Complexity of optical computing paradigms: Computational implications and a suggested improvement

Post, Arthur David, M.S.

The University of Arizona, 1992

COMPLEXITY OF OPTICAL COMPUTING PARADIGMS: COMPUTATIONAL IMPLICATIONS AND A SUGGESTED IMPROVEMENT

by

Arthur David Post

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfilment of the Requirements
For the Degree of

MASTER OF SCIENCE

WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 9 2

STATEMENT BY AUTHOR

SIGNED: [illegible]

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

[illegible]
Ahmed Louri
Assistant Professor of
Electrical and Computer Engineering

04-28-92
Date

ACKNOWLEDGMENTS

I would like to express my appreciation to Dr. Ahmed Louri of the Electrical and Computer Engineering Department of the University of Arizona for suggesting this area of study, guiding my research, and for introducing me to many of the leading areas of research in Computer Architecture.

I would further like to extend my thanks to Udi Manber of the Computer Science Department of the University of Arizona for many stimulating discussions. Thanks also to Abjit Mahalanobis, formerly of the Electrical and Computer Engineering Department of the University of Arizona for reviewing several of the arguments.

DEDICATION

I would like to dedicate this thesis to my parents for their support, my wife for her understanding, and my children for their wonder.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Optical computing has been suggested as a means of achieving a high degree of parallelism for both scientific and symbolic applications. While a number of implementations of logic operations have been forwarded, all have some characteristic which prevents their direct extension to functions of a large number of input bits. This paper will analyze several of these implementations and demonstrate that all these implementations require some measure of the system (area, space-bandwidth product, or time) to grow exponentially with the number of inputs. We will then suggest an implementation whose complexity is not greater than the best theoretical realization of a boolean function. We will demonstrate the optimality of the realization, to within a constant multiple, for digital optical computing systems realized by bulk space-variant elements.

CHAPTER 1
Introduction

The need for faster computers to increase computational performance is great, and numerous research activities targeted at achieving increased performance are under way in a wide variety of areas. From the device level to the architectural level to research into new computing paradigms such as neural networks, much effort is being expended to provide computing systems which are faster, highly parallel, or potentially more powerful on certain classes of problems. Within this arena, optical computing has attained a status of late, as one of the more promising areas of research. The apparent advantages of optics have been summarized elsewhere, but bear repetition here.

The ability of optical systems to process large amounts of data simultaneously, the non-interfering nature of photons which allows many signals to occupy the same region of space simultaneously, the high switching speed of recently developed optical devices, all hold out the promise of high speed massively parallel computation devices which could augment, and potentially replace, electronic computers. Optical subsystems have found niches in certain portions of the computing environment: laser printers are ubiquitous; optical techniques have contributed to the field of radar signal processing; fiber-optic networks provide high throughput inter-host communications. The central processing unit of the computer, however, has yet to yield to optical techniques, despite the large amount of research devoted to digital optical computing in the past decade. The causes of this lack of digital optical circuitry in commercially available systems will be explored in this paper, and some potential solutions will be advanced. We will begin with a short history of the field to attempt to gain some perspective on the current state of the art.

1.1 A brief history of optical computing

Shortly following the invention of the laser, the use of optical elements for performing computation was proposed. The potential for high speed computing with such devices was recognized very early, and several researchers proposed systems to achieve digital computation [1, 2]. These systems mostly relied on the use of one laser to quench a second laser, and hence obtain inversion, and other functions. These early hopes were mostly destroyed by the work of Keyes and Armstrong [3], who provided convincing evidence of the infeasibility of such systems, given the technology extant at the time. While these arguments were based on thermal considerations, and as such were technology dependent, the field of digital optical computing became dormant, until the discovery of devices possessing strongly non-linear behavior in the mid–1970's [4, 5, 6, 7, 8, 9, 10].

With the invention of these non-linear optical elements, the field of digital optical computing again became one of the more highly visible research fields. The potential for the fabrication of large arrays of these elements holds the promise of highly parallel architectures which can implement systems hither to unachievable.

1.2 Present status of optical computing.

While a large number of systems have been proposed, and have been demonstrated to achieve a variety of binary logic functions, there has been no demonstration of general-purpose optical computers. In the electronic domain, the conversion from vacuum tubes to discrete transistors to integrated circuits as the fundamental elements from which computers were constructed proceeded almost as quickly as each new technology developed. Further, as the size of components shrank, new techniques such as systolic arrays were developed to solve specialized problems. Within the optical domain, the reverse has been true; optical computing systems initially were applied to specialized problems, and these techniques extended toward performing general purpose computing. We now apparently possess the materials to construct reasonably efficient non-linear devices, and our effort is focused on finding efficient techniques for using them.

But the question of finding efficient ways of using these new optical non-linearities begs the further question of what is efficient, and by what measure. There are many possible ways to measure the efficiency of an optical system. One such measure might be the ratio of power performing useful computing to the total power supplied by the optical power supply. Another measure might be the ratio of the number of gates required to compute a given function in an optimal graphical realization of the circuit to the number of optical non-linearities a particular optical system requires in order to accomplish the same function. This second method has a particular strength, in that the field of boolean function theory is quite well established, and the optimal gate count required to implement a number of specific functions has been established. The field of boolean function theory continues to be an area of active research, and new results are constantly being established for various classes of functions and for functions implemented with restrictions on the structure of their circuit graphs [11, 12].

1.3 Research Approach.

The method used in this research has been to first explain the lack of commercial applicability of seemingly well-established digital optical computing paradigms; to attempt to demonstrate the features of these paradigms which impose unacceptable costs on systems so implemented; and, finally, to attempt to suggest ways in which the limitations of current paradigms can be overcome. In studying the feasibility of the various digital optical computing paradigms, a consistent framework was obtained by analyzing the costs incurred by each paradigm in terms of the complexity of implementing boolean functions under that paradigm. While the choice of a suitable measure of complexity for each paradigm is not difficult, the differences in the natures of the implementations under consideration makes a common measure difficult, and in some cases renders comparisons to the well-established methods of boolean complexity theory problematic. While some of the paradigms under consideration make use of gate functions directly, and hence can be compared to optimal theoretical implementations on the basis of gate count and depth, others employing spatial filtering have no simple correlative in boolean complexity. The costs of such spatial filtering systems can be quantified, however, and a functional correspondence obtained which can be compared to cost measures of other systems.

Having examined the principal digital optical computing paradigms, and described the associated cost increase as a function of the number of binary inputs, commonalities were discerned, and attributed to the common method of imaging which is used in these methods. These paradigms all attempt to employ spatially invariant imaging rather than spatially variant holographic imaging, apparently for the relative ease of implementing lenses, beam-splitters, mirrors, and the like. The thesis of this paper is that while such implementations avoid the costs of holographic imaging devices, they incur a high cost when attempting to implement boolean functions given the restrictions imposed on inter-gate routing by such spatially invariant devices.

To demonstrate this thesis, a fairly simple spatially variant imaging technique is postulated, and it is shown that with this technique, boolean functions can be realized having a cost both in number of gates and depth consistent with the best theoretical realization. In contrast, it is shown that the various spatially invariant paradigms can not approach these theoretical limitations, and yield implementations of several important functions which are not practically implementable.

1.4 Organization of the Thesis.

This thesis will investigate the complexity of boolean functions implemented by space invariant optical systems in order to compare the complexity of such systems to that of theoreticly optimum systems. We will show that almost all functions are more expensive to implement in space invariant optical systems, and that for many functions, space invariant optical implementations are very much more expensive. We will then propose a simple spatially variant system, and demonstrate that the complexity of this system is on the order of that obtainable in an optimum implementation. That spatially invariant optical systems are actually more expensive to implement than spatially variant systems may seem paradoxical at first: holograms implementing arbitrary interconnections are difficult to construct, and lenses, prisms, beam splitters, and masks are quite simple and relatively inexpensive. We will show that the nature of these elements imposes constraints on the circuit graphs which implement these functions, and that it is these constraints which cause spatially invariant systems to possess a greater complexity than is required in spatially variant systems.

The rest of the thesis is divided as follows: Chapter 2 will analyze a variety of proposed optical implementations of binary logic in terms of the complexity in some measure of each system, whether number of images, space bandwidth product, or gate count, and will demonstrate that all these systems require an exponential complexity in some measure of system cost, or that the number of functions which are implementable in the system is small compared to the number of binary functions. Chapter 3 will derive a method for constructing any of the two input functions using optical non-linearities as the basis function NOR, and bulk spatially variant elements. We will demonstrate, however that this system, when extended is actually spatially variant. Chapter 4 will extend this development to more complex functions, and will argue that the complexity of this implementation is no worse than that of a computation graph implementing the same functions. Chapter 5 will present some circuits which, due to the complexity constraints of earlier techniques, have not here to for been presented. Chapter 6 will summarize our conclusions, and suggest some areas for further research.

CHAPTER 2

Complexity Analysis of Several Optical Computing Paradigms

This section will analyze several of the proposed implementations of digital optical computing systems from a system complexity viewpoint, and will show that most of these systems are remarkably similar in system complexity. The systems we will detail are Symbolic Substitution Logic, implemented by additive methods and by convolution or correlation techniques, Shadow-Casting Logic, and the Programmable Logic of Murdocca, et al. The Combinatorial Logic Based systems of Guilfoyle and Wiley are considered for completeness. We will analyze these implementations in terms of their suitability for general purpose computation, especially those functions which must compute a result for a large number of inputs. This analysis will reveal that these systems share common undesirable characteristics, which impose large lower bounds to the complexity of the functions implemented.

While it might be argued that our choice of boolean function implementations is unsuitable for the optical domain, we would note that some model of what it means to compute is required, and that the two major bodies of study in computation center around Turing

Machines and boolean functions. Absent some other model of computation, we must formulate our theories of optical computation around these. Since the optical techniques we wish to study implement boolean functions, we will restrict our discussion to this model.

2.1 Symbolic Substitution Logic

While some might argue a somewhat earlier or later date, we can take as a starting point of the current interest in optical computing the publication of Alan Huang's paper "Parallel Algorithms for Optical Digital Computers." [13] From this paper grew the field of Symbolic Substitution Logic or SSL. In rapid succession came a number of papers detailing a variety of implementations of SSL [14, 15, 16, 17, 18, 19, 20]. The basic idea behind SSL is that one first detects the presence of one or more patterns in the input plane, and then substitutes some appropriate pattern for each detected pattern. This process is shown schematicly in figure 2.1.

Two primary methods for performing SSL have been advanced, both based on spatially invariant optics. The first method, and the oldest, known as Additive Logic, involves making copies of the data plane, shifting these copies, and superimposing the shifted copies to determine where in the data plane certain patterns occur. This process is also frequently referred to as a split-shift-recombine technique. The second implementation of SSL involves filtering in the spatial frequency domain, and is commonly known as spatial filtering logic. We will deal with these techniques separately.

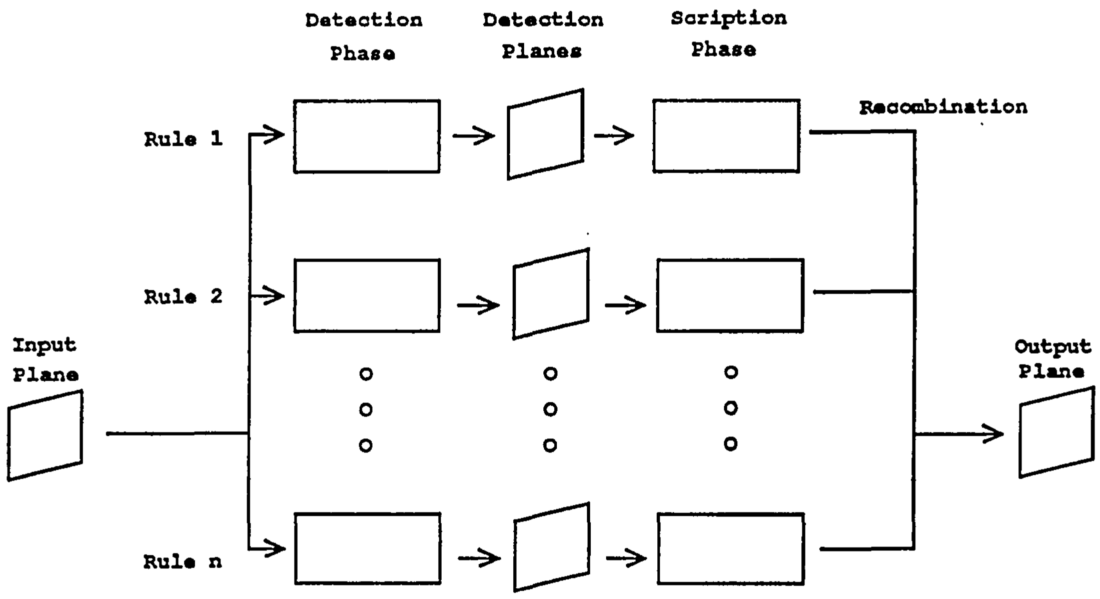


Figure 2.1: Principle of operation of Symbolic Substitution Logic (SSL).

2.1.1 Additive Logic

Data is typically represented in additive logic based SSL in a dual-rail intensity-coded format, wherein two pixels are used to encode each bit, although other codings are possible. For example, the rules which would be used to implement the NOR function are shown in figure 2.2. The application of these rules to a 4–by–4 pixel plane are shown in figure 2.3. The locations of these patterns may be defined by masking, or by providing detectors only at those locations to which the origin of the pattern is mapped. Thus in the conventional split-shift-recombine technique, the origin of the pattern is that location in the detector planes to which the dark pixels of each bit are mapped. When all the pixels mapped to a given detector are dark, the output of the detector will be a high intensity. If the combination of bits present in some pattern does not match the pattern which is to be detected, at least one of the pixels superimposed on the origin will be light, and the output of the detector will be low intensity or dark. In the ensuing substitution phase, the output pixels are shifted in a manner which realizes the appropriate output for the combination detected by a given plane of detectors. Those detectors upon which one or more light pixels have been imaged, being dark, will contribute no light responses to the substitution phase, so that when all resulting substitution patterns are merged, only the substituted patterns from the light detectors will be present.

The disadvantage to this approach is that for a given number, n, of inputs coded in any arbitrary manner, the total number of combinations of these inputs is 2^n. In order to implement completely specified digital logic, each of these 2^n combinations (corresponding to the 2^n possible minterms) must be generated. That this is so, may be seen from

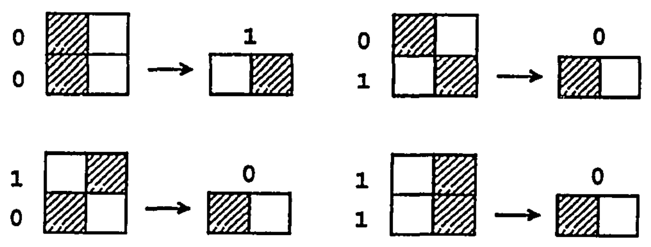


Figure 2.2: Rules for computing the NOR function.

the following argument. Suppose that less than the 2^n combinations are generated in space and imaged onto the corresponding detection planes. Then there will exist some combinations of inputs which will not be detected by the system. For these combinations, no determinate output will be generated, and some indeterminate value will result. But in order to implement digital logic correctly, even input combinations which are "don't care" combinations must be mapped to either the "1" or "0" value. In the typical dual-rail coding scheme, the result of not detecting all input combinations is the substitution of two dark pixels at that position in the output plane at which the function is to be computed. But in the dual-rail binary logic encoding, two dark pixels are undefined. Hence in this implementation, not specifying the output values for some combinations yields a mapping to a value outside of the set 0,1. If the data is to be processed simultaneously, 2^n image planes must be formed at discrete locations in space. If, on the other hand the data is to be processed in sequence (one combination at a time) the time required will increase exponentially.

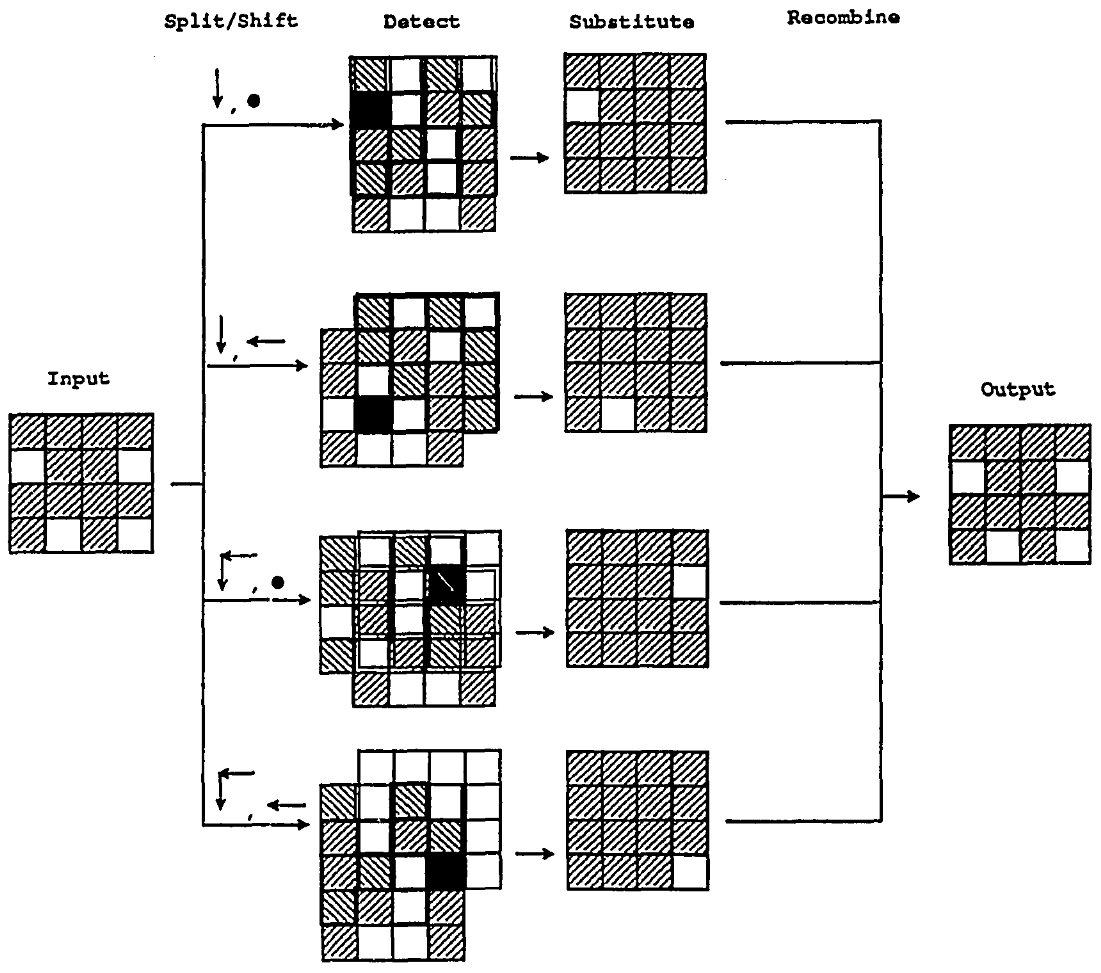


Figure 2.3: Application of the rules for the NOR function to a plane of 4 * 4 pixels.

This result is due to the nature of the paradigm, and not a function of the encoding scheme used to represent the data, since the paradigm requires the detection of all possible input patterns. Thus an intensity encoded SSL implementation of a completely specified boolean function of two inputs will need to detect each of the four patterns: { (dark,dark), (dark,light), (light,dark), (light,light) }, in order to be able to generate the appropriate output for each of these combinations. This observation extends to functions of larger numbers of inputs in the same manner as for the dual-rail encoding. While an intensity encoding can achieve a two-fold reduction in the area of the input plane required to represent the inputs, intensity encoding can not reduce the number of detection operations required to implement completely specified boolean logic.

The actual number of rules required by an SSL implementation may be greater than that shown above. The extra complexity comes from the necessity of providing extra rules to allow for the separation of data vectors or regions of computation from one another.

2.1.2 Spatial Filtering

The second method for implementing SSL is based on convolution/correlation techniques, wherein the input plane is transformed to its Fourier domain representation, then filtered, and the inverse Fourier transform taken of the filtered image. [20, 21, 22] The principle is the same as that depicted in figure 2.1. A number of input codings may be used: Casasent and Botha [21] suggest a dual rail coding similar to that discussed for the additive logic implementation. Brenner, Lohmann, and Merklein [20], and Weigelt [23] suggest several alternate codings which achieve a modulation of the spatial frequency to

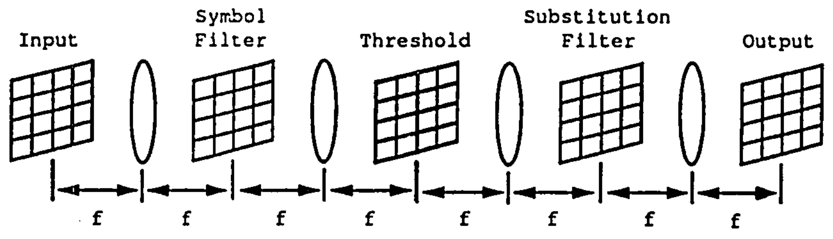


Figure 2.4: The optical system for SSL suggested by Casasent and Botha.

represent the data. The optical system employed by Casasent and Botha [21] is shown in figure 2.4, while that of Brenner, Lohmann, and Weigelt is shown in figure 2.5. What is common to these techniques is that in the implementations of the two-input functions outlined in these papers, four filters are required to detect the input combinations and substitute the appropriate response since there is one filter for every combination of inputs. Should we extend these methods to functions of a higher number of inputs n, the number of filters required in each of these schemes will have to be 2^n. These methods are not fundamentally different than additive logic, even though their outward appearance suggests otherwise.

To see that this is so, we will first analyze the method employed by Weigelt [22]. Weigelt uses for the inputs to the function, a pair of planes having the same optical axis, with each pixel in the first plane imaged onto a corresponding pixel in the second plane. The data encoding in each of these pixels is such that pixels in the first plane affect a shift in spatial frequency in the x direction, and pixels in the second plane affect a shift in spatial frequency in the y direction. Thus the overall spatial frequency of the light emerging from the second plane is a linear combination of the spatial frequency shifts of the two planes. It is this overall response which is then transformed to the spatial frequency domain. In order to facilitate the ensuing discussion, we reprint the following equation from [23], which Weigelt derives as the response of the system in the Fourier plane:

$$\begin{aligned}
\tilde{U}_2(\nu,\mu) &= F[U_2(x,y)] = (a^2 sinc[(\nu-\nu_1)a] sinc[(\mu-\mu_1)a] \\
&\times \sum_n \sum_m \exp\{-2\pi i[(\nu-\nu_1)x_n + (\mu-\mu_1)y_m]\} W^{(1)}_{nm} V^{(1)}_{nm}) \\
&+ (a^2 sinc[(\nu-\nu_1)a] sinc[(\mu-\mu_0)a] \\
&\times \sum_n \sum_m \exp\{-2\pi i[(\nu-\nu_1)x_n + (\mu-\mu_0)y_m]\} W^{(1)}_{nm} V^{(0)}_{nm}) \\
&+ (a^2 sinc[(\nu-\nu_0)a] sinc[(\mu-\mu_1)a] \\
&\times \sum_n \sum_m \exp\{-2\pi i[(\nu-\nu_0)x_n + (\mu-\mu_1)y_m]\} W^{(0)}_{nm} V^{(1)}_{nm}) \\
&+ (a^2 sinc[(\nu-\nu_0)a] sinc[(\mu-\mu_0)a] \\
&\times \sum_n \sum_m \exp\{-2\pi i[(\nu-\nu_0)x_n + (\mu-\mu_0)y_m]\} W^{(0)}_{nm} V^{(0)}_{nm})
\end{aligned}$$

where:

- ν_0, ν_1, μ_0, μ_1 are the spatial frequency shifts in the x and y directions respectively, corresponding to the encoding of 0 or 1 in either of the two planes.
- a is the dimension of an individual pixel in either plane.
- $W^{(0)}_{nm}$,$W^{(1)}_{nm}$,$V^{(0)}_{nm}$,$V^{(1)}_{nm}$, are multiplicative constants which are 0 if the particular pixel was not encoding (0) or (1) in the first (W) or second (V) plane. The use of

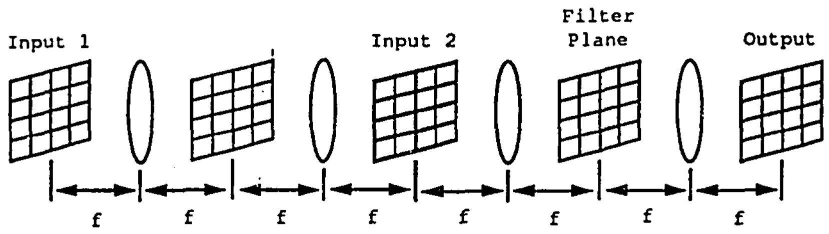


Figure 2.5: Implementation of SSL in Brenner and Lohmann, and Weigelt's methods.

these factors merely reflects that for a particular pair of pixels encoding some pair of input bits, the resulting response in the Fourier plane is contained within the corresponding sinc function, and is a mathematical convenience.

Now as can be seen, the resultant response in the Fourier plane is a set of four sinc functions which envelope a set of complex exponentials. The complex exponentials form a fine structure or speckle pattern in the Fourier plane, which Weigelt omits from her discussion, as being unresolvable in the output plane. But it is this fine structure which contains the information about which pair of cells produced a particular response, encoded in the frequency of the complex sinuoid in the Fourier plane. A given cell (x_n, y_m) will be represented in the Fourier plane as a complex sinusoid having frequency of x_n in the ν direction, and frequency y_m in the μ direction. It is exactly this information which is required to construct an appropriate response at the appropriate point in the final output plane.

Now in order to filter this information correctly, we will need to construct a filter response for each of the sinusoids which might occur under each of the four sinc envelopes. Thus the filter must provide a correct response even in the worst case of all pixels in each of the two input planes having the same coding (ie: $\forall n, m : x_n = a, y_m = b \mid a, b \in (0,1)$). Now we must sample in accordance with the Nyquist sampling theorem, and provide at least two samples per period of the highest frequency sinusoid present. But the highest frequency sinusoids in the Fourier plane are generated by the most extreme pixels in the input plane, and this frequency is proportional to the number of pixels in the x or y direction of the input plane. Thus we must have a number of samples in each filter proportional to the number pixels in the input plane. But if the area of the Fourier plane is to be the same as the area of either of the two input planes, then the area of any sinc must be on the order of 1/4 the area of the Fourier plane, and the entire filter corresponding to that particular combination of inputs must fall entirely within this area. Thus the space bandwidth of each filter must be on the order of four times that of either of the two input planes.

The extension of this argument to a greater number of input planes is direct. Ignoring limitations on the maximum spatial frequency shift which is possible, we may see that allowing a third input plane encoded so as to produce a shift in spatial frequency in the (say) x direction, will result in a set of eight sinc envelopes; a fourth plane encoded to achieve frequency shift in the y direction results in sixteen sinc envelopes, and so on. Thus extending the technique to allow for n input planes results in the requirement to provide 2^n filters, each of which must be capable of providing a correct response for its appropriate combination of inputs. Further, if these filters are to be fit within the same area as the input planes, then the spatial bandwidth must increase as 2^n as well.

Returning to the system of Brenner, Lohmann, and Merklein [20], we may see that their method differs from that of Weigelt's only in small detail, but not in the fundamental requirements imposed upon the filters, in terms of number or space bandwidth. The method proposed by Casasent and Botha [21] does not encode the data in spatial frequency but in a dual-rail intensity format. These authors further extend their system to allow for the computation of any of the two-input boolean functions. But this method also falls on the same difficulty evidenced in the other systems. That is, for any function which they implement, they require four separate filters spatially multiplexed, yielding a total space bandwidth for the composite filter of four times that of the input data plane. This method further increases the space bandwidth product by spatially multiplexing sets of filters for a number of different functions within the same overal filter.

In the technique proposed by Jeon et al. [24] the filters are separated in space, but one is required for each pattern detected. Thus the spatial filtering techniques require that the number of filters, and possibly the spatial bandwidth of the filters, increase exponentially with the number of inputs to correctly implement boolean functions, just as the number of detector planes increases exponentially in Additive Logic–based SSL.

2.2 Shadow-Casting Logic

A second implementation of digital optical computing, due to Tanida and Ichioka [25, 26, 27, 28], is known as shadow-casting logic. While seemingly quite different than

SSL, the discussion that follows will show that the complexity of this implementation is quite similar to that of SSL. Additionally, we will show that the system is incapable of use as a stand–alone digital optical computing system.

2.2.1 General Description.

In this method, shown schematicly in figure 2.6, data is mapped to multiple parallel spatial light modulators (SLM's). These SLM's are composed of liquid crystal light valves, driven by an electronic adjunct which is responsible for mapping the data to the appropriate cells of each SLM. The planes of the SLM's are set physically close to each other and parallel. The data encoding is by orthogonal dual-rail transparent/opaque regions within overlapping cells of the parallel planes. The orthogonality of the parallel codings results in a portion of the resulting combined cell being transparent. Thus, when illuminated, by a light source on the "front" side of the combined input planes, the particular combination of inputs may be detected by the location of the light incident on a detection plane placed behind the input planes.

By having multiple light sources combined with suitable locations for the detectors, any of the n-input functions may be computed. The number of light sources and detectors is dependant on the manner in which the system is to be used. For applications which compute only one function, it is possible to have as few as one light source; other applications may require more sources.

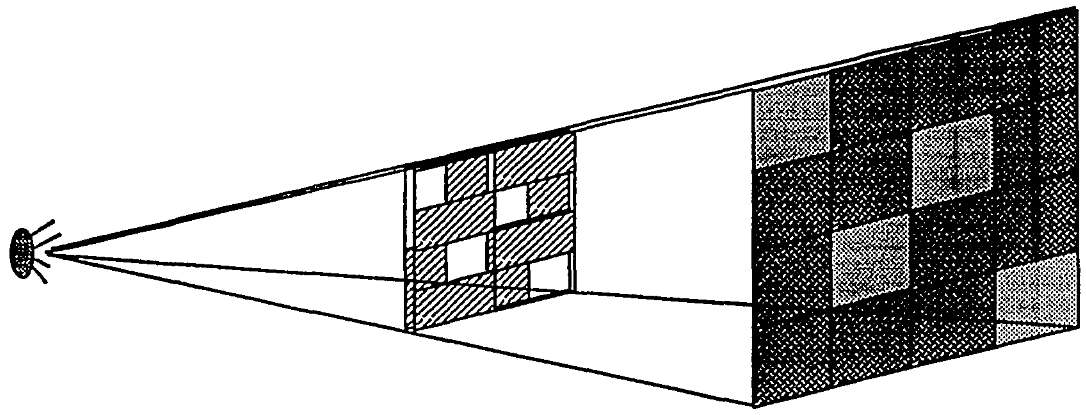


Figure 2.6: The general construction of a shadow-casting logic system.

2.2.2 Complexity Analysis

While this method has the advantage of simplicity in implementation (the light is provided by LED's or other inexpensive sources), the size of the transparent area in each cell decreases exponentially with the number of inputs mapped to each cell. Thus the available power at the detector is only 2^{-n} times the power incident on the cell. That this is so can be seen from figure 2.7, where the encodings required to achieve up to four inputs are shown. From the figure, it should be clear that the transparent portion of the input cell is one-fourth of the cell area for two inputs, one-eighth the area for three inputs, one sixteenth for four inputs and so on.

The exponential complexity of this system extends to the number of light sources also. The reason for this exponential increase is that there is only one detector for each

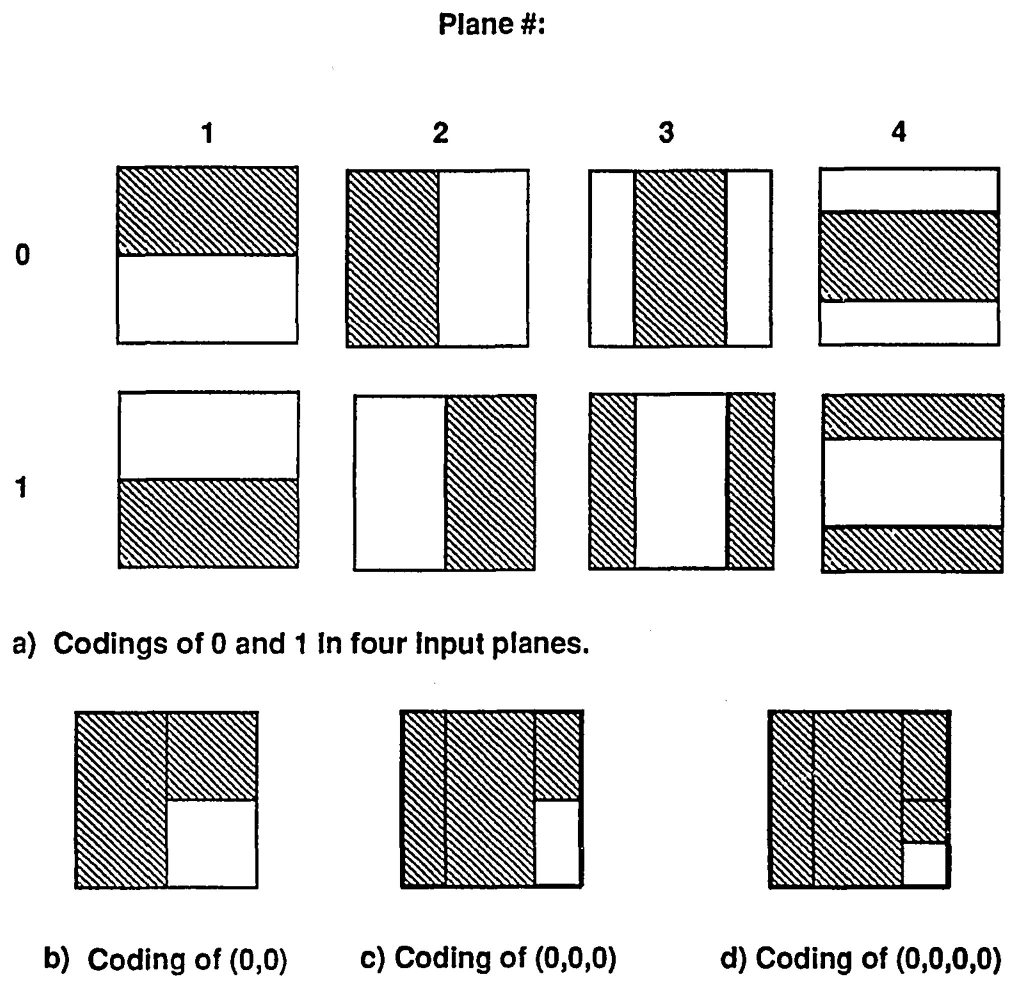


Figure 2.7: The encodings of 0 and 1 for up to four input planes for shadow casting, and the result of their juxtaposition.

cell in the image plane, and the system must be capable of mapping each of the ON-set combinations to this detector. Thus for every function to be computed, there must be a light source capable of illuminating the detector through the transparent region of the input plane corresponding to the particular minterm represented by the light source. Thus, if the system is to be used to compute all the different possible functions, light sources for each of the 2^n combinations of the n inputs must be provided. If, on the other hand, one wished to compute a function such as the AND whose ON-set contains only one of these combinations, then only one source would be required.

This exponential complexity was first described by Arrathoon and Kozaitis [29], who then proposed the use of this technique to provide an associative look-up for multi–valued logic. But while they were able to show some improvement in power and space bandwidth, their system has many of the same drawbacks as the original method.

2.2.3 Other Drawbacks.

The method has the drawback that the output is incompatible with the input, being intensity encoded rather than transparency encoded. Thus the system is not directly extensible, and the outputs must be converted to an electronic signal before being mapped to an additional input plane for further computation. But once the host has accomplished this mapping of data, the computations provided by the optical system are quite simple. Whether the additional complexity required of the host can be balanced by the parallel computation provided by such a system is problematic.

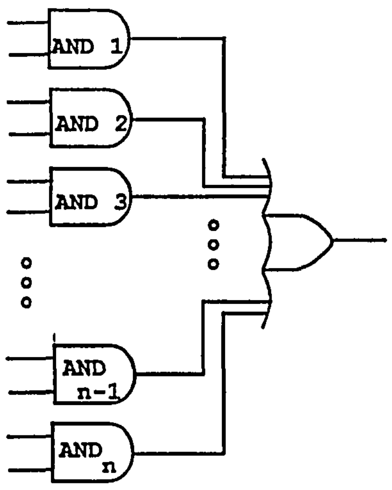


Figure 2.8: The AND–OR structure of the optical subsystem of Guilfoyle and Wiley.

2.3 The "Combinatorial Logic" of Guilfoyle et al.

Another computing paradigm, due to Guilfoyle and Wiley and called Combinatorial Logic, divides the computing between the electronic and optical domains [30].

2.3.1 General Characteristics.

In this method, a pair of multichannel acoustooptic cells are telecentricly imaged, data within the cells counterpropagating. The authors then demonstrate how this structure can be used to achieve an AND-OR structure, and relate their structure to a programmable logic array (PLA) implementation. The resulting AND-OR structure is shown in figure 2.8.

2.3.2 The computational weakness of the spatially–invariant subsystem.

As noted, this model does not perform all computations in the optical domain, and requires a certain amount of electronic logic to compute a number of terms prior to the application of the optical logic. This technique has the advantage that a large number of computations are performed in parallel by the optical devices. It has the disadvantage that the optical logic performs only simple computations. While these authors claim that this system is suitable for general purpose computations, the computational power of the spatially invariant implementation of their system is very weak. The spatially invariant implementation can be modeled as a set of n two input AND gates, whose outputs are connected to the inputs of a n-input OR gate. The inputs to the AND gates are two distinguished sets of n bits each. That this system is inherently weak can be established as follows:

- Since the structure of the AND/OR tree is fixed, the only way to compute a variety of different functions is by making different assignments to the leaves of the tree (the AND gate inputs).
- The number of different possible assignments of the variables in either set of n variables is n!. Thus the total number of possible assignments of the 2n input variables to the 2n inputs of the AND level is $n! * n! = (n!)^2$;
- The number of different possible boolean functions of 2n variables is given by:

$$|\{f_{2n}\}| = 2^{2^{2n}};$$

n	2n	n!	$n!^2$	$2^{2^{2n-2}}$	$2^{2^{2n-1}}$
1	2	1	1	2	4
2	4	2	4	16	256
3	6	6	36	65,536	4,294,967,296
4	8	24	576	2^{64}	2^{128}

Table 2.1: A comparison of the number of functions implementable in the spatially-invariant subsystem of Guilfoyle's technique, and the total number of non–degenerate functions versus the number of input variables.

- The number of degenerate functions of 2n variables (denoted $\tilde{f}_{2n}$) can easily shown to be (see appendix A):

$$|\{\tilde{f}_{2n}\}| \leq \begin{pmatrix} 2n \\ 2n-1 \end{pmatrix} * 2^{2^{2n-1}} = 2n * 2^{2^{2n-1}}$$

- Thus the number of non-degenerate functions of 2n inputs is given by:

$$\begin{aligned} |\{f_{2n} - \tilde{f}_{2n}\}| &\geq 2^{2^{2n}} - 2n * 2^{2^{2n-1}} \\ &\geq (2^{2^{2n-1}} - 2n) * 2^{2^{2n-1}} \\ &\geq 2^{2^{2n-1}} \end{aligned}$$

- Taking the limit of the ratio of the number of functions computable by the AND/OR tree to the number of non-degenerate functions proves the point:

$$\lim_{n\to\infty} \frac{(n!)^2}{2^{2^{2n-1}}} = \lim_{n\to\infty} \frac{n!}{2^{2^{2n-2}}} = 0.$$

That the fraction of the total functions implementable by this method is actually quite small, even for small values of n, can be seen from table 2.1.

Now this AND/OR structure is only part of the system, the larger portion of the logic being performed electronicly (appearantly in ECL). Thus the inputs to the AND/OR structure are more complex functions of the true input variables. Further, the system is capable of performing certain operations at very high speed. But the claims of general suitability of this system are not valid within the spatially invariant optical domain.

We also acknowlege that the authors have implemented a spatially variant system which they claim allows for "global broadcast." The essence of this system is that they are able to allow a greater number of inputs to the gates in the AND level of their implementation. This change to their spatially invariant system will allow for a greater number of functions to be computed by their system, and given enough complexity in their electronic logic, they might be able to implement boolean functions in general. Whether general boolean functions can be implemented by this system depends upon the manner in which the holographic system is used, and the level of complexity allowed in the electronic domain.

2.4 Programmable Logic

2.4.1 General Description

Murdocca, Huang, Jahns, and Streibl have proposed a method of digital optical computing which uses optical non-linearities directly as a form of logic gate [31, 32, 33, 34, 35]. In this method, planes of these logic gates are interconnected by a spatially invariant imaging system constructed from beam splitters and prism arrays, the resulting interconnection pattern forming a crossover network.

2.4.2 Complexity Analysis

The essence of the design technique used in this system is that the n input variables are mapped to the input array along with their complements. A series of n+1 interconnection stages consisting of one cross-over network and one mask per stage are then used, along with n+1 arrays of AND gates, to generate all the minterms of the function. Once the minterms of the function have been generated, the appropriate minterms are combined through a similar n+1 stage network where the AND gate functions have now been replaced by OR functions, to generate the appropriate functional output. The total number of gates required by this method is $\Omega(n2^n)$ where again, n is the number of inputs. That this number of gates is the minimum required is asserted by the authors.

Several claims that are made for Programmable Logic must be discussed however. The authors state that that the size of the circuits realized being $\Omega(n * 2^n)$ in number of inputs, and that the number of levels in the circuit being linear in the number of inputs is "optimal." While it is true that Shannon's theorem demonstrates that almost all boolean circuits have size $\Omega(2^n/n)$, and depth $\Omega(n)$, Shannon noted that this is not true for many fundamental circuits. But Programmable Logic has lower bounds that are exponential in number of inputs, regardless of the function implemented. Thus, while a given function might be computed by a circuit having size $O(n^2)$ and depth $O(\log_2 n)$ [36], Programmable Logic will not allow minimization to this level. Even for random functions whose size is $\Omega(2^n/n)$, Programmable Logic requires $O(n^2)$ more gates than the optimal circuit realization.

The inefficiency of this method is perhaps most graphic when one considers the manner in which the normal two input functions would be constructed. Initially, the designer would provide for a field of 2^{2+1} gates wide, and reserve $2(2+1)$ levels of gates to implement the two input functions. The two inputs and their complements must be mapped to the input level, and the minterms of these two inputs generated at the third level of the network. These minterms would then be combined in the next three levels to achieve the desired function. One desirable function would be the NOR. Now, a minimal realization of the NOR, given that AND and OR gates were available, and that the input variables and their complements were also available, would simply connect the complements of the inputs to the AND; the output of this one gate would then be the NOR of the two uncomplemented inputs. Thus Programmable Logic would require as many as 24 AND gates and 24 OR gates to accomplish what might otherwise be done with one AND gate. Even if the method could be minimized, the resulting gate count would be eight AND gates and eight OR gates, where a single AND would suffice.

Another claim made for Programmable Logic is that the design methodology is similar to that of PLA designs. Now a true PLA design has only two levels of logic regardless of the number of inputs, and does not typically require that all minterms of a function be generated. Rather, the PLA user develops one or more equations (depending on the number of outputs required), which are in minimized sum-of-products form. It is the product terms of these equations which are then mapped onto the PLA structure. While some functions (the parity function for instance), do not admit to minimization of their sum-of-products form, ring-sum expansion will frequently minimize these functions, and allow an economical circuit realization in XOR logic. Now while a minimized circuit is presented which does not generate all minterms and then select from them, the breadth of the circuit is still exponential in the number of inputs.

A third claim is that the regular interconnection scheme proposed for Programmable Logic does not increase the cost of the circuit much. As evidence for this claim, a circuit is presented for the full adder which requires 78 AND gates, 11 OR gates, and six levels, and termed a "minimized serial adder." This is compared to the 128 gates required of Programmable Logic. But the adder shown generates all the monoms (not minterms) of the function, resulting in a circuit which is not minimal. The minimal full adder circuit requires only five gates (2 XOR, 2 AND, 1 OR), and has a depth of three. An alternate realization of the full adder has a gate count of six (2 XOR, 3 AND, 1 OR) and a depth of two. Even when restricted to the use of NOR gates alone, a minimized full adder should require no more than 14 such gates (although the resulting depth is 7), whereas the smallest circuit for the full adder under Programmable Logic requires 128 gates of both AND and OR types. While it is possible this might have been reduced to 48 AND and OR gates, the resultant realization would still not be minimal.

Another claim made for this system is that it is synchronous, and hence computations can be pipelined through it. While this is true, any general circuit can be converted into a synchronous circuit by equalizing all paths in the circuit through the insertion of delay elements into any edge bypassing a level. This technique will not increase the depth of the resulting circuit, and will increase the complexity modestly (again see [36]).

2.5 Commonalities of the above implementations

We have examined several implementations of digital optical computing, and have found that all these systems require an exponential increase in the complexity of some parameter of the system as the implementation is used to compute functions of increasing numbers of inputs. This exponential increase impacts the feasibility of systems implemented by these means in a direct and profound way; the extensibility of the implementations is not feasible beyond a small number of inputs. These systems also lack the capability to interconnect even the small functions which have been implemented in the arbitrary manner which is required in order to construct larger functions. This lack of interconnect capability is both the root cause of the inherent exponential complexity of these systems and the constraint which prohibits the economical construction of large functions from the smaller basis functions. It is the rigid mapping of input to output imposed by spatially invariant imaging systems, as presently used, which has limited the application of these techniques.

In real computers we are confronted by functions which require the computation of a large number of outputs as functions of a large number of inputs, especially in the central control units. That the construction of such control functions is difficult is testified by the use of microprogramming in many modern computers. Such microprogramming requires the use of memory, however, and is consequently impractical for optical computing at the present time, given the lack of optical random access memory. Thus the only option for implementing the control functions required for complex computations in the optical domain seems to be the use of boolean logic.

In order for optical computing to be incorporated in computer design techniques it must provide a significant increase in functionality without too great a cost. If optical techniques are used as an adjunct to electronic computing, then the cost of adding these systems into the electronic computer must not be too extreme. An example of this can be found in the sequential full adder. It has been hypothesized that an optical implementation of the full adder could perform on the order of 100,000 adds in one step. But what is the cost in the electronic domain? The electronics must be able to support a pair of shift registers for all 100,000 pairs of data, must be able to synchronize the output of data and the input of results, and must be able to route these values to the correct locations within the physical space of the input and output electro-optic devices. Having thus encurred these costs, an optical full adder would compute a function requiring no more than 14 electronic NOR gates per data pair.

CHAPTER 3

A New Approach

In the previous section, we have seen that the complexity of the various optical computing paradigms which have thus far been proposed prohibits them from being acceptable for truly general–purpose computing. The inability of these systems to compute boolean functions with less than exponential complexity severely limits application of these systems even in computing arithmetic functions in conjunction with electronic hosts. Thus we must look for new ways to achieve digital optical computing which do not suffer the shortcomings of the previous implementations, and still provide the advantages which optical computing promises. To this end, we will first attempt to enumerate the features which would be desirable in an optical computing system, and then try to find a way to provide these features.

3.1 Desirable features of an optical computing system.

Whether optical computing remains an adjunct to electronic computing, or whether general purpose optical computers eventually are eventually developed will depend primarily on whether optical random access memory is developed. Since memory is not currently available, we will restrict our discussion to the use of optical computing as an adjunct to the electronic domain. In this regard then, we will enumerate several features which will be required of an optical subsystem of a computer. Our arguments will concern only architectural and register transfer issues, and will not concern questions involving device physics, compatibility of sources and detectors, power requirements and heat dissipation, or any of a number of other questions which still must be solved, but are outside the scope of this paper. We will keep our discussion as general as possible in order to allow for the possibility that new technology will not invalidate our conclusions.

3.1.1 Use of non–linear optics directly as logic gates.

In our discussion of symbolic substitution we noted that the implementation of the NOR function would require four planes of detectors whose functionality was essentially that of the NOR itself. Thus, we had the situation that four times as many gates were used as was necessary. We would wish rather to directly use the optical elements as logic gates, and without the large number of excess gates imposed by Programmable Logic. Essentially, we would like to implement the functions required of an ALU in a manner which does not impose a severe cost in unused gates.

3.1.2 Input and output.

Obviously, in order to access an optical computing subsystem, there will be a need for electro–optic transducers; the manner in which data is passed through these transducers is critical. We have previously argued that mapping data in a sequential manner through an optical subsystem, as would be the case with an optical sequential adder, will complicate the electronic subsystem more than the added speed and parallelism provided by the optics could compensate. If for every register in an ALU there is only one optical source then we must shift the data out of the register serially, whereas if we supply an optical source for every bit in each register, then we can simultaneously output all registers.

Assuming that the optical subsystem is capable of a resolution of 1024∗1024 pixels, and that each pixel can be used to represent one bit, then parallel mapping of the registers to the optical system would yield a potential of $1024 * 128$ eight bit inputs to the system for monadic operations or $1024 * 64$ pairs of such inputs to the system for dyadic operations. This would still provide a significant increase over the achievable parallelism in electronic systems.

3.1.3 Implementation of large functions.

The kinds of functions which have previously demonstrated by optical systems, are rather small, being generally limited to the binary functions and the full-adder. But once we have gone to the trouble to modify the electronic subsystem of a computer to allow for the use of an optical computing subsystem, we could include these functions to the logic of the electronic subsystem with only a small constatnt increase in gate count per register. For this reason, we would like to have an optical subsystem which performs a "large" amount of computation in entirity, and returns the results to us once the computation is done. Now the question of what is a "large" amount of computation is open to debate, but certainly something on the order of a complete eight bit addition for each register pair is near the lower end of this range. At the higher end of the range of computations which might be desirable, would be computations involving floating point representations.

3.1.4 Parallelism of implementation.

While we would like to map entire registers simultaneously to the optical subsystem, and perform operations such as addition and comparison on pairs of such registers, we would also like to exploit parallelism in the computation of these functions. While we might save some overhead in mapping registers in parallel rather than serially shifting the registers into the system, we would not achieve much increase in speed if the optical subsystem employed serial operations on these registers values. Thus, instead of a serial ripple–carry addition or a serial bit–by–bit comparison, we would desire the optical subsystem to compute addition by means of the carry look–ahead adder, and parallel comparison.

In summary, we wish the optical subsystem to require minimum overhead in the electronic subsystem, to compute functions on large data items simultaneously, and to compute these functions using the greatest amount of parallelism possible within each function.

3.2 Development of such a system.

In order to demonstrate our method we will model space invariant optical optical devices in the following manner. First, we will assume that geometric optical approximations hold. This assumption, while ignoring diffraction effects, provides a best-case for our analysis of lower bounds. While lens systems may be created which rely on the diffraction of light, the transformations so achieved are one-to-one (one pixel to one complex sinusoid), and from an information viewpoint, preserve all the information originally present. Thus while a lensing system may provide for a complex transformation of the input, all the information contained in the input is preserved, though the information contained in a given pixel is spread over a large area. The spatially invariant systems we present also possess this information preserving characteristic. We will be primarily concerned with prisms, and mirrors which possess the characteristic that the rays emanating from the object plane emerge from the device in parallel. It is this parallel nature of the light propagation which must be overcome in order to provide for the kinds of arbitrary interconnection which are characteristic of binary circuits.

Mirrors and prisms are not, by themselves, adequate to allow arbitrary interconnection patterns since their input-output mappings are one to one. This constraint imposes a restriction on the fanout of information contained in their object plane. The fanout so achieved is limited to one. While boolean formulas may be realized by circuits having a fanout of one, the formulas require the variables and their complements appear at the leaves of the resulting AND-OR tree many times, and hence require an arbitrary degree of the input nodes. Thus, even restricting the active circuit to unity fanout from the gates, will not overcome the problem of providing for arbitrary interconnection patterns. The answer to this problem is the beam splitter, which has the capability, in our model, to perform one-to-two mappings. Thus a beam-splitter can take as input an entire object plane and make two replicas thereof. Of course, beam-splitters could then be cascaded to produce a potentially unlimited number of copies of the input image. The obvious problem here is that the intensity of the two output images is half (approximately) of that of the input. We will shortly see that this halving of power level presents no fundamental problems, although practical problems may exist.

We will require two more components for our model. First, since all elements thus far discussed preserve all the information presented to them, we will need a means of selecting only those portions of the information which are of interest at a given point in computation. To this end, we will employ masks which pass only that portion of the image plane which impinges on the mask's transparent regions, and blocks those portions of the image plane which impinge on the mask's opaque regions. While masks are spatially variant over all, they are invariant within the two domains which their transparent and opaque regions define. The last element we require is a two dimensional array of optically non-linear devices. We will use these arrays as our computing elements, and as amplifiers. Such devices have been constructed, and are at the heart of many proposals for digital optical computers. The salient feature of such devices, for purposes of this discussion, are that they occupy one pixel position per device, and that they compute some suitable functions (eg. NOR). The model we will draw of these devices as logic functions will assume that the functions so formed are of a fanin and fanout of two. While the outputs of a plane of these gates could be split many times, the nature of the devices is such that an input can only be recognized as a logic value if its intensity is close to half the intensity of the device's output intensity. This presents no inherent problem, as gates having a greater fanout may be constructed from a constant number of gates having fanout of two by amplifying each image before subsequent splitting.

We now have all the elements required of our model: imaging elements which preserve information through a one-to-one mapping; elements capable of copying the information in their input; elements capable of passing some of their input while blocking other portions; and elements capable of implementing a basis function and amplification. We will now begin to construct some useful circuits from these elements.

3.3 Implementing the basis functions.

3.3.1 The NOR function.

Given an object plane of N = 2n pixels to which data has been mapped, we will begin by demonstrating the two input NOR function. We will compute n of these functions simultaneously. These computations are not only logically parallel, but physically parallel also. We will assume, for the moment, that the pairs of inputs to a given NOR gate are adjacent, and will require that all pairs are distinct, and in the same spatial relation to each other. This requirement is imposed by the spatially invariant nature of the imaging systems we use. The realization of this function is shown in figure 3.1, with a schematic representation of the realization shown in figure 3.2.

In figure 3.1, the input data are mapped into two pairs of rows with all four possible combinations of the inputs encoded in each pair of rows. The values '0' and '1' are shown numerically for clarity; in actuality, the inputs would be intensity encoded. The input array is replicated by means of a beam splitter. The replicated images are then masked so that alternate rows of pixels are blocked in either of the two images. A pair of prisms now recombines the images in such a way that every pair of pixels which are to be NOR'ed are incident on the nonlinear element which computes the NOR function. While the operations performed here could have been more directly implemented by a lenslet array since the data were mapped in adjacent pixels, our purpose is to demonstrate a more general technique which may be used to construct significantly larger functions in which computations may occur between inputs or intermediate results which are not physically adjacent.

The schematic representation of figure 3.2 is derived from the implementation shown in figure 3.1 by taking the "top" pair of rows from the implementation, dividing the rows into columns, and "stacking" the columns to achieve one column of eight elements. We will rely on schematic representations in all further discussions. The schematic representations will show a decrease in active device density with increasing numbers of inputs. This density decreases linearly with the number of inputs, and is not different in this respect than SSL. This decrease is consistent with the mapping of n inputs to one output.

In this regard, we note that the density of the NOR elements is only one–half that of the input plane. In contrast, the density of threshold elements in SSL (for two input functions) is one–fourth the density of the input plane when dual–rail encodings are used. Further, with SSL, one requires four planes of such devices, each essentially implementing the NOR function, to perform the same computation we perform with one such plane. Also, since we are using simple intensity coding, we achieve twice the number of computations with the same size planes. While Programmable Logic fully populates the logic planes with

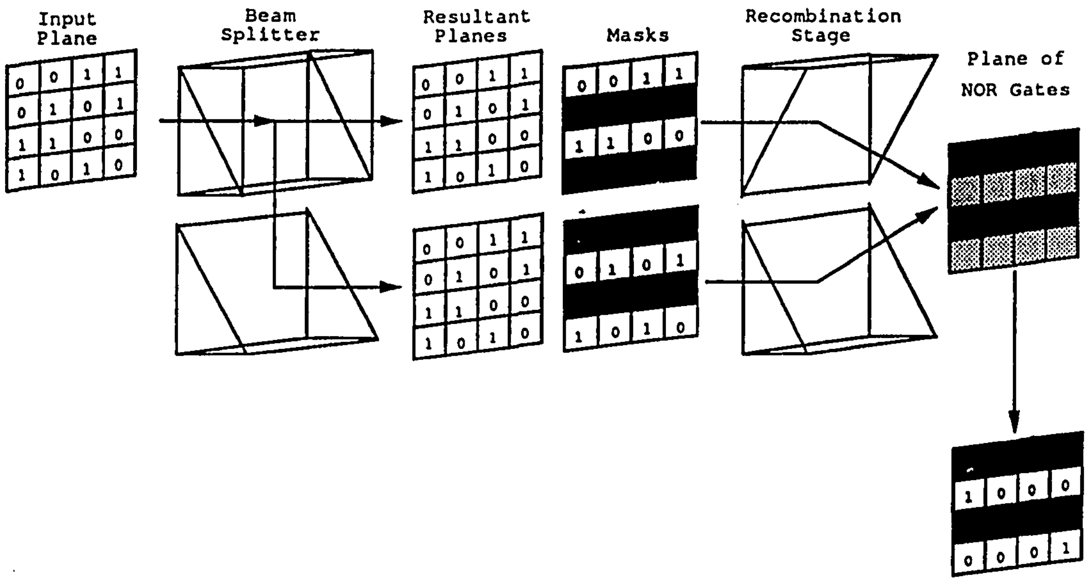


Figure 3.1: Implementation of the two input NOR function

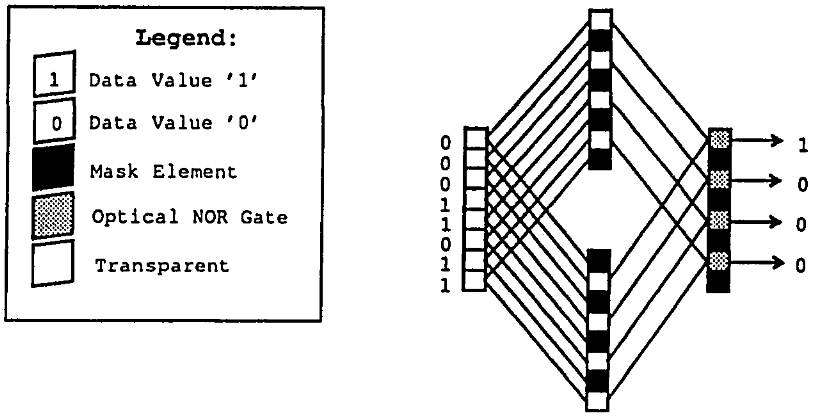


Figure 3.2: Schematic of NOR implementation

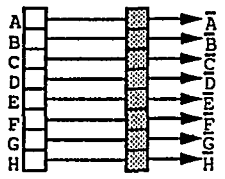


Figure 3.3: Schematic of NOT implementation.

devices, implementation of the NOR function by this method will also require a minimum of four planes of devices (operated in AND and OR modes). Programmable Logic also requires that a field of four pixel positions be reserved for each two bit vector, and that the complements of the inputs be provided in addition to the uncomplemented inputs.

3.3.2 The NOT function.

While not a two-input function, the NOT, or the ability to achieve the function by other gates, is essential for computing any non-monotone function. The implementation of this function by the NOR is direct, and is shown schematicly in figure 3.3 This implementation is trivial, and requires little comment. The main feature of this implementation to note is that if the intensity of the input plane is sufficient to allow splitting into two separate images, then, with proper biasing, the intensity is sufficient to drive the NOR into its low-transmission state. Conversely, a low intensity pixel in the input plane will cause the NOR to remain in a high-transmission state.

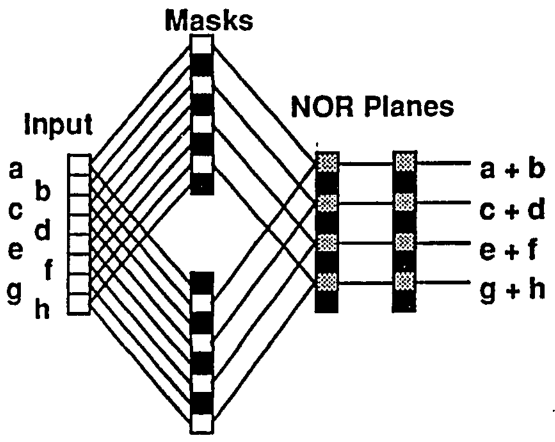


Figure 3.4: Implementation of the OR.

3.3.3 The OR function

The OR function may be obtained from the previous two by simply inverting the output of the NOR. This implementation is shown in figure 3.4. The NOT function must be implemented by planes of NOR functions with the same active device density as that of the plane implementing the NOR.

3.3.4 The AND function.

The construction of the AND in NOR logic is based on the tautology: $AB = \overline{\overline{A} + \overline{B}}$. A schematic of the implementation of this function is shown in figure 3.5. As with the OR and NOR functions, the input is densely mapped to the input plane, requiring no more than one pixel position per bit, while the density of active elements in the NOR planes is one–half that of the input. This construction achieves a savings of one plane of active elements over that which is required by SSL or Murdocca's method.

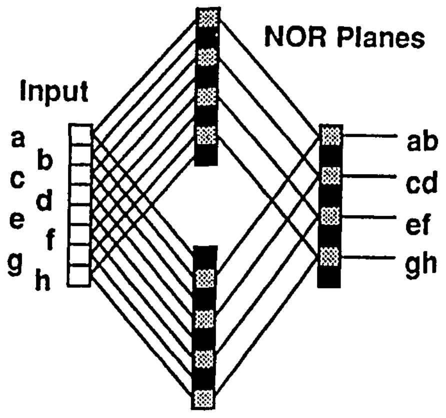


Figure 3.5: Implementation of the AND.

3.3.5 The Equivalence (X–NOR) and the XOR function.

In the previous sections, we have developed implementations for two complete basis sets: the set {NOR}, and the set {AND, OR, NOT}. Either of these sets is sufficient to compute all the boolean functions. Having the ability to compute the functions XOR, and the Equivalence or X–NOR, provides flexibility in designing a variety of circuits. We will first detail the design of the X–NOR.

The logic diagram of the X–NOR is shown in figure 3.8. As can be seen, construction this function from the NOR requires five gates and three levels. It would seem that this construction, when directly translated to an optical realization would require more gate planes than SSL, which we have argued requires four gate planes to implement any two–input function. The freedom allowed by spatially variant interconnections, however, allows us to implement the inverting NOR gates at level one, and the NOR gates in level two as a single fully populated NOR plane for each level. This realization is shown in figure 3.7. A careful examination of this figure will reveal that we have provided the

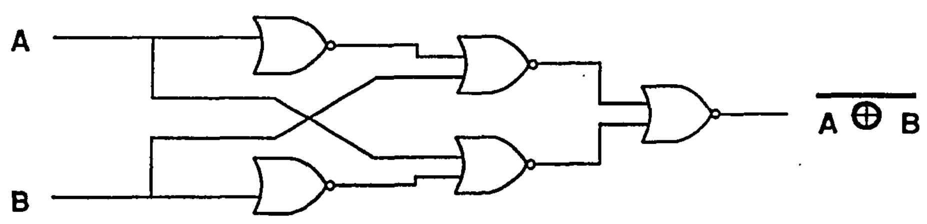


Figure 3.6: Logic diagram of the X-NOR or equivalence function.

five gates which the logical design requires, but have "collapsed" the gates at levels one and two into only two gate planes. Many designs may be able to take advantage of this technique, if the designer is clever, and if mass production of several densities of gate planes is possible.

We noted that the X-NOR and the XOR functions were merely complements of one another, and so the simple inversion of the output of the X-NOR could be used to implement the XOR. We have instead used an alternative implementation which requires only the same number of NOR gates as the X-NOR. The resultant logic diagram shown in figure 3.8. The implementation of this alternate realization of the XOR is shown in figure 3.9. This implementation is also somewhat more expensive than that which would result from simply inverting the X-NOR, since it requires an extra intensifier, and does not collapse the second gate level into a single plane.

We have argued that economy in implementation of optical computers will require that spatially variant interconnections be used to construct boolean functions. We have also stated several characteristics which we feel will be required of optical adjuncts to electronic computers. Further, we have demonstrated how a simple spatially variant imaging technique can be used to construct two basis sets, and two other useful functions.

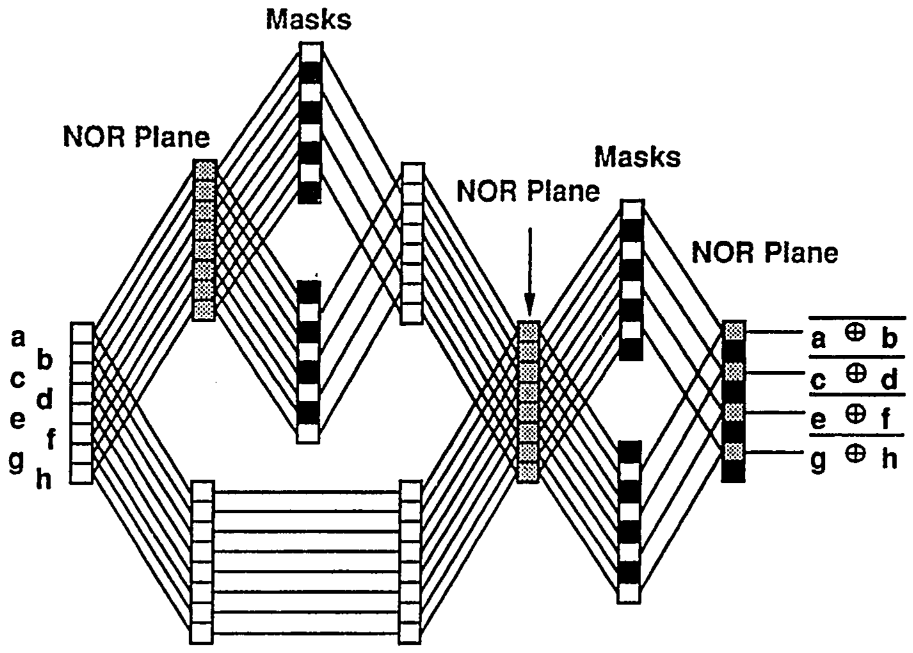


Figure 3.7: Implementation of the X-NOR or equivalence function.

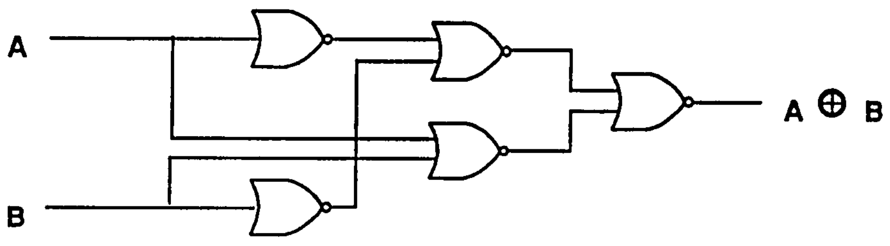


Figure 3.8: Logic diagram of the XOR.

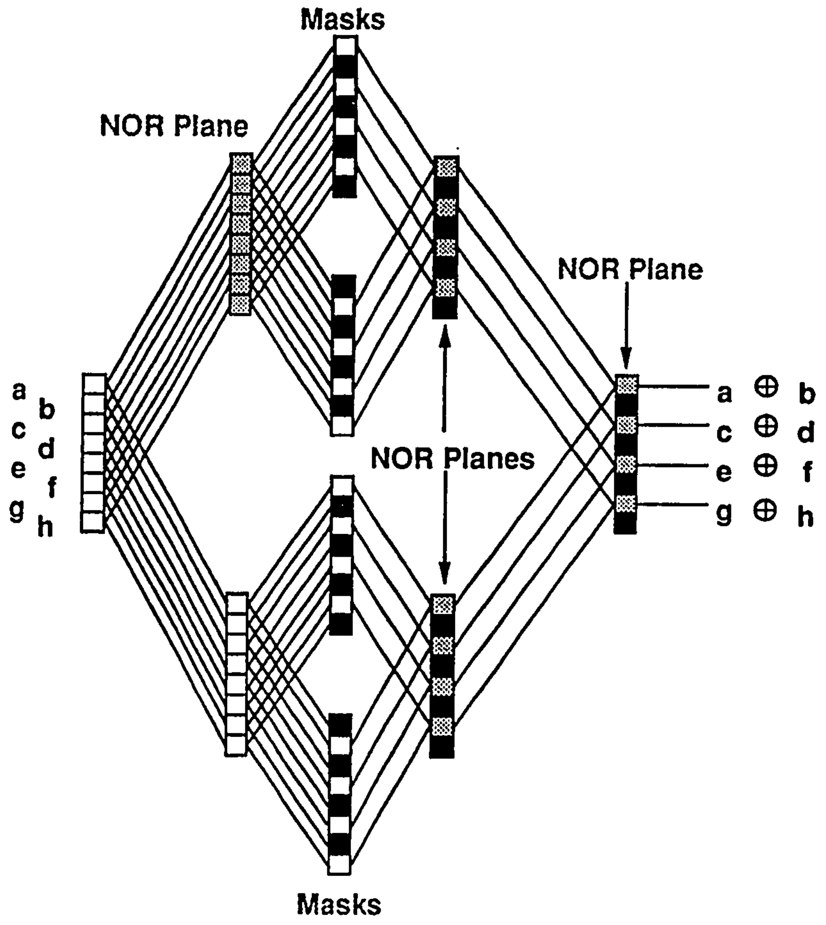


Figure 3.9: Implementation of the XOR.

In the next section, we will expand this discussion to construct larger functions, which are optimal in gate count and depth.

CHAPTER 4

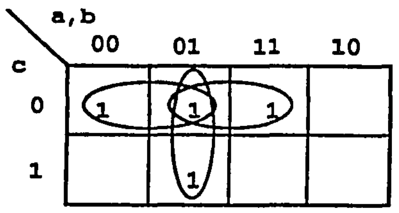
Construction of Larger Functions.

In the previous chapter, we have demonstrated the implementation of the two input boolean functions which comprise the basis of construction of larger functions which are the real goal of the development in this paper. In this chapter, we will illustrate the extension of this technique to functions of larger numbers of inputs. In this extension, we will develop all functions from the basis gate NOR, as was done in the previous chapter. This restriction on gate selection will increase the gate count we require to implement any given function, and thus will tend to impose worst case gate counts on these implementations. This is in keeping with the general desire to demonstrate that even when such conditions are imposed on this method, the method still retains a significant advantage over other methods which have been forwarded. This advantage is a reflection of the underlying complexities involved in the functions we will demonstrate. As these functions have implementations which require less complexity than that imposed by SSL and other paradigms, and as the use of the two input NOR gate as the sole gate function in these implementations can only affect the complexity of the systems we will implement by at most a constant multiple, we will be able to demonstrate the superiority of the proposed system.

4.1 Functions having only one output.

In this section we will be concerned with boolean functions which compute a single output as a function of several inputs. We will not dwell on logic minimization techniques in this development, as such topics may be found in any good introductory text on the subject (see for instance [37]). Rather, we will present Karnaugh maps for the functions we implement, and a minimized circuit implementation using the basis NOR for all gates. We will then present a direct optical implementation of the function whose computation graph is represented by the given circuit. The functions we present have been generated by randomly assigning minterms to their ON-set on the basis of a random event (the toss of a coin).

The circuit implementations have been minimized by finding the prime implicants of the function, factoring the resulting sum-of-products form to reduce the required fanout of the literals, and converting the resultant expression to a NOR implementation. Now, while the NOR may most cheaply be used to implement expressions in product-of-sums form, the converse form was used for two reasons: first, the sum-of-products form is the most frequently used in the optical computing literature and hence we present a familiar underlying notation; second, the slight increase in gate count which may occur from implementing a function expressed in sum-of-products form by NOR gates will further our goal of demonstrating that a worst case implementation of a function by the present method is still superior in gate count to that of other methods.



$$f = \bar{a}\bar{c} + \bar{a}b + b\bar{c} \quad = \bar{a}(b + \bar{c}) + b\bar{c}$$

Table 4.1: Karnaugh map of a random three-input function and minimized formula.

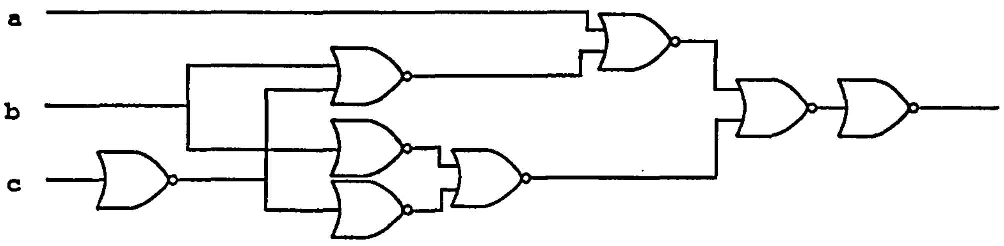


Figure 4.1: Circuit realization of the three-input function using NOR gates as a basis.

4.1.1 An arbitrary 3-input function.

The first function we will implement has three inputs. The Karnaugh map for this function is shown in table 4.1, with it's circuit graph shown in figure 4.1. This function would require two AND, two OR, and three NOT gates to implement in the factored form shown in table 4.1. In the implementation shown in the figure, the function requires only eight NOR gates. Notice that only the uncomplemented inputs are required, and that full use is made of the techniques of logical design.

Figure 4.2 shows the optical implementation of the function for three vectors of three bits in parallel. There are three basic elements: masks, image intensifiers, and NOR gates. The process of construction is as follows: first the input image (the column at the left) is replicated as many times as there are variables in the input vector. In this case, there are only three variables, and so the image is split into three images with one image representing the variables $a, b, and\ c$. These variables are then mapped in pairs to the first level of the computation graph. The outputs of this first level are a set of four images (including the image of the variable a, which does not participate in any computations at this level), which are then focused on the gate planes at the second level of computation. The process continues until the final gate computes the desired function.

Notice, in the figure, that the number of gates in each computation plane is one third the number of input variables, and that the density of these gates is similarly reduced. This reduction is caused by the inherent parallel nature of the light imaged by the spatially invariant devices used in the method. By isolating each variable in separate image planes, we can map any two images to a gate plane with no overlap of variables outside the two we wish to compute. Thus we can achieve both masking and gate functions with one plane. This will be the case in all the functions we demonstrate in this chapter.

4.1.2 An arbitrary 4-input function.

To continue our development, we discuss the design of a random four-input boolean function. As before, we provide the Karnaugh map for the circuit along with a factored boolean formula in table 4.2. The formula for this function has been rendered in product of sums form rather than sum of products. This results in a small decrease in the number of NOR gates required to implement the function, but does not provide a dramatic decrease.

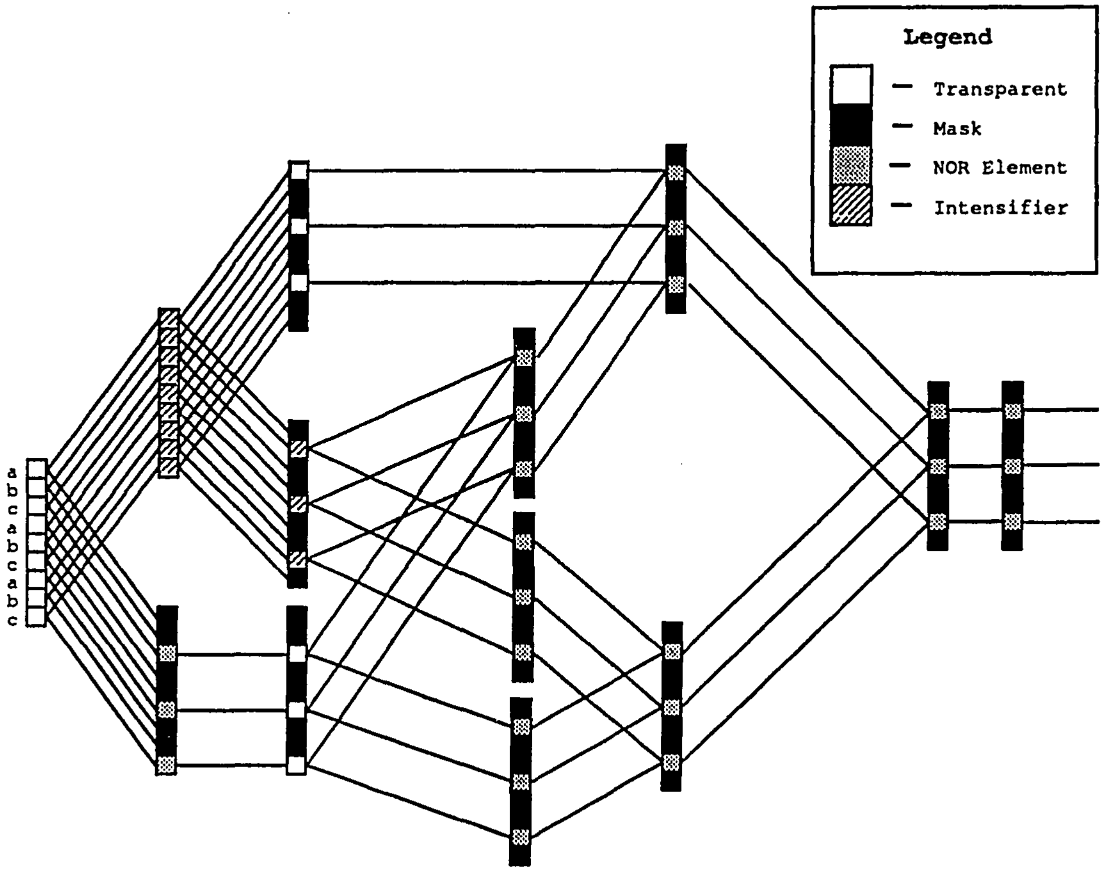
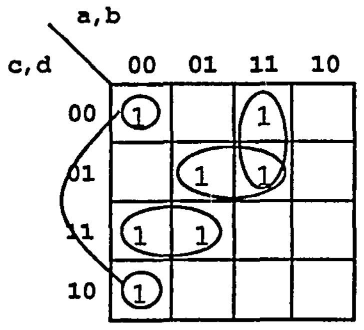


Figure 4.2: Optical implementation of the three-input function.



$$f(a,b,c,d) = a'(b'd' + c\,d) + b\,c'(a + d)$$
$$= (a' + b\,c')(a + b' + d)(b + c + d')$$

Table 4.2: Karnaugh map of the random four–input function, and the resultant minimized formula.

The optical implementation of this function is shown in figure 4.4. In this figure, a pair of four bit vectors are mapped to the input plane, again without providing the complements. In order to provide the fanout from the input variables, four image intensifying elements are employed in order to provide the required intensities at the seven images representing the inputs to the actual computation (a, a', b, b', c, d, d'). Once these seven images have been generated the computation proceeds through the spatially variant interconnected circuit.

Charging all active elements to the cost of this circuit, we see that sixteen elements are required in the NOR implementation. If AND and OR functions were used instead, inversions not counted, and given an arbitrary degree of fanout, the circuit could have been constructed from only eight active elements. But restrictions imposed on the gates, and the method of counting have doubled the gate count required of this implementation.

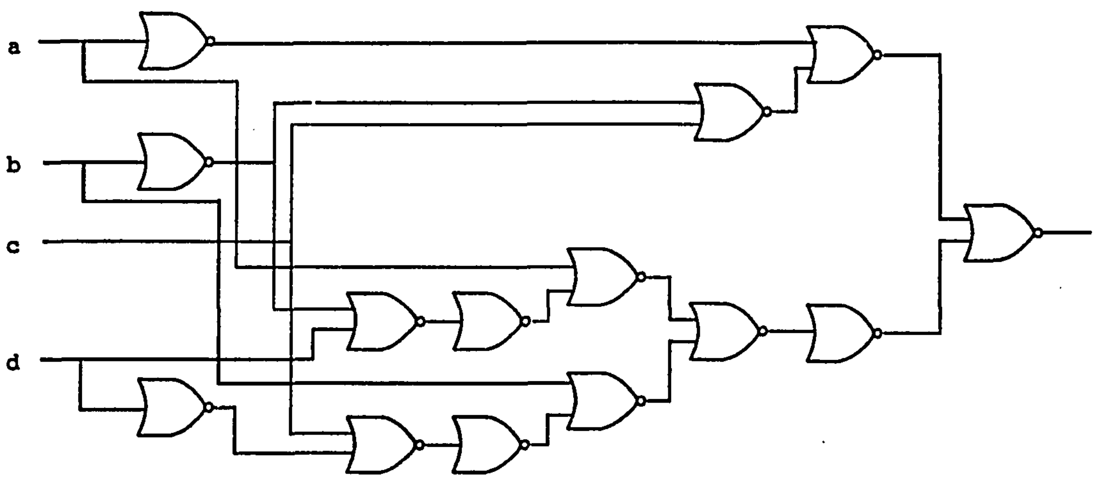


Figure 4.3: Circuit realization of the four–input function in terms of NOR gates.

4.1.3 An arbitrary 5-input function.

The last arbitrary function we describe has five inputs and one output. The Karnaugh map of this function is shown in table 4.3. This function has fifteen minterms which can be combined to form six prime implicants. Factoring the resulting sum of products formula to achieve a multi-level realization yields the formula shown at the bottom of the table.

The resulting implementation (excluding the network providing fanout from the inputs) is shown in figure 4.5. This NOR implementation requires twenty gates and seven levels. The optical implementation of this function for one input vector is shown in figure 4.6. Again, this figure is highly spatially variant, but achieves a one to one mapping from the graphical realization. The only increase in complexity in this implementation is that required by providing the required fanout from the input variables.

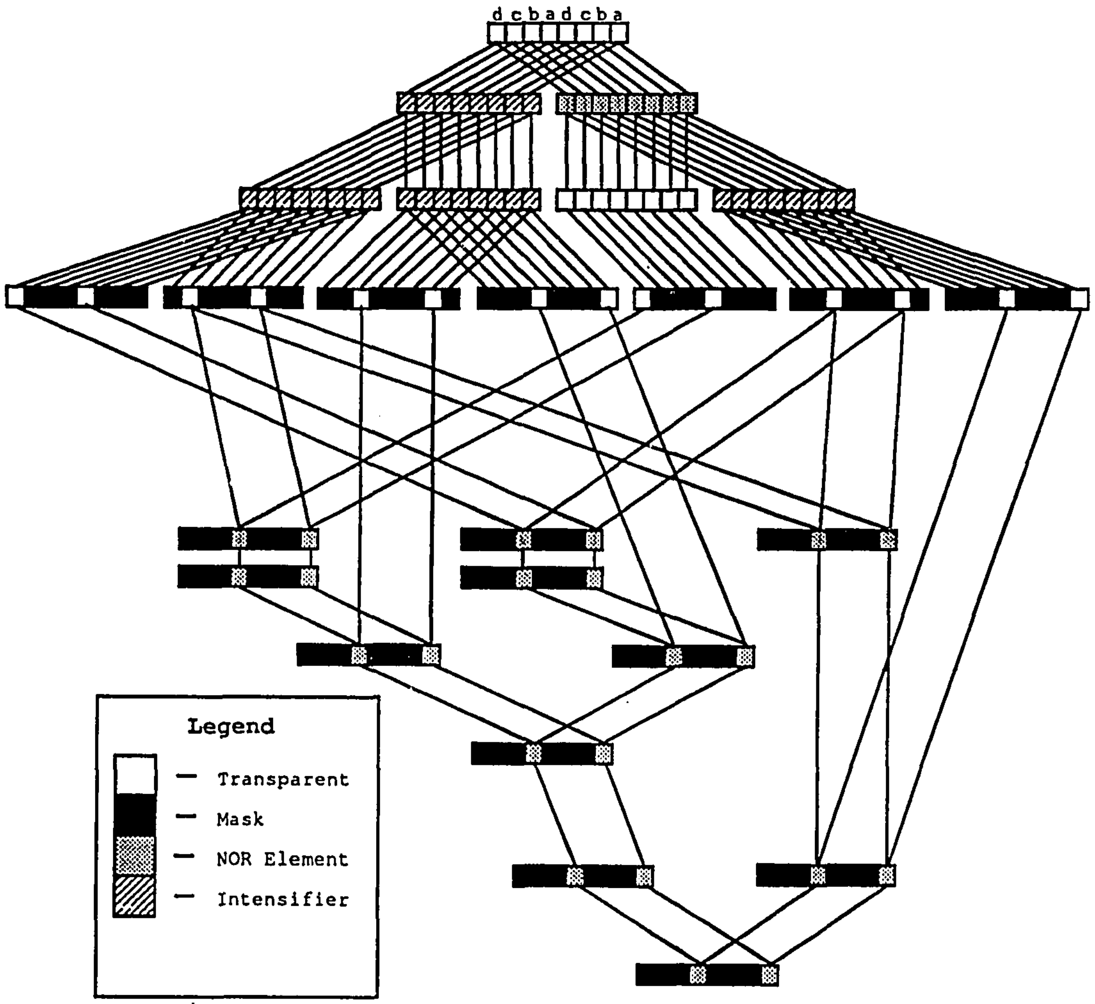


Figure 4.4: Optical implementation of the four–input function.

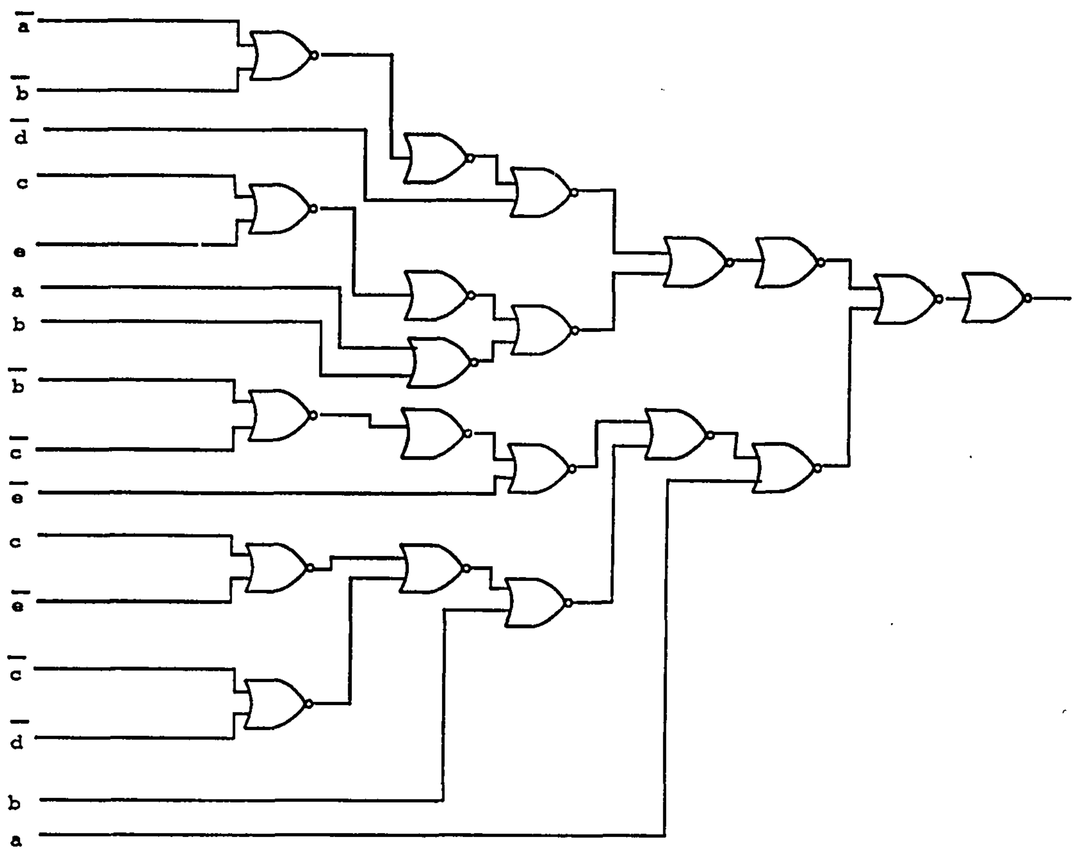


Figure 4.5: Circuit realization of the five-input function in terms of NOR gates.

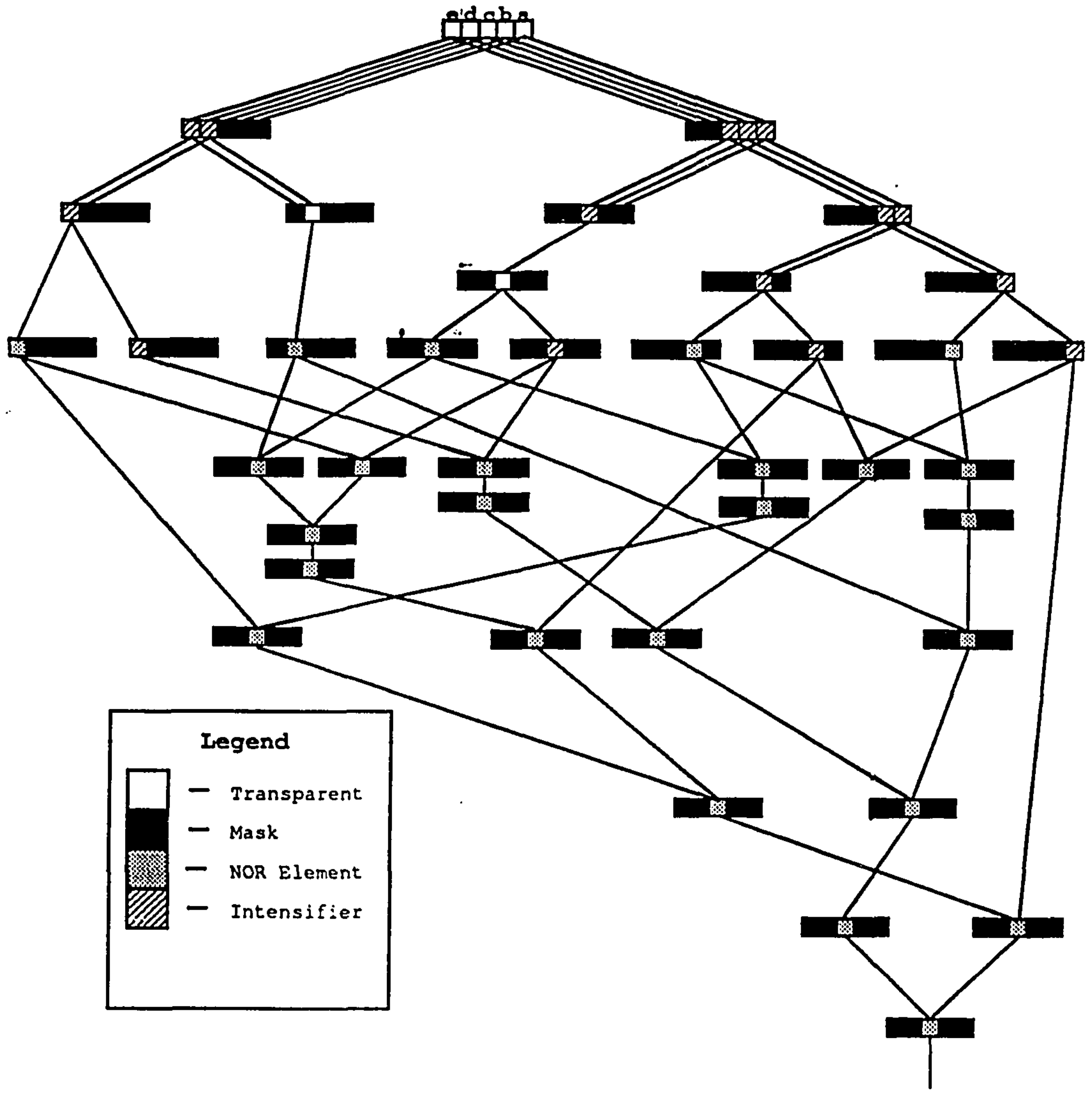
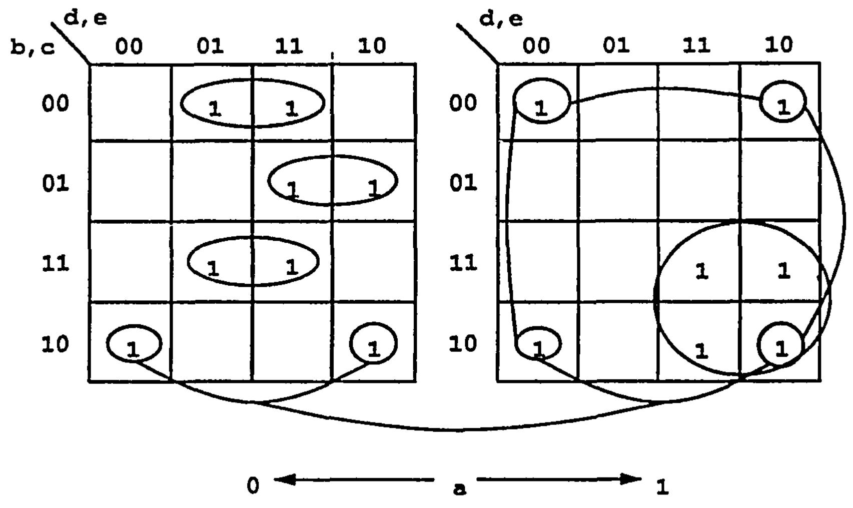


Figure 4.6: Optical implementation of the five-input function.



$$f(a,b,c,d,e) = \bar{a}(\bar{b}(\bar{c}e + cd) + bce) + \bar{c}\bar{e}(b + a) + abd$$

Table 4.3: Karnaugh map and resultant minimized formula for the five–input function

4.2 Implementations of Some Selected Functions

We now turn our attention to some practical applications, the full adder, and the carry look–ahead adder. While numerous optical papers have demonstrated designs of the full adder, we will go beyond this simple circuit, and demonstrate the design of the carry look–ahead adder, which is a parallel adder. Despite the apparent parallelism hypothesized for optical systems, most adders proposed to date are based on the full adder with ripple carry between full adder stages. This kind of addition is fundamentally serial, and does not truly exploit the high degree of parallelism which should accompany the use of optical computing elements. After first demonstrating the implementation of the full adder, we will demonstrate how the carry look–ahead adder might be implemented. Specificly, we will demonstrate the design of a carry look–ahead adder which takes as its inputs two 2–bit binary numbers, and a carry input, and produces two sum bits and a carry–out. Larger adders can be constructed from this circuit by rippling the carry between these stages, by increasing the size of the adder to accept a greater number of bits in each input vector, or by adding another level of carry look–ahead. While a four or eight bit stage would be more practical in a real application, the size of these adders would merely complicate the drawings without shedding any more light on the design technique we hope to demonstrate.

4.2.1 The full adder.

The normal full adder design is shown in a simplified form in figure 4.7, and its realization in NOR logic is shown in figure 4.8. Our optical design is shown in figure 4.9. While there are several ways to implement the full adder the one we have chosen has three logic levels in XOR–AND logic. While there is a two–level design, the total gate count of this adder is slightly greater, and the fanout of some of the nodes is greater also. From the preceding discussions, it should be clear, that we need not have chosen an adder circuit which has a maximum fanout from any one node of two. The choice of this adder circuit was driven by the minimality of the number of gates rather than fanout restrictions. We will not dwell on the design of this adder other than to note that the size of this adder, as measured by the number of image planes is quite comparable to that of Programmable Logic, and to that of any SSL implementation. While the number of detection planes in

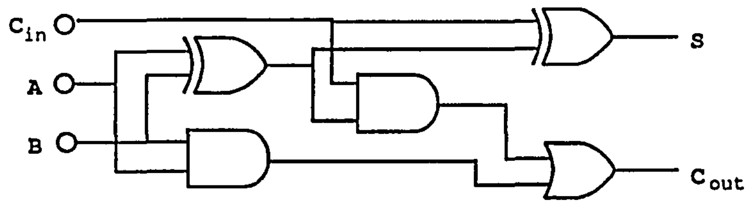


Figure 4.7: A minimized circuit graph for the full adder.

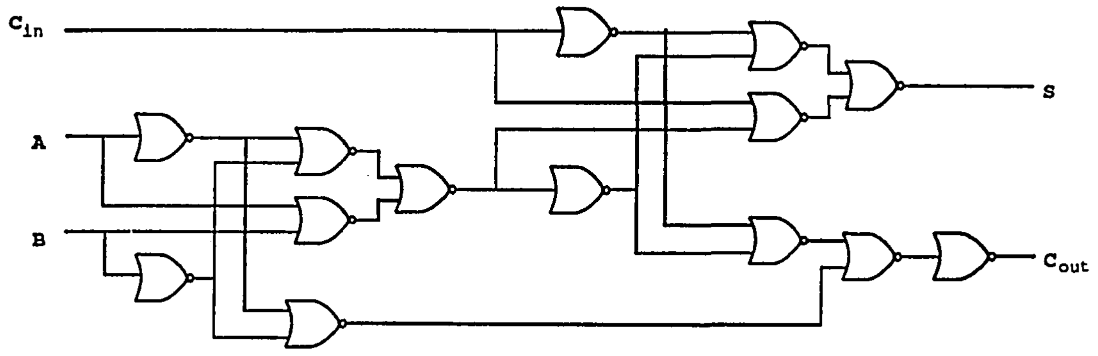


Figure 4.8: A minimized NOR gate implementation of the full adder.

Shadow Casting logic is limited to one, the space bandwidth product of the effective data plane in this implementation, is again eight times that of any one of the input planes.

4.2.2 The carry look–ahead adder.

As with any general adder, the carry look–ahead adder takes as its inputs two n-bit vectors and a carry as its inputs, and generates an $n + 1$-bit result. The two–bit carry look–ahead adder is shown in figure 4.10. In this figure the carry look–ahead adder is implemented under the basis {AND,OR,XOR}. The circuit operates by computing in

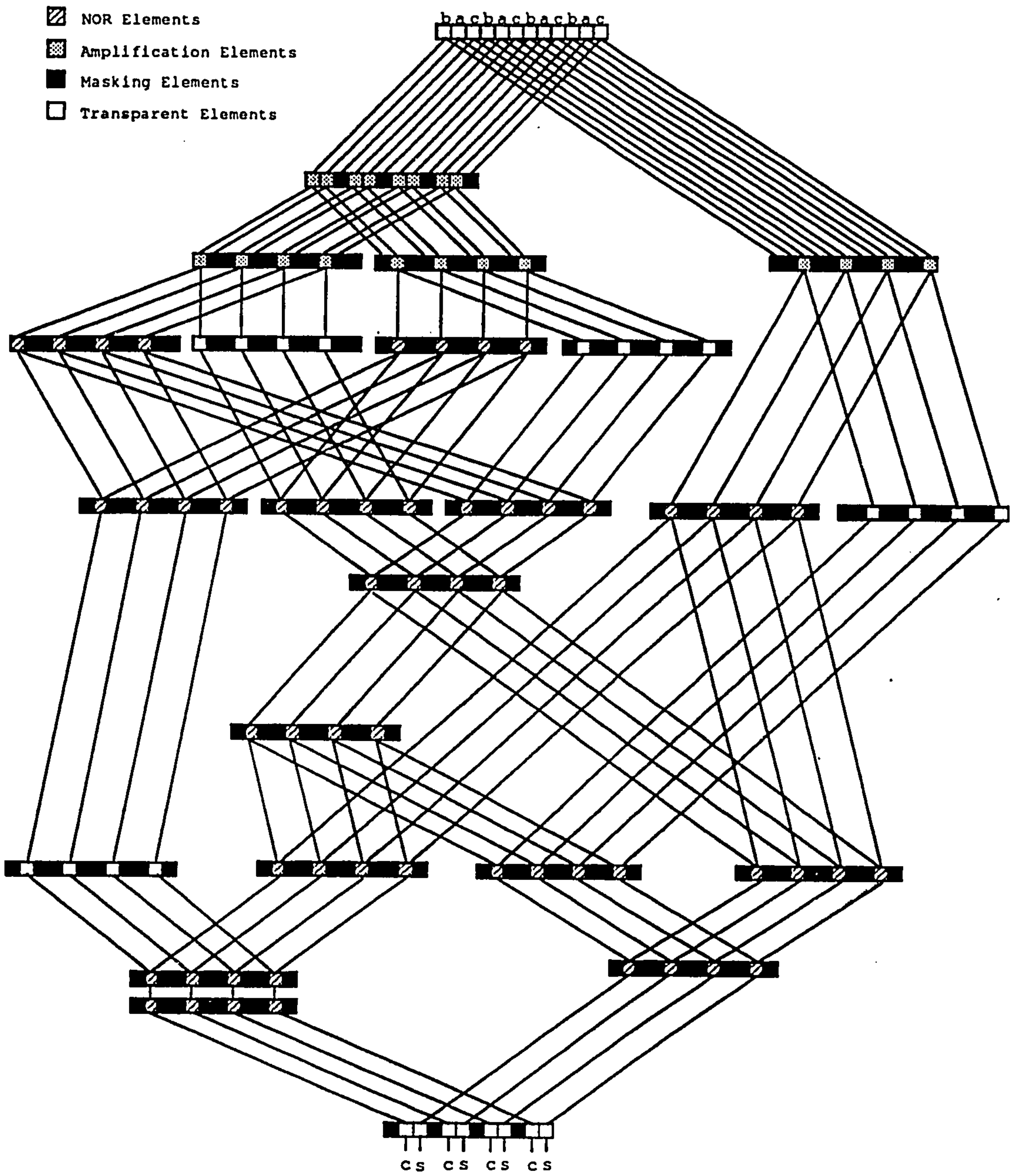


Figure 4.9: Optical implementation of the full adder.

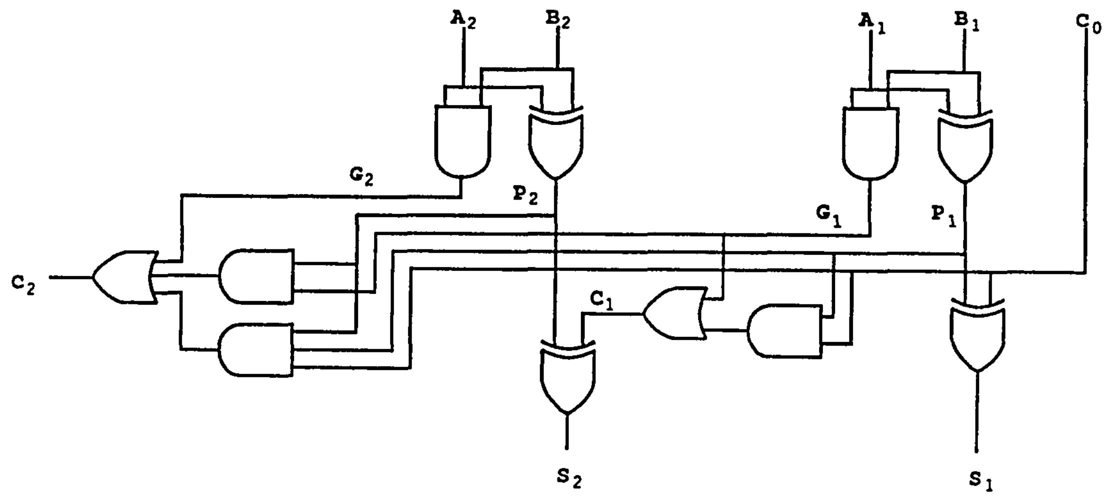


Figure 4.10: A minimized circuit graph for the carry look-ahead adder.

parallel a pair of intermediate signals for each pair of inputs. These signals are shown in figure 4.10 as G_i and P_i for $i = 1, 2$. The signal $G_i = A_i B_i$ is true if a carry is generated by inputs A_i and B_i. The signal $P_i = A_i \oplus B_i$ is true if a carry is propagated by the pair. A carry look-ahead adder having $2n + 1$ inputs will then generate $2n$ of these intermediate signals. From this information, the carry out from the ith pair can be computed as:

$$C_i = G_i + (P_i G_{i-1}) + \cdots + (P_i P_{i-1} \cdots P_2 G_1) + (P_i P_{i-1} \cdots P_1 C_0), \qquad (4.1)$$

while the corresponding sum is computed as:

$$S_i = P_i \oplus C_{i-1}. \qquad (4.2)$$

The cost in number of gates and the delay to realize the adder for pairs of n inputs under the basis {AND, OR. XOR} can be computed by a counting argument. The terms

G_i and P_i can be realized in unit size and depth. The computation of the sum S_i can also be realized in unit size and depth given P_i and C_{i-1}. Hence for each output we have at least three gates required. The carry C_i will require one $i+1$–input OR gate, and i AND gates having $(2,3,\ldots,i,i+1)$ inputs. The $i+1$–input OR gate may be implemented by a binary tree whose nodes are two–input OR gates. For $i=0$, there is no cost since the carry C_0 is an input. If $i=1$, then only one two–input OR will be required. For $i>1$, i two–input OR gates will be required, and the depth of the OR–tree will be $\lceil \log_2 i \rceil$. Thus the $i+1$–input OR may be replaced by a circuit consisting of i–OR gates, with depth $\lceil log_2 i \rceil$ exactly.

The analysis of the number of AND gates is somewhat more involved. We can easily construct a j–input AND gate from a binary tree of $j-1$ two–input AND gates ($j \geq 2$). Thus, the total number of two–input AND gates required to implement the carry for the ith bit is:

$$C_{AND's} = \sum_{j=2}^{i+1}(j-1) = \sum_{j=1}^{i} j = \frac{i(i+1)}{2}, \tag{4.3}$$

while the depth of the resulting tree is the depth of the largest sub-tree which is:

$$D_{AND's} = \lceil \log_2(i+1) \rceil. \tag{4.4}$$

Thus, the total cost of a carry look–ahead adder having two n–bit inputs is the sum over the costs of all outputs:

$$\begin{aligned} C_{CLA} &= \sum_{i=1}^{n}\left(3+i+\frac{i(i+1)}{2}\right) \\ &= \sum_{i=1}^{n}(3+\frac{3}{2}i+\frac{1}{2}i^2) \\ &= 3n+\frac{3}{2}\frac{n(n+1)}{2}+\frac{1}{2}(\frac{n}{6}+\frac{n^2}{2}+\frac{n^3}{3}) \end{aligned}$$

$$= \frac{1}{6}n^3 + n^2 + \frac{23}{6}n. \tag{4.5}$$

The depth is found as the maximum depth, which is associated with the final carry:

$$D_{CLA} = \lceil \log_2 n \rceil + \lceil \log_2(n+1) \rceil + 2. \tag{4.6}$$

The implementation of the two-bit carry look-ahead adder is shown in figure 4.11. An upper bound to the number of NOR gates required to implement this circuit can be calculated by multiplying the number of XOR, AND, and OR gates by the number of NOR gates required to implement each of these functions, and summing the results. This is an upper bound because the possibility exists that minimizations such as canceling two successive inversions can reduce the number of gates actually required. Even though the direct conversion of the carry look-ahead adder to a NOR-based implementation increases the gate count, this increase is bounded by at most a constant multiple, hence the implementation will still require only $O(n^3)$ gates, and $O(log_2 n)$ depth. The direct optical implementation of this circuit is shown in figure 4.12.

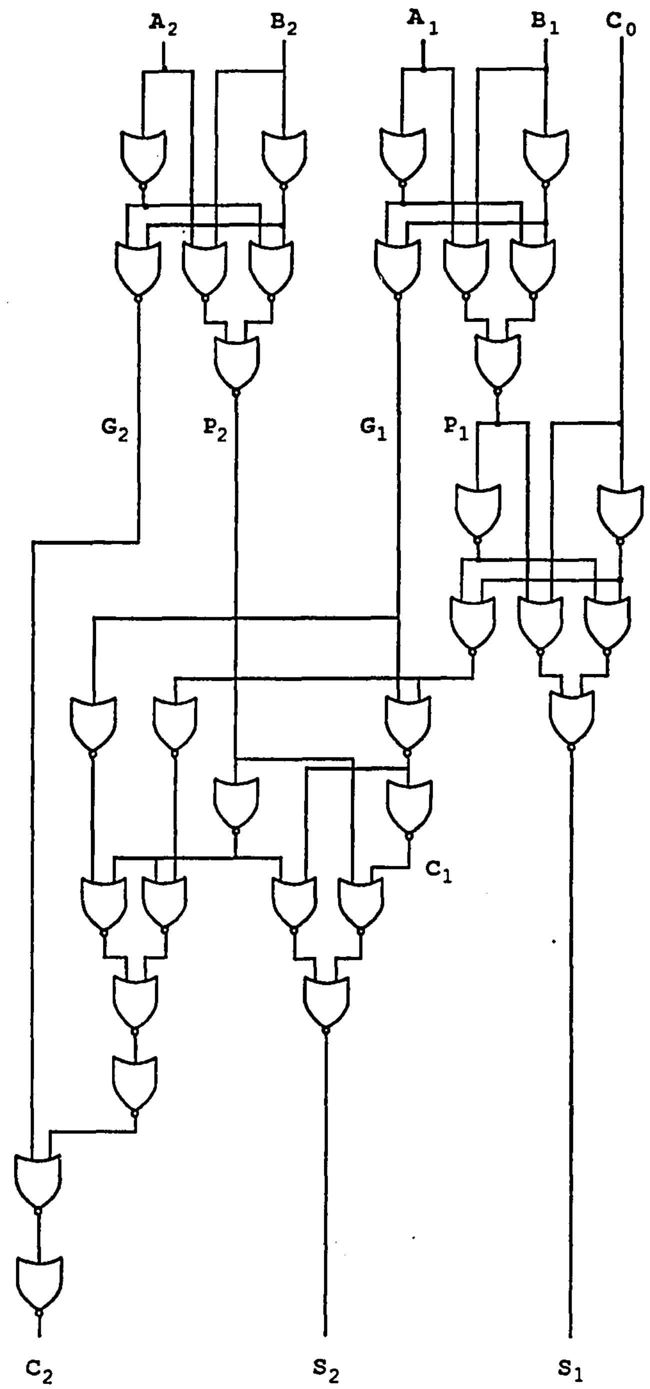


Figure 4.11: A minimized NOR gate implementation of the carry look-ahead adder.

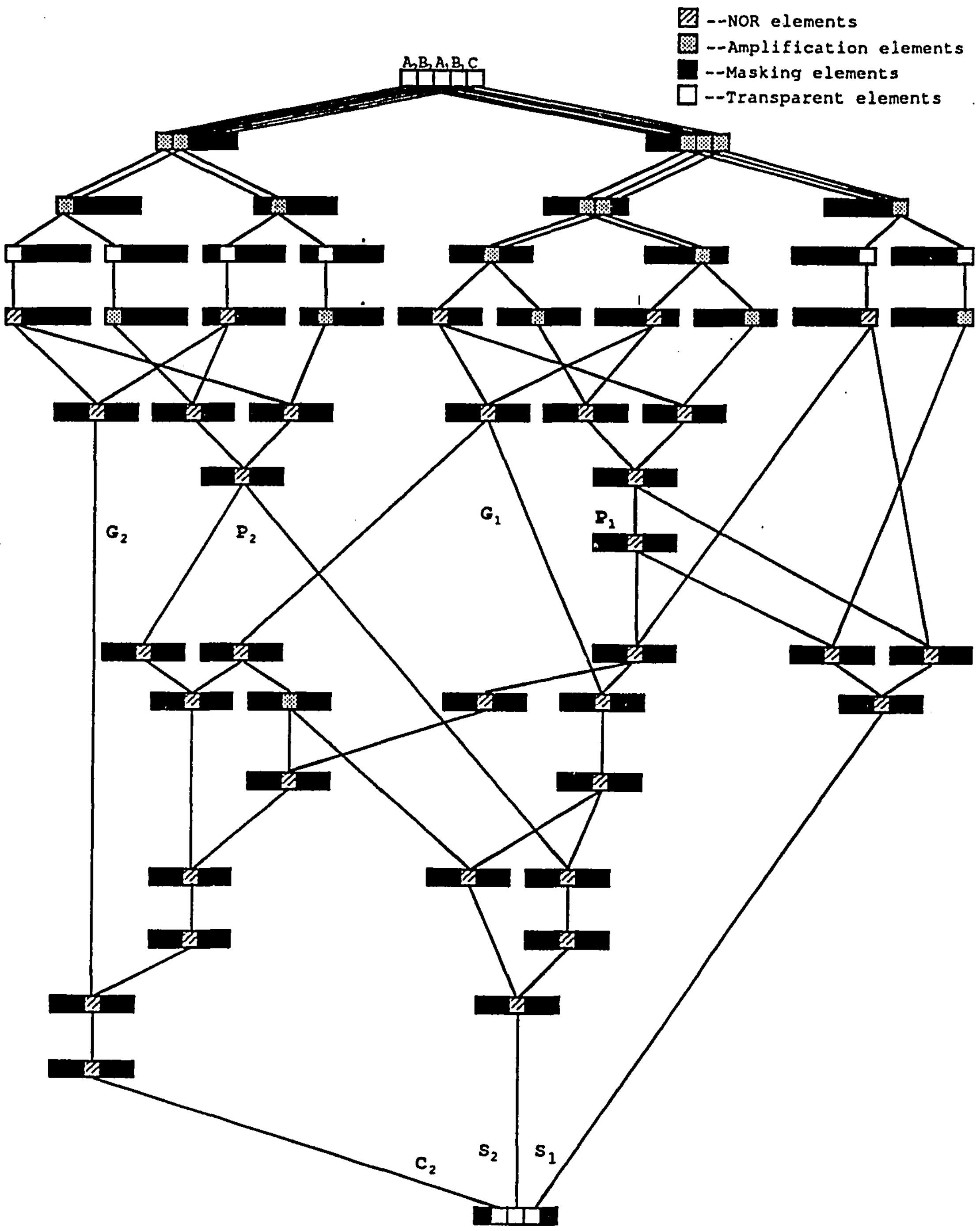


Figure 4.12: Optical implementation of the carry look-ahead adder.

CHAPTER 5

Comparisons with Other Methods

Having developed our method and demonstrated several functional implementations, we can now demonstrate the viability of this method in relation to other digital optical computing methods which have been advanced. In terms of complexity, this method has advantages over all the other methods analyzed, since these other methods are so similar in complexity. A tabular comparison of the orders of complexity of the proposed system, and the other systems addressed by this paper may be found in table 5.1, for three separate measures of complexity. The first two columns summarize the required complexity for the various paradigms under consideration in terms of the normal methods of boolean complexity (ie. gate count, and circuit depth). The third column summarizes the energy required for the computation in terms of the number of inputs to the function. This last measure provides a more uniform view of the alternatives, since the energy required to switch a given detector or gate is a fundamental limiting parameter of any logic implementation, and since the energy required at the input to a SSL implementation increases exponentially, and is not implicit in the count of detectors.

While the table 5.1 only reports results for one particular function, the results can be extended to arbitrary functions. Since spatially variant interconnections will allow

Method	Number of Gates, Detection Planes, or Sources	Circuit Depth or Time Complexity	Normalized Energy 1 Gate/Detector = 1
SSL	$\Omega(2^n)$	$\Omega(\log_2 n)$	$\Omega(n2^n)$
Shadow Casting	$O(2^n)$	$O(1)$	$\Omega(2^n)$
Programmable Logic	$\Omega(n2^n)$	$\Omega(n)$	$\Omega(n2^n)$
Proposed Spatially Variant Method (Carry Look-Ahead Adder)	$O(n^3)$	$O(\log_2 n)$	$O(n^3)$

Table 5.1: Order analysis of an n-bit parallel adder using the methods in this paper. Normalized Energy refers to the amount of energy required to compute the function in terms of the energy required to switch one gate or detector.

implementation of boolean functions at a cost in gate count and depth consistent with the best theoretical limits, any polynomially complex function could alternately be considered, and similar results obtained. A more detailed comparison of the advantages of spatially variant implementation of boolean functions will reveal further advantages. Such a comparison is provided in the following sections.

5.1 The present method vis-a-vis Symbolic Substitution.

In analyzing SSL, we saw that the power required at the input plane increased as $\Omega(n * 2^n)$ times the power required to switch the detector, and that there was no way to reduce this requirement without leaving the boolean domain. While SSL is a viable method for certain algorithms which can be transformed to serial implementations operating on many sets of a small number of bits [38], or for those algorithms which can make use of large numbers of "don't care" combinations, as a general-purpose computing paradigm, it suffers from the above mentioned limitations. In contrast, the method proposed in this paper provides several advantages.

Perhaps the greatest advantage with this method, is the ability to achieve near minimal complexity in gate count (and hence power dissipation), while retaining the high degree of parallelism promised by optics. Once the input has been duplicated to provide a number of separate images equal to the number of input variables (a process which has a O(n) gate or amplification count, and a depth $O(\lceil\log_2 n\rceil)$), the subsequent computations are performed in a minimal number of operations, providing the circuit has been properly minimized. Thus the design cycle is similar to that of the electronic domain, in that the initial effort is devoted to computations and methods which have only abstract connection to the physical realization, but which generate a "good" realization when translated to the physical realm. Hence we may use well established methods of circuit minimization in our initial design, with the certainty that any minimal design so conceived will be increased in complexity only by an additive linear term, and the depth increased by only an additive logarithmic term. More concretely, given a function whose implementation requires $O(n^3)$ gate count, and $O(\log_2 n)$ depth, we will be able to implement the function opticly with costs:

$$C(f_n) = O(n^3) + O(n); \quad D(f_n) = O(log_2 n) + O(log_2 n). \tag{5.1}$$

Thus for functions of a large number of inputs, the cost involved in duplicating the input plane is a small fraction of the cost of implementation, which is itself much smaller than that which could be achieved in SSL.

We achieve a further gain over most SSL implementations by using simple intensity coding. The factor of two increase in data density achieved by this encoding may be increased over that of SSL in those cases in which SSL must provide "dead space" between fields of operands. Our density of active elements is also twice that of SSL. In additive SSL, there is only one active element for every n input bits, corresponding to 2n pixel positions. The method we propose here will result in one active element for every n pixels.

5.2 The present method vis-a-vis Shadow Casting Logic.

Shadow Casting Logic is the most difficult of the proposed paradigms with which to compare the proposed system. This difficulty results from the ways in which the paradigm's complexity is measured. The decrease of available light with increasing numbers of inputs does show an exponential variance and the numbers of sources required is $O(2^n)$. There does not seem to be a good choice for a single unit of measure which can combine these two variances. In the best case for Shadow Casting, the implementation of a function having only one minterm in its On-set, the available light at the detector is only 2^{-n} of that which is incident on the corresponding cell. In this case, however, the method we propose can implement such a function by a circuit having $O(n)$ gates.

That this is true may be seen from the following argument. A single minterm of n variables may be implemented by a n-input AND gate whose inputs are suitably assigned the value of each variable or that variable's complement, depending upon the form of the minterm. Hence at most n inverters are required in addition to the n-input AND, if the complements of the inputs are not available. Now a n-input AND gate may be constructed from $n-1$ 2-input AND gates; the resulting tree having a depth of $\lceil \log_2 n \rceil$. Thus a function having n-inputs, and having only one minterm in its On-set may be implemented by a circuit containing at most n inverters, and $n-1$ AND gates, for a total of $2n - 1$ active elements. Even when restricted to NOR gates alone, each AND may be replaced by at most three NOR gates, and each inverter with one, resulting in a circuit having at most $4n - 3$ gates. Since the power required of a circuit is proportional to the number of gates, such a function can be constructed having $O(n)$ power requirements.

At the other extreme, to compute the parity function a Shadow Casting implementation will require 2^{n-1} sources or detectors. An implementation of this function in terms of the XOR will require only $O(n)$ XOR gates, each of which can again be replaced by a constant number of NOR gates. In between these extremes, there will be functions which will require a direct optical implementation of the best theoretical solution have $\Omega(2^n/n)$ gate functions, and hence power dissipation. This will still be an improvement over that achievable with Shadow Casting.

5.3 The present method vis-a-vis Programmable Logic.

With respect to Programmable Logic, this method provides many of the same advantages, while eliminating several of the former method's weaknesses. As with Programmable Logic, this method uses optical non-linearities directly as logic gates. The use of general interconnections, however, overcomes the exponential complexity imposed by the limited interconnection capabilities of the cross-over network.

Comparing the carry look–ahead adder with a parallel adder implemented with only the crossover interconnection, we can see that for the two–bit adder demonstrated, 34 NOR gates were required in the spatially variant implementation, while the latter method would require a minimum of $2 * 5 * 2^5 = 320$ AND and OR gates. Even given that the latter method can provide fault tolerance [35], a triple–modular–redundant version of the spatially variant circuit with voting would still require fewer gates, and hence be fundamentally more reliable.

Further, even when a function implementation can be minimized in Programmable Logic to a level requiring only $n * 2^n$ logic elements, the number of outputs which can then be computed is indeterminate, and is limited by the ability to find contention-free paths through the OR-stage of the network.

5.4 The present method vis-a-vis Combinatorial Logic.

Comparison with the method of Combinatorial Logic is difficult. While the spatially invariant version of this method has been shown to be capable of implementing only a small subset of the possible functions, allowing spatially variant interconnections may extend the capabilities of that system to allow a greater portion, perhaps all, of the possible boolean functions to be computed. We note however, that the system which we propose can implement all boolean functions, at a cost in power and speed commensurate with the best theoretical implementations. As our system is spatially variant overall, and uses optical elements directly as gates, there is no need to pre-compute partial terms electronicly, and hence the speed should be close to the maximum attainable opticly.

CHAPTER 6

Conclusions

We have seen that several seemingly dissimilar optical computing paradigms have complexity measures which are quite similar:

- SSL requires 2^n detector planes and $2n$ shift operations per detection operation.
- Shadow Casting logic requires n planes of spatial light modulators, $O(2^{n-1})$ light sources, with a resultant decrease in illumination on the detector plane a function of 2^{-n}.
- Programmable Logic requires a field in each plane of at least 2^n gates width, and a number of planes at least $2n$ be reserved.
- The spatially invariant subsystem of the Combinatorial Logic method can not compute all the possible boolean functions.

It has been our contention that all these systems share similar complexity measures because of the nature of the restrictions imposed by their imaging techniques.

We have further shown that simple spatially invariant interconnects can reduce the complexity required of an optical implementation. This reduction is achieved because spatially invariant implementations impose exponential lower bounds on any completely specified boolean function, whereas many functions have upper bounds which have only polynomial complexity in the best theoretical implementation, and allowing spatially variant patterns of interconnection yields a direct mapping of any arbitrary boolean circuit to the optical domain. Further, most functions peculiar to the arithmetic–logic unit of a computer have such polynomial upper bounds. Such reductions in complexity, from exponential to polynomial, become essential when attempting to construct systems having practical application, particularly so in high–end systems which typically have word lengths of 32, 64, or even 128 bits. The spatially invariant optical computing paradigms operating on such large operands can only implement functions which can be serialized, and the full potential parallelism of optical computing is not realized.

Obviously, what we have presented here is in the nature of a plausibility argument, rather than a strict proof. A rigorous model of spatially invariant imaging elements and spatially invariant interconnections applicable to the level considered here is needed. Such a model could be used as the basis of constructive proofs which could demonstrate rigorously whether there is any set of spatially invariant interconnections which can optimally construct boolean functions of higher order. The model of Thompson [39] has had a great impact on VLSI design; a similar model for spatially invariant optical systems, or possibly a model incorporating spatially variant techniques, should make a similar impact on the field of optical computing.

Modelling spatially invariant imaging elements will likely rely on the linearity of the transforms defined by the object plane to image plane mapping of such imaging systems. Proof of lower bounds to the gate count achievable by such means will probably involve either an argument which counts the number of interconnections achievable in this manner and compares this to the number of interconnections possible as a function of the number of inputs, or may resemble the counting argument advanced by McColl [12]. In addition, a model of spatially variant imaging techniques incorporating such physical aspects as focal length and aperture, will likely provide tight bounds on features such as delay, area, and volume for specific functions, allowing a designer to evaluate such a system prior to construction.

Appendix A

Notations and Conventions from Boolean Function Theory

This paper will use a variety of notations, conventions and results from the field of Boolean function theory. These notations are not normally used in introductory texts in the field of switching theory where the emphasis is on achieving a minimization of the cost of some particular boolean function, and not on studying the characteristics of boolean functions in general. The later field is principly involved with finding bounds on the complexity of given functions, in order to determine whether a given implementation of some function is as economical or as fast as possible. To this end, the normal measure of cost is the number of gates required to implement a given function, denoted $C(f)$. The normal measure of the speed of an implementation is the depth of the function, denoted $D(f)$.

By boolean function, we mean a function whose inputs are binary variables, and whose outputs are also binary variables. We will use the following forms as shorthand:

- We use the notation $\mathbf{B}$ to denote the set $\{0,1\}$.
- We use the notation $\mathbf{B}^2$ to denote the cartesian product: $\{0,1\} \times \{0,1\}$.
- The extension of the above to n dimensions is denoted: $\mathbf{B}^n$.

- We will denote by f_n a function mapping $\mathbf{B}^n \rightarrow \mathbf{B}$. In other words f_n denotes a function having n binary variables as its inputs, and computing a single output.
- A function mapping $\mathbf{B}^n \rightarrow \mathbf{B}^m$, will be denoted: $f_{n,m}$. This will be the notation used for a boolean function having n inputs and computing m results.
- The set of functions having n inputs will be denoted: $\{f_n\}$.
- The set of functions having n inputs and m outputs will be denoted: $\{f_{n,m}\}$.

We will also make use of the following notations which are standard in several branches of mathematics and computer science:

- A function is bounded above by (grows no faster than) another function, denoted f(n) = $O(g(n))$ if:

$$\lim_{n\rightarrow\infty} \frac{f(n)}{g(n)} \leq c; \quad c: \textit{ a constant.}$$

- A function is bounded below by (grows no slower than) another function, denoted f(n) = $\Omega(g(n))$ if:

$$\lim_{n\rightarrow\infty} \frac{g(n)}{f(n)} \leq c; \quad c: \textit{ a constant.}$$

- A function is bounded above and below by another function, denoted f(n) = $\Theta(g(n))$ if:

$$f(n) = O(g(n)), \textit{ and } f(n) = \Omega(g(n)).$$

Now the number of boolean functions of n variables computing one output may be found to be:

$$|\{f_n\}| = 2^{2^n}, \tag{A.1}$$

and the number of functions computing m outputs can be shown to be:

$$|\{f_{n,m}\}| = 2^{m2^n}. \tag{A.2}$$

One of the early results in the field of boolean function theory, due to Shannon [40], is that for "almost all" boolean functions, the complexity (cost in gates) to implement is:

$$C(f_n) = \Omega(\frac{2^n}{n}), \tag{A.3}$$

where:

- again, n is the number of inputs to the function, and
- "almost all" has the precise definition that the ratio of the number of functions having this property to the total number of functions is asymptotic to 1, as the number of inputs becomes arbitrarily large. Or written more concisely:

$$\text{almost all } f_n \text{ have Property P} \Leftrightarrow \lim_{n\to\infty} \frac{|\{f_n : f\ has\ P\}|}{2^{2^n}} = 1. \tag{A.4}$$

Now it is true that many functions do not grow in complexity as quickly as Shannon's theorem would indicate (a fact that Shannon, himself, recognized). For many functions and classes of functions polynomial upper bounds have been established [36]. Moreover, quadratic upper bounds have been established for some functions.

Appendix B

Derivation of Theorems

In this appendix we will establish a result which allows us to consider only those functions which depend on all of their variables, without considering those functions which do not. A function f is said to be essentially dependant on a given variable x_i if:

$$f(c_0, c_1, \ldots, c_{i-1}, 1, c_{i+1}, \ldots, c_{n-1}) \neq f(c_0, c_1, \ldots, c_{i-1}, 0, c_{i+1}, \ldots, c_{n-1}),$$

where the c_k are any constant values of their respective variables. A function which is essentially dependant on all its variables is termed non-degenerate. Now almost all functions of n variables are non-degenerate, in the sense of "almost all" defined above. To see this is so, we note that:

$$\{f_{n-1}\} \subset \{f_n\}, \; n \geq 1.$$

Now given those functions of $n-1$ variables out of the functions of n variables, there are $\binom{n}{n-1}$ ways of choosing the $n-1$ variables from the total set of n variables. Thus, we have:

$$|\{degenerate(f_n)\}| \leq \binom{n}{n-1} \times 2^{2^{n-1}}.$$

Taking the limit of the ratio of the number of degenerate functions to the total number of functions yields:

$$\begin{aligned}
\lim_{n\to\infty} \frac{|\{degenerate(f_n)\}|}{|\{f_n\}|} &\leq \lim_{n\to\infty} \frac{\begin{pmatrix} n \\ n-1 \end{pmatrix} * 2^{2^{n-1}}}{2^{2^n}}; \\
&\leq \lim_{n\to\infty} \frac{n}{2^{2^{n-1}}}; \\
&= 0.
\end{aligned}$$

Thus, only a vanishingly small number of the total number of boolean functions are degenerate. To make this somewhat more concrete, of the $2^{2^1} = 4$ functions of one variable, two are degenerate; of the 16 functions of two variables, six are degenerate; of the 256 functions of three variables, 38 are degenerate. Thus, when we consider boolean functions in general, we need only consider those which are non-degenerate.

REFERENCES

[1] M.I. Nathan, J.C. Marinace, R.F. Rutz, A.E.Michel, and G. Lasher, "GaAs injection laser with novel mode control and switching properties," *J. Appl. Phys.*, vol. 36, pp. 473 - 480, 1966.

[2] W. Smith, "Computer Applications of Lasers," *Proceedings of the IEEE*, vol. 54, no. 10, pp. 1295 - 1300, Oct. 1966.

[3] R.W.Keyes and J. Armstrong, "Thermal Limitations in Optical Logic," *Applied Optics*, vol. 8, no. 12, pp. 2549 - 2552, Dec. 1969.

[4] H. M. Gibbs, S. L. McCall, and T. C. Venkatesan, "Optical Bistable devices: The basic components of all-optical system," *Optical Engineering*, vol. 19, pp. 463–468, 1980.

[5] P. W. Smith and W. J. Tomlinson, "Bistable Optical Devices Promise Subpicosecond switching," *IEEE Spectrum*, pp. 26–33, June 1981.

[6] D. A. B. Miller, D. S. Chemla, D. J. Eilenberger, P. W. Smith, A. C. Gossard, and W. T. Tsang, "Large Room-Temperature Optical Nonlinearity in GaAs/$Ga_{1-x}Al_x$As Multiple Quantum Well Structures," *Appl. Phys. Lett.*, vol. 41, no. 8, pp. 697 - 681, 15 Oct. 1982.

[7] C. T. Seaton, S. D. Smith, F. A. P. Tooley, M. E. Prise, and M. R. Taghizadeh, "Realization of an InSb Bistable Device as an optical AND Gate and Its Use to Measure Carrier Recombination Times," *Appl. Phys. Lett.*, vol. 42, pp. 131 - 133, Jan. 1983.

[8] J. L. Jewell, Y. H. Lee, M. Warren, H. M. Gibbs, N. Peyghambarian, A. . C. Gossard, and W. Wiegmann, "3-pJ, 82 Mhz Optical Logic Gates in a Room Temperature GaAs-AlGaAs Multiple Quantum-Well Etalon," *in Appl. Phys. Lett.*, vol. 46, no. 10, pp. 918 - 920, 15 May 1985.

[9] G. Livescu, D. A. B. Miller, J. E. Henry, A. C. Gossard, and J. H. English, "Spatial light modulator and optical dynamic memory using a 6 × 6 array of self-electro-optic-effect devices," *Optics Letters*, vol. 13, no. 4, pp. 297 - 299, April 1988.

[10] A. L. Lentine, H. S. Hinton, D. A. B. Miller, J. E. Henry, J. E. Cunningham, and L. M. F. Chirovsky, "Symmetric Self-Electrooptic Effect Device: Optical Set–Reset Latch, Differential Logic Gate, and Differential Modulator/Detector," *IEEE J. of Quantum Electronics*, vol. 25, no. 8, pp. 1928 – 1936, Aug. 1989.

[11] W. F. McColl, "On the Planar Monotone Computation of Threshold Functions," in *Lecture Notes on Computer Science*, vol. 182, pp. 219 – 230, Springer-Verlag, 1985. Originally in Proc. 2nd Symp. on Theoretical Aspects of Computer Science.

[12] W. F. McColl, "Planar Circuits Have Short Specifications," in *Lecture Notes on Computer Science*, vol. 182, pp. 231 – 242, Springer-Verlag, 1985. Originally in Proc. 2nd Symp. on Theoretical Aspects of Computer Science.

[13] A. Huang, "Parallel algorithms for Optical Digital Computers," in *Proceedings IEEE Tenth Int'l Optical Computing Conf.*, pp. 13 – 17, 1983.

[14] K. H. Brenner and A. Huang, "An Optical Processor Based on Symbolic Substitution," in *Technical Digest, Topical Meeting on Optical Computing*, pp. WA4.1–WA4.3, 1985.

[15] K. H. Brenner, A. Huang, and N. Streibl, "Digital Optical Computing with Symbolic Substitution," *Applied Optics*, vol. 25, pp. 3054 – 3060, 15 Sept 1986.

[16] K. H. Brenner, "New Implementation of Symbolic substitution logic," *Applied Optics*, vol. 25, 15 September 1986.

[17] J. N. Mait and K. H. Brenner, "Optical Systems for Symbolic Substitution," *Topical Meeting on Optical Computing*, vol. 11, 1987. Optical Society of America, Washington, D. C.

[18] K. H. Brenner, "Programmable Optical Processor Based on symbolic substitution," *Applied Optics*, vol. 27, no. 9, pp. 1687 – 1691, 1 May 1988.

[19] J. N. Mait and K. H. Brenner, "Optical symbolic substitution: system design using Phase-only holograms," *Applied Optics*, vol. 27, no. 9, pp. 1692 – 1700, 1 May 1988.

[20] K. H. Brenner, A. W. Lohmann, and T. M. Merklein, "Symbolic Substitution implemented by spatial filtering logic," *Optical Engineering*, vol. 28, no. 14, pp. 390 – 396, 1989.

[21] D. P. Casasent and E. C. Botha, "Multifunctional optical processor based on symbolic substitution," *Optical Engineering*, vol. 28, no. 4, pp. 425 – 433, April 1989.

[22] J. Weigelt, "Binary logic by spatial filtering," *Optical Engineering*, vol. 26, no. 1, pp. 28 – 32, Jan. 1987.

[23] J. Weigelt, "Space-bandwidth product and crosstalk of spatial filtering methods for performing binary logic optically," *Optical Engineering*, vol. 27, no. 10, pp. 883 – 892, Oct. 1988.

[24] H. Jeon, M. A. G. Abushagur, A. A. Sawchuk, and B. K. Jenkins, "Digital Optical Processor Based on Symbolic Substitution Using Holographic Matched Filtering," *Applied Optics*, vol. 29, no. 4, pp. 2113–2125, May, 1990.

[25] J. Tanida and Y. Ichioka, "Optical Logic Array Processor Using Shadowgrams," *Journal of Optical Society of America A*, vol. 73, no. 6, June 1983.

[26] J. Tanida and Y. Ichioka, "Optical-logic-array processor using shadowgrams. III. Parallel neighborhood operations and an architecture of an optical digital-computing system," *J. Opt. Soc. Am. A*, vol. 2, no. 8, pp. 1245 – 1253, August, 1985.

[27] J. Tanida and Y. Ichioka, "OPALS: Optical Parallel Array Logic System," *Applied Optics*, pp. 1565 – 1570, 15 May 1986.

[28] J. Tanida and Y. Ichioka, "Modular Components for an Optical Array Logic System," *Applied Optics*, vol. 26, pp. 3954 – 3960, 15 Sept. 1987.

[29] R. Arrathoon and S. Kozaitis, "Shadow casting for multiple-valued associative logic," *Optical Engineering*, vol. 25, no. 1, pp. 29 – 37, Jan. 1986.

[30] P. S. Guilfoyle and W. J. Wiley, "Combinatorial Logic Based digital optical computing architectures," *Applied Optics*, vol. 27, no. 9, pp. 1661 – 1673, May 1988.

[31] M. J. Murdocca, A. Huang, J. Jahns, and N. Streibl, "Optical Design of Programmable Logic Arrays," *Applied Optics*, vol. 27, pp. 1651 – 1660, May 1988.

[32] M. J. Murdocca and T. J. Cloonan, "Optical Design of a Digital Switch," *Applied Optics*, vol. 28, no. 13, pp. 2505 – 2517, 1 July 1989.

[33] M. J. Murdocca, "Design of a symbolic substitution-based optical Random access memory," in *Technical Digest, Topical Meeting on Optical Computing*, vol. 9, pp. 92 – 95, Feb. 1989.

[34] M. J. Murdocca, "Computer–Aided Design of Digital Optical Circuits," in *Proc. Asilomar Conf. ?*, October, 1989.

[35] M. J. Murdocca, "Fault Avoidance for Optical Logic Arrays and Regular Free–Space Interconnects," in *Proc. 1990 O-E/Lase Conf.*, 1990.

[36] J. E. Savage, *The Complexity of Computing.* New York: Wiley, 1976.

[37] F. J. Hill and G. R. Peterson, *Switching Theory and Logical Design.* New York: J. Wiley, 1981.

[38] A. Louri, "Three–Dimensional Optical Architecture and Data–Parallel Algorithms for Massively Parallel Computing," *IEEE Micro*, pp. 24 – 27, 65 – 82, April, 1991.

[39] C. D. Thompson, *A Complexity Theory for VLSI.* PhD. Thesis, Carnegie-Mellon University, Computer Science Department, 1980.

[40] C. E. Shannon, "The Synthesis of Two-Terminal Switching Circuits," *Bell System Technical Journal*, vol. 28, pp. 59 – 98, 1949.