

Pipelined Adders

Luigi Dadda, *Member, IEEE*, and Vincenzo Piuri, *Member, IEEE*

Abstract—A well-known scheme for obtaining high throughput adders is a pipeline in which each stage contains an array of half-adders performing a carry-save addition. This paper shows that other schemes can be designed, based on the idea of pipelining a serial-input adder or a ripple-carry adder. Such schemes offer a considerable savings of components while preserving high throughput. These schemes can be generalized by using (p, q) parallel counters to obtain pipelined adders for more than two numbers.

Index Terms—Adders, high-speed adders, high-throughput adders, pipelined computation, skewed arithmetic.

1 INTRODUCTION

ADDING two binary numbers is a basic operation in any electronic processing system: It has received much attention and has been solved by using several approaches and architectures. In particular, in the case of bit-parallel structures, a wide spectrum of solutions is available: from the simple ripple-carry to the faster schemes of carry-look-ahead, conditional-sum, or carry-skip adders [2], [7]. The first approach is used when no severe constraint is imposed by the application on the operation latency, while the other solutions are usually adopted to achieve both high throughput and small latency.

In special-purpose computing systems (e.g., in some signal and image processing applications), dedicated adders are often required to have high throughput while constraints on latency are not so severe. In such cases, it may be convenient to adopt architectures that are less sophisticated than carry-look-ahead or conditional-sum structures. Pipelined architectures composed by stages of carry-save adders are the most widely [2], [7].

In this paper, we discuss the optimum design of pipelined adders with respect to specific constraints on throughput. Minimization of the circuit complexity (i.e., of the silicon area used by the integrated implementation) and latency are also considered by exploiting the computational characteristics of the pipeline granularity.

Two approaches to the design methodology are presented. The first one is based on the analysis of carry propagation in a ripple-carry parallel adder. The second one is derived by unrolling the scheme that is traditionally used for bit-serial addition. A considerable savings of components is obtained with respect to the traditional structure, while throughput is preserved. All architectural approaches are evaluated with respect to circuit complexity, throughput, and latency, to provide the basic guidelines for optimum design of pipelined adders: The traditional gate-count approach is used to obtain a high-level evaluation, independently from the specific inte-

gration technology. Optimization of the adder's features is then considered by exploiting the characteristics of fast adders' schemes instead of the traditional ripple-carry adder.

Our approaches can be generalized and applied to all arithmetic units in which the nominal operation can be described according to one of the two approaches discussed here for addition, i.e., whenever the computation may be defined in a serial way and unrolled, or whenever there is a unidirected computational wavefront in a bit-parallel arithmetic structure. A simple example is given by multi-operand adders when (p, q) parallel counters are used [1].

2 THE TRADITIONAL PIPELINED SCHEME FOR PARALLEL ADDITION

The traditional pipelined architecture for parallel addition of two n -bits numbers is well known in the literature [1], [2], [3], [5], [6], [7]: It is based on the carry-save addition scheme presented in Fig. 1a. The arithmetic operation of this circuit is conveniently described by using the notation introduced in [1] for full- and half-adders: The corresponding *arithmetic diagram* is shown in Fig. 2a. Each stage is composed by a linear array of half adders (HA) performing a carry-save addition (see Fig. 1a); adjacent stages of the pipeline are separated by latches (FF). The origin of this scheme can be traced down (e.g., see [2], [7]) to Braun's array multiplier, which is based on carry-save addition of successive rows in the multiplier array by using linear arrays of n full adders.

In Figs. 1a and 2a, in the case of natural operands, it is worth noting that the column producing $s_{n+1}(s_n)$ is usually implemented by HAs [2], [7] instead of two-input OR gates; OR gates can be adopted because only a single non-zero carry from the preceding s_n column can be generated. Similarly, in the case of integer operands, XOR gates are used instead of the OR gates.

The structure presented in Fig. 1a produces the final sum in a skewed form, starting from the least significant bit. An array of latches must be introduced to provide a bit-parallel output format by deskewing the adder output: They constitute a triangular array that fills the bottom-right part of Fig. 1a. For simplicity, they are not shown in our figures.

• The authors are with the Department of Electronics and Information, Politecnico di Milano, Piazza Leonardo da Vinci 32, I-20133 Milano, Italy.
E-mail: dadda@elet.polimi.it.

Manuscript received Apr. 11, 1994; revised Dec. 20, 1994.

For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number C95077.

Moreover, in several applications, the result generated by an individual adder or multiplier (e.g., in inner product units) is used for further arithmetic computations that can be implemented more efficiently when operands are in the skewed form, while deskewing is performed only on the final result of the whole computation.

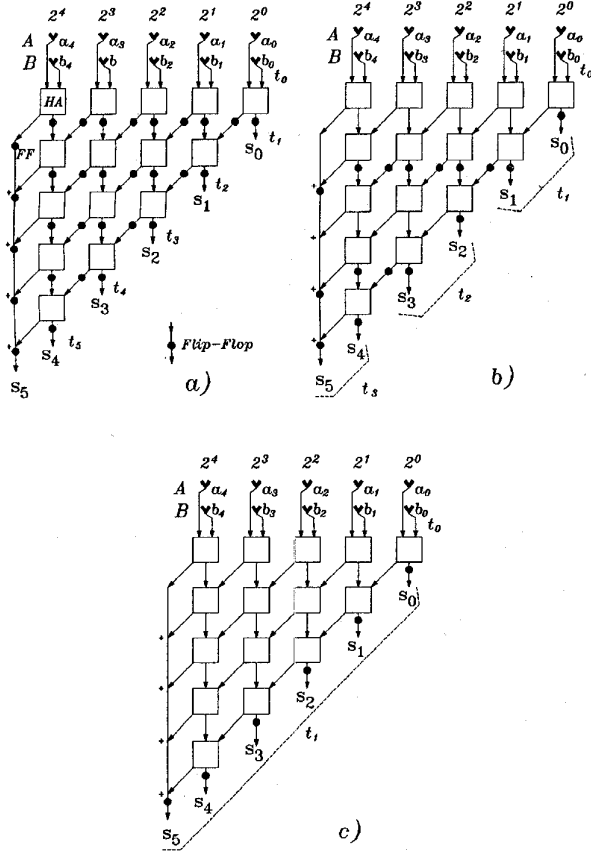


Fig. 1. Pipelined adders for two binary numbers of $n = 5$ bits, composed by carry-save adders, with different granularity: a) $g = 1$, b) $g = 2$, and c) $g = 5$.

In order to compare different schemes, we summarize here the complexity analysis of the schemes shown in Fig. 1. Since we are concerned with the core architecture of the adder, we consider neither the input latches storing the input operands nor the output latches holding the result.

The circuit complexity C of the traditional architecture is

$$C = (n+1) \frac{n}{2} C_{HA} + (n^2 - 1) C_{FF},$$

where C_{HA} and C_{FF} are the complexities of half adders and latches, respectively (we neglect the OR gates which generate the most-significant bit of the product).

The clock cycle τ must be long enough to execute one step of the pipelined addition algorithm and to store the results into the latches between stages; therefore, τ is equal to $\tau_{HA} + \tau_{FF}$ where τ_{HA} and τ_{FF} are the latencies of half adders and latches, respectively. The throughput F is given by $F = 1/\tau$.

The scheme of Fig. 1a achieves the maximum throughput for the adopted implementation technology. If the

throughput F imposed by the application is small enough so that more than one linear array of half adders can operate within a single clock cycle, we can collapse these linear arrays into the same stage of the pipelined architecture. In other words, when $g\tau_{HA} + \tau_{FF} \leq F^{-1}$ for a given g ranging in $[1, n]$, g pipeline stages can be collapsed into only one stage, composed by a trapezoidal array of half adders. We call g the *granularity* of the pipelined architecture.

The pipeline granularity of the solution presented in Fig. 1a is the minimum value ($g = 1$). Fig. 1b shows the case of two carry-save stages per pipeline stage ($g = 2$). Fig. 1c shows the case of maximum granularity ($g = n$), i.e., the case of no-pipelining; this circuit is purely combinational and operates as a ripple-carry adder, even if it is rather unconventional and highly expensive. The arithmetic diagrams for these cases are given in Fig. 2b and in Fig. 2c, respectively.

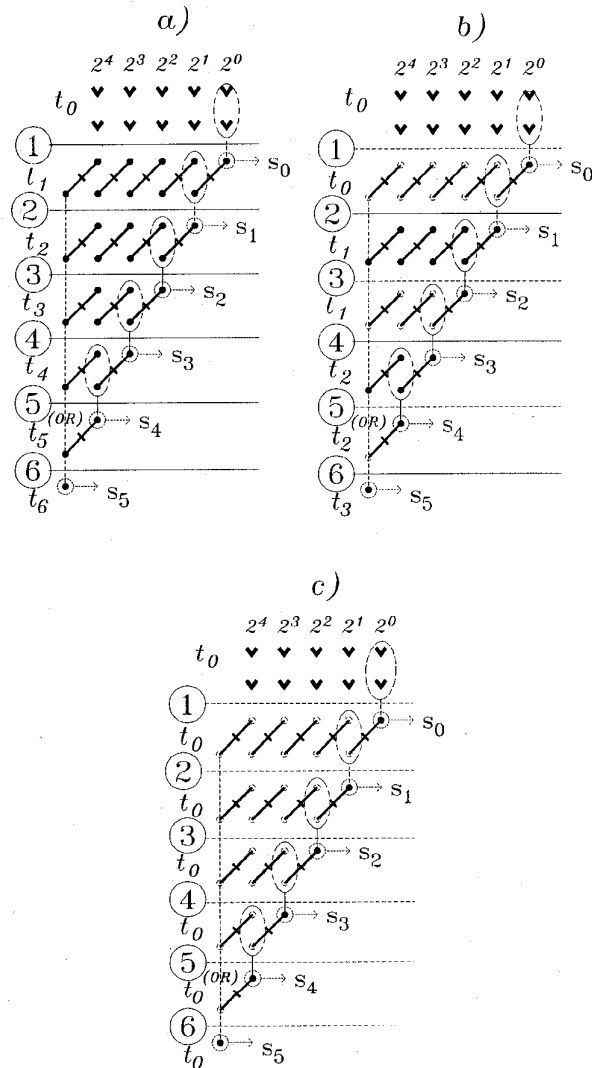


Fig. 2. Arithmetic diagrams of the schemes present in Fig. 1.

The circuit complexity of these architectures can be derived as in the basic structure. In case of $g = 2$, it is:

$$C = (n+1) \frac{n}{2} C_{HA} + (n+1) \left(\frac{n}{2} - 1 \right) C_{FF}.$$

In case of n divisible by g , it is

$$C = (n+1) \frac{n}{2} C_{HA} + (n+1) \left(\frac{n}{g} - 1 \right) C_{FF},$$

while in the general case, it is

$$C = (n+1) \frac{n}{2} C_{HA} + \left\lceil \frac{n}{g} \right\rceil \left(2n - g \left\lceil \frac{n}{g} \right\rceil - g \right) + \left\lceil \frac{n}{g} \right\rceil - 1 C_{FF},$$

with $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ being the floor and the ceiling functions, respectively.

The clock cycle τ is equal to $g \tau_{HA} + \tau_{FF}$; the adder's latency L is $\left\lceil \frac{n}{g} \right\rceil \tau$, while the throughput F is still $1/\tau$.

In Fig. 6a, the circuit complexity C is shown for different values of the pipeline granularity g : The cases of the traditional architectures are drawn in continuous line and are labelled by T1, T4, and Tn, for granularity g equal to 1, 4, and n , respectively. For the same granularities, the clock cycle τ and the throughput F are given in Figs. 6c and 6e, respectively. The latency is not shown since it is simply a multiple of the clock cycle. The actual value of these figures of merit for a given n is related to the specific implementation adopted for the basic components (half adder, full adder, latch), for which a traditional design has been considered (see [2], [7]).

3 ANOTHER DESIGN METHOD FOR PIPELINED ADDERS

An alternative approach to derive the structure of pipelined adders can be obtained by analyzing the computation in a ripple-carry adder; Fig. 3a shows the arithmetic diagram of this adder. Let us partition the diagram of the addition by separating the columns associated with the individual bit weights. Consider first the section $a-a$, separating the bits having weight 2^0 from bits at weight 2^1 . The first slice (corresponding to the weight 2^0) can be pipelined to the rest of the scheme by delaying (through latches) both the carry generated by the half adder in such a slice and all the bits of the operands that are on the left side of section $a-a$ (having weight greater than 2^0). Then, we repeat this operation for the adjacent sections on the left side, until the slice corresponding to the weight 2^n is reached.

The complete arithmetic diagram describing this design method is shown in Fig. 4a: The circuit implementing this approach can be derived directly as shown in Fig. 5a. In Fig. 4a, the three bits in each stage (two bits in the first one) are circled to mean that they are the inputs of the full adder associated to that stage (of the half adder in the first one). There is one full- or half-adder in each slice. One bit of the final result is generated in each pipeline stage and output from the corresponding slice; as in the traditional architecture, the result bits are produced in a skewed form.

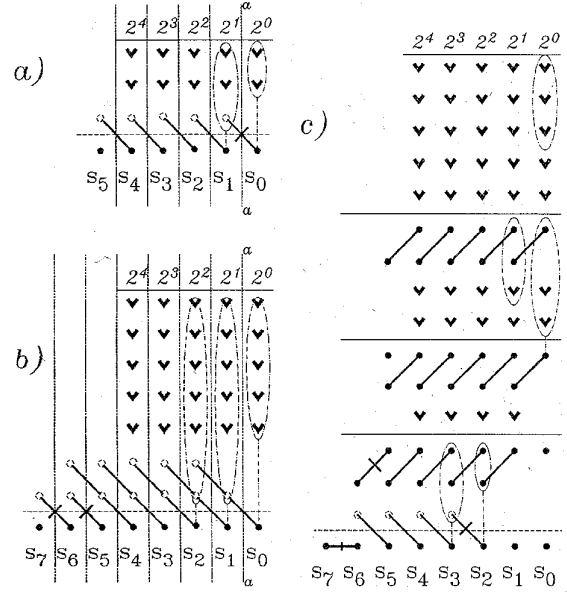


Fig. 3. Arithmetic diagrams for multiple-input addition. a) The arithmetic diagram of a ripple-carry adder for two 5-bit numbers; b) the arithmetic diagram of a ripple-carry adder for five 5-bit numbers composed by (7; 3) parallel counters, and c) the arithmetic diagram of the adder for five 5-bit numbers composed of a cascade of carry-save adders (which reduces the number of operands progressively from five to only two addends) and the final ripple-carry adder (which produces the final sum).

By comparing this circuit with the scheme of Fig. 2a, reduction of circuit complexity is evident: a number of adders have been removed, while most of the remaining half adders are replaced by full adders.

The circuit complexity of this architecture is

$$C = C_{HA} + (n-1) C_{FA} + (n^2 - 1) C_{FF}$$

where C_{FA} is the complexity of the full adder.

To allow the correct operation of the architecture, the clock cycle τ must be not smaller than the maximum value between $\tau_{FA}S$ and $\tau_{FA}C + \tau_{FF}$, where $\tau_{FA}S$ and $\tau_{FA}C$ are the latencies of the sum bit and the carry bit in the full adder, respectively (usually, it is $\tau_{FA}S < \tau_{FA}C + \tau_{FF}$). Also, for this architecture, the throughput F is $1/\tau$, since one result is produced at each clock cycle, while the adder's latency L is $n \tau$.

As in Section 2, we can consider a larger pipeline granularity to reduce the latency. In the case of $g = 2$, we obtain the arithmetic diagram of Fig. 4b and the scheme of Fig. 5b. The first slice contains one half adder generating the least significant sum bit and one carry bit transmitted to the adjacent slice without latching since it belongs to the same pipeline stage. The full adder of the second slice generates the sum bit having weight 2^1 and the carry bit for the subsequent stage. Both sum bits are output. In Fig. 4b, the operands' bits above the horizontal line ① are the input bits of the first linear array of the first stage in Fig. 5b; in Fig. 4b, bits between the horizontal lines ① and ② are dotted since they belong to the same pipeline stage of the previous section and are not latched (i.e., they are only "virtually" stored). The output bits generated by the first pipeline stage are then latched before entering the second stage; in Fig. 4b, bits between the horizontal lines ②

and ③ are filled since they are actually stored in latches. The same analysis can be performed for the other sections in Fig. 4b. Dashed horizontal lines separate the operations performed within the same pipeline stage, while continuous lines give the boundaries between subsequent stages.

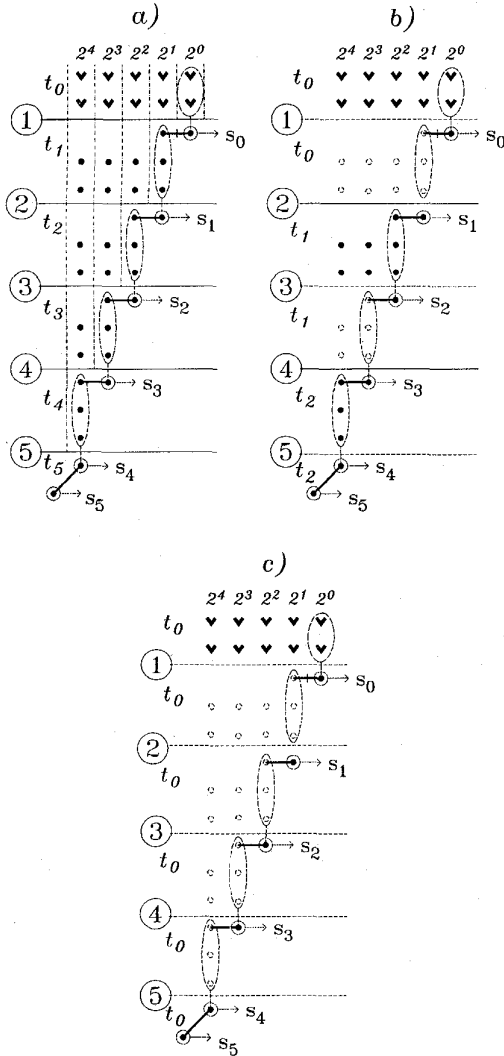


Fig. 4. Arithmetic diagrams of schemes for pipelined adders based on pipelining the computation of a ripple-carry adder with granularity a) $g = 1$, b) $g = 2$, c) $g = 5$.

The arithmetic diagram and the circuit scheme for the case of maximum granularity ($g = n$) are given in Fig. 4c and in Fig. 5c, respectively; it is straightforward to note that the pipelined adder degenerates into a traditional ripple-carry adder.

In the general case of granularity g , each stage of the pipelined architecture contains a ripple-carry adder and a number of latches. The ripple-carry adder is composed of g full adders to generate the g least significant sum bits associated to such stage. The latches are used to propagate the unused input bits and the inter-stage carry to the subsequent stage. This approach has been adopted in prototype implementations for the application discussed in [4].

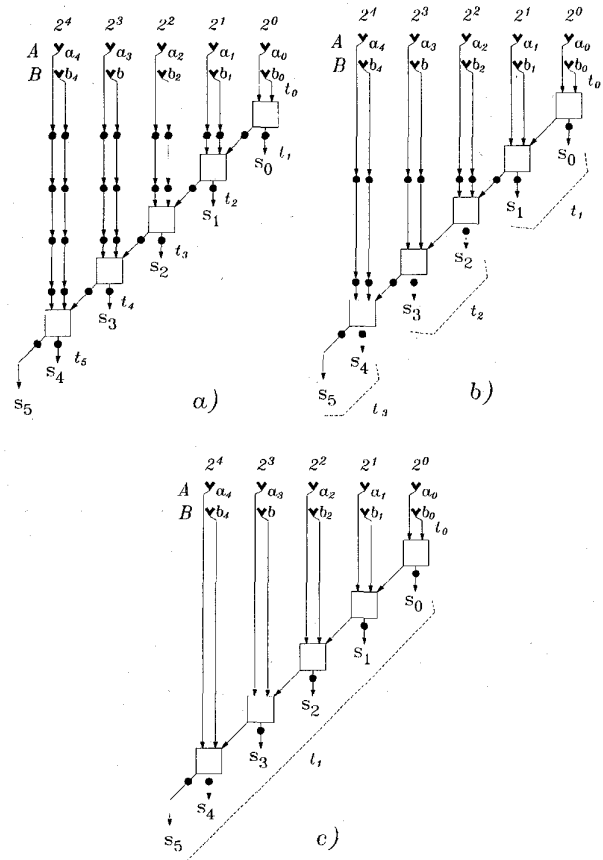


Fig. 5. Pipelined adders corresponding to the arithmetic diagrams of Fig. 4.

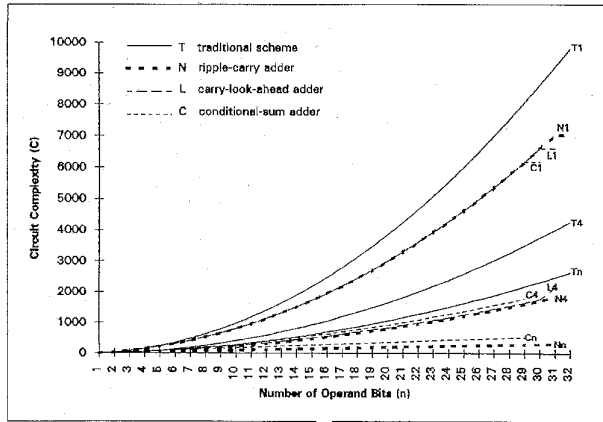
The circuit complexity of this architecture is

$$C = C_{HA} + (n-1)C_{FA} + \left\lceil \frac{n}{g} \right\rceil \left(2n - g \left\lceil \frac{n}{g} \right\rceil - g \right) + \left\lceil \frac{n}{g} \right\rceil - 1 C_{FF}.$$

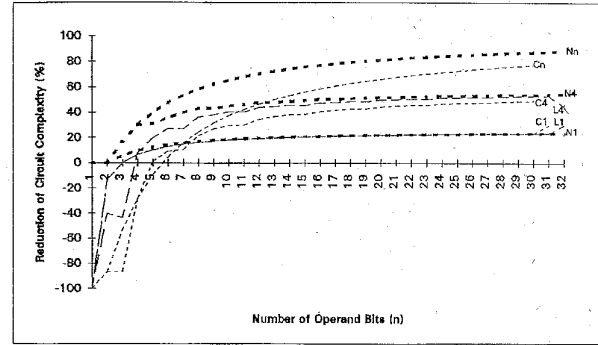
The clock cycle τ is $\tau = (g-1)\tau_{FA} + \max(\tau_{FA}, \tau_{FA} + \tau_{FF})$; usually, it is $\tau = g\tau_{FA} + \tau_{FF}$. However, the minimum value of the clock cycle, which allows completion of the nominal operations in each stage, may be smaller than the one given above since, in specific implementations, some operations in the chain generating the inter-stage carry may be performed in parallel; the value given above is therefore an upper bound for the actual clock cycle.

The adder's latency L and the throughput F are given by the same formulas given for the case of the traditional pipelined architectures with granularity g (however, the actual values are slightly different since the new expression for the clock cycle must be considered).

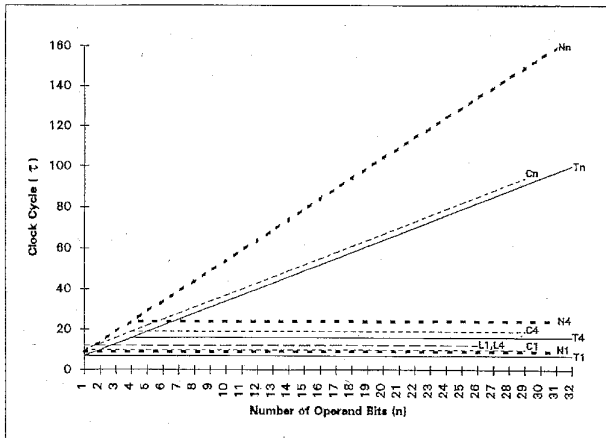
The circuit complexity C , the clock cycle τ , and the throughput F are shown in Figs. 6a, 6c, and 6e, respectively, for different values of the pipeline granularity g . The cases of the novel design schemes are drawn in thick-dashed line and are labelled by N1, N4, and Nn, for granularity g equal to 1, 4, and n , respectively. The percentage reduction of such figures of merit with respect to the traditional architecture having the same granularity is shown in Figs. 6b, 6d, and 6f, respectively. Also in this case, the latency is not shown since it is simply a multiple of the clock cycle.



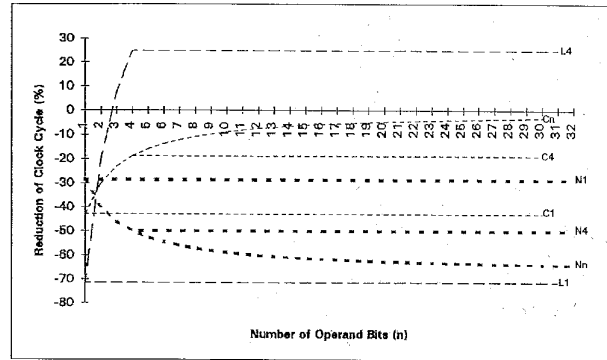
a)



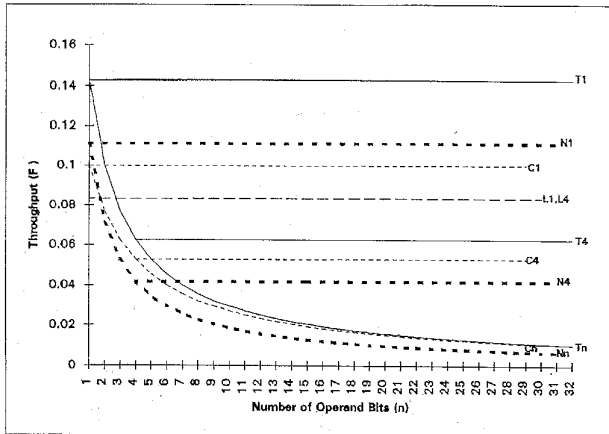
b)



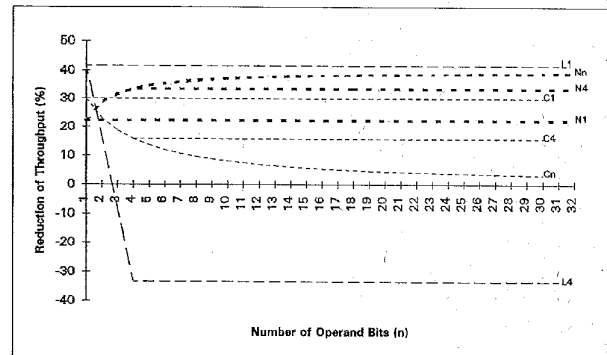
c)



d)



e)



f)

Fig. 6. Evaluation of the optimized pipeline adders for different pipeline granularities g versus the number n of operand bits: a) circuit complexity C ; b) percentage reduction of C with respect to the traditional architecture. The traditional architectures are labeled by T1, T4, and Tn, for the granularities g equal to 1, 4, and n , respectively. The architectures described in Section 3 are labeled by N1, N4, and Nn; carry-look-ahead adders of Section 5 are labeled by L1 and L4, while conditional-sum adders are identified by C1, C4, and Cn; c) clock cycle τ ; d) percentage reduction of τ ; e) throughput F ; f) percentage reduction of F .

The use of the novel approach is always very convenient with respect to the traditional one by considering the circuit complexity: For $n > 4$, the area reduction ranges from 10% to more than 80%. Conversely, the clock cycle and the throughput are worse than in the traditional case: in fact, half adders are used in the linear array for the traditional case, while they are replaced by full-adders (slower than half-adders) in the novel approach. The increase of the clock cycle and, as a consequence, the latency ranges from 30% to 60%, while throughput reduction ranges about from 20% to 40%.

The same optimized structures of pipelined adders may be obtained by using another approach based on unrolling and pipelining a traditional serial-input adder. In such an adder (see Fig. 7), the addends are stored into two shift registers: addition is performed by a single full adder, providing a delay in the feedback loop from the carry output to the carry input. At each iteration, a new sum bit and a new carry bit are generated starting from the least significant ones, while addends are shifted to the right so that the correct operands' bits are presented to the arithmetic unit.

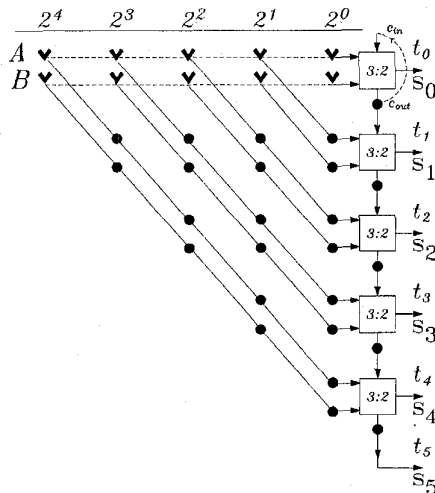


Fig. 7. The pipelined adder for the arithmetic diagram of Fig. 4a, obtained by unrolling the computation performed by a bit-serial adder.

Unrolling this architecture can be obtained by means of the following operations. First, we "photograph" the computation and the distribution of operands within the architecture itself. In each photograph, the computation consists in a three-bit (two inputs and one carry input) addition with carry output generation; addends are progressively reduced one bit at a time. Second, we associate an individual digital structure to the computation performed in each photograph. One full adder is used in each stage as arithmetic unit; one storage device is used to delay the carry output for the subsequent stage, while two registers are required to store the unused part of the addends (their length decreases by one bit at each stage). Third, the digital structures are properly cascaded to propagate the computation and the operands as required by the algorithm. At the k th stage, the addends' bits at weight 2^k are added with the carry input generated at stage $(k - 1)$; all addends' bits at

weight higher than 2^k are propagated to the subsequent stage. The final circuit coincides with the one shown in Fig. 5a.

In a third approach, the circuit operation can be described by the following operations: First of all, addends are transformed in the skewed form by the vertical shift registers, and then they are added by a traditional ripple-carry adder in which carries are latched between full adders to guarantee the correct data timing.

4 MULTIOPERAND ADDITION USING PARALLEL COUNTERS

All schemes presented in the previous sections for the case of two-input addition are based on half and full adders. These may be viewed (e.g., see [1]) as parallel counters having two or three input bits, respectively, of the same arithmetic weight, and two output bits (one of which has the same weight of the inputs, while the other has twice such a weight).

The architectures discussed above can be generalized to deal with the case of three or more addends. For example, in the case of five 5-bits addends, the arithmetic diagram presented in Fig. 3a for the ripple-carry parallel addition of two operands can be generalized as shown in Fig. 3b for five addends. The k th slice corresponds to the bit weight 2^k ; $p_k = 7$ input bits are present in each slice of the same arithmetic weight 2^k (one for each input addend plus the carries from the slices at arithmetic weight smaller than k). The number of 1s in the k th set of $p_k = 7$ bits is a binary natural number which can be represented with $q_k = 3$ bits ($p_k < 2^{q_k}$); the weight of the i th bit is 2^{k+i} ($i = 0, 1, \dots, q_k$).

The number of 1s can be computed in the k th slice by using a $(p_k; q_k)$ parallel counter. In the least significant slice of the adder, there are as many operands' bits as the addends; in the example of Fig. 3b, they are five and thus we need a $(5; 3)$ parallel counter. In the subsequent slices, higher order parallel counters are required since carries must taken into account. In the second slice of the example a $(6; 3)$ counter is required, while a $(7; 3)$ counter must be adopted for the subsequent three slices. The slices at weight higher than 2^{n-1} need smaller-order parallel counters since only carries must be treated. In the example, two $(3; 2)$ and one $(2; 2)$ parallel counters are required. The architecture corresponding to this arithmetic diagram is shown in Fig. 8: It has been derived by applying the same reasoning adopted for the two-operands case.

A $(p; q)$ parallel counter can be implemented by a network of half and full adders [7]. The use of dedicated circuits implementing the $(p; q)$ parallel counters instead of such a network may lead to an architecture with higher performance and lower circuit complexity. Customized design strategies and structures can be adopted to optimize the scheme of specific parallel counters; parallel counters with $p > 3$ and composite counters (compressors) have been recently proposed for multipliers [8], [9].

A different approach to multiple-operand addition is based on pipelining three-operand additions, as is shown in Fig. 3c. In this case, each pipeline stage performs the carry-

save addition of three operands at most. Three of the initial (five in the example of Fig. 3c) operands are transformed into one row of sum bits and one row of carries by the first stage of full-adder operators, while the other initial operands are propagated to the subsequent stage. The second stage considers therefore four operands. Again, a stage of full-adder operators transforms the stage's operands into the sum bits and carry bit for the third stage, while the remaining initial operand is propagated. Operands are progressively reduced through the pipeline (one for each stage) to only two operands; the last stage of the adder can thus be implemented by using the two-operands adder of Fig. 3a. Higher order counters may be used, obtaining smaller latency and throughput.

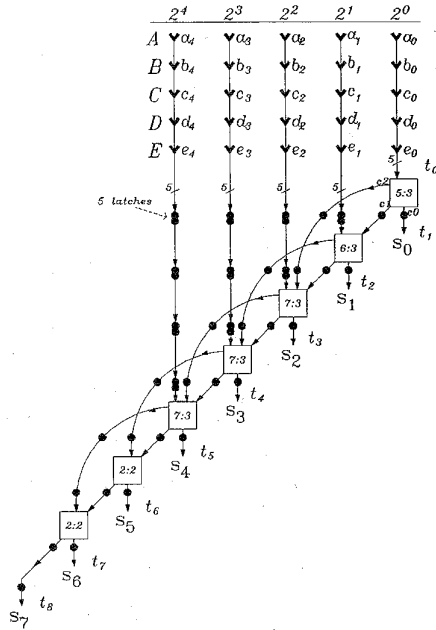


Fig. 8. A pipelined adder for five 5-bit numbers, obtained by pipelining the ripple-carry adder of Fig. 3b.

5 PIPELINED ADDERS WITH FAST ADDERS

In Section 3, we have showed that, when throughput is smaller than $(\tau_{FA} + \tau_{FF})^{-1}$, it is possible to reduce the circuit complexity and the latency by increasing the pipeline granularity g , i.e. by collapsing several arithmetic operators into the same pipeline stage. In such stages, addition over several bits (namely, over g bits) is performed by a ripple-carry adder of length g .

To increase throughput we can reduce the clock cycle by replacing ripple-carry adders with faster parallel adders [2, 7] having smaller latency over the same number g of the operands' bits. Circuit complexity will be obviously increased according to the adopted adder architecture.

The use of fast adders may be exploited also to reduce the circuit complexity by increasing the pipeline granularity. In fact, for a given clock cycle, we can increase the number of bits that can be added during the same cycle, i.e., the granularity. This reduces the number of latches since fewer operands' bits must be propagated through the

pipeline and, possibly, reduces the number of pipeline stages (in this last case, also the latency is decreased). Again, complex fast adders will require a higher circuit complexity.

The optimum solution is therefore related to the specific application and to the actual implementation constraints, since circuit complexity, latency, and throughput are conflicting characteristics that must be balanced.

A first solution is based on the use of carry-look-ahead adders. In the k th stage, the carry-in signal (coming from the $(k-1)$ th stage) and all the g operands' bits are treated in parallel to compute the carry-generate signals and the carry-propagate signals for each position from the weight 2^{kg} to the weight $2^{(k+1)g-1}$. The sum bits are then computed from these signals in the corresponding weights; the carry-out signal that must be delivered to the subsequent stage is derived from the above signals at the same time.

The clock cycle τ is greatly decreased with respect to the previous cases: $\tau = \tau_{CLA}(g) + \tau_{FF}$, where $\tau_{CLA}(g)$ is the latency of the carry-look-ahead adder of length g . Note that it is quite independent from the granularity g : Since only the carry-look-ahead circuit—a two-level combinational structure—spans on all the g bits, its latency is loosely related to g by the fan-in of its gates. Also in this case, the latency L is $\left\lceil \frac{n}{g} \right\rceil \tau$, while the throughput F is $1/\tau$.

The clock cycle and the throughput are shown in Figs. 6c and 6e; the novel design schemes with carry-look-ahead adders are drawn in thin-dashed line and are labelled by L1 and L4, for granularity g equal to 1 and 4, respectively. The percentage reduction of these figures of merit with respect to the traditional architecture having the same granularity is shown in Figs. 6d and 6f. Even for small granularities ($g \geq 3$), the clock cycle is reduced while the throughput is increased both with respect to the traditional solutions and to the novel ones with ripple-carry adders, when the number of operands' bits is at least equal to the granularity. For example, for $g = 4$ and $n \geq 4$, τ is reduced by about 25% with respect to the traditional architecture and 50% with respect to the novel ripple-carry solution; F is increased by about 35% and 50%, respectively. For $g = 2$, the timing performances of the carry-look-ahead solution are better than the ripple-carry approach, but are worse than the traditional structure; for $g = 1$, the architecture based on carry-look-ahead adders is the worst.

The circuit complexity is given by

$$C = \left\lceil \frac{n}{g} \right\rceil C_{CLA}(g) + \left(\left\lceil \frac{n}{g} \right\rceil - \left\lceil \frac{n}{g} \right\rceil \right) C_{CLA}(\left\lceil \frac{n}{g} \right\rceil) + \left\lceil \frac{n}{g} \right\rceil \left(2n - g \left\lceil \frac{n}{g} \right\rceil - g \right) + \left\lceil \frac{n}{g} \right\rceil - 1 C_{FF}$$

where $C_{CLA}(x)$ is the circuit complexity of the carry-look-ahead adder of length x , and $\left\lceil \frac{n}{g} \right\rceil$ is the number of operands' bits in the last stage. The circuit complexity is shown in Fig. 6a; its percentage increase with respect to the traditional solution is given in Fig. 6b. The solutions based on carry-look-ahead adder and on ripple-carry adder have approximately the same complexity: therefore, for $g > 1$, the

first one can be effectively used to enhance the clock cycle and the throughput with respect to the structure based on ripple-carry adders. However, when the number n of the operands' bits is less than four, this approach is not suited since the circuit complexity is higher than in the traditional scheme; in fact, for such values of n , the complexity increase of carry-look-ahead circuits exceeds the complexity saving due to elimination of several adders of the traditional scheme.

A second approach is based on conditional-sum adders of length g in each pipeline stage. Consider the k th stage. The sum bit of weight 2^{kg} and the corresponding carry are generated by a full adder from the operands' bits at weight 2^{kg} and from the carry signal produced by the $(k-1)$ th stage. Two dedicated adding circuits are used to generate all possible values of the sum bit at weight 2^{kg+1} and of the corresponding carries, according with the possible values (0 and 1, respectively) of the carry generated at the 2^{kg} position; these circuits are full adders with a fixed value of the carry-in signal. The actual values of the sum and carry bits are selected by multiplexers controlled by the actual value of the carry signal generated at weight 2^{kg} . Similar circuits are used also for each of the other bits from the weight 2^{kg+2} to the weight $2^{(k+1)g-1}$. The carry bit value selected at the weight $2^{(k+1)g-1}$ is delivered at the subsequent stage as carry-in signal.

The computation of the possible values of the sum bits and of the carry bits is performed in parallel. Selection of the actual values that must be delivered as final outputs is performed sequentially within the individual stage, from the least significant bit towards the most significant one of the conditional-sum adder.

The clock cycle is thus given by $\tau = \tau_{CSA}(g) + \tau_{FF}$, where $\tau_{CSA}(g)$ is the latency of the conditional-sum adder of length g . Also $\tau_{CSA}(g)$ is quite independent from the pipeline granularity g since it is given by $\tau_{CSA}(g) = \tau_{FA} + (g-1)\tau_M$, where τ_M is the latency of the two-inputs multiplexer. The latency and the throughput are given by the same formulas discussed for the other cases, even if the actual values are different since the clock cycles are different. Also in this case, the clock cycle and the throughput are shown in Figs. 6c and 6e; the examples of the novel design using conditional-sum adders are drawn in dotted line and are labelled by C1, C4, and Cn, for granularity g equal to 1, 4, and n , respectively. The percentage reduction of these figures of merit with respect to the traditional architecture is shown in Figs. 6d and 6f, respectively. Again, we consider the same specific implementation of the basic units (adders and latches) adopted in Section 3, to give typical shapes of these characteristics.

For granularities higher than 1, τ is increased with respect to the traditional approach much less than in the case of the ripple-carry solution. On the contrary of this last case, the clock cycle increase is smaller at high granularities: it tends to less than 10% for granularity equal to n , while it is about 60% in the ripple-carry case. However, τ is worse in the conditional-sum adders than in the carry-look-ahead solutions (e.g., it is decreased by 60% for $g=4$). Similarly, F is reduced much less than in the ripple-carry case (e.g., about 15% for $g=4$, and less than 10% for $g=n$), but it is worse than the carry-look-ahead case (e.g., it is 35% less for $g=4$).

For granularity equal to 1, the traditional solution has better performances; neither the carry-look-ahead approach nor the conditional-sum adders are capable to exploit their intrinsic computational parallelism, while the ripple-carry technique uses slower adders than the traditional one.

The circuit complexity is

$$C = \left\lfloor \frac{n}{g} \right\rfloor C_{CSA}(g) + \left(\left\lfloor \frac{n}{g} \right\rfloor - \left\lfloor \frac{n}{g} \right\rfloor \right) C_{CSA}(n|g) + \left\lfloor \frac{n}{g} \right\rfloor \left(2n - g \left\lfloor \frac{n}{g} \right\rfloor - g \right) + \left\lfloor \frac{n}{g} \right\rfloor - 1 C_{FF},$$

where $C_{CSA}(x)$ is the circuit complexity of the conditional-sum adder of length x . It can be easily shown that $C_{CSA}(x) = C_{FA} + 2(x-1)C_{HA} + 2(x-1)C_M$, where C_M is the circuit complexity of the two-input multiplexer.

The circuit complexity is shown in Fig. 6a and its percentage increase with respect to the traditional solution is given in Fig. 6b. For the considered specific implementations of the basic units, the complexity reduction is worse than in the cases of ripple-carry and carry-look-ahead adders. As the scheme based on carry-look-ahead adders, also this approach induces a complexity increase for small value of n , since the advantage in the simplification of the linear array of adders in each pipeline stage is vanished by the high complexity of the conditional-sum adders.

Even if the conditional-sum adders enhance both the clock cycle and the throughput with respect to the ripple-carry adders for $g > 1$, they are not as effective as the carry-look-ahead adders. Therefore, the use of carry-look-ahead adders is preferred to the conditional-sum adders.

6 DESIGN GUIDELINES AND CONCLUDING REMARKS

A traditional pipelined adder scheme (based on carry-save additions) has been first recalled in order to determine its complexity, throughput, and latency. A new scheme for the pipelined adder has been obtained by analyzing the standard ripple-carry adder or, equivalently, the bit-serial adder; this scheme requires far fewer components than the traditional one. The approach has been also extended to the case of multi-operand pipelined adders and can be generalized to any arithmetic unit whenever computation may be defined in a serial way and unrolled, or whenever there is a unidirected computational wavefront in a bit-parallel arithmetic structure.

A scheme for a given pipeline granularity has been developed to obtain a further saving of circuit complexity by reducing the number of latches; this scheme uses a short ripple-carry adder to generate the output bits of each pipeline stage. These adders can be replaced by faster schemes (carry-look-ahead or conditional-sum adders) allowing for higher throughput and smaller latency or, alternately, for higher granularity.

A detailed analysis of the proposed schemes has been developed to provide general design guidelines. The traditional structure has the highest circuit complexity, while the other solutions have approximately the same complexity (the novel architecture with ripple-carry adders is slightly

better than the others). For the granularity equal to 1, the traditional architecture has the minimum clock cycle, the minimum latency, and the maximum throughput; for all the other granularities, the solution that provides complexity reduction at the best latency and throughput is the novel architecture with carry-look-ahead adders. For all architectural approaches, we can decrease the circuit complexity by increasing the pipeline granularity with a throughput reduction.

The use of conditional-sum adders can be discarded a priori since all characteristics (circuit complexity, latency, and throughput) are worse than the corresponding ones for ripple-carry or carry-look-ahead adders, at all granularities.

The optimum choice for the pipelined-adder scheme must therefore consider the traditional structure, the novel scheme proposed in Section 3, and the modified version based on carry-look-ahead adders; for a given application, the conflicting constraints on complexity and performance must be carefully balanced. First of all, for the given set of possible constraints on circuit complexity, clock cycle (i.e., latency), and throughput, the designer should select the architectural solutions at any pipeline granularities that simultaneously satisfy such constraints. If no solution is available, it is necessary to relax at least one constraint; if several solutions are acceptable, a preferred figure of merit should be identified (according to the specific application) in order to complete the scheme selection.

The actual values of the figures of merit, used to evaluate the architectural approaches, depend on the specific implementation of the basic units (adders and latches). Therefore, the choice of the optimum approach should be performed on these values. The analysis here presented—even if quite generally valid—holds exactly only for the specific implementation adopted.

ACKNOWLEDGMENT

The authors are grateful to the anonymous referees for providing comments and suggestions that greatly helped in improving this paper.

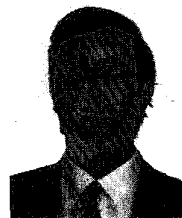
REFERENCES

- [1] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, vol. 34, pp. 349-356, May 1955.
- [2] J.M. Muller, *Arithmétique des Ordinateurs*. Paris: Masson, 1989.
- [3] G. Corbaz, J. Duprat, B. Hoher, and J.M. Muller, "Implementation of a VLSI Polynomial Evaluator for Real-Time Applications," *Proc. Int'l Conf. Application-Specific Array Processors (ASAP'91)*, pp. 13-24, Barcelona, Aug. 1991.
- [4] G. Goggi, B. Lofstedt, et al., "A Digital Front-End and Read-Out Microsystem for Calorimetry at LHC—Digital Filters," *Report on the FERMILAB Project of the European Organization for Nuclear Research, CERN/DRDC/92-26 RD-16*, pp. 36-41, May 1, 1992.
- [5] D. Somasekhar and V. Visvanathan, "A 230 MHz Half Bit-Level Pipelined Multiplier Using True Single-Phase Clocking," *Proc. Sixth Int'l Conf. VLSI Design*, pp. 347-350, Bombay, Dec. 1993.
- [6] S.P. Johansen, "Systolic Evaluation of Functions: Digit-Level Algorithm and Realization," *Proc. Int'l Conf. Application-Specific Array Processors (ASAP'93)*, pp. 514-523, Venice, Oct. 1993.
- [7] I. Koren, *Computer Arithmetic Algorithms*. Englewood Cliffs, N.J.: Prentice-Hall, 1993.
- [8] M. Mehta, V. Parmar, and E.E. Swartzlander, "High-Speed Multiplier Design Using Multi-Input Counters and Compressor Circuits," *Proc. Int'l Symp. Computer Arithmetic*, pp. 43-50, 1991.
- [9] S. Kawahito, M. Ishida, T. Nakamura, M. Kamayama, and T. Higuchi, "High-Speed Area-Efficient Multiplier Design Using Multiple-Valued Current-Mode Circuits," *IEEE Trans. Computers*, vol. 43, no. 1, pp. 34-42, Jan. 1994.



Luigi Dadda received the DrIng degree in electrical engineering in 1947 from Politecnico di Milano, Italy. He has been a professor there since 1960 teaching courses in electrical engineering and computer science.

Dr. Dadda has done research in electromagnetic field theory and measurement, switching theory, and computer arithmetic. His current research interests include computer arithmetic, signal processing, and fault tolerance. He is a member of IEEE.



Vincenzo Piuri received the DrIng degree in electronic engineering in 1984 and the PhD in information engineering in 1989 from Politecnico di Milano, Italy. He is an associate professor in operating systems at Politecnico di Milano.

Dr. Piuri's research interests include distributed and parallel computing systems, computer arithmetic, neural networks, and fault tolerance. He is a member of IEEE, AEI, IMACS, and INNS.