Complexity analysis of optical-computing paradigms

Ahmed Louri and Arthur Post

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 that prevents their direct extension to functions of a large number of input bits. We analyze several of these implementations and demonstrate that all these implementations require that some measure of the system (area, space–bandwidth product, or time) grow exponentially with the number of inputs. We then suggest an implementation whose complexity is no greater than the best theoretical realization of a Boolean function. We demonstrate the optimality of that realization, to within a constant multiple, for digital optical-computing systems realized by bulk spatially variant elements.

1. Introduction

Measures of gate count are as important in designing Boolean functions as are measures of operations in program design. In designing algorithms, the notion of an upper bound to the number of computations required for implementing a given function is fundamental to the analysis of the run time of an algorithm. On the other hand, a knowledge of lower bounds on the number of computations can provide an understanding of when an algorithm is optimal, and when no further reduction in operations is possible. Similar observations can be made of Boolean functions. In general, a design method that can implement any Boolean function may not provide optimum solutions for certain special cases. Shannon[1] demonstrated lower bounds to the cost, in numbers of gates, for almost all functions, mapping n Boolean input variables to a single output of $\Omega(2^n/n)$, where $\Omega()$ represents the lower bound notation

$$f(n) = \Omega[g(n)] \Leftrightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} \le c, \tag{1}$$

where c is a constant, $f(n)$ is, in this instance, the cost of implementing a Boolean function, measured in numbers of gates, $g(n)$ represents the function that bounds the cost, and n is the number of inputs. Thus Shannon's theorem states that the number of gates that are required for implementing almost all functions cannot be less than a constant multiple or fraction of $2^n/n$ as the number of inputs increases. The qualification that the theorem is true for almost all functions of n inputs means that the ratio of the number of functions for which the theorem is true to the total number of functions of n inputs tends to unity as n increases without bound. A more detailed discussion is included in Appendix A.

While the Shannon limit applies to the general case, certain functions may have an inherent structure that can be exploited to provide significant reductions in complexity order, which a general design methodology cannot achieve. As a concrete example of this sort of reduction in gate count, consider the simple full adder. By using standard minimization techniques, we see that the carry function has three prime implicants, while the sum function has four prime implicants, and these implicants are minterms. Thus it would appear that the sum function cannot be minimized, and the resultant realization as a direct implementation of the sum-of-products formulas requires four 3-input AND gates and one 4-input OR gate for computing the sum, along with three 2-input AND gates and one 3-input OR gate to compute the carry. Further, in the computation of the sum function, all the inputs appear in each of the terms, requiring that each input be mapped four times (twice as the negation of the variable), while the computation of the carry requires that each input be mapped twice; the result is a total fan-out of 5 from the input nodes and 2 from the inverters. Construct-

The authors are with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, Arizona 85721.
Received 24 May 1991.
0003-6935/92/265568-16$05.00/0.

ing this circuit from 2-input gates alone requires eleven AND gates, three inverters, and five OR gates. But the fact that the Karnaugh map of the sum is that of the odd-parity function (which can be implemented by two XOR gates as $A \oplus B \oplus C$), along with the fact that the carry can be realized as

$$C = AB + (A \oplus B)C \tag{2}$$

results in an implementation that has two XOR gates, two AND gates, and one OR gate while requiring a maximum fan-out from any one input of only 2. While the above example is quite simple, taking advantage of inherent structures of certain functions can produce large reductions in the complexity of many functions.

Other considerations, such as restrictions imposed by the nature of the target environment, can impact the size of a circuit realization to a significant degree. For example, imposing restrictions on the nature of a circuit graph can significantly increase the complexity of a design. McColl[2] has studied the behavior of circuits when they are restricted to strictly planar implementations (i.e., no edges may cross), and has shown that almost all functions, when implemented in this way, require a minimum of 2^n gates, where n is, again, the number of inputs to the function. This minimum is a factor of n greater than the lower bounds for almost all functions derived by Shannon. It is interesting to note that the restriction imposed on circuit graphs in this paper is a restriction on the edges of the graph; most previous work in Boolean function theory has imposed limitations on only the basis functions employed. That such a seemingly simple restriction on the graphs that implement a circuit should have this kind of impact on circuit size has important implications for optical computing since the use of spatially invariant interconnections imposes rigid limitations on the interconnections in a circuit graph.

Systems that employ both spatially invariant and spatially variant imaging have been proposed in the literature. For example, Brenner *et al.*[3] discuss spatially invariant imaging systems for interconnections between gates, while Jenkins *et al.*[4] implement interconnections by spatially variant imaging techniques. It is the thesis of this paper that, while systems constructed from spatially invariant imaging techniques seem to be cheaper to build than spatially variant systems, the nature of the restrictions imposed on the graphs that can be constructed from spatially invariant interconnections forces these systems to be much larger and costlier than spatially variant systems, particularly when the number of inputs to a function constructed in this manner becomes large.

In discussing this issue we make use of the order notations frequently used in computer science. In addition to the $\Omega()$ notation used above, we use two other similar notations in this text. The $O()$ notation is the dual of the $\Omega()$ notation in the sense that it is an upper-bound notation and has the precise meaning

$$f(n) = O[g(n)] \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c. \tag{3}$$

Equivalently the function $f(n)$ can increase no faster than a constant multiple of $g(n)$, or, in the limiting case, as n grows without bound. A special case of this notation is $f(n) = O(1)$, which has the meaning that the function is a constant, or finds a limit in a constant value.

We also make use of the $\Theta()$ notation, which can be defined in terms of the $\Omega()$ and $O()$ notations as

$$f(n) = \Theta[g(n)] \Leftrightarrow$$
$$f(n) = O[g(n)] \wedge f(n) = \Omega[g(n)]. \tag{4}$$

That is, again in the limiting case, $f(n)$ will increase no faster than a constant multiple of $g(n)$, nor slower than a (possibly different) constant multiple of $g(n)$; less precisely, $f(n)$ is bounded above and below by $g(n)$.

We briefly analyze several proposed optical-computing paradigms constructed by spatially invariant techniques in Section 2. In Section 3, we implement several of the 2-input functions by using a simple spatially variant imaging technique. This same technique is extended in Section 4 for constructing some larger functions. The system we present in Sections 3 and 4 is described in architectural and computational terms. A comparison of the various techniques in terms of underlying complexity is presented in Section 5, with conclusions and suggestions for further research discussed in Section 6. We include an amplification of Shannon's theorem Appendix A.

2. Complexity Analysis of Several Optical-Computing Paradigms

In the interest of continuing the development of digital optical computing, and to demonstrate the applicability of the field of Boolean complexity theory to optical systems, in this section we analyze several of the proposed implementations of digital optical-computing systems from a system complexity viewpoint, and show that these systems have similar system complexities. The systems that we detail are symbolic substitution logic, which is implemented by additive methods[3,5,6] and by convolution or correlation techniques,[7–10] shadow-casting logic,[11–14] the programmable logic of Murdocca *et al.*,[15,16] and the combinatorial logic-based system of Guilfoyle and Wiley.[17] We analyze these implementations in terms of their complexity and, hence, the ability of these systems to compute Boolean functions of sizes comparable to those currently achieved in electronic systems.

While it might be argued that our choice of Boolean function implementations is unsuitable for the optical domain, we note that some model of what computing means is required, and that the two major bodies of study in computation center around Turing machines and Boolean functions. Because of the lack of some other model of computation, we must formulate our theories of optical computation around one of these models. Since the optical techniques we wish to study incorporate the implementation of Boolean functions, we restrict our discussion to this model.

A. Symbolic Substitution Logic

The basic idea behind symbolic substitution logic (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. Three primary methods for performing SSL have been advanced, both based on spatially invariant optics. The first method, which is 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. The second method involves filtering in the spatial frequency domain, and is commonly known as spatial filtering logic. A more recent method, proposed by Louri,[18] combines shadowcasting logic (which is discussed below) with polarization techniques to address some of the optical engineering questions involving SSL. We analyze the first two techniques separately.

1. *Additive Logic*

Data is typically represented in additive logic-based SSL in a dual-rail intensity-coded format, in which two pixels are used to encode each bit, although other codings are possible (see, for instance, Refs. 19 and 20). The locations of these patterns may be defined by masking, or by providing detectors at only 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 the location in the detector planes to which the dark pixels of each bit are mapped.

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 bits is 2^n. In order to implement completely specified digital logic, each of these 2^n combinations (which correspond to the 2^n possible minterms) must be generated. The reason for this may be seen from the following argument. Suppose that less than the 2^n combinations are generated in space and imaged onto the corresponding detection planes. Then some combinations of inputs will exist that will not be detected by the system. For these combinations, no determinate output will be generated, and some indeterminate value will result. In the typical dual-rail coding scheme, the result is the substitution of two dark pixels in the position in the output plane at which the function is to be computed. But in 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 the set $\{0, 1\}$. If the data are to be processed simultaneously, 2^n image planes must be formed at discrete locations in space. If each image plane has the same area as the input plane, the area occupied by the detection planes will be $\Omega(2^n)$ times that of the input plane.

This result is due to the nature of the paradigm, not to a function of the encoding scheme that is used to represent the data. Intensity encoding can achieve a twofold reduction in the area of the input plane that is required for representing the inputs; or, alternatively, a twofold increase in the density of data in this plane. The nature of the paradigm does not change, however, and still requires separate detection of each of the possible combinations of the inputs. 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), and (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 dual-rail encoding. Louri[19] has proposed a technique in which pairs of dual-rail inputs are superimposed before imaging, which reduces the number of rules in the initial detection phase to three; he has also demonstrated an increase in computational throughput for addition by this method. The method does require that a fourth rule be applied as a separator in the final step to discriminate between outputs that the technique has mapped to a code outside the range of the input coding of data.

The actual complexity of an SSL implementation may be greater than that discussed 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. *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, filtered, and the inverse Fourier transform is then taken of the filtered image. The principle is the same as for additive logic: all combinations of the inputs are detected, and suitable outputs are generated for these combinations. A number of input codings may be used: Casasent and Botha[9] suggest a dual-rail coding similar to that discussed above for the additive logic implementation, while Jeon et al.[10] propose a somewhat more complex encoding to improve cross-talk characteristics. Brenner et al.[8] and Weigelt[21] suggest several alternate codings that achieve a spatial frequency modulation representing the data. What is common to these techniques is that a separate filter must be used for each pixel pattern to be detected, regardless of the encoding technique. Thus four filters are required in order to detect patterns representing two binary inputs. Implementing Boolean functions of three inputs requires eight filters; ultimately 2^n filters are required for computing a function of n inputs. In the methods of Weigelt,[7,21] and Casasent and Botha[9] these filters are spatially multiplexed to create a single filter upon which the input plane or planes are imaged. But each each filter has a space–bandwidth product that is a function of the space–bandwidth product of the input plane, with the resultant bandwidth increasing as a function of the number filters, and hence as a function of 2^n. In the technique proposed by Jeon *et al.*[10] 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 that spatial bandwidth of the filters, increase exponentially with the number of inputs to implement Boolean functions correctly, just as the number of detector planes increases exponentially in additive logic-based SSL.

B. Shadow-Casting Logic

A second implementation of digital optical computing, attributable to Tanida and Ichioka,[11–14] is known as shadow-casting logic. In this method 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 one another and are 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 resultant combined cell being transparent. By having multiple SLM's combined with suitable locations for the detectors, any of the n-input functions may be computed. The number of light sources and detectors is dependent on the manner in which the system is to be used.

While this method has the advantage of simplicity in implementation [the light is provided by light-emitting diodes 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. Further, the number of sources is equal to the number of minterms in the ON set of the implemented function (those minterms that map to the value 1 in the function's output). The number of sources can then vary from one to a maximum of 2^{n-1}. (Should there be more than 2^{n-1} minterms constituting the ON set of a function, simply detecting the minterms mapping to 0, which is the OFF set, and inverting yields a number less than 2^{n-1}.) For good functions, such as the n-input AND, only one source is required, while a function such as the parity function requires 2^{n-1}.

This exponential complexity was first described by Arrathoon and Kozaitis,[22] who then proposed the use of this technique to provide an associative look-up for multivalued logic. While these authors are able to show some improvement in power and space bandwidth, their system has many of the same drawbacks as the original method.

The method also 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 the input plane for further computation.

C. Combinatorial Logic

Another computing paradigm, from Guilfoyle and Wiley, is called combinatorial logic, which divides the computational load between the electronic and the optical domains.[17] In this method, a pair of multichannel acousto-optic cells are telecentrically imaged, with data within the cells counterpropagating. We 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 resultant AND–OR structure is shown in Fig. 1.

As noted above, 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 before 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 a fraction of the computations and that the computational capability of the optical portion of the system is limited. The spatially invariant optical subsystem can be modeled as a set of n-2-input AND gates, whose outputs are connected to the inputs of an n-input OR gate. We assume that the function to be computed depends on all inputs (is nondegenerate), and that 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 func-

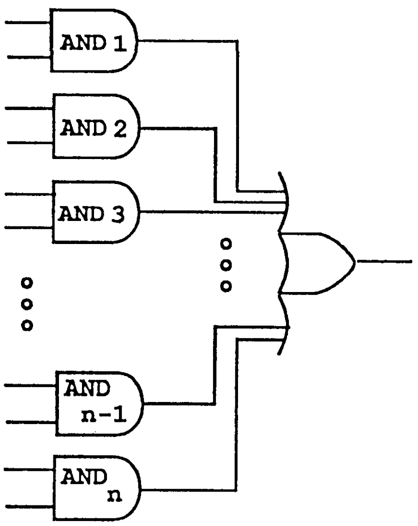

Fig. 1. AND–OR structure of the optical subsystem of combinatorial logic.

tions 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 in $n!$. Thus the total number of possible assignments of the $2n$-input variables to the $2n$ inputs of the AND level is $n! \times n! = (n!)^2$.
- The number of different possible Boolean functions of $2n$ variables is given by

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

- The number of degenerate functions of $2n$ variables (which are denoted by $\tilde{f}_{2n}$) can be shown to be

$$|\{\tilde{f}_{2n}\}| \leq \binom{2n}{2n-1} 2^{2^{2n-1}} = 2n 2^{2^{2n-1}}.$$

- Thus the number of nondegenerate functions of $2n$ inputs is given by

$$\begin{aligned} |\{f_{2n} - \tilde{f}_{2n}\}| &\geq 2^{2^{2n}} - 2n2^{2^{2n-1}} \\ &\geq (2^{2^{2n-1}} - 2n)2^{2^{2n-1}} \\ &\geq 2^{2^{2n-1}}. \end{aligned}$$

- Taking the limit of the ratio of the number of functions computable by the AND–OR tree to the number of nondegenerate functions establishes the result:

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

One example of a function that this spatially invariant optical subsystem cannot perform is the parity function over $2n$ inputs. 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 1. Hence the optical subsystem alone is not generally capable of computing Boolean functions.

D. Programmable Logic

Murdocca *et al.* have proposed a method of digital optical computing that uses optical nonlinearities directly as a form of logic gate.[15,16] In this method, planes of these logic gates are interconnected by a spatially invariant imaging system that is constructed from beam splitters and prism arrays, with the resultant interconnection pattern forming a crossover network.

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 crossover network and one mask per stage is 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 in which 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.

It has been asserted that the size of the circuits realized in this manner is optimal, being $\Omega(n2^n)$ in the number of inputs, and that the number of levels in the circuit is optimal, being linear in the number of inputs. 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 this approach has lower bounds that are exponential in the number of inputs, regardless of the function implemented. Thus, while a given function might be optimally computed by a circuit with size $O(n^2)$ and depth $O(\log_2 n)$ (see Ref. 23), programmable logic will not allow minimization to this level. Even for random functions whose size is $\Theta(2^n/n)$, this technique requires $O(n^2)$ more gates than the optimal circuit realization.

In order to argue that the spatially invariant interconnection pattern does not greatly increase the cost of implementation over that of a spatially variant implementation, a spatially variant implementation of the full adder[15] that requires 78 AND gates, 11 OR gates, and 6 levels is compared with the 128 gates and 8 levels required of the present method. But the spatially variant implementation generates all the monoms (not minterms) of the function, resulting in a circuit that is not minimal. The minimal full adder circuit, which was discussed above, requires only 5 gates (two XOR, two AND, and one OR), and has a depth of 3. An alternate realization of the full adder has a gate count of 6 (two XOR, three AND, and one OR) and a depth of 2. Even when restricted to the use of NOR gates alone, a minimized full adder should require no more than 14 such gates (although the resultant depth is 7). While it is possible that a spatially invariant full adder circuit could be reduced to 48 AND and OR gates, the result would still not be minimal.

While this technique does share the features of topological regularity and the use of AND and OR gates with a PLA design, a true PLA design has only two levels of logic, regardless of the number of inputs, and typically does not require that all minterms of a function be generated. Rather, the PLA user develops one or more equations (depending on the number

Table 1. Comparison of the Number of Functions Implementable in the Spatially Invariant Subsystem of Combinatorial Logic, and the Total Number of Nondegenerate Functions versus the Number of Input Variables

n	$2n$	$n!$	$n!^2$	$2^{2^{2n-2}}$	$2^{2^{2n-1}}$
3	6	6	36	65 536	4 294 967 296
4	8	24	576	2^{64}	2^{128}
5	10	120	14 400	2^{256}	2^{512}

of outputs required) that are in minimized sum-of-products form. It is the product terms of these equations that 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, a ring-sum expansion will frequently minimize these functions and permit an economical circuit realization in XOR logic. Although a minimized circuit has been demonstrated that does not generate all minterms and then select from them, the breadth of the ciruit is still exponential in the number of inputs; hence the circuit is no more minimal than would be an electronic circuit that consists of an n-to-2^n demultiplexer, whose outputs are then connected to one or more trees of OR gates, which generate the resultant function.

E. 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 direct and profound ways:

- Only those functions, such as addition, for which there are serial implementations, can be exploited, and the full potential for parallelism in optical computing will not be developed.
- Control functions, which typically cannot be serialized, will remain in the electronic domain.
- Other functions that require global communications will require some other computational domain for solution.

These systems also lack the capability of interconnecting even the small functions that have been implemented in the arbitrary manner that 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 that prohibits the economical construction of large functions from the smaller basis functions.

3. New Approach

The inability of the systems discussed above to compute Boolean functions with less than exponential complexity severely limits the application of these systems even in terms of their ability to perform special-purpose computing in conjunction with electronic hosts. Thus we must look for new ways of achieving digital optical computing that do not suffer from the shortcomings of the previous implementations, but that still provide the advantages that optical computing promises.

Whether optical computing remains an adjunct to electronic computing, or whether general-purpose optical computers are eventually developed, will depend primarily on whether optical random access memory is developed. Since optical memory is not currently available, we restrict our discussion to the use of optical computing as an adjunct to the electronic domain. In this regard, then, we would expect an optical-computing adjunct to:

- Use nonlinear optics directly as logic gates, and require a number of such gates that is commensurate with the best theoretical implementation;
- Allow the parallel mapping of registers, or register pairs, to the input of the optical system, and output the results to one or more entire register;
- Perform a large amount of computation on the registers before returning the results to the electronic adjunct (the addition of two 8-bit registers would be near the low end of this range);
- Perform the computations as quickly as possible, utilizing parallel implementations of the functions rather than serial implementations.

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 by using the greatest amount of parallelism possible within each function.

While many of the optical-computing paradigms proposed in the past have demonstrated the basis functions that we demonstrate, these previous implementations have relied on spatially invariant interconnections of one form or another to achieve their goals. What we submit here is a simple spatially variant construction of the basis functions, which can then be extended for constructing larger functions from these elementary ones. While the implementation proposed here is constructed from simple spatially invariant imaging elements, advances in holographic techniques could substantially improve the speed and size of our implementation.

A. NOR Function

We begin our development by demonstrating the 2-input NOR function applied to an object plane of $N^2 = 2n$ pixels, which computes n of these functions simultaneously. We assume that the data pairs are distinct and have the same spatial relation to each other. This requirement is imposed by the spatially invariant nature of the imaging systems we use. We do not, in general, assume that the data elements are adjacent, and we make use of the simple spatially variant interconnections to perform computations on pairs of elements that are not adjacent. The realization of the NOR function is shown in Fig. 2, with a schematic representation of the realization shown in Fig. 3.

In Fig. 2, 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

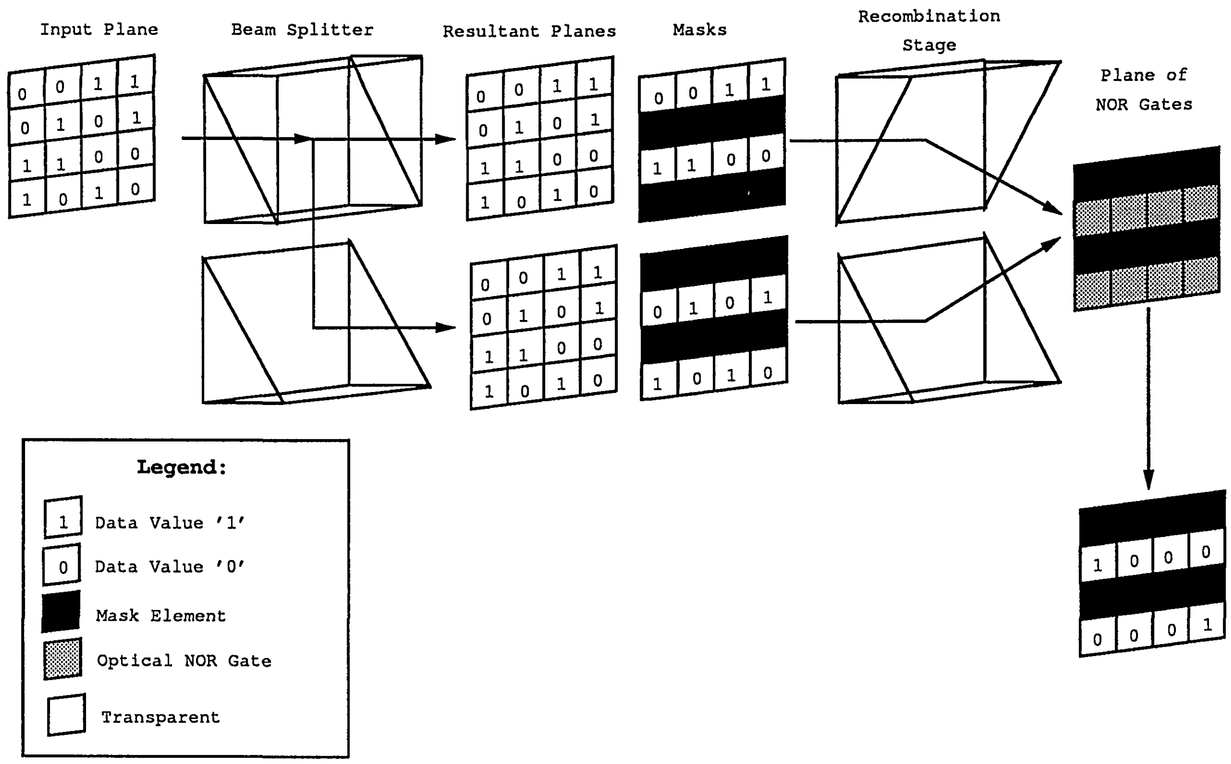

Fig. 2. Implementation of the 2-input NOR function.

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 that is to be NORed is incident on the nonlinear element that 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 that may be used to construct significantly larger functions in which computations may occur between inputs or intermediate results that are not physically adjacent.

The schematic representation of Fig. 3 is derived from the implementation shown in Fig. 2 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 rely on schematic representations in the discussions below.

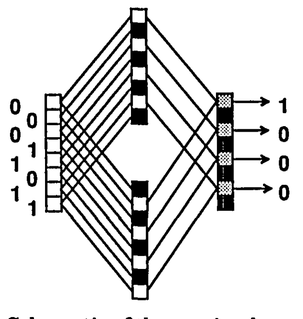

Fig. 3. Schematic of the NOR implementation.

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 from SSL.

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 2-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, for performing 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 devices, implementation of the NOR function by this method also requires 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 2-bit vector, and that the complements of the inputs be provided in addition to the noncomplemented inputs.

B. NOT Function

While not a 2-input function, the NOT function, or the ability to achieve the function by other gates, is essential for computing any nonmonotonic function. The implementation of this function by the NOR is direct, and is shown schematically in Fig. 4. This implementation is trivial and requires little comment. The main feature of this implementation that should

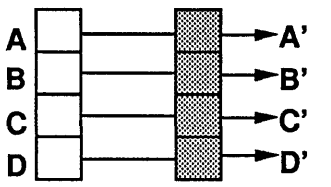

Fig. 4. Schematic of the NOT implementation.

be noted is that if the intensity of the input plane is sufficient to permit 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.

C. OR Function

The OR function may be obtained from the above two functions by simply inverting the output of the NOR. This implementation is shown in Fig. 5. The NOT function must be implemented with planes of NOR functions with the same active device density as that of the plane implementing the NOR.

D. 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 Fig. 6. 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 required by SSL or programmable logic.

E. Equivalence (X-NOR) and the XOR Function

In the above sections, we developed implementations for two complete basis sets, the set {NOR} and the set {AND, OR, NOT}. Either of these sets is sufficient for computing all the Boolean functions. The ability to compute the XOR and the equivalence or X-NOR functions provides flexibility in designing a variety of circuits. We first detail the design of the X-NOR.

The logic diagram of the X-NOR is shown in Fig. 7. As can be seen, construction of 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 for implementing any 2-input function. The freedom allowed by spatially variant interconnections, however, permits us to implement the inverting NOR gates at level one and the NOR gates in level two as single fully populated NOR planes for each level. This realization is shown in Fig. 8. A careful examination of this figure reveals that we have provided the five gates that 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 used instead an alternative implementation that requires only the same number of NOR gates as the X-NOR. The resultant logic diagram shown in Fig. 9. The implementation of this alternate realization of the XOR is shown in Fig. 10. This implementation is also somewhat more expensive than that which results 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 requires that spatially variant interconnections be used to construct Boolean functions. We have also given several characteristics that we believe are 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. In Section 4 we expand this discussion to the construction of larger functions, which are optimal in gate count and depth.

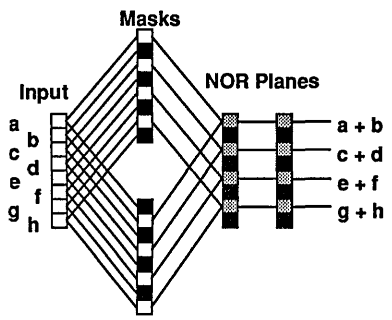

Fig. 5. Implementation of the OR function.

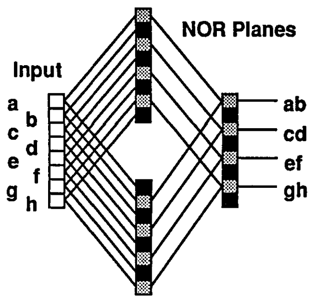

Fig. 6. Implementation of the AND function.

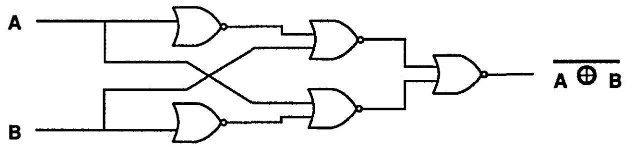

Fig. 7. Logic diagram of the X-NOR or equivalence function.

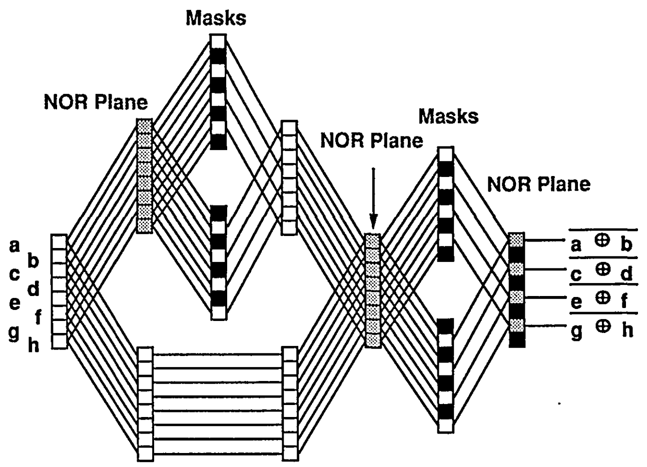

Fig. 8. Implementation of the X-NOR or equivalence function.

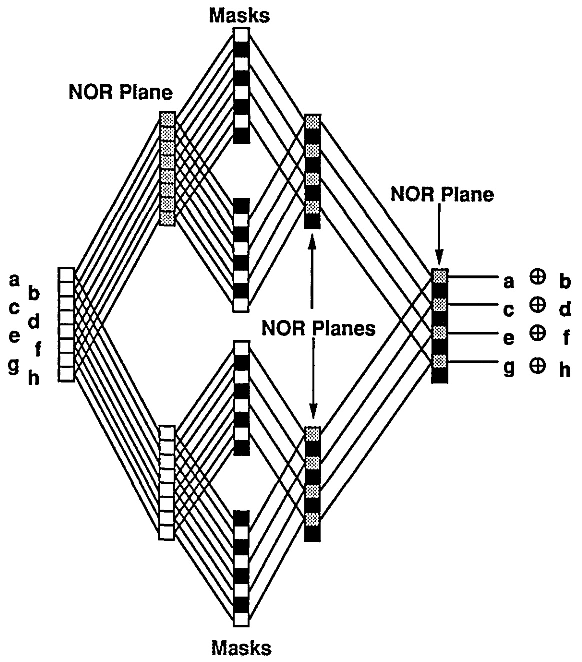

Fig. 10. Implementation of the XOR.

4. Construction of Larger Functions

Here we illustrate the extension of our technique to functions of larger numbers of inputs. In this extension, we develop all functions from the basis gate NOR, as was done in Section 3. This restriction on gate selection increases the gate count we require for implementing a given function, and thus tends to impose worst-case gate counts on our implementations. This is in keeping with our general desire to demonstrate that, even when such conditions are imposed on the proposed method, this method still retains a significant advantage over other methods that have been forwarded. This advantage is a reflection of the underlying complexities involved in the functions we demonstrate. Since these functions have implementations that are less complex than those imposed by SSL and other paradigms, and since the use of the 2-input NOR gate as the sole gate function in our implementation can affect only the complexity of the systems we implement by, at most, a constant multiple, we are able to demonstrate the superiority of our proposed method.

A. Arbitrary 3-Input Function

The first function that we implement has 3 inputs, and can be described by the shorthand notation for a sum-of-products form:

$$\begin{aligned} f(a, b, c) &= \sum (m_0, m_2, m_3, m_6) \\ &= \bar{a}b + \bar{a}\bar{c} + b\bar{c} \\ &= \bar{a}(b + \bar{c}) + b\bar{c}. \end{aligned} \quad (5)$$

This function would require two AND, two OR, and three NOT gates to implement in factored form. In the implementation shown in Fig. 11, the function requires eight NOR gates. It should be noted that only the uncomplemented inputs are mapped to the input plane, and that all complementation results in an increased gate count.

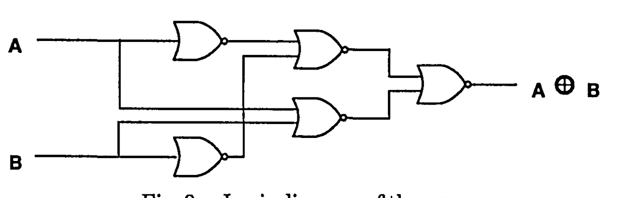

Fig. 9. Logic diagram of the XOR.

In Fig. 11 there are three basic elements: masks, image intensifiers, and NOR gates. The process of constructing the optical circuit begins by replicating the input image (the column at left) as many times as there are input variables. 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 $\bar{c}$. These variables are then mapped in pairs to the first level of the computation graph. The output of this first level is a set of five images (including the image of the variable a, which does not participate in any computation at this level), which is then focused on the gate planes at the second level of computation. The process continues until the final gate computes the desired function.

Notice from Fig. 11 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 due to the inherent parallel nature of 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 upon which we operate. Thus we can achieve both masking and gate functions with one plane. This will be the case with all functions we demonstrate in this section.

Comparing this circuit realization to those obtained by other methods, we note that there are actually ten active elements in the optical realization, including two intensifying elements. An additive

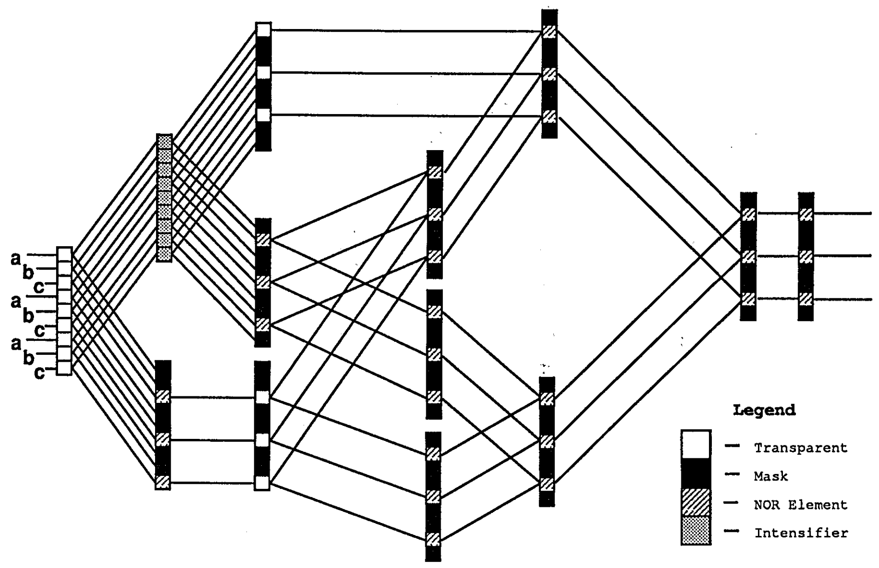

Fig. 11. Optical implementation of the 3-input function.

logic SSL implementation requires eight detection planes, which is an apparent saving over our realization. An analysis of the energy requirements of the two implementations shows that the energy required at each pixel of the input plane in an additive logic implementation is 24 times the switching energy threshold of a given detector because of the need to provide three shifted copies of the input plane for each detection plane. In the above implementation, the energy required is proportional only to the number of NOR gates and the number of intensifiers. Hence the energy required for implementing this circuit is less than half that of an SSL implementation. A programmable logic implementation requires a minimum of 48 gates, and hence more than four times the energy required here. A shadow-casting implementation requires four sources, but masks one-eighth of the light incident on each computation cell from the detector, hence requiring that the input energy per cell be 32 times the detector threshold energy.

B. Implementation of Two Adders

We now turn our attention to some practical applications, the full adder, and the carry look-ahead adder. While numerous papers have demonstrated designs of the serial ripple-carry adder, we extend this circuit to demonstrate the design of the parallel carry look-ahead 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 that should accompany the use of optical computing elements. After first demonstrating the implementation of the full adder, we demonstrate one possible implementation of the carry look-ahead adder with spatially variant imaging. Specifically, we demonstrate the design of an adder that has two 2-bit vectors and a carry as its inputs, and that computes a 2-bit sum vector and a carry as its outputs. 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 4- or 8-bit adder would be more practical in a real application, the size of these adders merely complicate the figures without illustrating the design technique we demonstrate.

1. *Full Adder*

The design of the full adder was addressed in Section I. We synthesize an optical realization of the full adder on the basis of NOR from those equations. Although there are several ways to implement the full adder, the method we have chosen requires five gates and three levels in XOR-AND-OR logic, and has a maximum fan-out from any one gate of two. While there is a two-level design, the total gate count is six, and the fan-out of some of the nodes is greater also. The choice of this implementation was dictated by the minimality in the number of gates rather than fan-out restrictions. This realization is shown in Fig. 12.

An adder computing an $n + 1$-bit result from a $2n + 1$-bit input can be constructed from this circuit by a process of replication, masking, and spatially variant imaging in the same manner as the construction of the full adder. The resultant ripple-carry adder would have a gate count that would increase $[O(n)]$ and a depth [also $O(n)$]. The details of this adder are omitted so that we may concentrate on the carry look-ahead adder.

2. *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

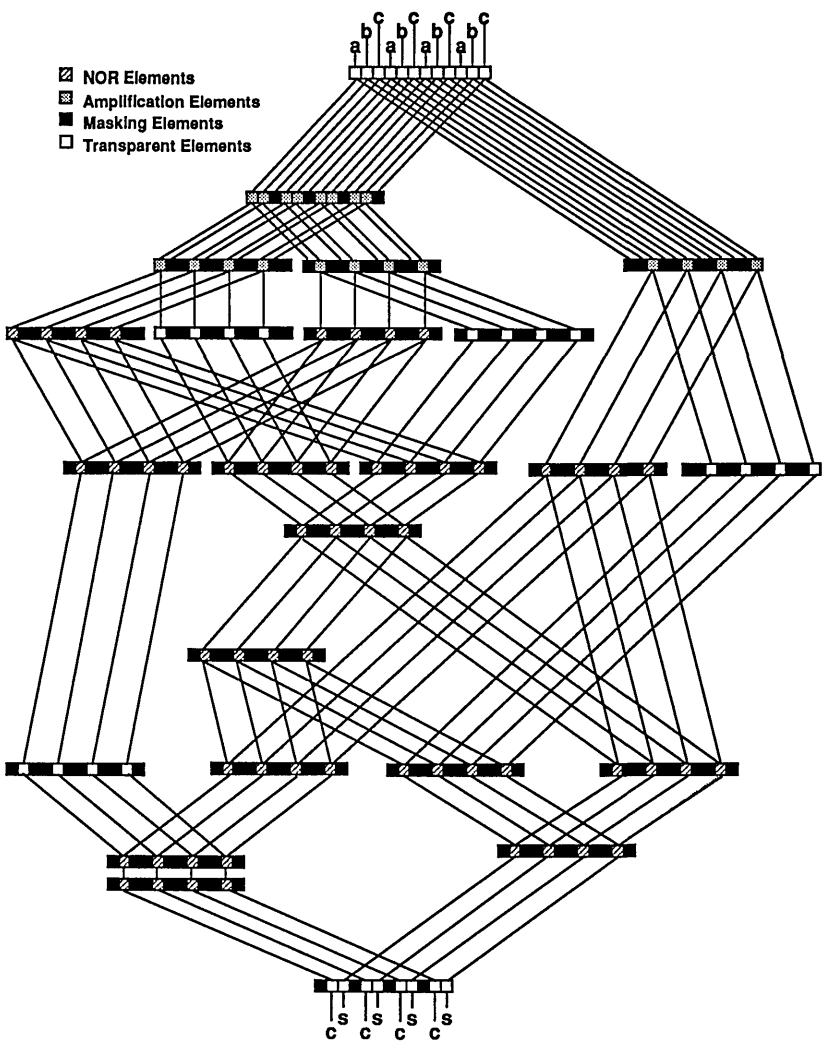

Fig. 12. Optical implementation of the full adder.

inputs, and generates an $n + 1$-bit result. The 2-bit carry look-ahead adder is shown in Fig. 13. In Fig. 13 the carry look-ahead adder is implemented on the basis of the set {AND, OR, XOR}. The circuit operates by computing in parallel a pair of intermediate signals for each pair of inputs. These signals are shown in Fig. 13 as G_i and P_i, where $i = 1, 2$. The signal $G_i = A_iB_i$ 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 with $2n + 1$ inputs will then generate $2n$ of these intermediate signals. From this information, the carryout of the ith bit position can be computed as

$$C_i = G_i + (P_iG_{i-1} + \cdots + (P_iP_{i-1} \cdots P_2G_1) + (P_iP_{i-1} \cdots P_iC_0), \tag{6}$$

while the corresponding sum is computed as

$$S_i = P_i \oplus C_{i-1}. \tag{7}$$

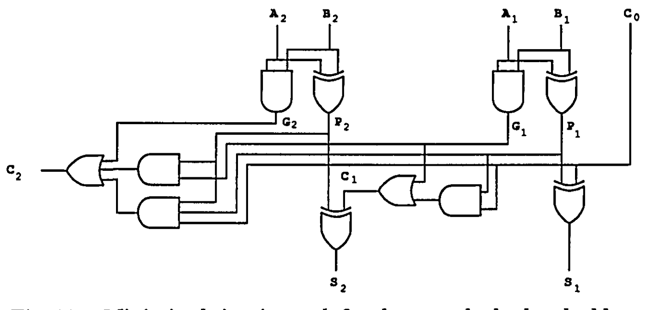

Fig. 13. Minimized circuit graph for the carry look-ahead adder.

The cost in the number of gates and the delay in realizing the adder for pairs of n inputs on the basis of the set {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. Hence for each sum output at least three gates are required. The carry C_i requires one $i + 1$-input OR gate, and i AND gates with $(2, 3, \ldots, i, i + 1)$ inputs. The $i + 1$-input OR gate may be implemented by a binary tree whose nodes are 2-input OR gates. For $i = 0$, there is no cost since the carry C_0 is an input. If $i = 1$, then only one 2-input OR is required. For $i > 1$, i 2-input OR gates are required, and the depth of the OR is $\lceil \log_2 i \rceil$. Thus the $i + 1$-input OR may be replaced by a circuit consisting of i OR gates, with a depth of $\lceil \log_2 i \rceil$ 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 $j - 1$ 2-input AND gates ($j \geq 2$). Thus, the total number of 2-input AND gates required to implement the carry for the ith bit is

$$C_{\text{AND's}} = \sum_{j=2}^{i+1} (j - 1) = \sum_{j=1}^{i} j = \frac{i(i + 1)}{2}, \tag{8}$$

while the depth of the resulting tree is the depth of the largest subtree, which is

$$D_{\text{AND's}} = \lceil \log_2(i + 1) \rceil. \tag{9}$$

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

$$\begin{aligned} C_{\text{CLA}} &= \sum_{i=1}^{n} \left(3 + i + \frac{i(i + 1)}{2}\right) \\ &= \sum_{i=1}^{n} \left(3 + \frac{3}{2} i + \frac{1}{2} i^2\right) \\ &= 3n + \frac{3}{2}\frac{n(n + 1)}{2} + \frac{1}{2}\left(\frac{n}{6} + \frac{n^2}{2} + \frac{n^3}{3}\right) \\ &= \frac{1}{6} n^3 + n^2 + \frac{23}{6} n, \end{aligned} \tag{10}$$

where CLA is the carry look-ahead adder.

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

$$D_{\text{CLA}} = \lceil \log_2 n \rceil + \lceil \log_2(n + 1) \rceil + 2. \tag{11}$$

The implementation to the 2-bit carry look-ahead adder is shown in Fig. 14. 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 minimization 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 Fig. 15.

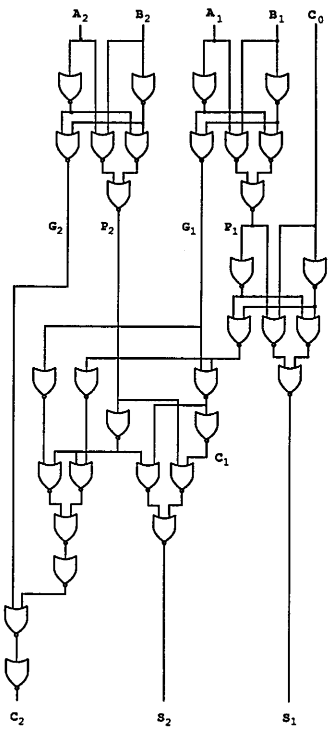

Fig. 14. Minimized NOR gate implementation of the carry look-ahead adder.

5. Comparisons with Other Methods

Having developed our method and demonstrated several functional implementations, we can now demonstrate the viability of our method in relation to other digital optical-computing methods that have been advanced. In terms of complexity, our method has advantages over all the other methods analyzed, since the 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 2.

A. Present Method Compared With 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 not way to reduce this requirement without leaving the Boolean domain. While SSL is a viable method for certain classes of algorithms that can be transformed to serial implementations operating on many sets of a small number of bits,[24] or those algorithms 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 has 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 that is equal to the

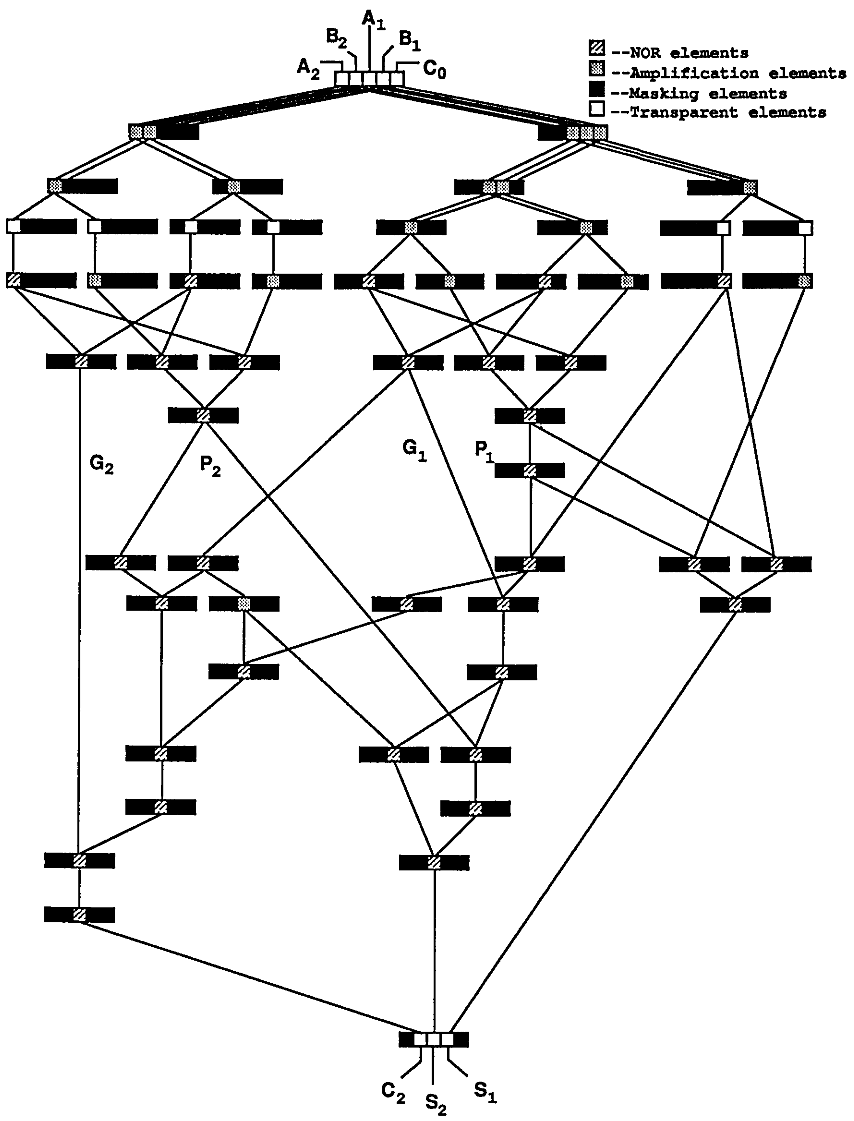

Fig. 15. Optical implementation of the carry look-ahead adder.

Table 2. Order Analysis of an n-Bit Parallel Adder by Using the Methods in This Paper[a]

Method	Number of Gates, Light Sources, or Detection Planes	Circuit Depth or Time Complexity	Normalized Energy 1 Gate/ Detector = 1
SSL	$\Omega(2^n)$	$O(1)$	$\Omega(n2^n)$
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)	$O(n^3)$	$O(\log_2 n)$	$O(n^3)$

[a]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.

number of input variables {a process which has an $O(n)$ gate or amplification count, and a depth $O[(\log_2 n)]$}, the subsequent computations are performed in a minimal number of operations, provided that 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 that have an abstract connection only to the physical realization, but that 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 by an additive linear term only, and the depth will be increased by an additive logarithmic term only. More concretely, given a function whose implementation requires an $O(n^3)$ gate count and an $O(\log_2 n)$ depth, we can implement the function optically with costs:

$$C(f_n) = O(n^3) + O(n); \quad D(f_n) = O(\log_2 n) + O(\log_2 n). \qquad (12)$$

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. A factor of 2 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 bit, which corresponds to $2n$ pixel positions. The method we propose here will result in one active element for every n pixel.

B. Present Method Compared With Shadow-Casting Logic

Shadow-casting logic is the most difficult of the proposed paradigms with which to compare our proposed system. This difficulty results from the measures by which the paradigm's complexity is measured. The decrease of available light with increasing numbers of inputs does show an exponential variance and the number of sources required is $O(2^n)$. There does not seem to be a good choice for a single unit of measure that can combine these two variances. In the best case for shadow casting, i.e., the implementation of a function with 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, a circuit with $O(n)$ gates may be constructed by spatially variant techniques to compute the function. 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 suitable assigned the value of each variable or that variable's complement, depending on 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 an n-input AND gate may be constructed from $n - 1$ 2-input AND gates; the resulting tree has a depth of $(\log_2 n)$. Thus a function with n-inputs, and with one minterm only 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, which results in a circuit with, 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 with $O(n)$ power requirements. At the other extreme, computing the parity function a shadow-casting implementation requires 2^{n-1} sources or detectors. An implementation of this function in terms of the XOR requires only $O(n)$ XOR gates, each of which can again be replaced by a constant number of NOR gates. In between these extremes, there are functions that require a direct optical implementation of the best theoretical solution and that have $\Omega(2^n/n)$ gate functions and, hence, power dissipation. This will still be an improvement over that achievable with Shadow Casting.

C. Present Method Compared With Programmable Logic

Our method provides many of the same strengths while eliminating several of programmable logic's weaknesses. As with programmable logic, our method uses optical nonlinearities directly as logic gates. The use of general interconnections, however, overcomes the exponential complexity imposed by the limited interconnection capabilities of the crossover network.

When we compare the carry look-ahead adder with a parallel adder implemented with the crossover interconnection only, we can see that, for the 2-bit adder demonstrated, 34 NOR gates were required in the spatially variant implementation, while the latter method requires a minimum of $2 \times 5 \times 2^5 = 320$ AND and OR gates. Even given that the latter method can provide fault tolerance,[25] 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 $n \times 2^n$ logic elements, the number of outputs that can then be computed is indeterminate, and is limited by the ability to find contention-free paths through the OR stage of the network.

D. Present Method Compared With Combinatorial Logic

Comparison with the method of combinatorial logic is difficult. While the spatially invariant version of the combinatorial logic 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 this system to allow a greater portion, perhaps all, of the possible Boolean functions to be computed. We note, however, that the system that we propose can implement all Boolean functions, at a cost of power and speed that is commensurate with the best theoretical implementations. Since our system is spatially variant overall and uses optical elements directly as gates, there is no need to precompute partial terms electronically, and, hence, the speed should be close to the maximum that is attainable optically.

6. Conclusions

We have shown that simple spatially invariant interconnects can reduce the complexity required of an optical implementation. This reduction was achieved because the spatially invariant implementations impose such great restrictions on the capabilities of an optical-computing system. We have seen that several seemingly dissimilar optical-computing paradigms have complexity measures that are quite similar. SSL requires 2^n detector planes and $2n$ shift operations per detection operation. Shadow-casting logic requires n planes of SLM's and $O(2^{n-1})$ light sources, with a resultant decrease in illumination on the detector plane that is 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, of which at least $2n$ are reserved. The spatially invariant subsystem of the combinatorial logic method cannot 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.

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 that could demonstrate rigorously whether there is any set of spatially invariant interconnections that can optimally construct Boolean functions of a higher order. The model of Thompson[26] has had a great impact on very large-scale integrated 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.

Appendix A. Some Results in Boolean Complexity

The limits imposed by Shannon's theorem[1] may seem surprising, but are well established. To clarify the implications of this theorem the following discussion may prove helpful. The normal two-level realization of a minimized sum-of-products Boolean formula constitutes a general design methodology, and, hence, can result in expensive implementations of most functions. The worst-case function in this regard, the parity function, has 2^{n-1} product terms, which require that either the actual value of each variable or that variable's complement be present. Implementing this function directly from its minimized sum-of-products representation requires the construction of 2^{n-1} n-input AND gates and one 2^{n-1}-input OR gate. While each of the n-input AND gates may be constructed from a Binary tree of $n - 1$ 2-input AND gates, realization of the 2^{n-1}-input OR gate requires a binary tree of $2^{n-1} - 1$ OR gates. While the complexity of this realization is in excess of that of the Shannon limit, there exists a realization of this function that requires only $n - 1$ XOR gates and has a depth of $\lceil \log_2 n - 1 \rceil$. This example demonstrates that the Karnaugh map or the Quine–McCluskey reduction techniques can fail to yield a truly minimum realization, and other techniques such as a ring-sum expansion (which is used to derive the XOR implementation) or factoring may achieve reductions that other techniques cannot.

The problem of finding a minimal realization of a function becomes worse when functions with mutiple outputs are considered. Each output is, in general, a function of all inputs and hence may require a large number of gates to implement. In these cases, finding terms that are common to two or more outputs may decrease the number of gates required. All these techniques can fail to find a minimum realization in cases in which a function has an underlying structure that can be exploited. Hence, careful analysis of the nature of the function being implemented is required.

It is interesting to note that increasing the fan-out of gates does not decrease the complexity of the implementation of a function by more than a constant multiple (restricting fan-out to 1 imposes a special case). Additionally, increasing the fan-in of the gate functions (actually changing to a different basis set) increases the number of gates required to implement a function by, at most, a constant multiple. For proofs of these two statements the reader is referred to Ref. 23. Consequently, from a theoretical viewpoint, we need only analyze the cost of implementing a function with any of the 2-input basis sets to determine the order of complexity of an implementation, as long as we allow a fan-out of 2.

This research was supported by National Science Foundation grant no. MIP-8909326. We are indebted to Udi Manber of the Computer Science Department, University of Arizona, for many fruitful discussions.

References

1. C. E. Shannon, "The synthesis of two-terminal switching circuits," Bell Syst. Tech. J. **28**, 59–98 (1949).
2. W. F. McColl, "Planar circuits have short specifications," in *Lecture Notes on Computer Science,* G. Goos and J. Hartmanis, eds., (Springer-Verlag, New York, 1985), Vol. 182, pp. 231–242.
3. K. H. Brenner, A. Huang, and N. Streibl, "Digital optical computing with symbolic substitution," Appl. Opt. **25**, 3054–3060 (1986).
4. B. K. Jenkins, P. Chavel, R. Forchheimer, A. A. Sawchuk, and T. C. Strand, "Architectural implications of a digital optical processor," Appl. Opt. **23**, 3465–3474 (1984).
5. A. Huans, "Parallel algorithms for optical digital computers," in *Proceedings of the Tenth International Optical Computing Conference* (Institute of Electrical and Electronics Engineers, New York, 1983), p. 13.
6. K. H. Brenner, "New implementation of symbolic substitution logic," Appl. Opt. **25**, 3061–3064 (1986).
7. J. Weigelt, "Binary logic by spatial filtering," Opt. Eng. **26**, 28–32 (1987).
8. K. H. Brenner, A. W. Lohmann, and T. M. Merklein, "Symbolic substitution implemented by spatial filtering logic," Opt. Eng. **28**, 390–396, (1989).
9. D. P. Casasent and E. C. Botha, "Multifunctional optical processor based on symbolic substitution," Opt. Eng. **28**, 425–433 (1989).
10. H. Jeon, M. A. G. Abushagur, A. A. Sawchuk, and B. K. Jenkins, "Digital optical processor based on symbolic substitution using holographic matched filtering," Appl. Opt. **29**, 2113–2125 (1990).

11. J. Tanida and Y. Ichioka, "Optical logic array processor using shadowgrams," J. Opt. Soc. Am. **73,** 800–809 (1983).
12. 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 **2,** 1245–1253 (1985).
13. J. Tanida and Y. Ichioka, "OPALS: optical parallel array logic system," Appl. Opt. **25,** 1565–1570 (1986).
14. J. Tanida and Y. Ichioka, "Modular components for an optical array logic system," Appl. Opt. **26,** 3954–3960 (1987).
15. M. J. Murdocca, A. Huang, J. Jahns, and N. Streibl, "Optical design of programmable logic arrays," Appl. Opt. **27,** 1651–1660 (1988).
16. M. J. Murdocca and T. J. Cloonan, "Optical design of a digital switch," Appl. Opt. **28,** 2505–2517 (1989).
17. P. S. Guilfoyle and W. J. Wiley, "Combinatorial logic based digital optical computing architectures," Appl. Opt. **27,** 1661–1673 (1988).
18. A. Louri, "Parallel implementation of optical symbolic substitution logic using shadow-casting and polarization," Appl. Opt. **30,** 540–548 (1991).
19. A. Louri, "Throughput enhancement for optical symbolic substitution computing systems," Appl. Opt. **29,** 2979–2981 (1990).
20. G. Eichmann, A. Kostrzewski, D. H, Kim, and Y. Li, "Optical higher-order symbolic recognition," Appl. Opt. **29,** 2135–2147 (1990).
21. J. Weigelt, "Space-bandwidth product and crosstalk of spatial filtering methods for performing binary logic optically," Opt. Eng. **27,** 883–892, (1988).
22. R. Arrathoon and S. Kozaitis, "Shadow casting for multiple-valued associative logic," Opt. Eng. **25,** 29–37 (1986).
23. J. E. Savage, *The Complexity of Computing* (Wiley, New York, 1976).
24. A. Louri, "Three-dimensional optical architectural and data-parallel algorithms for massively parallel computing," **IEEE Micro II,** 24–27, 65–82 (1991).
25. M. J. Murdocca, "Fault avoidance for optical logic arrays and regular free-space interconnects," in *Digital Optical Computing II,* R. Arrathoon, ed., Proc. Soc. Photo-Opt. Instrum. Eng. **1215,** 116–122) (1990).
26. C. D. Thompson, "A complexity theory for VLSI," Ph.D. dissertation (Carnegie-Mellon University, Pittsburgh, Pa., 1980).