A1CSA: An energy-efficient fast adder architecture for cell-based VLSI design

Conference Paper · December 2011

DOI: 10.1109/ICECS.2011.6122308 · Source: DBLP

3 authors:

Jucemar Monteiro
Universidade Federal do Rio Grande do Sul
10 PUBLICATIONS **34** CITATIONS
SEE PROFILE

José Luís Almada Güntzel
Federal University of Santa Catarina
87 PUBLICATIONS **206** CITATIONS
SEE PROFILE

Luciano Volcan Agostini
Universidade Federal de Pelotas
324 PUBLICATIONS **1,709** CITATIONS
SEE PROFILE

A1CSA: An Energy-Efficient Fast Adder Architecture for Cell-Based VLSI Design

Jucemar Monteiro, José Luís Güntzel
Embedded Computing Lab. - Computer Sciences Department
Federal University of Santa Catarina
Florianópolis, Brazil
{jucemar,guntzel}@inf.ufsc.br

Luciano Agostini
Group of Architectures and Integrated Circuits (GACI)
Federal University of Pelotas
Pelotas, Brazil
agostini@inf.ufpel.edu.br

***Abstract*—Energy-efficient fast adders are needed in the design of battery-powered portable devices. Although many fast adder architectures exist, most of them require transistor-level optimizations that prevent their synthesis in a standard-cell flow. This paper presents two energy-efficient Add-One Carry-Select Adders (A1CSA and A1CSAH) suited for standard-cells synthesis. Synthesis results showed that the A1CSA is the smallest fast adder requiring, on average, 22.2% less area than the Carry-Select Adder. They also showed that the A1CSAH is, on average, 10.8% faster and 3.4% more energy-efficient than the Carry-Lookahead Adder, thus corresponding to the best choice for high speed and high efficiency addition.**

I. INTRODUCTION

Addition is the most commonly used arithmetic operation within contemporary electronic systems. It has great importance not only in general purpose CPUs, but also in acceleration blocks, as part of arithmetic circuits. Besides used as a proper operation, it serves as the basis for many other arithmetic operations.

The choice of which adder architecture to use is of utmost importance, since the performance of adders may determine the whole system performance [1]. Area and power consumption are also relevant figures of merit to be considered, especially when the design targets VLSI realization. Recently, energy-efficiency has also become an important metric due to the dramatic growth of battery-powered portable device marked.

Most works addressing adder architectures are focused on critical delay reduction by optimizing the carry propagation chain [1-3]. Such optimization can be accomplished at the logic-level, at transistor-level or at both. Transistor-level optimizations may use pass transistors, dynamic logic, etc. On the other hand, conventional physical design flow relies on standard-cells for layout generation. Although typical standard-cells libraries contain up to hundreds of cells, they are all designed using static CMOS style. The inclusion of user-created cells with other styles, as pass transistor or dynamic logic, is generally not allowed due to limitations of the design flow itself. Therefore, when designing high performance addition-based arithmetic circuits using cell-based VLSI design, designers must rely on fast adder architectures that are optimized at the logic-level, which reduces the number of fast adder architecture to a few classic ones, as the Carry-Select Adder (CSA) [4] and the Carry-Lookahead Adder (CLA) [5].

In this paper we propose the use of the Add-One Carry-Select Adder (A1CSA) [6-7] in the cell-based design flow. The A1CSA has originally been proposed as a low-cost version of the CSA. Later on, a low power version of the A1CSA was also presented [8]. However, in order to achieve area/transistor reduction, those A1CSA circuits explore pass transistor logic, which prevents their implementation in a standard-cells design flow. In a previous work, we have proposed and evaluated a logic-level optimized A1CSA version targeting FPGA-based design [9].

This paper presents as main contribution two versions of A1CSA, especially tailored for standard-cells based VLSI design. While both versions target energy-efficiency, one required the smallest amount of area among all investigated fast adders, while the other achieved the highest speed and efficiency among all fast adders.

The remaining of this paper is organized as follows. Section II presents the original A1CSA architecture and comments on its synthesis in a cell-based design flow. Section III presents the proposed optimizations to the A1CSA, which gave rise to two A1CSA circuits. Synthesis results are presented and analyzed in Section IV. Section IV draws the most relevant conclusions of this work.

II. FAST ADDER ARCHITECTURES FOR CELL-BASED DESIGN

The Carry-Ripple Adder (CRA) is known as the most area efficient, and at the same time, the slowest adder architecture [2]. It has both time and area complexities $\mathbf{O}(n)$, where n is the bit-width of each input operand [3]. In the CSA, addition is divided into m modules of k bits each. Each module has two k-

This work was partially supported by the Brazilian Council for Scientific and Technological Development (CNPq).

bit wide adders (generally, CRAs), which perform two additions at the same time: one with CIN=0 (CIN means carry-in) and another with CIN=1. Such degree of parallelism makes the CSA one of the fastest adder architectures. On the other hand, the CSA area is more than twice as that of the CRA, for a given bit-width. The time and area complexities of the CSA are **O**(n/k) and **O**($2n$), respectively. In the CLA, the addition is divided into m modules of k bits (usually k=4, for fanout reasons). The carries within each module are computed in parallel by using "generate" and "propagate" in order to anticipate the module carry out. When CLA modules are connected in hierarchical manner (as assumed in the current work), the CLA presents time complexity **O**($\log_k n$) [3] and area complexity **O**($n\log n$) [10].

The A1CSA is inspired by the CSA. However, as shown in Fig. 1, the A1CSA has half the number of adders that the CSA. This is because the adder with CIN=1 in a CSA is replaced by a less expensive logic, known as "add-one logic" or "A1" block. Most related works optimize the A1 block in the transistor-level using non-static CMOS logic (e.g., pass transistors) [6-8]. However, the design at the transistor-level prevents the standard-cells based A1CSA realization.

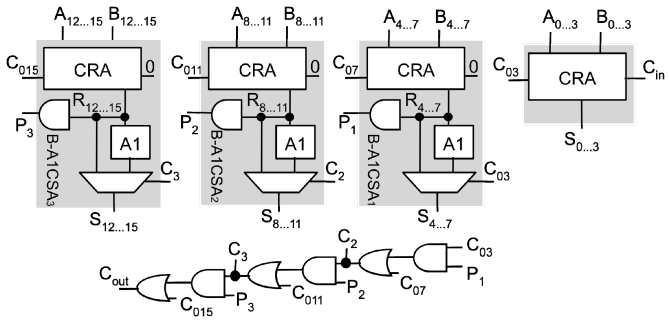

Figure 1. Original 16-bit Add-One Carry-Select Adder (A1CSA).

In this work we consider logic-level implementations only of the adders. Therefore, in the A1CSA case, the A1 block, the multiplexer, the carry propagation logic and addition modules themselves are optimized only at the logic-level. On the other hand, the optimization of the A1 block is of utmost relevance for the area, speed and energy efficiency of the A1CSA. Therefore, we start by the synthesis of the A1 block, which is to take into account the following properties of the binary addition [7] [9].

Property 1*: In the addition of two n-bit operands with the least significant bit being "0" for both operands, if the carry-in bit is complemented, then the LSB of the addition is complemented and the other bits remain unchanged.*

Property 2*: If the addition of two n-bit operands with a carry-in of "0" has m "1s" before the first occurrence of a "0" (starting from the LSB), then the least significant m+1 bits of the addition with a carry-in of "1" will have values complementary to the first m+1 bits of the addition with carry-in "0".*

Fig. 2 shows examples for Properties 1 and 2. Using these properties to synthesize the A1 block, we arrive to the simplified (unmapped) equations (1), (2), (3) and (4), where R_i identifies the bit i of the CRA adder result.

$$S_0 = R_0 \quad (1)$$

$$S_1 = R_0 \oplus R_1 \quad (2)$$

$$S_2 = R_0 \,.\, R_1 \oplus R_2 \quad (3)$$

$$S_3 = R_0 \,.\, R_1 \,.\, R_2 \oplus R_3 \quad (4)$$

Cin	0		Cin	1	
A	1 1 0 0	(12)	A	1 1 0 0	(12)
B	0 1 0 0	(4)	B	0 1 0 0	(4)
Output	1 0 0 0 **0**	(16)	Output	1 0 0 0 **1**	(17)

(a)

Cin	0		Cin	1	
A	0 0 0 1	(1)	A	0 0 0 1	(1)
B	0 1 1 0	(6)	B	0 1 1 0	(6)
Output	0 **1 1 1**	(7)	Output	**1 0 0 0**	(8)

(b)

Figure 2. Examples of Property 1 (a) and Property 2 (b).

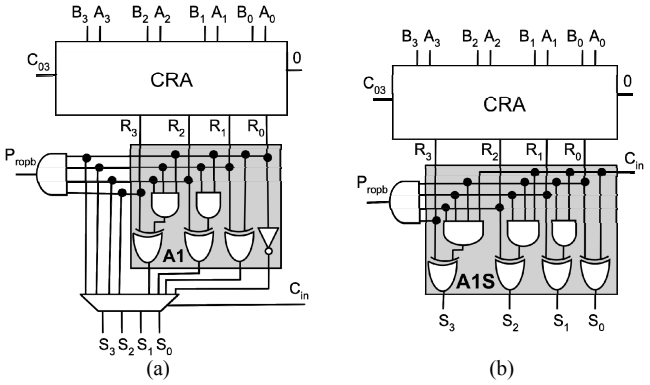

Figure 3. A1 block (a) and A1S block (b).

III. NEW A1CSA CIRCUITS FOR CELL-BASED DESIGN

Fig. 3(a) shows the A1 block proposed in our previous work [9]. By analyzing such circuit and the related equations, one can notice that it is possible to incorporate the result selection into the XOR gate, therefore eliminating the multiplexer. The contribution of such optimization is twofold: it reduces the complexity of the A1 block (resulting in less area and less power consumption) and, not less important, it reduces the capacitive charge at the inputs of the logic that generates the "Propb" signal. Since such signal serves to accelerate the carry propagation, this has an important impact on the circuit critical delay. The modified A1 block, called "A1S" block, is shown in Fig. 3(b).

The A1S block was used to replace both the A1 block and the multiplexer in the A1CSA of Fig. 1, thus giving rise to the A1CSA version used in this work for cell-based synthesis. It is important to notice that in such new A1CSA version, each module is 4-bit wide (k=4) and is realized as a CRA. A second version of A1CSA, focusing in high speed addition, was also developed. This new adder, presented in Fig. 4 and hereafter referred to as A1CSAH, also uses the A1S block. However, the carry propagation logic is computed in a hierarchical manner, by a dedicated block called GPH, resulting in a logarithmic dependence on the number of addition modules.

This provides high speed carry propagation, but at the cost of extra hardware resources. Also to reinforce the high speed features, we have chosen the CLA (instead CRA) to perform each 4-bit addition in the A1CSAH. This also increases resource use.

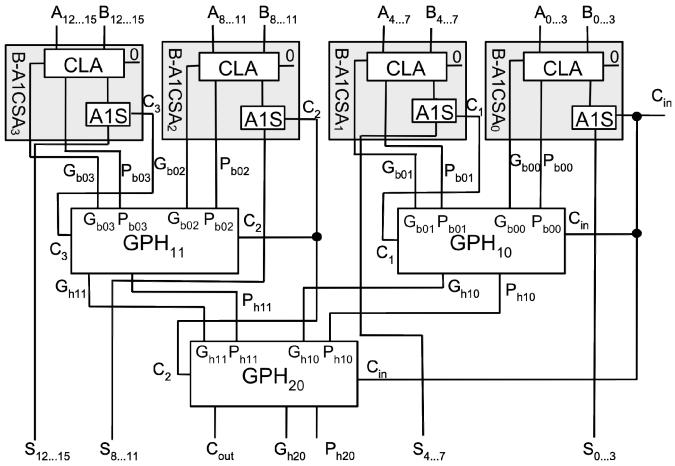

Figure 4. A 16-bit A1CSAH.

The implementation of other fast adder architectures at the logic level is straightforward, except for the choice of the carry propagation chain, in the case of the CLA. To provide fair comparisons, in this work we assume a hierarchical carry propagation chain for the CLA, as that shown in Fig. 4.

IV. Synthesis Results and Analysis

To evaluate the two proposed adder architectures and to compare them to other fast adders, 8, 16, 32, 64, 128 and 256-bit wide A1CSA, A1CSAH, CSA and CLA were described in Verilog and synthesized for 45 nm TSMC standard-cells library using Synopsys' Design Compiler (DC) [10]. To provide realistic post-layout timing and power estimates, we used the "Topographical" mode of Synopsys DC. The CRA was also synthesized for the same bit-widths to obtain reference data.

The CSA and the A1CSA were built up from 4-bit wide CRA modules, whereas the CLA and the A1CSAH were built up from 4-bit wide CLA modules. Both the CSA and the A1CSA employ the carry propagation logic as shown in Fig. 1. The CLA uses hierarchical carry propagation logic similar to that employed in the A1CSAH (as shown in Fig. 4). The 4-bit adder modules, as well as A1, A1S and other dedicated blocks (as those used in A1CSAH and CLA carry propagation) were described as separate Verilog files, so as to prevent Synopsys DC from performing optimizations that could degenerate fast adder architectures (e.g., converting CLA into CRA). We also instruct Synopsys DC to map every CRA using the full adder (FA) cell available from the standard-cells library, thus providing the best possible CRA realizations within the standard-cells based design flow.

TABLE I shows area estimates for the investigated adder architectures, as provided by Synopsys DC. As one would expect, the CRA requires the smallest amount of area among all architectures. Therefore, its area was used as baseline to normalize fast adders area, as shown in TABLE II. Among the fast adders, the A1CSA requires the smallest amount of area to be implemented (50% more than the CRA, on average), whereas the A1CSAH requires the largest amount of area (119% more than the CRA, on average). In comparison to the A1CSA and to the A1CSAH, the CLA and the CSA require an intermediate amount of area to be implemented (on average, 93% and 103% more than the CRA, respectively). In a direct comparison, the A1CSA requires 22.2% less area than the CLA, on average. However, for bit-widths between 32 and 256, the CSA needs approximately the same area as the A1CSAH, whereas the A1CSA remains as the most compact fast adder architecture for cell-based design, requiring 19.4% less area than the CLA, on average.

TABLE I. Adders area [m^2]

Adder	Bit-width					
	8	**16**	**32**	**64**	**128**	**256**
CRA	35.3	70.6	141.1	282.2	564.5	1129.0
CSA	57.0	135.7	293.0	607.7	1237.1	2495.9
A1CSA	45.9	102.3	215.2	441.0	892.6	1795.8
CLA	66.5	135.3	272.9	548.1	1098.4	2199.2
A1CSAH	75.7	153.6	309.6	621.5	1245.2	2492.7

TABLE II. Normalized Fast adders area

Adder	Bit-width						Aver.
	8	**16**	**32**	**64**	**128**	**256**	
CSA	1.62	1.92	2.08	2.15	2.19	2.21	2.03
A1CSA	1.30	1.45	1.53	1.56	1.58	1.59	1.50
CLA	1.89	1.92	1.93	1.94	1.95	1.95	1.93
A1CSAH	2.15	2.18	2.19	2.20	2.21	2.21	2.19

TABLE III shows Synopsys DC estimates for the critical delays. As expected, the CRA exhibits the longest delay for all bit-widths. To allow for a broad comparison, the fast adders critical delays are shown in TABLE IV as normalized values, assuming the critical delays of CRA as reference. Among the fast adder architectures, the A1CSAH is the fastest one (67% as fast as the CRA, on average), followed by the CLA (62% as fast as the CRA, on average). The A1CSA and the CSA are the slowest ones (on average, 53% and 52% as fast as the CRA, respectively). Not surprisingly, the A1CSA and the CSA exhibit a similar timing behavior. Indeed, this is because both CSA and (our version of A1CSA) use 4-bit wide CRAs as building blocks. Moreover, they employ the same carry propagation logic. On the other hand, for bit-widths between 32 and 256, the A1CSA is 12.8% faster than the CSA, on average. Such speedup is due to the A1S block, which replaces the multiplexor and the A1 block in the original A1CSA, thus providing delay reduction. The A1CSAH and the CLA have also similar delay behavior because they both use 4-bit wide CLAs as building blocks, besides employing similar hierarchical carry propagation logic. However, since in the A1CSAH the carry propagates through A1S blocks (in the CLA, it propagates through CLA blocks, which are less optimized), the A1CSAH is, on average, 10.8% faster than the CLA. For bit-widths between 32 and 256, the A1CSAH is still 10.3% faster than the CLA, which makes it the fastest cell-based adder architecture among all investigated adders.

TABLE III. ADDERS CRITICAL DELAY [ns]

Adder	Bit-width					
	8	16	32	64	128	256
CRA	406.4	740.1	1415.5	2681.1	5187.8	10316.4
CSA	293.8	386.9	618.8	1037.0	1860.7	3481.6
A1CSA	365.9	440.6	608.1	883.5	1550.3	2855.7
CLA	356.2	430.9	516.2	595.9	685.9	785.2
A1CSAH	312.9	381.6	453.5	535.3	626.4	706.2

TABLE IV. NORMALIZED FAST ADDERS CRITICAL DELAY

Adder	Bit-width						Aver.
	8	16	32	64	128	256	
CSA	0.72	0.52	0.44	0.39	0.36	0.34	0.46
A1CSA	0.90	0.60	0.43	0.33	0.30	0.28	0.47
CLA	0.88	0.58	0.36	0.22	0.13	0.08	0.38
A1CSAH	0.77	0.52	0.32	0.20	0.12	0.07	0.33

We used the critical delay and the power estimates given by Synopsys DC to compute the adders power-delay product (PDP), as presented in TABLE V. For a given bit-width, the PDP corresponds to the amount of energy required by the adder to perform one addition, when operating at its maximum speed. Considering the 16 to 256 bit-width range, the CRA is the adder architecture with highest PDP and hence, the least efficient one. TABLE VI presents normalized PDP values for the fast adders, assuming CRA PDP values as reference. Relying on these data, the A1CSAH and the CLA appear as the two most efficient fast adder architectures, followed by the A1CSA. The A1CSAH and the CLA are, on average, 48% and 46% more efficient than the CRA, whereas the A1CSA is, on average, 41% more efficient than the CRA. The CSA appears as the least efficient fast adder, being only 18% more efficient than the CRA. In particular, when we exclude 8-bit and 16-bit wide adders from the analysis, the relative efficiency increases considerably for the three most efficient adders, A1CSAH, CLA and A1CSA, they become, on average, 71%, 70% and 57% more efficient than the CRA, respectively. In a direct comparison between the A1CSA and the CSA, the former appears as 31.9% more efficient than the latter. Comparing the two most efficient fast adders, the A1CSAH is 3.4% more efficient than the CLA, on average, becoming thus the best fast adder choice in terms of energy efficiency.

TABLE V. ADDERS PDP [fJ]

Adder	Bit-width					
	8	16	32	64	128	256
CRA	9.5	906.6	3469.0	13140.7	50854.2	202334.3
CSA	9.0	846.1	2905.3	10071.7	36749.9	138649.0
A1CSA	9.9	670.7	1910.7	5638.8	19953.3	73796.0
CLA	10.8	799.9	1933.9	4484.0	10344.1	23715.4
A1CSAH	10.5	764.6	1832.1	4343.4	10188.8	22991.5

TABLE VI. NORMALIZED FAST ADDERS PDP

Adder	Bit-width						Aver.
	8	16	32	64	128	256	
CSA	0.95	0.93	0.84	0.77	0.72	0.69	0.82
A1CSA	1.04	0.74	0.55	0.43	0.39	0.36	0.59
CLA	1.13	0.88	0.56	0.34	0.20	0.12	0.54
A1CSAH	1.11	0.84	0.53	0.33	0.20	0.11	0.52

V. CONCLUSION

This paper presented two versions of Add-One Carry-Select Adder (A1CSA), optimized for cell-based VLSI design flow. These architectures, along with the CRA and other classic fast adder architectures (CSA and CLA) were synthesized for 45 nm TSMC standard-cells library by using Synopsys Design Compiler in Topographical mode.

Synthesis results showed that, for bit-widths between 8 and 256, the A1CSA is the best choice for cell-based design when area is a limiting factor. According to those results, the A1CSA appeared as the most compact fast adder, requiring just 50% more area than the CRA. The A1CSA is, on average, 22.2% smaller than the CLA, which appeared as the second best choice. On the other hand, the A1CSA showed to be, on average, 12.8% faster and 31.9% more energy efficient than the CSA, which would be its direct competitor. The synthesis results also pointed out the A1CSAH as the best choice for cell-based design when both high speed and high energy-efficiency are required. In particular, the A1CSAH showed to be, on average, 10.8% faster and 3.4% more energy efficient than the CLA, which appears as the second best choice for the same figures of merit. Finally, the presented fast adder architectures can help the designer to meet area, delay and energy efficiency by chosen fast adder architectures that may be more appropriate for arithmetic-intensive integrated systems, as for instance, those found in signal processing and video coding.

REFERENCES

[1] J. M. Rabaey, A. Chandrakasan, B. Nikolic, Digital Integrated Circuits: a design perspective, 2nd ed., Upper Saddle River, N. J.: Prentice Hall, 2003, pp.559-586.

[2] M. D. Ercegovac and T. Lang, Digital Arithmetic, San Francisco: Elsevier Science, 2004.

[3] V. Oklobdzija, "High-speed VLSI arithmetic units: adders and multipliers," in Design of High-Performance Microprocessor Circuits, A. Chandrakasan, W. J. Bowhill and F. Fox, Eds. Piscataway, N. J.: IEEE Press, 2001, pp. 181-204.

[4] O. J. Bedrij, Carry-Select Adder, In: IRE Transactions on Electronic Computers, pp. 340, 1962.

[5] A., Weinberger and J.L. Smith, A Logic for High-Speed Addition, In: National Bureau of Standards, Circulation 591, pp. 3-12, 1958.

[6] T. Y. Chang and M. J. Hsiao, "Carry-select adder using single ripple-carry adder," Electronics Letters, vol. 34, no. 22, pp. 2101-2103, Oct. 1998.

[7] Y. Kim and L. S. Kim, "64-bit carry-select adder with reduced area," Electronics Letters, vol. 37, no. 10, pp. 614-615, May 2001.

[8] Y. He, C.-H. Chang, and J. Gu, "An area ef• cient 64-bit square root carry-select adder for low power applications," *IEEE ISCAS 2005*, pp. 4082–4085.

[9] E. Mesquita, H. Franck, L. Agostini and J. L. Güntzel. "RIC fast adder and its SET-tolerant implementation in FPGAs," *IEEE FPL 2007*, pp. 638-641.

[10] B. Amelifard, F. Fallah and M. Pedram, "Closing the gap between carry select adder and ripple carry adder: a new class of low-power high-performance adders", ISQED 2005, pp. 148-152.

[11] Synopsys's Design Compiler User Guide, Version C-2009.06, June 2009.