Parallel Prefix Adder Design

Andrew Beaumont-Smith† and Cheng-Chew Lim
Department of Electrical and Electronic Engineering, The University of Adelaide
Adelaide, 5005, Australia
{abeaumon,cclim}@eleceng.adelaide.edu.au
† Now with Compaq Computer Corporation, MA, USA.

Abstract

This paper introduces two innovations in the design of prefix adder carry trees: use of high–valency prefix cells to achieve low logical depth and end–around carry adders with reduced fan–out loading (compared with the carry select and flagged prefix adders). An algorithm for generating parallel prefix carry trees suitable for use in a VLSI synthesis tool is presented with variable parameters including carry tree width, prefix cell valency, and the spacing of repeated carry trees. The area–delay design space is mapped for a 0.25μm CMOS technology for a range of adder widths as a comparative study.

1 Introduction

VLSI integer adders are critically important elements in processor chips, they are used in floating-point arithmetic units, ALUs, memory addressing, and program counter update. The requirements of the adder are that it is primarily fast and secondarily efficient in terms of power consumption and chip area. Adders are also often responsible for setting the minimum clock cycle time in a processor. Discussions of addition techniques can be found in [Omo94, Kor93, Hwa79].

Parallel prefix (or tree prefix) adders provide a good theoretical basis to make a wide range of design trade-offs in terms of delay, area, regularity and power. Previous work on parallel prefix adders can be found in [Kno99, Zim98, BK82, KS73, KTM91, HC87, LF80] and a recent implementation study can be found in [MKA+01]. The addition problem can be expressed in terms of kill (k_i), generate (g_i) and carry (c_i) signals at each bit position, i for a width, w adder with the following equations (where $i = 0, \ldots, w-1$ and c_0 is the carry–in):

$$k_i = \overline{a_i}.\overline{b_i} \quad (1)$$

$$g_i = a_i.b_i \quad (2)$$

$$c_i = g_{i-1} + \sum_{j=0}^{i-2}(g_j.\overline{\sum_{k=j+1}^{i-1} k_k}) \quad (3)$$

$$s_i = p_i \oplus c_i \quad (4)$$

The direct implementation of these expressions creates an adder with large complex gate towards the *msb* position of the carry assimilation path. This single large complex gate will be too slow in CMOS VLSI, so the design is modularised by breaking it into trees of smaller and faster adders which are more readily implemented [BK82]. The *group* generate, G_n^m and *group* kill, K_n^m signals at significance n, are calculated from bit positions m to n:

$$G_n^m = g_n + \sum_{i=m}^{n-1}(\overline{K_n^{i+1}}.g_i) \qquad m < n \quad (5)$$

$$G_n^m = g_n \qquad m = n \quad (6)$$

$$K_n^m = \sum_{i=m}^{n} k_i \qquad m \leq n \quad (7)$$

The carry expression in equation 3 can now be expressed recursively using the group generate and group kill signals:

$$c_i = G_{i-1}^0 = G_{i-1}^n + \overline{K_{i-1}^n}.G_{n-1}^0 \qquad 1 \leq n \leq i-1$$

This allows modular sub–adders to be constructed and combined to form trees. At one extreme case, the slowest and smallest ripple carry adder can be recursively constructed using equations 5 and 7 for $m = n-1$, although it is *not* a parallel prefix adder. All adders in the parallel prefix adder design space differ only in the carry tree structure – the bitwise kill, generate and sum signals are the same.

Brent and Kungs • operator [BK82] is now introduced. Let GK_n^m represent the pair (G_n^m,K_n^m), then the following expression:

$$GK_n^m = (G_n^k + (\overline{K_n^k}.G_{k-1}^m), K_n^k + K_{k-1}^m) = GK_n^k \bullet GK_{k-1}^m \quad (8)$$

holds for $0 \leq m \leq k-1$ and $k \leq n \leq w-1$.

0-7695-1150-3/01 $10.00 © 2001 IEEE

1.1 Properties

There are three properties that can be applied to modify the structure of the carry assimilation process using the '•' operator:

- **associativity**: $GK_n^m = (GK_n^k \bullet GK_{k-1}^j) \bullet GK_{j-1}^m$
  $= GK_n^k \bullet (GK_{k-1}^j \bullet GK_{j-1}^m)$
  GK sub–expressions have no serial dependence and can be evaluated in parallel. This is the most important property as it allows the design of *parallel* prefix adders.
- **non-commutativity**:
  $GK_n^k \bullet GK_{k-1}^m \neq GK_{k-1}^m \bullet GK_n^k$
  A generate signal can only be propagated in the direction of increasing significance (group kill is commutative, however this property is not particularly useful).
- **idempotency**: $GK_n^m = GK_n^k \bullet GK_j^m$
  for $m \leq k < j \leq n$ or $m < k = j < n$.
  Two sub–expressions must meet or overlap if they are to form a valid contiguous GK expression.

1.2 Prefix cell operators

Six prefix cells (three black and three grey) can now be defined which implement the • operator as a function of the number of GK inputs (or valency) of the prefix cell. The valency–2, valency–3 and valency–4 black prefix cells (denoted as $\bullet_2$, $\bullet_3$ and $\bullet_4$, respectively) are shown in Figure 1 and represent a range of implementable gates in submicron VLSI technology with a fan-in limit of four. A static CMOS $\bullet_4$ prefix cell schematic is shown in Figure 2 with optional inverters at the outputs to buffer high fan-outs. Grey prefix cells (denoted as $\circ_2$, $\circ_3$ and $\circ_4$) replace black prefix cells at the final cell position in the carry tree before the sum is calculated as only the group generate term is required to calculate the final carry vector $C = c_0, \ldots, c_{w-1}$.

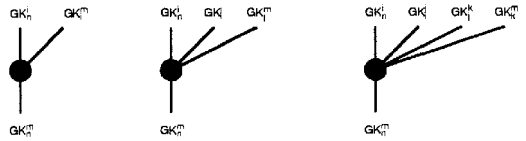

Figure 1. Black Prefix Cell • Operators: (a) $\bullet_2$ (valency–2), (b) $\bullet_3$ (valency–3) and (c) $\bullet_4$ (valency–4).

1.3 Previous prefix carry tree designs

Parallel prefix adders which represent a range of the near–minimum logic depth design space are discussed below. All of the adders use valency–2 prefix cells and exploit the associativity property to parallelise the carry assimilation. The Brent-Kung parallel prefix adder [BK82] has a low fan–out from each prefix cell but has a long critical path and is not capable of very high speed addition. The Ladner–Fischer adder [LF80] has a minimum logic depth and recursively doubles the fan-out at each gate until the last stage has a fan–out of $w/2$, and a wire that runs half the adder width. It is the parallel prefix version of the carry–select adder. The Kogge–Stone prefix architecture [KS73] uses more than twice as many prefix cells as the Brent–Kung adder to achieve the theoretical minimum logic depth, $O(log_2 w)$ (using valency–2 prefix cells). The Ladner–Fischer and Kogge–Stone carry trees represent two end designs of minimum depth adders using valency–2 prefix cells. Knowles [Kno99] gives an insight into the parallel prefix adder design space and how they may be specified and studies several key 32–bit minimum logic depth carry tree designs using commercial VLSI layout tools. The study was restricted to using valency–2 prefix cells, in common with all of the other published work presented so far. This work has been extended to larger wordlength adders using higher valency prefix cells, and an algorithm is presented in the next section to generate parallel prefix adders.

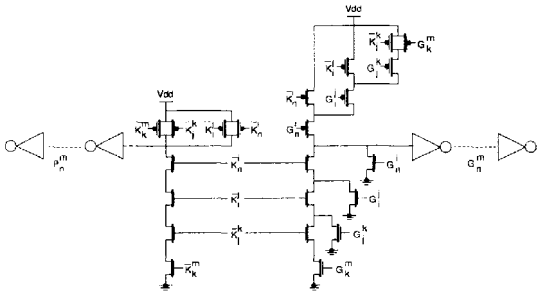

Figure 2. A CMOS $\bullet_4$ (valency–4) black prefix cell schematic with optional output buffers.

2 Synthesis of Parallel Prefix Carry trees

There has been much interest in the synthesis of VLSI circuits to reduce the design time [WE95, Zim98]. Adders are one of the more frequently used circuits and their regularity makes them good candidates for VLSI synthesis, which can also be used to assess design trade-offs. Two such trade-offs are:

- **Higher valency prefix cells**
  By using rows of higher valency prefix cells, there can be less prefix cells in the critical path (and less used overall) or shorter interconnect lengths since the num-

ber of assimilated carries is more than recursively doubled at each row (tripled or quadrupled). A higher valency used in the first row of prefix cells leads to a more compact layout, if the cell pitch is not already limited by wiring pitch, however, the delay of the prefix cell increases with valency (fan–in).

- **Repeated carry chains**
  Repeated carry chains at fixed intervals can be made to overlap and share common sub–adders. This results in the availability of more carry signals to drive the fan-out in the last stage of the carry tree. The Kogge–Stone carry tree is the minimum case (repeat every bit) for this technique, Han–Carlson's carry tree repeats every second bit and the Ladner–Fisher adder has one only carry chain with maximum fan–out in the last stage.

2.1 Parallel prefix–tree algorithm

An algorithm is now presented which can construct the design space of prefix carry trees including those discussed in Section 1.3. The carry tree is specified by the width of the adder (w), the maximum number of prefix cell inputs (q_i) in each row (i), and the stride (d) of the repeated carry chains. A set of rules are defined for the iterative construction of parallel prefix adders (with the minimum number of prefix cell rows and the maximum carry assimilation distance per row) below:

- **Rule 1:** Each $\bullet_p$ prefix cell in row i has a number of inputs, $p \leq q_i$ and of the set $p = \{2, 3, 4\}$ and prefix cells have a constant stride across the row.
- **Rule 2:** The span of bits assimilated in row i is recursively multiplied by q_i. This maximises carry propagation distance and minimises logic depth of the carry tree.
- **Rule 3:** The product of all q_i equals the width. This guarantees that all q_i are a factor of the width (w).
- **Rule 4:** Black prefix cells are converted to grey prefix cells if they are in the final position of the carry tree (output to a sum cell) *and/or* drive another grey prefix cell input at the least significance.

Carry trees may also be truncated to a width less than the product of all q_i (rule 3) although this may not lead to the best solution for small delay. Additional conditions for the straight forward construction of end-around carry trees include forcing the carry tree stride to be a factor of the width (q_1 is then a common factor of the stride and width) and enforcing rule 3. This ensures that repeated and shifted carry trees are correctly aligned when prefix cells are placed modulo the width. A carry tree is specified by the set:

$$w, d, eac, [q_1 q_2 \ldots] \tag{9}$$

where $eac = 1$ to generate an end–around carry adder (see Section 3). It was not necessary to specify the lateral fan–out as reported in other work [Kno99] as this can be determined for a minimum depth tree. The iterative algorithm describing how each adder is built is given in Figure 3. The carry tree is constructed by creating a stride width slice and copying it contiguously across the carry tree from the lsb to msb. Sub-adders are needed to complete the carry assimilation in the first rows if the stride is greater than the stride is greater than q_1. The prefix cells which extend past the end of the carry tree are placed in cell locations modulo the width. Redundant inputs or cells are trimmed from the carry tree near the lsb if no carry-in is required. A carry-in may be incorporated by retaining these inputs and injecting the carry into them.

2.2 Algorithm implementation and design study

A MATLAB program was written to implement the algorithm and run to find the area and delay for all valid 8–bit, 32–bit and 64–bit adder carry trees for a commercial $0.25\mu m$ CMOS technology. This study is intended to compare relative delays of carry trees under a set of implementation assumptions and is not an absolute indicator of adder delay.

The design strategy is that prefix cells are constructed using static CMOS gates and consist of a complex gate and zero or more inverters to buffer the outputs. For higher valency prefix cells beyond 4, the complex gates need to be split to reduce the delay. The transistor sizing for each complex gate is the transistor stack height times minimum size and follows standard CMOS sizing rules [WE95]. Zero or more inverters are added to each prefix cell output to minimise the delay based on this model. The buffers are individually sized based on three times the previous stage size to minimise delay. Although this is a simplistic approach to transistor sizing, it achieves an acceptable trade-off with design complexity and delay. The alternative is to minimise the delay of the each adder using linear programming techniques to tune each transistor size which is beyond the scope of this paper for such a large number of designs.

The prefix cells were characterised by building a table of delays and areas based on HSPICE simulations derived from layouts for each prefix cell using typical process parameters at $105C$. The prefix cell delay parameters (as a function of driving prefix cell valency) include the delay of the complex gate and per fan-out delay for each receiving cell valency or inverter as a function of drive strength. The effect of wiring was taken into account based on RC delays by adding a delay per cell pitch traversed for each wire as a function of driving cell valency or inverter strength. It was determined that the time of flight delay is negligible for the

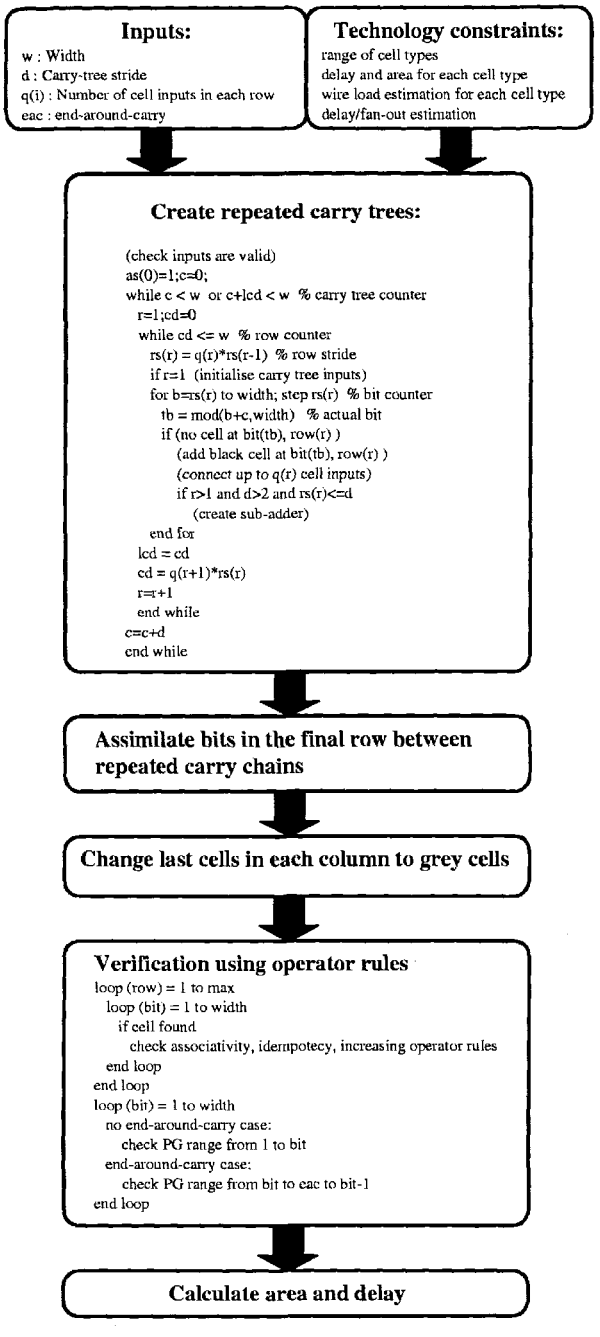

Figure 3. Algorithm for generating parallel prefix adders.

short distances traversed (up to 0.5mm) and it is assumed a good ground return strategy is in place to reduce its effect [CBF01]. It is also assumed that all input operands arrive simultaneously.

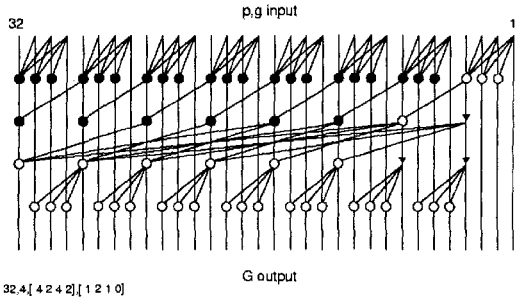

Figure 4. 32–bit carry tree : good solution for small $area - delay$ and $area - delay^2$ metric.

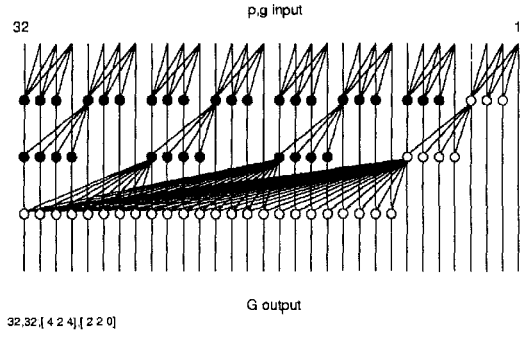

Figure 5. 32–bit carry tree : small delay solution (after Kogge–Stone adder).

Generated drawings of noteworthy carry trees are shown in Figures 4 to 6. The string in the bottom left-hand corner of each carry tree is the input specification (9) followed by the number of inverters needed to minimise the delay for each row (top to bottom) using the model described. Each adder is functionally verified by the program after construction by applying the $\bullet$ operator properties from each output bit to check the range of G.

A map of the area–delay relationship for around 120 generated carry trees are shown in Figures 7, 8 and 9 for nominal widths of 8–bit, 32–bit and 64–bit, respectively. The delay does not include the bitwise propagate and generate or the sum cells. Two curves are also shown in each figure for $area - delay = constant$ and $area - delay^2 = constant$ where the constant value is the adder design at the minimum of the respective metric. Other solutions close to these curves are also a good choice if this is the design

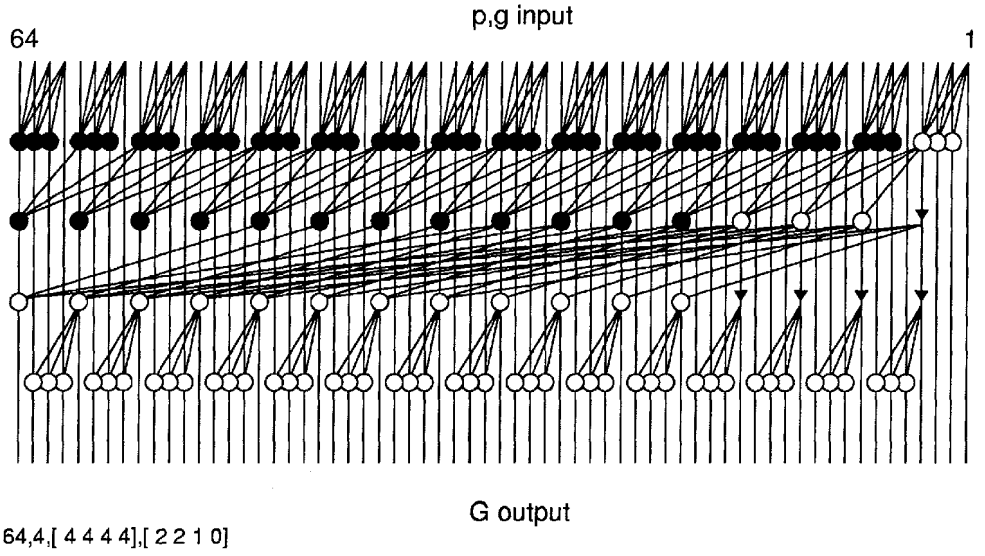

Figure 6. 64–bit carry tree : good solution for small $area - delay$ and $area - delay^2$ metric.

metric. The end point carry tree designs of Kogge-Stone and conditional sum along with the Han-Carlson carry tree are marked. Wider adders than that specified also appear in these figures and are due to the product of the row input sizes being larger than the specified width (see Rule 4). These wider adders may be truncated to the specified width, however, this does not significantly reduce the delay. Although the 64, 1, [4 4 4] adder in Figure 9 has the smallest delay in this study (three rows of valency–4 prefix cells with low fan–out), it probably contains too many wires to be practically implemented.

3 End–around carry Adders

Floating–point adders that speculatively calculate two results based on exponent differences (near and far path) use a significand adder in the near path which provides the magnitude only of the result and performs no rounding. In this case, an end–around carry (EAC) adder [Tya93] can be used by effectively connecting the carry–out to carry–in. In the context of a prefix adder [Bur98], the delay is the carry propagation time across the adder plus the extra delay of the selection logic (two gate delays but a large fan-out). The focus of the prefix EAC (or flagged) adder design is to reduce the EAC operation to a single signal (carry–out) and drive a large fan-out (carry–in) which can also be used for incrementing the result in the case of a rounding adder. An EAC adder has the property that the carry distance is bounded by the width of the adder. For example, if the carry–out was as a result of a carry generated at the i^{th} bit and propagated along the width of the adder, the carry–in may propagate at most to bit position $i - 1$. Flagged prefix and dual carry chain (carry select) adders propagate the carry along twice the adder width. A parallel prefix carry tree analogous to this is shown in Figure 10. Equations 5 and 7 can be extended in equations 10 and 11 to account for the EAC propagation.

$$G_n^m = g_n + \sum_{i=m}^{n+w-1} (P_n^{<i+1>_w} . g_{<i>_w}) \quad m > n \quad (10)$$

$$P_n^m = \prod_{i=m}^{n+w-1} p_{<i>_w} \quad m > n \quad (11)$$

Example 16-bit EAC parallel prefix adders were constructed using the algorithm in the previous section and are shown in Figures 11–13. These EAC carry trees are potentially faster than the flagged prefix adder and carry select structures since it does not perform a reduction to a single carry–out signal then drive the large number of flag cells or multiplexers, however there are more long wires to route.

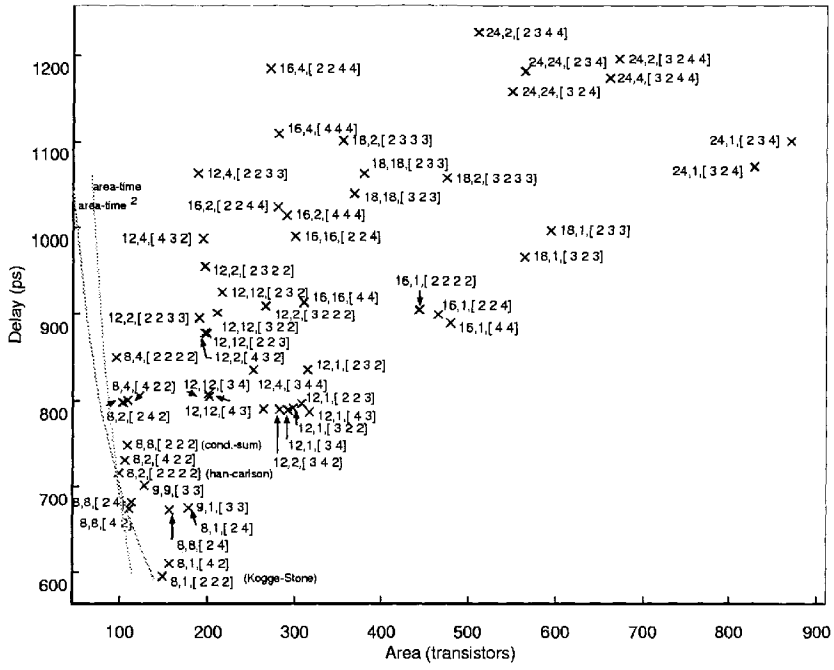

Figure 7. Area vs. delay for 8-bit carry trees.

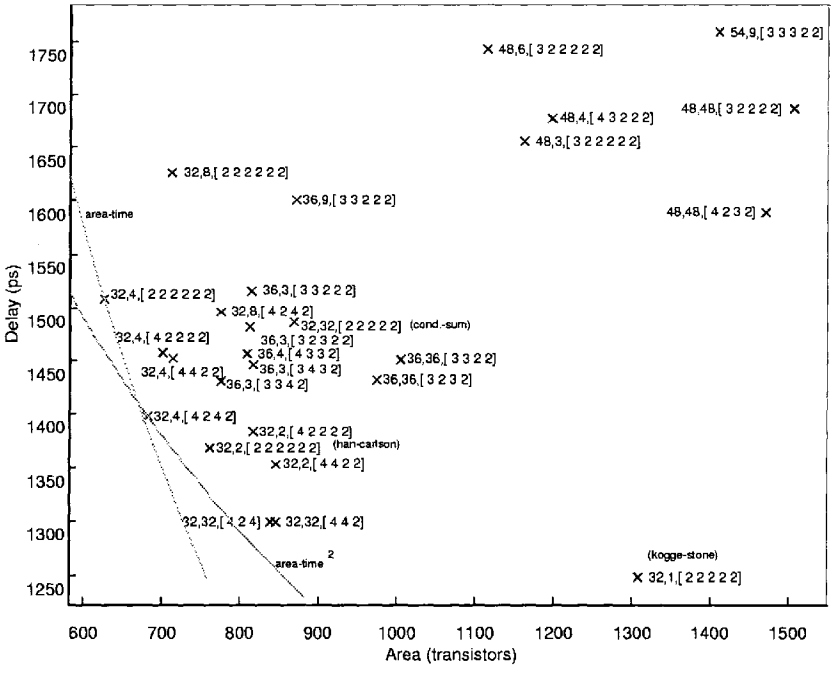

Figure 8. Area vs. delay for 32-bit carry trees.

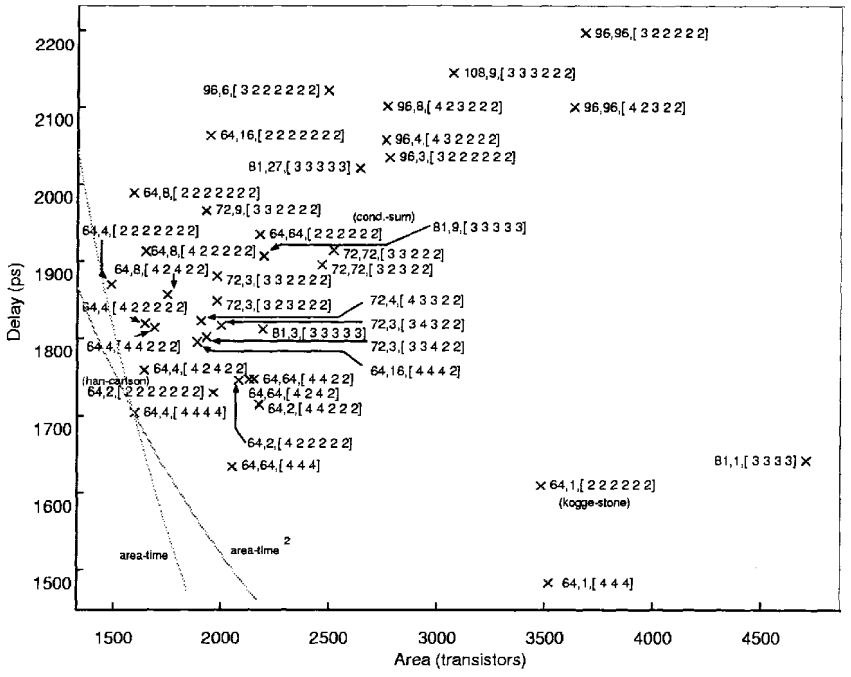

Figure 9. Area vs. delay for 64-bit carry trees.

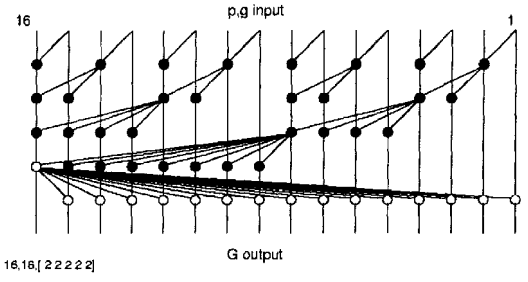

Figure 10. EAC carry tree

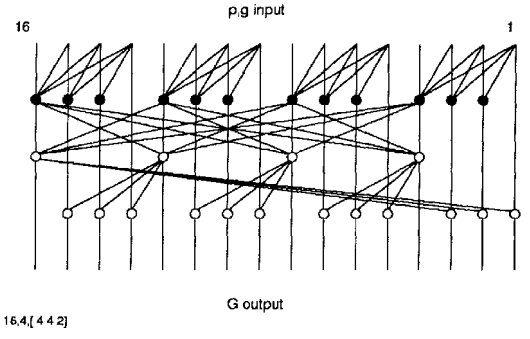

Figure 11. EAC valency-4 carry tree

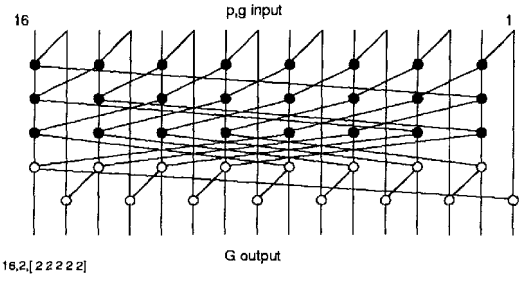

Figure 12. EAC carry tree(Han-Carlson)

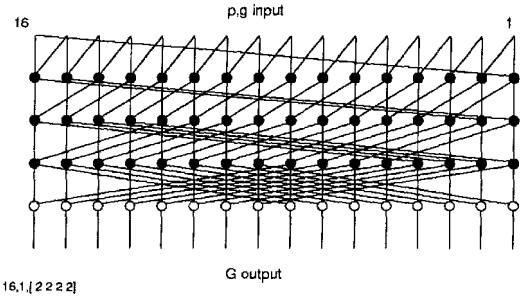

Figure 13. EAC carry tree(Kogge-Stone)

4 Summary

This paper has introduced two innovations in the design of prefix adder carry trees: high–valency adders with low logical depth and end–around carry adders with reduced fan–out loading (compared with the flagged prefix adder). An algorithm for generating parallel prefix trees with variable parameters was introduced and performance results derived from SPICE simulations using a uniform sizing model have been presented. It is clear that prefix adder trees provide a rich solution space for the design and implementation of high–performance adders, owing to its structural flexibility. Although this study did not give clear indications of which carry tree solution to use, it provides a set of carry tree architectures which can be further studied to find the best solution for a particular application and design metric.

5 Acknowledgement

Thanks to Neil Burgess at The University of Cardiff for help in the preparation of this paper. Thanks also to Shannon Morton at Compaq Computer Corp. for discussions on interconnect analysis and the reviewers for helping to improve the content.

References

[BK82] R.P. Brent and H.T. Kung. A Regular Layout for Parallel Adders. *IEEE Transactions on Computers*, 31:260–264, Mar. 1982.

[Bur98] Neil Burgess. The Flagged Prefix Adder for Dual Additions. In *Proc. SPIE ASPAAI-7*, volume 3461, pages 567–575, San Diego, Jul. 1998.

[CBF01] Anantha Chandrakasan, William J. Bowhill, and Frank Fox. *Design of High Performance Microprocessor Circuits*. IEEE Press, 2001.

[HC87] T. Han and D.A. Carlson. Fast area-efficient VLSI adders. In *Proc. 8th Symposium on Computer Arithmetic*, pages 49–56, september 1987.

[Hwa79] K. Hwang. *Computer Arithmetic:Principles, Architecture and Design*. John Wiley & Sons, 1979.

[Kno99] Simon Knowles. A Family of Adders. In *14^{th} IEEE Symposium on Computer Arithmetic*, Adelaide Australia, April 1999.

[Kor93] Israel Koren. *Computer Arithmetic Algorithms*. Prentice Hall, 1993.

[KS73] P.M. Kogge and H.S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence relations. *IEEE Transactions on Computers*, 22:786–793, Aug. 1973.

[KTM91] J. Kowalczuk, S. Tudor, and D. Mlynek. A new architecture for an automatic generation of fast pipeline adders. In *Proc. ESSCIRC*, pages 101–104, September 1991.

[LF80] R.E. Ladner and M.J. Fischer. Parallel Prefix Computation. *Journal of the ACM*, 27(4):831–838, Oct. 1980.

[MKA+01] Sanu Mathew, Ram Krishnamurthy, Mark Anders, Rafael Rios, Kaizad Mistry, and K. Soumyanath. Sub-500ps 64b ALUs in 0.18μm SOI/Bulk CMOS: Design and Scalaing Trends. In *ISSCC Dig. Tech. Papers*, pages 318–319, Feb. 2001.

[Omo94] Amos R. Omondi. *Computer Arithmetic Systems, Algorithms, Architecture and Implementations*. Prentice Hall, 1994.

[Tya93] A. Tyagi. A reduced-area scheme for carry-select adders. *IEEE Transactions on Computers*, 42:1163–1179, 1993.

[WE95] Neil Weste and Kamran Eshragian. *Principles of CMOS VLSI Design*. Addison Wesley, Second edition, 1995.

[Zim98] Reto Zimmermann. *Binary Adder Architectures for Cell-Based VLSI and their Synthesis*. Hartung-Gorre, 1998.