Optimal Purely Systolic Addition

Lars Kühnel
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität
W–2300 Kiel 1, Germany

Abstract

We introduce a purely systolic hardware algorithm for addition which is based on a mesh connected arrangement of cells. The proposed $\mathcal{FASTA}$-algorithm is well-suited for realization in integrated technologies. Its area, computation time, and period are satisfying $A(n) = O(n)$, $T(n) = O(\sqrt{n})$, $P(n) = O(\sqrt{n})$, respectively, where n denotes the operand length. Therefore, this adder is T-, APT-, and AT^2-optimal in the linear model for signal propagation delays. In the class of $\Theta(\sqrt{n})$ time adders it is optimal with respect to A, P, T, AP, AT, APT, AP^2, and AT^2. The suggested algorithm essentially is a solution to the general problem of "parallel prefix computation". Therefore, it can serve as a paradigm for the design of optimal purely systolic hardware algorithms in a wide range of application domains.

1 Introduction

We consider the addition of non-negative integers in conventional positional binary n-bit representation. Given the $2n$ bits of two operands a and b, the task is to compute the $(n+1)$-bit representation of the sum s.

$$s = \sum_{i=0}^{n} s_i 2^i = a + b = \sum_{i=0}^{n-1} a_i 2^i + \sum_{i=0}^{n-1} b_i 2^i$$

For each $n \in \mathbb{N}$ (the *problem size*) this formulation defines a mapping $ADD_n : \mathbb{B}^n \times \mathbb{B}^n \rightarrow \mathbb{B}^{n+1}$ that maps each pair of n-bit operands to the corresponding $(n+1)$-bit tuple of the sum, leading to the sequence of mappings $ADD = (ADD_n)_{n \in \mathbb{N}}$.

In this paper we introduce a novel globally clocked hardware algorithm $\mathcal{FASTA}$ for the computation of ADD. $\mathcal{FASTA}$ actually is a sequence $(\mathcal{FASTA}_n)_{n \in \mathbb{N}}$ of hardware algorithms with $\mathcal{FASTA}_n$ computing ADD_n. This new adder is well-suited for realization in integrated technologies like VLSI, ULSI, and WSI. The algorithm can be easily extended in order to cover two's complement addition and subtraction, too.

According to the model introduced in [12], a hardware algorithm H is a tuple (G, L, S). G is a finite directed graph, the *computation graph* of H. The nodes correspond to processing elements (PEs) resp. input and output nodes and the edges correspond to the wires connecting the cooperating PEs. The *layout* L describes an embedding of G into the plane $\mathbb{R}^2$, thus suggesting a two-dimensional realization of G. S is the *I/O-scheme* of H and specifies the times (i.e. clock ticks) and locations (i.e. input resp. output nodes) at which values for $\{a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}\}$ are read resp. for $\{s_0, \ldots, s_n\}$ are written.

Our complexity analysis refers to the usual VLSI complexity measures. Let $\mathcal{H} = (H_n)_{n \in \mathbb{N}}$ be a sequence of hardware algorithms. The *area* $A_{\mathcal{H}}(n)$ of H_n is the area of the smallest convex subset of $\mathbb{R}^2$ that contains the layout of H_n. The *time* (or *latency*) $T_{\mathcal{H}}(n)$ of H_n is the time elapsed between the application of the first input bit and the generation of the last output bit with respect to a single computation. Since hardware algorithms are working on infinite streams of arguments, we are additionally considering the *period* $P_{\mathcal{H}}(n)$ of H_n. The period measures the time elapsed between the application of the first input bits of two successive computations (i.e. the time between two successive sets of inputs). Note that the reciprocal of $P_{\mathcal{H}}(n)$ corresponds to the *throughput* of H_n. Moreover, we investigate the behaviour with respect to the usual product measures $AP_{\mathcal{H}}(n)$, $AT_{\mathcal{H}}(n)$, $APT_{\mathcal{H}}(n)$, $AT^2_{\mathcal{H}}(n)$, and $AP^2_{\mathcal{H}}(n)$.

In [5, 9] Foster and Kung have identified a set of properties that make a hardware algorithm especially well-suited for realization in integrated technologies. These *systolic* algorithms employ a high degree of parallelism and pipelining for the efficient solution of a given problem. Based on the above papers, Schmeck [12] suggests the following concise definition of systolic algorithms.

- The computation graph uses only few types of simple cells.
- The layout suggests a realization with simple and regular data and control flows.
- For each node, the number of direct neighbours is small and does not depend on the problem size.
- The design uses extensive concurrency.
- For each node, the area occupied by its realization does not depend on the problem size.
- The suggested realization uses only physically local communication for the exchange of data between PEs. In particular, the length of the longest wire used for data communication does not depend on the problem size.
- The realization can be operated at a clock frequency that does not depend on the problem size.
- Average and maximum data rates have the same magnitude.

It is crucial to note that *purely systolic* hardware algorithms totally avoid long–distance or irregular wires for data communication (cf. [9]).

Purely systolic hardware algorithms are in particular promising low costs for the design of realizations, high throughput, modular expansibility without slowing down the system clock, and steady use of the interface to their environment. Moreover, some of the above properties prove to be advantageous with respect to *testability* aspects. Mesh connected array-like hardware algorithms with simple cells are important examples for purely systolic systems and are thus ideal candidates for the realization in integrated technologies.

Note that the well-known fast adders (see e.g. [1, 2, 13]) are based on tree-like hardware structures and thus don't belong to the class of purely systolic systems. A first hint concerning fast *and* purely systolic addition can be found in [3]. Unfortunately, Chazelle and Monier's description is not very detailed . Moreover, their algorithm is not *purely* systolic in the above strong sense.

In this paper we propose a new hardware algorithm for fast purely systolic addition ($\mathcal{FASTA}$). The $\mathcal{FAST}$ adder is T-, APT-, and AT^2-optimal in the linear model for signal propagation delays. Moreover, it is optimal with respect to *all* usual measures in the class of time optimal (i.e. $\Theta(\sqrt{n})$ time) adders. Since the algorithm is based on the well-known carry lookahead technique, it suggests a purely systolic solution to the general problem of *parallel prefix computation* (see [10]). Thus the $\mathcal{FASTA}$ hardware algorithm is of significant relevance for a wide range of application domains.

The paper is organized as follows. Section 2 contains a concise summary of lower bounds in the linear model and includes a special treatment of time optimal hardware algorithms for addition. Based on the list of lower bounds, we develop the $\mathcal{FASTA}$ algorithm in Section 3. Section 4 contains some remarks concerning additional properties and possible extensions of the $\mathcal{FAST}$ adder.

2 Lower Bounds for Purely Systolic Addition

Obviously, the value of the mapping ADD_n depends on the operands. Thus a hardware algorithm $\mathcal{H} = (H_n)_{n\in\mathbb{N}}$ for addition has to read the argument bits and to perform a calculation. Since there exists a minimum feature width (cf. e.g. [4]), this results in

$$A_{\mathcal{H}}(n) = \Omega(1) .$$

A simple counting argument, involving the number of bits to be produced and the number of output nodes that are used for these output purposes, leads to (see e.g. [4, 8])

$$AT_{\mathcal{H}}(n) = \Omega(n) \text{ and } AP_{\mathcal{H}}(n) = \Omega(n) . \quad (1)$$

If the number of output nodes used by H_n is $O(n)$, then

$$P_{\mathcal{H}}(n) = \Omega(1) \text{ and thus } AP^2_{\mathcal{H}}(n) = \Omega(n) . \quad (2)$$

Clocked versions of a ripple carry adder (see e.g. [6]) with a constant number of full-adder cells resp. of an n-cell ripple carry adder show that the above lower bounds are tight.

The lower bound for the computation time heavily depends on the underlying model for signal propagation delays. The widely used *constant model* assumes that the time for propagating a signal across a wire of length L does not depend on L. Thus this model should not be used as a basis for the derivation of purely systolic hardware algorithms since it does not punish the designer for using long wires. In a purely systolic algorithm, every datapath of length L crosses $\Theta(L)$ processing elements with each of these elements contributing at least one clock tick to the overall delay of a signal that has to be propagated across this datapath. Thus it is appropriate to base the analysis on the *linear model* advocated in [4]. In this model, the time for the propagation of a signal across a wire of length L is assumed to be $\Omega(L)$. Chazelle and Monier [4] have shown that this assumption leads to the following lower bounds for addition.

$$\begin{aligned} T_{\mathcal{H}}(n) &= \Omega(\sqrt{n}) && (3)\\ APT_{\mathcal{H}}(n) &= \Omega(n^2) && (4)\\ AT^2_{\mathcal{H}}(n) &= \Omega(n^2) . && (5) \end{aligned}$$

Note that the "conventional" fast adders are not time optimal in the linear model since they are based on tree structures and therefore employ wires whose length depend on n. Due to the latter fact, the length of a basic clock cycle depends on n, too. In general, this leads to a computation time in $\omega(\sqrt{n})$. For instance, the carry lookahead adder suggested in [2] has a time complexity $\Omega(n)$ in the linear model.

Section 3 shows that there exists a purely systolic adder whose time is $\Theta(\sqrt{n})$. Thus the lower bound 3 is tight. We derive from 5 that the area of a time optimal adder Opt satisfies

$$A_{Opt}(n) = \Omega(n) . \quad (6)$$

Moreover, "reasonable" time optimal adders satisfy

$$P_{Opt}(n) = \Omega(\sqrt{n}) \quad (7)$$

(this bound is shown in [8]). A combination of 3, 6, and 7 yields

$$AP_{Opt}(n) = \Omega(n\sqrt{n}) , \ AT_{Opt}(n) = \Omega(n\sqrt{n}) ,$$

$$AP^2_{Opt}(n) = \Omega(n^2) .$$

Table 1 contains a compilation of the above bounds. Complete proofs can be found in [8].

3 $\mathcal{FASTA}$: A hardware algorithm for fast purely systolic addition

The $\mathcal{FAST}$ adder is based on the well-known technique of carry lookahead addition. The following subsection outlines the basic ideas of this technique.

3.1 Carry Lookahead Technique

The bits of the sum s can be computed through the use of carry signals c_i, $i \in [-1, n-1]$. (Given two integers i, j, let $[i, j]$ denote the set of integers $\{k | i \le k$ *and* $k \le j\}$. Furthermore, let $\wedge, \vee, \oplus$ denote the logical AND, OR, and exclusive OR, respectively.)

$$\begin{aligned} c_{-1} &= 0 \\ c_i &= (a_i \wedge b_i) \vee (a_i \wedge c_{i-1}) \vee (b_i \wedge c_{i-1}) \\ s_i &= a_i \oplus b_i \oplus c_{i-1} \\ s_n &= c_{n-1} \end{aligned}$$

The carry lookahead technique involves assigning a pair of bits $(G[i,j], P[i,j])$ to each group of bit positions $[i,j] \subseteq [0, n-1]$ with G and P satisfying

$(G[i,j] = 1)$	$\Longleftrightarrow$	The group $[i,j]$ generates a carry into position $(j+1)$ regardless of an incoming carry.
$(P[i,j] = 1)$	$\Longleftrightarrow$	The group $[i,j]$ propagates an incoming carry from position $(i-1)$ into position $(j+1)$.

Obviously, we have $G[0,i] = c_i$ for all $i \in [0, n-1]$. The (G, P) pair of a single bit position i can be computed according to

$$G[i,i] = a_i \wedge b_i \ ; \ P[i,i] = a_i \oplus b_i \ .$$

Introducing the operator $\circ : \mathbb{B}^2 \times \mathbb{B}^2 \to \mathbb{B}^2$ with

$$(G, P) \circ (G', P') = (G' \vee (P' \wedge G), P \wedge P') \ ,$$

we derive the property

$$(G[i,j], P[i,j]) = (G[i,k-1], P[i,k-1]) \circ (G[k,j], P[k,j])$$

for all $k \in [i+1, j]$.

The concept of carry lookahead addition is based on the observation that the operator '$\circ$' is associative. Due to the latter fact, the carries $G[0,i]$ can be computed from the single bit (G, P) pairs in any order as long as only associativity is exploited.

3.2 Design Considerations

The derivation of a purely systolic time optimal adder was guided by the following crucial ideas. Throughout the rest of the paper, let $n = m^2, m \in \mathbb{N}$.

Firstly, the n bit positions are divided into $\sqrt{n}$ groups of $\sqrt{n}$ bit positions each. This leads to the sequence of *blocks*

$$([q\sqrt{n}, (q+1)\sqrt{n} - 1])_{q \in [0, \sqrt{n}-1]} \ .$$

The (G, P)-pair of each of these blocks is computed in a ripple carry adder like fashion using the property

$$(G[q\sqrt{n}, k], P[q\sqrt{n}, k]) = (G[q\sqrt{n}, k-1], P[q\sqrt{n}, k-1]) \circ (a_k \wedge b_k, a_k \oplus b_k) \ .$$

Thus each of these block computations takes $O(\sqrt{n})$ time. It is possible to pipeline the block computations and therefore to complete the computation of *all* $\sqrt{n}$ block-(G, P)s in $O(\sqrt{n})$ time.

The sequence of *cumulative block carries*

$$(G[0, q\sqrt{n} - 1])_{q \in [1, \sqrt{n}]}$$

can be computed iteratively from the block-(G, P)s using the equality

$$G[0, q\sqrt{n}-1] = G[(q-1)\sqrt{n}, q\sqrt{n}-1] \vee \vee (P[(q-1)\sqrt{n}, q\sqrt{n}-1] \wedge G[0, (q-1)\sqrt{n}-1]). \quad (8)$$

The generation of the sum bits s_i is divided into two stages. Let $i \in [q\sqrt{n}, (q+1)\sqrt{n} - 1]$, $i = q\sqrt{n} + k$, i.e. i belongs to the qth block. s_i satisfies the equation

$$s_i = a_i \oplus b_i \oplus c_{i-1} = a_i \oplus b_i \oplus G[0, i-1]$$

and thus

$$s_i = a_i \oplus b_i \oplus \oplus(\underbrace{G[q\sqrt{n}, i-1]}_{\text{intra-block carry}} \vee (P[q\sqrt{n}, i-1] \wedge \underbrace{G[0, q\sqrt{n} - 1]}_{\text{carry into } q\text{th block}})).$$

Note that $(G[q\sqrt{n}, i-1], P[q\sqrt{n}, i-1])$ are intra-block (G, P) signals, whereas $G[0, q\sqrt{n} - 1]$ is the cumulative block carry entering the qth block.

Stage 1 of the sum bit generation process performs the computation of *preliminary sum bits* $s[q\sqrt{n}, i]$ from the operands a_i, b_i and the intra-block carries:

$$s[q\sqrt{n}, i] = a_i \oplus b_i \oplus G[q\sqrt{n}, i-1] \ .$$

Exploiting the property

$$s_i = s[q\sqrt{n}, i] \oplus (P[q\sqrt{n}, i-1] \wedge G[0, q\sqrt{n} - 1]) \ ,$$

the second stage of the sum bit generation computes the s_is from the preliminary sum bits, the intra-block propagate signals, and the sequence of cumulative block carries.

The generation of the preliminary sum bits and of the cumulative block carries can be integrated into the ripple carry adder like computation mentioned above. Stage 2 of the sum generation is performed by using an additional linear array of identical PEs with a time skewed input.

The following list summarizes the basic ideas that have led to the $\mathcal{FASTA}$ hardware algorithm.

- Carry lookahead addition, using $\sqrt{n}$ blocks of $\sqrt{n}$ successive bit positions.
- Ripple carry adder-like (G, P) computation within blocks, using a linear array of $\sqrt{n}$ PEs.
- Pipelining of block computations on the latter array.

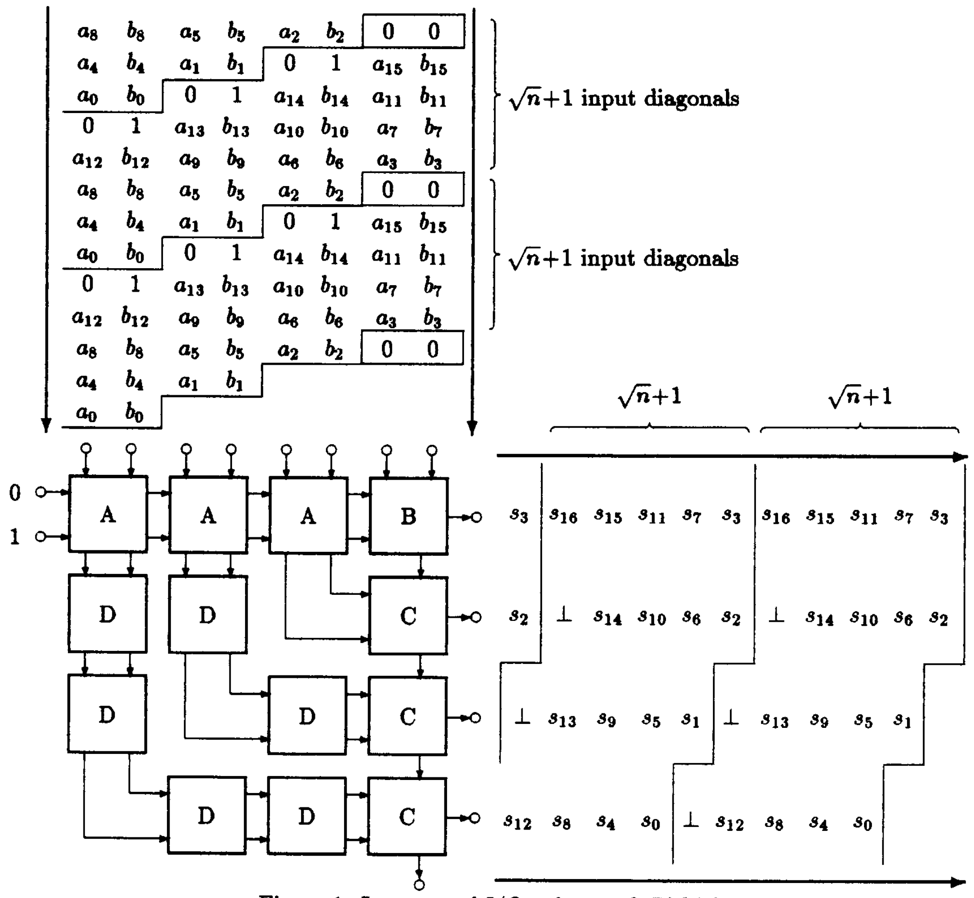


Figure 1: Layout and I/O–scheme of $\mathcal{FASTA}_{16}$.

- Iterative computation of the cumulative block carries by the rightmost cell.
- Two-stage generation of sum bits. Computations of Stage 1 incorporated into the above ripple carry adder-like computation. Computations of Stage 2 performed by a second linear array of $\sqrt{n}$–1 cells. Intermediate results of first stage are delayed appropriately, using a sequence of $\sqrt{n}$–2 shift registers of decreasing length.

3.3 An Overview

Figure 1 gives the layout of the computation graph and sketches the I/O–scheme of $\mathcal{FASTA}_n$ for $n = 16$. The layout uses a mesh connected array like arrangement of $n - (\sqrt{n} - 1)$ cells with the linear array of $\sqrt{n} - 1$ A–cells and the B–cell in the top row performing the ripple carry adder like computation mentioned above. This linear array produces the preliminary sum bits and provides the sequence of cumulative block carries via the bottom output line of the B–cell. The latter sequence is read by the top cell of a linear array of $\sqrt{n}-$ 1 C–cells in the rightmost column, which implements the second stage of sum bit generation. Accordingly, the intermediate results of the linear array of A–cells have to be delayed appropriately in order to meet the corresponding cumulative block carries in the array of C–cells. This can be achieved by using the indicated arrangement of $n - 3\sqrt{n} + 2$ D–cells with each D–cell simply delaying its two input signals by one clock tick. The input is read in the form of $\sqrt{n}$ diagonals of $2\sqrt{n}$ operand bits, each input diagonal corresponding to one of the blocks introduced in 3.2. An additional diagonal of constant pairs separates the inputs of successive computations. The output is generated in the form of $\sqrt{n} + 1$ distorted diagonals of $\sqrt{n}$ sum bits each. Again, each diagonal corresponds to one of the blocks of 3.2. The last output diagonal of a computation contains the most significant sum bit s_n and $\sqrt{n}-1$ "don't cares".

3.4 Components

First of all, we will describe the linear array of A– and B–cells. Let $i = q\sqrt{n} + k < (q + 1)\sqrt{n}$. The A–cells are used for computing the intra–block signals $(G[q\sqrt{n}, i], P[q\sqrt{n}, i])$ and the preliminary sum bits $s[q\sqrt{n}, i]$ from the inputs $(G[q\sqrt{n}, i-1], P[q\sqrt{n}, i-1])$, a_i, and b_i (see Fig. 2). Moreover, an A–cell propagates the incoming intra–block P–signal via the second bottom output line. Note that the outputs are produced on the basis of a global clock. Thus the output signals indicated in Fig. 2 are available at clock tick $t + 1$, provided that the shown inputs are applied to the cell

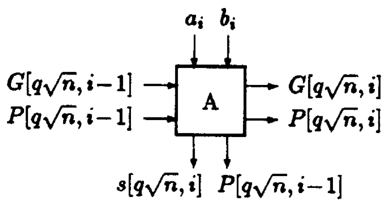


Figure 2: I/O-behaviour of the A-cells

input lines at clock tick t. Obviously, the A-cells can be realized by using a small number of logical gates and flipflops.

A linear array of $\sqrt{n}$ A-cells together with a time skewed input (as shown in Fig. 1) could be used for generating all the preliminary sum bits and their corresponding intra-block propagate signals. It is crucial to note that the rightmost A-cell produces the signals that are needed for computing the sequence of cumulative block carries (cf. 8 and Figure 3). Moreover, since

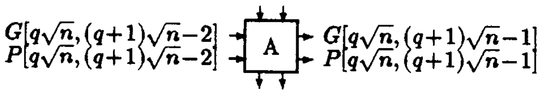

$G[q\sqrt{n},(q+1)\sqrt{n}-2]$
$P[q\sqrt{n},(q+1)\sqrt{n}-2]$
A
$G[q\sqrt{n},(q+1)\sqrt{n}-1]$
$P[q\sqrt{n},(q+1)\sqrt{n}-1]$

Figure 3: I/O of a hypothetical $\sqrt{n}$th A-cell

$$s_{(q+1)\sqrt{n}-1} = s[q\sqrt{n},(q+1)\sqrt{n}-1] \vee \\ \vee \left(P[q\sqrt{n},(q+1)\sqrt{n}-2] \wedge G[0,q\sqrt{n}-1]\right) ,$$

the same holds for the sequence of sum bits $(s_{(q+1)\sqrt{n}-1})_{q\in[0,\sqrt{n}-1]}$, too. Therefore, we incorporate the corresponding computations into the hypothetical $\sqrt{n}$th A-cell, thus introducing a new cell type B. Figure 5 outlines a possible realization of the B-cell. The symbols used for standard logical components are described in Fig. 4.

The I/O-behaviour of the complete linear array is given in Fig. 6. At the beginning of a computation, the D-flipflop controlling the O2 output line of the B-cell has to be cleared. Therefore, the input pair $(0,0)$ precedes the first pair of operand bits in the sequence of inputs for the B-cell. At the end of a computation, this D-flipflop contains the value $G[0,n-1] = s_n$. In Fig. 6, the last input diagonal of $\sqrt{n}-1$ $(0,1)$-pairs and a single $(0,0)$-pair generates the input $I1 = I2 = I4 = 0$ and $I3 = 1$ with respect to the B-cell. It can be easily verified that this input combination makes the content of the O2-flipflop (i.e. s_n) appear on the O1 output line of the B-cell.

Since the input pair $(0,0)$ of the last input diagonal clears the O2-flipflop of the B-cell, successive computations can be pipelined as indicated in Fig. 1.

The sequences of values available at the bottom output lines of the above linear array are comprising exactly the inputs needed for the second stage of sum bit generation. Thus a linear array of $\sqrt{n}-1$ cells can be used to perform the corresponding computations. The latter array uses cells of type C, whose I/O-behaviour is given by Fig. 7. Figure 8 outlines the I/O-behaviour of the complete linear array of C-cells. The time skewed input format for this array can be generated by employing the arrangement of D-cells given in Fig. 1.

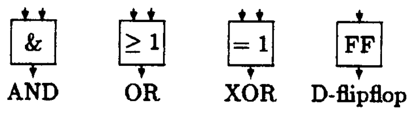


Figure 4: Symbols for logical components

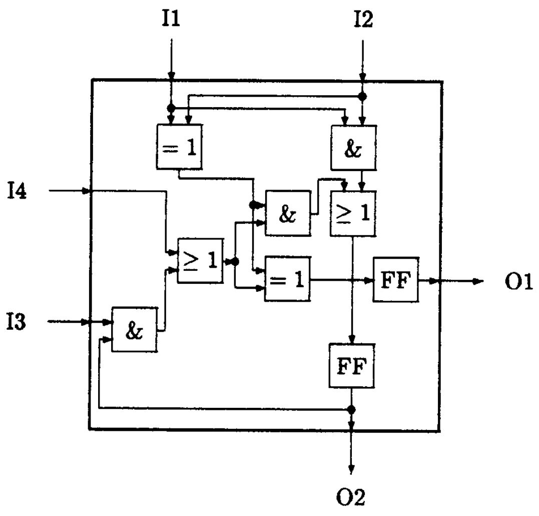


Figure 5: Realization of the B-cell

3.5 Analysis

Each cell of the $\mathcal{FAST}$ adder occupies only constant area. Thus the complete layout can be embedded into an $O(\sqrt{n}) \times O(\sqrt{n})$ array of constant area rectangles. Thus

$$A_{\mathcal{FASTA}}(n) = O(n) .$$

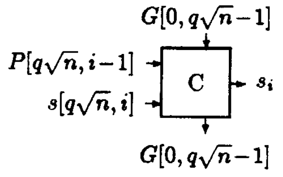
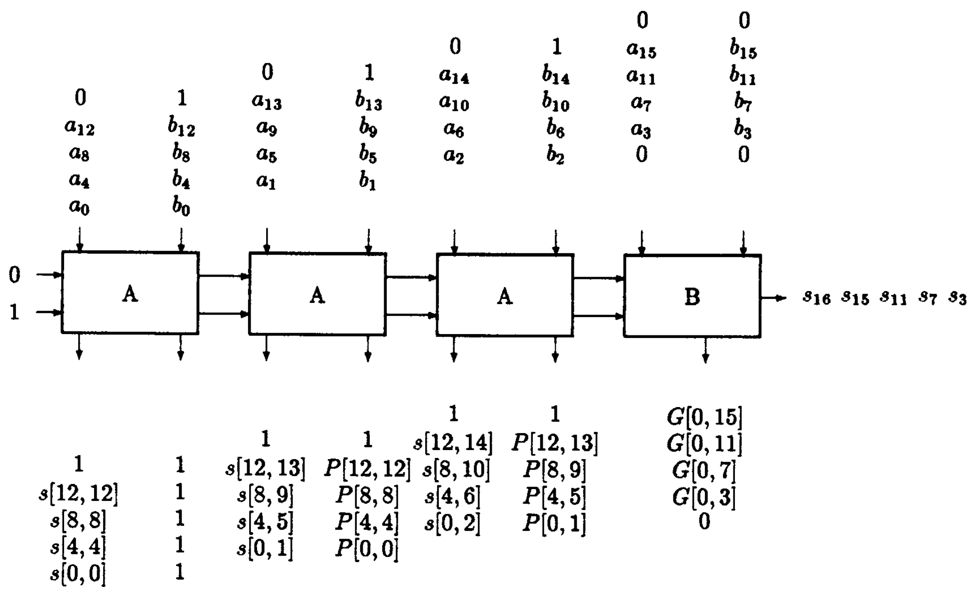
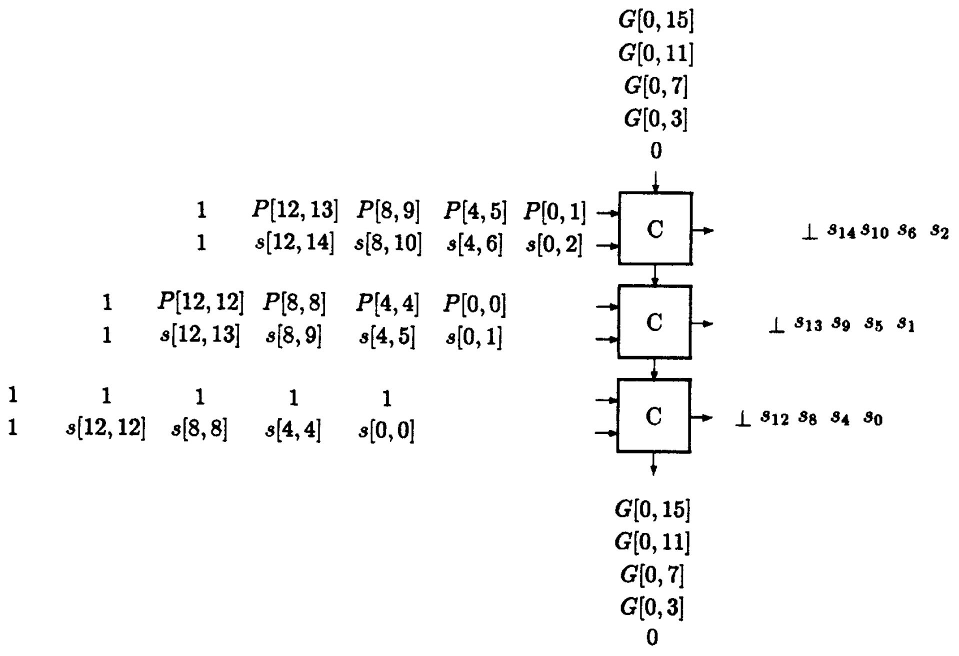


Figure 7: I/O-behaviour of the C-cells



0 a12 a8 a4 a0
1 b12 b8 b4 b0
0 a13 a9 a5 a1
1 b13 b9 b5 b1
0 a14 a10 a6 a2
1 b14 b10 b6 b2
0 a15 a11 a7 a3 0
0 b15 b11 b7 b3 0
0
1
A
A
A
B
s16 s15 s11 s7 s3
1 s[12,12] s[8,8] s[4,4] s[0,0]
1 1 1 1 1
1 s[12,13] s[8,9] s[4,5] s[0,1]
1 P[12,12] P[8,8] P[4,4] P[0,0]
1 s[12,14] s[8,10] s[4,6] s[0,2]
1 P[12,13] P[8,9] P[4,5] P[0,1]
G[0,15] G[0,11] G[0,7] G[0,3] 0

Figure 6: I/O-behaviour of the top row of cells



G[0,15] G[0,11] G[0,7] G[0,3] 0
1 P[12,13] P[8,9] P[4,5] P[0,1]
1 s[12,14] s[8,10] s[4,6] s[0,2]
C
⊥ s14 s10 s6 s2
1 P[12,12] P[8,8] P[4,4] P[0,0]
1 s[12,13] s[8,9] s[4,5] s[0,1]
C
⊥ s13 s9 s5 s1
1 1 1 1 1
1 s[12,12] s[8,8] s[4,4] s[0,0]
C
⊥ s12 s8 s4 s0
G[0,15] G[0,11] G[0,7] G[0,3] 0

Figure 8: I/O-behaviour of the array of C-cells

	General lower bound	Lower bound for time-optimal addition	$\mathcal{FASTA}_n$
$A(n)$	$\Omega(1)$	$\Omega(n)$	$O(n)$
$P(n)$	$\Omega(1)$	$\Omega(\sqrt{n})$	$O(\sqrt{n})$
$T(n)$	$\Omega(\sqrt{n})$	$\Theta(\sqrt{n})$	$O(\sqrt{n})$
$AP(n)$	$\Omega(n)$	$\Omega(n^{3/2})$	$O(n^{3/2})$
$AT(n)$	$\Omega(n)$	$\Omega(n^{3/2})$	$O(n^{3/2})$
$APT(n)$	$\Omega(n^2)$	$\Omega(n^2)$	$O(n^2)$
$AP^2(n)$	$\Omega(n)$	$\Omega(n^2)$	$O(n^2)$
$AT^2(n)$	$\Omega(n^2)$	$\Omega(n^2)$	$O(n^2)$

Table 1: Tight bounds for addition

Let $\tau_{\mathcal{FASTA}}(n)$ denote the minimum length of a basic clock cycle of $\mathcal{FASTA}_n$. The switching time of each cell is $O(1)$. Therefore, we have $\tau_{\mathcal{FASTA}}(n) = O(1)$. One can easily verify that

$$\begin{aligned} T_{\mathcal{FASTA}}(n) &= (3\sqrt{n}-3)\cdot\tau_{\mathcal{FASTA}}(n) \\ P_{\mathcal{FASTA}}(n) &= (\sqrt{n}+1)\cdot\tau_{\mathcal{FASTA}}(n)\ . \end{aligned}$$

Thus we deduce

$$\begin{aligned} T_{\mathcal{FASTA}}(n) &= O(\sqrt{n}) \\ P_{\mathcal{FASTA}}(n) &= O(\sqrt{n}) \end{aligned}$$

for the time and period of the $\mathcal{FASTA}$ hardware algorithm.

Table 1 summarizes the asymptotic behaviour of $\mathcal{FASTA}_n$ with respect to A, P, T, and the usual compound measures. Moreover, it is confronting these upper bounds with the corresponding lower bounds given in Section 2.

We derive from the entries of this table that the $\mathcal{FAST}$ adder is T-, APT-, and AT^2-optimal. In the class of time optimal adders the $\mathcal{FASTA}$ algorithm is optimal with respect to ***all*** usual measures.

Note that the computation graph uses only four types of simple cells. Due to the mesh connected array-like structure, the data flow is ***particularly*** simple and regular. The maximum number of direct neighbours of a node is determined by the leftmost A-cell and thus equals 6 (the input nodes have to be considered, too). The chosen arrangement results in ***very*** short data communication lines, the maximum wire length being bounded from above by a constant. The adder is extensively exploiting concurrency mainly through a high degree of parallelism. Moreover, the average and maximum data rates have the same magnitude. Thus we conclude that the $\mathcal{FASTA}$ hardware algorithm is purely systolic.

4 Additional Properties and Extensions

Employing a standard technique outlined in [6, p. 71], the $\mathcal{FAST}$ adders can be easily extended in order to be capable of performing addition/subtraction of integers in two's complement representation.

Each single adder $\mathcal{FASTA}_n$ is quite adaptable to varying operand lengths. Note that the computation graph of $\mathcal{FASTA}_N$, N satisfying $N < n$, appears as a subgraph of the computation graph of $\mathcal{FASTA}_n$ (see Fig. 1). Therefore, ADD_N can be computed by the computation graph of $\mathcal{FASTA}_n$ at no increase in computation time and period, when compared to the performance of $\mathcal{FASTA}_N$. Since the underlying computation principle essentially does not depend on the number of input diagonals, the computation graph of $\mathcal{FASTA}_n$ can tackle the case $N > n$, too. There exists a straightforward modification of the $\mathcal{FASTA}_n$-I/O-scheme which leads to a latency $(N/\sqrt{n}+2\sqrt{n}-3)\cdot\tau_{\mathcal{FASTA}}(n)$ and period $(N/\sqrt{n}+1)\cdot\tau_{\mathcal{FASTA}}(n)$ for the computation of ADD_N.

Besides from the criteria considered above, there exists another criterion of increasing importance for the valuation of the practical relevance of a hardware algorithm: the ***testability*** with respect to a given fault model. The testability of a hardware algorithm relates to the minimum number of input patterns needed to detect all possible faults considered by the underlying fault model. There exists a slightly modified version of the $\mathcal{FASTA}$ hardware algorithm which is ***C-testable*** with respect to the widely used ***single stuck-at*** fault model (see e.g. [11]). This means that the minimum number of test patterns needed to detect all such faults does not depend on n. It suffices to furnish each of the A-cells with an additional OR-gate and an additional flipflop in order to allow an 11 pattern test sequence to detect each possible stuck-at fault (see [8]).

The application of the carry lookahead technique does not depend on the radix r of the underlying number representation. Thus, choosing $r = 2^m$ and employing a conventional m-bit representation for the r-digits leads to a class of coarse-grained $\mathcal{FAST}$ adders in which each cell handles m-bit blocks of bit positions. Note that the A-, B-, and C-cells of these adders essentially have to perform an m-bit addition. Thus it may be appropriate to generate ***hybrid*** adders which use the purely systolic $\mathcal{FASTA}$ structure for the high level inter-node communication and a binary tree based addition scheme within the PEs.

It is well-known that carry lookahead adders are essentially a solution to the general problem of ***parallel prefix computation*** (PPC, see [10]). PPCs occur in the solution of many other problems such as the simulation of finite-state machines, linear recurrences, digital filtering, various graph problems, sorting, and others. Therefore, the $\mathcal{FAST}$ adder suggests a general paradigm for the derivation of purely systolic hardware algorithms in a wide range of application domains.

5 Conclusion

In this paper we have introduced a novel purely systolic hardware algorithm for integer addition which is well-suited for realization in integrated technologies like VLSI, ULSI, WSI, and discrete technologies as well. It is shown in [7] that the use of few types of simple cells and interconnection patterns together with a rigorous specification of the I/O-behaviour lead to a straightforward and comparatively simple proof for this parallel algorithm.

If the linear model for signal propagation delays is assumed, the $\mathcal{FASTA}$ hardware algorithm turns out to be T-, APT-, and AT^2-optimal. In the class of time optimal adders it is optimal with respect to all usual complexity measures for VLSI-algorithms. Its concrete area requirements are essentially determined by $(2\sqrt{n}-1)$ logical nodes. The overall $\mathcal{FASTA}$ area of $O(n)$ has to be compared with the lower bound $\Omega(n^2)$ for the area of very fast non-purely systolic adders (cf. [14]).

The $\mathcal{FAST}$ adder can be easily adapted to varying operand lengths. Moreover, there exist straightforward augmentations of the computation graph which render possible the processing resp. generation of non-diagonal input and output formats without changing the asymptotical properties of the $\mathcal{FASTA}$ algorithm. Besides from these properties, this new class of adders is especially advantageous with respect to testability aspects.

Since the suggested algorithm allows the generation of optimal adders at low costs and, on the other hand, gives a basic principle for the purely systolic solution of many other problems, the $\mathcal{FASTA}$ hardware algorithm seems to be of significant practical and theoretical relevance.

Future work should include the practical realization of the $\mathcal{FAST}$ adder. Moreover, the testability analysis should be extended to more general fault models. The incorporation of fault tolerance mechanisms is another important topic for future research.

Due to lack of space, this paper merely outlines the concept of the $\mathcal{FAST}$ adder. For a complete description of all details and a proof of correctness see [7]. [8] contains a detailed and systematic treatment of purely systolic hardware algorithms for parallel prefix computations.

Acknowledgements

I am especially indebted to Dr. H. Schmeck for many valuable discussions and suggestions. Moreover, I wish to thank Prof. Dr. W. Thomas and my colleagues of the VLSI group in Kiel for their support. My thanks also go to the anonymous referees. Their comments have helped to improve the presentation of the paper.

References

[1] B. Becker, R. Kolla. On the construction of optimal time adders. In *Proc. STACS 88*, pages 18–28, Springer-Verlag 1988. LNCS 294.

[2] R. P. Brent, H. T. Kung. A regular layout for parallel adders. *IEEE Trans. Computers*, C-31(3):260–264, 1982.

[3] B. Chazelle, L. Monier. Optimality in VLSI. In J. P. Gray, ed., *VLSI 81*, pages 269–278, London, 1981. Academic Press.

[4] B. Chazelle, L. Monier. A model of computation for VLSI with related complexity results. *J. ACM*, 32(3):573–588, 1985.

[5] M. J. Foster, H. T. Kung. The design of special-purpose VLSI chips. *IEEE Computer*, 13(1):26–40, 1980.

[6] K. Hwang. *Computer Arithmetic. Principles, Architecture, and Design.* Wiley, New York, 1979.

[7] L. Kühnel. Optimal purely systolic addition. Technical Report 9002, Inst. f. Informatik, Universität Kiel, W-2300 Kiel 1. Germany, April 1990.

[8] L. Kühnel. Optimale systolische Präfixberechnungen. Dissertation. In preparation, 1991.

[9] H. T. Kung. Why systolic architectures? *IEEE Computer*, 15(1):37–46, 1982.

[10] R. E. Ladner, M. J. Fischer. Parallel prefix computation. *J. ACM*, 27(4):831-838, 1980.

[11] P. K. Lala. *Fault Tolerant and Fault Testable Hardware Design.* Prentice-Hall, London, 1985.

[12] H. Schmeck. Modellierung und Bewertung von VLSI-Algorithmen. Habilitationsschrift. Institut für Informatik. Universität Kiel, 1989.

[13] J. Sklansky. Conditional-sum addition logic. *IRE Trans. Electr. Comp.*, EC-9(2):226–231, 1960.

[14] B. Sugla, D. A. Carlson. Extreme area-time tradeoffs in VLSI. *IEEE Trans. Computers*, 39(2):251–257, 1990.