Shuffled Serial Adder: An Area-Latency Effective Serial Adder

G.P. Saggese (1), A.G.M. Strollo (1), N. Mazzocca (2), D. De Caro (1)

(1) Dept. of Electronics and Telecommunications Engineering, Univ. of Naples – "Federico II",
Via Claudio 21, 80125 Napoli, Italy - Fax: +39 081 5934448 - Phone +39 081 7683118 - Email: saggese@unina.it
(2) Dept. Of Information Engineering, Second Univ. of Naples,
Via Roma 29, 81031 Aversa (CE), Italy - Fax: +39 081 5037042

ABSTRACT

Shuffled Adder, a novel serial adder with a latency proportional to $\log_2 N$, where N is the operand width, is presented. It is derived from Kogge-Stone adder, reordering the cells obtaining a slice-based structure that allows an area-time effective serial implementation. It can be fed with parallel inputs and produce parallel outputs, differently from digit-serial approach that results in need of data-formatter to convert parallel input in digit, and to collect digits of output word. A standard-cell VLSI implementation for different values of N is presented: area-time performances are evaluated through circuit simulations, and compared with digit-serial adder with various digit-size, proving the attractiveness of proposed structure.

I. INTRODUCTION

Many signal-processing systems are submitted to fixed real-time constraints, but the real-time requirements can vary from low sample rates (as digital processing of speech) to very high sample rates (as video and image processing). When implementing dedicated systems, a range of possible architectures exists: *bit-serial* architectures [1] perform computations on a bit at a time and so are better suited for low-speed applications, whereas *bit-parallel* architectures process all input bits of a word in one clock cycle, and are ideal for high speed applications. Bit-serial structures require less hardware, simple and few interconnections, and less pin-out, while parallel systems need a large amount of silicon area, interconnection, and input/output pins. An interesting design option is *digit-serial* systems [2], which process more than one input bit in one cycle (this number is referred as digit size), appearing suitable for intermediate-throughput applications, and allowing to trade-off area, circuital complexity and performances.

The above-mentioned approaches can be used with adder architectures, which are the most frequently used arithmetic primitive [3], involved not only in simple addition, but also in more complex operations like multiplication, division and in other basic building block of digital processing systems.

In this paper we present a novel architecture of serial adder, that overcomes problem of data conversion from and to parallel format. It is derived from Kogge-Stone adder using a particular reordering permutation of cells in processing core of the adder, obtaining a highly-repetitive structure which allows a slice-based architecture that can be turned in a serial-style implementation. In order to show effectiveness of proposed adder, it is compared in terms of area and throughput, with a serial version of Kogge-Stone adder, obtained without reordering of cells, and with a digit-serial version of Kogge-Stone, for different operand width and digit-size.

II. PARALLEL-PREFIX ADDERS

The addition of two N-bit binary numbers $\underline{A} = (a_{N-1}, a_{N-2}, \dots, a_0)$ and $\underline{B} = (b_{N-1}, b_{N-2}, \dots, b_0)$ with an input carry $c_{in}=c_0$ can be formulated as:

$$(1) \quad \begin{cases} c_{i+1} = a_i b_i + (a_i + b_i)c_i = g_i + (p_i c_i) \\ s_i = a_i \oplus b_i \oplus c_i = p_i \oplus c_i \end{cases} \qquad i = 0, \dots, N-1$$

yielding the sum $\underline{S} = (s_{N-1}, s_{N-2}, \dots, s_0)$ and $c_{out} = c_N$. In (1) g_i is the *generate signal* defined by $a_i b_i$, when $1 \le i \le N-1$, and $a_0 b_0 + a_0 c_0 + b_0 c_0$ when $i = 0$, while p_i is the *propagate signal* defined as $p_i = a_i \oplus b_i$, where $0 \le i \le N-1$.

It is obvious that speed of addition computation depends on the speed of calculation of the intermediate carries c_i, assuming equal input $\underline{A}$, $\underline{B}$, c_{in} arrival times.

Defined the operation • on ordered pairs of bit as:

$$(2) \qquad (g_a, p_a) \bullet (g_b, p_b) = (g_a + p_a g_b, p_a p_b)$$

it follows that:

$$(3) \qquad (c_{i+1}, p_0 p_1 \dots p_i) = (g_i, p_i) \bullet \dots \bullet (g_1, p_1) \bullet (g_0, p_0)$$

It is easy to verify that the operator • is *associative* (that is: $\underline{v}_a \bullet \underline{v}_b \bullet \underline{v}_c = \underline{v}_a \bullet (\underline{v}_b \bullet \underline{v}_c) = (\underline{v}_a \bullet \underline{v}_b) \bullet \underline{v}_c$), *idempotent* ($\underline{v}_a \bullet \underline{v}_a = \underline{v}_a$), but *not commutative* ($\underline{v}_a \bullet \underline{v}_b \neq \underline{v}_b \bullet \underline{v}_a$).

The associativy means that serial iteration of equation (3) can be parallelized, idempotency allows these sub-terms to be overlapped, obtaining some useful flexibility in parallelization: together these properties allow great parallelism, and hence faster circuits.

Carry-propagation of binary addition is a prefix problem [4],[5] and the carries c_i can be calculated using a prefix algorithm. A general implementing scheme of binary addition, based on (1) in terms of g_i-p_i, is reported in Fig.1. An interesting comparative study of VLSI implementation of several classes of parallel adders, in terms of area, time and power, is reported in [6].

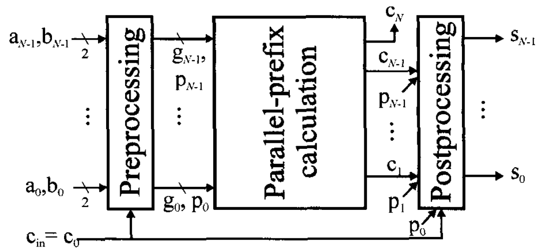


Fig. 1: The three stages of a general implementation structure for parallel-prefix addition.

III. SHUFFLED ADDER

In Fig. 2 the usual representation of Kogge-Stone adder [7] (with $N = 8$) is reported. Adder is represented by a direct acyclic graph, drawn in the traditional prefix description style [5]: graph *edges* represent signal interconnections, while graph *nodes* stand for logic cell (*white nodes* depict no logic or simple buffer block, and *black nodes* consist in computation logic cells corresponding to the equations explained in Fig. 3).

Each wire in Fig. 2 carries two bits, except signals which are output c_i and carry one bit. Near each graph node (that implements operations explained in Fig. 3), a couple of numbers i-j is reported, indicating that the output of that cell is the result of operation $(g_j, p_j) \bullet \dots \bullet (g_i, p_i)$. According to (3), the cell with 0-j produces the carry c_{j+1}.

Parallel-prefix computation stage of Kogge-Stone adder has the

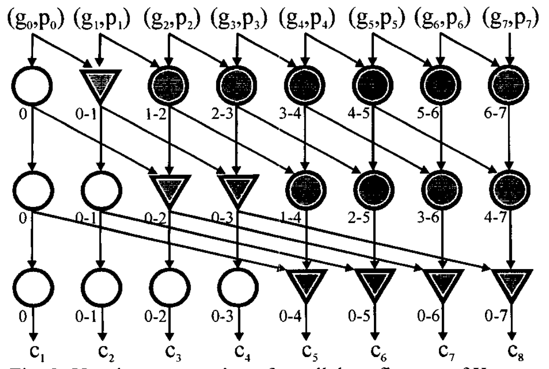


Fig. 2: Usual representation of parallel-prefix core of Kogge-Stone adder, with $N = 8$.

minimum logic depth $\log_2 N$ (as Sklansky adder [8]) and a fan-out for each cell of only 2 cells, so the delay is proportional to $\log_2 N$. It requires a number of cells equal to $N{\cdot}(\log_2 N-1) + 1$.

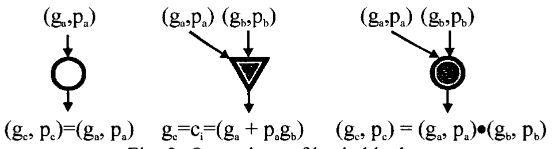


Fig. 3: Operations of basic blocks.

It is possible to reorder the elementary cells of parallel-prefix calculation stage of Kogge-Stone adder, using the *perfect shuffle* connection [9], obtaining a more suitable structure for serialization.

In Fig. 4 the *perfect shuffle* net with $N = 8$ is reported; the r-th node is linked with the s-th node and the relation between r and s is given by:

$$(4)\quad \begin{cases} r = \sum_{i=0}^{N-1} a_i \cdot 2^i = a_{N-1}\cdot 2^{N-1} + a_{N-2}\cdot 2^{N-2} + \ldots + a_1\cdot 2 + a_0 \\ s = \sum_{i=0}^{N-1} a_{(i+1)\bmod N}\cdot 2^i = a_{N-2}\cdot 2^{N-1} + a_{N-3}\cdot 2^{N-2} + \ldots + a_0\cdot 2 + a_{N-1} \end{cases}$$

where N is any integer. As the above equation reports, s simply consists in a right circular shift of r. In other words the indices on the top of Fig. 4 are connected with indices on the bottom of Fig. 4, according to the permutation P:

$$(5)\qquad P_i = \begin{cases} 2i & 0 \le i \le N/2 - 1 \\ 2i + 1 - N & N/2 \le i \le N - 1 \end{cases}$$

The obtained scheme of Kogge-Stone, called below *Shuffled Kogge-Stone*, is reported in Fig. 5, where for sake of clarity the triangle-cells, that are simplified cell implementing equation (2), are not drawn: anyway it is trivial replace in Fig. 5 the circle-cells with appropriate triangle-cells.

In Fig. 5 we have added also dashed wires in order to highlight regularity of the structure obtained through the application of perfect shuffle. White cells in Fig. 5 aim to propagate one of his two 2-bit inputs, in order to obtain same functional behaviour of the scheme of Fig. 2.

The structure of Shuffled Kogge-Stone is composed by $\log_2 N$

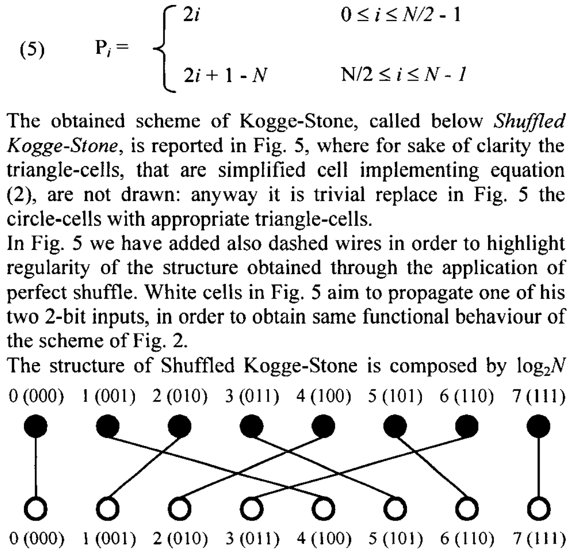


Fig. 4: Two vectors of 8 elements connected through perfect shuffle

slices, whereas connection between cells of any pair of contiguous slices is arranged as a perfect shuffle net.

The number of the cells used in scheme of Fig. 2 and 5 is the same because the operation performed is a simple permutation of cells and introduction of slightly different white cells. It is worth point out that i-th slice, where $i = 0, \ldots, \log_2 N-1$, is composed of $N - 2^i$ calculation cells and 2^i propagation cells. Also the number of wires is the same, but different is the number of wire intersections, which is:

$$\frac{N}{4}\cdot\left(\frac{N}{2}-1\right)\cdot(\log_2 N - 1)$$

for Shuffled Kogge-Stone of Fig. 5, and

$$\frac{2}{3}\cdot\left(N^2-1\right) - N\cdot\log_2 N$$

for scheme of Fig. 2, as it is simple to prove.

It is also possible to obtain output carries c_i in an orderly manner ($c_1, c_2, \ldots c_N$ from the right to the left of the bottom of Fig. 5) using another perfect shuffle connection between c_i of scheme and actual output of parallel-prefix processing block.

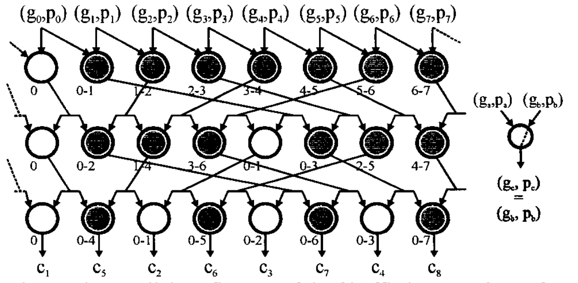


Fig. 5: The parallel-prefix core of the Shuffled Kogge-Stone for $N = 8$, and operations performed by white cells.

IV. SERIAL SHUFFLED ADDER

In order to turn a parallel structure into serial one, we need logic blocks realizing a given operation (same operation in all iterations, or not) on different data. An example is a digit-serial adder of N-bits words, which sums K-bit digit using previous stage carry: repeated N/K operations are always the same, but on different digits.

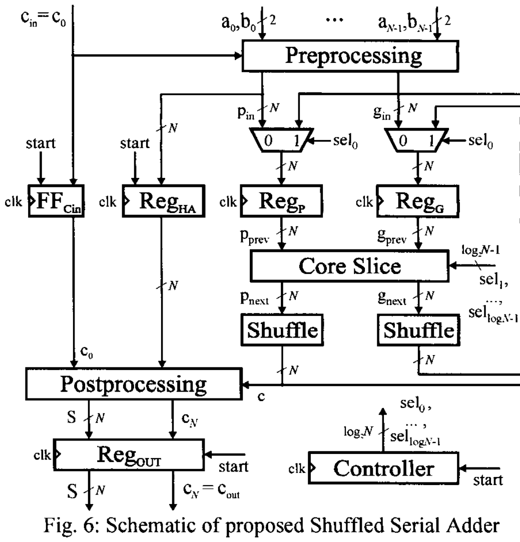


Fig. 6: Schematic of proposed Shuffled Serial Adder

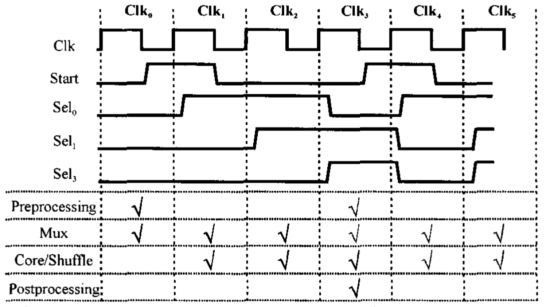
The *Shuffled Serial Adder*, proposed in this paper, is derived from Shuffled Kogge-Stone architecture, that is made serial using reconfigurable slices. The reordering of the basic cells makes it suitable for an effective reconfiguration, as we prove in next sections, showing saving of area and reduction of addition time, in respect a straight serialization of the structure of Fig. 2. The presented adder is suitable for serialization mainly for two reasons:

1) the connection net is the same for each slice (i.e. for each temporal iteration), so does not request to reconfigure;
2) the slice can be reconfigured with a small overhead of silicon area and delay, using muxes, as we will show in the following sections.

In Fig. 6, schematic of the proposed serial structure is shown. It is simple to demonstrate that the proposed architecture, with timing reported in Fig. 7, and implementation of basic blocks described in Fig. 8, serially realizes the computation of Kogge-Stone adder.

Obviously the actual way of providing data input, and the desired form of the output, heavily influence the structure of the adder. We consider the most general case, that is when the data are fed in parallel stably only for one clock tick, and the data output are required in parallel and stable for any number of ticks.

Following, we will explain the functionality and timing of the basic blocks making up Shuffled Serial Adder.



	Clk₀	Clk₁	Clk₂	Clk₃	Clk₄	Clk₅
Preprocessing	√			√		
Mux	√	√	√	√	√	√
Core/Shuffle		√	√	√	√	√
Postprocessing				√		

Fig. 7: Timing required for Shuffled Serial Adder with $N = 8$

Signal start marks the start point of the addition: the input data $\underline{A}$, $\underline{B}$, c_{in} must be provided T_{setup} before the rising edge of the clock, to meet the setup constraint of the system, and kept stable in order to guarantee the right sampling of registers. $T_{Set\text{-}up}$ is given by $T_{PRE} + T_{MUX} + T_{Set\text{-}up,REG}$, where T_{PRE} and T_{MUX} are the combinational delay of the respective blocks, and $T_{Set\text{-}up,REG}$ is the setup time of flip-flops making up registers Reg_P and Reg_G. Start also activates *Controller* block that issues signal $Sel_0, \ldots, Sel_{\log N\text{-}1}$ controlling flow of data inside structure. With the arrangement of wires reported in Fig. 5,6 *Controller* can be realized as a simple $\log_2 N$ bit shift register, asynchronously reset by start. In the first clock tick of the adder operation (Clk_0 in Fig. 7) the *Preprocessing* block (that corresponds to Preprocessing stage of Fig. 1) computes g_i, p_i using input signals $\underline{A}$, $\underline{B}$, c_{in}. Flip-flop FF_{Cin} and register Reg_{HA} are loaded on the subsequent rising edge of clock, respectively, with the value of carry-in c_{in} and the p_i. This is essential since carry-in and calculated p_i cannot be stable, depending on data timing assumptions, until the last clock of the addition, when they are used by *Postprocessing* block to produce final s_i according to the second equation of (1). The two *Mux* blocks, in the first clock tick, load registers Reg_P and Reg_G with value of all g_i, p_i and, in the subsequent $\log_2 N$-1 clock ticks, are used to close in feedback the series composed by *Core Slice* and *Shuffle net* blocks, in order to obtain the calculation realized by the Parallel-prefix Calculation block of Fig. 5. Reg_P and Reg_G are used to hold the partial results of the addition process. The implementation of *Core Slice* block is reported in Fig. 8 for $N = 8$. It is simple to generalize this structure for any case of N, considering that this block should behave as single slice of Fig. 5 in different clock ticks: numbering cells (from the left) $0\ldots N$-1, where N is the length of word to add, it follows that the i-th cell is white if $i = 0$, is black if (i mod 2) = 1, is crossed otherwise and wire Sel_j (with j = 0, … , N –1) controls 2^{j-1} crossed-cells of indexes $k\cdot 2^{N-j}$ where k = 1, … , 2^{j-1}. In the last clock cycle of the addition, start enables Reg_{OUT} that stores the resulting c_i produced, by the last iteration of the *Core Slice* and *Shuffle nets*, combined with p_i stored in FF_{Cin} and Reg_{HA}, by *Postprocessing* block that implements the operation $s_i = p_i \oplus c_i$.

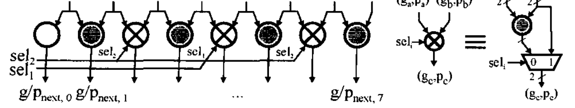


Fig. 8: Schematic of a Core Slice of Fig. 6

Fig. 7 reports controlling signals' behaviour and timing, and which blocks work in each clock period of an addition process. The latency of Shuffled Serial Adder can be computed as:

$$T_{Latency} = T_{Set-up} + (\log N + 1)\cdot\left(T_{Core} + T_{DQ} + \max\{T_{MUX}, T_{POST}\}\right)$$

where T_{DQ} is the sum of set-up and hold time of flip-flops.

It is worth point out that if we need to realize a stream of addition (a common case in digital signal processing) it is possible use this structure starting a new addition, one clock cycle before the previous has finished (overlapping one cycle of two different operations), obtaining a throughput of one addition every $\log_2 N$. The scheduling of the blocks in this case is reported in Fig. 7 where dark tick represents use of the corresponding block for the previous addition, and light tick points out successive addition.

V. IMPLEMENTATION AND PERFORMANCE COMPARISON

We have realized a VHDL RTL description of proposed architecture, completely parameterized in respect of N, using a technique similar to "building-block" technique described in [10]. We have synthetized a gate-level description of the circuit, using CADENCE NAVIGATES with CSI AMS 0.35μm standard-cell technology and 3.3V power supply. The timing performances of the circuit have been obtained through CADENCE PEARL simulations.

We demonstrate initially the effectiveness of cells reordering in terms of silicon area and delay, comparing the slice of Shuffled Serial Adder (Fig. 8) with the slice derived directly from scheme of Fig. 2 and reported in Fig. 9, where equations implemented by the cells are the same of previous schemes.

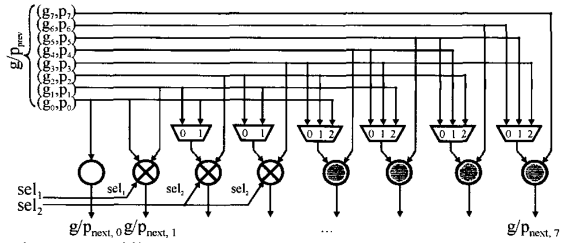


Fig. 9: Core Slice without cell reordering, directly derived from scheme of Fig. 2

In Tab. 1 we report the comparison in terms of area and delay between reordered slice of Fig. 8 and slice of Fig. 9. Note that p/g_{prev} to p/g_{next} delay of slice of Fig. 8 does not increase and area grows linearly, while the same does not occur with structure of Fig. 9 because of muxes that connect processing cells with the available input signals p/g_{prev}. In Tab.1 the mux area overhead (ratio between mux area and computational blocks' area) is also reported.

	Core Slice of Figure 7			Core Slice of Figure 8		
N	Area [μm^2]	Delay [ns]	Mux Overhead	Area [μm^2]	Delay [ns]	Mux Overhead
8	3003	0.5	57 %	7480	0.87	291 %
16	6552	0.49	86 %	22386	1.12	535 %
32	13176	0.51	43 %	65744	1.33	616 %
64	24752	0.5	34 %	197323	1.44	968 %
128	39257	0.52	69 %	371716	1.57	1496 %

Table 1: Comparison of slice of proposed architecture and the slice of Fig. 8

To compare performances of proposed architecture, we use a digit-serial adder, with the same functional behavior, and various values of N (i.e. number of bit of adder) and K (that is the digit size) in order to compare performances of proposed circuit and performances of different digit-size adders trading off area and latency.

The digit-serial adder, with digit size K, computing the sum of two N bit long words $\underline{A}$, $\underline{B}$ with carry c_{in}, is depicted in Fig. 10.

Since the inputs are stable only for one clock tick, the adder needs two N bit registers $ParToSer_A$, $ParToSer_B$, that load their input for load active, and when load is low, shift and show on output their contents, K bit at a time. Each register can be implemented exploiting N-K multiplexers. Since the resulting sum is produced through N/K additions on digits composing $\underline{A}$ and $\underline{B}$, it is essential use an output formatter, that can be implemented as a digit-serial to parallel register with N-K flip-flop. Furthermore, since the output must be stable for any subsequent clock ticks after the addition, we added an N+1 bit register that stores the result of addition and carry output.

When the digit-size is equal to N, i.e. the addition is executed by a completely parallel adder, it is sufficient use only three N-bit register, without the circuitry needed to shifting.

Any K bit adder can be used as digit-serial adder. A Kogge-Stone adder is used in this paper, in order to perform a fair comparison with proposed Shuffle Serial Adder.

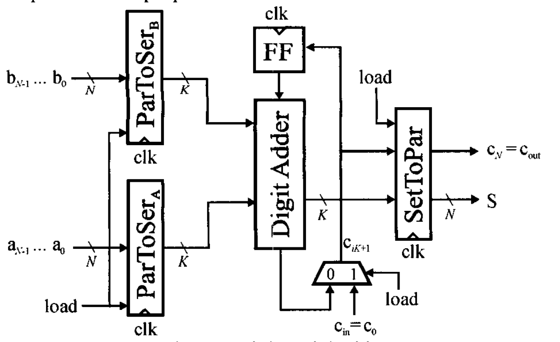


Fig. 10: Digit-serial adder

For given N, we designed digit-serial adders with different K value, so that we obtain a design space highlighting the trade-off between area and latency. We include in latency the set-up times, so the expression of latency for digit-serial adder is:

$$T_{Latency} = T_{Set-up} + \left(\frac{N}{K} + 1\right) \cdot \left(T_\bullet \cdot \log_2 K + T_{DQ} + T_{MUX}\right)$$

where T_{Set-up} is the set-up time due to mux of registers $ParToSerA$, $ParToSerB$, T_{DQ} is the sum of set-up and hold time of flip-flops, and $T_\bullet$ is the delay of a black cell of Fig. 2.

The results are depicted in log-log diagram of Fig. 11, where plus indicates Shuffled Serial Adder, and squares design points for digit-serial adder. To make comparison simpler, we also report in Fig.12, area and latency of digit-serial adders normalized, respectively, to area and latency of Shuffled Serial Adder with the same N.

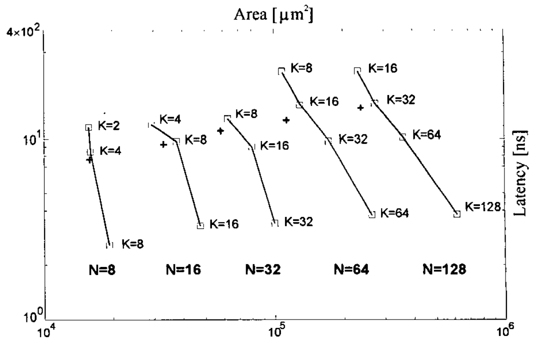


Fig. 11: Results of comparison in log-log diagram

Fig. 11-12 prove the effectiveness of proposed architecture, in terms of area-latency, in comparison with a digit-serial adder with different digit-size values: for instance, using the same latency, area reduction shown by Shuffled Serial Adder ranges between 3.4 % (N=8) and 32 % (N=64), in comparison with digit-serial adder.

VI. CONCLUSIONS

A novel serial adder with a delay proportional to logarithm of the number of bit of adding words, has been presented. It is derived from Kogge-Stone adder, through an appropriate re-arrangement of the cells that leads to a structure with highly repetitive connections and cell organization, effectively suitable for a serial implementation. It shows a word-parallel input to word-parallel output behaviour. A standard-cell implementation for different value of N is presented, and its performances are compared with digit-serial adder with various digit-size, proving the effectiveness of proposed structure.

VII. REFERENCES

[1] R. Hartley et al.: "Digit-Serial Processing Techniques", IEEE Trans. on Circuits and Systems, Vol.37, No.6, June 1990, pp. 707-719.

[2] K.K. Parhi: "A Systematic Approach for Design...", IEEE Trans.on Circuits and Systems, Vol. 38, No. 4, April 1991, pp. 358-375.

[3] D.C. Chen et al.: "An integrated system for rapid prototyping of ...", in Proc. Appl. Specific Arrays Proc., Aug. 1992, pp.134-148.

[4] H. Lindkvist et al.: "Techniques for fast CMOS...", In Proc. IEEE Int. Conf. Comp. Design, Cambridge, USA, Oct. 1994, pp.626-635.

[5] S. Knowles: "A Family of Adders", Proc. 14th IEEE Symp. on Computer Arithmetic, 1999, pp. 30-34

[6] C. Nagendra et al.: "Area-Time-Power Tradeoffs...", IEEE Trans. On Circ. and Systems-II, Vol. 43, No. 10, Oct. 1996, pp.689-702.

[7] P.M. Kogge et al.: "A parallel algorithm for the efficient...", IEEE Trans. on Computer, Vol. 22, No. 8, Aug 1973, pp. 783-791.

[8] J. Sklansky: "Conditional-Sum Addition Logic", IRE Trans., EC-9:226-231, June 1960.

[9] H.S. Stone: "Parallel Processing with the Perfect Shuffle", IEEE Trans. on Computer, Vol. 20, No. 2, Feb 1971.

[10] H. Kunz et al.: "High-performance Adder Circuit generators in Parametrized Structural VHDL", Tech. Rep. 96/7, August 1996

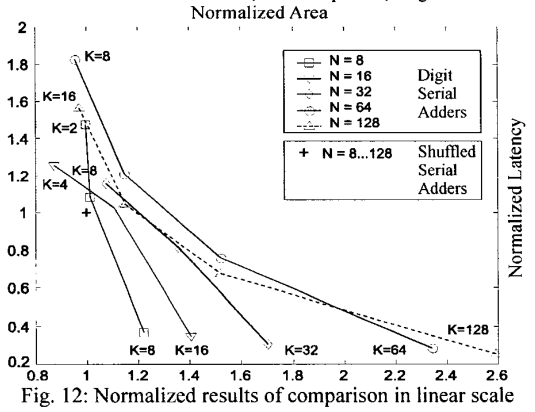


Fig. 12: Normalized results of comparison in linear scale