Fast Adder Architectures: Modeling and Experimental Evaluation

Nuno Roma and Tiago Dias and Leonel Sousa
I.S.T. / INESC-ID
R. Alves Redol, 9, 1000-029 Lisboa, Portugal
Email: nuno.roma@inesc-id.pt, tdias@sips.inesc-id.pt, las@inesc-id.pt

***Abstract*— This paper presents a detailed comparison analysis of several fast adder architectures for high performance VLSI design. The evaluation of those architectures is firstly carried out based on a simple gate-count model for area and gate-delay units for time. The results obtained with such model were then validated by using two entirely different real-world implementation technologies, namely CMOS integrated circuits and Field Programmable Gate Arrays (FPGA). Experimental results show that among the modeled and evaluated topologies, the adder architecture based on the radix-2 redundant format converter offered the lowest delay when implemented with any of the considered technologies. However, it was also the topology that required the highest amount of hardware. The presented results can be seen as an invaluable resource in the selection of the most appropriate adder topology that will be used to implement a given arithmetic operation in a specified technology.**

I. Introduction

In the design of specialized or dedicated VLSI architectures, the usage of particularly efficient processing units is often of great importance to achieve the required performance levels for the overall system. This performance is usually adjusted in terms of the tradeoff: processing time *versus* circuit area.

From a careful analysis of the whole processing system, one frequently concludes that the used arithmetic units are often the most responsible for the achievement of such requirements. Consequently, it has been widely accepted that high performance levels can only be achieved by using arithmetic units that have been optimized to the characteristics of the operation that they are intended to perform. Nevertheless, despite the immensity of different arithmetic operations that can be implemented in VLSI circuits, one can usually decompose them into elementary operators, such as additions and subtractions. Consequently, it is consensus that it is often preferable and easier to perform such optimizations by selecting the most appropriate architectures for addition/subtraction.

Therefore, the main objective of the research presented in this paper is to perform a detailed comparison analysis of several fast adder architectures, to ease the selection of the most appropriate topology for a given operation and a specified technology. Several different architectures were considered in this study: binary-tree *carry-lookahead* structures [1], [2], *redundant number system* adders [3], [4], [5] and Sklansky *prefix-adders* [6], [7]. Since the obtained performance level is also greatly dependent on the technology used for the implementation, the main concern in the comparison of the several topologies was to use a neutral criteria that could evaluate each architecture independently of the target implementation. Consequently, it was decided to adopt the model proposed by Tyagi, which relates the required circuit area with the propagation time of each logic cell [8], [6]: each two-input monotonic gate (w.g., AND,NAND) counts as one gate (area and delay); an XOR counts as two gates both to the area and to the delay; and an m-bit logic cell derived from the elementary logic cells count as $m-1$ gates to the area and $\lceil \log_2 m \rceil$ gates to the delay. To assess the accuracy of this model, the obtained performance measures will be contrasted with the implementation results obtained using two distinct technology supports: a StdCell library based on a $0.25\mu m$ CMOS process [9] that can be used to implement the desired circuit on an ASIC and a VIRTEX-E general purpose FPGA [10] from Xilinx.

II. Adder architectures

During the presentation of the several architectures, it will be used, for comparison purposes, the performance levels obtained for the simpler (and slower) carry-propagate structure: the *ripple-carry* (RC) adder. This topology is composed by w cascaded *full-adder* circuits that perform the computation of the sum of the two w-bit operands $A = a_{w-1}, \ldots, a_0$ and $B = b_{w-1}, \ldots, b_0$ with an input carry bit $(c_{in} = c_0)$, according to the following expressions:

$$s_i = a_i \oplus b_i \oplus c_i \; ; \; c_{i+1} = a_i b_i + a_i c_i + b_i c_i \qquad (1)$$

By using the described hardware model, the processing time of this circuit is:

$$\begin{cases} T^{c_{out}} = T^{c_w} = w \cdot T_{FA}^{c_{i+1}} = 3w \\ T^{s_w} = (w-1) \cdot T_{FA}^{c_{i+1}} + T_{XOR}^{s_i} = 3(w-1) + 2 \approx 3w \end{cases} \qquad (2)$$

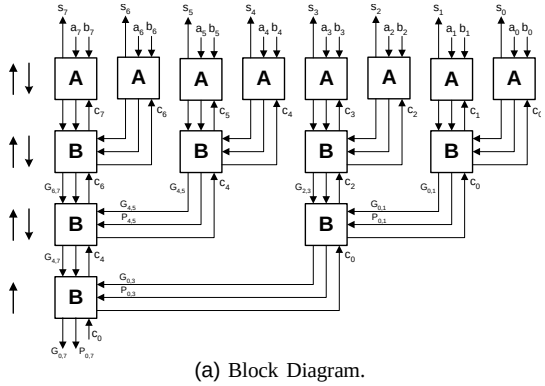
and the required circuit area, considering a direct implementation of eq. 1, is given by:

$$A = w \cdot A_{FA} = 9w \qquad (3)$$

A. Carry-lookahead adder

The main characteristic of the carry-lookahead (CLA) adder with a binary-tree structure is the fact that it is composed by only $\lceil \log_2 w \rceil$ hierarchical logic levels, where w states the number of representation bits of the operands.

The operation of this adder lies in the computation of a w-dimensional carry vector, which is generated and propagated from the A and B input operands through the whole adder structure. Let $G_{i,k}$ be the carry signal generated from inputs i and k and $P_{i,k}$ the propagation signal generated with the same inputs. The carry vector can be calculated using the following



(a) Block Diagram.

$$\begin{cases} T_A^{\downarrow} = 1 \;\; ; \;\; T_A^{\uparrow} = 2 \\ A_A^{\downarrow} = 2 \;\; ; \;\; A_A^{\uparrow} = 4 \end{cases} \qquad \begin{cases} T_B^{\downarrow} = 2 \;\; ; \;\; T_B^{\uparrow} = 2 \\ A_B^{\downarrow} = 3 \;\; ; \;\; A_B^{\uparrow} = 2 \end{cases}$$

(b) Time and Area corresponding to the A and B blocks.

Fig. 1. 8-bit carry-lookahead adder.

recursive equations:

$$c_{k+1} = G_{i,k} + P_{i,k} \cdot c_i \tag{4}$$

$$G_{i,k} = G_{j+1,k} + P_{j+1,k} \cdot G_{i,j} \tag{5}$$

$$P_{i,k} = P_{i,j} \cdot P_{j+1,k} \; ; \; i \leq j < k \tag{6}$$

where $G_{i,i} = g_i$, $P_{i,i} = p_i$, $g_i = a_i.b_i$ and $p_i = a_i + b_i$.

In fig. 1 it is presented the hierarchical structure of this binary-tree adder. This adder performs the computation in three phases: *i*) the p_i and g_i values are computed in the A blocks and fed into the B blocks, where the P and G values are computed using eqs. 5 and 6; *ii*) the input carry bit (c_0) is fed into the B block at the bottom of the tree; B blocks are used to compute the w-dimensional carry bit vector in an ascending data flow, according to eq. 4; finally *iii*) the sum vector is computed by supplying the carry bit vector to the A type blocks. The carry out bit can be easily calculated using the c_0 input signal and the $G_{0,w-1}$ and $P_{0,w-1}$ outputs of the B block at the bottom of the tree. The logic circuit that implements this expression is entirely similar to the ascending part of B type blocks, characterized by $T_{c_w} = 2$ and $A_{c_w} = 2$.

Hence, by following the two data-flow directions, the overall processing time of the binary-tree shown in fig. 1 is:

$$T_S = T_A^{\downarrow} + (\lceil \log_2 w \rceil - 1) \times T_B^{\downarrow} + \lceil \log_2 w \rceil \times T_B^{\uparrow} + T_A^{\uparrow} \tag{7a}$$

$$= 4 \lceil \log_2 w \rceil + 1 \tag{7b}$$

The processing time of the carry out signal with ($T_{c_w} \equiv T_B^{\uparrow}$) is:

$$T_{c_{out}} = T_A^{\downarrow} + \lceil \log_2 w \rceil \times T_B^{\downarrow} + T_{c_w} \tag{8a}$$

$$= 2 \lceil \log_2 w \rceil + 3, \tag{8b}$$

It should be noted that due to its binary-tree structure, the number of representation bits of each operand of a carry-lookahead adder like this is usually an integer power of two. Under certain circumstances, this fact can lead to a misuse of the hardware resources. Therefore, from this point on the number of representation bits of each input operand of this type of structures shall be referred by $\hat{w}$, where $\hat{w} = 2^{\lceil \log_2 w \rceil}$.

The circuit area required to implement this adder is:

$$A = \hat{w} \times A_A + (\hat{w} - 1) \times A_B + A_{c_w} \tag{9a}$$

$$= \hat{w} \times (2 + 4) + (\hat{w} - 1) \times (3 + 2) + 2 \tag{9b}$$

$$= 11\hat{w} - 3 \tag{9c}$$

B. Sklansky prefix adder

Prefix-adder architectures [6] are generally characterized by the fact that the outputs $(y_{w-1}, y_{w-2}, \ldots, y_0)$ are computed from the w-bit inputs $(x_{w-1}, x_{w-2}, \ldots, x_0)$ by means of an associative binary operator $\star$ in a recursive form:

$$y_0 = x_0 \; ; \; y_i = x_i \star y_{i-1} \;\; ; \;\; i = 1, 2, \ldots, w-1 \tag{10}$$

Consequently, every output signal depends on the input signals of equal or lower weight. Similarly, any input signal influences all the output signals of equal or higher weight. Due to the associative properties of this generic $\star$ operator, the operations may be executed in any arbitrary order. In particular, subsets of operands can be grouped together to obtain, in parallel, partial solutions of the main problem corresponding to subsets of the input bits, giving rise to group variables $Y_{i,k}$. The variable $Y_{i,k}^{\ell}$ will stand for the output of the operation that takes the set of bits $(x_k, x_{k-1}, \ldots, x_i)$ at level ℓ. The group variables at the last level m will comprise the entire range of the input bits, from 0 to i $\left(Y_{0,i}^{m}\right)$:

$$Y_{i,k}^{\ell} = Y_{i,j}^{\ell-1} \star Y_{j+1,k}^{\ell-1} \; , \;\; i < j < k \;\; ; \;\; \ell = 1, 2, \ldots, m \tag{11a}$$

$$Y_{i,i}^{0} = x_i \; ; \; y_i = Y_{0,i}^{m} \; , \;\; i = 0, 1, \ldots, (w-1) \tag{11b}$$

Among the several different architectures that have been proposed to deal with this kind of problems [6], the Sklansky topology [7] is one of the most efficients (fig. 2). By adopting the tree-type model, all intermediate signals are computed in a minimal tree structure and are fed, in parallel, to all higher levels that require those signals. Nevertheless, this architecture still presents some disadvantages concerned with the fanout of the output nodes, which shows a significant increment as the number of bits of the adder increases.

Each bit (c_i) of the carry vector is computed from the signals of the last level $\left(G_{0,i}^{m}, P_{0,i}^{m}\right)$. The sum bits are computed in a post-processing operation. The variables $G_{i,k}^{l}$ and $P_{i,k}^{l}$ at the intermediate levels l are computed by the blocks presented in fig. 3.

The Sklansky structure presents the following characteristics in what concerns the processing time and circuit area:

$$T_s^{a,b} = T_s^{c_{in}} = 1 \times T_{\square}^{*} + \log_2 \hat{w} \times T_{\bullet} + 1 \times T_{\lozenge} \tag{12a}$$

$$= 3 + 2 \log_2 \hat{w} + 2 = 2 \log_2 \hat{w} + 5 \tag{12b}$$

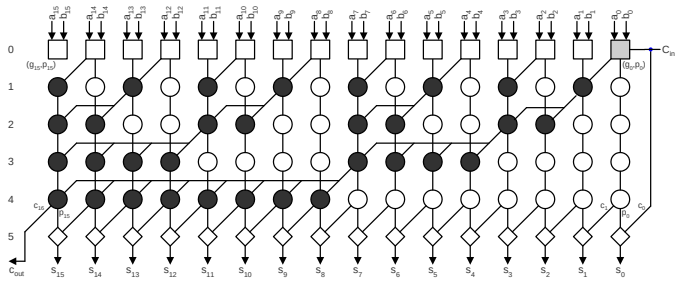
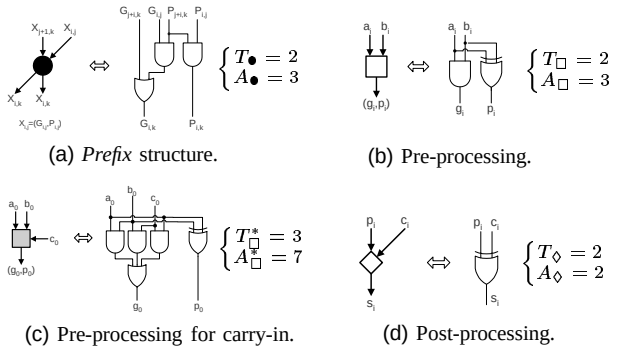


Fig. 2. Structure of the Sklansky prefix-adder.



(a) *Prefix* structure. (b) Pre-processing.

(c) Pre-processing for carry-in. (d) Post-processing.

Fig. 3. Internal structure of the blocks that compose the Sklansky adder.

$$A = (\hat{w}-1) \times A_\square + 1 \times A^*_\square + \left(\frac{1}{2}\hat{w}\log_2\hat{w}\right) \times A_\bullet + \hat{w} \times A_\diamond \tag{12c}$$

$$= 2(\hat{w}-1) + 7 + 3\left(\frac{1}{2}\hat{w}\log_2\hat{w}\right) + 2\hat{w} \tag{12d}$$

$$= \frac{3}{2}\hat{w}\log_2\hat{w} + 4\hat{w} + 5 \tag{12e}$$

where $\hat{w}$ refers to the extension of the number of bits of the input operands to an integer power of two.

An alternative solution for the processing of the carry input consists in using an extra level of the $\bullet$ type operators [6]. Since the carry bit is only used in this last extra level, the processing of the carry signal is quite fast. For this reason, this type of structures is particular suitable for the implementation of *incrementers*, where the critical path usually lies between the c_{in} input and the c_{out} output.

The time associated with this structure is given by:

$$T_s^{a,b} = 1 \times T_\square + (\log_2\hat{w}+1) \times T_\bullet + 1 \times T_\diamond \tag{13a}$$

$$= 2 + 2\,(\log_2\hat{w}+1) + 2 = \ 2\log_2\hat{w} + 6 \tag{13b}$$

Assuming that the critical path of the circuit does not include the processing of the input operands, the time required to obtain the output sum depends only on the c_{in} input bit and is given by:

$$T_s^{c_{in}} = \ 1 \times T_\bullet + 1 \times T_\diamond = \ 2 + 2 = \ 4 \tag{14a}$$

The circuit area is given by:

$$A = \hat{w} \times A_\square + \left(\frac{1}{2}\hat{w}\log_2\hat{w} + \hat{w}\right) \times A_\bullet + \hat{w} \times A_\diamond \tag{15a}$$

$$= \frac{3}{2}\hat{w}\log_2\hat{w} + 8\hat{w} \tag{15b}$$

C. Redundant arithmetic adder

The attractiveness of the redundant signed-digit number systems [3], [4], [5] lies in their "carry-free" addition property, by limiting the carry propagation to a few bit positions, which is usually independent of the word length w. For the particular case of a radix-2 signed-digit system, any number may be coded using two unsigned binary numbers, one positive and one negative, as $X = X^+ - X^-$. Hence, each signed digit $x_i = x_i^+ - x_i^-$ is represented by two bits $\left(x_i^+, x_i^-\right)$, where $x_i^+, x_i^- \in \{0,1\}$ and $x_i \in \{\overline{1}, 0, 1\}$ [5]. According to this notation, the 0 value can be represented by either the $(1,1)$ or $(0,0)$ values. However, it can be shown that if the $(1,1)$ representation is not allowed, faster and simpler circuits can be obtained [4]. A straightforward approach to avoid the $(1,1)$ representation consists in its replacement by $(0,0)$ before performing any operation, which can be done using the circuit presented in fig. 4.

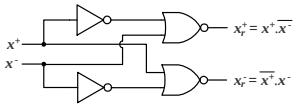


Fig. 4. Digit conversion to perform the replacement of $(1,1)$ by $(0,0)$.

The conversion between the redundant format (X^+, X^-) and the two's complement nonredundant format (X) can be carried out by considering X^+ and X^- as two independent unsigned numbers and subtracting X^- from X^+ as follows [5]:

$$x_i^+ - x_i^- - c_i = 2c_{i+1} + X_i, \tag{16}$$

where $c_0 = 0$. The carry and the X_i bits are given by:

$$c_{i+1} = c_i \cdot \overline{x_i^+} + \overline{c_i} \cdot x_i^- \tag{17}$$

$$X_i = c_i \cdot \left(\overline{x_i^+ + x_i^-}\right) + \overline{c_i} \cdot \left(x_i^+ + x_i^-\right) \tag{18}$$

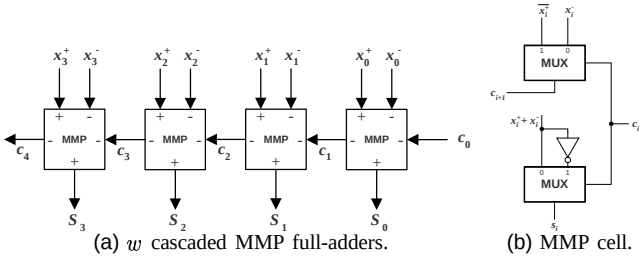
A simple format conversion circuit is shown in fig. 5(a) for a word length $w = 4$. The arithmetic operation required by the conversion can be performed by a dedicated full-adder, usually known as *Minus-Minus-Plus* adder (MMP). It is interesting to note that these two equations can be directly implemented using two multiplexers, whose selection signal is the c_i signal. Such an implementation is only possible due to the pre-conditioning transform presented in fig. 4.

To understand the usefulness of this redundant number representation to implement fast adder structures consider the elementary operation $S = A + B$, which may be rewritten as:

$$S = A + B = A - (-B) = \left(A - \overline{B}\right) + (-1). \tag{19}$$

In the last term of eq. 19, the $\left(A - \overline{B}\right)$ value can be seen as a number represented in the redundant number system, where each $\left(x_i^+, x_i^-\right) = \left(a_i, \overline{b_i}\right)$ digit has the value $\left(a_i - \overline{b_i}\right)$ [4]. Hence, the radix-2 redundant to nonredundant format converter using MMP full-adders presented in fig. 5 can be directly applied to obtain the sum vector. In fact, the (-1) term presented in eq. 19 can be easily integrated in the conversion scheme by considering the initial value of the input carry bit $c_0 = 1$.

Similarly to what happens in the *carry-propagate* adder topologies [6], the sum bits of this adder architecture are only



(a) w cascaded MMP full-adders. (b) MMP cell.

Fig. 5. Radix-2 redundant to nonredundant format converter using four cascaded MMP full-adders.

computed after the evaluation of all carry bits. Nevertheless, since the conversion from the redundant into the two's complement binary representations is quite similar to a binary addition, it is possible to implement fast conversion circuits based on the same techniques used in the *carry-lookahead* adders and by making use of the simpler circuits presented in fig. 5(b), which can be even implemented using *transmission-gates*. Once more, the adopted binary-tree structure makes it necessary to extend the number of bits of the operands to $\hat{w}$.

To illustrate the implementation of an adder architecture based on the redundant number representation consider, as an example, an adder using a tree structure and $\hat{w} = 4$ bit operands. According to eq. 17 and assuming that $\overline{f_0(i)} = x_i^+$ and $g_0(i) = x_i^-$:

$$c_{i+2} = c_{i+1} \cdot \overline{x_{i+1}^+} + \overline{c_{i+1}} \cdot x_{i+1}^- \tag{20a}$$

$$= \left[c_i \cdot \overline{x_i^+} + \overline{c_i} \cdot x_i^-\right] \cdot \overline{x_{i+1}^+} + \left[c_i \cdot x_i^+ + \overline{c_i} \cdot \overline{x_i^-}\right] \cdot x_{i+1}^- \tag{20b}$$

$$= c_i \cdot \underbrace{\left[\overline{x_i^+} \cdot \overline{x_{i+1}^+} + x_i^+ \cdot x_{i+1}^-\right]}_{\overline{f_1(i+1)}} + \overline{c_i} \cdot \underbrace{\left[x_i^- \cdot \overline{x_{i+1}^+} + \overline{x_i^-} \cdot x_{i+1}^-\right]}_{g_1(i+1)} \tag{20c}$$

$$= c_i \cdot \overline{f_1(i+1)} + \overline{c_i} \cdot g_1(i+1) \tag{20d}$$

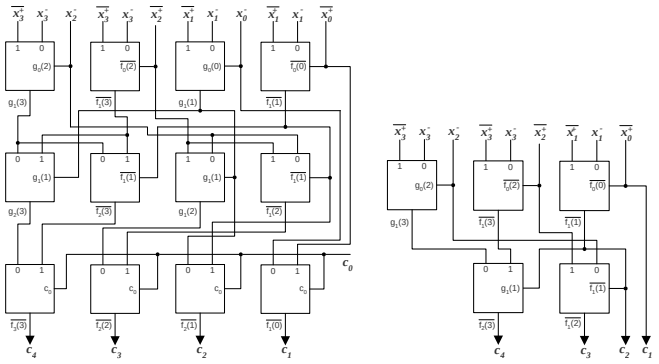
where:

$$\overline{f_1(i+1)} = \overline{f_0(i)} \cdot \overline{f_0(i+1)} + f_0(i) \cdot g_0(i+1) \tag{21}$$

$$g_1(i+1) = g_0(i) \cdot \overline{f_0(i+1)} + \overline{g_0(i)} \cdot g_0(i+1) \tag{22}$$

Hence, it can be shown that all carry signals can be computed by using only multiplexer blocks. This structure has $(\log_2 \hat{w} + 1)$ processing levels and uses a total of $\hat{w} \times (\log_2 \hat{w} + 1)$ multiplexers. Fig. 6(a) presents the circuit used to compute the carry bits for this considered case ($\hat{w} = 4$). However, since the input carry bit for this particular situation is known ($c_0 = 1$), the circuit presented in fig. 6(a) can be considerably simplified, as shown in fig. 6(b). This simplified circuit is only composed by 5 multiplexers and has 2 processing levels.

The two circuits of fig. 6 can be combined to obtain an 8-bit adder. Such circuit is obtained by connecting the output carry (c_4) of the simplified circuit shown in fig. 6(b) to the input carry (c_0) of the circuit shown in fig. 6(a). The latency of this 8 bit converter corresponds to 3 processing levels and



(a) Generic circuit, consisting of $\lceil \log_2 4 \rceil + 1 = 3$ levels.

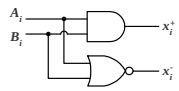
(b) Simplified circuit, considering $c_0 = 1$.

Fig. 6. Circuit to compute the carry vector for the particular case of using 4 bit operands in a redundant number representation.

requires a total of 17 multiplexers. In the general case, this topology computes all the $\hat{w}$ carry bits using $(\log_2 \hat{w} - 1)\hat{w} + 1$ multiplexers in a structure with $\log_2 \hat{w}$ levels [4].

The complete circuit that computes the operation $S = A + B$ is composed by three distinct blocks:

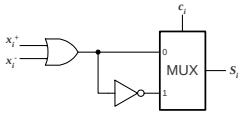
- Pre-processing input block - By taking into account the first complement representation of the $(-B)$ operand, the circuit shown in fig. 4 can be simplified to the circuit shown in fig. 7.



$$\Rightarrow \quad \begin{cases} T = 1 \\ A = 2 \end{cases}$$

Fig. 7. Pre-conditioning circuit.

- Computation of the carry vector $c_w, \ldots, c_1$ - Considering that each multiplexer that composes this block is characterized by $T = 2$ e $A = 3$, the structure of the overall circuit that computes the carry vector will be characterized by $T = 2\log_2 \hat{w}$ and $A = 3\left[(\log_2 \hat{w} - 1)\,\hat{w} + 1\right]$.
- Computation of the sum output value - According to eq. 18, each bit of the sum output value can be computed using the circuit shown in fig. 8. Consequently, the full circuit is characterized by $T = 2$ and $A = 4\hat{w}$.
  It should be noted that neither the OR nor the INV logic gates placed at the input ports of the multiplexer are part of the critical path of this circuit.



$$\Rightarrow \quad \begin{cases} T = 3 \\ A = 4 \end{cases}$$

Fig. 8. Circuit for computing the sum output value.

Hence, the characteristics of the whole adder circuit are [8]:

$$\begin{cases} T = 2\log_2 \hat{w} + 3 \\ A = 3\hat{w}\log_2 \hat{w} + 3\hat{w} + 3 \end{cases} \tag{23}$$

III. Comparative evaluation

This section presents a set of results in order to evaluate the relative efficiency of the several adder architectures considered in the previous sections. The model proposed in [8] will be used in the comparison to relate the *processing time* and the *circuit area* required by each topology. In table I it is summarized the obtained values for the modeled adder architectures. The charts presented in figs. 9 and 10 illustrate the relations between the processing time and the circuit area for all the considered structures in function of the number of bits of the operands (w).

The chart corresponding to the processing time (see fig. 9) evidences the significant performance gains that can be achieved by using the fast adder architectures towards the simpler ripple-carry adder. This gain will be even greater when the number of representation bits (w) increases. Moreover, one can also conclude from this chart that, apart from the Sklansky prefix-incrementer architecture, which shows the best performance under certain specific circumstances, the redundant arithmetic adder and the Sklansky prefix-adder structures have significantly low processing times.

TABLE I

COMPARISON OF THE DIFFERENT ADDER ARCHITECTURES.

Adder Topology	**Processing time**	**Circuit area**
Ripple-Carry (RC)	$3w$	$9w$
Carry-Lookahead (CLA)	$4\log_2 \hat{w} + 1$	$11\hat{w} - 3$
Redundant Arithmetic (RA)	$2\log_2 \hat{w} + 3$	$3\hat{w}\log_2 \hat{w} + 3\hat{w} + 3$
Sklansky (SA)	$2\log_2 \hat{w} + 5$	$\frac{3}{2}\hat{w}\log_2 \hat{w} + 4\hat{w} + 5$
Sklansky Increm.† (SI)	4	$\frac{3}{2}\hat{w}\log_2 \hat{w} + 8\hat{w}$

† For this case, it is assumed that the processing of the operands is not included in the critical path. This can be most useful in the last operation of an addition/increment.

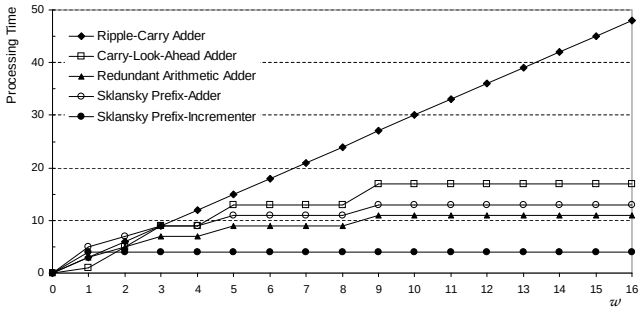


Fig. 9. Comparison of the processing time.

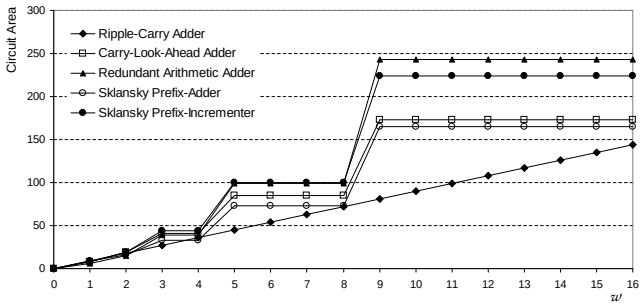


Fig. 10. Comparison of the circuit area.

As expected, in what concerns the circuit area, the ripple-carry adder architecture requires the lowest area (see fig. 10). Since all other topologies are based on binary-tree structures, they present step variations whenever the operands width w is an integer power of two. In these specific cases the circuit area required to implement any fast adder architecture gets closer to the circuit area required for a ripple-carry adder, since the efficiency of the logic circuits is maximized. Despite all these facts, and among the set of analyzed structures, the redundant arithmetic architecture is the topology that requires the greatest circuit area for its implementation.

IV. EXPERIMENTAL RESULTS

A. StdCell library implementation

To assess the results obtained with the model, the considered adder structures were described in VHDL and implemented using a StdCell library based on a $0.25\mu m$ CMOS process from Virtual Silicon Technology [9] and using the set of tools provided by Synopsys. These implementations were carried out with $w = 8$ bit operands.

Table II presents the implementation results for each considered topology, including the estimation of the power dissipated

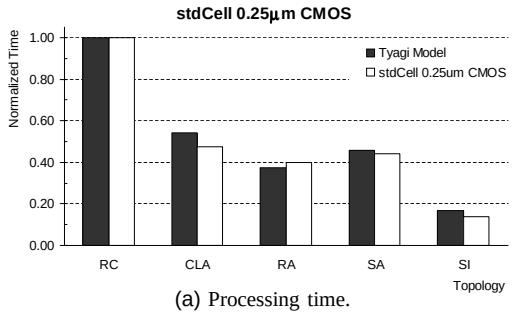
TABLE II

IMPLEMENTATION RESULTS OBTAINED FOR $w = 8$ AND USING THE STDCELL LIBRARY BASED ON A $0.25\mu m$ CMOS PROCESS.

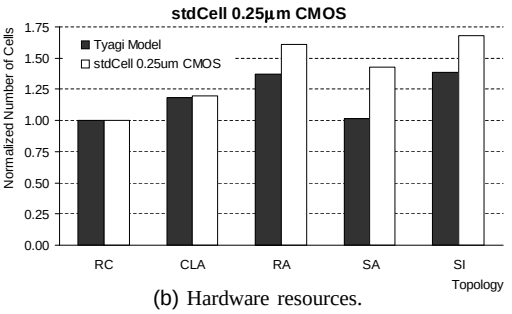
Topology	**Time** $[ns]$	**# Cells**	**Area** $[\mu m^2]$	**Power** $[mW]$
RC	4.68	56	2663	11.387
CLA	2.22	67	3056	13.000
RA	1.86	90	2882	15.161
SA	2.06	80	3057	13.278
SI	0.64	94	3506	14.622

for each architecture when a system clock corresponding to the maximum allowed frequency for each circuit is used. Among the considered adders, the redundant arithmetic structure is the one that dissipates the greatest amount of power. In fig. 11(a) and 11(b) it is presented the comparison of the propagation time and of the required circuit area obtained, respectively, using the Tyagi model [8] and the StdCell library. To carry out this comparison, it was performed a normalization of the values obtained using both the adimensional theoretical model and the StdCell technology, so that both the processing time and the implementation area of the *ripple-carry* circuit are equal to 1 for these two models. Moreover, this normalization was performed by considering that the adimensional measure that represents the circuit area in the model and the number of gates used by each adder circuit implemented using the StdCell library represent the same performance measure that states, in this case, the amount of hardware resources required by each circuit.

As it can be seen in fig. 11(a), the processing time obtained using this library complies very closely with the values predicted by the theoretical model. The same conclusion can be taken from the chart presented in fig. 11(b), concerning the number of logic cells required by each adder circuit. The observed differences can be justified by the synthesis process



(a) Processing time.



(b) Hardware resources.

Fig. 11. Comparison of the results obtained with the theoretical model and with the StdCell library based on a $0.25\mu m$ CMOS process.

performed by the design tools to map the VHDL description of each circuit to the logic cells of the StdCell library. Even so, one observes that the variation of the magnitudes of the obtained values comply very closely with the variation predicted by the model.

B. FPGA based implementation

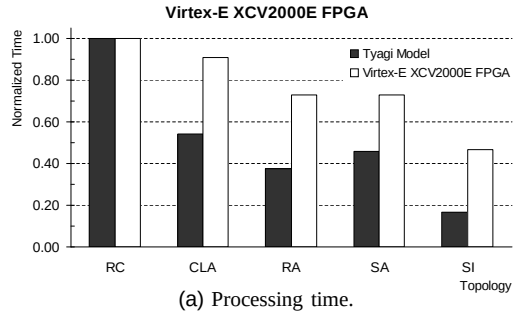
The methodology followed for the StdCell library was also adopted to implement the considered adder circuits using a general purpose FPGA. For these implementations, it was decided to use the XCV2000E VIRTEX-E FPGA device [10], using the set of tools provided by Xilinx. As before, it was adopted for the parameterization of the several architectures operand widths of $w = 8$ bits.

In table III it is presented the implementation results for each analyzed topology, where the hardware resources were measured in terms of the number of required CLB slices and the number of look-up tables (LUTs) used in each implementation.

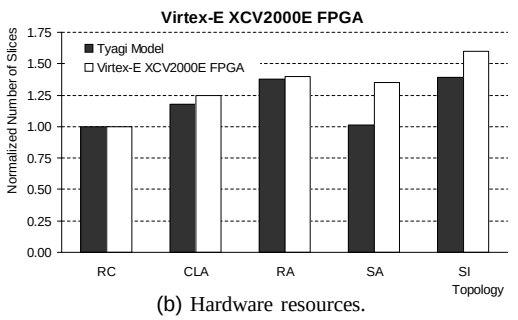
As before, it was performed a normalization of the values obtained using the model and the experimental results. It is important to note that, for this specific case, where logic functions are usually implemented using look-up tables instead of the traditional logic cells, the results obtained from this comparison should be taken with some care, since two completely different implementation supports are being compared with each other. Nevertheless, one can conclude from the charts presented in fig. 12(a) and 12(b), relating the variations of the processing time and of the hardware resources, that the model still provides reasonable prediction of the performance measures obtained with FPGA based implementations. The greatest observed differences can be found in the variation of the processing time and are justified by the completely different implementation supports: logic cells *versus* look-up tables.

TABLE III
IMPLEMENTATION RESULTS OBTAINED FOR $w = 8$ AND USING THE XCV2000E VIRTEX FPGA [10].

Topology	**Time** $[ns]$	**# Slices**	**# LUTs**
RC	4.427	20	16
CLA	4.029	25	26
RA	3.233	28	36
SA	3.233	27	32
SI	2.068	32	44



(a) Processing time.



(b) Hardware resources.

Fig. 12. Comparison of the results obtained for the processing time and hardware resources (number of CLB slices) using the theoretical model and the XCV2000E VIRTEX FPGA.

V. CONCLUSION

A detailed comparison analysis of several fast and efficient adder architectures for high performance VLSI design was presented. A theoretical model was applied to assess the performance of those adders independently of the target technology and the obtained results were then validated by using two real and entirely different implementation technologies: a StdCell library based on a $0.25\mu m$ CMOS process and a VIRTEX-E general purpose FPGA.

The obtained results showed that the adder architecture based on the radix-2 redundant format converter offered the lowest processing time when implemented with any of the considered technologies. However, this adder was also the topology that presented the highest amount of required hardware resources.

The presented theoretical and thorough analysis and its validation using two distinct real-world implementation technologies is an invaluable resource in the selection of the most appropriate adder topology that should be used to implement a given arithmetic operation in a high performance VLSI design.

REFERENCES

[1] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, volume C-31, no. 3, pp. 260–264, March 1982, reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 2, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[2] T. F. Ngai, M. J. Irwin and S. Rawat, "Regular, area-time efficient carry-lookahead adders," *Journal of Parallel and Distributed Computing*, volume 3, no. 3, pp. 92–105, 1986, reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 2, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[3] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions on electronic computers*, volume 10, pp. 389–400, 1961, reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 2, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[4] Keshab K. Parhi, "Fast VLSI binary addition," in *Proc. of 1997 IEEE Workshop on Signal Processing Systems: Design and Implementation*, Leicester, U.K., November 1997, pp. 232–241.

[5] Keshab K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, Wiley-Interscienc, 1999.

[6] Reto Zimmermann, *Binary Adder Architectures for Cell-Based VLSI and their Synthesis*, Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, 1997.

[7] J. Sklansky, "Conditional sum addition logic," *IRE Transactions on Electronic Computers*, volume EC-9, no. 6, pp. 226–231, June 1960.

[8] Akhilesh Tyagi, "A reduced-area scheme for carry-select adders," *IEEE Transactions on Computers*, volume 42, no. 10, pp. 1163–1170, October 1993.

[9] Virtual Silicon Technology Inc., *Diplomat-25 Standard Cell Library - $0.25\mu m$ UMC Process*, December 1999.

[10] Xilinx Inc., *Virtex-E 1.8V Field Programmable Gate Arrays Datasheet*, v2.3 edition, July 2002.