-electronic approach. We have maintained an all-optical computing model throughout the effort, however, for two major reasons: (1) the circuit layout problems are similar for both all-optical and opto-electronic approaches; and (2) a gate-level optical approach may be appropriate when all of the logic gates need to be individually addressed. Gate-level optical computing is important for reconfigurable component applications, such as in the area of **rapid prototyping**, and in the novel area of **hardware caching** which we describe in Section 4.

2.1. The Model

A simplified form of the optical computing model supported by the effort [1] is shown in Figure 1, which consists of arrays of optical logic gates interconnected in free space. The more general form of the model includes greater complexity in the logic arrays, such as electronic processing elements with optical I/O ports (such as smart pixels). For the simplified form, binary 1's and 0's are represented as intensities of light beams. The interconnects have a regular pattern such as a perfect shuffle or a crossover [2]. This type of interconnect is used experimentally in a number of AT&T SEED based optical processor testbeds, and also in an S-SEED system under development at Optivision, a microlaser based system at Siemens, and an S-SEED system that we are jointly developing with Air Force personnel in the Photonics Center at Rome Laboratory (see Section 9). The reason for using regular interconnects at the gate level is to allow the beams to share the same field of a single lens. If we choose to use irregular interconnects instead, then each channel requires a separate imaging system. This can be achieved through holography or through micro-optics techniques, although the small diameters of the resulting lenses limit propagation distance [3] due to diffractive coupling between neighboring channels. The two extreme cases that are characterized by trade-offs between regular and irregular interconnects do not necessarily exclude one another. We have performed an extensive analysis of this issue (see Section 5) and have found that a combination of the regular and irregular approaches, which are loosely representative of space-invariant and

DTIC QUALITY INSPECTED 3

Dist | Avail and/or Special
A-1

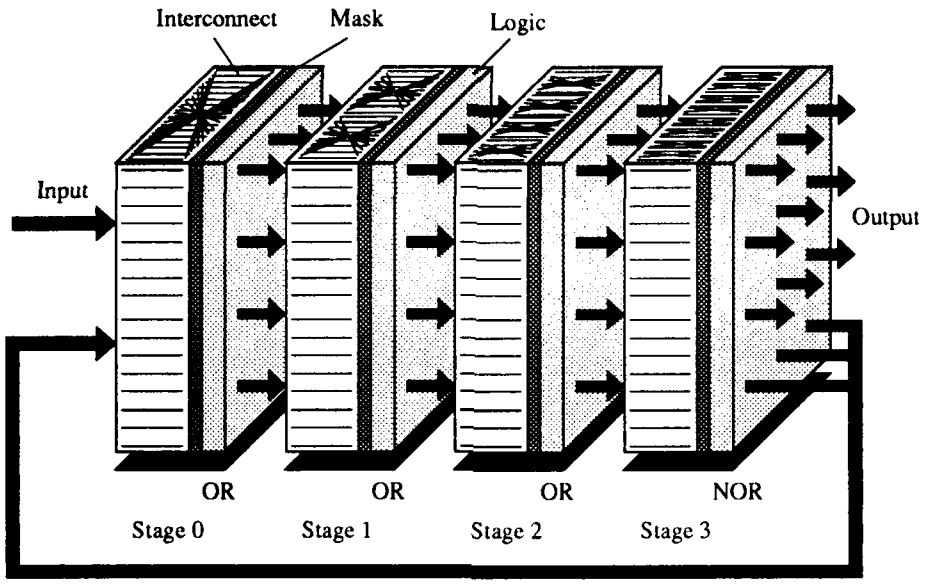


Figure 1: *The optical computing model.*

space-variant approaches, respectively, may support dense packing of devices while balancing the limitations imposed by practical realizations.

Our efforts on layout algorithms have focused primarily on the more restrictive regular approach, however, since the boundary between regular and irregular interconnects is not clear. A practical system may contain a mixture of regular and irregular interconnects, in which case the regular approach is appropriate for the regular interconnects, and conventional electronic layout schemes may be suitable for the irregular interconnects.

The optical computing model shown in Figure 1 is composed of alternating arrays of optical logic gates and free-space interconnects. Masks in the image planes block light at selected locations and customize the interconnects to perform specific logic functions. There is no need to use masks if the light sources are independently controlled, such as through electrically driven microlasers, or if a beam-steering approach is used, which is addressed in Section 4. The system is fed back onto itself and an input channel and an output channel are provided. Feedback is imaged with a vertical shift so that data spiral through the system, allowing a different section of each mask to be used on each pass. Optical signals travel orthogonal to the device substrates.

2.2. Programmable Logic Array (PLA) Generation Software

Medium scale integration (MSI) components serve as the lowest level building blocks of conventional digital computers, other than the discrete logic gates of which they are composed. Examples of digital circuits of MSI complexity are decoders, multiplexers, arithmetic/logic units (ALUs), and programmable logic arrays (PLAs). We have developed algorithms for designing MSI circuits that map onto the model shown in Figure 1. The algorithms translate low level behavioral descriptions of simple circuits specified in the form of truth tables into optical gate layouts of MSI complexity. The software we created that implements the algorithms includes special capabilities that allow the

Inputs				Outputs			
x	x'	y	y'	$x'y'$	$x'y$	xy'	xy
0	1	0	0	1	0	0	0
0	1	0	0	0	1	0	0
1	0	1	1	0	0	1	0
1	0	1	1	0	0	0	1

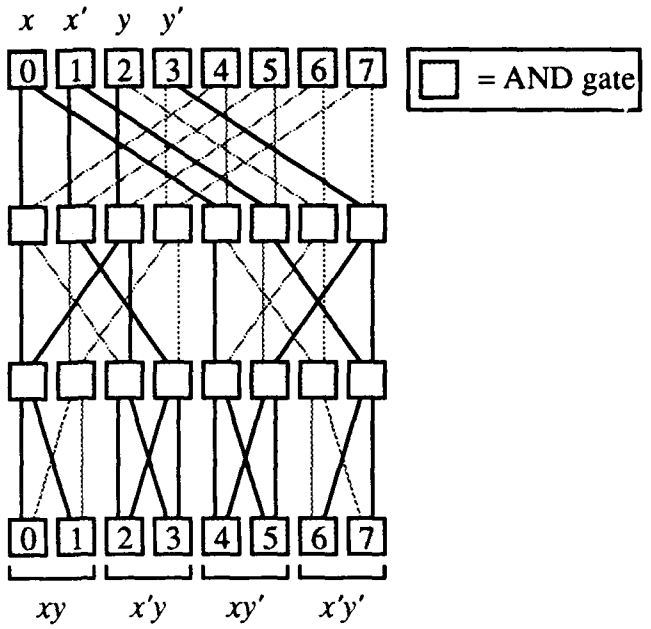


Figure 2: *AND stage of programmable logic array (PLA) generated by the decoder tool.*

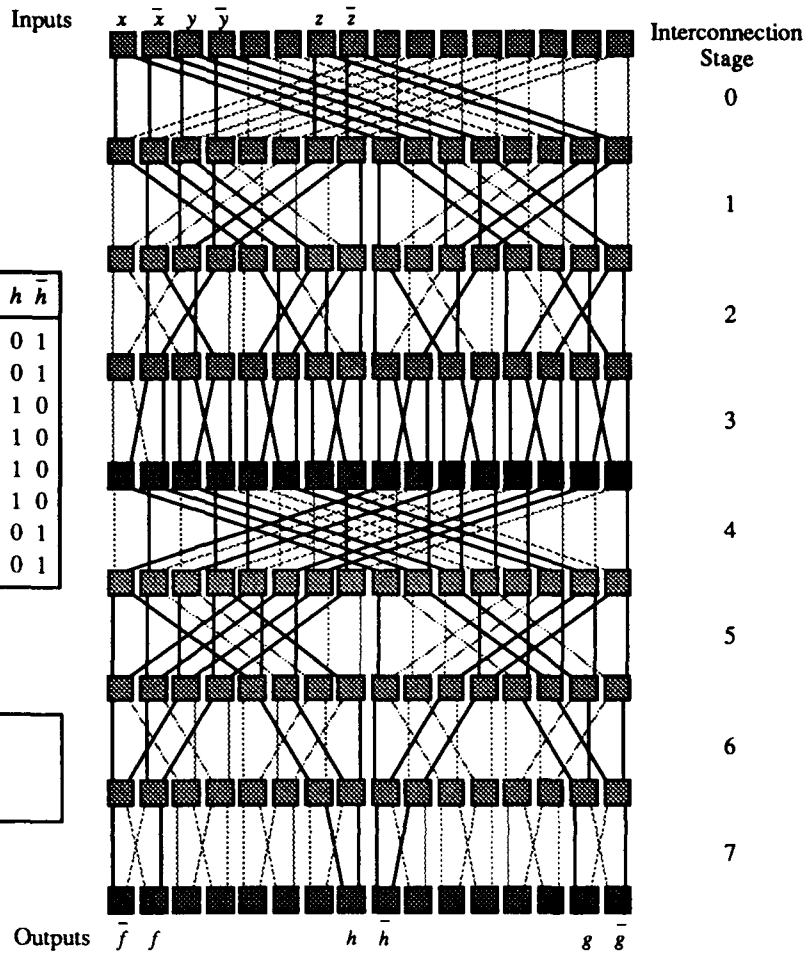
computer designer to specify permutations of inputs and outputs, which helps tile subcircuits into complex configurations.

A sample layout created by the **decoder tool** is shown in Figure 2. The shaded boxes represent optical logic gates. Inputs enter the circuit in the top row and outputs are generated in the bottom row. The variables x, y, and their complements x' and y' are the input variables. Logically ANDed combinations of variables (called **minterms**) are produced at the bottom of the circuit. Solid lines represent optical paths that are enabled, and lightly shaded lines indicate disabled connections that correspond to either opaque areas of fixed masks, or correspond to some other method of beam-blocking or beam-steering. The decoder tool deals with both functional behavior and physical layout simultaneously. These two areas are normally decoupled in digital electronic designs, but here, we need to consider them together.

The interconnection pattern shown in Figure 2 is a banyan, but perfect shuffle and crossover interconnection patterns are also supported. Mappings to other topologically equivalent interconnects are not difficult to automate, but mappings to interconnects that are not topologically equivalent must be handled in some other way, such as interactively, which is addressed in Section 2.3.

A decoder translates a logical encoding into a spatial location. In Figure 2, the binary encodings on the input lines x, x', y, and y' determine which of the output minterms xy, $x'y$, xy', or $x'y'$ will be enabled (that is, set to logical 1). Decoders are used in the addressing structures of random access memories (RAMs) and in more complex MSI circuits such as PLAs.

A PLA consists of a customizable AND matrix (a simple decoder here) followed by a customizable OR matrix. For an electronic implementation, a programmable fuse is placed at each crosspoint in



= OR gate
= NOR gate

x	$\bar{x}$	y	$\bar{y}$	z	$\bar{z}$	f	$\bar{f}$	g	$\bar{g}$	h	$\bar{h}$
0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	1	1	0	1	0	0	1	0	1
0	1	1	0	0	1	1	0	0	1	1	0
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	0	1	1	0	0	1	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	0	1	1	0	0	1
1	0	1	0	1	0	1	0	1	0	0	1

$$f(x,y,z) = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$$
$$g(x,y,z) = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$$
$$h(x,y,z) = \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z$$

Figure 3: *Circuit layout for three functions and their complements generated by the PLA tool.*

the AND and OR matrices. The matrices are then customized for specific functions by disabling fuses. For an optical implementation, a method of beam-blocking or beam-steering can be used.

PLAs are a workhorse component that are used throughout digital electronics. An advantage of using PLAs in electronics is that there are only a few inputs and outputs (pins), while there is a large number of logic gates between the inputs and outputs. Since free-space optics supports parallel access to the optical logic devices, the pin argument does not carry over to the optical domain. However, PLAs are still very important because they can efficiently implement low-level functions.

The **PLA tool** makes use of the decoder tool in mapping a set of functions specified as truth tables into a layout. Consider the truth table shown in Figure 3, which describes the functions $f(x,y,z)$, $g(x,y,z)$, $h(x,y,z)$ and their explicitly generated complements. Function f is the addition of x, y, and the carry-in z, function g is the carry generated in computing f, and h is the exclusive-OR (XOR) of x and y. Figure 3 also shows a layout of optical logic gates that implements the functions f, g, and h using a banyan interconnection topology. The PLA tool generates the layout automatically from a data file description of the truth table. Stages 0 – 3 represent logic gates in the top rows of stages

0 – 3 in the model shown in Figure 1. Stages 4 – 7 would then be mapped onto the second rows of stages 0 – 3 in the model shown in Figure 1. The model shows a crossover interconnect, but the PLA uses a banyan. Mappings among these interconnects and the perfect shuffle are trivial, and the software handles all three interconnects.

The fact that we can automatically generate a layout for any three functions and their complements (a total of six functions) for an arbitrarily sized PLA is an important accomplishment, especially considering that the depth is near-optimal regardless of the interconnection pattern [4], but a problem with strictly following the optical computing model shown in Figure 1 is that the connections are determined by the complexity of the optics, rather than by the complexity of the circuit design. By maintaining strict regularity at the gate level, the only flexibility left to the designer is in choosing which connections to disable, and where to place the inputs and outputs. We have managed the circuit depth issue reasonable well, and even the circuit breadth is good in terms of complexity theory [4], but overall gate count is high, typically two to eight times greater than if a completely irregular interconnect is supported.

Although we have created algorithms and software to automatically generate layouts for MSI circuits, good algorithms for generating general higher level circuits do not exist. There are several circuit design situations we have encountered for which the only algorithms we can envision employ an exhaustive search of all possible permutations of enabled and disabled connections. This is a prohibitively time-consuming approach. For situations such as this, in which the essence of a good design cannot be captured by an algorithm, a better approach is to allow an expert to create a design interactively. Although complete automation may not be practical for these larger circuits, partial automation turns out to be practical, which we describe in the next section.

2.3. Interactive Design Software

We created an X-windows Optical Programmable Interactive Design tool (XOPID) that uses the X graphical interface. The XOPID tool allows logic gates to have fan-ins and fan-outs that vary, and allows circuits to have irregular interconnections between gates and between higher level structures such as PLAs. These features allow us to study the trade-offs involved when fan-in/fan-out values higher than two are used and when connections are not constrained to being topologically equivalent to the perfect shuffle, banyan, or crossover. Other issues we have studied with XOPID include functional decomposition and PLA tiling.

In order to manage the complexity of circuit design for regular interconnects, the interactive design tool makes use of a **collision detection** strategy that guides the design process so that signals do not collide as a result of using common paths, which is a situation commonly referred to as **blocking** in switching applications. Collision detection is a significant issue here because the physical circuit layouts and their functional behaviors are tightly coupled. The theory behind the method is described in Ref. [5].

In more detail, XOPID is a menu-driven tool that allows the user to draw and manipulate digital circuits interactively in an X window. The user interface to XOPID is shown in Figure 4. Five

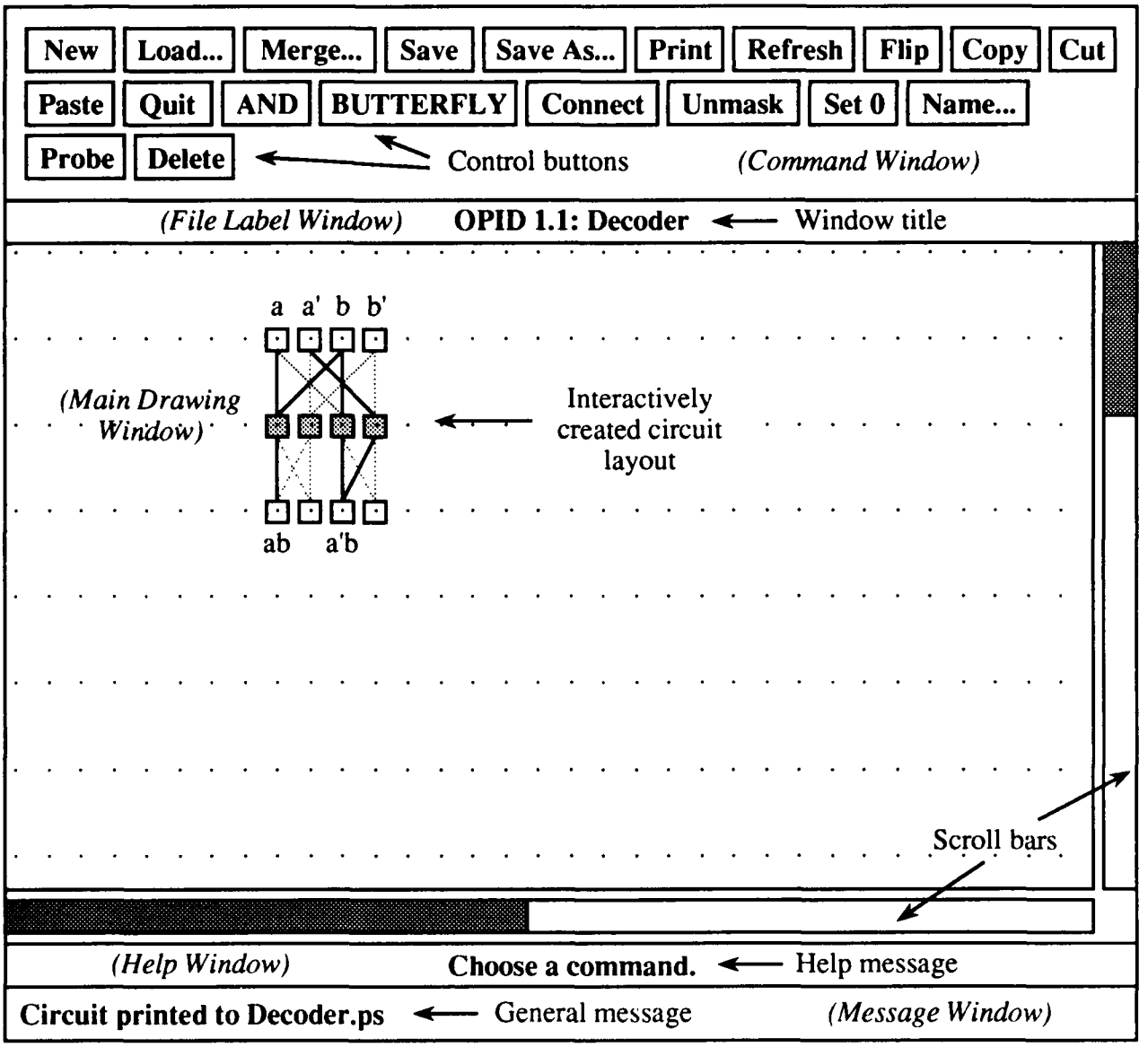


Figure 4: *The user interface to XOPID. Shaded horizontal and vertical bars serve dual functions as scrollbars and as indicators of the virtual drawing area.*

vertically stacked windows comprise the display area: the **command window**, the **file-label window**, the **main drawing window**, the **help window**, and the **message window**. The command window contains control buttons that the user selects for different circuit manipulation operations. When a button is selected, it is highlighted and a brief message describing its function is displayed in the help window. The main drawing window displays the circuit that is being designed. The virtual drawing area is larger than the main drawing window, which displays a portion of the virtual drawing area. The main drawing window can be moved over the virtual drawing area by using the scrollbars, which also indicate the relative sizes of the main drawing window and the virtual drawing area. If the execution of a user command results in an error or some other exceptional behavior, a message

is displayed in the message window. The file-label window displays the name of the circuit being manipulated.

A synopsis of the functions available to the user is given below. Any command that ends with an ellipsis (...) indicates that a dialog box appears that prompts the user for more information. A complete description of XOPID is given in the user manual [6].

NEW Clears the current circuit.

LOAD... Prompts the user to specify a `.cir` file (a circuit file stored in XOPID format, with the filename extension '`.cir`'). The circuit described in this file then becomes the current circuit. If the specified file does not exist, an empty circuit becomes the current circuit.

MERGE... Prompts the user to specify a `.cir` file. The circuit in this file is merged into the current circuit at a position that the user selects with the mouse. The merge operation fails if a circuit overlap situation exists.

SAVE Saves the current circuit in the file named by extending the filename displayed in the file-label window with a `.cir` extension.

SAVE AS... Prompts the user to specify a `.cir` file. The current circuit is then saved in this file.

PRINT Prints the current circuit in PostScript format to the file named by extending the filename displayed in the file-label window with a `.ps` extension.

REFRESH Redraws the circuit on the bitmap that is displayed in the main drawing window.

FLIP Waits for the user to specify a rectangular region by depressing the left mouse button on the upper left corner of the region, dragging the pointer to the lower right corner of the region and then releasing the button. A copy is made of the sub-circuit corresponding to the user-specified rectangular region, which is flipped along a vertical axis passing through the center of the region and stored in a file named `.Clipboard.cir`, from where it can be pasted using the PASTE option. This operation is useful in building crossover circuits from smaller crossover circuits, which are symmetric about a vertical axis.

COPY Similar to FLIP except that the sub-circuit is not flipped before it is stored in `.Clipboard.cir`.

CUT Similar to COPY but deletes the sub-circuit corresponding to the user-specified region from the current circuit.

PASTE Waits for the user to specify a position with the mouse, which is where the upper left corner of the circuit stored in `.Clipboard.cir` is merged into the current circuit, providing the operation does not result in an overlap.

QUIT Exit from XOPID, discarding the current circuit.

OR/NOR/AND/NAND Waits for the user to specify a rectangular region (as described in FLIP) and fills the rectangular region with logic gates of the type displayed in the help window. If a gate already exists in the region, its type is changed to that displayed in the help window. The user can toggle through the gate types by repeatedly selecting this command button.

BUTTERFLY/SHUFFLE/CROSSOVER Waits for the user to specify a rectangular region (as described in FLIP) and inserts connections corresponding to the current interconnection pattern between gates in this region. The user can toggle through the interconnection patterns by repeatedly selecting this command button. Note: the terms "butterfly" and "banyan" are used interchangeably here.

CONNECT/DISCONNECT Waits for the user to depress the left mouse button over a gate, drag the pointer till it is over another gate, and then release the button. If the CONNECT option is active, a new connection is made between an output of the first gate and an input of the second gate if one does not already exist. If the DISCONNECT option is active, the existing connection, if any, between an output of the first gate and an input of the second is removed. The operation is performed between successive rows only. The active option is displayed in the help-window. The user toggles between the two options by selecting this command button.

MASK/UNMASK Waits for the user to specify two gates (as described in CONNECT/DISCONNECT). If the UNMASK option is active, a path of connections, if one exists, leading from the output of the first gate to the input of the second gate is found and all connections on this path are unmasked (that is, connections are enabled). If the MASK option is active, all connections on the path are masked (disabled). The active option is displayed in the help window. The user toggles between the two options by selecting this command button.

SET 0/SET 1/UNSET Waits for the user to select a gate. If the SET 0 option is active, the output of the selected gate is set to 0. If the SET 1 option is active, the output of the selected gate is set to 1. If the UNSET option is active, any Boolean value to which the output of the selected gate had been tied is removed. The active option is displayed in the help window. The user toggles between the options by selecting this command button.

NAME... Prompts the user to specify a name for a gate and waits for the user to select a gate. The output signal of the selected gate is then given the specified name. If a name is not specified, and if the output of the gate already has a name, that name is removed.

a	b	c	x_0	$\overline{x_0}$	x_1	$\overline{x_1}$	x_2	$\overline{x_2}$	x_3	$\overline{x_3}$	x_4	$\overline{x_4}$
0	0	1	0	1	1	0	0	1	1	0	1	0
0	0	0	0	1	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1	0	0	1
0	1	0	0	1	1	0	1	0	1	0	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	0	0	1	0	1	0	1	0	1	1	0
1	1	1	1	0	1	0	1	0	0	1	0	1
1	1	0	1	0	1	0	1	0	0	1	1	0

Functions		Operations
$x_0 = ab$	$\overline{x_0} = \overline{ab}$	AND/NAND
$x_1 = \overline{a} + b$	$\overline{x_1} = \overline{\overline{a} + b}$	OR/NOR
$x_2 = b$	$\overline{x_2} = \overline{b}$	Buffer/NOT
$x_3 = \overline{a}$	$\overline{x_3} = a$	NOT/Buffer
$x_4 = \overline{ab} + ac$	$\overline{x_4} = \overline{a}b + a\overline{c}$	XNOR/XOR

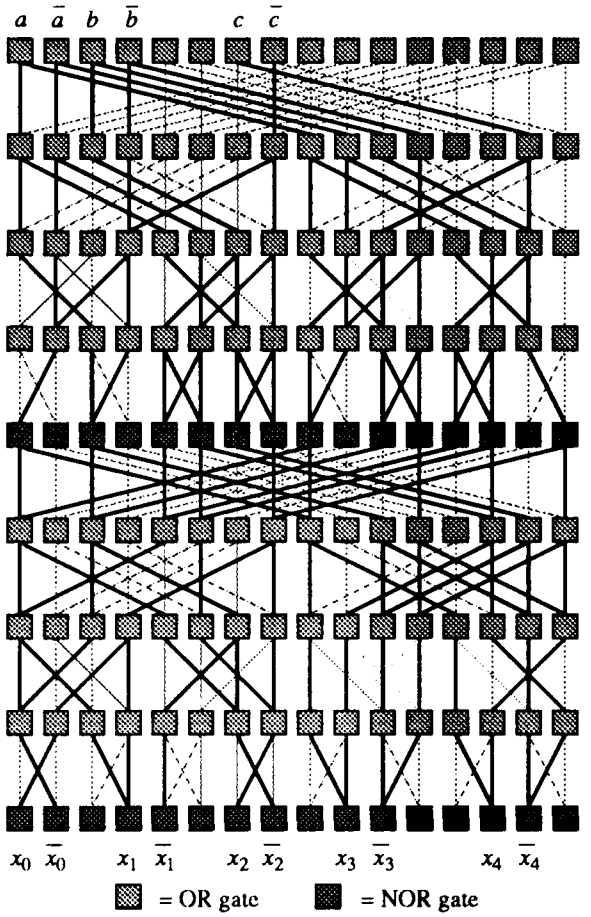


Figure 5: *A truth table description of 10 functions, and the corresponding manually designed PLA.*

PROBE Waits for the user to select a gate. The output value generated at the gate and the Boolean expression representing the gate's output are displayed in the message window.

DELETE Waits for the user to specify a rectangular region as described in FLIP. All gates that lie within this region are deleted from the current circuit as well as all connections that are incident on any gate in the region.

We have used XOPID for designing PLAs. Although we have fully automated the PLA tool for this task, we have found that more sophisticated PLA designs can be created interactively. The circuit layout shown in Figure 5 implements 10 functions (five functions and their explicit complements). The PLA tool guarantees a layout for any three functions and their complements (a total of six functions), in a 16×8 area for three variables, but the PLA tool is not capable of implementing five functions and their complements in the same area. The solution shown in Figure 5 was obtained manually. The approach used in designing this circuit was to start first by automatically generating the top half of the circuit using the PLA tool, which generates all of the minterms for the input variables, and then working from the bottom of the circuit (where the x_i are) up to the minterms. This

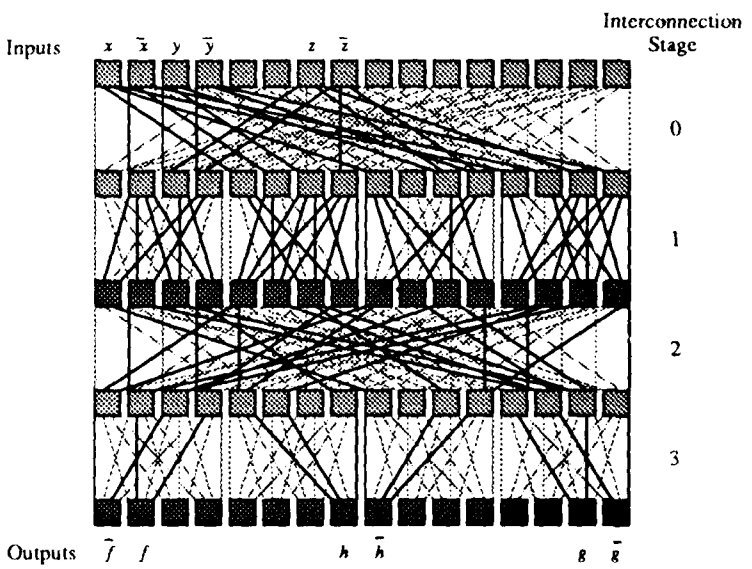
is a rule of thumb approach that works well in practice. We have not found a good method of automating the approach because some permutation is needed in repositioning the minterms (a permutation tool is described in Section 2.6). The significance of this example is that it shows that partial automation, coupled with a good designer, can produce better designs in this domain than can be achieved with full automation.

2.4. Increasing Fan-In and Fan-Out

A potential pitfall with using regular interconnects at the gate level is that a high gate count is incurred, and the circuits are deeper than their electronic counterparts. Circuit depth and gate count can be improved, however, by increasing fan-in and fan-out.

Consider again the circuit shown in Figure 3. The circuit is 16 logic gates wide by 8 levels deep, giving a total gate count of $16 \times 8 = 128$ logic gates. If we increase the fan-in and fan-out from 2 to 4, then the depth decreases from 8 to 4, which results in a circuit that has only half the latency of the original circuit and a gate count of $16 \times 4 = 64$ logic gates, as shown in Figure 6. The circuit is derived from the circuit shown in Figure 3 by collapsing interconnection stages 0 and 1 from Figure 3 into Stage 0 of Figure 6, and correspondingly stages 2 and 3 into stage 1, stages 4 and 5 into stage 2, and stages 6 and 7 into stage 3.

In theory, any number of Boolean functions can be implemented in just two stages, given arbitrary fan-in and fan-out. In practice, fan-in and fan-out are limited to small numbers, especially for high speed logic, regardless of the technology. The fact that most of our examples are given for fan-ins and fan-outs of two is only due to our sensitivity to the limitations of logic devices we have been working with, such as S-SEEDs, and is in no way an indication of a limitation to our design methods.



= OR gate
= NOR gate

x	$\bar{x}$	y	$\bar{y}$	z	$\bar{z}$	f	$\bar{f}$	g	$\bar{g}$	h	$\bar{h}$
0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	1	1	0	1	0	0	1	0	1
0	1	1	0	0	1	1	0	0	1	1	0
0	1	1	0	1	0	0	1	1	0	1	0
1	0	0	1	0	1	1	0	0	1	1	0
1	0	0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	0	1	1	0	0	1
1	0	1	0	1	0	1	0	1	0	0	1

$$f(x,y,z) = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$$
$$g(x,y,z) = \bar{x}yz + x\bar{y}z + xy\bar{z} + xyz$$
$$h(x,y,z) = \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z$$

Figure 6: *An alternative implementation of the circuit shown in Figure 3, using a fan-in and fan-out of 4.*

The circuit shown in Figure 6 shows that an extension of the design techniques to higher fan-ins and fan-outs is trivial, and in fact, fan-ins and fan-outs can be unevenly matched while still obtaining a significantly improved circuit. A 75% reduction in circuit depth and gate count would be obtained for a fan-in of 2 and a fan-out of 4, for instance, rather than the 50% reduction shown in Figure 6 for a fan-in and fan-out of 4.

ALU truth table

x	y	a	b	c	z	z'	s	s'
0	0	0	0	0	0	1	0	1
0	0	0	0	1	1	0	0	1
0	0	0	1	0	1	0	0	1
0	0	0	1	1	0	1	1	0
0	0	1	0	0	1	0	0	1
0	0	1	0	1	0	1	1	0
0	0	1	1	0	0	1	1	0
0	0	1	1	1	1	0	1	0
0	1	0	0	0	0	1	0	1
0	1	0	0	1	0	1	0	1
0	1	0	1	0	1	0	0	1
0	1	0	1	1	1	0	0	1
0	1	1	0	0	1	0	0	1
0	1	1	0	1	1	0	0	1
0	1	1	1	0	1	0	0	1
0	1	1	1	1	1	0	0	1
1	0	0	0	0	0	1	0	1
1	0	0	0	1	0	1	0	1
1	0	0	1	0	1	0	0	1
1	0	0	1	1	1	0	0	1
1	0	1	0	0	1	0	0	1
1	0	1	0	1	1	0	0	1
1	0	1	1	0	0	1	0	1
1	0	1	1	1	0	1	0	1
1	1	0	0	0	0	1	0	1
1	1	0	0	1	0	1	0	1
1	1	0	1	0	0	1	0	1
1	1	0	1	1	0	1	0	1
1	1	1	0	0	0	1	0	1
1	1	1	0	1	0	1	0	1
1	1	1	1	0	1	0	0	1
1	1	1	1	1	1	0	0	1

(a)

ALU functions

a	b	z	s
0	0	$x + y$	Carry
0	1	a OR y	0
1	0	a XOR y	0
1	1	a AND y	0

(b)

Decomposition of ALU

$z(x, y, a, b, c) =$
(00 (001)) +
(01 (000 + 010 + 011 + 100 + 101)) +
(10 (000 + 010 + 011 + 100 + 101)) +
(11 (001 + 010 + 011 + 110 + 111))
↑↑ ↑↑↑
xy abc

$z'(x, y, a, b, c) =$
(00 (000 + 010 + 011 + 100 + 101 + 110 + 111)) +
(01 (001 + 110 + 111)) +
(10 (001 + 110 + 111)) +
(11 (000 + 100 + 101))

$s(x, y, a, b, c)$ +
(00 ()) +
(01 (001)) +
(10 (001)) +
(11 (000 + 001))

$s'(x, y, a, b, c) =$
(00 (000 + 001 + 010 + 011 + 100 + 101 + 110 + 111)) +
(01 (000 + 010 + 011 + 100 + 101 + 110 + 111)) +
(10 (000 + 010 + 011 + 100 + 101 + 110 + 111)) +
(11 (010 + 011 + 100 + 101 + 110 + 111))

(c)

Figure 7: *Truth table descriptions (a) and (b) and functional decomposition (c) of an ALU.*

2.5. MSI-Level Interconnection

We have had success in fully automating the layout of PLAs using strictly $\log_2 N$ interconnects such as the banyan, but we have not had a similar success in automatically interconnecting PLAs. Design at this higher level is as important as PLA design. Even for an opto-electronic smart pixel approach, in which the PLAs are implemented electronically, we still have to deal with interconnecting the smart pixels optically, possibly again using perfect shuffles, banyans, or crossovers if regular interconnects are used. We encountered this problem early on, because we found that automatic PLA generation was effective for only a few input variables. For six or more variables, we needed to develop a decomposition strategy in order to reduce the overall circuit size.

In Figures 7a and 7b, a truth table represents two functions z and s and their complements z' and s' in terms of five variables x, y, a, b, and c. We can use the PLA tool to generate a corresponding circuit, which will be 64 logic gates wide and 12 levels deep, giving an area cost of $64 \times 12 = 768$. By

Factor out constants	**Assign function names to innermost parenthesizations**
$z(x, y, a, b, c) =$	$z(x, y, a, b, c) =$
(00 (001)) +	(00 g0) +
(01 (000 + 010 + 011 + 100 + 101)) +	(01 g1) +
(10 (000 + 010 + 011 + 100 + 101)) +	(10 g1) +
(11 (001 + 010 + 011 + 110 + 111))	(11 g2)
$z'(x, y, a, b, c) =$	$z'(x, y, a, b, c) =$
(00 (000 + 010 + 011 + 100 + 101 + 110 + 111)) +	(00 g3) +
(01 (001 + 110 + 111)) +	(01 g4) +
(10 (001 + 110 + 111)) +	(10 g4) +
(11 (000 + 100 + 101))	(11 g5)
$s(x, y, a, b, c)$ +	$s(x, y, a, b, c)$ +
(01 (001)) +	(01 g0) +
(10 (001)) +	(10 g0) +
(11 (000 + 001))	(11 g6)
$s'(x, y, a, b, c) =$	$s'(x, y, a, b, c) =$
(00) +	(00) +
(01 (000 + 010 + 011 + 100 + 101 + 110 + 111)) +	(01 g3) +
(10 (000 + 010 + 011 + 100 + 101 + 110 + 111)) +	(10 g3) +
(11 (010 + 011 + 100 + 101 + 110 + 111))	(11 g7)
(a)	(b)

Figure 8: *Factorization of constant terms (a) and assignment of labels to innermost parenthesization groups (b).*

Assign function names to next level of parenthesization

$z(x, y, a, b, c) =$
h0 +
h1 +
h2 +
h3

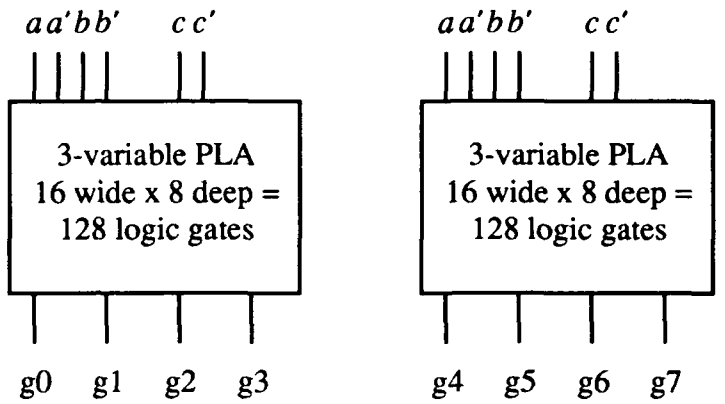
$z'(x, y, a, b, c) =$
h4 +
h5 +
h6 +
h7

$s(x, y, a, b, c) +$
h8 +
h9 +
h10

$s'(x, y, a, b, c) =$
h11 +
h12 +
h13 +
h14

(a)

Create PLAs for innermost parenthsization level



(b)

Figure 9: *Assignment of labels for outermost level (a) and PLA organization (b).*

decomposing the circuit into smaller PLAs, we can reduce the area cost to 512, although circuit depth will increase to 16 as we will see.

The functions z, z', s, and s' are factored (decomposed) so that variables x, x', y, and y' appear at the outer level, and variables a, a', b, b', c, and c' appear at the inner level as shown in Figure 7c. In Figure 8, constants (where a, b, c, and their complements have the same value for every combination) are factored out as 0's or 1's, which reduces the sizes of the equations. The innermost parenthesized groups are then assigned names. Note that there are 14 innermost parenthesized groups for a, b, c, and their complements, but only eight labels (g0 – g7) are needed to represent the 14 groups. This is because some of the groups, such as g0 = 001, are shared among more than one function.

Labels are assigned to the outermost parenthesized groups in Figure 9a, and the external view of the PLAs that implement g0 – g7 is shown in Figure 9b. In Figure 10, The PLA/MUX layout is shown for the decomposed ALU (a MUX is a variation of a PLA). The outputs of the PLAs were sorted by hand into the order shown in order to allow a simple interconnect (essentially the finest level of a

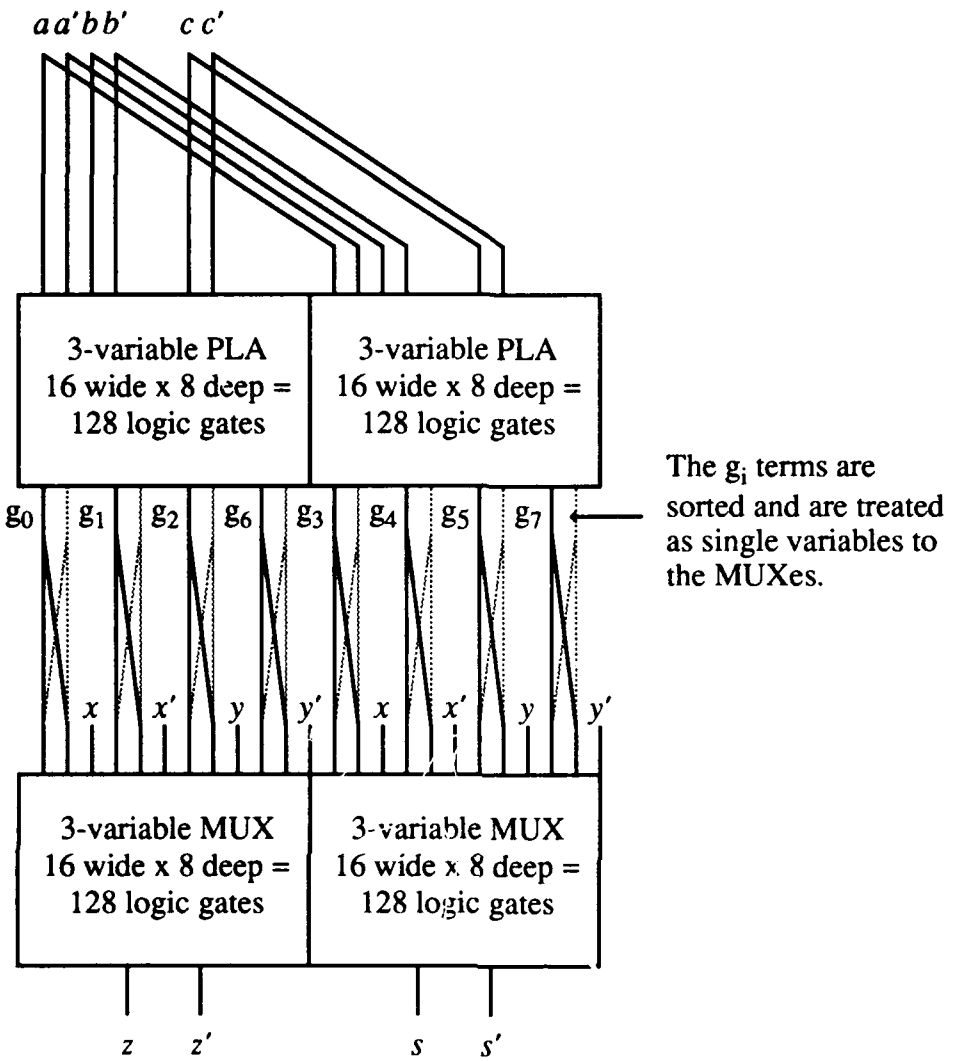


Figure 10: *Layout for decomposed ALU.*

banyan) to interconnect the PLAs. We used a time constrained search technique to find the best parenthesization, which we have found works well in practice.

We know from permutation theory that an arbitrary permutation of N items can be obtained in $3\log_2 N - 1$ levels of perfect shuffle interconnected bypass/exchange switches. The width of the circuit shown in Figure 10 is 32 logic gates, which would require a multistage interconnection network (MIN) of depth $3\log_2(32) - 1 = 14$ levels between the PLA and MUX stages, rather than the single stage that we achieved here. An advantage of the 14 level approach is that there exist good algorithms for configuring the networks. One level of interconnection is good, but 14 levels of interconnection is much too deep to be practical. We can reduce the depth by constraining PLA outputs to equally spaced positions (every fourth gate, for example) which has the effect of reducing N in proportion to the spacing. Although this helps, it does not help enough. A conventional electronics approach would require only a single level of interconnection, and in fact, we did manage

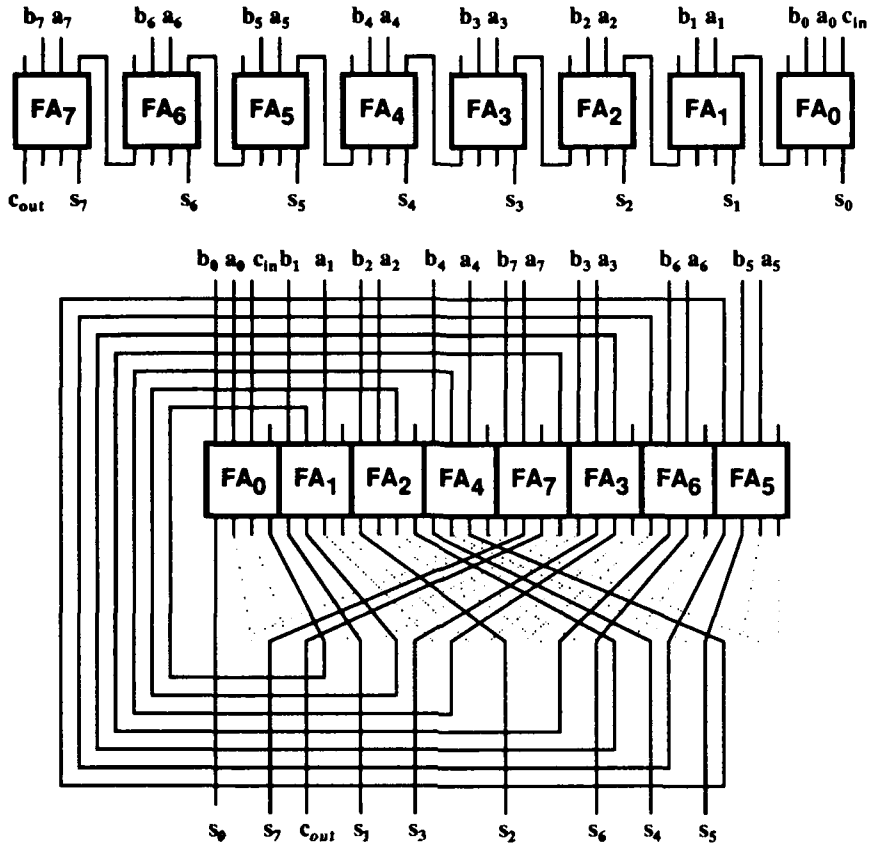


Figure 11: *A single perfect shuffle stage connects full adder elements of an eight-bit ripple-carry adder.*

a single level between the PLA and MUX components in Figure 10, but it required the help of designer.

Although we need $3\log_2 N - 1$ levels of shuffle/exchange stages to achieve an arbitrary permutation of N items, we are not trying to achieve *every* permutation: we just need one permutation that works. As a general approach to the problem, we looked at ways to interconnect PLAs using one or two levels of a perfect shuffle interconnect, with the added flexibility of choosing a tiling as to where the PLAs fall.

In Figure 11, a mapping is shown of an eight-bit ripple-carry adder onto a single perfect shuffle stage, using one-bit full adder (FA) PLAs. The abstract layout as a digital electronic designer might represent it is shown at the top of the diagram. The problem is to map the connections between FAs into a *single* perfect shuffle. The perfect shuffle layout at the bottom of the diagram was obtained by feeding the connectivity pattern of the abstract layout into a program we developed at Rutgers. This example is significant because it shows that the MSI interconnects for at least one circuit can be forced into a perfect shuffle structure without adding depth to the circuit. An implementation using a completely unconstrained interconnection topology would be just as deep, so we have incurred no depth penalty here. We continued our investigation and found that many types of regular structures map easily onto a single stage of the perfect shuffle whereas irregular structures do not. As it turns

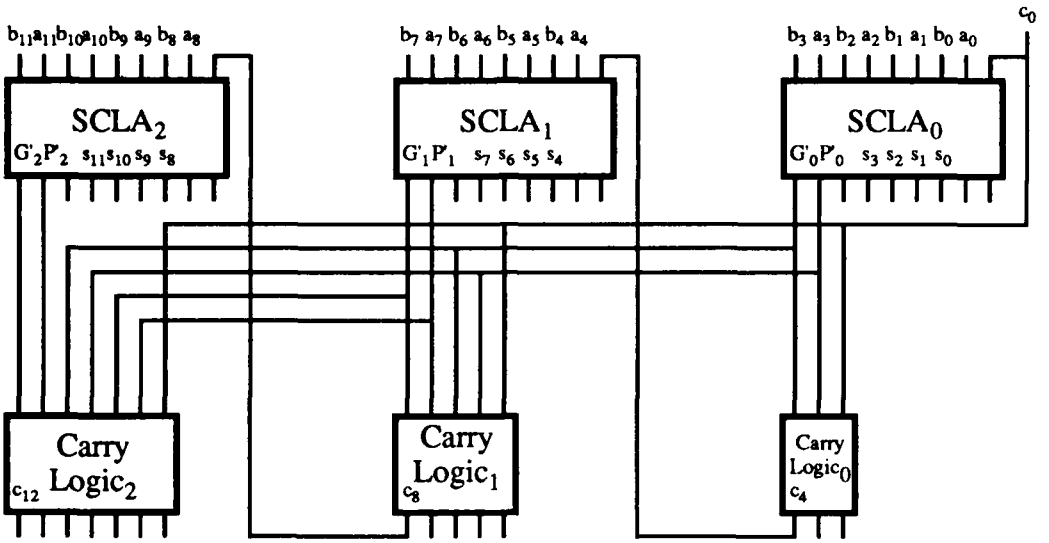


Figure 12: *A 12-bit SCLA.*

out, for the case of the ripple-carry adder, the ripple connections for an arbitrarily large adder can be trivially mapped onto a perfect shuffle by taking a "walk" through the perfect shuffle such that every link is traversed exactly once.

Irregular structures are not impossible to map onto a single perfect shuffle stage, but they do require more work. We investigated another case, which involves interconnecting PLAs for a 12-bit section carry lookahead (SCLA) adder. The schematic for this circuit is shown in Figure 12, which shows the layout as a digital electronic designer might draw it. Notice that the interconnects appear to be irregular, that the PLAs have different sizes as indicated by the varying numbers of inputs and

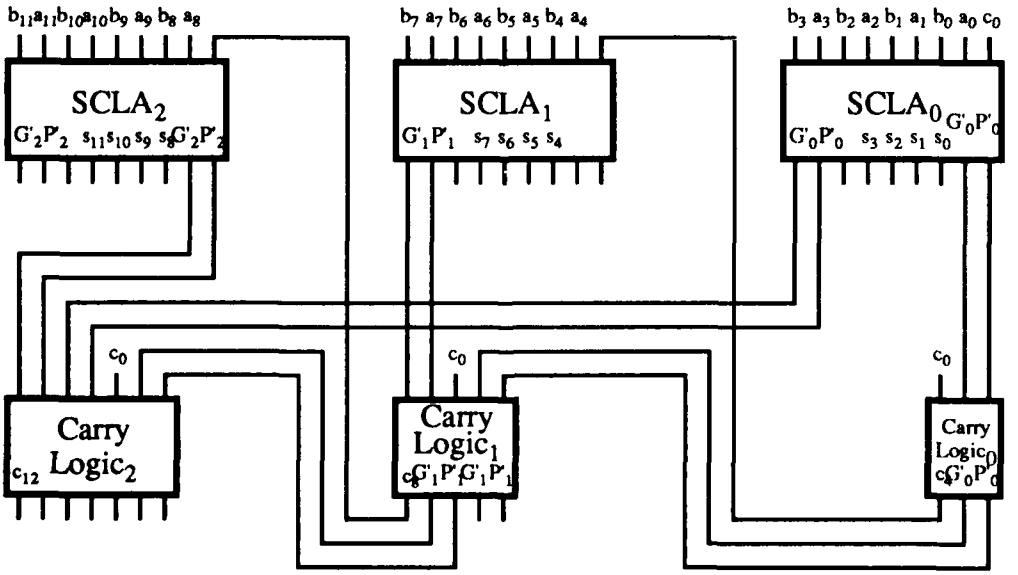


Figure 13: *The remapped 12-bit SCLA.*

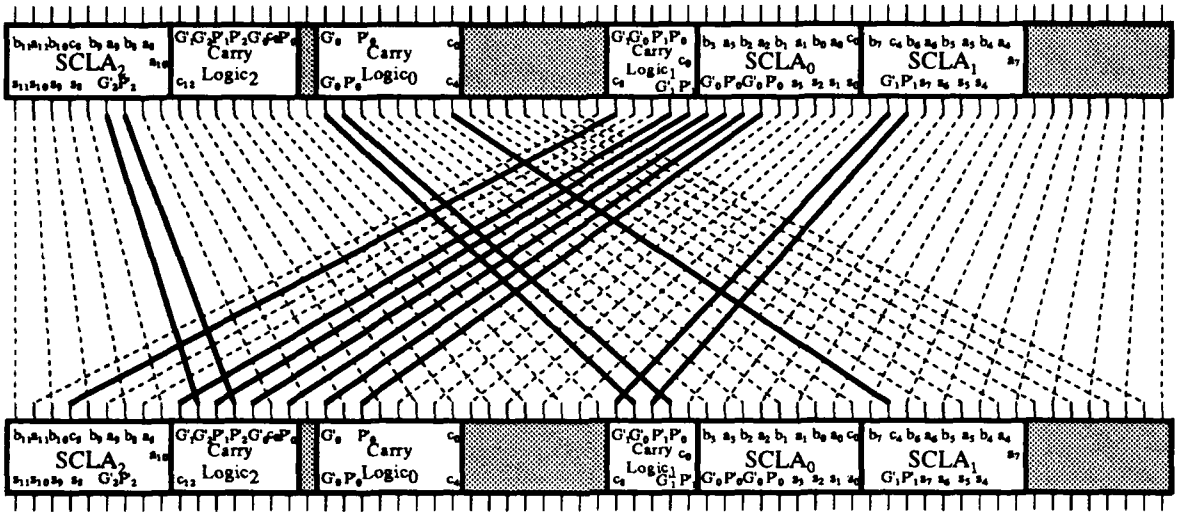


Figure 14: *A successful PLA tiling of the 12-bit SCLA shown in Figure 13, using a single perfect shuffle stage.*

outputs, and that there is fan-out within the interconnect itself. That is, a connection may be tapped in more than place. Since our point-to-point perfect shuffle does not support fan-out, our first step is to push the fan-out back to the originating PLAs. Figure 13 shows the remapped circuit. If an originating PLA is already at the maximum size allowed by the technology, then we must add a fan-out PLA. For this case, there is no need to add a fan-out PLA.

Again, we attempt to map all of the connections shown in this diagram into a single stage of a perfect shuffle. The problem is small enough to attempt an exhaustive search, which we did, and failed. There is no possible repositioning of PLAs with a 64-wide perfect shuffle that will work. We then tried a second approach, which involves repositioning the boxes (PLAs), and growing the smaller boxes up to the sizes of the larger boxes in order to absorb more input and output ports. This approach succeeded for the 12-bit SCLA as shown in Figure 14. Exhaustive search is no longer practical because the sizes of the PLAs vary, and so we took the best of the failed solutions for the original PLAs, after remapping to remove fan-out, and then used manual trial-and-error to obtain the solution.

We then attempted a similar mapping for a 16-bit SCLA. The original schematic for the circuit is shown in the upper diagram of Figure 15. The next step is to push the fan-out from the interconnect to the PLAs. We encounter a problem, because the G_0 and P_0 outputs from $SCLA_0$ are fanned out to four other PLAs, and there are not enough unused output ports in $SCLA_0$ to produce four copies of G_0 and P_0. We can extend the width of $SCLA_0$ to create four more output ports, or we can add a PLA that fans out G_0 and P_0, and thereby avoid increasing the size of the largest PLA. We tried the latter approach first. A fan-out PLA is added to the circuit, as shown in the lower diagram of Figure 15. The fan-out PLA produces two of the four needed copies of G_0 and P_0. The third copy goes to the Carry $Logic_0$ PLA, which produces the fourth copy of G_0 and P_0 at two of its unused output ports.

As before, we took the best of the failed solutions for a 64-wide perfect shuffle and then attempted to manually grow the smaller PLAs up to the sizes of the larger PLAs. We could find no solution using

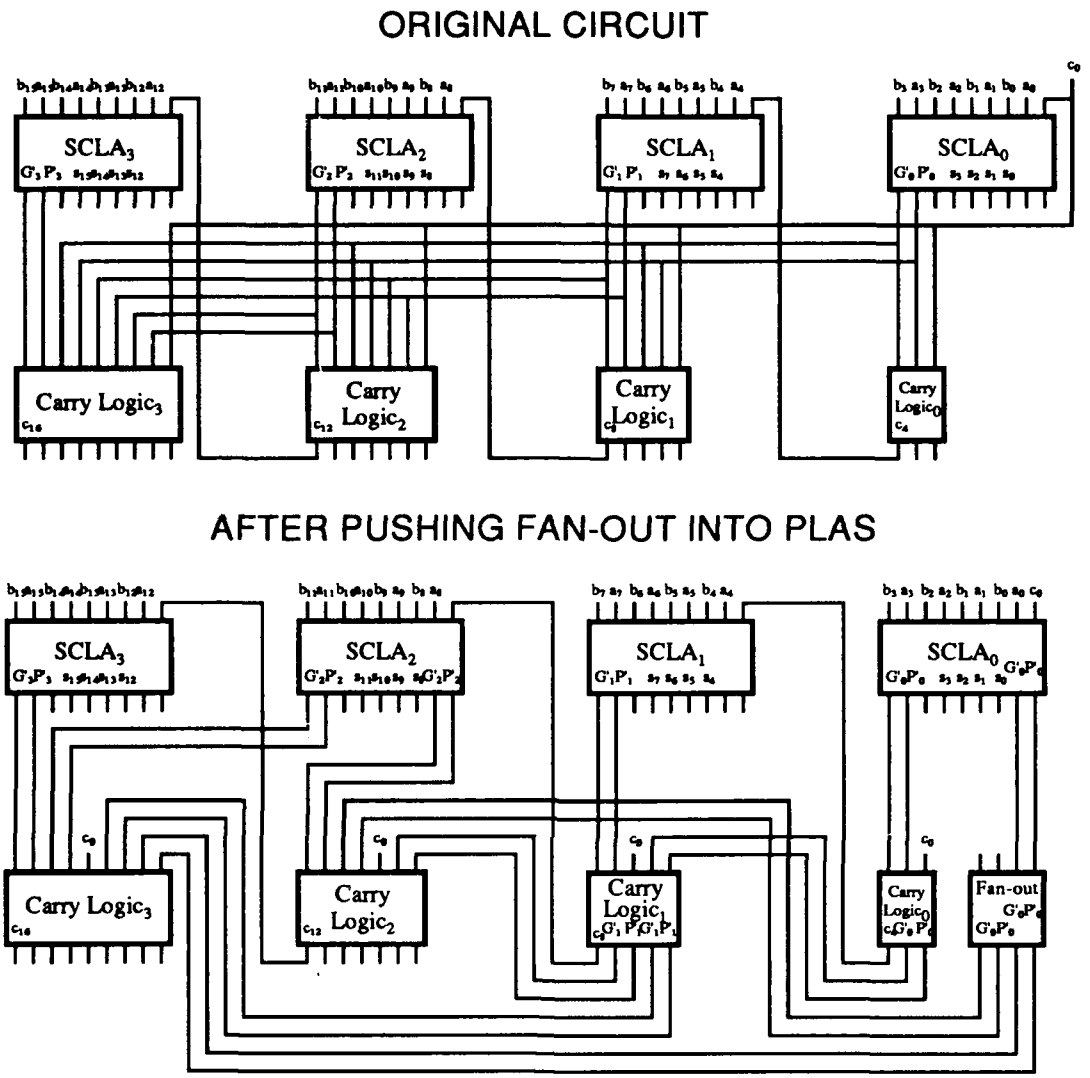


Figure 15: *Original 16-bit SCLA (upper diagram), and remapped circuit (lower diagram) in which all connections are point-to-point (no fan-ins or fan-outs).*

this approach, although a solution may exist. We did obtain a solution for the full 16-bit SCLA by extending the width of the $SCLA_0$ PLA, which allowed the fan-out PLA that was introduced in Figure 15 to be eliminated, and by extending the widths of some of the remaining PLAs. The width of the perfect shuffle was also doubled from 64 to 128. The solution is shown in Figure 16. We could not find a solution for the 16-bit SCLA using a single perfect shuffle stage that did not also increase the width of the widest PLA.

To summarize: a single point-to-point perfect shuffle interconnect without fan-out can implement all of the MSI-level interconnects. Although this is a significant accomplishment to achieve by hand, we have found no suitable method of fully automating the approach. At the moment, automatic MSI-level interconnection using regular interconnects for irregularly structured circuits is an open problem.

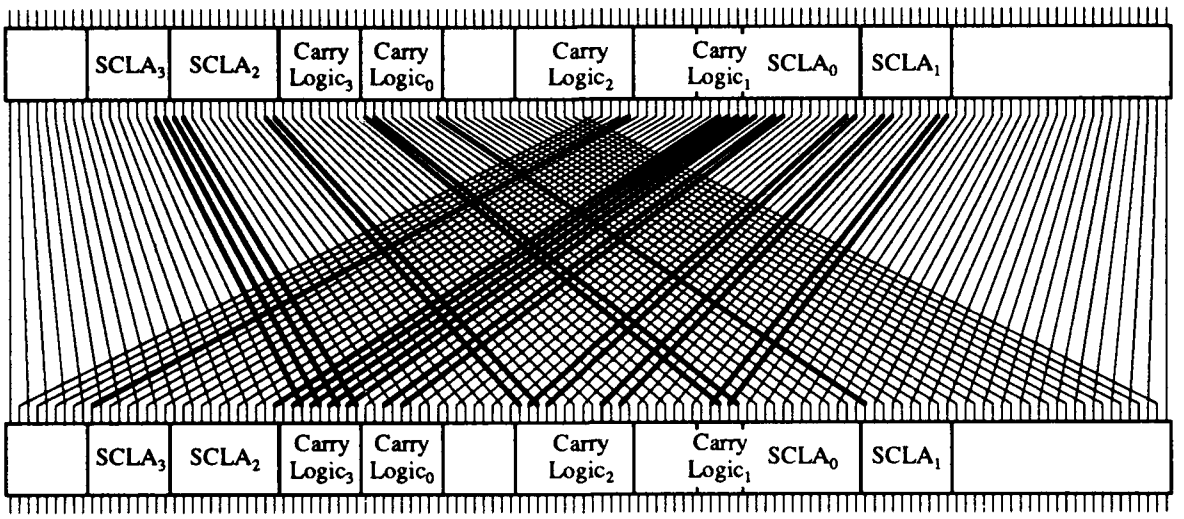


Figure 16: *A successful PLA tiling of the 16-bit SCLA shown in Figure 15, using a single perfect shuffle stage.*

2.6. Permuting Inputs, Outputs, and Minterms

In order to simplify the mapping of MSI components to a single stage of a perfect shuffle in the previous section, we assumed that we could simply have a signal enter somewhere at the top of a PLA, without regard to the actual position where it enters. For an electronic approach this might be allowable, but it might also not be allowable because VLSI design is typically done with "standard cell" libraries in which the inputs and outputs are constrained to certain positions. For our regular interconnect approach, the PLA tool chooses the positions of the inputs, and allows some flexibility in the positions of the outputs, but even with this flexibility, the outputs of one PLA rarely line up with the inputs of another. For this reason, we developed a **permutation tool**, which allows input

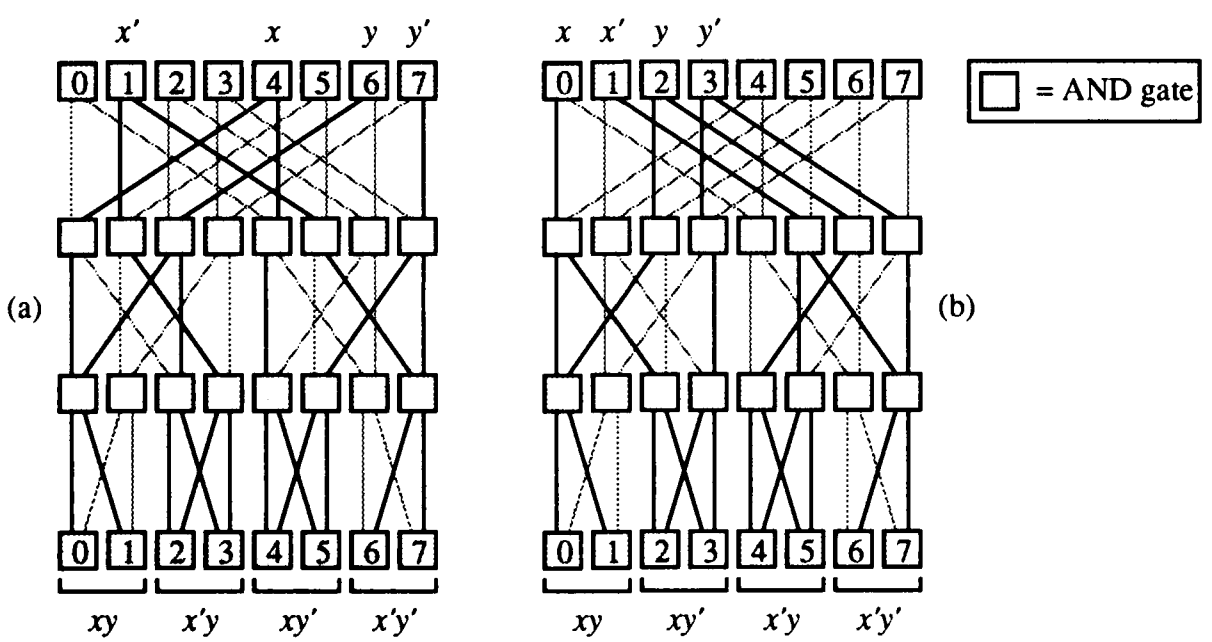


Figure 17: *Two functionally equivalent designs for a 2-to-4 decoder.*

positions and minterm positions (the middle stage of a PLA) to be arbitrarily specified by the designer. If a solution is not possible, the program responds quickly and the designer can try an alternate configuration, otherwise a solution is produced in reasonable time (no more than a few minutes).

As an illustration, Figure 17a shows a 2-to-4 decoder with the inputs x, x', y, and y' in positions that are permuted with respect to the 2-to-4 decoder shown in Figure 2, and the outputs at the bottom appear in the same positions as in Figure 2. Figure 17b shows a functionally equivalent circuit, in which the inputs are in the same positions as in Figure 2, while the minterms xy' and $x'y'$ have been interchanged.

3. FAULT AVOIDANCE

A potential problem with relying on novel GaAs based optical logic devices such as SEEDs or VCSELs is that defect densities are greater than for conventional Si electronics. Feature sizes tend to be large for photonic components when compared with conventional electronic components, and the pitches (center-to-center spacings) among elements are large due to cooling and diffraction limitations. In order to achieve comparable computation complexity to a digital electronics technology, either large perfect device arrays will have to be fabricated, or fabrication and processing defects will have to be dealt with some other way, such as at the architectural level.

We investigated two approaches of dealing with defects at the architectural level. One approach reorients and translates arrays so that defects colocate with unused positions in the arrays. We describe a bipartite matching algorithm in Ref. [7] for deciding how to orient the arrays so that the best match is made between the available arrays and the system being constructed. Consider an optical processor that makes use of 10 device planes with 32×32 optical devices in each plane, with an average of approximately one device fault per array. The **design density** is a measure of how many of the available logic elements are actually used, since some percentage is wasted due to redundancy that is necessary for avoiding faults. Here, 80% design density is used, which is typical for circuits designed with the PLA tool, since approximately 20% of the optical devices are unused as a result of the regular interconnection pattern, which leaves them in "landlocked" (inaccessible) positions. For this approach, we assume that a system design cannot be changed in order to deal with faults, and so we are faced with the problem of selecting device arrays so that faults colocate with unused devices. A **greedy** approach, which we consider for the purpose of comparison, tries to match an array with a given slot in the system by taking an array from a bin and trying it at one location, and if the match fails, then the array is discarded and another is tried. The graph in Figure 18 shows that this approach allows 53% of the arrays to be used. If rotation of an array by 180° is allowed, as well as translation by one position vertically and horizontally, and if the best fit is found between an array and the empty slots in the processor, then 91% of the arrays are utilized as illustrated in Figure 19.

A second approach redesigns circuits to route around faults. With this approach, greater flexibility is allowed by redesigning the circuits. The interconnects are changed after the initial design is created and after the locations of the faults are known, which is a simpler task than generating fault-free device arrays. An enumeration study shows that arrays can be used with an average fault ratio of 10%

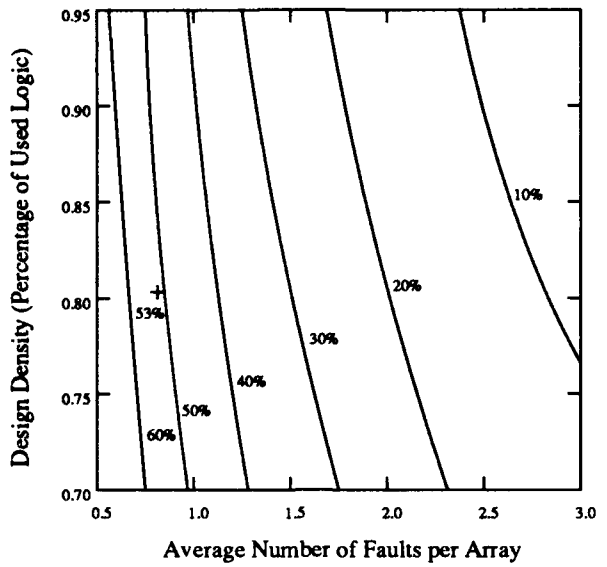


Figure 18: *Contour plot showing utilization of device arrays as a function of design density and fault density for greedy match.*

for small circuits. Figure 20 shows a sample of data taken from the study which shows different designs for a 2-to-4 decoder. Inputs at the top and minterms at the bottom of each circuit are permuted in order to bypass faults. Two simultaneous faults are considered for the upper 24 logic gates of each circuit in this sample. 87% of the faults are correctable for this situation, and when there are three randomly placed faults, over half of all fault combinations are correctable.

The reorientation / translation approach works best when defect densities are low, on the order of just a few faults per 1000 devices. The redesign approach is more appropriate when defect densities are

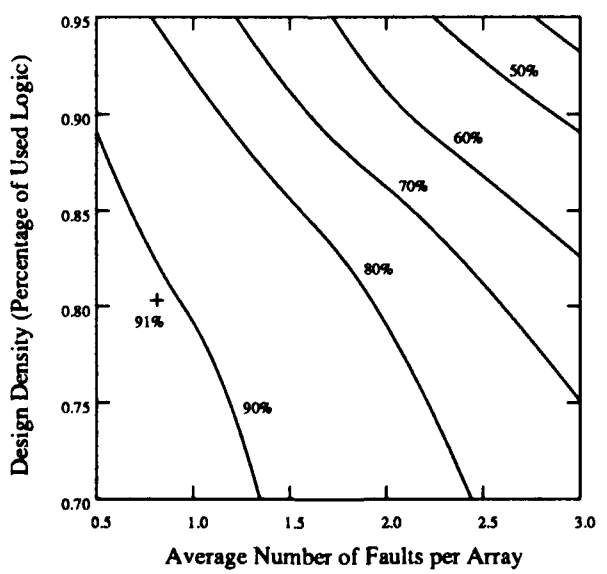


Figure 19: *Contour plot showing utilization of device arrays as a function of design density and fault density for translation and 180° rotation match.*

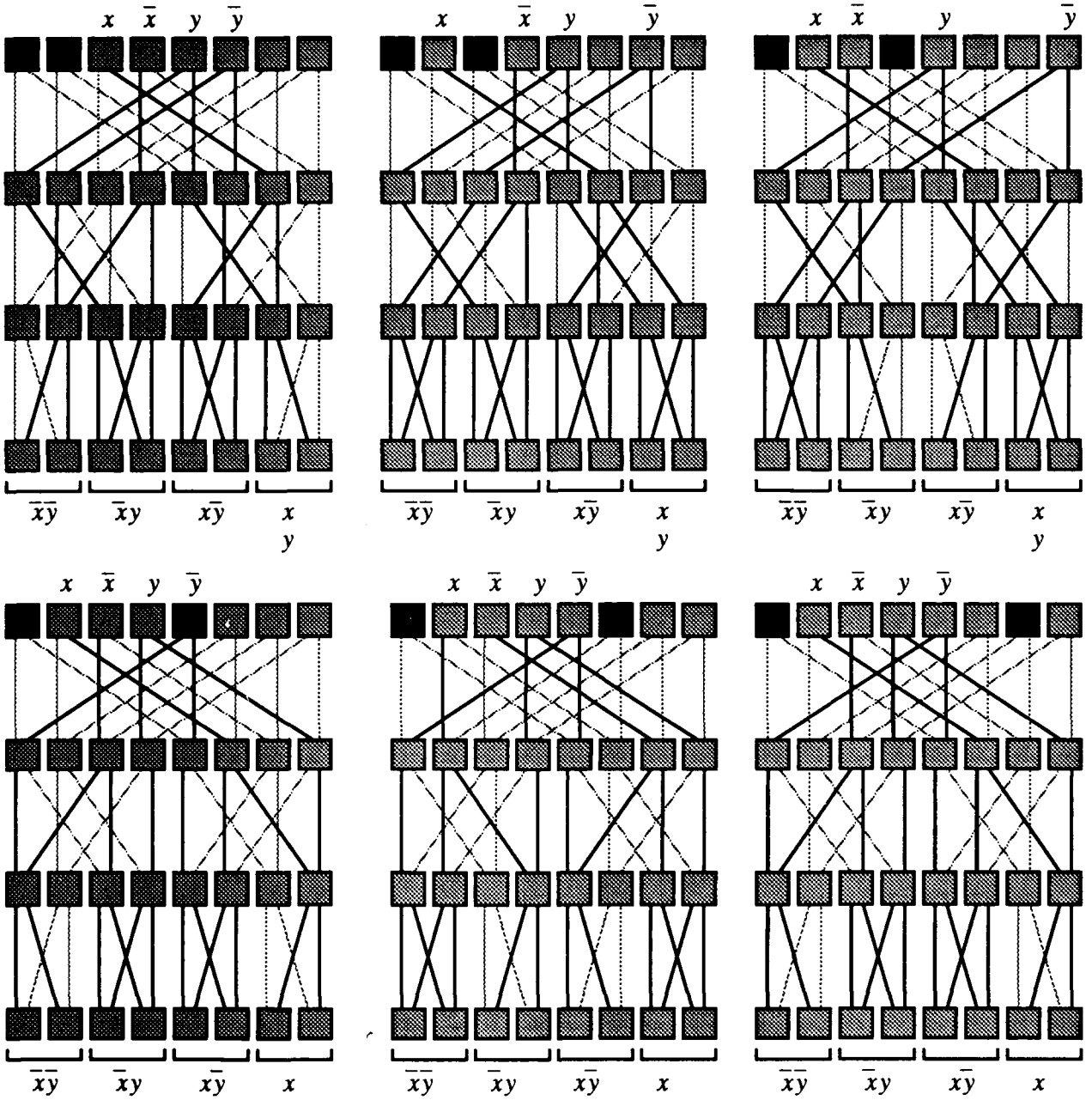


Figure 20: *Portion of exhaustive fault analysis for all two-fault combinations in a two-variable banyan connected PLA.*

high, on the order of 100 faults per 1000 devices. The redesign approach does not produce a completely new design for each circuit: the basic structure of the original design is maintained, and only the inputs and outputs are permuted for the individual PLAs. We feel that this constraint will simplify the development of a tractable redesign algorithm, which is an unexplored area.

4. INVESTIGATION INTO RECONFIGURABLE INTERCONNECTS

The notion of **functional locality** was introduced by Murdocca in the final report of an SDIO sponsored Phase I SBIR project that was administrated by AFOSR (contact is Dr. Alan Craig). The report suggests that an ordinary computer program displays locality in terms of the kinds of instructions it executes, that is, a program is likely to execute an instruction from the small set of instructions that are most recently executed. We investigated that notion by performing experiments, and found that functional locality exists in ordinary computer programs. This is a significant observation, because it may allow us to develop architectures that reconfigure themselves to match the natural form of the computation being performed, potentially leading to high performance architectures.

Just as spatial and temporal locality exhibited by a program are utilized to speed up memory delays by using cache memories, we believe that functional locality makes a strong case for the introduction of **function caches**, in which the interconnection patterns for just a few operations are maintained in the processor at any time. Reconfigurable processors that implement only a small set of machine instructions at any time but at rates faster than non-reconfigurable processors can exploit functional locality to achieve higher performance. Our belief that such a reconfigurable processor will execute its instructions faster than a comparable non-reconfigurable processor is based on the design guideline that smaller hardware is faster [8].

4.1. Measurements of Instruction Set Usage

Hennessy and Patterson [8] report on the instruction set usage for a number of application programs running on different architectures, and conclude that ordinary programs use only a small part of the total instruction set provided by the architecture, and that an even smaller set of instructions (about twelve or so) account for as much as 80% of the total number of instructions executed. This observation motivated us to look for functional locality in programs. Our study [9] is carried out in two parts. We look first at the extent of functional locality that arises solely from the fact that some instructions are executed more often than others. In this part of the study, using the run-time frequency count information collected in the instruction usage study reported in Ref. [8], we synthesized random runs with uniform frequency distributions of machine instructions matching those reported, and studied the hit-ratio that a reconfigurable processor would achieve for different sizes of the function cache using a first-in/first-out (FIFO) instruction replacement strategy. The idea is that it is the actual hardware that is being replaced, and not simply the codewords that represent instructions, which would be the case for a conventional electronic processor. This part of the study uses two architectures – the DEC VAX, and the DLX, which is a generic Load/Store architecture described in Ref. [8]. Three different programs are used on each machine: **gcc** (a C compiler), **spice** (a circuit simulator), and **tex** (a text formatter).

Figure 21 shows a plot of hit-ratio against code size for the DLX running gcc for different function cache sizes. Plots for **spice** and **tex** for the DLX, and also for these three programs on the VAX are nearly identical in form to Figure 21. For our purposes, the hit-ratio is the percentage of the total instructions executed for which reconfiguration is required assuming that at any given time, the processor implements only as many instructions as the size of the function cache allows and

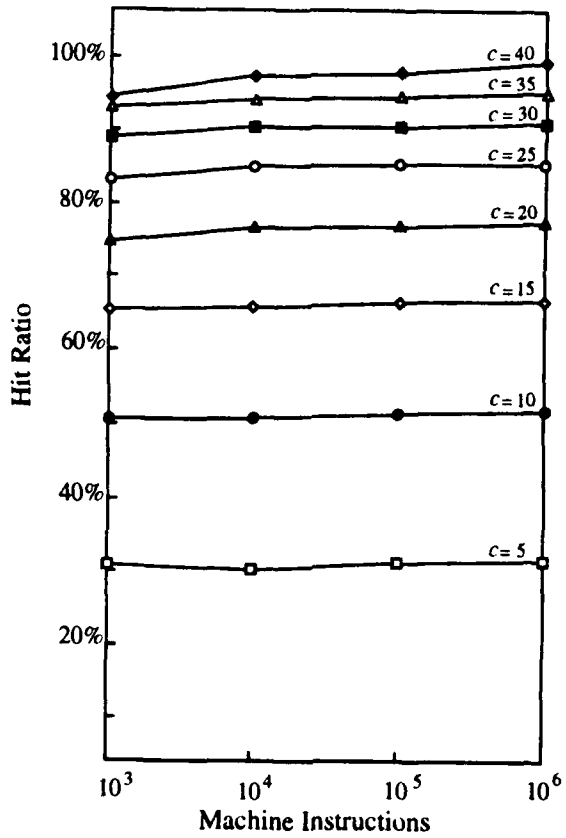


Figure 21: *Hit ratio versus code size for different sizes,* c *of the function cache for synthesized runs of* **gcc** *on the DLX.*

reconfigures itself to implement an instruction that is not in the function cache. In each case the hit-ratio increases as the function cache size increases but is almost completely insensitive to code size. For this reason, we use a code size of no more than 1,000,000 machine instructions for the measurements that follow.

Figures 22 and 23 show plots of hit ratios as functions of cache sizes for the DLX and VAX, for synthesized runs based on the instruction mixes found in **gcc**, **spice**, and **tex**. As shown in the plots, high hit ratios are obtained for small cache sizes. Motivated by these results, we developed software tools for the second part of the study, which allowed us to gather statistics on entire runs of sample programs. The architecture used for this part of the study is the SPARC based Sun-4 and the programs studied are **latex** and the **gcc** components: **gcc-cpp**, **gcc-cc1**, **as**, and **ld**. Collecting a program trace in this fashion slows down the program being traced by a large factor. For example, one trace of seven million instructions required nine hours of actual time. For this reason, the programs were executed using small sample files. Figure 24a shows the effect of changing the size of the function cache on the hit-ratio for different programs. For the programs shown in Figure 24a, we also gathered data on the run-time frequency distribution, generated runs with matching frequency distributions, and studied the effect of function cache size on hit-ratio. Our observations are shown in Figure 24b. Note

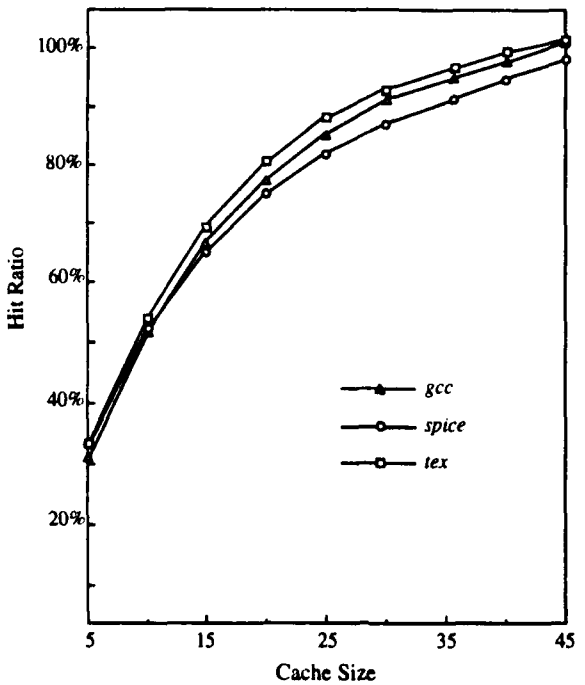


Figure 22: *Hit ratio as a function of the size of the function cache for synthesized runs on the DLX.*

that the hit-ratio values we see in Figure 24a are higher than corresponding values seen in Figure 21 for the same function cache size. This indicates that the programs in our study exhibit functional

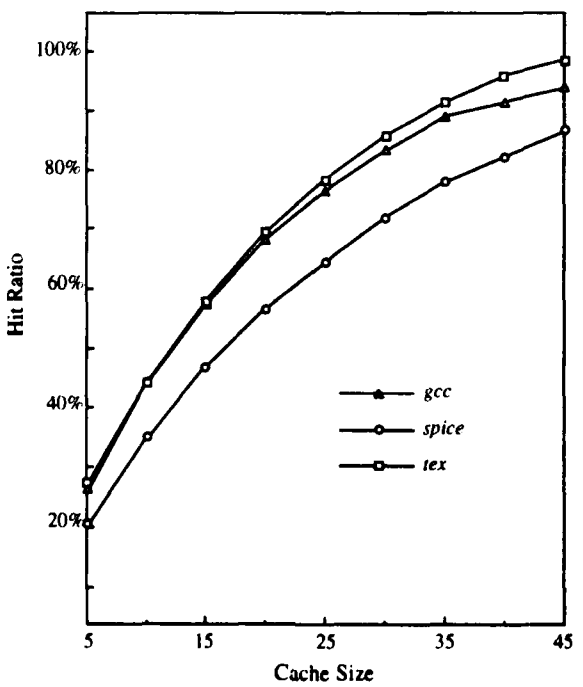


Figure 23: *Hit ratio as a function of the size of the function cache for synthesized runs on the VAX.*

locality to a higher degree than would be exhibited simply because some instructions are executed more frequently than others.

4.2. Discussion

The data we have collected provides evidence for the existence of functional locality in ordinary programs. Based on the evidence, we hypothesize that a reconfigurable processor that modifies its hardware to execute a slowly changing set of machine instructions can exploit functional locality to achieve higher performance than a non-reconfigurable processor. In order to quantify the performance gain, we define β (> 1) as the ratio by which the execution of an instruction which is not in the function cache is slowed down compared to the execution of an instruction in the cache. The factor by which a reconfiguring processor is slowed down because of misses is then given by *slowdown* =

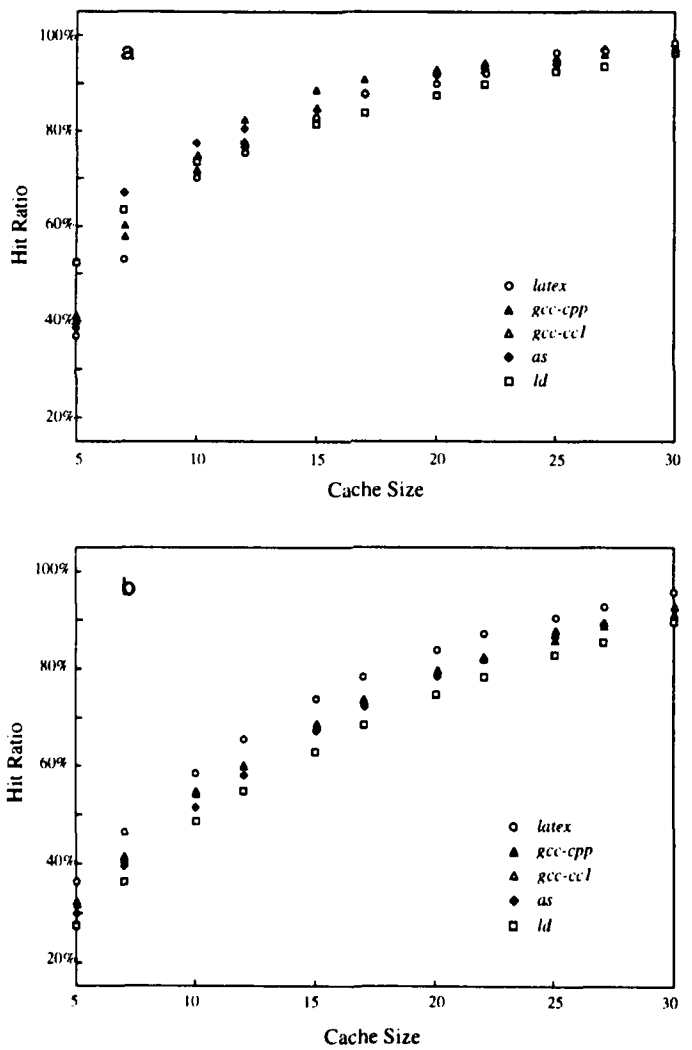


Figure 24: *Hit ratio as a function of the size of the function cache for synthesized runs on the SPARC for (a) actual runs, and (b) synthesized runs with the same distribution of instructions as the actual runs.*

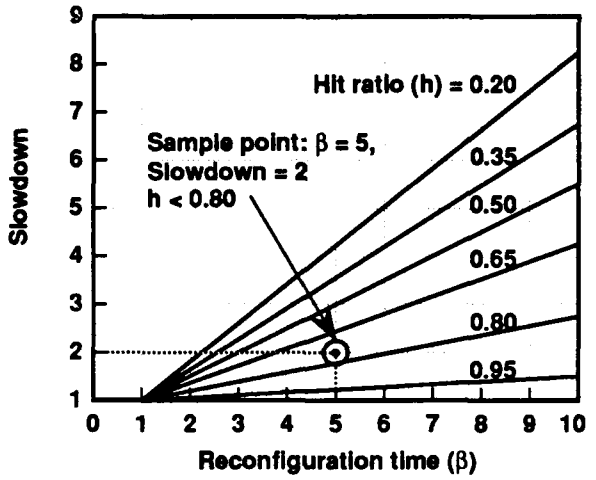


Figure 25: *Slowdown factor as a function of the reconfiguration cost and hit-ratio.*

$h + \beta \times (1 - h)$ where h is the hit-ratio. If α (< 1) is the ratio of the speed with which the reconfigurable processor executes an instruction that it finds in its cache to the speed at which a non-reconfigurable processor executes an instruction, then in order for the reconfigurable processor to be faster than the non-reconfigurable processor, α should be less than $1/slowdown$.

From Figure 25, it is clear that the higher the hit-ratio, the lesser the sensitivity of the slowdown factor to the cost of reconfiguration. We choose a sample point $\beta = 5$, based on the expected reconfiguration time as compared to the bit rate of matrix addressable devices that are being developed at Photonics Research Incorporated (PRI) under NASA support, and which have also been developed at AT&T in Breinigsville, PA. We choose h near 0.8, which is typical for execution runs we have observed. This sample point gives us a slowdown of 2, which means that the processor runs twice as slow as a result of misses than it would run if there are no misses at all. A reconfigurable processor is assumed to be faster than a non-reconfigurable processor as a result of its reduced size, however, and so the speedup must compensate for the slowdown. In order for the reconfigurable processor to break even, it must execute instructions at twice the rate of a non-reconfigurable processor. Based on our knowledge of conventional computer architecture, however, we do not believe that reducing the instruction set by as much as 90% will produce a speedup of much over 2, and so we chose not to pursue this style of reconfiguration. Instead of trying to speed up a conventional serial processor through reconfiguration, we feel that we can use the same principle of functional locality to speed up a parallel processor. We describe an approach we are pursuing that is based on dataflow computing in Section 10.

Although we do not feel there will be a significant payoff in using the function cache concept in a conventional processor, a potential application of these results is for the DOC II optical processor [10] under development at OptiComp Corporation. The DOC II processor is being developed for a SPARC-like instruction set, but only a subset of the instruction set can be implemented at any one time. Our investigation here can be used both to determine how to organize the cache (which is implemented by the DOC II instruction mask) and to characterize the effectiveness of the strategy.

4.3. Gate-Level Reconfiguration

In a few of the optical processors developed at AT&T Bell Labs, SEED based architectures are customized by placing fixed masks in the image planes of the interconnects. There is no need for the customizing masks to remain fixed, and in fact, they may be implemented in any of a number of ways that support runtime reconfiguration, such as with ferroelectric liquid crystals, matrix addressable logic arrays, or through beam steering elements. Although reconfiguration times may be longer than the bit rates with these approaches, the existence of functional locality helps to offset the mismatch in speeds. We have developed the notion of a reconfigurable logic/interconnection component (RELIC) which makes use of reconfigurable optical interconnects at the gate level. In a very simple form, a RELIC consists of independently addressable optical logic gates and static optical interconnects. Reconfiguration at the gate-level is obtained by selectively enabling or disabling logic gates.

A RELIC may compete with existing electronic field programmable gate arrays (FPGAs) such as the Xilinx line of reconfigurable components. The internal configuration of a Xilinx chip is shown in the left side of Figure 26. A number of PLAs, shown as rectangles, are interconnected through an embedded arrangement of crossbar switches. The PLAs contain lookup tables (LUTs) for two seven-variable Boolean functions, and provide two bits of internal feedback to the LUTs. Each PLA generates two one-bit outputs. A small number of channels (five shown in the figure) pass through each crossbar in horizontal and vertical directions. The LUTs and crosspoints of the crossbars are configured by loading static flip/flops, one per decision element (or crosspoint). The Xilinx chips are popular in the area of rapid prototyping, in which a hardware implementation of a target processor

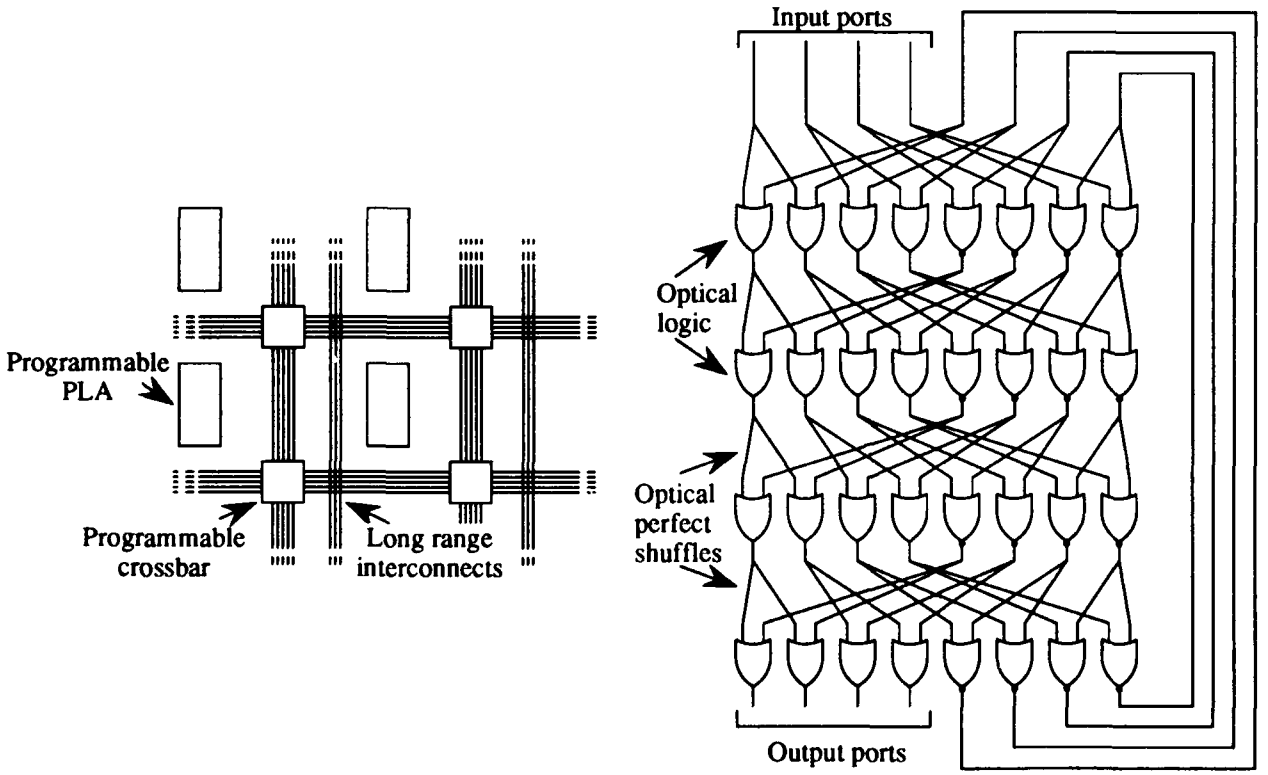


Figure 26: *Xilinx (left) and RELIC (right) models.*

is realized with reconfigurable components, but at a greater cost and with reduced performance than with a custom hardware design. Commonly, Xilinx chips are used in end-products as well as in transition hardware, particularly when production quantities are small (less than 1000 units).

With regard to reprogrammability, the Xilinx line is very flexible, but the user is forced to decompose large circuits into a number of interconnected one-bit circuits. This often unnatural decomposition sacrifices performance. For example, a ripple-carry adder maps well to the Xilinx approach because of its regular form (see Figure 11), but a fast parallel adder that has an irregular form (see Figure 12) does not. As an illustration of why this is the case, consider the general layout of the Xilinx chip, which is clustered into one-bit logic units and narrow communication channels. Although it is possible to create a gate-level switching matrix that allows a user to modify interconnects at the gate or component level, it would be nearly impossible to maintain a clock speed of 50 MHz (a typical Xilinx internal clock speed) due to the enormous wiring complexity of such a chip.

The RELIC approach allows gate-level and component-level interconnects to be allocated as needed, without suffering a large increase in wiring complexity. This is accomplished through a regular gate-level interconnection pattern such as a perfect shuffle, in which either the logic gates or the connections are reconfigured during operation. The gate-level layout of a RELIC is illustrated in the right side of Figure 26. A few logic gates are devoted to inputs and outputs, and the remaining logic gates are interconnected in a regularly structured "sea-of-gates" with no pre-allocation of local or global interconnects. We have found that gate counts using this method are higher than for a fixed arbitrary interconnect, as noted in Section 2, but that the flexibility of modifying the gate-level interconnects may override the cost in gate count. In comparison, less than 10% of the available logic is used for any particular application with a Xilinx approach, and our optical approach is not far from the electronic approach in terms of gate count.

5. OPTICAL INTERCONNECTS

We carried out an investigation into novel interconnection technology, in order to understand trade-offs between the complexity of the optics and the complexity of the architecture. As an example of an issue in the trade-off investigation, consider that irregular interconnects can be achieved with diffractive optical elements. Our studies show that there is a trade-off between lens size and propagation distance (see Section 5.3). A completely irregular interconnect will effectively require a separate imaging system for each optical signal, and the resulting propagation distance of a few millimeters may not allow for steep angles of incidence, thus complicating the implementation of a completely irregular interconnect. A mix of regular and irregular interconnects appears to be a reasonable compromise when the trade-offs among the optics and architecture are considered together. One rule of thumb that we have used is to maintain regular interconnects for clusters of signals, 4×4 for example, and then use irregular interconnects between clusters. In this way, propagation distance can be increased while simultaneously reducing the circuit depth that is attributed to the regularity.

We have also studied the AT&T S-SEED based optical processor in detail, and we achieved a familiarity with S-SEED switching devices by working for several weeks with Dr. Robert Morgan

in the AT&T Solid State Technology Center in Breinigsville, Pennsylvania. An outcome of the AT&T visits is the understanding that optical logic gate imaging with single large lenses is difficult to extend beyond modest array sizes. A visit to Scott Hinton's group at AT&T Bell Labs in Naperville, Illinois confirmed this finding. The problem is due to the incompatibility of large field sizes and small diffraction limited spot formation. A micro-optics approach is an important alternative, and we studied various aspects of this approach. A detailed analysis has been made of the trade-offs between micro-element size and crosstalk-free propagation distance imposed by diffraction. Contours of this trade-off have been plotted and serve to guide in the formulation of micro-optic interconnection architectures.

In a separate but related Rome Laboratory sponsored effort, we are collaborating with the Photonics Center at Griffiss AFB in the design and construction of an all-optical digital processor based on S-SEED devices. A significant problem for the RL project is in how to implement the interconnects. The calcite approach described below is one method we have developed that has been transitioned to the Photonics Center. The method has influenced the types of interconnects that we support in the tools. For example, the calcite approach is ideal for a split-and-shift topology that might be studied in XOPID.

5.1. Birefringent Array Generation and Interconnection

A hardware solution to two related problems has been demonstrated: (1) the generation of arrays of spots from a single source or from multiple sources, and (2) the interconnection of optical logic gates. Spot-array generation is a significant problem in providing power beams to modulator devices like the S-SEEDs, which are used in optical processors under development at AT&T, Boeing Aerospace, Optivision, and the Photonics Center.

Cascaded slabs of birefringent materials can be used for efficient spot-array generation and for providing fan-out in optical interconnection. This was first shown, for the case of cascaded Wollaston Prisms, by Jewell *et al.* [11]. In the approach investigated here, collimated or converging beams are repeatedly split by propagation through simple slabs of birefringent media. These media can include, for example, calcite, rutile, quartz, or form-birefringent materials [12]. In the first stage shown in Figure 27, a spot of light polarized at 45° to the axes is imaged through a uniaxial crystal slab which is oriented with its reference plane (a plane containing both the ordinary and extra-ordinary rays) parallel to an axis. The output image of the single input spot is now resolved into two spots which are orthogonally polarized. The ordinary spot is not displaced and is polarized perpendicularly to the reference plane. The extra-ordinary spot is polarized parallel to the reference plane and is displaced by a distance proportional to the thickness of the crystal slab. In the second stage the process is repeated. Since the input spots are now polarized along the axes, the crystal slab is rotated by 45° so that each input spot retains equal components of ordinary and extra-ordinary light. The output image now consists of four spots, with orthogonal polarizations as shown. With each subsequent stage, the crystal is rotated by 45° and the number of spots are doubled. In practice, all the crystals may be optically contacted or cemented to reduce surface reflections and scatter, and a single imaging stage can be used for the entire cascade. For the case of calcite, ordinary and extraordinary rays are internally separated by an angle of 6.2°, and the crystal slabs are thus about 10 times thicker than the spot separation desired for each stage.

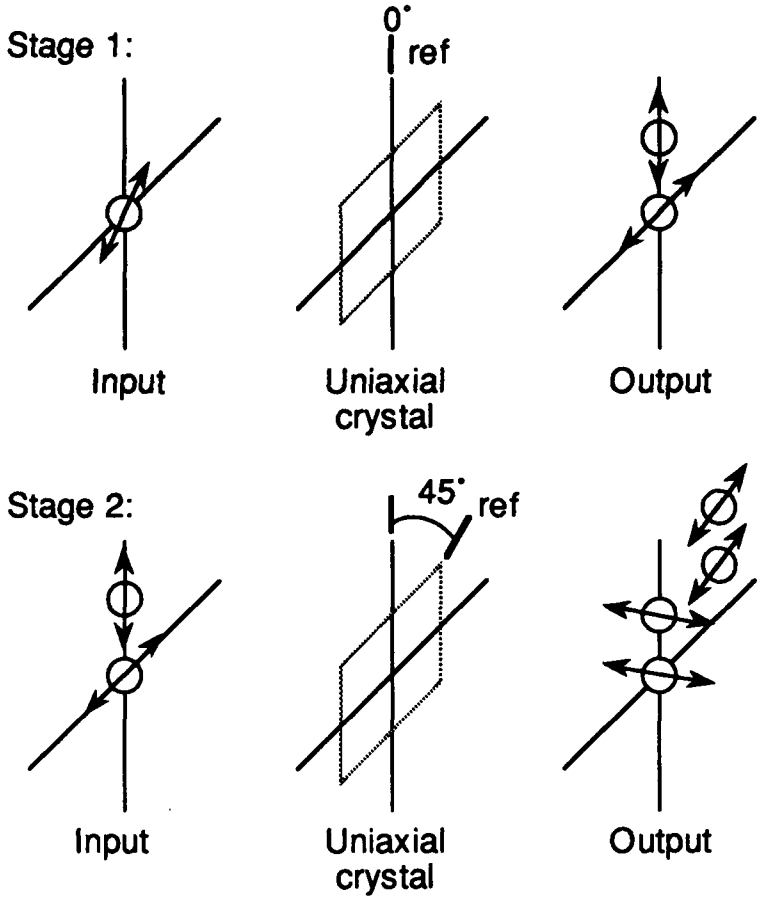


Figure 27: *Birefringent array generation.*

Thin birefringent slabs with large lateral dimensions may be readily cleaved from inexpensive crystals such as calcite. The lateral extents of such slabs are not limited as with crystal prism approaches using Rochon, Wollaston, or related prisms. Other practical advantages of the cascaded slab approach include compactness, ease of manufacture, and integrability. The method has been demonstrated, and has been transitioned to the Photonics Center at Rome Laboratory, where it is being considered for interconnection and for spot-array generation for their S-SEED based processor.

5.2. Sub-Array Generation, Interconnection, and Redundancy

The birefringent slab technique may be particularly useful for sub-array generation in which a sparse regular array of beams is transformed into a much denser spot array with approximately the same lateral extent. For example, an array of surface emitting microlasers may be fabricated with relatively wide element spacings to facilitate cooling. Such a coarse array can produce a dense array of spots by imaging the array through a few cascaded birefringent slabs. This is shown in Figure 28 which is a digitized photograph of an output array consisting of 48 spots. In this experiment, 12 input beams at a wavelength of 0.85 μm are derived from three diode lasers and beamsplitters. These 12 beams are aligned into a regular array (simulating the output of an array of microlasers) and are focused to form spots through two cascaded calcite slabs. The two calcite slabs quadruple the density of the

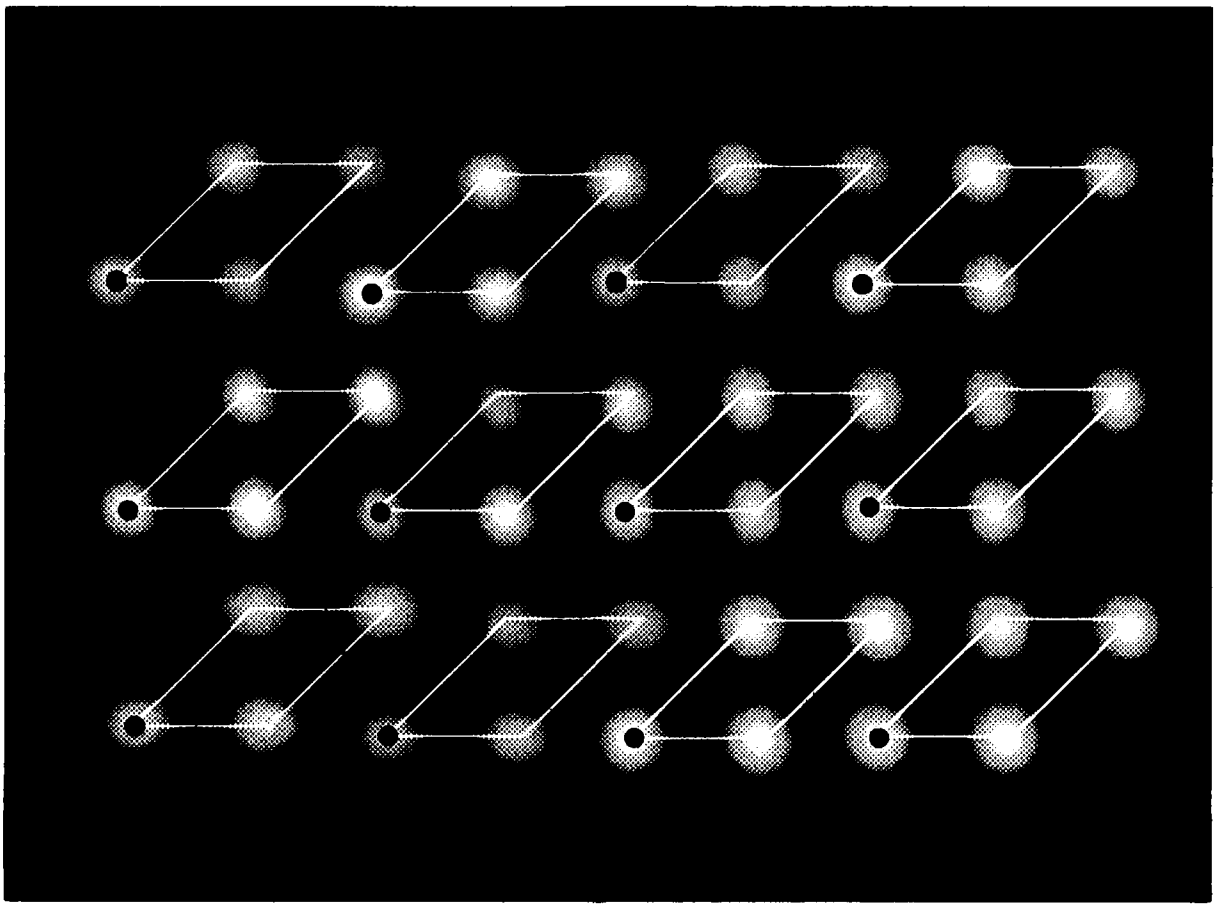


Figure 28: *Sub-array generation.*

resulting spot array, causing each input beam to produce a local cluster of four spots. The 12 input beams and local spot clusters are shown in the diagram. The filled circles that overlay 12 of the 48 spots indicate positions in the source array. The vertices of each overlaid parallelogram indicate positions of spots that are generated from the corresponding source. This experiment also demonstrates the application of birefringent slabs to provide fan-out for optical interconnections. The light from each input beam in this example is now equally divided among four locations. Similarly, the opposite case of fan-in can be accomplished in which crystals are used to overlay light from neighboring spots. In addition to being highly efficient, birefringent array generation and interconnection are much less dispersive than diffractive techniques (like Dammann gratings, which are used at AT&T in their S-SEED processors) and are therefore useful with multiple wavelengths, broadband light, or in situations where wavelengths may drift.

An important use of cascaded birefringent slabs is as an efficient method to establish redundancy in spot arrays. It was shown by Lohmann [13] that spot homogeneity and system reliability can be greatly enhanced by using spot arrays in which the spots are formed by superposing the outputs of a multiplicity of sources, rather than the usual case in which a single source supplies one or more spots exclusively. The reduced coherence resulting from uncorrelated source superposition greatly enhances the homogeneity of the spots by averaging out coherence related structure. Further, a large

degree of source fault tolerance is achieved with this redundancy. For example, if each spot contains equal input from 32 sources, a failure of one of the input sources will only reduce the overall array uniformity by 3%.

5.3. Micro vs. Macro-optics

We have made an investigation into trade-offs between approaches using micro-optics and macro-optics for interconnecting arrays of optical logic devices. This is an important contribution to the overall effort since it forms a basis for developing the design tools so that they satisfy both fundamental and practical constraints to interconnection. Results of these analyses are reported in a book chapter that we prepared for *Optical Computing Hardware*, edited by Sing Lee and Jürgen Jahns [3]. An excerpt from this study is given below:

> *Critical Distance for Collimated Array*
>
> An example of the diffraction-based trade-offs in device spacing and propagation distance is given for the case involving a collimated array of beams. The critical distance is a function of the microlens diameter. For example, consider an array with a device spacing $\Delta = 200$ μm and light of wavelength 0.85 μm. If the lens is only 10 μm in diameter ($D/\Delta = 0.05$), there will be a buffer zone of width $B = 95$ μm on each side of the microlens over which the light may spread before crossing into the neighboring channel. The diffraction spread angle of the beam from such a small lens, however, is large (4.9°), and after only $L_c = 1.1$ mm the beam begins to spread beyond the 95 μm buffer and mix with the neighboring signal. Similarly, as the microlens diameter approaches the gate spacing, the critical distance L_c is also very small. Near this other extreme, if $D/\Delta = 0.95$ ($D = 190$ μm) the diffraction angle is a much smaller .26°, but the buffer zone width is now reduced to $B = 5$ μm, and L_c is again only 1.1 mm. However, for less extreme values of D/Δ (*e.g.*, near 0.5), L_c is much larger (nearly 6 mm). Figure 29 shows a plot of L_c (given in millimeters) as a function of varied fill factor D/Δ and gate spacing Δ. Since the diffraction angle decreases with increasing lens diameter, one might suspect that low crosstalk could be maintained over longer distances if the full width Δ could be utilized for the microlens apertures. Effective use of these larger apertures can be accomplished by slightly focusing the beam emerging from the microlens, thus avoiding the condition in which any spreading of the collimated beam from a lens with $D = \Delta$ results in crosstalk.

This focused array configuration is discussed in the book chapter, as well as several other results of the investigation.

The micro/macro-optics study has been useful in identifying applications that are best served with conventional optics, and those where diffractive optics are more reasonable to apply. This aspect of the work influences the design of digital circuits, and we have used the XOPID tool to investigate the architectural implications of using various combinations of micro and macro-optics.

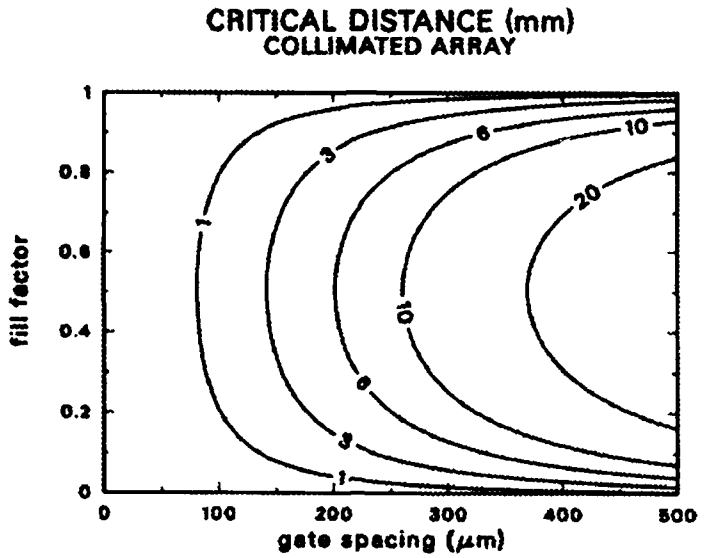


Figure 29: *Plot showing critical distance as a function of gate spacing and fill factor.*

5.4. Achromatic Optical Interconnects.

A lesson learned from the studies carried out in this project is the highly constrained nature of the plane-to-plane optical interconnect problem. The trade-offs between gate or device spacing, spot size, number of devices, plane separation, imaging system complexity, *etc.* are strongly related to each other, and reasonable compromises immediately push optical system complexity near practical limitations. It has become increasingly apparent that using wavelength as a new dimension in order to extend the interconnect system performance without a corresponding increase in complexity is a promising direction. Further, it has been demonstrated by others that the dispersion of "single wavelength" interconnect systems can be a problem (particularly with systems utilizing diffractive elements) with the typical wavelength drift or mode hopping in sources. In response to these considerations, the topic of achromatic optical interconnection systems has been investigated.

In this area, several methods for achromatizing finite and infinite conjugate interconnects have been studied. The dispersion resulting from using simple refractive and diffractive single element lenses was characterized and compared with systems using refractive and hybrid refractive-diffractive achromats. Further, new combinations of simple dispersive elements comprising an overall achromatic interconnect system were developed. The advantage of the latter approach is that the complexity of the elements in the system is greatly reduced over the all-achromat approaches. Typically some chromatic difference of magnification is left as a residual, but this is not a problem in most cases and can be eliminated if required. These results are being prepared for publication.

6. REFERENCES

[1] Murdocca, M. J., *A Digital Design Methodology for Optical Computing*, The MIT Press, (1990).

[2] Jahns, J., and M. J. Murdocca, "Crossover Networks and their Optical Implementation," *Applied Optics*, **27**, pp. 3155-3160, (Aug. 1, 1988).

[3] Smith, D., M. Murdocca, and T. Stone, "Parallel Optical Interconnections," book chapter in *Optical Computing Hardware*, vol. 2, edited by S. Lee and J. Jahns, Academic Press, (1993, to appear).

[4] Murdocca, M. J., A. Huang, J. Jahns, and N. Streibl, "Optical Design of Programmable Logic Arrays," *Applied Optics*, **27**, pp. 1651-1660, (May 1, 1988).

[5] Murdocca, M. J., V. Gupta, and M. Majidi, "New Approaches to Digital Optical Computing Using Parallel Optical Array Logic," book chapter for *Photonics in Switching*, vol. I, J. Midwinter, ed., Harcourt, Brace and Jovanovich, pp. 195-223, (1993, to appear).

[6] Gupta, V., "XOPID: An Interactive Design Tool for Digital Optical Circuits," unpublished user manual, (1992).

[7] Smith, D. E., "Fault Avoidance for Fixed-Interconnect Optical Computers," *Applied Optics*, **31**, pp. 167-177, (Jan.10, 1992).

[8] Hennessy, J. L. and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, (1990).

[9] Murdocca, M. J., and V. Gupta, "Architectural Implications of Reconfigurable Optical Interconnects," *Journal of Parallel and Distributed Computing*, vol 17, no. 3, pp. 200-211, (Mar. 1993).

[10] Guilfoyle, P. S., "Digital Optical Computer Fundamentals, Implementation, and Ultimate Limits," *Digital Optical Computing*, Proc. of *O-E/Lase'90*, SPIE Press, **CR35**, pp. 288-309, (1990).

[11] See for example: Jewell, J. L., S. L. McCall, Y. H. Lee, A. Scherer, A. C. Gossard, and J. H. English, "Optical Computing and Related Microoptic Devices," *Appl. Opt.*, **29**, pp. 5050-5052, (1990).

[12] Shiraishi, K., T. Sato, and S. Kawakami, "Experimental Verification of a Form-Birefringent Polarization Splitter," *Appl. Phys. Lett.*, **58**, pp. 211-212, (1991).

[13] Lohmann, A. W., and S. O. Sinzinger, "Spatial Noise Reduction in Array Illuminators," *OSA Optical Computing Technical Digest Series*, vol. 6, pp. 38-40, (1991), (Digest from Topical Meeting on Optical Computing, March 4-6, Salt Lake City, Utah, 1991).

7. PUBLICATIONS AND PRESENTATIONS

The following publications and presentations were made for the effort.

7.1. Publications

Murdocca, M. J., "Computer-Aided Design of Digital Optical Computers Using Free-Space Interconnects," *Optics in Complex Systems*, Lanzl, F., H.-J. Preuss and G. Wiegelt, eds., *Proc. SPIE*, vol. 319, Garmisch, Bavaria, pp. 126-127, (1990).

Murdocca, M. J., V. Gupta, and M. Majidi, "Logic and Interconnects in Optical Computers," pp. 129-134, *Photonics Spectra*, (Dec. 1990).

Murdocca, M. J., "High-Level Design of Digital Computers Using Optical Logic Arrays," *Proceedings of the 1991 Annual SPIE Symposium*, vol. 1474, pp. 176-187, Orlando, (Mar. 1991).

Murdocca, M. J., V. Gupta, and M. Majidi, "A Hardware Compiler for Digital Optical Computing," *Optical Computing*, 1991 Technical Digest Series, (Optical Society of America, Washington, D. C.), pp. 191-194, (1991).

Murdocca, M. J., and S. Levy, "Design of a Gaussian Elimination Architecture for the DOC II Processor," *1991 O/E-Lase Symposium, Proc. SPIE*, vol. 1563, pp. 255-266, San Diego, (Jul. 1991).

Smith, D. E., "Fault Avoidance for Fixed-Interconnect Optical Computers," *Appl. Opt.*, **31**, pp. 167-177, (Jan. 10, 1992).

Murdocca, M., "Architectural Implications of Optical Computing," *Proceedings of the 12th GI/ITG Conference on Architecture of Computing Systems*, Kiel, Germany, pp. 131-142, (Mar. 1992).

Murdocca, M. J., and A. Sharma, "An Application of Optically Reconfigurable Interconnects to the Dataflow Parallel Computing Paradigm," *Advances in Optical Information Processing V*, Dennis R. Pape, ed., *Proc. SPIE*, vol. 1704, pp. 453-464, Orlando, (1992).

Gerasoulis, A. and T. Yang, "Scheduling Program Task Graphs on MIMD Architectures," book chapter in *Algorithm Derivation and Program Transformation*, edited by R. Paige, J. Reif, and R. Wachter, Kluwer, (1992).

Yang, T. and A. Gerasoulis, "PYRROS: Static Task Scheduling and Code Generation for Message Passing Multiprocessors," *ACM International Conference on Supercomputing*, (July 1992).

Gerasoulis, A. and T. Yang, "Automatic Program Scheduling for Distributed Memory Scalable Architectures," Submitted to: CONPAR92/VAPPV Conference, (Sep. 1992).

Murdocca, M. J., and V. Gupta, "Architectural Implications of Reconfigurable Optical Interconnects," *Journal of Parallel and Distributed Computing*, vol 17, no. 3, pp. 200-211, (April, 1993).

Murdocca, M., "A Case for All-Optical Digital Computing," in *Optical Computing Technical Digest*, (Optical Society of America, Washington, D. C., 1993), vol. 7, pp. 309-312, (1993).

Smith, D., M. Murdocca, and T. Stone, "Free-Space Optical Interconnection," book chapter for *Optical Computing Hardware*, edited by J. Jahns and S. Lee, Academic Press, (1993, to appear).

Murdocca, M. J., V. Gupta, and M. Majidi, "New Approaches to Digital Optical Computing Using Parallel Optical Array Logic," book chapter for *Photonics in Switching*, vol. I, J. Midwinter, ed., Harcourt, Brace and Jovanovich, pp. 195-223, (1993, to appear).

Gerasoulis, A. and T. Yang, "On the Granularity and Clustering of Directed Acyclic Task Graphs," accepted for publication in *IEEE Transactions on Parallel and Distributed Systems*.

Stone, T. and J. Battiato, "Optical Array Generation and Interconnection Using Birefringent Slabs," accepted for publication in *Applied Optics*.

7.2. Presentations

Murdocca, M. J. (presented by Saul Levy), "Computer-Aided Design of Digital Optical Computers Using Free-Space Interconnects," *SPIE Optics in Complex Systems*, Garmisch, Bavaria, (1990).

Murdocca, M. J., "High-Level Design of Digital Computers Using Optical Logic Arrays," *Annual SPIE Symposium*, Orlando, (Mar. 1991).

Murdocca, M. J. (presenter), V. Gupta, and M. Majidi, "A Hardware Compiler for Digital Optical Computing," *OSA Topical Meeting on Optical Computing*, 1991 Technical Digest Series, Salt Lake City, (1991).

Murdocca, M. J. (presenter), and S. Levy, "Design of a Gaussian Elimination Architecture for the DOC II Processor," *SPIE 1991 O/E-Lase Symposium*, San Diego, (Jul. 1991).

Gerasoulis, A., 10th Parallel Circus, Oak Ridge National Lab, (Oct. 1991).

Stone, T., "Quadratic Optical Pulse Compressor," *Opt. Soc. Am. Annual Meeting*, San Jose, California, (Nov. 3-8, 1991).

Stone, T. (presenter) and J. Battiato, "Spot-Array Generation and Optical Interconnection Using Birefringent Crystals," *Opt. Soc. Am. Annual Meeting*, San Jose, California, (Nov. 3-8, 1991).

Gerasoulis, A., *ACM/IEEE Supercomputing '91*, Albuquerque, New Mexico, (Nov. 1991).

Murdocca, M., "Architectural Implications of Reconfigurable Optical Interconnects," presented at the Boulder Workshop on Optical Interconnects, (Feb. 1992).

Murdocca, M., "Architectural Implications of Optical Computing," *12th GI/ITG Conference on Architecture of Computing Systems*, Kiel, Germany, (Mar. 1992).

Murdocca, M. J. (presenter), and A. Sharma, "An Application of Optically Reconfigurable Interconnects to the Dataflow Parallel Computing Paradigm," *SPIE Advances in Optical Information Processing V*, Orlando, (1992).

Gerasoulis, A., *IEEE Scalable High Performance Computing Conference SHPCC-92*, Williamsburg, (Apr. 1992).

Gerasoulis, A., *ACM International Conference on Supercomputing (ICS92)*, Washington, (Jul. 1992).

Murdocca, M., "A Case for All-Optical Digital Computing," *OSA Topical Meeting on Optical Computing*, Palm Springs, (1993).

8. PATENT DISCLOSURES

The following patent disclosures were made during the course of the effort:

T. Stone, "Birefringent Array Generator and Birefringent Optical Interconnector."

T. Stone, "Achromatic and Dispersive Retarders and Compensators."

The Rutgers Office of Corporate Liaison and Technology Transfer did not carry either disclosure through to a patent.

9. TECHNOLOGY TRANSFER, AND COORDINATION WITH OTHER INSTITUTIONS

9.1. AT&T Bell Labs

At the beginning of the effort, Stone worked for several weeks at AT&T Bell Labs in the Solid-State Technology Center (STC) facility in Breinigsville, Pennsylvania. This interaction provided us with expertise in S-SEED devices, which influenced our work on diffractive optical interconnects. After visiting Scott Hinton's Photonic Switching group at Bell Labs in Naperville, Illinois, in a coordinated effort with members of the Photonics Center, Hinton's group made their binary phase grating spot array generation technology available to us. We may use the binary phase gratings in conjunction with Stone's birefringent interconnection methods in the Photonics Center processor.

9.2. Rome Laboratory

The design tool development has moved into a transition phase in which practical circuits are being designed. In a separate effort, we are using the tools to design and simulate a small decoder circuit for an S-SEED testbed processor that is under development in the Photonics Center at Rome Laboratory. Our work on birefringent array generation has also been transitioned to the Photonics Center, where RL personnel have participated in demonstrating the technique.

9.3. OptiComp Corp.

Murdocca and Gupta visited Peter Guilfoyle's group at OptiComp in October 1990. A collaboration was initiated, and as a result, Murdocca created an architecture and a pipeline scheduling for a modified DOC II processor that inverts a system of linear equations in 10 unknowns (see [Murdocca and Levy, 1991] in Section 7.1). The intended application is null steering for phased array radar.

We believe an opportunity for technology transfer exists for our work on functional locality. The OptiComp DOC II project makes use of a spatial light modulator (SLM) that implements only a subset of a RISC instruction set. Our analyses demonstrate that this approach is not only feasible, but that an architectural gain can be realized if only 10% of the instruction set is implemented at a time. Further, our analyses are based on a first-in/first-out (FIFO) strategy for instruction replacement, which corresponds to the FIFO strategy used in the DOC II project. We described early results to P. Guilfoyle of OptiComp in the Spring of 1992.

9.4. NEC Research Institute

During the last year of the effort, we created a formal collaboration with Dr. Eugen Schenfeld of NEC Research Institute in Princeton. The collaboration has been underway since July 1992, primarily between Gupta and Schenfeld, in the area of reconfigurable optical interconnects. Since the collaboration began, Gupta has focused on a model of reconfiguration that makes use of a large but slowly changing reconfigurable optical interconnect that connects a number of small but fast electronic reconfigurable interconnects. Many parallel applications exhibit **switching locality** in which each process tends to communicate directly with only a small set of other processes for a given interval of time. An Interconnection Cached Network exploits switching locality to gain high performance. In such a network the set of communicating processes is partitioned into small clusters. Processes in the same cluster communicate over small, fast electronic switching networks. A slowly reconfiguring high-bandwidth optical network interconnects processes in different clusters. Efficient embeddings of communication structures can be obtained for such a network to ensure that the large, slowly reconfiguring network need not switch very often, which is the area of work where Gupta is concentrating his efforts.

9.5. Siemens Corporate Research

During the last year of the effort, we created a formal collaboration with Siemens Corporate Research in Princeton. The Optical Computing Research group at Siemens is supporting a Ph.D. student, who is following up on the reconfigurable architecture work described in Section 4. The Siemens group intends to construct a prototype optoelectronic processor and is using our reconfiguration model as the basis for their demonstration.

10. PLANS FOR FUTURE WORK

Although we intend to continue all of the work that we have reported here, we have made plans to pursue two specific directions, which are described below.

10.1. Design Tools

An extensive software task lies ahead in making the design tools more user friendly, and in refining the tools to conform to practical considerations that arise in laboratory experiments. Some of the practical problems that have been discovered that the tools do not address are:

(1) The need for a simulator that supports functional modeling and debugging of logic circuits.

(2) The need to XY-fold rectangular circuit designs (this is the shape that the CAD tools produce) onto square logic arrays.

(3) The need to Z-fold circuit designs into the linear order supported by arrays (*e.g.* OR-OR-NOR-repeat may be all that is supported when OR-NOR-OR-NOR is needed). A related issue is the need to automate the mapping of associative logic (such as AND and OR) onto non-associative logic arrays (such as NOR).

Although we would like to address these practical considerations at Rutgers, it poses a considerable time investment. Instead, the tools are being upgraded through the help of two programmers who are being supported through a Phase II SBIR effort at Rome Laboratory (Project Engineer is Robert Kaminski, 315-330-4092).

10.2. Reconfigurable Optical Interconnects

Our discovery of functional locality in ordinary programs motivated us to organize an effort that exploits this property. As a result, we were recently awarded a three-year NSF grant to investigate an architecture for a digital optical processor that reconfigures itself to match the natural form of the computation. Our approach is to write programs in a high-level dataflow language such as ID, and then automatically extract a dataflow graph from the program. The dataflow graph, which exposes parallelism in the program, then serves as a template for automatically configuring a parallel processor. The idea is to reconfigure the interconnects among the processing elements (PEs) to implement the dataflow graph directly, as if a custom piece of hardware is being created to implement the high-level program. This approach is novel, because conventional electronic architectures are forced into mapping dataflow graphs onto physical architectures that are not well-suited for the computation at hand. We plan on using interconnects such as the perfect shuffle, banyan, and crossover among PEs, and so our work on decomposition and MSI interconnection will be used here.

A second NSF sponsored effort has recently started, in which we are exploring a memory with an optically reconfigurable decoder tree, and so the decoder tool will be used here. The memory concept allows arbitrarily shaped data objects to be manipulated in constant time. The idea is to first reconfigure the decoder tree so that the irregular object looks regular (fills a sub-branch of the

decoding tree), followed by a parallel read or write to the sub-branch, followed by a final step that restores the object to its irregular form. This is extremely difficult to achieve with a conventional electronic approach due to the need for parallel access to the decoder tree. A free-space optical approach, however, may support the needed parallel access. We have coupled this idea to Optoelectronic Data Systems (ODS) of Boulder, Colorado, which has submitted an SBIR proposal to ARPA to pursue an acousto-optic implementation of the reconfigurable decoder.

11. CONCLUSION

The most significant accomplishments for the effort include: (1) the development of novel automated layout methods for regularly interconnected optical circuits; (2) the development of an interactive design tool and supporting methods for designing circuits interactively for both regular and irregular interconnects; (3) the creation of two methods for bypassing faults in optical device arrays; (4) the discovery and characterization of functional locality in ordinary computer programs, which is applied to hardware caching; (5) the creation of a trade-off study involving optical interconnects and architectural complexity; and (6) the development of a novel method of optical spot array generation and interconnection. A transfer of technology took place to Rome Laboratory for our work in birefringent interconnects and spot-array generation, which resulted in one of two patent disclosures for the effort.

A few of the lessons learned during the course of the effort are: (1) Regular interconnects, such as crossovers, can be maintained at every level of a computer architecture, from the logic gates up to the system level; (2) The expense of maintaining strict regularity can introduce significant cost in gate count. It appears from initial studies that a few irregular connections placed in selected positions in a circuit can have a significant impact on the size of the circuit being designed; (3) As an alternative, a greater fan-out than two, such as four, can have a significant impact on reducing circuit depth. A superlinear improvement is sometimes possible for the regular interconnect model, since the increased fan-out eliminates problems associated with blocking. (4) Irregular interconnects can be achieved with diffractive optical elements. However, our studies show that there is a trade-off between lens size and propagation distance. A completely irregular interconnect will effectively require a separate imaging system for each optical signal, and the resulting propagation distance may not allow for steep angles of incidence, thus complicating the realization of a completely irregular interconnect. A mix of regular and irregular interconnects appears to be a reasonable compromise when the trade-offs among the optics and architecture are considered together. One rule of thumb that we consider is to use regular interconnects for clusters of signals, 16×16 for instance, and then use irregular interconnects between clusters. In this way, propagation distance can be increased while simultaneously reducing the circuit depth that is attributed to the regularity.