Metrics for Fast, Low-Cost Adders in FPGA

Daniel Simonelli and Martín Vázquez
INCA/INTIA – Facultad de Ciencias Exactas – UNICEN
Paraje Arroyo Seco, Campus Universitario – Tandil, Argentina
{dsimonel|mvazquez}@exa.unicen.edu.ar

ABSTRACT: In this paper several adder design techniques that probed to be very effective in full-custom integrated circuit design are presented as well as the conclusions regarding its implementation on FPGA.
Particularly, in this work, Xilinx XC4000E family is selected as target technology and results achieved without using dedicated carry logic present in these devices are evaluated.
This paper aims to substantiate the fact that these techniques indeed reduce delay time in other technologies than full custom design and from these results decide if it is worth trying implementations using XC4000E dedicated carry logic.

I. INTRODUCTION

Adders are of fundamental importance in a wide variety of digital systems. Many fast adders exist, but they are almost always implemented in a full-custom approach what allows taking advantage of the very nature of the technique involved. For example, if it is possible to avoid using certain space-consuming digital components as big multiplexers this produces space savings and, eventually, adds fast.
When working with FPGAs this assumption rarely is true. For one side, when using synthesis tools, is not always easy determine how a certain construction will be implemented. By other hand, when floorplanning techniques are used, topological restrictions may impose new variables other than those directly involved with the technique to implement. Nonetheless, FPGAs present some special features that may help the designer reverse an adverse scenario.
This work present metrics obtained without using this kind of special features over some of the most interesting adders found up today. In the next section these adders are presented and briefly discussed.

II. ADDER MODELS USED IN THIS WORK: AN OVERVIEW

Among of the many adders' styles that exist, those that became most general because of their best speed/size ratio in full custom designs were used for this work As can be seen, all these styles range between ripple and carry-look-ahead.
Ripple adders are the smallest but also the slowest.

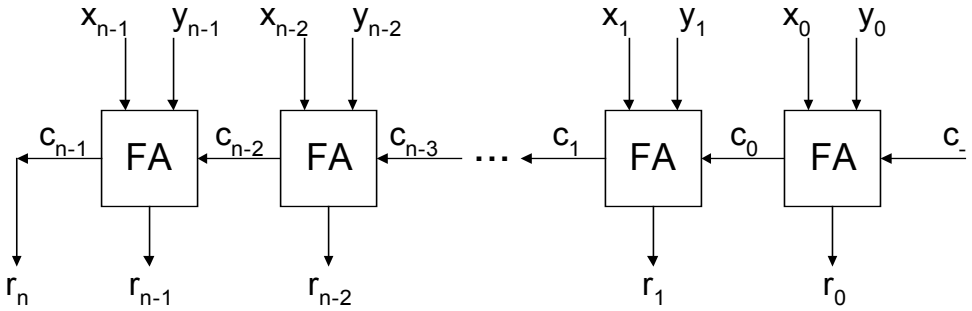

Fig. 1 – Ripple-Carry adder

Carry-look-ahead [Wei58] and carry-select adders [Bed62] are very fast but far larger and consume much more power than ripple or carry-skip adders.

More recently, carry-skip adders [Kan93a] gained popularity due to their high speed and relatively small size. The underlying idea in this adders is that, in an *N*-bit carry-skip adder divided into a proper number of k-bit blocks, a long-range carry signal starts at a generic block B_i, rippling through some bits in that block, then skips some blocks, and ends in a block B_j. If the carry does not end at the LSB of B_j then rippling occurs in that block and an additional delay is needed to compute the valid sum bits.

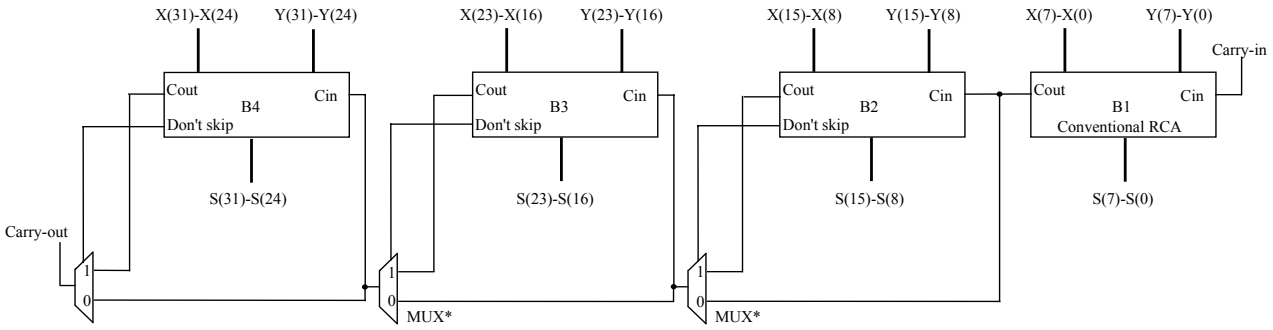

Fig. 2 – General structure of a carry-skip adder

The carry-skip adder is a pretty good circuit [Leh61][Mso61]. However, the implementation reveals that with all the skip blocks of the same size, the latter blocks finish switching quickly and then stay idle for a while waiting for the carry signal to pass through all the bypass multiplexers. For example, in Fig. 3 a diagram of a 32-bit carry-skip adder is shown where the carry-out for bits 4-7 will be ready at the same time as the carry-out for bits 0-3. This second block will stay around doing nothing while the first multiplexer does its job.

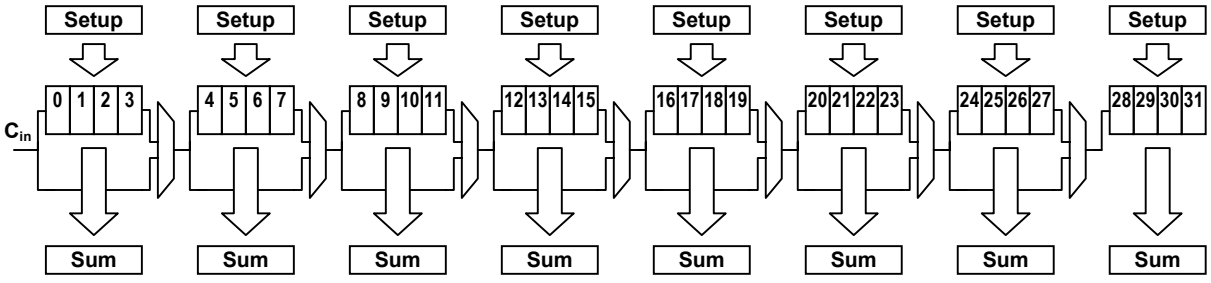

Fig. 3 – 32-bit carry-skip adder with block size 4

Let's analyze the worst case delay for a simple n-bit carry skip adder with k-bit blocks. In that situation, the delay time consists of:
i – the time of propagating the carry-in throughout the initial block, i.e. $t_{sum} = k - 1$
ii – multiplexer time, i.e. $t_{mux} = 0.5$
iii – inner blocks skip time. Let $m = n/k$ be the number of blocks, then $t_{prop} = m - 2$
iv - the time of propagating the carry-in throughout the final block, i.e. $t_{sum} = k - 1$

$t_{delay} = t_{sum} + t_{mux} + t_{prop} + t_{sum} =$
$= (k - 1) + 0.5 + (n/k – 2) + (k – 1) = 2 (k -1) + 0.5 + (n/k – 2) =$
$= 2k – 3.5 + n/k$
$= 2k + m – 3.5$

Now, it is possible to find out which is the optimal block size:

$dt/dk = 0 \Rightarrow k_{opt} = (n/2)^{1/2}$

And the optimal number of blocks

$m_{opt} = (2n)^{1/2}$

Therefore, the optimal delay time with fixed block width is:

$tdelay_{opt} = 2 (n/2)^{1/2} + (2n)^{1/2} – 3.5 =$
$= (4n/2)^{1/2} + (2n)^{1/2} – 3.5 =$
$= (2n)^{1/2} + (2n)^{1/2} – 3.5 =$
$= 2 (2n)^{1/2} – 3.5$

$$\boxed{tdelay_{opt} = 2 (2n)^{1/2} – 3.5} \qquad [1]$$

To speed up the circuit, the size of the skip blocks may be variable. In the case in which the size of two consecutive blocks differ by 1, the previous analysis can be generalized considering that:

$n = m + (m + 1) + ... + (m + k/2 – 1) + (m + k/2 – 1) + ... + (m + 1) + 1$

Then, $k = (n/m) – (m/4) + 0.5$ [2]

$t_{delay} = t_{sum} + t_{mux} + t_{prop} + t_{sum} = 2(k – 1) + 0.5 + (m – 2) = 2k – 3.5 + m$ [3]

Replacing [2] in [3] yields, $\boxed{t_{delay} = 2(n/m) – 2(m/4) + 1 – 3.5 + m = 2(n/m) + m/2 – 2.5}$

The optimal number of blocks comes from $dt/dm = 0 \Rightarrow m_{opt} = 2n^{1/2}$

The optimal block size is $k_{opt} = \lceil 1/2 \rceil = 1$ at the ends and goes up to $n^{1/2}$ en $m_{opt}/2$

Therefore, the optimal delay time with variable block width is:

$$tdelay_{opt} = 2k_{opt} + m_{opt} – 3.5 = 2 (1/2) + 2n^{1/2} – 3.5 = 2n^{1/2} – 2.5$$

$$\boxed{tdelay_{opt} = 2(n)^{1/2} – 2.5} \qquad [4]$$

Comparing [1] and [4] can be concluded that variable block width is $2^{1/2}$ (about 40%) times faster than fixed block width.

Oklobdzija and Barnes [Okl88], established that the optimal configuration for a 32-bit adder is that shown in fig. 2

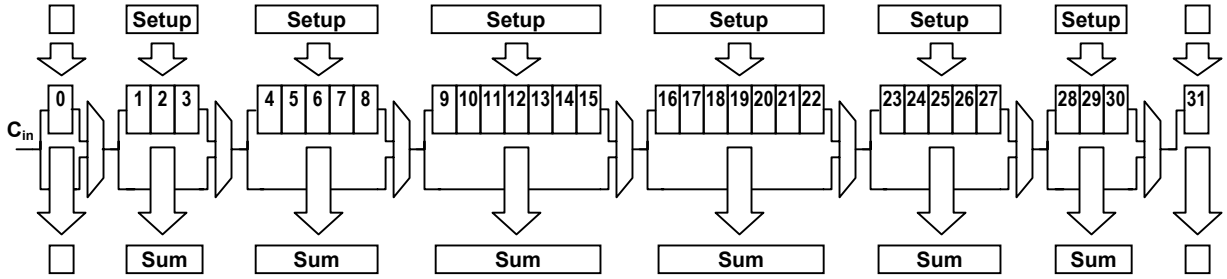

Fig. 4 – 32-bit carry-skip adder with variable block size

Corsonello, Kantabutra and Perri [Cor00] presented a new bit block structure that computes propagate signals called "carry-strength" in a ripple fashion, giving rise to a family of new carry-skip adders that are significantly faster than traditional carry-skip adders while not much larger.
In this new type of carry-skip adder, the new block structure eliminates the delay due to the rippling at the end of the life of a long-range carry signal. The main idea is, that for each bit position k in a block B_j we compute whether the carry-in to position k comes from the carry-in to block B_j, or whether this carry is internally generated in block B_j. To this purpose a new type of bit block is used, in which propagate signals that start at the LSB of the block and end at every bit position are calculated. Fig. 5 shows an 8-bit block such that. Hereafter, we'll refer to this adder as Carry-Skip-CKP.

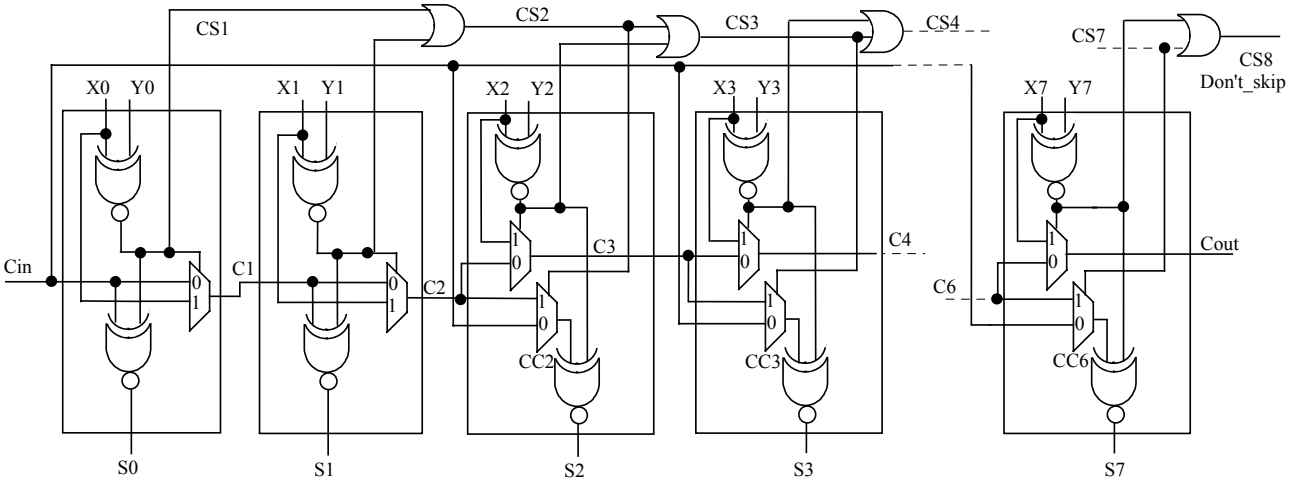

Fig. 5 – 8-bit block for carry-skip-CKP adders

Two of the fastest known addition circuits are the Lynch-Swartzlander's [Lyn92] and Kantabutra's [Kan93b] hybrid carry-look-ahead adders. They are based on the usage of a carry tree [Skl60] that produces carries into appropriate bit positions without back propagation. In order to obtain the valid sum bits as soon as possible, in both Lynch-Swartzlander's and Kantabutra's adders the sum bits are computed by means of carry-select blocks, which are able to perform their operations in parallel with the carry-tree.

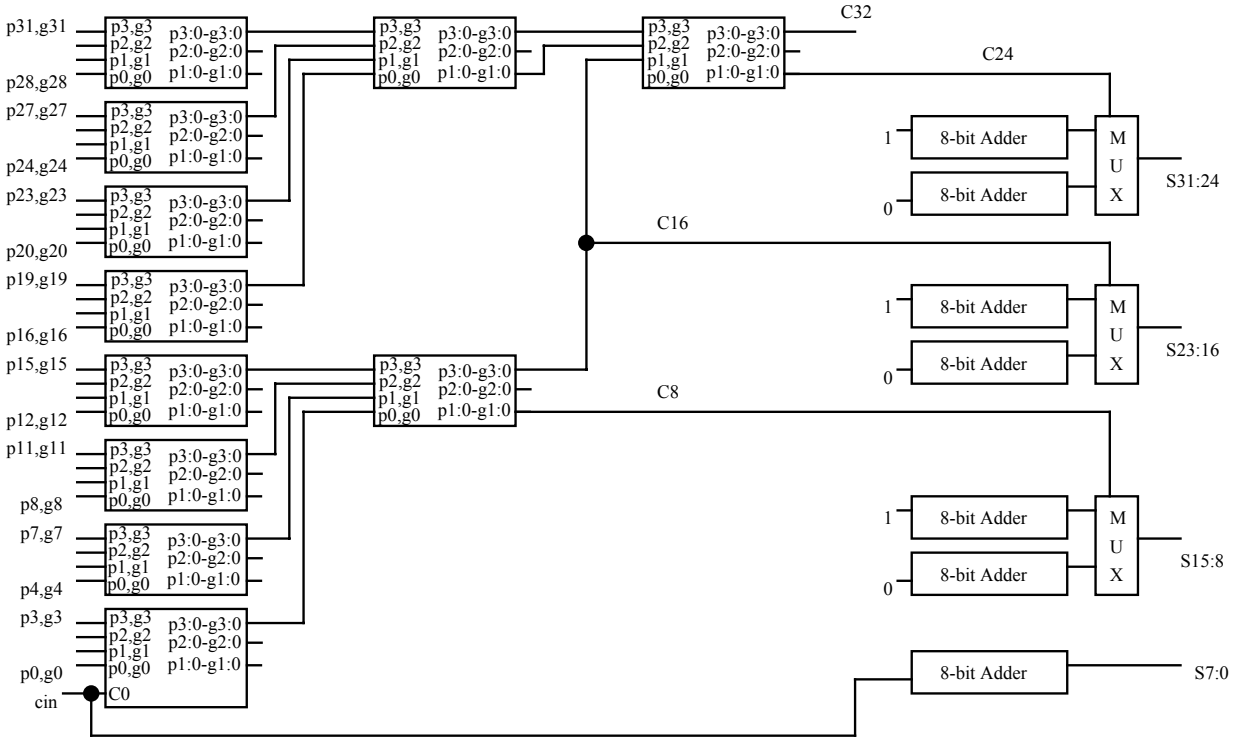

Fig. 6 – Carry tree of Lynch-Swartzlander's adder

III. FULL CUSTOM RESULTS FOR 32-BIT WIDE OPERANDS

In table 1, data obtained for full-custom adder designs are reported. Obviously, it is not possible to contest to these values by means of FPGA technology but through the techniques discussed in the previous section, it does make feasible to succeed in the quest for better performances in FPGA-adders. Values belong to adders materialized using Austria Mikro Systeme p-sub, 2-metal, 1-poly, 5 V, 0.6μm CMOS process (CUB process)[Cor00]

32-bit adders	Area [μm^2]	Delay [ns]
Ripple-Carry	137700	15.80
Carry-Skip (8)	160173	9.00
Carry-Skip-CKP (8)	181944	5.80
Lynch-Swartzlander	419244	4.08

Table 1 – Performances in full-custom implementations

Carry-Skip-CKP with 8-bit blocks yields a speed about 3 times faster than Ripple-Carry with as little as a 30% area increase. Lynch-Swartzlander is even faster but area involved is thrice that of Ripple-Carry. It is worth noting that, because of the nature of space when speaking in FPGA terms, this performance tendencies not necessarily are to be found in these devices implementations. To establish what effectively happens, is the main purpose of this work.

IV. RESULTS

Results obtained in successive implementations for the different adders selected are shown below. From the study above, two adders became reference patterns for comparing:

- Ripple Carry because of its simplicity. Comparations with this adder are used to settle down if techniques adopted effectively improve performance when implemented on FPGA.
- The adder implementation provided by the synthesis tool, in this case Synopsys in Foundation 3.1i environment, because these adder library macros use the dedicated carry logic features of XC4000E devices. Contrasting against this adder, the prospective advantage through the use of floorplanning techniques in FPGA are estimated. Throughout this work it will be referred as "Library adder".

The remaining adders implemented are, as follows:

- Carry-Skip with 8-bit blocks
- Carry-Skip with 16-bit blocks
- Carry-Skip CKP with 8-bit blocks
- Carry-Skip CKP with 16-bit blocks
- Lynch-Swartzlander

Adder Type	XC4003E-PC84	XC4005E-PC84
Library	29.9	31.9
Ripple-Carry	40.2	35.9
Carry-Skip (4)	30.9	41.8
Carry-Skip (8)	36.5	37.9
Carry-Skip-CKP (4)	33.4	37.1
Carry-Skip-CKP (8)	36.9	46.7
Lynch-Swartzlander	33.6	39.6

Table 2 – Speed for 16-bit adders (nS)

In Table 2 are summarized the speed (worst-case) values obtained for 16-bit adders implemented in two XC4000E devices and table 5 shows CLB occupation. In XC4003E, the best behavior (excepting, of course Library Adder) is rendered by Carry-Skip adder with 4-bit blocks while in XC4005E the best performance belongs to Ripple-Carry from every point of view: speed and CLB occupation. As a matter of fact, the bigger is the FPGA size, the greater are the delays imposed for the inner routing. These delays may be overcame introducing local CLBs constraints but it is not the goal of the present work.

Adder Type	XC4005E-CB164	XC4010E-BG225	XC4025E-HQ304
Library	33.2	37.2	40.3
Ripple-Carry	47.4	60.7	65.2
Carry-Skip (8)	47.0	55.2	61.7
Carry-Skip (16)	43.8	51.9	62.5
Carry-Skip-CKP (8)	44.0	54.8	68.5
Carry-Skip-CKP (16)	42.4	50.2	68.7
Lynch-Swartzlander	40.4	52.8	64.5

Table 3 – Speed for 32-bit adders (nS)

For 32-bit adders, larger FPGAs were used as presented in Tables 3 and 6. When implementations were performed on a XC4005E the best speed performance was that of Lynch-Swartzlander's adder even though it also was the most space consuming. In this latter sense, Carry-Skip with 8-bit block was the cheapest implementation.
Carry-Skip-CKP with 16-bit blocks got the best speed on XC4010E and regarding space Carry-Skip with 8-bit block was again the less expensive implementation. This same adder was the best both in speed and space when implemented on a XC4025E.

Adder Type	XC4025E-HQ304
Library	54.7
Ripple-Carry	77.9
Carry-Skip (8)	87.8
Carry-Skip (16)	77.9
Carry-Skip (32)	92.5
Carry-Skip-CKP (8)	88.2
Carry-Skip-CKP (16)	100.6
Lynch-Swartzlander	76.3

Table 4 – Speed for 64-bit adders (nS)

At last, Tables 4 and 7 show that for 64-bit adders, the only available FPGA that fitted such operand requirements was the XC4025E on which the best performance was that of Lynch-Swartzlander. Its speed slightly advantaged those of Ripple-Carry and Carry-Skip with 16-bit block while the best occupation ratio was that of Ripple Carry.

Adder Type	XC4003E-PC84		XC4005E-PC84	
	CLB	IOB	CLB	IOB
Library	9 (9%)	50 (81%)	9 (4%)	50 (81%)
Ripple-Carry	35 (35%)	50 (81%)	35 (17%)	50 (81%)
Carry-Skip (4)	38 (38%)	50 (81%)	38 (198%)	50 (81%)
Carry-Skip (8)	39 (39%)	50 (81%)	39 (19%)	50 (81%)
Carry-Skip-CKP (4)	42 (42%)	50 (81%)	42 (21%)	50 (81%)
Carry-Skip-CKP (8)	57 (57%)	50 (81%)	57 (29%)	50 (81%)
Lynch-Swartzlander	46 (46%)	50 (81%)	46 (23%)	50 (81%)

Table 5 – CLB occupation for 16-bit adders

Adder Type	XC4005E-CB164		XC4010E-BG225		XC4025E-HQ304	
	CLB	IOB	CLB	IOB	CLB	IOB
Library	17 (8%)	98 (87%)	17 (4%)	98 (61%)	17 (1%)	98 (38%)
Ripple-Carry	81 (41%)	98 (87%)	81 (20%)	98 (61%)	81 (7%)	98 (38%)
Carry-Skip (8)	76 (38%)	98 (87%)	76 (19%)	98 (61%)	76 (7%)	98 (38%)
Carry-Skip (16)	80 (40%)	98 (87%)	80 (20%)	98 (61%)	80 (7%)	98 (38%)
Carry-Skip-CKP (8)	91 (46%)	98 (87%)	91 (22%)	98 (61%)	91 (8%)	98 (38%)
Carry-Skip-CKP (16)	88 (44%)	98 (87%)	88 (22%)	98 (61%)	88 (8%)	98 (38%)
Lynch-Swartzlander	101 (51%)	98 (87%)	101 (25%)	98 (61%)	101 (9%)	98 (38%)

Table 6 – CLB occupation for 32-bit adders

Adder Type	XC4025E-HQ304	
	CLB	IOB
Library	33 (3%)	194 (75%)
Ripple-Carry	169 (16%)	194 (75%)
Carry-Skip (8)	175 (17%)	194 (75%)
Carry-Skip (16)	183 (17%)	194 (75%)
Carry-Skip (32)	175 (17%)	194 (75%)
Carry-Skip-CKP (8)	187 (18%)	194 (75%)
Carry-Skip-CKP (16)	172 (18%)	194 (75%)
Lynch-Swartzlander	185 (18%)	194 (75%)

Table 7 – CLB occupation for 64-bit adders

V. CONCLUSIONS

Although the speed values for the different adders implemented are not so diverse, occupation factors are really unlike, particularly when they are contrasted with those yielded through the automatic synthesis tool.

Table 8 shows that while the speed difference between the average and Library adders is relatively low in most cases, the CLB occupation difference is high in all cases. Therefore, it is perfectly acceptable to expect that if XC4000E's dedicated carry logic is used, occupation factors for these techniques decrease greatly as well as it will be possible to achieve a significant increase in computation speed. For that reason, it is full worthy to try the mentioned implementation as a future work.

	Speed					CLB Occupation			
	Library	Avg.	S.D.	Dif.	Adder size	Library	Avg	SD	Dif.
XC4003-PC84	29.90	30.68	3.28	0.78	16-bit	9	43	8	34
XC4005-PC84	31.90	34.71	3.94	2.81	16-bit	9	43	8	34
XC4005-CB164	33.20	38.24	2.68	5.04	32-bit	17	87	10	70
XC4010-BG225	37.20	47.04	3.66	9.84	32-bit	17	87	10	70
XC4025-HQ304	40.30	56.29	2.94	15.99	32-bit	17	87	10	70
XC4025-HQ304	54.70	76.28	9.03	21.58	64-bit	33	177	7	144

Table 8 – Relative comparison

VI. REFERENCES

Bed62 Bedrij, O. 1962. “Carry select adder.” IRE Transactions on Electronic Computers , vol. 11, pp. 340–346. Original reference to carry-select adder. See also [Weste, 1993] p. 532. [p. 84]

Cor00 P. Corsonello, V. Kantabutra, S. Perri, "Fast, Low-Cost Adders Using Carry Strength Signals", SSGRR 2000, l'Aquila, Italy, July 31-Aug 6

Kan93a V. Kantabutra, “Designing optimum one-level carry-skip adders”, IEEE Trans. on Comp., 1993, Vol. 42, n°6, pp.759-764.

Kan93b V. Kantabutra, “A recursive carry-look-ahead/carry-select hybrid adder”, IEEE Trans. on Comp., Vol. 42, n°12, Dec. 1993.

Leh61 M. Lehman and N. Burla, "Skip Technique for High Speed Carry Propagation in Binary Arithmetic Units", IRE Transactions on Electronic Computers, Vol. EC-10, 1961, pp. 691-698

Mso61 O. L. MacSorley, "High-Speed Arithmetic in Binary Computers", Proceedings of the IRE, Vol. 49, 1961, pp. 67-91

Lyn92 T. Lynch, E.E. Swartzlander, “A spanning-tree carry-look-ahead adder”, IEEE Trans. on Comp., Vol. 41, n°8, Aug. 1992.

Okl88 Vojin G. Oklobdzija, Earl R. Barnes: On Implementing Addition in VLSI Technology. Journal of Parallel and Distributed Computing 5(6): 716-728 (1988)

Skl60 J. Sklansky, "Conditional-Sum Addition Logic", IRE Transactions on Electronic Computers, Vol. EC-9, 1960, pp. 226-231

Wei58 A. Weinberger and J. L. Smith, "A Logic for High-Speed Addition", National Bureau of Standards Circular, No. 591, 1958, pp. 3-12