Non-Heuristic Optimization and Synthesis of Parallel-Prefix Adders

Reto Zimmermann

Integrated Systems Laboratory
Swiss Federal Institute of Technology (ETH)
CH-8092 Zürich, Switzerland
E-mail: zimmermann@iis.ee.ethz.ch

Abstract

The class of parallel-prefix adders comprises the most area-delay efficient adder architectures – such as the ripple-carry, the carry-increment, and the carry-lookahead adders – for the entire range of possible area-delay trade-offs. The generic description of these adders as prefix structures allows their simple and consistent area optimization and synthesis under given timing constraints, including non-uniform input and output signal arrival times. This paper presents an efficient non-heuristic algorithm for the generation of size-optimal parallel-prefix structures under arbitrary depth constraints.

Keywords

Parallel-prefix adders, non-heuristic synthesis algorithm, circuit timing and area optimization, computer arithmetic, cell-based VLSI.

1 Introduction

Cell-based design techniques, such as standard-cells and FPGAs, together with versatile hardware synthesis are prerequisites for a high productivity in ASIC design. For the implementation of arithmetic components, the designer must rely on a comprehensive library or on efficient synthesis of optimized adder circuits. Many different adder architectures for speeding up binary addition have been studied and proposed over the last decades. For cell-based design techniques they can be well characterized with respect to circuit area and speed as well as suitability for logic optimization and synthesis.

The *ripple-carry adder* is the obvious solution for lowest area and lowest speed requirements. The *carry-skip adder* realizes a considerable speed-up at only small area increase. However, it is not suited for synthesis because of the complex block size computation and the inherent logic redundancy, which disallows automatic circuit optimization. Redundancy removal is possible only at a considerable area increase [1]. The massive speed-up of the *carry-select adder* is accompanied by a large amount of hardware overhead due to duplicate sum computation and selection circuitry. However, this structure can be reduced to a single sum calculation with a subsequent incrementation. In this relatively new *carry-increment adder* scheme [2, 3] summation and incrementation circuitries are merged. The carry-increment adder has the same delay and a similar structure as the corresponding carry-select adder, but occupies a significantly smaller area. Multiple levels of carry-increment logic can be applied for reducing delay further, thereby leading to multilevel carry-increment adders [3]. The *conditional-sum adder*, being a hierarchical carry-select adder, also suffers from the very area-intensive multilevel selection circuitry. The same conversion from a selection to an incrementation structure can be applied here to improve area efficiency. The resulting *carry-lookahead adders* come with different carry propagation schemes offering a variety of high-performance adder circuits.

Investigations on cell-based adder architectures by way of placed-and-routed standard-cell implementations showed that the set containing the ripple-carry, carry-increment, and carry-lookahead adders offers the most efficient adder architectures for the entire range of possible area-delay trade-offs [3]. These architectures all base on the same *basic adder structure* which computes a generate, a propagate, a carry, and a sum signal for each bit position. They only differ in the way of computing the carries, which depend on the generate and propagate signals of lower bit positions.

The synthesis of adder circuits with distinct performance characteristics is standard in today's ASIC design packages. However, only limited flexibility is usually provided to the user for customization to a particular situation. The most common circuit constraints arise from dedicated timing requirements, which may include arbitrary input and output signal arrival profiles e.g. as found in the final adder of multipliers [4]. The task of meeting all timing constraints while minimizing circuit size is usually left to the logic optimization step which starts from an adder circuit designed for uniform signal arrival times. Taking advantage of individual signal arrival times is therefore very limited and computation intensive. On the other hand, taking timing specifications into account earlier during adder synthesis may result in more efficient circuits as well as considerably smaller logic optimization efforts. The task of adder synthesis is therefore to generate an adder circuit with minimal hardware which meets all timing constraints. This, however, asks for an adder architecture which has a simple, regular structure and results in well-performing circuits, and which provides a wide range of area-delay trade-offs as well as enough flexibility for accomodating non-uniform signal arrival profiles.

All these requirements are met by the class of parallel-prefix adders introduced in Section 2. Section 3 describes their efficient optimization and synthesis at the structural level while relying on state-of-the-art software tools for gate-level timing optimization and technology mapping. Experimental results are given and discussed in Section 4.

2 Parallel-Prefix Adders

2.1 Prefix Addition

In a prefix problem, n inputs $x_{n-1}, x_{n-2}, \ldots, x_0$ and an arbitrary associative operator $\bullet$ are used to compute n outputs $y_i = x_i \bullet x_{i-1} \bullet \cdots \bullet x_0$, $i = 0, \ldots, n-1$. Thus, each output y_i is dependent on all inputs x_j of same or lower magnitude ($j \leq i$). Carry propagation in binary addition is a prefix problem. *Prefix addition* can be expressed as follows:

$$\left.\begin{array}{rcl} g_i & = & \begin{cases} a_0b_0 + a_0c_0 + b_0c_0 & \text{if } i = 0, \\ a_ib_i & \text{otherwise} \end{cases} \\ p_i & = & a_i \oplus b_i \\ (G^0_{i:i}, P^0_{i:i}) & = & (g_i, p_i) \end{array}\right\} \textit{preprocessing}$$

$$\left.\begin{array}{rcl} (G^l_{i:k}, P^l_{i:k}) & = & (G^{l-1}_{i:j}, P^{l-1}_{i:j}) \bullet (G^{l-1}_{j:k}, P^{l-1}_{j:k}) \\ & = & (G^{l-1}_{i:j} + P^{l-1}_{i:j} G^{l-1}_{j:k}, P^{l-1}_{i:j} P^{l-1}_{j:k}) \end{array}\right\} \begin{array}{l}\textit{prefix computation} \\ (m \text{ levels})\end{array}$$

$$\left.\begin{array}{rcl} c_{i+1} & = & G^m_{i:0} \\ s_i & = & p_i \oplus c_i \end{array}\right\} \textit{postprocessing}$$

for $i = 0, \ldots, n-1$ and $l = 1, \ldots, m$, where a_i and b_i are the operand input, g_i and p_i the generate and propagate, c_i the carry, and s_i the sum output signals at bit position i. c_0 and c_n correspond to the carry-in c_{in} and carry-out c_{out}, respectively. $G^l_{i:k}$ and $P^l_{i:k}$ denote the group generate and propagate signals for the group of bits $i, \ldots, k$ at level l. The $\bullet$ operator is repeatedly applied according to a given prefix structure of m levels (i.e. depth m) in order to compute the group generate signal $G^m_{i:0}$ for each bit position i.

Prefix structures and adders can nicely be visualized using directed acyclic graphs (DAGs) with the edges standing for signals or signal pairs and the nodes representing the four logic operators depicted in Figure 1.

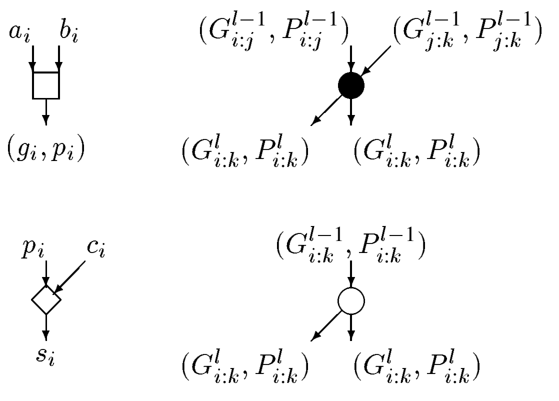


Figure 1 Prefix adder logic operators.

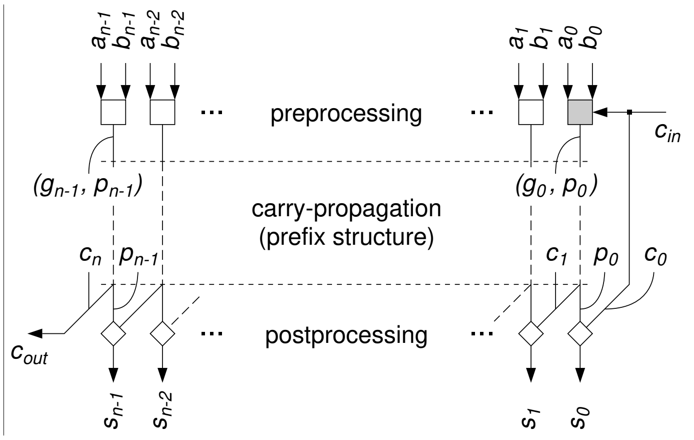


Figure 2 Prefix adder structure.

Figure 2 shows the general prefix adder structure. The square (□) and diamond (◇) nodes form the preprocessing and postprocessing stages. The black nodes (•) evaluate the prefix operator • and the white nodes (○) pass the signals unchanged to the next level in the prefix carry-propagation stage, which is visualized by prefix graphs in the sequel. These prefix graphs consist of n columns (i.e. bit positions) and m rows (i.e. prefix levels) of black or white nodes, where each row corresponds to one black node delay. The top and bottom margins of a column reflect the input and output signal arrival times for that particular bit. Because white nodes do not contain any logic, they are neglegted in graph size measures.

The general prefix adder structure of Figure 2 corresponds to the basic adder structure mentioned in Section 1. Thus, the efficient ripple-carry, carry-increment, and carry-lookahead adders all belong to the class of prefix adders.

2.2 Parallel-Prefix Structures

Due to the associativity of the prefix operator •, a sequence of operators can be evaluated in any order. Serial evaluation from the LSB to the MSB has the advantage that all intermediate prefix outputs are generated as well. The resulting *serial-prefix* structure does with the minimal number of $n-1$ black nodes but has maximal evaluation depth of $n-1$ (Figure 3). It corresponds to ripple-carry addition. Parallel evaluation of operators by arranging them in tree structures allows a reduction of the evaluation depth down to $\log n$. In the resulting *parallel-prefix* structures, however, additional black nodes are required for implementing evaluation trees for all prefix outputs. Therefore, structure depth (i.e. number of black nodes on critical path, circuit delay) – ranging from $n-1$ down to $\log n$ depending on the degree of parallelism – can be traded off versus structure size (i.e. total number of black nodes, circuit area). Furthermore, the various parallel-prefix structures also differ in terms of wiring complexity and fan-outs.

Adders based on these parallel-prefix structures are called *parallel-prefix adders* and are basically carry-lookahead adders with different lookahead schemes. The fastest but largest adder uses the parallel-prefix structure introduced by Sklansky [5] (Figure 4(c)). The prefix structure proposed by Brent and Kung [6] offers a good trade-off having almost twice the depth but much fewer black nodes (Figure 4(d)). The linear size-depth trade-off described by Snir [7] allows for mixed serial/parallel-prefix structures of any depth between $2\log n - 3$ and $n-1$, thus filling the gap between the serial-prefix and the Brent-Kung parallel-prefix structure. The carry-increment parallel-prefix structures exploit parallelism by hierarchical levels of serial evaluation chains rather than tree structures (Figures 4(a) and (b)). This results in prefix structures with a fixed maximum number of black nodes per bit position ($\#^{max}_{\bullet/b}$) as a function of the number of applied increment levels (i.e. $\#^{max}_{\bullet/b} - 1$ levels). They are also called *bounded*-$\#^{max}_{\bullet/b}$ prefix structures in the sequel. Note that, depending on the number of increment levels, this carry-increment prefix structure lies somewhere between the serial-prefix ($\#^{max}_{\bullet/b} = 1$) and the Sklansky parallel-prefix structure ($\#^{max}_{\bullet/b} = \log n$).

Table 1 Characteristics of common prefix structures.

prefix structure	$D_\bullet$	$\#_\bullet$	$\#_{\bullet/b}^{max}$	$\#_{tracks}$	$FO_\bullet^{max}$	synthesis (this work)	perform. A	perform. T
serial	$n-1$	$n-1$	1	1	2	yes	$++$	$--$
1-level carry-incr. par.	$\sim \sqrt{2n}$	$\sim 2n - \sqrt{2n} - 2$	2	2	$\sim \sqrt{2n}$	yes	$+$	$-$
2-level carry-incr. par.	$\sim \sqrt[3]{6n}$	$\sim 3n - \ldots$	3	3	$\sim (6n)^{2/3}$	yes	$-$	$+$
Sklansky parallel	$\log n$	$\frac{1}{2}n \log n$	$\log n$	$\log n$	$\frac{1}{2}n+1$	yes	$-$	$++$
Brent-Kung parallel	$2\log n - 2$	$2n - \log n - 2$	$\log n$	$2\log n - 1$	$\log n + 1$	yes	$+$	$-$
Kogge-Stone parallel	$\log n$	$n \log n - n + 1$	$\log n$	$n-1$	2	no	$--$	$++$
Han-Carlson parallel	$\log n + 1$	$\frac{1}{2}n \log n$	$\log n$	$\frac{1}{2}n+1$	3	no	$-$	$+$
Snir variable ser./par.	$n-1-k$ *	$n-1+k$ *				yes	variable	

* range of size-depth trade-off parameter $k : 0 \leq k \leq n - 2\log n + 2$

All these prefix structures have growing maximum fan-outs (i.e. out-degree of black nodes) if parallelism is increased. This, however, has a negative effect on speed in real circuit implementations. A fundamentally different prefix tree structure was proposed by Kogge and Stone [8] which has all fan-outs bounded by 2 at the minimum structure depth of $\log n$. However, the massively higher circuit and wiring complexity (i.e. more black nodes and edges) undoes the advantages of bounded fan-out in most cases. A mixture of the Kogge-Stone and Brent-Kung prefix structures proposed by Han and Carlson [9] corrects this problem to some degree. Also, these two bounded fan-out parallel-prefix structures are not compatible with the other structures and the synthesis algorithm presented in this paper, and thus were not considered any further in this work.

Table 1 summarizes some characteristics of the serial-prefix and the most common parallel-prefix structures with respect to maximum depth ($D_\bullet$, number of black nodes on the critical path), size ($\#_\bullet$, total number of black nodes), maximum number of black nodes per bit position ($\#_{\bullet/b}^{max}$), wiring complexity ($\#_{tracks}$, horizontal tracks in the graph), maximum fan-out ($FO_\bullet^{max}$), synthesis (compatibility with the presented optimization algorithm), and area/delay performance (A/T). The area/delay performance figures are obtained from a very rough classification based on comparing standard-cell implementations [3].

3 Optimization and Synthesis

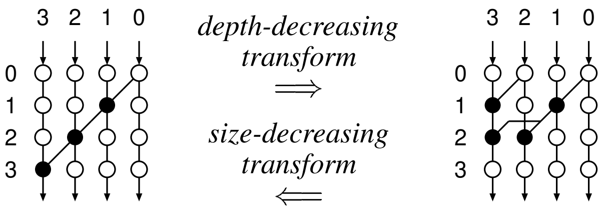
3.1 Prefix Transformation

The optimization of prefix structures bases on a simple local equivalence transformation (i.e. factorization) of the prefix graph [10], called *prefix transformation* in this paper. By using this basic transformation, a serial structure of three black nodes with $D_\bullet = 3$ and $\#_\bullet = 3$ is transformed into a parallel tree structure with $D_\bullet = 2$ and $\#_\bullet = 4$ (see Figure below). Thus, the depth is reduced while the size is increased by one $\bullet$-operator. The transformation can be applied in both directions in order to minimize structure depth (i.e. depth-decreasing transform) or structure size (i.e. size-decreasing transform), respectively.

This local transformation can be applied repeatedly to larger prefix graphs resulting in an overall minimization of structure depth or size or both. A transformation is possible under the following conditions, where (i, l) denotes the node in the i-th column and l-th row of the graph:

$\Longrightarrow$: nodes $(3,1)$ and $(3,2)$ are white,

$\Longleftarrow$: node $(3,3)$ is white and nodes $(3,1)$ and $(3,2)$ have no successors $(i,2)$ or $(i,3)$ with $i > 3$.



It is important to note that the selection and sequence of local transformations carried out is crucial for the quality of the final global optimization result. Different heuristic and non-heuristic algorithms exist for solving this problem.

3.2 Heuristic Optimization Algorithms

Heuristic algorithms based on local transformations are widely used for delay and area optimization of logic networks [11]. Fishburn applied this technique to the timing optimization of prefix circuits and of adders in particular [10]. The same basic transformation as described above is used. However, more complex transforms are derived and stored in a library. An area-minimized logic network together with the timing constraints expressed as input and output signal arrival times are given. Then, repeated local transformations are applied to subcircuits until the timing requirements are met. These subcircuits are selected heuristically, that is, all possible transforms on the most critical path are evaluated by consulting the library, and the simplest one with the best benefit/cost ratio is then carried out.

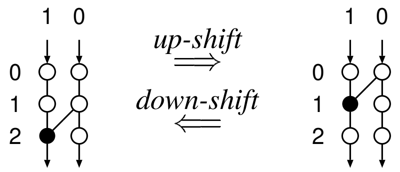
The advantage of such heuristic methods lies in their generality, which enables the optimization of arbitrary logic networks and graphs. On the other hand, the computation effort – which includes static timing analysis, search for possible transformations, and the benefit/cost function evaluation – is very high and can be lessened only to some degree by relying on comprehensive libraries of precomputed transformations. Also, general heuristics are hard to find and only suboptimal in most cases. In the case of parallel-prefix binary addition, very specific heuristics are required in order to obtain perfect prefix trees and the globally optimal adder circuits reported by Fishburn.

3.3 Non-Heuristic Optimization Algorithm

In the heuristic optimization algorithms, only the depth-decreasing transformations are applied which are necessary to meet the timing specifications and which are therefore selected heuristically. Another approach is to first perform all possible depth-decreasing transformations (prefix graph *compression*), resulting in the fastest existing prefix structure. In a second step, size-decreasing transformations are applied wherever possible in order to minimize structure size while remaining in the permitted depth range (prefix graph *expansion*). It can be shown that the resulting prefix structures are optimal in most cases and near-optimal otherwise if the transformations are applied in a simple linear sequence, thus requiring no heuristics at all. Only a trivial up- and down-shift operation of black nodes is used in addition to the basic prefix transformation described above. The conditions for the shift operations are:

$\Longrightarrow$: nodes $(1,1)$ and $(0,1)$ are white,

$\Longleftarrow$: node $(1,2)$ is white and node $(1,1)$ has no successor $(i,2)$ with $i > 1$.



Timing constraints are taken into account by setting appropriate top and bottom margins for each column.

Step 1) *Prefix graph compression*: Compressing a prefix graph means decreasing its depth at the cost of increased size, resulting in a faster circuit implementation. Prefix graph compression is achieved by shifting up the black nodes in each column as far as possible using depth-decreasing transform and up-shift operations. The recursive function `compress_column(i,l)` shifts up a black node (i,l) by one position by applying an up-shift or depth-decreasing transform, if possible. It is called recursively for node $(i,l-1)$ starting at node (i,m), thus working on an entire column from bottom to top. The return value is true if node (i,l) is white (i.e. if a black node (i,l) can be shifted further up), false otherwise. It is used to decide whether a transformation at node (i,l) is possible. The procedure `compress_graph()` compresses the entire prefix graph by calling the column compressing function for each bit position in a linear sequence from the LSB to the MSB. It can easily be seen that the right-to-left bottom-up graph traversal scheme used always generates prefix graphs of minimal depth, which in the case of uniform signal arrival times corresponds to the Sklansky prefix structure. The pseudo code for prefix graph compression looks as follows (left hand side):

```
boolean compress_column(i,l)
  // processes node (i,l)
  // return value = node (i,l) is white
  if node at top of column i
    return false
  else if white node
    compress_column(i,l-1)
    return true
  else if black node with white predecessor
    if predecessor at top of column j
      return false
    else
      shift up black node by one
      compress_column(i,l-1)
      return true
  else // black node with black predecessor
    shift up black node by one
    if compress_column(i,l-1)
      complete depth-decreasing transform
      return true
    else
      undo above up-shift
      return false

compress_graph()
  for i = 0 to n-1
    compress_column(i,m)
```

```
boolean expand_column(i,l)
  // processes node (i,l)
  // return value = node (i,l) is white
  if node at bottom of column i
    return false
  else if white node
    expand_column(i,l+1)
    return true
  else if black node with at least one successor
    expand_column(i,l+1)
    return false
  else if node (i,l+1) is white
    shift down black node by one
    expand_column(i,l+1)
    return true
  else // black node from depth-decr. transform
    shift down black node by one
    if expand_column(i,l+1)
      complete size-decreasing transform
      return true
    else
      undo above down-shift
      return false

expand_graph()
  for i = n-1 to 0
    expand_column(i,1)
```

This simple compression algorithm assumes to start from a serial-prefix graph (i.e. only one black node exists per column initially). The algorithm can easily be expanded by an additional case distinction in order to work on arbitrary prefix graphs. However, in order to get a perfect minimum-depth graph, it must first be expanded to a serial-prefix graph by the following step.

Step 2) *Prefix graph expansion*: Expanding a prefix graph basically means reducing its size at the cost of an increased depth. The prefix graph obtained after compression has minimal depth on all outputs at maximum graph size. If depth specifications are still not met, no solution exists. If, however, graph depth is smaller than required, the columns of the graph can be expanded again in order to minimize graph size. At the same time, fan-outs on the critical nets are reduced resulting in faster circuit implementations. The process of graph expansion is exactly the opposite of graph compression. In other words, graph expansion undoes all unnecessary steps from graph compression. This makes sense since the necessity of a depth-decreasing step in column i is not a priori known during graph compression because it affects columns $j > i$ which are processed later. Thus, prefix graph expansion performs down-shift and size-decreasing transform operations in a left-to-right top-down graph traversal order wherever possible (`expand_column(i,l)` and `expand_graph()`). The pseudo code is therefore very similar, as illustrated above (right hand side).

This expansion algorithm assumes to work on a minimum-depth prefix graph obtained from the above compression step. Again, it can easily be adapted in order to process arbitrary prefix graphs. Under relaxed timing constraints, it will convert any parallel-prefix structure into a serial-prefix one.

3.4 Synthesis of Parallel-Prefix Graphs

The synthesis of size-optimal parallel-prefix graphs – and with that of parallel-prefix adders – under given depth constraints is now trivial. A serial-prefix structure is first generated which then undergoes a graph compression step and a depth-controlled graph expansion step. For a more intuitive graph representation, a final up-shift step can be added which shifts up all black nodes as far as possible without performing any transformation, thus leaving the graph structure unchanged (used in Figures 5–11).

Carry-increment (i.e. bounded-$\#^{max}_{\bullet/b}$) prefix structures are obtained by limiting the number of black nodes per column ($\#^{max}_{\bullet/b}$) through an additional case condition in the graph compression algorithm.

4 Experimental Results and Discussion

The described synthesis algorithm was implemented and tested for a wide range of word lengths and depth constraints. The runtime efficiency of the program is very high thanks to the simple graph traversal algorithms, resulting in computation times below 1s for prefix graphs of up to several hundreds bits.

4.1 Uniform Signal Arrival Profiles

Figures 6(a)–(e) depict the synthesized parallel-prefix structures of depth 5–8 and 12 for uniform signal arrival times. Structure depth ($D_\bullet$) and size ($\#_\bullet$) are also given for each graph. The numbers in parenthesis correspond to structure depth and size after the compression but before the expansion step.

The structures (a) and (d) are size-optimized versions of the Sklansky and Brent-Kung prefix graphs.

For depths in the range of $(2\log n - 3) \leq D_\bullet \leq (n-1)$ a linear trade-off exists between structure depth and size [7]. This is expressed by the lower bound $(D_\bullet + \#_\bullet) \geq (2n-2)$ which is achieved by the synthesized structures, i.e. the algorithm generates size-optimal solutions within this range of structure depths. This linear trade-off exists because the prefix structures are divided into an upper serial-prefix region (with one black node per bit) and a lower Brent-Kung parallel-prefix region (with two black nodes per bit on the average). Changing the structure depth by some value therefore simply moves the border between the two regions (and with that the number of black nodes) by the same amount (Figures 6(c)–(e)). In other words, one depth-decreasing transform suffices for an overall graph depth reduction by one.

In the depth range $\log n \leq D_\bullet < (2\log n - 3)$, however, decreasing structure depth requires shortening of more than one critical path, resulting in an exponential size-depth trade-off (Figures 6(a)–(c)). Put differently, an increasing number of depth-decreasing transforms has to be applied for an overall graph depth reduction by one, as depth gets closer to $\log n$. Most synthesized structures in this range are only near-optimal (except for the structure with minimum depth of $\log n$), while the size-optimal solution is obtained by a bounded-$\#^{max}_{\bullet/b}$ prefix structure with a specific $\#^{max}_{\bullet/b}$ value (compare Figures 5 and 6(b)).

4.2 Non-Uniform Signal Arrival Profiles

Various non-uniform signal arrival profiles were applied, such as late upper/lower half-words, late single bits, and negative/positive slopes on inputs, and vice versa for the outputs. For most profiles, size-optimal or near-optimal structures were generated using the basic algorithm with unbounded $\#^{max}_{\bullet/b}$. As an example, Figures 8(a) and (b) show how a single bit which is late by four black node delays can be accommodated at any bit position in a prefix structure with depth $D_\bullet = \log n + 1$. The structure of Figure 7 has a fast MSB output (corresponds to the carry-out in a prefix adder) and is equivalent to the Brent-Kung prefix algorithm. Figures 9(a)–(d) depict the synthesized prefix graphs for late input and early output upper and lower half-words.

Input signal profiles with steep negative slopes (i.e. bit i arrives earlier than bit $i-1$ for each i) are the only exceptions for which inefficient solutions with many black nodes in some columns are generated. This, however, can be avoided by using bounded-$\#^{max}_{\bullet/b}$ prefix structures. It can be observed that by bounding the number of black nodes per column by $\log n$ ($\#^{max}_{\bullet/b} = \log n$), size-optimal structures are obtained. This is demonstrated in Figure 10 with a typical input signal profile found in the final adder of a multiplier, originating from an unbalanced Wallace tree adder. This example shows the efficient combination of serial and parallel substructures generated, which smoothly adapts to the given signal profiles. In Figure 11, the same signal profile with less steep slopes is used.

4.3 Discussion

As mentioned above, cases exist where size-optimal solutions are obtained only by using bounded-$\#^{max}_{\bullet/b}$ parallel-prefix structures. However, near-optimal structures are generated throughout by setting $\#^{max}_{\bullet/b} = \log n$. Note that this bound normally does not come into effect since most structures (e.g. all structures with uniform signal arrival profiles) have $\#^{max}_{\bullet/b} \leq \log n$ by default.

The synthesis algorithm presented works for any word length n. Because it works on entire prefix graphs, it can be used for structural synthesis but not for the optimization of existing logic networks directly. For the latter, the corresponding prefix graph has first to be extracted which, however, resembles the procedure of subcircuit optimization in the heuristic methods.

Fan-outs significantly influence circuit performance. The total sum of fan-outs in an arbitrary prefix structure is primarily determined by its degree of parallelism and thus by its depth. In the prefix structures used in this work, the accumulated fan-out on the critical path, which determines the circuit delay, is barely influenced by the synthesis algorithm. This is why fan-out is not considered during synthesis. Appropriate buffering and fan-out decoupling of uncritical from critical signal nets is left to the logic optimization and technology mapping step which is always performed after logic synthesis.

Validation of the results on silicon was done by standard-cell implementations in [3], where the prefix adders used in this work showed the best performance measures of all adder architectures.

5 Conclusions

The generality and flexibility of prefix structures proves to be perfectly suited for accommodating arbitrary depth constraints at minimum structure size, thereby allowing for an efficient implementation of custom binary adders. The algorithm described for optimization and synthesis of prefix structures is simple and fast, and it requires no heuristics and knowledge about arithmetic at all. All generated prefix structures are optimal or near-optimal with respect to size under given depth constraints.

Acknowledgment

The author would like to thank Dr. H. Kaeslin for his encouragement and careful reviewing. This work was funded by MICROSWISS (Microelectronics Program of the Swiss Government).

References

[1] K. Keutzer, S. Malik, and A. Saldanhai, "Is redundancy necessary to reduce delay?", *IEEE Trans. Computer-Aided Design*, vol. 10, no. 4, pp. 427–435, Apr. 1991.

[2] A. Tyagi, "A reduced-area scheme for carry-select adders", *IEEE Trans. Comput.*, vol. 42, no. 10, pp. 1162–1170, Oct. 1993.

[3] R. Zimmermann and H. Kaeslin, "Cell-based multilevel carry-increment adders with minimal AT- and PT-products", To be published in *IEEE Trans. VLSI Syst.*

[4] V. G. Oklobdzija, "Design and analysis of fast carry-propagate adder under non-equal input signal arrival profile", in *Proc. 28th Asilomar Conf. Signals, Systems, and Computers*, Nov. 1994, pp. 1398–1401.

[5] J. Sklansky, "Conditional sum addition logic", *IRE Trans. Electron. Comput.*, vol. EC-9, no. 6, pp. 226–231, June 1960.

[6] R. P. Brent and H. T. Kung, "A regular layout for parallel adders", *IEEE Trans. Comput.*, vol. 31, no. 3, pp. 260–264, Mar. 1982.

[7] M. Snir, "Depth-size trade-offs for parallel prefix computation", *J. Algorithms*, vol. 7, pp. 185–201, 1986.

[8] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations", *IEEE Trans. Comput.*, vol. 22, no. 8, pp. 783–791, Aug. 1973.

[9] T. Han and D. A. Carlson, "Fast area-efficient VLSI adders", in *Proc. 8th Computer Arithmetic Symp.*, Como, May 1987, pp. 49–56.

[10] J. P. Fishburn, "A depth-decreasing heuristic for combinational logic; or how to convert a ripple-carry adder into a carry-lookahead adder or anything in-between", in *Proc. 27th Design Automation Conf.*, 1990, pp. 361–364.

[11] K. J. Singh, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Timing optimization of combinational logic", in *Proc. IEEE Conf. Computer-Aided Design*, 1988, pp. 282–285.

$D_\bullet$	$\#_\bullet$
31	31

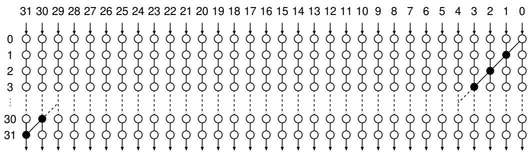
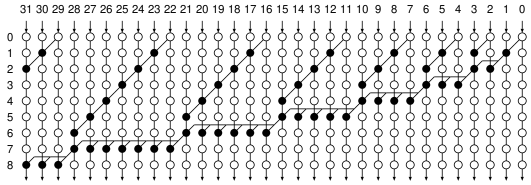


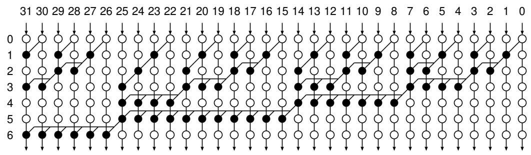
Figure 3 Ripple-carry serial-prefix structure.

$D_\bullet$	$\#_\bullet$
8	54



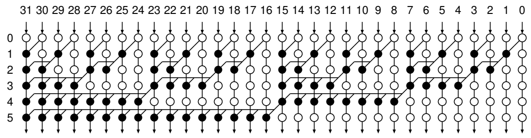
(a)

$D_\bullet$	$\#_\bullet$
6	68



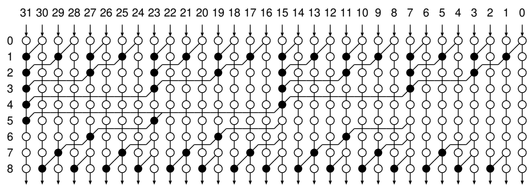
(b)

$D_\bullet$	$\#_\bullet$
5	80



(c)

$D_\bullet$	$\#_\bullet$
8	57



(d)

Figure 4 (a) 1-level carry-increment, (b) 2-level carry-increment, (c) Sklansky, and (d) Brent-Kung parallel-prefix structure.

$D_\bullet$	$\#_\bullet$
6	56
(6)	(68)

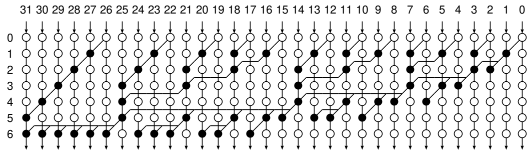
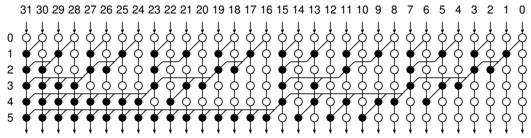


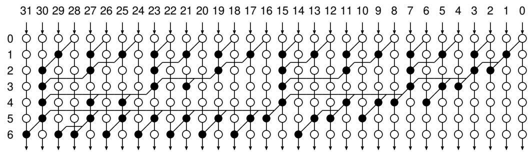
Figure 5 Synthesized minimum-depth bounded-$\#^{max}_{\bullet/b}$ prefix structure ($\#^{max}_{\bullet/b} = 3$).

$D_\bullet$	$\#_\bullet$
5	74
(5)	(80)



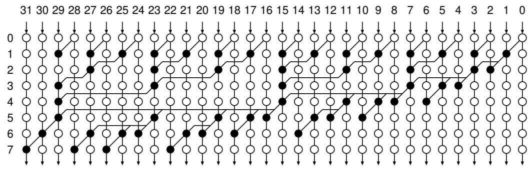
(a)

$D_\bullet$	$\#_\bullet$
6	59
(5)	(80)



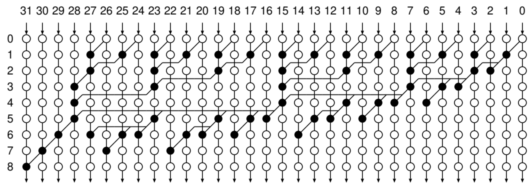
(b)

$D_\bullet$	$\#_\bullet$
7	55
(5)	(80)



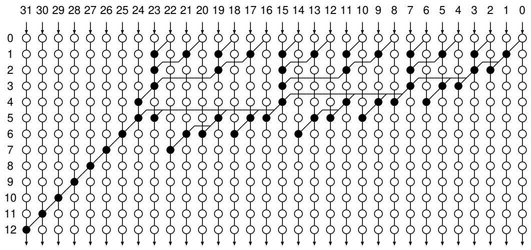
(c)

$D_\bullet$	$\#_\bullet$
8	54
(5)	(80)



(d)

$D_\bullet$	$\#_\bullet$
12	50
(5)	(80)



(e)

Figure 6 Synthesized prefix structures (a)–(e) of depths 5–8 and 12.

$D_\bullet$	$\#_\bullet$
8	57
(8)	(80)

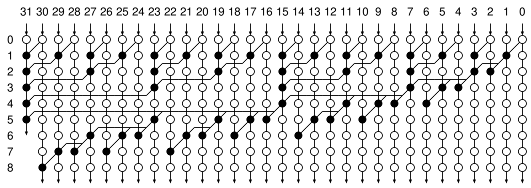


Figure 7 Synthesized minimum-depth prefix structure for the MSB output early by 3 $\bullet$-delays.

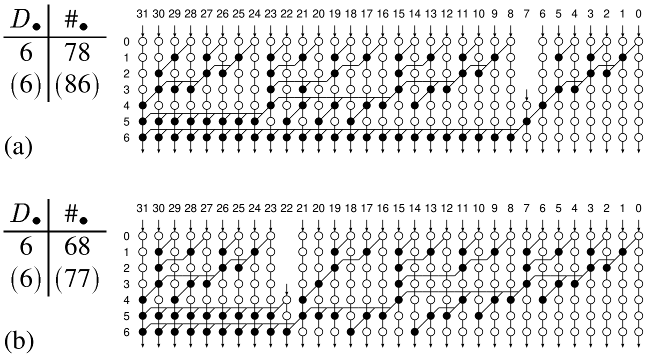


Figure 8 Synthesized minimum-depth prefix structures (a), (b) for a single input bit late by 4 •-delays.

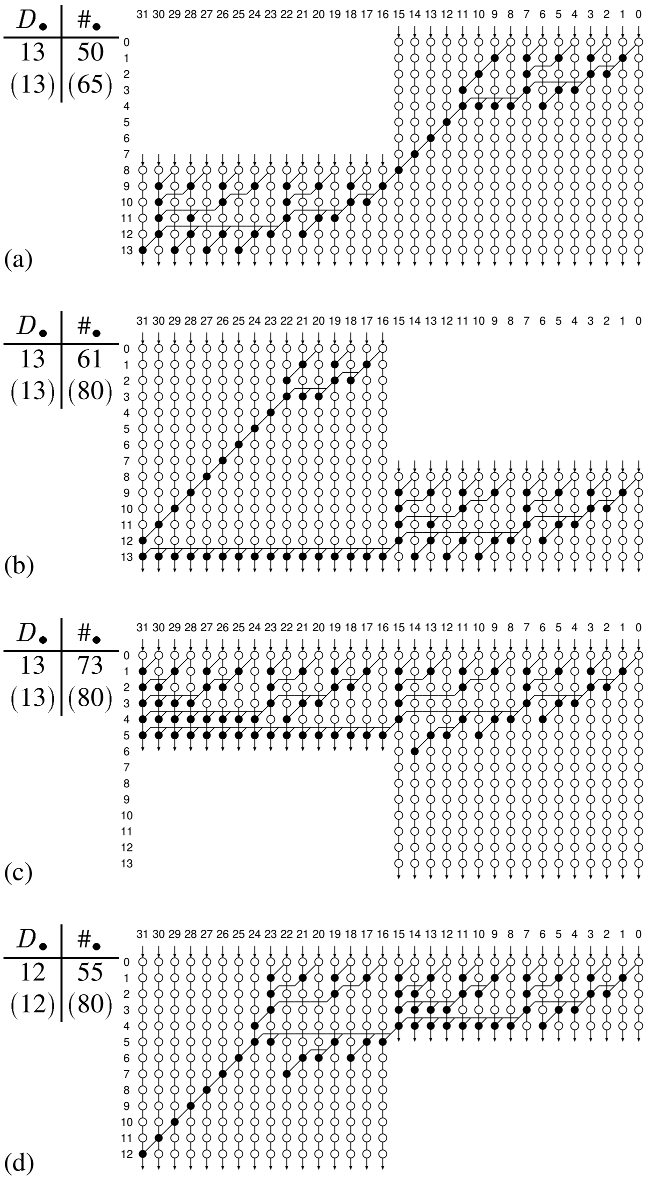


Figure 9 Synthesized minimum-depth prefix structures for (a) late input upper word, (b) late input lower word, (c) early output upper word, and (d) early output lower word by 8 •-delays.

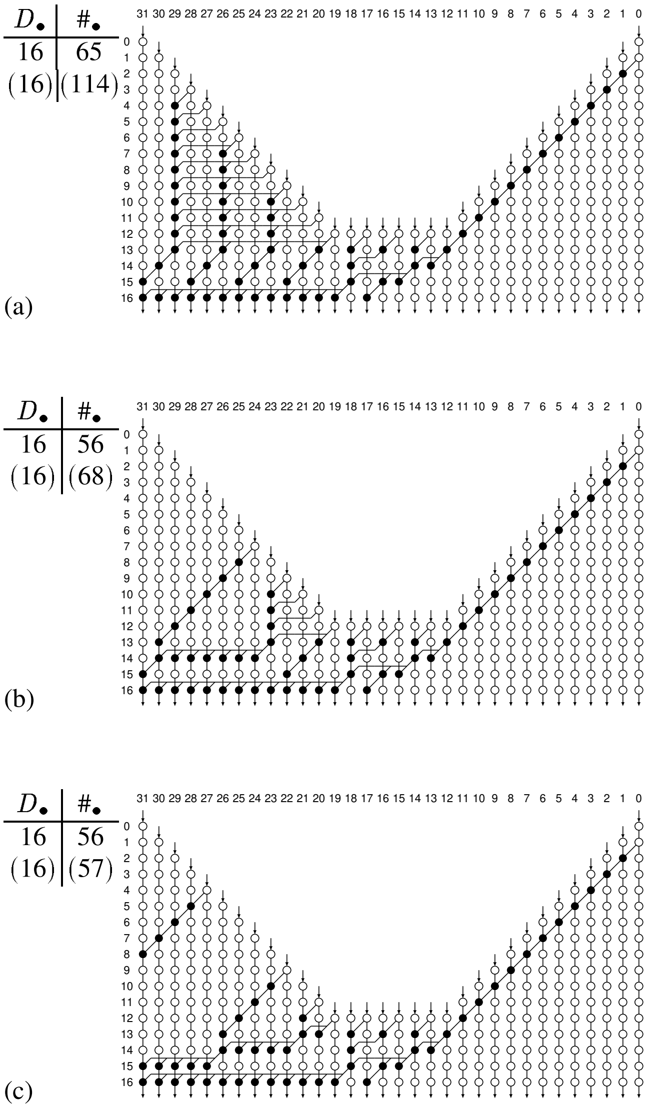


Figure 10 Synthesized minimum-depth prefix structures with (a) no $\#_{\bullet/b}^{max}$ bound, (b) $\#_{\bullet/b}^{max} = 5$ $(= \log n)$ bound, and (c) $\#_{\bullet/b}^{max} = 3$ bound for typical input signal arrival profile in the final adder of a multiplier (steep slopes).

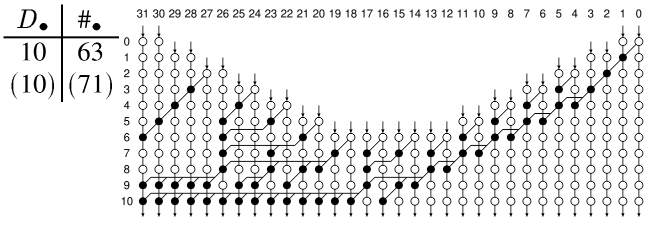


Figure 11 Synthesized minimum-depth prefix structures with $\#_{\bullet/b}^{max} = 5$ $(= \log n)$ bound for typical input signal arrival profile in the final adder of a multiplier (flat slopes).