Fast Area-Efficient VLSI Adders

Tackdon Han and David A. Carlson[1]
Department of Electrical and Computer Engineering,
University of Massachusetts, Amherst, MA 01003

Abstract

In this paper, we study area-time tradeoffs in VLSI for prefix computation using graph representations of this problem. Since the problem is intimately related to binary addition, the results we obtain lead to the design of area-time efficient VLSI adders. This is a major goal of our work: to design *very low latency addition circuitry that is also area efficient.* To this end, we present a new graph representation for prefix computation that leads to the design of a fast, area-efficient binary adder. The new graph is a combination of previously known graph representations for prefix computation, and its area is close to known lower bounds on the VLSI area of parallel prefix graphs. Using it, we are able to design VLSI adders having area $A = O(n \log n)$ whose delay time is the lowest possible value, i. e. the fastest possible area-efficient VLSI adder.

1 Introduction

Addition circuitry is an important part of any computer architecture, since it is the major component of an ALU, floating point processor, or a special purpose chip ([Gr85], [FW85], [ST85], etc.). Since the performance of the adder can dominate the performance of the architecture, the design of low latency circuitry has been the subject of a large amount of research. In this paper, we continue these studies with the design of a fast area-efficient VLSI binary adder. We start by deriving a new "hybrid" algorithm for prefix computation (a combination of previously known algorithms for prefix computation which are either the best case in area or time complexity), and investigate the trade-off behavior between area and time of this algorithm. Since graph representations of prefix algorithms map directly into circuitry for binary addition, our hybrid prefix algorithm leads to the design of fast and area efficient VLSI adders. One of the results of this research is that we are able to design the fastest possible circuitry occupying a given amount of area for both prefix computation and binary addition.

In VLSI design, there are three important factors that affect the performance of a design. The first is area complexity, the second is time/delay performance, and the third is regularity of interconnection. Usually, a pattern rich in connections occupies a lot of area, while a pattern with limited connections takes a long time to compute. Because the cost of area is very expensive, every effort should be made to find the optimal area complexity for a given value of time performance.

Area in VLSI is expressed as the total amount of silicon used. Interconnection of wires can consume most of the area of a VLSI circuit. Therefore, in this paper, we take the complexity of interconnection into consideration rather than just a count of "active elements", "gates", or "registers".

In the methodology of finding a better VLSI circuit, we suggest the following steps.

a Find a lower bound on the chosen metric.

b Devise algorithms that are close to the lower bound or analyze known algorithms with respect to the chosen metric.

c By investigating the trade-off behavior between area and time complexity, find the optimal range of area-time complexity.

d Discover corresponding layouts that are optimal in area-time complexity for given values of computation time (including minimum computation time).

Brent and Kung [BK82] found a regular layout for a graph representation of prefix computation and used it to perform binary addition. We apply the above steps on a combination of their algorithm and an algorithm for solving linear recurrences developed by Kogge and Stone [KS73]. We then substitute actual logic for the nodes and wires for the edges of our graphs in order to implement VLSI circuitry for binary addition that is both fast and area-efficient.

This paper is organized as follows. In the next section, we derive upper bounds on area for a graph representation of our new prefix algorithm. In section 3, we design VLSI circuitry for binary addition based on our hybrid prefix algorithm. In section 4, we compare our model and circuitry with others in terms of area and delay time performance. We conclude this paper with suggestions for future research in section 5.

2 Graph Representations of Algorithms for Prefix Computation

A parallel prefix circuit is a combinational circuit that takes n inputs $x_1, x_2, \cdots x_n$ and produces the n outputs $x_1, x_1 \circ x_2, \cdots, x_1 \circ x_2 \circ \cdots \circ x_n$ where $\circ$ represents an arbitrary associative binary operation. Parallel prefix computation has been used for many applications, for instance, in regular layouts of parallel adders [BK82], generating consecutive terms of a linear recurrence [GLPG82], and fast addition of two binary numbers [Of63]. Therefore, it is of interest to find area-time efficient VLSI implementations of prefix computation. We are interested here in the design of the fastest possible prefix circuitry occupying a given amount of VLSI area.

This section is organized as follows. In section 2.1, we review past work on the subject of prefix computation. Section 2.2 describes a new algorithm for prefix computation that forms a basis for the rest of the paper. In sections 2.2 and 2.3, we derive two different VLSI layouts for the graph representation of our new algorithm and discuss their properties. We obtain upper bounds

[1]present address: Supercomputing Research Center 4380 Forbes Blvd. Lanham, MD 20706

on area for a given value of delay time for the layouts, and from these upper bounds, we find the range of delay time for which area is $O(n \log n)$. In section 2.4, we discuss properties of the graph's layout when its nodes are interpreted to implement the circuitry of a binary adder.

2.1 Review of Past Work

Past work on parallel algorithms for prefix computation is as follows. Ladner and Fischer [LF80] gave a general construction for parallel prefix circuits when gates have unbounded fan-out and Fich [Fi83] provided new lower and upper bounds for circuits with gates having bounded fan-out. Brent and Kung [BK82] suggested a regular layout for computing prefixes. Their algorithm is suitable for implementation in VLSI, and when it is combined with pipelining, time $O(\log n)$ and area $O(n)$ can be achieved. In the early 70's, Kogge and Stone [KS73] discovered the technique of recursive doubling and gave an algorithm for solving a large class of recurrence problems on parallel computers such as the Illiac IV. Their algorithm can be applied to compute prefixes in parallel and a graph representation can be developed which leads to a possible VLSI layout. In papers by Carlson and Sugla [CS85] and Sugla [Su85], a lower bound on the VLSI area of a prefix graph was obtained using the minimum bisection width (MBW) of the graph. Thompson [Th79,80] pioneered this approach to deriving VLSI lower bounds. In [CS85] and [Su85], the lower bound on the area of a parallel prefix graph is $A = \Omega(n^2 2^{2(\log_2 n - T)} + n)$ where $\log n \leq T \leq 2 \log n$. They indicated the possibility of designing circuitry with minimum values of both area and time.

The work of Ladner and Fischer, and Fich, while interesting, uses gate count rather than area as a complexity measure. This ignores the wiring area complexity needed to interconnect circuit elements. Brent and Kung [BK82] use area as a complexity measure, however, their techniques do not produce minimum depth parallel prefix circuits. In the following section, we use methods similar to those employed by Brent and Kung, and Carlson and Sugla to find the minimum depth of an area efficient prefix circuit. The result immediately suggests area-efficient VLSI implementations of high-speed binary addition.

2.2 The New Hybrid Prefix Algorithm, a Graph Representation, and a VLSI Layout.

In this subsection, we discuss results on area-time tradeoffs for prefix computation. Our discussions lead to a graph representation of a new algorithm for computing prefixes, which will form the basis of the fast area-efficient VLSI adder designs presented later in this paper. We obtain a straightforward VLSI layout of the graph, and upper bound the area of the layout when it is constructed to have a given value of delay time. From this upper bound, we find the range of delay time for which area is efficient, i.e. $A = O(n \log n)$.

The Brent-Kung (B-K) prefix graph [BK82] can be laid out in area $O(n \log n)$ with $T = 2 \log n - 1$, as shown in Figure 2.1. A generalization of the Kogge-Stone (K-S) linear recurrence algorithm [KS73] computes prefixes and has a graph representation that can be laid out in area $\Omega(n^2)$ with $T = \log n$ (see Figure 2.2). The large area is mainly due to the large number of vertical tracks required to embed wires in the upper stages of the graph. The last stage requires $n/2$ vertical tracks, the stage before $n/4$, etc., which adds to a total of $n - 1$ vertical tracks. Figure 2.2 illustrates this, eventhough some lines do not follow grid tracks.

Comparing these graph representations for prefix computation, we find extremes in area and time performance (K-S: high area, low time; B-K: low area, high time), and we can see that only a constant factor reduction in delay time from $2 \log n - 1$ to $\log n$ results in a significant increase in area from $O(n \log n)$ to $\Omega(n^2)$. It becomes of interest to discover upper and lower bounds on area-time tradeoffs for prefix computation when delay time is restricted to lie in the range between $\log n$ and $2 \log n - 1$.

Carlson and Sugla [CS85] obtained lower bounds on the area of a prefix graph for delay time between $\log n$ and $2 \log n - 1$. Among other things, the lower bounds are matched by the B-K and the K-S graphs at the extremes of the range. Thus, it is likely that a proper combination of the B-K and K-S graphs will be both area and time efficient.

By combining the B-K and K-S graphs as shown in Figure 2.3, we obtain a new hybrid prefix graph that achieves intermediate values of area and time. The graph is easily observed to compute prefixes correctly. We divide it into three separate blocks, the outer of which are multiple B-K graphs and the inner of which is the K-S graph. A straightforward VLSI layout of the graph follows from Figure 2.3, which is drawn with the correct number of vertical tracks between each stage.

If we define k as the extra depth of the hybrid prefix graph and n as the number of inputs, then the area of the embedded K-S graph will be $n^2/2^k$. To see this, consider a hybrid prefix graph of extra depth k composed of an embedded K-S graph on $n/2^k$ inputs surrounded by B-K graphs each with 2^k inputs. The embedded K-S graph has $n/2^k$ inputs and thus requires $n/2^k$ vertical tracks to layout in VLSI. However, since its inputs are spread out with 2^k tracks between each pair, it requires n horizontal tracks. The area of the embedded B-K graphs in this model will be $2kn$ (n horizontal tracks and $2k$ vertical tracks). Therefore the total VLSI area of this layout of the hybrid prefix graph will be $A = 2kn + n^2/2^k$. In the above expression for area, it can be shown that $A = O(n \log n)$ when k is the following range: $\log n - \log\log n \leq k \leq \log n - 1$. Thus, a small additive factor in delay time performance is gained over the B-K graph with no degradation in area consumption.

When the above upper bounds are compared with Carlson and Sugla's [CS85] lower bound of $A = \Omega(n^2/2^{2k})$, it can be seen that the layout does not achieve optimality. This is mainly due to that area wasted in the middle K-S graph, whose inputs and subsequent nodes are spread by 2^k tracks. However, this layout for the entire graph does have some desirable properties, including a regular connection pattern, and all inputs and outputs on the boundaries. As we will see later in this paper, efficient area utilization can be made for small values of n by folding nodes of the K-S graph into unused tracks.

2.3 An Area Optimal Layout for the Hybrid Prefix Graph

In subsection 2.2, we derived a VLSI layout for prefix graphs that achieves intermediate values of area and time performance. The layout has some desirable properties, especially for small to intermediate values of n, and we will use it as a basis for designing fast area-efficient VLSI adders later in this paper. For large values of n however, area is wasted in the middle of the graph that cannot be used for the placement of other components.

In this subsection, we present an alternative layout for the hybrid prefix graph that is more suitable for large values of n. The layout achieves area consumption that comes within a constant factor of the lower bound of Carlson and Sugla[CS85], however it loses the desirable property of having all inputs and outputs on the boundary.

This asymptotically optimal layout is shown in Figure 2.4. It is a result mainly of laying out the embedded K-S graph in minimum area. Inputs to the K-S graph are placed so that consecutive inputs are separated by at most a constant distance of 3 tracks (some separation is needed to route connecting wires between the K-S and the B-K graphs). Below the K-S graph, subsets of B-K graphs are laid out on top of each other. Each subset is formed so that the number of inputs/outputs to the subset matches the number of inputs/outputs of the K-S graph. In this way, the horizontal space spanned by the B-K graphs is the same as that of the K-S graph.

Using the same steps as in subsection 2.2, we now derive an upper bound on the VLSI area of the layout. The embedded K-S graph occupies area $O((n/2^k)^2)$, since its inputs are a constant distance from each other, i.e. $O(n/2^k)$ horizontal tracks are used. Each B-K graph is laid out in area $O(k2^k)$, so that the total area of the B-K graphs in the layout is $O(kn)$. Thus, the total area of the layout is $A = O(kn + (n/2^k)^2)$. Alternatively, the number of horizontal tracks used by the layout is $O(n/2^k)$, and the number of vertical tracks used is $O(n/2^k)$ by the K-S graph plus $O(k2^{2k})$ by the multiple B-K graphs. This yields the same expression for the area of the layout.

From the above expression for area, it can be shown that $A = O(n \log n)$, when k is in the following range: $\frac{1}{2}(\log n - \log\log n) \leq k \leq \log n - 1$. Thus, a further improvement in delay time can be made while maintaining area $O(n \log n)$.

Our upper bound of $A = 2kn + (n/2^k)^2$ matches the lower bound of Carlson and Sugla [CS85] to within a constant factor for almost all values of extra depth k. If an H-tree like layout is used for each B-K graph, optimality is achieved for all values of k. We also find that $T = 3/2(log n) - 1/2(\log\log n - O(1))$ is the lowest value of delay time for which area $O(n \log n)$ can simultaneously be achieved. In Table 2.1, we compare the area and time performance of the layouts presented in this and the previous subsection for different values of area, time, and extra depth. We leave it as an open question whether the lower bound of Carlson and Sugla [CS85] can be met under the more restrictive condition that all inputs/outputs appear at the boundary of the graph's layout.

2.4 Accounting for the Area of Nodes of the Graph

When a prefix graph is used as a basis for designing binary addition circuitry in VLSI, each node of the graph represents a set of logic equations that must be computed/implemented in the underlying technology. Thus, each node can be thought of as a "leaf cell" or "processing element" and will expand from being a point in the grid to occupy a fixed amount of area in the layout. In this subsection, we determine the effect that the area of the nodes has on the area of the entire layout.

When the area of nodes/processing elements is accounted for, the area of addition circuitry derived from a hybrid prefix graph can be expressed as follows:

$$\begin{aligned} \text{Area} &= \text{Area of Processing Elements} \\ &\quad + \text{Area of Interconnection Layers} \\ &= \alpha\beta(\text{Depth Times Width of Layout}) \\ &\quad + \text{Area of Interconnection Layers} \\ &= \alpha\beta n(k + \log n) + (2kn + \{(\frac{n}{2^k})^2 \text{ or } \frac{n^2}{2^k}\}) \end{aligned}$$

Here, α (β) is the ratio of the height (width) of one processing element to one grid line of the interconnection layer. In our designs, both α and β are approximately 4.

When reducing the depth of the hybrid prefix graph, the total area of the processing elements increases (more are required) and interconnection layer area increases (the embedded K-S graph has more inputs), vice versa with increasing the depth of the hybrid prefix graph. When we take the constant factors α and β into consideration, we find that the following facts are worthy of further exposition. For small values of n, the area of processing elements dominates the total area, and for large values of n, the area of interconnection layers is the dominant factor. Furthermore, when $n \leq 64$, the total area with $k = 2$ or 3 is almost identical to the area with $k = 1$ due to the effect of the constant factors α and β. When n is large, i.e. $n > 64$, extra depth $k = \log n - \log\log n$ or $k = 1/2(\log n - \log\log n)$ should be used to obtain area-efficient circuitry.

In addition to these facts, if we are allowed to use more than one metal layer, or a folding method (explained further in the next section), the total area may be even further reduced. When using more than one metal layer, the area can be expressed as follows:

$$\begin{aligned} \text{Area} &= \text{Area of the processing elements} \\ &\quad + (2kn + \frac{1}{m}\{\frac{n^2}{2^k} \text{ or } (\frac{n}{2^k})^2\}) \end{aligned}$$

where m is the number of metal layers. With the folding method, the total area will be

$$\begin{aligned} \text{Area} &= 1/f(\text{Area of the processing elements}) \\ &\quad + (\text{Area of interconnections}) \end{aligned}$$

where f is the number of folds. Therefore with either or both of the above two methods, we can decrease the depth by at least one unit and still achieve the same area consumption. We note that reducing the depth of the hybrid prefix graph to its absolute minimum ($k = 0$, K-S case) results in the design of area inefficient VLSI circuitry, because the folding method cannot be applied and it is the worst case for the number of the transistor gates (see Table 2.2 for a comparison of upper bounds on the number of gates).

To summarize, for small values of n ($n \leq 64$), we have found that the case when extra depth k is equal to 1 leads to VLSI circuitry for binary addition that is both fast and area efficient when compared to other VLSI designs of carry look-ahead adders. Speed is obtained by using a small value of k, and area-efficiency is obtained through the use of a folding method. The actual design is explained in more detail in the following sections.

3 Detailed VLSI Design of a Fast Area-Efficient Adder

As was mentioned in the previous subsection, our new hybrid prefix algorithm can be applied to the problem of carry look-ahead addition. By using this new method, we are able to overcome the main difficulties of implementing a standard carry look-ahead adder in VLSI: interconnection irregularity and fan-in, fan-out limitations. Our method improves the performance of a standard carry look-ahead adder in following ways.

i Simple leaf cells are used, each having one gate level delay.

ii The overall design is area-efficient, since it is based on the new prefix algorithm and employs a folding technique.

iii The circuitry is fast, since it is based on a near minimum depth prefix algorithm.

iv Fan-in, fan-out to leaf cells is bounded by 2 logical wires (4 physical wires).

v Interconnection wires have reasonable lengths (linear in the number of bits being added).

This section is organized as follows. First we introduce the general formulation of carry look-ahead addition. Then, we explain our method of implementing a carry look-ahead adder based on the new hybrid prefix algorithm. We follow this with a method for further reducing the area of our layouts, and a discussion of how our circuitry compares with other designs.

3.1 The Carry Look-Ahead Adder.

Let $a_{n-1}, \cdots, a_0$ and $b_{n-1}, \cdots, b_0$ be n-bit binary numbers with sum $s_n, \cdots, s_0$. The following equations can be used to compute the s_i's:

$$\begin{aligned} c_{-1} &= 0 \\ c_i &= (a_i \cdot b_i) + (a_i + b_i) \cdot c_{i-1} \\ s_i &= (\neg c_i) \cdot (a_i + b_i + c_{i-1}) + (a_i \cdot b_i \cdot c_{i-1}) \\ s_n &= c_{n-1} \\ &\text{where } i = 0, 1 \cdots, n-1 \end{aligned}$$

Note the alternative equation for the ith sum bit, which is usually expressed as $s_i = a_i \oplus b_i \oplus c_{i-1}$.

It is well known that computing the c_i's quickly is the key to high-speed addition, and that they can be determined using the following scheme.

$$c_i = g_i + (p_i \cdot c_{i-1}) \quad (1)$$

$$\text{where } g_i = a_i \cdot b_i \text{ and } p_i = a_i + b_i$$

Here, we do not use exclusive-ors in the equations [BK82], so that it is not necessary to invert the a's and b's to produce the p's and g's.

The recurrence relation in equation (1) can be applied repeatedly to obtain the following set of carry equations in terms of g_i, p_i and c_{-1}.

$$c_i = g_i + (\sum_{j=0}^{i-1}(\prod_{k=j+1}^{i} p_k) \cdot g_j) + ((\prod_{k=0}^{i} p_k) \cdot c_{-1})$$

Also, when the operator $\circ$ is defined on ordered pairs (p, g) by $(p_i, g_i) \circ (p_j, g_j) = (p_i \cdot p_j, p_j \cdot g_i + g_j)$. It is easily verified that $(p_1, g_1) \circ \cdots \circ (p_i, g_i) = (p_1 \cdots p_i, c_i)$. Thus, computing the carries can be performed using any valid algorithm for prefix computation where $\circ$ is implemented as given above.

3.2 VLSI Implementation

The hybrid prefix algorithm is straightforward to convert into circuitry for binary addition using techniques similar to those employed by Brent and Kung [BK82]. We use the same terminology as [BK82]. Figure 3.1 diagrams a carry look-ahead adder derived from the hybrid prefix algorithm. With this graph, we can reduce the depth of the prefix algorithm from 7 (B-K case) to 5 (our way) with n=16. We use only the standard layers of interconnection available in NMOS VLSI (metal, silicides, polysilicon). If the designer is allowed to use more than 2 metal interconnection layers, then a further reduction in depth can be made while maintaining the given area. We follow $4\mu m$ NMOS topological design rules.

There are 4 different types of leaf cells in our circuitry: pggen, black, white, and sum. All are illustrated in Figure 3.1b. As shown in [NI85], we can employ the following strategy to eliminate the use of inverter gates in each leaf cell.

```
If (level is even number)
    then take positive input; produce negative output
    else take negative input; produce positive output
```

This is explained in Figure 3.1a. We now explain the internal circuitry of each leaf cell. We use paired subscripts i, k in our descriptions, where i denotes the input (output) number and k the level number.

i) Pggen cell
This cell produces the initial p and g signals (carry propagation and generation signals). Comparing our cell with an exclusive-or scheme, the size can be reduced in half and delay time of only one gate level can be achieved.

$$\begin{aligned} \neg p_{i,1} &= \neg(a_i + b_i) \\ \neg g_{i,1} &= \neg(a_i \cdot b_i) \end{aligned}$$

ii) Black cell
A black cell implements the $\circ$ operator on p and g signals. We use two different types of black cells: bp.ca (positive input, negative output) and bn.ca (negative input, positive output). This is explained in Figure 3.1b. Each cell is used alternatively in even and odd levels (see Figure 3.1a).

bn.ca (negative input)

$$\begin{aligned} g_{j,2k} &= \neg(((\neg p_{j,2k-1}) + (\neg g_{i,2k-1})) \cdot (\neg g_{j,2k-1})) \\ p_{j,2k} &= \neg((\neg p_{i,2k-1}) + (\neg p_{j,2k-1})) \end{aligned}$$

bp.ca (positive input)

$$\begin{aligned} \neg g_{j,2k+1} &= \neg(p_{j,2k} \cdot g_{i,2k} + g_{j,2k}) \\ \neg p_{j,2k+1} &= \neg(p_{i,2k} \cdot p_{j,2k}) \end{aligned}$$

iii) White cell
This cell is a simple inverter; notation is as follows.

$$\begin{aligned} p_{i,k} &= \neg p_{i,k-1} \\ g_{k,l} &= \neg g_{i,k-1} \end{aligned}$$

iv) Sum cell
The equation for the ith sum bit in terms of the p, g variables and negative logic is:

$$s_i = \neg((c_i + (\neg p_{i,1}) \cdot (\neg c_{i-1})) \cdot ((\neg g_{i,1}) + (\neg c_{i-1})))$$

Using this equation allows us to avoid the use of exclusive ors, which would require the input variables and their complements $(a_i, \neg a_i, b_i, \neg b_i)$ to be provided to the sum bit circuitry. Taking into account that carries produced by the carry generation circuitry alternate between being positive and negative, we design two types of sum cells. One takes its inputs "naturally" and the other must invert its two carry inputs.

$$\begin{aligned} sumcell_1 &= F(c_i, \neg c_{i-1}, \neg p_{i,1}, \neg g_{i,1}) \\ sumcell_2 &= F(\neg c_i, c_{i-1}, \neg p_{i,1}, \neg g_{i,1}) \end{aligned}$$

We note the following advantages of using leaf cells designed as given above:

i Black cells are simple.

ii Pggen cells are simple.

iii Sum cells are half of a full adder cell.

iv All cells combine speed (minimum number of gate delays) with area efficient layouts.

In Figure 3.2, we show the leaf cell layouts. The leaf cells for the sum circuitry contain super buffers in order to more easily drive a large capacitance output pad. Since all leaf cells have good layouts, the overall design is competitive in area with a standard ripple carry adder. Area increases in our design only because of the extra processing elements and interconnection area required by basing the design on a hybrid prefix graph. To reduce the charge time of the capacitance being driven by the leaf cells, we used minimum sized pull up structures ($2\lambda \times 2\lambda$).

3.3 A Densely Packed Layout Using a Folding Method.

Our carry look-ahead adder based on the hybrid prefix algorithm can be packed densely by using a folding method. With this method, we can reduce in half the area of the overall design for reasonable sized adders ($n \leq 64$). Equivalently we could decrease the depth by one unit and still achieve the same area consumption by employing our folding method. For large sized adders, the folding method is not as effective in reducing the depth, because the interconnection area becomes dominant.

Our folding scheme is shown in Figure 3.3. It works by placing two levels of the hybrid prefix graph into one level of the layout, since space is available to embed leaf cells. It should be obvious from the figure that very efficient area consumption is realized, especially for circuitry designed from a hybrid prefix graph with extra depth $k = 1$.

In Figure 3.4, we introduce two ways of arranging the direction of input and output to our carry look-ahead adder circuitry. In Figure 3.4a, the inputs and outputs go through in the same direction (inputs and outputs both at the bottom). In Figure 3.4b, the outputs come out the top, the opposite side of the inputs, which come in the bottom. This gives the designer some flexibility in attaching the circuitry to different styles of bus structures, or different arrangements of I/O pads. Other designs of carry look-ahead adders in VLSI [NI85] [NIR86] have ignored the placement of inputs and outputs, instead assuming that external inputs and outputs are available wherever they are required. Such assumptions are impractical in a typical ALU where I/O and bussing conventions are generally adapted in advance of the design of the circuitry. We conclude this section by giving a diagram of the entire chip including I/O pads (Figure 3.5).

4 Comparison with Other VLSI Adder Designs

In this section, we compare our VLSI circuitry for carry look-ahead addition with other designs. It is not accurate to compare each design by counting only the number of gates because of the differences in fan-in and fan-out, types of gates, and wiring capacitance effects. Accounting for these differences using the method of [OB85], we compare our circuitry with the carry skip adder of [OB85] and the commercial carry look-ahead adder presented in [Cav84] for which fan-in and fan-out are bounded by at least 4 logical wires. Then, with the same fan-in and fan-out restriction, we compare our circuitry with [BK82] [NI85] [NIR86].

First, some general comparisons are in order. Although the time performance of our circuitry is proportional to $O(\log n)$, it still compares favorably to a standard carry look-ahead adder implemented in VLSI. This is because of the large fan-out of a carry look-ahead adder, and the fact that fan-out proportional to n requires delay time $O(\log n)$ to drive as a capacitive load under realistic models of a VLSI circuit.

The area of our circuitry is very efficient when compared with other designs, and even is competitive with the area of a regular ripple-carry adder for small values of n. When $n \leq 64$, the area values of our circuitry and of a ripple carry adder are compared in Table 4.1. The area of our circuitry is only a factor of 2 or 3 higher than that of the ripple carry adder. For wire capacitance effects, the speed of the circuitry is dependent on the length of the wires (their capacitance) and the resistance of long wires. In Table 4.2, we compare for each design the total interconnection wire length from inputs to outputs. From this table, we can conclude that it takes almost same amount of time for each design to charge and pass signals along the interconnection wires. Therefore we can ignore the delay time due to wire capacitance in comparing these designs (but we do have to consider this wire capacitance when measuring delay time in an actual implementation).

We now compare our circuitry with the carry skip adder of [OB85], and the carry look-ahead adder of [Cav84] using the method of [OB85]. They associate a time unit t with one gate propagation delay time (fan-in and fan-out bounded by at least 4). When we compare our leaf cells with their single gate, ours is quite simple (half the size; fan-in and fan-out bounded by 2). Therefore, the propagation delay time of any of our leaf cells is at most $\frac{1}{2}t$. As discussed in section 2.3, in our designs we have fixed the value of extra depth to $k = 1$ when $n \leq 64$. The delay time comparison of our designs with [OB85], [Cav84] is shown in Table 4.3.

Next we compare our circuitry with the designs of [BK82] [NI85] and [NIR86] using the method outlined in [NI85]. Here, we simply count the worst-case number of gates along any path from inputs to outputs to measure the delay time. The fan-in and fan-out restrictions are almost the same in all these designs and thus are ignored. As we discussed previously, extra depth $k = 1$ is used in our designs with $n \leq 64$. For large values of n ($n > 64$), we use the value $k = \log n - \log\log n$. The delay time comparison between our design and [BK82], [NI85], [NIR86] is shown in Table 4.4. Using our hybrid prefix algorithm, delay time is the lowest possible value with optimal area $O(n \log n)$. Furthermore, by adding various strategies (folding, simple leaf cells and multiple metal layers), our designs are better than those of [BK82], [NI85], [NIR86] with respect to delay time, and almost as area-efficient as a ripple carry adder.

5 Conclusion

We have presented a new prefix algorithm and have applied it to obtain a fast area-efficient VLSI implementation of a binary adder. With the other techniques presented here (folding, multiple metal layers and the design of simple leaf cells), the area of this hybrid adder becomes competitive with the area of a ripple carry adder. Also, the time performance of this adder is better than any other reported in the open literature.

The design strategy used here can be applied to various technologies including ECL, TTL, CMOS, NMOS, etc. Since speed and power dissipation depend on the technology being used, and the lamda scale factor becomes smaller in a new technology, it may be desirable to extend our fan-out 2 algorithm to fan-out f ($f \geq 2$) due to the reduced fan-out capacitances being driven [OA83]. Our current investigations are concentrating on this topic.

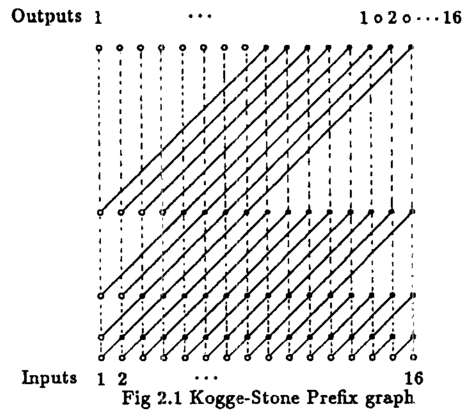


Fig 2.1 Kogge-Stone Prefix graph

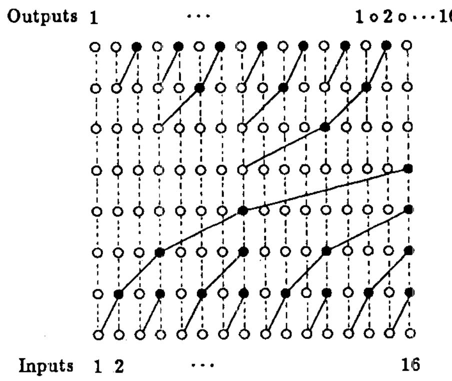


Fig 2.2 Brent-Kung Prefix Graph

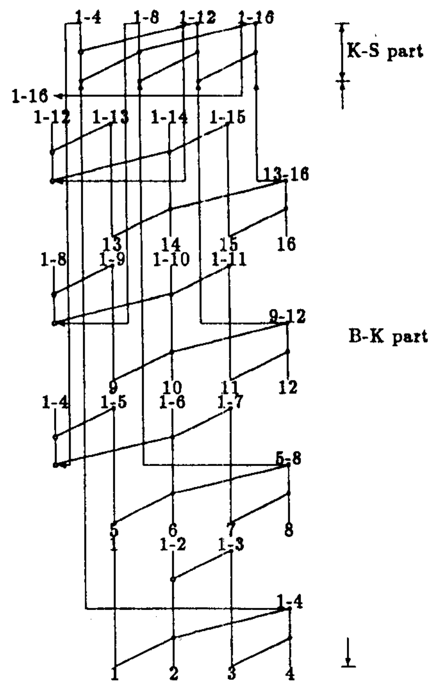
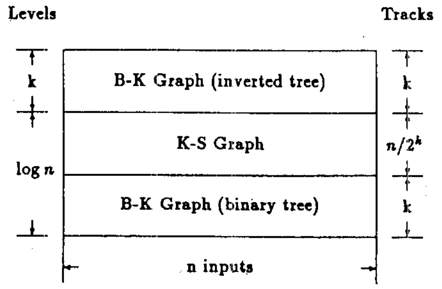
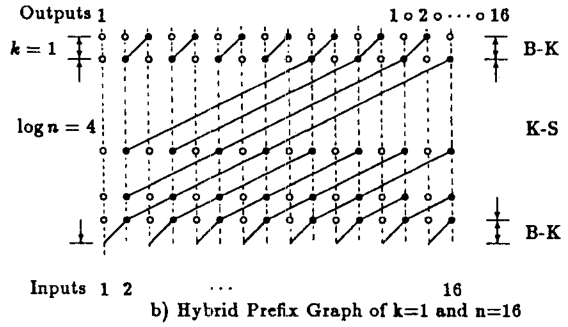


Fig 2.4 Area Optimal Layout (K=2, n=16)

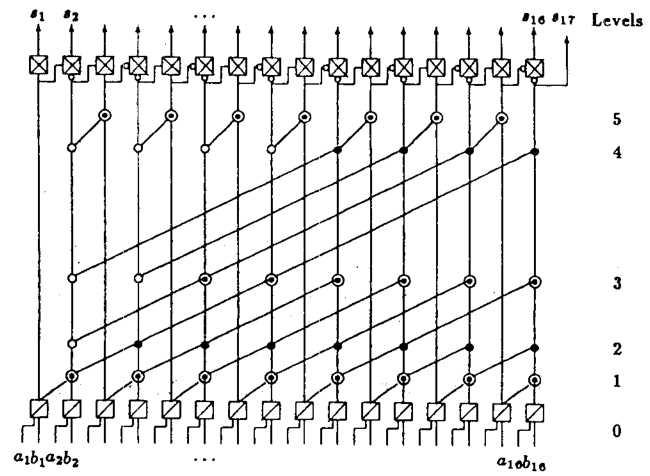


a) Block Diagram of HPC

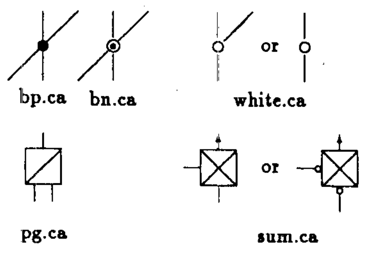


b) Hybrid Prefix Graph of k=1 and n=16

Fig 2.3 Hybrid Prefix Graph and Block Diagram


s_1 s_2 ··· s_{16} s_{17}
Levels
5
4
3
2
1
0
$a_1b_1a_2b_2$ ··· $a_{16}b_{16}$

a) A 16 bit CLA by using HPC



b) notations of leaf cell

Fig 3.1 Hybrid Prefix Adder

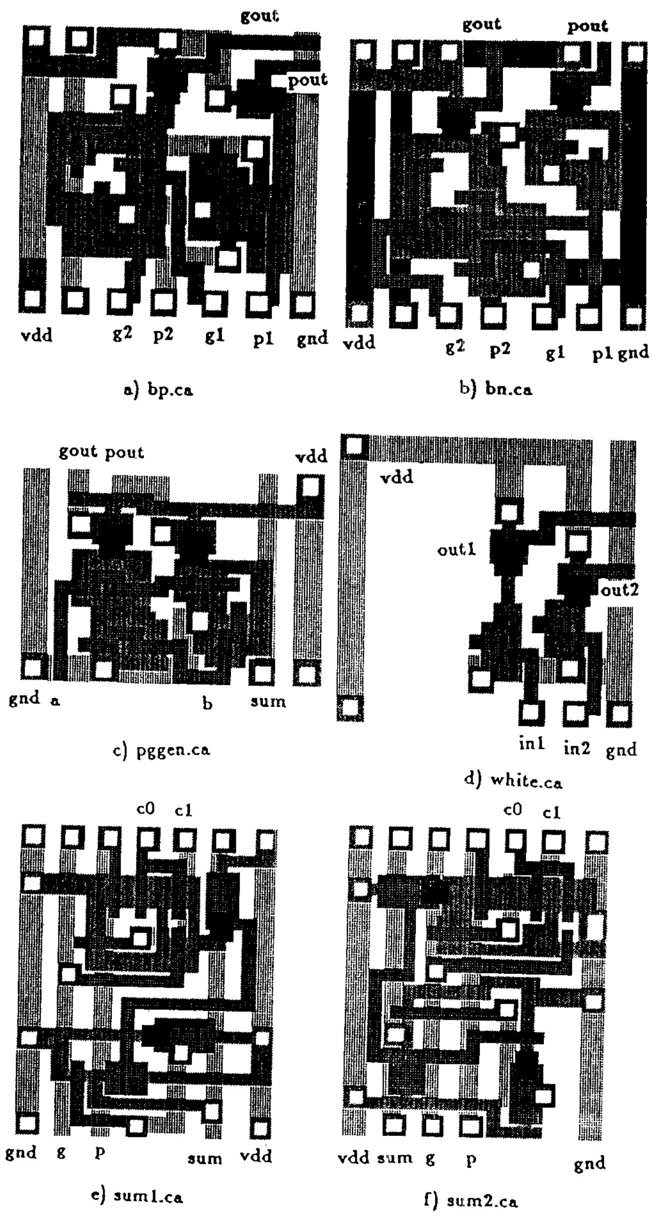


Fig 3.2 Layout of each leaf cell.

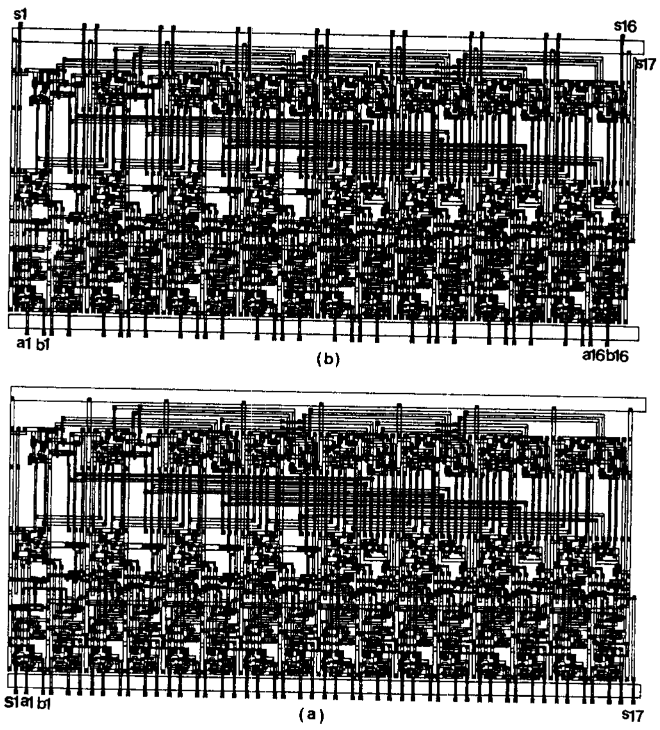


Fig 3.4 Two different styles of HPA (16 bit)
(Different arrangements of I/O pads)

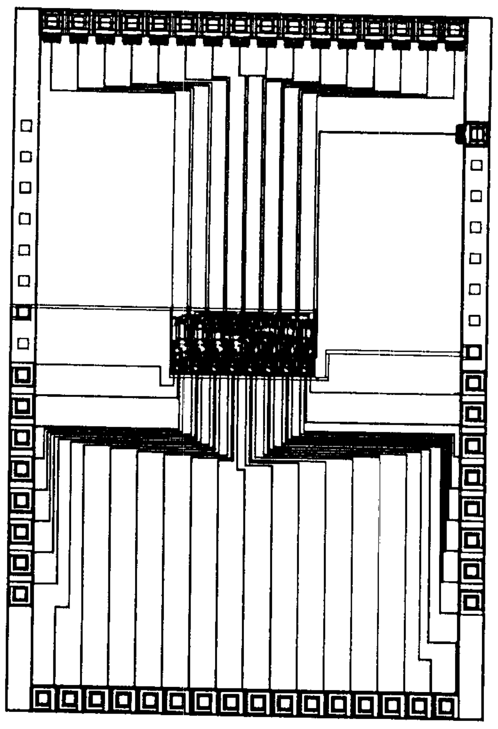


Fig 3.5 Layout of a 16-bit Hybrid Prefix Adder

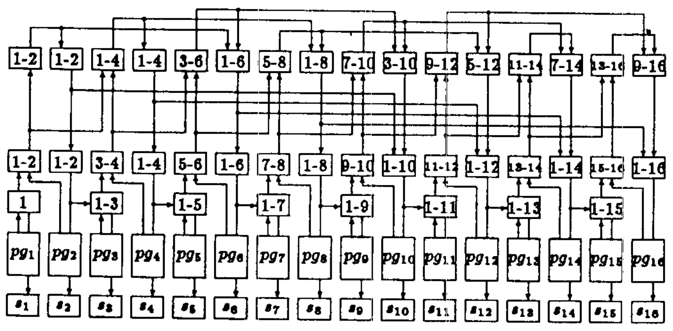


Fig 3.3 Densely Packed Layout using folding method (n=16)

	k (extra depth)	T (time)	Area
K-S case	$k = 0$	$\log n$	$\Omega(n^2)$
HPC (sec 2.2)	$\frac{1}{2}(\log n - \log\log n) \le k \le \log n - 1$	$\frac{3}{2}\log n - \frac{1}{2}\log\log n$	$O(n\log n)$
HPC (sec 2.3)	$\log n - \log\log n \le k \le \log n$	$2\log n - \log\log n$	$O(n\log n)$
B-K case	$k = \log n - 1$	$2\log n - 1$	$O(n\log n)$

Table 2.1 Comparison of Area-Time Complexity for each case of graph.

	number of gates	delay time	k
B-K	$2.5n$	$2\log n - 1$	$\log n - 1$
K-S	$n\log n$	$\log n$	0
Fich	$2.5n + 3\sqrt{n}\log n\log n$	$\frac{3}{2}\log n - \frac{1}{2}\log\log n$	$\frac{1}{2}(\log n - \log\log n)$
HPC	$2.5n + \sqrt{n}\log n\log n$	$\frac{3}{2}\log n - \frac{1}{2}\log\log n$	$\frac{1}{2}(\log n - \log\log n)$

Table 2.2 Comparison of Upper bounds by counting the number of gates (fan-out bounded by 2)

	Area of Ripple Carry Adder	Area of Hybrid Prefix Adder
$n = 16$	$200 \times 700\lambda^2$	$350 \times 700\lambda^2$
$n = 32$	$200 \times 1400\lambda^2$	$500 \times 1400\lambda^2$
$n = 64$	$200 \times 2100\lambda^2$	$650 \times 2100\lambda^2$

Table 4.1 Area of Hybrid Prefix Adder and RCA

	Total interconnection wire length
RCA	0
K-S	n
B-K	n
HPA	n
CSA	n
N-I	$\sqrt{n}$

Table 4.2 Total wire length from input to output for each case

n	CLA	CSA	HPA
16	6t	5t	3.5t
32	8t	7t	4t
48	10t	9t	
64	10t	10t	4.5t

where CLA : Carry Look-Ahead Adder
(Fan-in, Fan-out bounded by 4)
CSA : Carry Skip Adder
(Fan-in, Fan-out bounded by 4)
HPA : Hybrid Prefix Adder
(Fan-in, Fan-out bounded by 2)

Table 4.3 Comparison in T using the method of [OB 85]

n	RCA	B-K	N-I	HPA
8	18	15		7
9	20		8	
16	34	19		8
27	56		12	
32	66			9
64	130	27		10
108	218		16	
256	514	35		17
432	866		20	
512		39		19

Table 4.4 Comparison in T using the method of [NI 85]

References

[BK 82] R.P. Brent and H.T. Kung, "A Regular Layout for Parallel Adders," *IEEE Transactions on Computers*, Vol. C-31, No. 3, March 1982.

[Cav 84] J.J.F. Cavanagh, *Digital Computer Arithmetic Design and Implementation*, McGraw-Hill, New York, NY, 1984.

[CS 85] D.A. Carlson and B. Sugla, "On the Area Requirements of Very Fast VLSI Adders," Technical Report, Dept. of Electrical and Computer Engineering, Univ. of Massachusetts, Amherst, MA, 1985.

[Fi 83] F. Fich, "New Bounds for Parallel Prefix Circuits," *Proceedings 15th ACM Symposium on Theory of Computing*, pp. 100-109, April 1983.

[FW 85] J. Fandrianto and B.Y. Woo, "VLSI Floating Point Processors," *Proceedings 7th Symposium on Computer Arithmetic*, pp. 93-100, June 1985.

[GLPG 82] A.G. Greenberg, R.E. Ladner, M.S. Paterson and Z. Galil, "Efficient Parallel Algorithms for Linear Recurrence Computation," *Information Processing Letters*, Vol. 15, No. 1, pp. 31-35, August 1982.

[Gr 85] T. Gross, "Floating-Point Arithmetic on a Reduced Instruction-Set Processor, "*Proceedings 7th Symposium on Computer Arithmetic*, pp. 86-92, June 1985.

[KS 73] P.M. Kogge and H.S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, Vol. C-22, No. 8, pp. 783-791, August 1973.

[LF 80] R.E. Ladner and M.J. Fischer, "Parallel Prefix Computation," *Journal of the ACM*, Vol. 27, No. 4, pp. 831-838, October 1980.

[NI 85] T.F. Ngai and M.J. Irwin, "Regular Area-Time Efficient Carry-Lookahead Adders," *Proceedings 7th Symposium on Computer Arithmetic*, pp. 9-15, July 1985.

[NIR 86] T.F. Ngai, M.J. Irwin and S. Rawat, "Regular Area-Time Efficient Carry-Lookahead Adders," *Journal of Parallel and Distributed Computing*, pp. 92-105, 1986.

[OA 83] S. Ong and D.E. Atkins, "A Comparison of ALU Structures For VLSI Technology," *Proceedings 6th Symposium on Computer Arithmetic*, pp. 10-16, 1983.

[OB 85] V.G. Oklobdsija and E.R. Barnes, "Some Optimal Schemes for ALU Implementation in VLSI Technology," *Proceedings 7th Symposium on Computer Arithmetic*, pp. 2-8, June 1985.

[Of 83] Y. Ofman, "On the Algorithmetic Complexity of Discrete Functions," *Cybernetics and Control Theory, Soviet Physics Doklady*, Vol. 7, No.7, pp. 589-591, 1963.

[ST 85] S.P. Smith and H.C. Torng, "Design of a Fast Inner Product Processor," *Proceedings 7th Symposium on Computer Arithmetic*, pp. 38-43, June 1985.

[SU 85] B. Sugla, "Parallel Computation with Limited Resources," Ph.D. Dissertation, Dept. of Electrical and Computer Engineering, Univ. of Massachusetts, Amherst, MA, May 1985.

[Th 79] C.D. Thompson, "Area-Time Tradeoffs in VLSI," *Proceedings 11th ACM Symposium on Theory of Computing*, pp. 81-88, April 1979.

[Th 80] C.D. Thompson, "A Complexity Theory for VLSI," Ph.D. Dissertation, Carnegie-Mellon University, Dept. of Computer Science, 1980.