Techniques for Fast CMOS-based Conditional Sum Adders

Hans Lindkvist and Per Andersson

Department of Computer Engineering, Lund University, Sweden

Abstract

Conditional Sum Adders, CSAs, and Carry-Lookahead Adders, CLAs, both have logarithmic gate depth. However, CLAs require a final add stage while CSAs produce the sum bits in parallel with the final carry bit. For CMOS implementations, the depth advantage of CSA has been difficult to exploit since the traditional structure of CSAs have some heavily loaded internal nodes.
In this report we show that the CSA-operation forms a monoid and that all circuit structures, corresponding to parallel prefix algorithms, used with CLA to reduce internal fan-out, are applicable also to CSAs. Furthermore, we show that all time critical computations in a CSA can be performed with monotone functions which allow efficient dynamic CMOS logic to be used. Finally we evaluate a variety of transistor level adder implementations with respect to speed and we show that in almost all cases the CSA has lower delay than its CLA counterpart.

1. Introduction

Binary adders are basic building-blocks for a very wide range of high performance bit-parallel applications, and they are often in the critical delay path in effect limiting the total system performance. The key to optimizing adder performance is to find the best combination of logic structure and circuit technique.

The dominant technology of today, silicon CMOS, is characterized by:

- High area efficiency
- Relatively low intrinsic gate delay for small gates but rapidly growing with number of inputs (when series transistors are required)
- Low drive capability, leading to long delays for nets with large fan-out and/or wiring capacitance
- Various forms of dynamic logic are useful to improve both speed and area

From a performance point of view this favours logic that is structured as trees of small gates when many inputs must be combined to produce one output, e.g., to produce the most significant output of an adder. Furthermore gate output load (the sum of fan-out and wire capacitance) is just as important for the total adder delay as the gate depth. In addition, the possibility to use dynamic logic can have a large speed impact. However, the full advantage of dynamic logic is available only when it is used to implement monotone switch functions.

The techniques used in the distant-carry adder family, i.e., the Conditional-Sum Adder (here abbreviated CSA) [1], the Carry-Select Adder [2] and the Carry-Lookahead Adder (CLA)[3][5] all have the advantage of logarithmic gate depth and these techniques are the basis for virtually all aggressive adder designs. When the logic is properly structured and implemented with good circuit techniques, impressive speed can be achieved even with moderate technology. Examples of fast CMOS adders include the 56-bit adder used in the Advanced Micro Devices Am29050, performing well under 4ns [19] [20] which was improved by V. Kantabutra [21] to perform below 2 ns in a 1.0μm CMOS process, the integer adder in the 275MHz, CMOS, DECchip 21064 which is an hybrid of CLA with dynamic logic and a Conditional-Sum tree [22], and a very aggressive 64-bit adder implemented in 0.8μm CMOS with a 2ns add time [17] [18]. The last example uses a novel dynamic CMOS circuit technique [16] to implement a carry-lookahead-block which produces all 64 carry bits with only low fan-out gates.

The basic difference between the CSA and the CLA scheme is that CSA produces the sum bits directly with a minimum logarithmic gate depth whereas CLA produces only the carry signals with equal depth and less logic. For CLA, an additional level of half-adders is required to produce the sum. CSAs are thus potentially faster than their CLA counterparts.

In spite of its lower depth, the way CSA is normally implemented gives it a speed disadvantage compared to CLA. The main reasons are fan-out problems (exponentially growing load on some gates towards the final stages of the adder) and less effective circuit technique due to the use of multiplexers rather than simpler gates.

In this paper we show that virtually all the methods used in CLAs to decrease fan-out and to improve logic speed with special circuit techniques are applicable also to CSA design. We show that all parallel prefix algorithms used to restructure CLAs, e.g., the scheme by Brent and Kung [3], can be used with CSA. We demonstrate that all multiplexers related to the carry information can be realized with

1063-6404/94 $4.00 © 1994 IEEE

the generate function typically used in CLAs instead of multiplexers and that dynamic circuit techniques can be used since these functions are monotone.

Finally we make a comparison between CSA and CLA based adders for different structures and widths. The comparison is based on simulation of transistor level implementations in a standard CMOS technology.

2. Taxonomy for distant-carry adders

Carry lookahead addition can be described as a three stage calculation. First a preprocessing stage, second a carry calculation stage (the lookahead) and last a summation stage. In the first stage bit-sum and carry-information signals are produced. Usually the carry-information signals are bit-generate and bit propagate, but alternatives exists, e.g. conditional carry calculation. The second stage is a prefix circuit using an operation which is usually the generate-propagate operation (GP-operation). This stage will efficiently produce all signals corresponding to the intermediate carry signals in a ripple carry adder. Once all these carry signals are produced they will be used as carry-input to the closest higher addition. Note that neither the algorithm used to produce the carry-signals nor the operation used in the circuit is defined. It has been shown that the carry-signals in carry-lookahead addition can be produced using any prefix algorithm [8].

Conditional-sum addition is normally associated with the structure developed by Sklansky [1]. We consider that misleading. Instead we generalize the term conditional-sum to describe the set of recurrence equations described by Sklansky [1], organized according to any correctly behaving structure. Such adders produce all sums and all carries simultaneously. Normally we are not interested in all the carry-outputs, which makes it possible to reduce the circuit somewhat.

3. Definitions

The operation which is the most commonly used for adding with prefix algorithms is the Generate-Propagate operation (GP-operation).

Let $a_{n-1}a_{n-2}...a_0$ and $b_{n-1}b_{n-2}...b_0$ be the n-bit binary numbers used as operands in a binary addition

Definition 1: The bit-generate, bit-propagate and bit-sum are defined by

$$g_i = a_i \wedge b_i$$
$$p_i = a_i \vee b_i$$
$$s_i = a_i \oplus b_i$$

Definition 2: The GP-operation "•" is defined by

$$\langle g_B, p_B \rangle = \langle g_Z, p_Z \rangle \bullet \langle g_A, p_A \rangle$$

where

$$g_B = g_A \vee p_A g_Z$$
$$p_B = p_A \wedge p_Z$$

Definition 3: The group GP-operation is defined by

$$\langle G_{k:i}, P_{k:i} \rangle = \begin{cases} \langle g_i, p_i \rangle & \text{;if } k = i \\ \langle G_{k:j}, P_{k:j} \rangle \bullet \langle G_{j:i}, P_{j:i} \rangle & \text{;if } k \neq i \end{cases}$$
$$\text{where } k \leq j \leq i$$

where k:i denotes a block ranging from bit k to bit i.

When $G_{0:i}$ for all $i<n$ have been computed, the sum-bits $S_{0:i}$ are formed in an additional step.

$$S_{0:i} = \begin{cases} s_0 & \text{;if } i = 0 \\ s_i \oplus G_{0:i-1} & \text{;if } 0 < i < n \end{cases}$$

Similar definitions can be made for the Conditional-Sum operation (CSA-operation).

Definition 4: The bit-sums and bit-carries are defined as

$$s_i^0 = a_i \oplus b_i$$
$$s_i^1 = \overline{a_i \oplus b_i}$$
$$c_i^0 = a_i \wedge b_i$$
$$c_i^1 = a_i \vee b_i$$

Definition 5: The CSA-operation "◊" is now defined by

$$\langle c_B^0, c_B^1, s_B^0, s_B^1 \rangle = \langle c_Z^0, c_Z^1, s_Z^0, s_Z^1 \rangle \lozenge \langle c_A^0, c_A^1, s_A^0, s_A^1 \rangle$$

where

$$c_B^0 = c_A^0 c_Z^{\overline{0}} \vee c_A^1 c_Z^0$$
$$c_B^1 = c_A^0 c_Z^{\overline{1}} \vee c_A^1 c_Z^1$$
$$s_B^0 = s_A^0 c_Z^{\overline{0}} \vee s_A^1 c_Z^0$$
$$s_B^1 = s_A^0 c_Z^{\overline{1}} \vee s_A^1 c_Z^1$$

Definition 6: The Group CSA-operation is defined by

$$\langle C^0_{k:i}, C^1_{k:i}, S^0_{k:i}, S^1_{k:i}\rangle$$

$$= \begin{cases} \langle c^0_i, c^1_i, s^0_i, s^1_i\rangle \text{ ;if } k = i \\ \langle C^0_{k:j}, C^1_{k:j}, S^0_{k:j}, S^1_{k:j}\rangle \lozenge \langle C^0_{j:i}, C^1_{j:i}, S^0_{j:i}, S^1_{j:i}\rangle \text{ ;if } k \neq i \end{cases}$$

where $k \leq j \leq i$

where k:i denotes a block ranging from bit k to bit i.

When $S_{0:i}$ for all $i<n$ have been produced these form the sum.

We proceed by stating the common definitions of the monoid, the parallel prefix algorithm and the parallel prefix circuit.

Definition 7: A monoid is a pair $\langle S, \circ\rangle$ where $\circ$ is a binary operation on the non empty set S such that for every a, b and c in S the following conditions hold:

1. $a \circ b \in S$ (closed)
2. $[a \circ b] \circ c = a \circ [b \circ c]$ (associative)
3. There is an element e in S such that for every a in S: $e \circ a = a \circ e = a$ (neutral element)

Definition 8: A parallel prefix algorithm is an algorithm that takes n inputs $x_1, x_2, \ldots, x_n$ and in parallel produces the n outputs $x_1, x_1 \circ x_2, \ldots, x_1 \circ x_2 \circ \ldots \circ x_n$ where $x \in S$ and $\langle S, \circ\rangle$ is a monoid.

Definition 9: A parallel prefix circuit is a combinational circuit that takes n inputs $x_1, x_2, \ldots, x_n$ and in parallel produces the n outputs $x_1, x_1 \circ x_2, \ldots, x_1 \circ x_2 \circ \ldots \circ x_n$ where $x \in S$ and $\langle S, \circ\rangle$ is a monoid.

The dataflow graph of a parallel prefix algorithm directly corresponds to the structure of parallel prefix circuit.

Example: The GP-operation has previously been shown to be a monoid [3] [5] [8] and can therefore be used together with a prefix algorithm to produce all carry signals.

If this is applied to adder design we obtain a parallel prefix circuit such as the one used by Brent and Kung.

4. Conditional-Sum addition as a prefix problem

We will now prove that the CSA-operation is a monoid and therefore can be used together with parallel prefix algorithms to form parallel prefix circuits.

Theorem 1: Let $S = \langle a_1, a_2, a_3, a_4\rangle ; a_1, a_2, a_3, a_4 \in 0, 1$ then $\langle S, \lozenge\rangle$ is a monoid.

Proof:

1. Check that $a \lozenge b \in S$

S contains the combinations:

$\langle a_1, a_2, a_3, a_4\rangle ; a_1, a_2, a_3, a_4 \in 0, 1$

Thus all possible combinations of four binary variables are allowed and the operation cannot extend the set S.

2. Check that $[a \lozenge b] \lozenge c = a \lozenge [b \lozenge c]$ by expanding to disjunctive normal form, DNF

$$\begin{aligned}
&\left[\langle C^0_Z, C^1_Z, S^0_Z, S^1_Z\rangle \lozenge \langle C^0_A, C^1_A, S^0_A, S^1_A\rangle\right] \lozenge \langle C^0_C, C^1_C, S^0_C, S^1_C\rangle \\
&= \langle \overline{\left(\overline{C^0_Z}C^0_A \vee C^0_Z C^1_A\right)}C^0_C \vee \left(\overline{C^0_Z}C^0_A \vee C^0_Z C^1_A\right)C^1_C \\
&\quad , \left(\overline{C^1_Z}C^0_A \vee C^1_Z C^1_A\right)C^0_C \vee \left(\overline{C^1_Z}C^0_A \vee C^1_Z C^1_A\right)C^1_C \\
&\quad , \overline{\left(\overline{C^0_Z}C^0_A \vee C^0_Z C^1_A\right)}S^0_C \vee \left(\overline{C^0_Z}C^0_A \vee C^0_Z C^1_A\right)S^1_C \\
&\quad , \overline{\left(\overline{C^1_Z}C^0_A \vee C^1_Z C^1_A\right)}S^0_C \vee \left(\overline{C^1_Z}C^0_A \vee C^1_Z C^1_A\right)S^1_C\rangle \\
&= \langle C^0_Z\overline{C^1_A}C^0_C \vee \overline{C^0_Z}\,\overline{C^0_A}C^0_C \vee \overline{C^0_Z}C^0_A C^1_C \vee C^0_Z C^1_A C^1_C \\
&\quad , C^1_Z\overline{C^1_A}C^0_C \vee \overline{C^1_Z}\,\overline{C^0_A}C^0_C \vee \overline{C^1_Z}C^0_A C^1_C \vee C^1_Z C^1_A C^1_C \\
&\quad , C^0_Z\overline{C^1_A}S^0_C \vee \overline{C^0_Z}\,\overline{C^0_A}S^0_C \vee \overline{C^0_Z}C^0_A S^1_C \vee C^0_Z C^1_A S^1_C \\
&\quad , C^1_Z\overline{C^1_A}S^0_C \vee \overline{C^1_Z}\,\overline{C^0_A}S^0_C \vee \overline{C^1_Z}C^0_A S^1_C \vee C^1_Z C^1_A S^1_C\rangle
\end{aligned}$$

and

$$\begin{aligned}
&\langle C^0_Z, C^1_Z, S^0_Z, S^1_Z\rangle \lozenge \left[\langle C^0_A, C^1_A, S^0_A, S^1_A\rangle \lozenge \langle C^0_C, C^1_C, S^0_C, S^1_C\rangle\right] \\
&= \langle \overline{C^0_Z}\left(\overline{C^0_A}C^0_C \vee C^0_A C^1_C\right) \vee C^0_Z\left(\overline{C^1_A}C^0_C \vee C^1_A C^1_C\right) \\
&\quad , \overline{C^1_Z}\left(\overline{C^0_A}C^0_C \vee C^0_A C^1_C\right) \vee C^1_Z\left(\overline{C^1_A}C^0_C \vee C^1_A C^1_C\right) \\
&\quad , \overline{C^0_Z}\left(\overline{C^0_A}S^0_C \vee C^0_A S^1_C\right) \vee C^0_Z\left(\overline{C^1_A}S^0_C \vee C^1_A S^1_C\right) \\
&\quad , \overline{C^1_Z}\left(\overline{C^0_A}S^0_C \vee C^0_A S^1_C\right) \vee C^1_Z\left(\overline{C^1_A}S^0_C \vee C^1_A S^1_C\right)\rangle \\
&= \langle C^0_Z\overline{C^1_A}C^0_C \vee \overline{C^0_Z}\,\overline{C^0_A}C^0_C \vee \overline{C^0_Z}C^0_A C^1_C \vee C^0_Z C^1_A C^1_C \\
&\quad , C^1_Z\overline{C^1_A}C^0_C \vee \overline{C^1_Z}\,\overline{C^0_A}C^0_C \vee \overline{C^1_Z}C^0_A C^1_C \vee C^1_Z C^1_A C^1_C \\
&\quad , C^0_Z\overline{C^1_A}S^0_C \vee \overline{C^0_Z}\,\overline{C^0_A}S^0_C \vee \overline{C^0_Z}C^0_A S^1_C \vee C^0_Z C^1_A S^1_C \\
&\quad , C^1_Z\overline{C^1_A}S^0_C \vee \overline{C^1_Z}\,\overline{C^0_A}S^0_C \vee \overline{C^1_Z}C^0_A S^1_C \vee C^1_Z C^1_A S^1_C\rangle
\end{aligned}$$

Thus

$$\left[\langle C_C^0, C_C^1, S_C^0, S_C^1\rangle \lozenge \langle C_A^0, C_A^1, S_A^0, S_A^1\rangle\right]$$
$$\lozenge \langle C_Z^0, C_Z^1, S_Z^0, S_Z^1\rangle = \langle C_C^0, C_C^1, S_C^0, S_C^1\rangle$$
$$\lozenge \left[\langle C_A^0, C_A^1, S_A^0, S_A^1\rangle \lozenge \langle C_Z^0, C_Z^1, S_Z^0, S_Z^1\rangle\right]$$

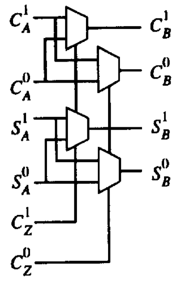
3. For the element $e = \langle 0, 1, 0, 1\rangle \in S$ we note that
$$\langle 0, 1, 0, 1\rangle \lozenge \langle a_1, a_2, a_3, a_4\rangle = \langle a_1, a_2, a_3, a_4\rangle$$
$$(a_1, a_2, a_3, a_4) \lozenge \langle 0, 1, 0, 1\rangle = \langle a_1, a_2, a_3, a_4\rangle$$

❑

5. Reduction of Conditional-Carry generation

The CSA-operation is typically implemented with multiplexers according to figure 1.

Figure 1: Typical implementation of the CSA-operation



We note that

1. C_B is independent of S_A and S_Z
2. S_B is independent of S_Z

The first point makes it possible to consider the carry-calculation separately. Hereinafter we will refer to the carry calculation used in Conditional-Sum as Conditional-Carry. Furthermore, the second point indicates that S_B will be less loaded than C_B - which controls both C_A and C_Z in the following stage - and that optimization should focus on the carry calculation.

Theorem 2: $C^0\overline{C^1} \neq 1$

Proof: From the expressions in definition 4 we directly obtain that $c^0 \subset c^1$ which leads to $c^0\overline{c^1} \neq 1$. We state the truth-table for the carry-part of the CSA-operation to verify that this property is retained under the group-operation. The output combination $\langle 1, 0\rangle$ does not appear in the table.

$\langle C_A^0, C_A^1\rangle$	$\langle C_Z^0, C_Z^1\rangle$	$\langle C_B^0, C_B^1\rangle$
⟨0, 0⟩	⟨0, 0⟩	⟨0, 0⟩
⟨0, 0⟩	⟨0, 1⟩	⟨0, 0⟩
⟨0, 0⟩	⟨1, 1⟩	⟨0, 0⟩
⟨0, 1⟩	⟨0, 0⟩	⟨0, 0⟩
⟨0, 1⟩	⟨0, 1⟩	⟨0, 1⟩
⟨0, 1⟩	⟨1, 1⟩	⟨1, 1⟩
⟨1, 1⟩	⟨0, 0⟩	⟨1, 1⟩
⟨1, 1⟩	⟨0, 1⟩	⟨1, 1⟩
⟨1, 1⟩	⟨1, 1⟩	⟨1, 1⟩

❑

This leads to the following theorem:

Theorem 3: For the carry component, $\langle C_B^0, C_B^1\rangle$, of the CSA-operation it holds that

$$C_B^0 = C_A^0 \vee C_A^1 C_Z^0$$
$$C_B^1 = C_A^0 \vee C_A^1 C_Z^1$$

Proof: Verify that equations $C_B^0 = C_A^0 \vee C_A^1 C_Z^0$ and $C_B^1 = C_A^0 \vee C_A^1 C_Z^1$ satisfy the table in the proof of theorem 2.

❑

This simplification eliminates the need for an inverter in each Conditional-Carry multiplexer. The multiplexers for sum calculation cannot be reduced, but the normal situation is that the carry calculation is the critical part of the adder. Using the reduced carry-equations we now redefine the CSA-operation. The question whether this reduced operation is still a monoid arises.

Definition 10: The Reduced Conditional-Sum operation (CSA-operation) is defined as

$$\langle c_B^0, c_B^1, s_B^0, s_B^1\rangle = \langle c_Z^0, c_Z^1, s_Z^0, s_Z^1\rangle \lozenge \langle c_A^0, c_A^1, s_A^0, s_A^1\rangle$$

where

$$c_B^0 = c_A^0 \vee c_A^1 c_Z^0$$
$$c_B^1 = c_A^0 \vee c_A^1 c_Z^1$$
$$s_B^0 = s_A^0 \overline{c_Z^0} \vee s_A^1 c_Z^0$$
$$s_B^1 = s_A^0 \overline{c_Z^1} \vee s_A^1 c_Z^1$$

Theorem 4: Let

$S = \langle a_1, a_2, a_3, a_4 \rangle; \quad a_1, a_2, a_3, a_4 \in 0, 1 \quad \wedge \quad a_1 \overline{a_2} \neq 1$

then $\langle S, \Diamond \rangle$ is a monoid.

Proof:

1. Check that $a \Diamond b \in S$

From the table in the proof of theorem 2 we see that removing the elements which corresponds to $\overline{C^0} C^1 \neq 1$ from the set of input elements also removes them from the output set. Thus S is still closed over $\Diamond$.

2. Check that $[a \Diamond b] \Diamond c = a \Diamond [b \Diamond c]$

Removing the element $\langle 1, 0, 1, 0 \rangle$ from S in theorem 1 does not affect the associativity. It is still associative. Then to thereafter reduce the operation in definition 5 to the one in definition 10 does not affect the behaviour over the reduced set S. Thus the operation is still associative.

3. We note that the element $e = \langle 0, 1, 0, 1 \rangle$ still belongs to S.

❑

The reduced Conditional-Carry operation is similar to the Generate function in the GP-operation. In fact the relation between the Conditional-<carry operation and the GP-operation is quite simple:

$$C^0 = G$$
$$C^1 = G \vee P$$

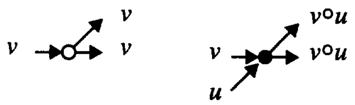
Reducing the Conditional-Carry operation this way results in a monotone function. The main advantages for implementation are that fewer inverters is needed and that dynamic circuit techniques can be used.

6. Prefix algorithms

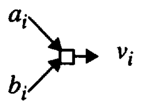
There are a large variety of algorithms that solve the prefix problem. Their qualities for use in adding are largely dependant on properties of their graph representation, e.g., out-order for a node which corresponds to fan-out, amount of communication which corresponds to amount of wires, and number of nodes which corresponds to device-count. The most commonly used algorithms are the ones used in Sklansky's adder [1] and the one used by Brent and Kung [3]. The one Sklansky used has the advantage of optimal depth but has an exponentially increasing fan-out towards the final stages. Brent and Kung's algorithm on the other hand, has limited and low fan-out but has almost double depth compared to the one Sklansky used [1]. There is however an algorithm, invented by Kogge and Stone [4] as a solution to a larger class of problems, which has both optimal depth and low and limited fan-out. The price to pay for both low depth and fan-out is increased complexity and communication. The Hybrid Prefix Computation algorithm invented by Han and Carlson [5] combines Brent and Kung's algorithm with Kogge and Stone's. This increases the depth somewhat but decreases the communication.

There are many papers on the practical and theoretical aspects of prefix algorithms and on the generation of prefix algorithms [4] [5] [6] [7].

We start with some definitions adopted from Brent and Kung [3] namely the black and the white operator. The white operator copies the input to the output(s) while the black operator performs the Conditional-Sum or the Generate-Propagate operation, depending on the method used.



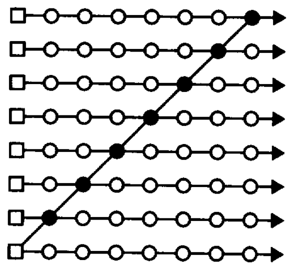
Furthermore we extend their notation with the square.



The vector v_i is either $\langle c_i^0, c_i^1, s_i^0, s_i^1 \rangle$ if the CSA-operation is used or $\langle g_i, p_i \rangle$ if the GP-operation is used. In the following representations of graph algorithms the a_i and b_i arcs are suppressed.

For reference we start with a serial prefix algorithm corresponding to ripple-carry addition. It has the lowest area and low inter-bit communication, but it has a large delay.

Figure 2: Serial prefix algorithm

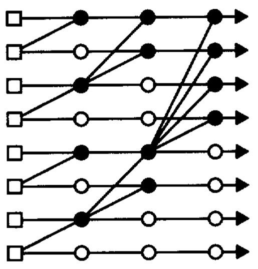


As indicated above many of the algorithm properties are apparent in the graph representations. All arcs are implicitly directed to the right. The number of columns are the number of steps needed to perform the calculation and can be interpreted as delay or number of pipeline steps in a realization of the algorithm. The share of black operations in a column is the degree of parallelism in that step. The out-degree of an operation node will determine the fan-out of the corresponding sub-circuit in the realization and will together with the number of columns have a large impact on the resulting delay.

The most common parallel prefix algorithm is the one used by Sklansky for conditional-sum addition [1], by many others for carry-lookahead addition [9] [10] and by Kelliher et al. for conditional-sum like addition but with a different set of equations in the operation [12]. In spite of its large variations in fan-out it has even been used in an wave-pipelined adder [11]. Wei and Thompson modified this algorithm slightly to make place for cells handling the increasing fan-out [9] [10].

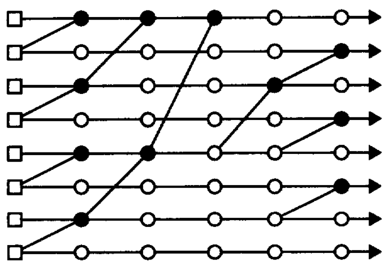
This algorithm has optimal depth but the fan-out increases exponentially towards the final stages and is linear in the number of bits. This results in a large delay when operands are large.

Figure 3: Sklansky's prefix algorithm



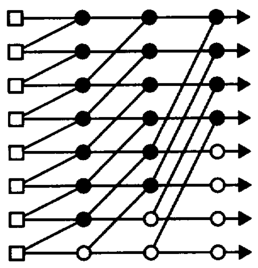
The fan-out can be handled by the use of another algorithm. Brent and Kung invented an algorithm with low fan-out everywhere but with a depth of about double the optimal depth. This algorithm has linear circuit complexity, but when laid out according to Brent- and Kung it requires O(n log n) area. This is due to the tree-structures (the tree in the first half and the inverted tree in the last half). It may not be area efficient but the low number of nodes together with the small amount of inter-bit communication makes an adder using this algorithm quite power efficient. The algorithm has been used for carry-look-ahead addition using the GP-operation [3][13].

Figure 4: Brent and Kung's prefix algorithm



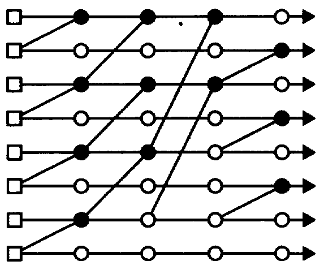
An optimal depth algorithm that keeps fan-out low results when the linear recurrence algorithm invented by Kogge and Stone is generalized [4]. It has been used for Carry-Lookahead adding by Papadopoulou [14], Klein [17] and Yuan [18]. The cost is a higher gate complexity and a higher inter-bit communication complexity, which results in higher power consumption and possibly larger area than Sklansky's algorithm.

Figure 5: Kogge and Stone's algorithm



By combining Brent and Kung's algorithm [3] with Kogge and Stone's [4] algorithm, Han and Carlson achieves intermediate values on delay and area [5]. They allow k levels to be added to the optimal depth. They suggest $k=1$ for $n \leq 64$ and $\log n - \log\log n \leq k \leq \log n - 1$ for $n>64$. Han and Carlson claim that the area due to inter-bit communication is considerably reduced compared to Kogge and Stone's algorithm. This may be true in theory, and in practice for large n's, but for moderate values on n there is one factor they do not mention. In both the generate-propagate and in the Conditional-Sum equations, more information than needed by the last stage is produced, e.g. propagate in the GP-operation and C^1 in the CSA-operation. If these nodes are reduced to produce only necessary information, the edges in the last stage of any prefix algorithm should be weighted by one half. The Kogge and Stone method is the one that can benefit most from this, while Han and Carlson hardly has any advantage at all in their method.

Figure 6: Han and Carlson's prefix algorithm

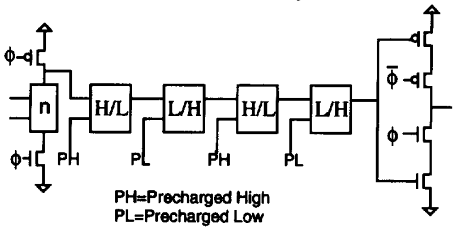


7. Realization and simulation

A number of adders have been implemented and simulated on transistor level. Three different algorithms have been used; Sklansky's, Kogge-Stone's and Brent-Kung's, together with both the GP- and the CSA-operation. Each of these have been implemented in 5 different sizes; 4, 8, 16, 32 and 64 bits. The operations have been implemented mainly using the clock-and-data precharged dynamic (CDPD) circuit technique [16]. The CDPD-technique share some properties with Domino-logic, e.g., the functions implemented in the circuit technique are limited to monotone functions. All the adder configurations have been implemented both with and without capacitances representing the long inter-bit communication wires. The capacitance added corresponds to a 45 μm long and 2 μm wide metal-1 wire segment per bit spanned by the wire.

In the CDPD circuit technique the acronyms H/L and L/H are used to describe the precharge polarity. A H/L block requires signals that are precharged to high on it's inputs, and produces signals which are precharged low on the outputs. The L/H block is the other way around. The blocks can form a chain with a clock precharged n-stage in the beginning, followed by alternating data precharged H/L and L/H blocks. This is called a n-chain [16]. In such a chain it should be noted that the precharging of the gates propagate along the chain with a delay in each gate. Therefore great caution must be exercised when sizing the gates so that the precharge-time of the chain does not exceed the evaluation time. The cascaded chains of H/L and L/H blocks that follow the clock-precharged n-stage are terminated by ordinary NORA-latches [15]. Such chains are used for carry calculation in both CSA and CLA. In the last stage of a CLA, the non-monotone addition prevents the use of a H/L or L/H block. Therefore a pass-transistor multiplexer is used for the addition.

Figure 7: n-chain terminated by an NORA-latch



The transistors in the CSA- (figure 9 and figure 10) and GP-blocks (figure 11 and figure 12) have been sized according to very simple and identical rules. The width of both the evaluation and the precharge-transistors are minimum width for the device times the number of transistors in series between the output and the power-rail. The bit-level sum- and carry-generation blocks (figure 8) have been sized the same way, except for the larger clock-driven precharge-transistors. The final NORA-latch is identically sized for CSA and CLA realizations and has a minimum inverter as load on its output. The number placed in connection with each transistor symbol is the width of the transistor. The gate length is 1 μm for all transistors.

When realizing an adder one is typically not interested in any intermediate carry outputs. Only the resulting sum's and the resulting carry-out, that is $S^0_{0:n-1}$ down to $S^0_{0:0}$ and $C^0_{0:n-1}$ are needed. This implies that some of the gates in the operation blocks are redundant. These gates have been removed, something which mainly affects the last two stages of CSA-adders.

The chains of multiplexers used to form the sums in the CSA realizations are pass-transistor multiplexers with two levels of pass-transistors between each inverting buffer, according figure 9 and figure 10.

Figure 8: Block for bit sum and carry generation (n-type)

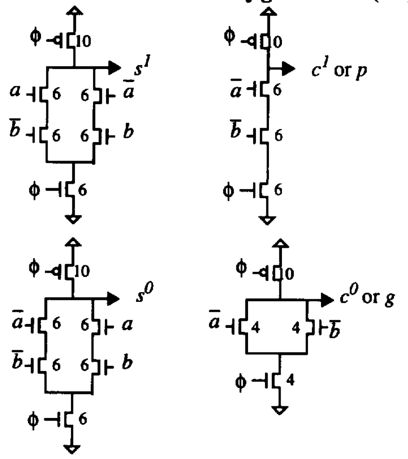


Figure 9: CSA block L/H type

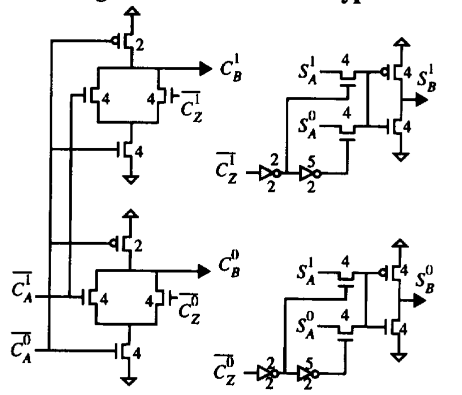


Figure 10: CSA block H/L type

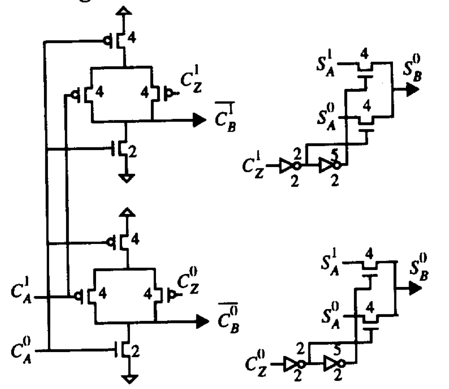


Figure 11: GP block L/H type

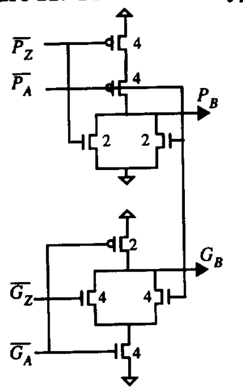
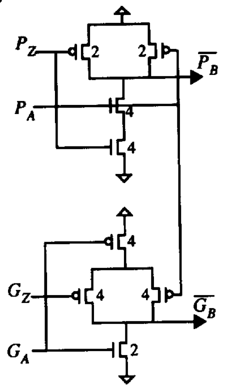


Figure 12: GP block H/L type



The simulations have been carried out using Mentor Graphics Lsim simulator running in ADEPT mode [23]. This circuit simulator uses SPICE-like device models. Transistor parameters correspond to a standard industrial 1.0 μm CMOS process. Complete adder circuits have been simulated and their delays have been measured as the time from rising clock edge to a stable sum on the output. The carry-out signal is faster than the sum-bits in all the simulations. Input patterns have been carefully selected to stress the critical path.

It should be noted that the lines between the data-points in the diagrams are present only as a visual aid, they have no meaning as intermediate values.

All simulations have been made both with and without wire capacitances. Only delays have been measured, the aspects of power consumption or hardware complexity have not been considered in the comparison.

Figure 13: Addition using Sklansky's algorithm

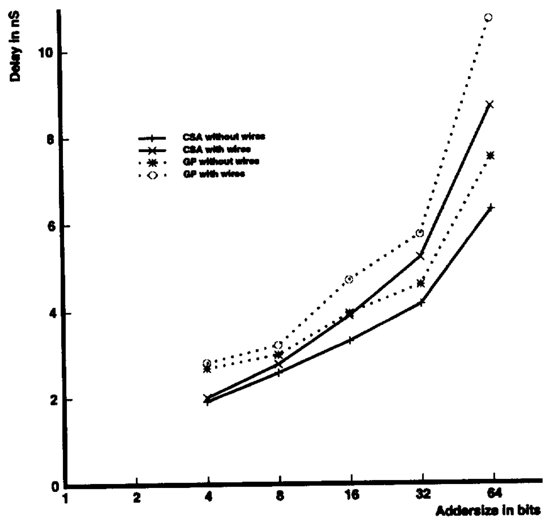


Figure 14: Addition using Kogge and Stone's algorithm

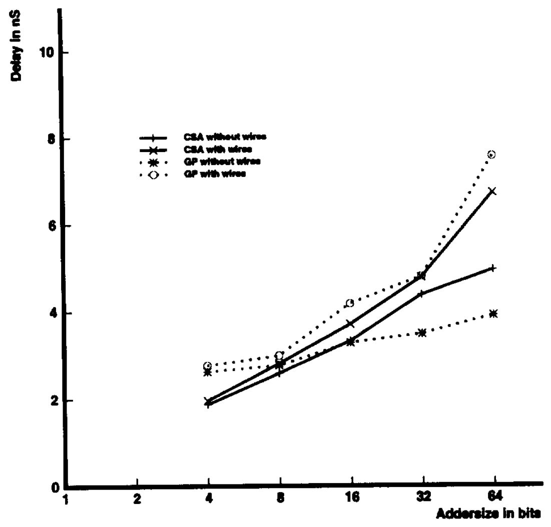
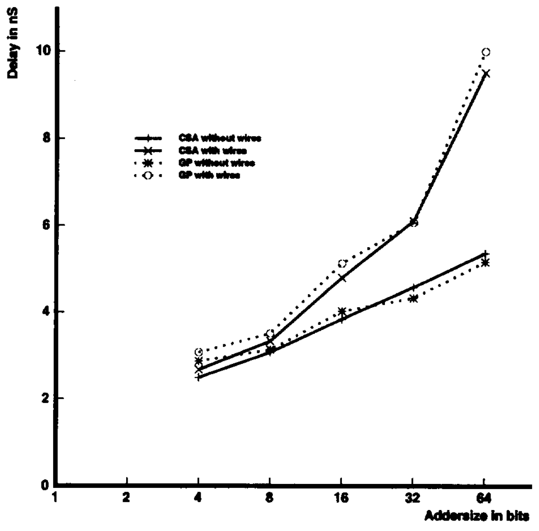


Figure 15: Addition using Brent and Kung's algorithm



8. Discussion

As can be seen from the diagrams all of the implemented adders are sensitive to wire capacitances. The effect can be reduced by careful sizing, but the problem still remains; The length and thereby the capacitance of the inter-bit communication wires is proportional to the width of the adder. This can be seen as an exponential component in the diagrams related to the logarithmic x-axes.

We note that Brent and Kung's algorithm has a higher sensitivity to long inter-bit communication wires than the others. In Sklansky and Kogge and Stone based circuits, the critical path is loaded by $n-1$ length-units of inter-bit wiring. For Brent and Kung based adders the corresponding load is $\frac{3}{2}n-2$ length-units, which is almost 50% higher.

Since all adders have logarithmic depth the expected delay, without wiring capacitances, should correspond to a straight line in the diagrams due to the logarithmic x-axis. This holds for the Brent and Kung and for the Kogge and Stone based adders. For Sklansky's adder structure the curve bends upwards due to higher fan-out in the latter stages.

9. Summary and conclusions

Binary adders, which are critical system building blocks, must be implemented using techniques that properly match the characteristics of the underlaying technology. For CMOS-designs, delay is as much related to node capacitance as to gate depth while the high area efficiency often allows increased complexity to be traded for increased speed.

To date, most aggressive CMOS adder designs have been based on carry-lookahead schemes structured according to a low fan-out prefix algorithm. Conditional sum adders have smaller depth but when they are structured according to Sklansky some internal nodes are heavily loaded and it is difficult to achieve high speeds for CMOS implementations.

We have demonstrated that virtually all the methods used in CLA design are applicable also in the CSA case. In particular, the CSA-operation together with the sum- and carry-information it operates on, has been shown to be a monoid which allows any parallel prefix algorithm to be used for the sum calculation. In fact, all schemes suggested to restructure CLAs, e.g., to decrease internal fan-out, can be applied also to CSAs. Furthermore, we have shown that all computations related to the carry-information can be reduced to use an operation which is monotone and similar to the generate-operation. This allows fast dynamic logic to be used for all critical parts of a CSA.

Transistor level implementations of CSAs and CLAs have been used as a basis for performance comparison. Adders, based on three different prefix algorithms and five different bit widths, ranging from 4 to 64 bits, were designed and simulated. In almost all cases the CSA-based implementation is faster than its CLA counterpart.

We conclude that when conditional-sum adders are organized as low fan-out parallel prefix circuits and implemented in CMOS with fast dynamic logic, then their lower gate depth result in a speed improvement over carry-lookahead adders even though most internal nodes have slightly higher fan-out. Thus, CSA structures should be considered as main candidates for future high performance CMOS adders.

10. Acknowledgements

This work was sponsored by the Swedish National Board for Technical Development (NUTEK) under contract number 93-02633.

References

[1] J. Sklansky, "Conditional-Sum Addition Logic", In *IRE Transactions on Electronic Computers,* Vol. EC-9, No. 2, pp. 226-231, June 1960.

[2] O. J. Bedrij, "Carry Select Adder", In *IRE Transactions on Electronic Computers,* Vol. EC-11, No. 3, pp. 340-346, June 1962.

[3] R. T. Brent and H. T. Kung, "A Regular Layout for Parallel Adders", In *IEEE Transactions on Computers,* Vol. C-31, No 3, pp. 260-264, March 1982.

[4] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations", In *IEEE Transaction on Computers,* Vol C-22, No. 8, pp. 786-792, August 1973.

[5] T. Han and D. A. Carlson, "Fast Area-Efficient VLSI Adders", In *Proceedings 8th Symposium on Computer Arithmetic,* pp. 49-56, May 1987.

[6] R. E. Ladner and M. J. Fischer, "Parallel Prefix Calculation", In *Journal of the Association for Computing Machinery,* Vol. 27, No. 4, pp. 831-838, October 1980.

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

[8] Ingo Wegener, In *The Complexity of Boolean Functions,* ISBN 0 471 91555 6 (Wiley), ISBN 3 519 02107 2 (Teubner), pp. 47-50, 1987.

[9] B. W. Y Wei, C. Thompson and Y. Chen, "Time-Optimal Design of a CMOS Adder", In *Proceedings 19th Asilomar Conference on Circuits, Systems and Computers,* pp. 186-191, November 1985.

[10] B. W. Y Wei and C. Thompson, "Area-Time Optimal Adder Design", In *IEEE Transactions on Computers,* Vol. 39,pp. 666-675, May 1990.

[11] D. Fan, C. T. Gray, W. Farlow, T. Hughes, W. Liu and R. K. Cavin, "A CMOS Parallel Adder Using Wave Pipelining", In *Advanced Research in VLSI and Parallel Systems, Proceedings of the 1992 Brown/MIT Conference,* pp.147-164.

[12] T. P. Kelliher, R. M. Owens, M. J. Irwin and T.-T. Hwang, "ELM - A Fast Addition Algorithm Discovered by a Program", In *IEEE Transactions on Computers,* Vol. 41, No. 9, September 1992.

[13] W. Schardein, B. Weghaus, O. Maas, B. J. Hosticka and G. Tröster, "A Technology Independent Module Generator for CLA Adders", In *Proceedings 18th European Solid State Circuits Conference,* pp. 275-278, September 1992.

[14] E. Papadopoulou, "A Fast Parallel Binary Adder", In *Proceeding of the IASTED International Symposium: Applied Simulation and Modelling - ASM '85,* pp. 29-31, June 1985.

[15] N. F. Goncalves and H. J. De Man, "NORA: A racefree dynamic CMOS technique for pipelined logic structures", In *IEEE Journal of Solid State Circuits,* Vol. SC-18, pp 261-266, 1983.,

[16] J.-R. Yuan, C. Svensson and P. Larsson, "New Domino logic precharged by clock and data", In *Electronics Letters,* Vol. 29, No. 25, pp. 2188-2189, December 1993.

[17] Michael Klein, "Entwurf und Vergleich von schnellen Addierern zur Integration von Microprozessoren" (content in english), Matr.-Nr. 163098, Lehrstuhl fur Allgemeine Elektrotechnik, Rheinisch-Westfahlische Technische Hochschule, Aachen.

[18] J.-R. Yuan, "An ultrafast adder arrangement", Swedish patent application no. 9302158-2, June 1993.

[19] T. Lynch, E. Swartzlander, "The Redundant Cell Adder", In *Proceedings 10th Symposium of Computer Arithmetic,* pp. 165-170, 1991.

[20] T. Lynch, E. Swartzlander, "A Spanning Tree Carry Lookahead Adder", In *IEEE Transactions on Computers,* Vol. 41, No. 8, August 1992.

[21] V. Kantabutra, "A Recursive Carry-Lookahead/Carry-Select Hybrid Adder, In *IEEE Transactions on Computers,* Vol. 42, No. 12, pp. 1495-1499, December 1993.

[22] D. W. Dobberpuhl, R. T. Witek et. al., "A 200-MHz 64-bit Dual-issue CMOS Microprocessor", In *IEEE Journal of Solid-State Circuits,* Vol. 27, No. 11, pp. 1555-1567, November 1992.

[23] P. Odryna et al., "A Workstation-Based Mixed Mode Circuit Simulator", In *Proceedings 23rd Design Automation Conference,* pp. 186-192, 1986.