Efficient Mapping of Addition Recurrence Algorithms in CMOS

Bart R. Zeydel
ACSEL
University of California
Davis, CA 95616
brzeydel@ucdavis.edu

Theo T.J.H. Kluter
EPFL
Lausanne, Switzerland
ties.kluter@epfl.ch

Vojin G. Oklobdzija
ACSEL
University of California
Davis, CA 95616
vojin@ucdavis.edu

Abstract

Efficient adder design requires proper selection of a recurrence algorithm and its realization. Each of the algorithms: Weinberger's, Ling's and Doran's were analyzed for its flexibility in representation and suitability for realization in CMOS. We describe general techniques for developing efficient realizations based on CMOS technology constraints when using Ling's algorithm. From these techniques we propose two high-performance realizations that achieve 1 FO4 delay improvement at the same energy and 50% energy reduction at the same delay than existing Ling and Weinberger designs.

1. Introduction

For almost half a century realizations of addition algorithms have been continually refined to improve performance due to changing technology and operating constraints. With each technology generation the gap between the underlying algorithms for addition and efficient realization of those algorithms has grown. As a result many of the adders in use today were developed for another technology and under a different set of constraints than those imposed by current technology, such as energy-efficiency. To alleviate this problem we developed a method for analyzing designs in the energy-delay space [14] which allowed for the energy-delay tradeoffs to be taken into account. However we did not provide guidance for algorithm selection and realization. In this paper we intend to reduce the gap between algorithm selection and efficient realization. We explore the leading addition recurrence algorithms and their realizations that have been developed, to identify favorable characteristics of each for efficient realization in modern CMOS technology.

The paper is organized as follows. Section 2 examines Weinberger's historic recurrence for addition. Section 3 analyzes Ling's transformation of Weinberger's recurrence. Section 4 evaluates Doran's recurrence transforms. In Section 5 techniques for resolving the implications of CMOS realization are presented. In Section 6 adder realizations for Ling under different constraints are proposed. Section 7 presents a comparison of various schemes in the energy-delay space to demonstrate the relative performance and energy-efficiency of the proposed structures. Section 8 concludes the work.

2. Weinberger's Recurrence for Addition

Weinberger presented a general form for carry recurrence which was not limited in group sizes and number of levels for carry computation. The traditional carry-look-ahead (CLA) adder is a specific case of this general carry recurrence. We use the phrase Weinberger's recurrence to describe the general recurrence form as originally presented by Weinberger [1]. The sum and carry are defined and indexed as follows.

$$S_i = a_i \oplus b_i \oplus C_i$$

$$C_{i+1} = a_i b_i + (a_i + b_i) \cdot C_i \qquad (1)$$

In Weinberger's recurrence, the carry propagation speed is improved through the use of generate and propagate. Propagate can either be implemented using an OR or an XOR. To distinguish them we refer to the OR realization of propagate as transmit, t, and the XOR realization as, p. We define Weinberger's bit operations as:

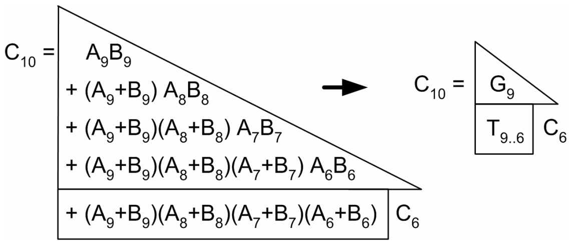
$$g_i = a_i b_i$$

$$t_i = a_i + b_i$$

Substituting into (1) obtains:

$$C_{i+1} = g_i + t_i C_i$$

Weinberger demonstrated that the recurrence applies to any prefix variation [1], through the use of group generate, G, and group transmit, T (Fig.1).



C10 = A9B9
+ (A9+B9) A8B8
+ (A9+B9)(A8+B8) A7B7
+ (A9+B9)(A8+B8)(A7+B7) A6B6
+ (A9+B9)(A8+B8)(A7+B7)(A6+B6) C6
C10 = G9
T9..6 C6

Figure 1. Weinberger's recurrence for addition

The computations of G and T are associative and idempotent, which allows for a wide range of recurrence tree possibilities for the carry computation. Kogge-Stone [7], Han-Carlson [8], Brent-Kung [9], and Ladner-Fischer [10] are just common recurrence tree variations for addition using Weinberger's recurrence as discussed in [11].

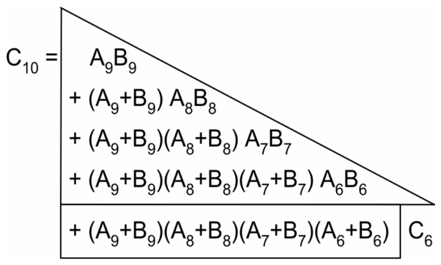
3. Ling's Transformation

Technology limitations on fan-in and wired-OR in ECL (transistor stack height in CMOS) motivated simplifying Weinberger's recurrence. Ling [2] developed a transformation which was able to achieve this simplification by factoring transmit, t, from carry.

$$C_{i+1} = g_i + t_i C_i$$

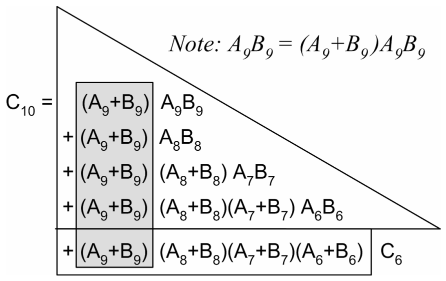
$$C_{i+1} = t_i \left(g_i + C_i\right)$$

The transformation from Weinberger's recurrence to Ling's is shown in Fig. 2.

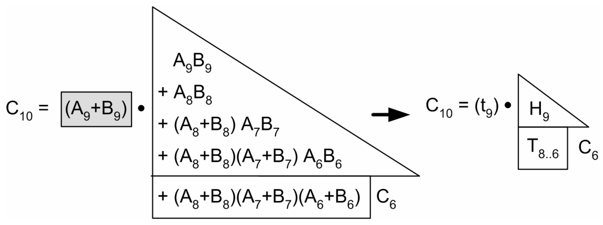


C10 = A9B9
+ (A9+B9) A8B8
+ (A9+B9)(A8+B8) A7B7
+ (A9+B9)(A8+B8)(A7+B7) A6B6
+ (A9+B9)(A8+B8)(A7+B7)(A6+B6) C6

(a)



(b)



(c)

Figure 2. (a) Weinberger's recurrence (b) Intermediate form (c) Ling's transformation

To create Ling's recurrence, this transformation is applied to C_6 which allows for a recurrence for C_{10} to be created using H and T as shown in Fig. 3. For the recurrence, H_9 has one less term than G_9 in Weinberger's recurrence. To allow for recurrence $T_{8..6}$ is combined with t_5, resulting in the same number of terms as $T_{9..6}$ used in Weinberger's recurrence.

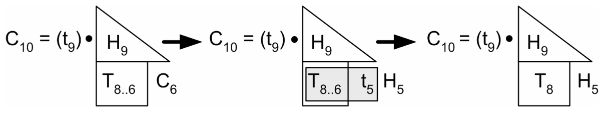


Figure 3. Ling's recurrence

Ling's recurrence is performed on H_i :

$$C_{i+1} = t_i H_i$$

$$H_i = g_i + t_{i-1} \cdot H_{i-1}$$

The group recurrence relation for H and T allows for parallel prefix computation:
(H^+ and T^+ denote the next logic level)

$$H_i^{\,+} = H_i + T_{i-1} \cdot H_{i-1}$$

$$T_{i-1}^{\,+} = T_{i-1} T_{i-2}$$

An advantage of Ling's transformation is compatibility with the prefix operator "●" for the recurrence of H_i and T_{i-1}

$$\begin{pmatrix} H_i \\ T_{i-1} \end{pmatrix} \bullet \begin{pmatrix} H_j \\ T_{j-1} \end{pmatrix} = \begin{pmatrix} H_i + T_{i-1} H_j \\ T_{i-1} T_{j-1} \end{pmatrix}$$

As a result, Ling's H and T have the same favorable properties as Weinberger's G and T [11] when using the prefix operator "$\bullet$" such as associativity:

$$\begin{pmatrix} H_{i..j+1} \\ T_{i-1..j} \end{pmatrix} \bullet \begin{pmatrix} H_{j..k} \\ T_{j-1..k-1} \end{pmatrix} = \begin{pmatrix} H_{i..n+1} \\ T_{i..n} \end{pmatrix} \bullet \begin{pmatrix} H_{n..k} \\ T_{n-1..k-1} \end{pmatrix}$$

where $i > j \geq n > k$

and idempotency:

$$\begin{pmatrix} H_{i..j+1} \\ T_{i-1..j} \end{pmatrix} \bullet \begin{pmatrix} H_{j..k} \\ T_{j-1..k-1} \end{pmatrix} = \begin{pmatrix} H_{i..k} \\ T_{i-1..k-1} \end{pmatrix}$$

Ling's transformation reduces the complexity of the recurrence by one term, t_i, in the first stage of the carry tree through the use of H_{i-1} instead of C_i. However the reduction in recurrence complexity is achieved at the expense of an increase in sum complexity.

$$S_i = a_i \oplus b_i \oplus (t_{i-1} \cdot H_{i-1})$$

The increased sum complexity can be mitigated through the use of conditional logic [12,13]. The summation can be implemented using a multiplexer, with H_{i-1} as the select.

$$S_i = \begin{cases} a_i \oplus b_i & \text{if } H_{i-1} = 0 \\ a_i \oplus b_i \oplus t_{i-1} & \text{if } H_{i-1} = 1 \end{cases}$$

A multiplexer allows for sum to be computed with no increased complexity on the critical path compared to Weinberger's recurrence. The net benefit from using Ling's recurrence compared to Weinberger's is the delay improvement achieved from reducing the recurrence by one term.

4. Doran's Transformations

Doran demonstrated that there are four transformations with Ling-like properties [3]. Two of the transformations fit directly into Ling's transformation, and the other two fit into a type we refer to as Doran's transformation.

The two forms of Ling's transformation eliminate t_i from the recurrence. In Ling's original form:

$$C_{i+1} = t_i \cdot (g_i + C_i)$$

and in the second form:

$$C_{i+1} = t_i \cdot (\overline{p_i} + C_i)$$

The second form is more complex as it requires that an XOR be added to the first stage for the recurrence calculation of carry, which does not simplify the sum calculation. Therefore the first form is more suitable for CMOS realization.

The two forms of Doran's transformation remove g_i from Weinberger's recurrence.

$$C_{i+1} = g_i + (t_i \cdot C_i)$$

Doran's transformation attempts to simplify Weinberger's recurrence by removing g_i from the recurrence for C_{i+1} (Fig. 4).

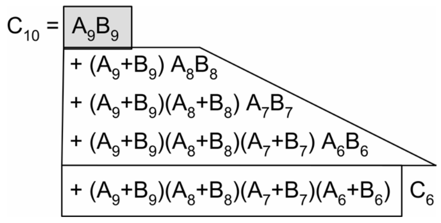

C_{10} =
A_9B_9
+ (A_9+B_9) A_8B_8
+ $(A_9+B_9)(A_8+B_8)$ A_7B_7
+ $(A_9+B_9)(A_8+B_8)(A_7+B_7)$ A_6B_6
+ $(A_9+B_9)(A_8+B_8)(A_7+B_7)(A_6+B_6)$
C_6

Figure 4. Factoring g_9 from C_{10}

Removing g_i reduces the number of terms in the OR while maintaining the same stack height as Weinberger. However the complexity of T remains unchanged. A general recurrence form for Doran's transformation is shown in Fig. 5.

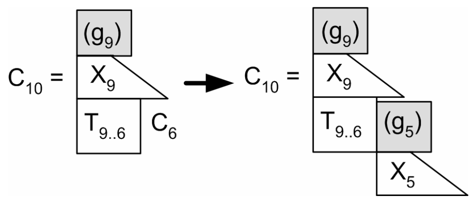


Figure 5. Doran's g_i + X_i transformation

X_9 results in one less OR term than G_9 used in Weinberger's recurrence. However, the group recurrence relation for X is more complex. The group recurrence relation for X_i and T_i are shown below:
(X^+ and T^+ denote the next logic level)

$$X_i^+ = X_i + T_i \cdot (g_{i-1} + X_{i-1})$$

$$T_i^+ = T_i \cdot T_{i-1}$$

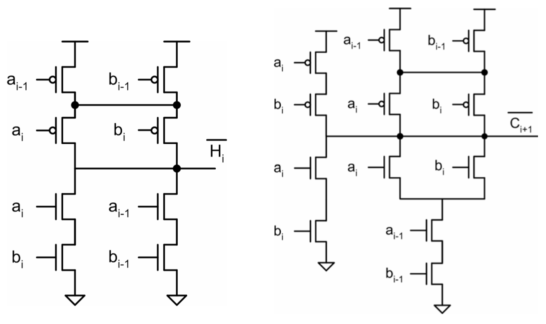
Doran noted that this formulation can be achieved in two forms, using either t_i or p_i to calculate T_i. Due to the increased complexity of computing X_i at each level of the recurrence compared with Ling's and Weinberger's recurrence, the two forms of Doran's transformation are not suitable for efficient CMOS realization.

5. Considerations for CMOS Realization

Technology characteristics impose limits on realizations of Weinberger's and Ling's recurrences for addition. The primary constraint in current CMOS technology is transistor stack height, which is commonly limited to between 2 and 5 for nMOS stacks and 2 for pMOS stacks. In addition energy has become as important as performance, requiring that structures be analyzed over a range of operating points [14]. Several realization techniques have been developed to efficiently map recurrence algorithms to CMOS technology under these constraints [14].

5.1. Combined Bit-Operator and 1st Carry Stage

In dynamic adder implementations, one logic stage can be removed by combining the computation of *g* and *t* into the first prefix computation stage [4, 5]. This technique is more favorable to Ling's recurrence than to Weinberger's. Under the same stack height constraint a Ling realization can combine more bits in the first stage than a realization using Weinberger's recurrence. The first recurrence block with combined bit operators for each recurrence is shown in Fig 6.



$$\overline{H_i} = \overline{a_i b_i + a_{i-1} b_{i-1}} \qquad \overline{C_{i+1}} = \overline{a_i \cdot b_i + (a_i + b_i)(a_{i-1} b_{i-1})}$$

Figure 6. Combined first stage static CMOS prefix 2 realization of $\overline{H_i}$ and $\overline{C_{i+1}}$

By combining the *g* and *t* operators with the first prefix block of the carry tree, the resulting logic for Ling's transformation uses a CMOS logic block containing one less transistor in the nMOS stack for H_i than Weinberger's recurrence uses for C_{i+1}. However, subsequent blocks have the same stack height since the recursion is performed using either *H* and *T* or *G* and *T* with the same prefix operator "●".

5.2. Conditional Computation of Sum

Mathew [6] made use of conditional logic for the computation of sum in an attempt to reduce energy. The use of 4-bit conditional logic allowed for only 1 in 4 carries to be computed, reducing the size of the carry tree. Additionally the conditional logic was implemented using static gates, which allowed for the switching activity of the gates to be reduced. Both Weinberger's and Ling's recurrences map efficiently to conditional computation of sum, with C_i and H_{i-1} used as the select signal respectively. To take full advantage of conditional computation, the conditional sum path must have fewer stages than the recurrence path. As the difference between the number of stages in the recurrence tree path and the conditional computation path increases, there exists more potential for reducing energy through gate sizing [14]. However, as the difference between the number of stages in the recurrence tree and the conditional path decreases, the energy consumed by the conditional path increases and the possibility of the conditional sum becoming the critical path increases. The optimal number of bits to be computed conditionally and the realization of the conditional computation are determined by technology and adder recurrence structure. The conditional computation can be performed either by rippling or through the use of separate recurrence trees.

6. Proposed Realizations

We propose two 64-bit dynamic adder realizations which utilize Ling's recurrence combined with an energy efficient realization that takes full advantage of compound-domino logic, which we found to be the most energy-delay efficient for CMOS implementation [14].

6.1. A Three Stage Ling Adder (TSL)

We developed a three stage 64-bit adder by using a fully parallel prefix tree with Ling's transformation. Under the technology limitation for dynamic gates of a stack with no more than 5 nMOS transistors, a prefix-4 CMOS block can be used in the first dynamic gate for the recurrence. Using compound domino logic the static recurrence gates are implemented using prefix-2. The proposed full parallel prefix tree with prefix 4, 2, 4, and 2 for the first, second, third and forth blocks respectively is shown in Fig. 7.

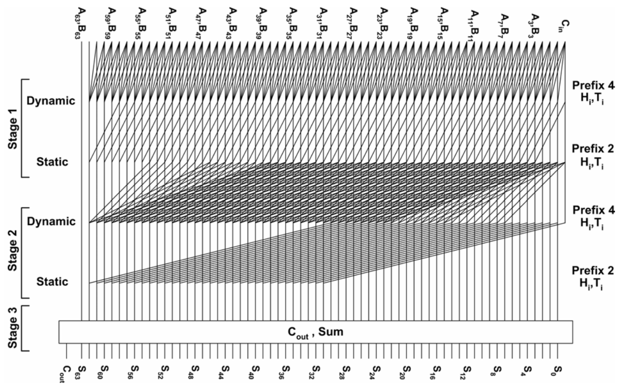


Figure 7. 64-bit Three Stage parallel prefix Ling adder (TSL)

The equations for the first level H_i and T_{i-1} are:

$$H_i = a_i b_i + a_{i-1}b_{i-1} + (a_{i-1}+b_{i-1})a_{i-2}b_{i-2} + (a_{i-1}+b_{i-1})(a_{i-2}+b_{i-2})a_{i-3}b_{i-3}$$

$$T_{i-1} = (a_{i-1}+b_{i-1})(a_{i-2}+b_{i-2})(a_{i-3}+b_{i-3})(a_{i-4}+b_{i-4})$$

The result is a worst case stack height of 4 nMOS transistors for both equations. However, in the first stage of a dynamic adder both gates must be footed, which increases the worst case stack height to the technology limit of 5. The second, third and fourth level H_i and T_i computations follow traditional dot product operations for prefix 2 and prefix 4 and do not violate the stack height limitations. C_{out} is computed conditionally and is selected in the same stage as S_{63} using H_{62}.

6.2. Energy-Efficient Three Stage Conditional Sum Ling Adder (CSL)

The speed and energy consumption of an adder can be greatly influenced by the amount of wire it uses. This wire impact can easily offset any advantage obtained by using a more efficient recurrence. We reduced the amount of wiring and gates in our proposed adder by generating every other H_i without increasing the number of stages (Fig. 8). This was achieved by conditionally computing the two-bit sum and selecting each group with the corresponding H_i. The equations for S_i and S_{i+1} are:

$$S_i = \begin{cases} a_i \oplus b_i & if\ H_{i-1} = 0 \\ a_i \oplus b_i \oplus t_{i-1} & if\ H_{i-1} = 1 \end{cases}$$

$$S_{i+1} = \begin{cases} a_{i+1} \oplus b_{i+1} \oplus g_i & if\ H_{i-1} = 0 \\ a_{i+1} \oplus b_{i+1} \oplus (g_i + t_i t_{i-1}) & if\ H_{i-1} = 1 \end{cases}$$

The number of bits for conditional sum was chosen such that the critical path of the conditional sum did not exceed the delay of the recurrence path.

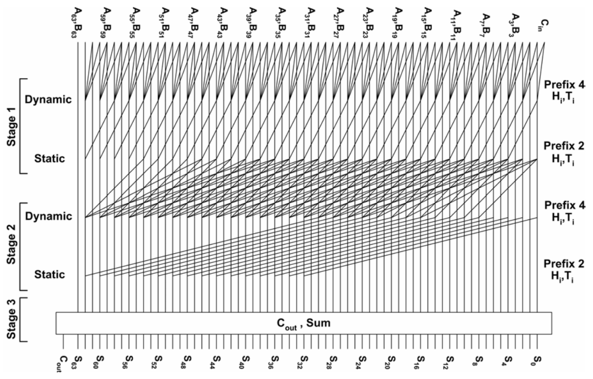


Figure 8. 64-bit Three Stage Conditional Sum Ling adder (CSL)

7. Results

All of our results were obtained using estimates in 130nm technology by applying the energy-delay estimation method developed in [15] to the entire adder. The technology parameters are: L_{eff} = 110nm, nominal V_{dd} = 1.2V. Characterization of the technology was performed for the typical process corner at a temperature of 25C.

A comparison of 32-bit static adder implementations using Weinberger's recurrence and Ling's transformation is shown in Fig. 9. Each adder is delay optimized for a path gain (C_{out}/C_{in}) of 4.

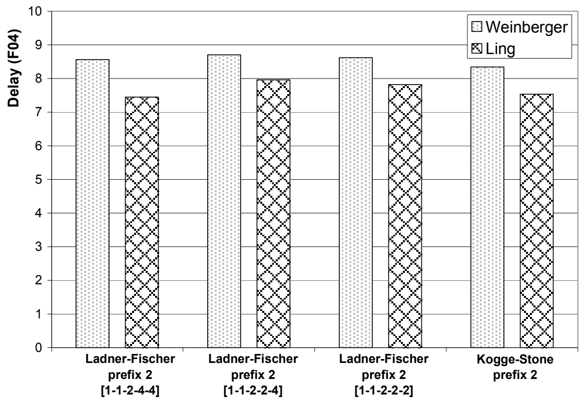


Figure 9. Comparison of 32-bit static Weinberger and Ling adders

Ling's transformation yields an improvement in delay of up to 12%, confirming the benefit that Ling can achieve in static adder implementations limited to a stack of 2 pMOS and 2 nMOS transistors.

7.1. Impact of Conditional Sum

A comparison of 64-bit dynamic adders with and without conditional sum is shown in Fig. 10. Inter-stage wire lengths are estimated using a 4μm bit pitch. Each adder is sized to achieve optimal performance for a given path gain (C_{out}/C_{in}). The output of each adder is attached to a 1mm wire, and the input size is varied to obtain path gains of 2 through 8. The length of the wire attached to the output is representative of the loading of an adder in a typical high-performance ALU configuration [6]. The resulting points create an energy-delay curve for each design representative of various high-performance operating conditions. For each design a switching activity of 50% was applied to the dynamic recurrence path and a 15% switching activity was applied to the static gates on the conditional sum path. The switching factors are based on experimental observations in industry [14].

The results show an energy savings for the 2-bit conditional sum variants. This is primarily due to the reduced switching activity of the conditional path. In the 6 stage recurrence tree of the fully parallel prefix-2 Ling, applying a 2-bit conditional sum improves energy at a slight increase in delay. The delay penalty is a result of increased loading on the adder input caused by the static gates of the conditional sum path. The CSL design results in improved performance and a slight energy savings compared to the TSL design. The performance improvement is due to the reduced input loading associated with the first gates of the conditional sum path relative to the first prefix-4 gates of the recurrence path.

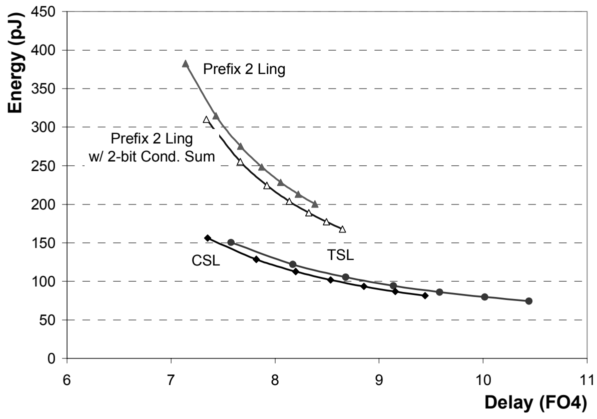


Figure 10. Impact of conditional sum in high-performance 64-bit dynamic adders

7.2. Impact of Wiring

The proposed TSL and CSL adders are shown in Fig. 11, with and without wiring, in comparison to the fastest Kogge-Stone implementation by Park [5], and a fully parallel prefix 2 Ling adder to demonstrate the impact of wiring on high-performance adders.

The inclusion of wire causes each design to run slower, however the impact of wire on CSL is less than Park and TSL. Additionally each design requires more energy to obtain the same performance. This increase in energy for the same performance is the smallest for CSL. Without wiring TSL obtains better delay and energy than the CSL design. However, with wire included, CSL obtains better delay and energy compared to the TSL design. This confirms the benefit obtained from the techniques used in CSL.

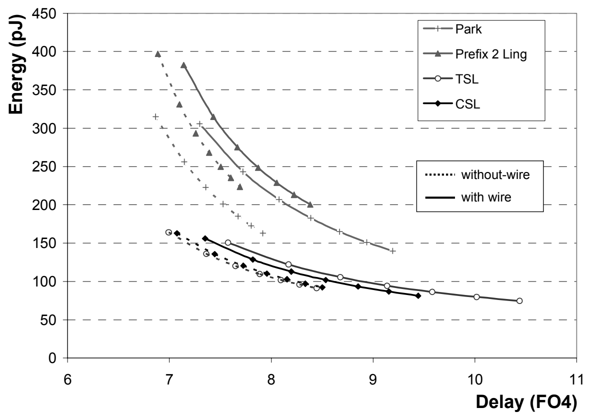


Figure 11. Impact of wiring in high-performance 64-bit dynamic adders

7.3. 64-bit Dynamic Adder Comparison

A comparison of TSL, CSL and the best published realizations of 64-bit dynamic adders is shown in Fig. 12. The best recurrence based adder realization is a three stage Kogge-Stone implementation proposed by Park [5]. The best Ling realization is Naffziger's six stage implementation, which utilizes a single stage for the 4-bit group *H* and *T*, followed by a four stage ripple carry, and finally a single stage sum select.

The designs which have the least number of stages (CSL, TSL and Park's) are the most energy-efficient. The reduced energy is a result of the decreased number of stages in the design, which allows for the same delay to be achieved while using a greater fan-out per stage. The energy reduction and performance improvement of these designs is limited due to the increased branching and gate complexity. The fully parallel prefix 2 adder is able to achieve high performance due to its balancing of branching and redundancy with the number of stages. However, this comes at a substantial cost in energy. The increased number of stages results in a smaller fanout per stage requiring twice the amount of energy to maintain the same performance as the proposed CSL design.

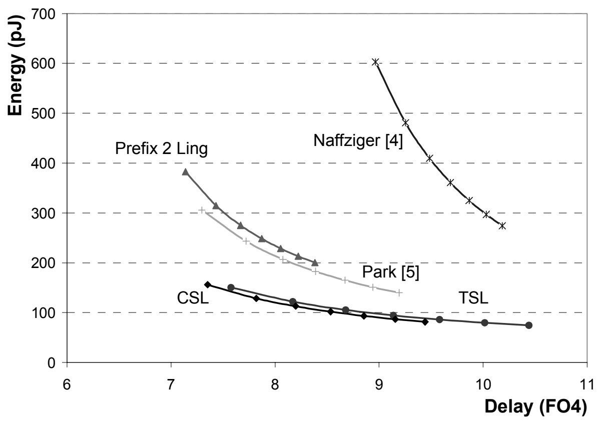


Figure 12. Energy-Delay Space comparison of CSL, TSL and the best 64-bit Dynamic Adders

The realizations proposed in this paper demonstrate better performance than the best published implementations of Weinberger and Ling adders. The best delay is obtained by the fully parallel prefix 2 design, Park's Kogge-Stone design and CSL. The most energy-efficient design is the proposed CSL adder which resulted in a savings of up to 50% in energy for the same delay or 1 FO4 delay improvement for the same energy versus the fastest published realization of Kogge-Stone by Park [5], and a 2 FO4 delay improvement and 3x energy reduction versus the best published realizations of Ling by Naffziger [4].

8. Conclusion

Ling's and Weinberger's recurrence algorithms for addition demonstrate favorable characteristics for efficient CMOS realization. For high-performance dynamic adders Ling shows a fundamental advantage in CMOS by reducing the complexity of the first stage of the recurrence tree. We have demonstrated that existing recurrence trees based on Weinberger's recurrence can be applied directly to Ling's transformation with only a modification of the first stage and sum computation. For static adders Ling's transformation demonstrated up to 12% delay improvement versus adders using comparable recurrence trees with Weinberger's recurrence. Efficient realizations of Ling's transformation are presented for both: prefix selection for the best use of compound-domino in successive levels of recurrence and optimal conditional sum computation size. The proposed CSL adder demonstrates 50% savings in energy at the same delay and 1 FO4 delay improvement at the same energy when compared to the fastest published designs.

Acknowledgements

This work has been supported by SRC Research Grant No. 931.001 and California MICRO 01-063.

References

[1] A. Weinberger, J.L. Smith, " A Logic for High-Speed Addition," Nat. Bur. Stand.. Circ., 591:3-12, 1958.

[2] H. Ling, "High-Speed Binary Adder," IBM Journal of Research and Development, vol. 25, no.3, pp. 156-166, May 1981.

[3] R. W. Doran, "Variants of an Improved Carry Look-Ahead Adder," IEEE Transactions on Computers, Vol. 37, No.9, Sept. 1988.

[4] S. Naffziger, "A Sub-Nanosecond 0.5μm 64-b Adder Design", 1996 IEEE International Solid-State Circuits Conference, Digest of Technical Papers, Feb. 1996, pp.362-363.

[5] J. Park, et. al., "470ps 64-Bit Parallel Binary Adder", 2000 Symposium on VLSI Circuits Digest of Technical Papers.

[6] S. K. Mathew et al, "A 4GHz 130nm Address Generation Unit with 32-bit Sparse-tree Adder Core," 2002 Symposium on VLSI Circuits Digest of Technical Papers, pp.126-127.

[7] P.M. Kogge and H.S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations", IEEE Trans. Computers Vol. C-22, No. 8, Aug. 1973, pp.786-793.

[8] T. Han, D.A. Carlson, "Fast Area-Efficient VLSI Adders," 8th IEEE Symposium on Computer Arithmetic, Como, Italy, pp.49-56, May 1987.

[9] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," IEEE Transactions on Computers, C-31(3), March 1982.

[10] R. E. Ladner, M.H. Fischer, "Parallel Prefix Computation," JACM, 27(4):831-838, Oct. 1980.

[11] S. Knowles, "A Family of Adders," 14th IEEE Symposium on Computer Arithmetic, Adelaide, Australia, April 14th-16th, 1999.

[12] J. Sklanski, "Conditional-Sum Addition Logic," IRE Trans. on Electronic Computers, Vol. EC-9, No2, pp.226-231, 1960.

[13] O. J. Bedrij, "Carry-Select Adder," IRE Trans. on Electronic Computers, Vol. EC-11, pp. 340-346, 1962.

[14] V. G. Oklobzija, B. R. Zeydel, H. Dao, S. Mathew, R. Krishnamurthy, "Energy-Delay Estimation Technique for High-Performance Microprocessor VLSI Adders," 16th IEEE Symposium on Computer Arithmetic, Santiago de Compostela, Spain, June 15-18th 2003.

[15] V. G. Oklobdzija, B. R. Zeydel, H. Q. Dao, S. Mathew, R. Krishnamurthy, "Comparison of High-Performance VLSI Adders in Energy-Delay Space", IEEE Transaction on VLSI Systems, in press, 2005