Open Access Library Journal
2020, Volume 7, e6799
ISSN Online: 2333-9721
ISSN Print: 2333-9705

Parallel Self-Timed Adder with Lookahead-Carry Generator

Mohammad Ashfak Habib

Department of Computer Science and Engineering, Chittagong University of Engineering and Technology, Chittagong, Bangladesh
Email: ashfak@cuet.ac.bd

How to cite this paper: Habib, M.A. (2020) Parallel Self-Timed Adder with Lookahead-Carry Generator. *Open Access Library Journal*, **7**: e6799.
https://doi.org/10.4236/oalib.1106799

Received: September 7, 2020
Accepted: September 20, 2020
Published: September 23, 2020

Abstract

Parallel self-timed adder (PASTA) is a newly introduced asynchronous adder. It shows appreciable average-case performance without any special speedup circuitry or look-ahead schema, but its worst-case performance is almost similar to that of ripple carry adder. It is therefore an important research issue to find a technique to improve its worst-case performance without any significant compromise in its other performances. This paper investigates the possibility of such performance improvement of the basic architecture of PASTA by changing its carry propagation schema. The existing ripple fashioned carry propagation schema is replaced by four different lookahead-carry generators. Four different implementations of PASTA with four different types of lookahead-carry generators are presented. The carry propagation delays of the proposed implementations are compared with that of the basic implementation of PASTA. More impressive worst case performances are found for the proposed implementations. The amount of improvement is minimum 45.25% and maximum 61.09%. The proposed designs are regular and do not have any practical limitations of fan-ins or fan-outs. Simulation-based results validate the practicality as well as the superiority of the proposed architecture over the existing architecture of PASTA.

Subject Areas

Computer Architecture, Digital Electronics

Keywords

Arithmetic Circuit, Binary Adder, Asynchronous Circuit, Self-Timed Adder, Parallel Prefix Adder

1. Introduction

Statistical analysis shows that, in a prototypical RISC machine, 72 percent of the instructions perform addition (or subtraction) in the datapath [1] [2]. It is even reported to reach 80 percent in ARM processors [3]. Therefore, binary addition is one of the most common arithmetic operations that a computer processor performs. According to the hardware design principal, *make the common case fast* [4], faster hardware for addition operation can achieve faster processor. Designing a faster binary adder is therefore an interesting research issue.

Researchers are thoroughly investigating the addition operation since the beginning of modern computing [5] and they are still working on it. Recently the architecture and performance of a new adder PASTA have been discussed by Rahman *et al.* [6], which uses recursive process for generating the final result. Though PASTA is an asynchronous adder and uses ripple fashioned carry propagation technique, its performance is compared with various reputed synchronous and asynchronous adders and shown competitive in all respect [6]. The mentionable achievements of PASTA are: simple design (area and interconnection-wise equivalent to ripple carry adder), logarithmic average time performance and highly practical and efficient completion detection unit.

PASTA is a self-timed adder and it has a completion detection mechanism. An adder which can announce the completion of its operation can take the advantage of the shorter average-case propagation delay and, in turn, exhibit average case performance [7]. Some recent studies [8] [9] [10] [11] [12] further investigated the architecture of PASTA but none of those studies changed its carry propagation technique. Though PASTA has appreciable average case performance, improving its worst case performance can make it more acceptable. This paper presents four different enhanced architectures of PASTA with modified carry propagation technique. This modification makes its worst case delay nearly half without adversely degrading its other performances. The proposed implementations are not only appropriate for asynchronous systems but also suitable for Globally Asynchronous Locally Synchronous (GALS) systems [13] because of their improved worst case performance and efficient completion detection mechanism. For the ease of explanation, the term *Enhanced Parallel Self-Timed Adder* (EPASTA) is used to indicate the proposed adder circuits.

2. Methods

Total nine different 16-bit adders are implemented. Five of them are existing adders and the remaining four are the four different varieties of the proposed EPASTA. The carry propagation delays of the proposed adders are compared with the delays of the existing adders.

2.1. Existing Adders

The basic architecture of an *n*-bit PASTA is adopted from [6] and it is illustrated in **Figure 1**. One can intuitively understand, by examining the architecture of

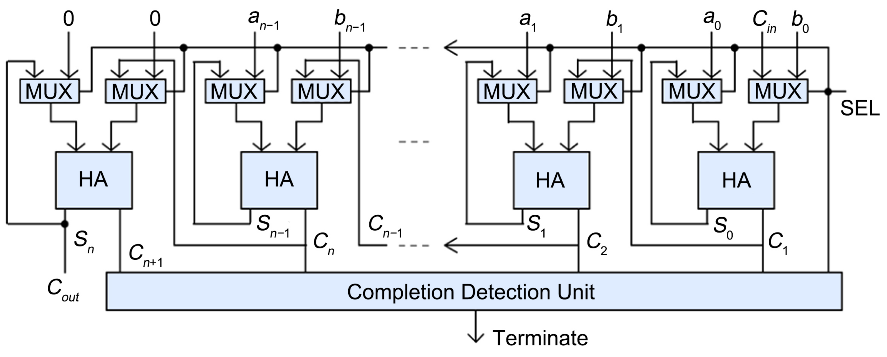

Figure 1. General block diagram of PASTA [6].

PASTA, that the carry propagation mechanism of this adder is similar to that of the basic ripple carry adder. Its operation is divided into two phases namely *Initial Phase* and *Recursive Phase*. The common selector (*SEL*) sends signal for all the 2 × 1 multiplexers. It determines the appropriate phase. In initial phase *SEL* = 0 and in recursive phase *SEL* = 1. In initial phase the multiplexer in the *i*th bit position allows a_i and b_i to go to the corresponding half adder and the half adder produces the initial values of the sum (S_i) and the output carry (C_{i+1}) bits. In the recursive phase, the feedback path allows the initial sum to be added to the input carry C_i recursively. The values of the sum and the output carry bits are recalculated for every recursion cycle. The recursion is terminated when the stopping criterion is met. The completion detection unit produces an asserted *Terminate* signal for indicating the completion of operation.

The lookahead-carry generation techniques of four well known tree-like parallel synchronous adders are used in the EPASTA. Chosen parallel synchronous adders are: Block Carry Lookahead Adder (BCLA), Kogge Stone Adder (KSA), Brent-Kung Adder (BKA) and Sklansky's Conditional Sum Adder (SCSA). These adders are well known and have preferable worst-case performances. The basic structures of these adders were explained in [14] [15] [16] [17]. Four types of sub-circuits are repeatedly used in these adders. For the ease of explanation, these sub-circuits are termed in this paper as modules and are represented by four different symbols. These sub-circuits and their corresponding symbols are shown in **Figure 2**. Here, *i*, *j* and *k* indicate the bit positions, where $i \geq j \geq k$. The X-module is used in all four types of adders. This module of the *i*th bit position computes the following outputs:

$$\text{Carry Propagate, } p_i = a_i \oplus b_i \tag{1}$$

$$\text{Carry Generate, } g_i = a_i \cdot b_i \tag{2}$$

$$\text{Sum, } S_i = a_i \oplus b_i \oplus c_i \tag{3}$$

Here, a_i and b_i are the *i*th bit of the *n*-bit operands *A* and *B* respectively. The symbol, C_i represents input carry of *i*th position or the carry output of the (*i*-1)th position. For all these expressions $i = 0,1,\cdots,n-1$. The BCLA uses the Y-module that computes the carry bits. It also computes the block-carry-propagate and the block-carry-generate signals as follows:

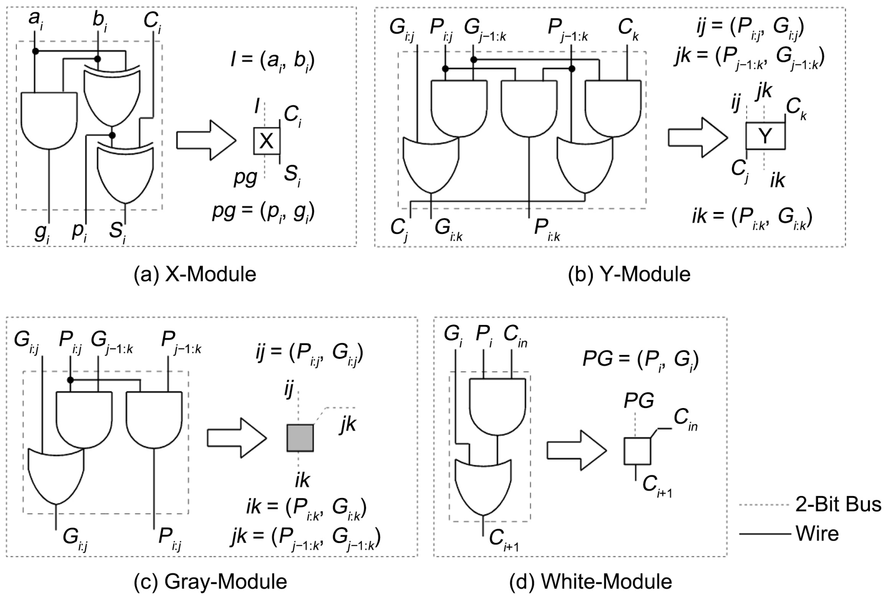

Figure 2. Four repeatedly used sub-circuits and their corresponding symbols.

$$\textit{Block Carry Propagate}, P_{i,k} = P_{i,j} P_{j-1,k} \tag{4}$$

$$\textit{Block Carry Propagate}, G_{i,k} = G_{i,j} + P_{i,j} G_{j-1,k} \tag{5}$$

$$\textit{Carry}, C_j = G_{j-1,k} + P_{j-1,k} C_k \tag{6}$$

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

The other three adders (KSA, BKA and SCSA) use gray-module and white-module in addition with X-module. These adders also operate on the principle of *Block Carry Propagate* and *Block Carry Generate* [14] [15]. The gray-module and the white-module are the subdivisions of the Y-module of BCLA. The gray-module computes the *Block Carry Propagate* and *Block Carry Generate* whereas the white-module computes the carry C_j. The internal circuits of the gray-module and the white-module are shown in Figure 2(c) and Figure 2(d) respectively.

2.2. Enhanced Parallel Self-Timed Adder (EPASTA)

The basic architecture of PASTA has a uniform design. An *n*-bit PASTA is constructed from *n*+1 similar blocks of circuitry. In the rest of this paper the circuitry of each block of PASTA is termed as a *PASTA-block*. Structure of a PASTA-block is shown in Figure 3(a). This PASTA-block is used in the proposed architecture. In order to increase the visibility of the EPASTA architecture, a symbol is defined to represent this PASTA-block as shown in Figure 3(b). The new symbol is termed as EP-module. For the sake of simplicity of the figure, the *SEL* terminal is not shown in the EP-module.

Carry propagation in PASTA is similar to that of ripple carry adder. Therefore the worst case computation time of PASTA is not so satisfactory. Adders having lower worst case computation time, computes carry earlier by using lookahead

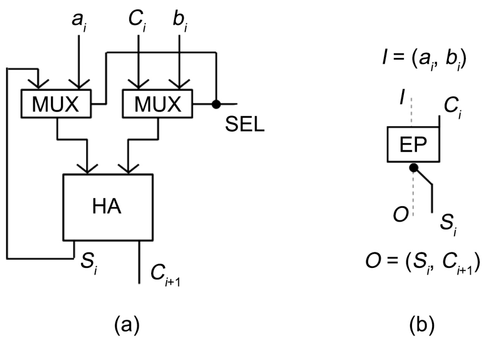

Figure 3. PASTA-block and a symbol to represent it in the EPASTA.

carry generation techniques. Any carry can be computed by using only two levels of basic logic gates [4], but the size of the circuit increases exponentially with the size of the operands. For large operands ($n > 4$) it is impractical and inefficient to use two level of logic for carry computation because of the limitation of fan-in and fan-out, irregular structure, use of many long wires, etc. [1] [18]. Therefore, the practical carry-lookahead schemas usually use tree-like circuits that have simple regular structures [1] [16] [19] [20] [21]. Four different architectures of EPASTA, having four different lookahead-carry generation schemes, are presented below.

The S_i^0 and the C_{i+1}^0 terminals of the EP-module, in the initial phase, can be expressed as $S_i^0 = a_i \oplus b_i$ and $C_{i+1}^0 = a_i \cdot b_i$. These outputs are same as the p_i, g_i outputs of the X-module. In the first cycle of iterative phase, the *Sum* terminal of the EP-module will produce $S_i^1 = S_i^0 \oplus C_i = a_i \oplus b_i \oplus C_i$. This output is exactly same the *Sum* output of the X-module. Therefore, the EP-module is equivalent to the X-module. The EPASTA circuits are implemented by replacing the X-modules of the four selected tree-like parallel adder circuits. The first of the four architectures of 16-bit EPASTA, which is constructed by replacing the X-modules of BCLA with the EP-modules, is shown in **Figure 4**. In order to store the correct value of C_{out} an additional EP-module is attached after the most significant bit position. Moreover for sensing the completion of operation a completion detection unit similar to that of PASTA is attached. This version of EPASTA uses the lookahead-carry generator of BCLA for computing the carry bits.

It has been shown that the X-module and the EP-module are interchangeable. So, the second version of EPASTA, which uses the lookahead-carry generator of KSA, can be constructed easily by replacing the X-modules of KSA with the EP-modules. Similar to the previous version of EPASTA an additional EP-module should be added for storing the correct value of C_{out}. A completion detection unit should be added for sensing the completion of operation. **Figure 5** illustrates the architecture of a16-bit EPASTA with lookahead-carry generator of KSA.

Similar procedure is followed to construct the other two versions of EPASTA. The X-modules of the BKA and SCSA are replaced by the EP-modules. Both the

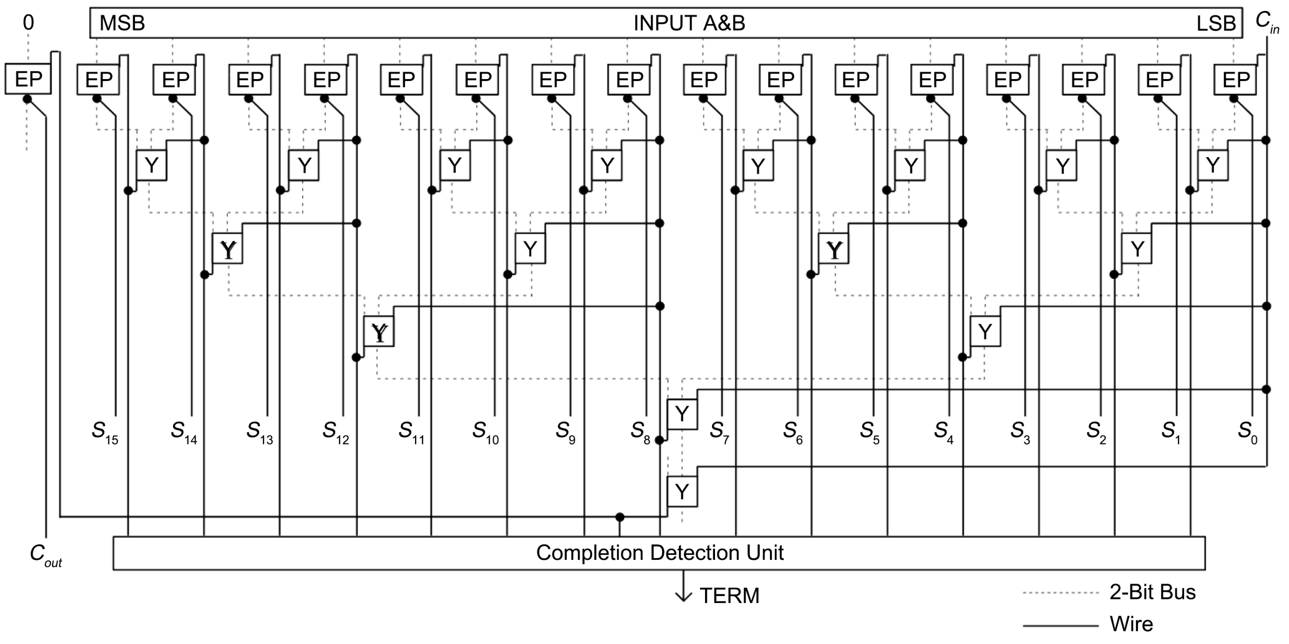
0
MSB
INPUT A&B
LSB
C_{in}
EP
Y
S_{15} S_{14} S_{13} S_{12} S_{11} S_{10} S_9 S_8 S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0
C_{out}
Completion Detection Unit
TERM
2-Bit Bus
Wire

Figure 4. Architecture of a 16-bit EPASTA with lookahead-carry generator of BCLA.

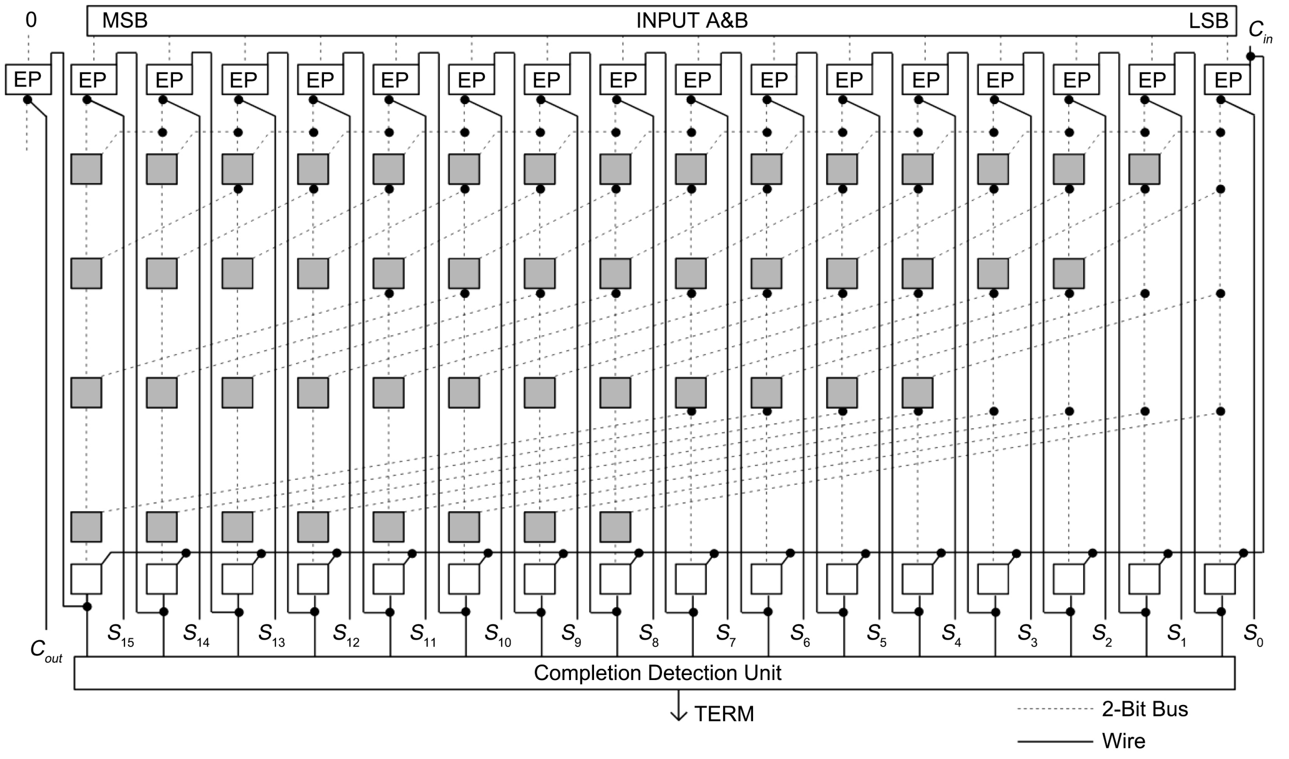
0
MSB
INPUT A&B
LSB
C_{in}
EP
S_{15} S_{14} S_{13} S_{12} S_{11} S_{10} S_9 S_8 S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0
C_{out}
Completion Detection Unit
TERM
2-Bit Bus
Wire

Figure 5. Architecture of a 16-bit EPASTA with lookahead-carry generator of KSA.

implementations require an additional EP-module for capturing the final value of C_{out}. The completion detection unit is also included in both the implementations for realizing the completion of operations. The EPASTA implementations with the lookahead-carry generators of BKA and SCSA are illustrated in Figure 6 and Figure 7 respectively.

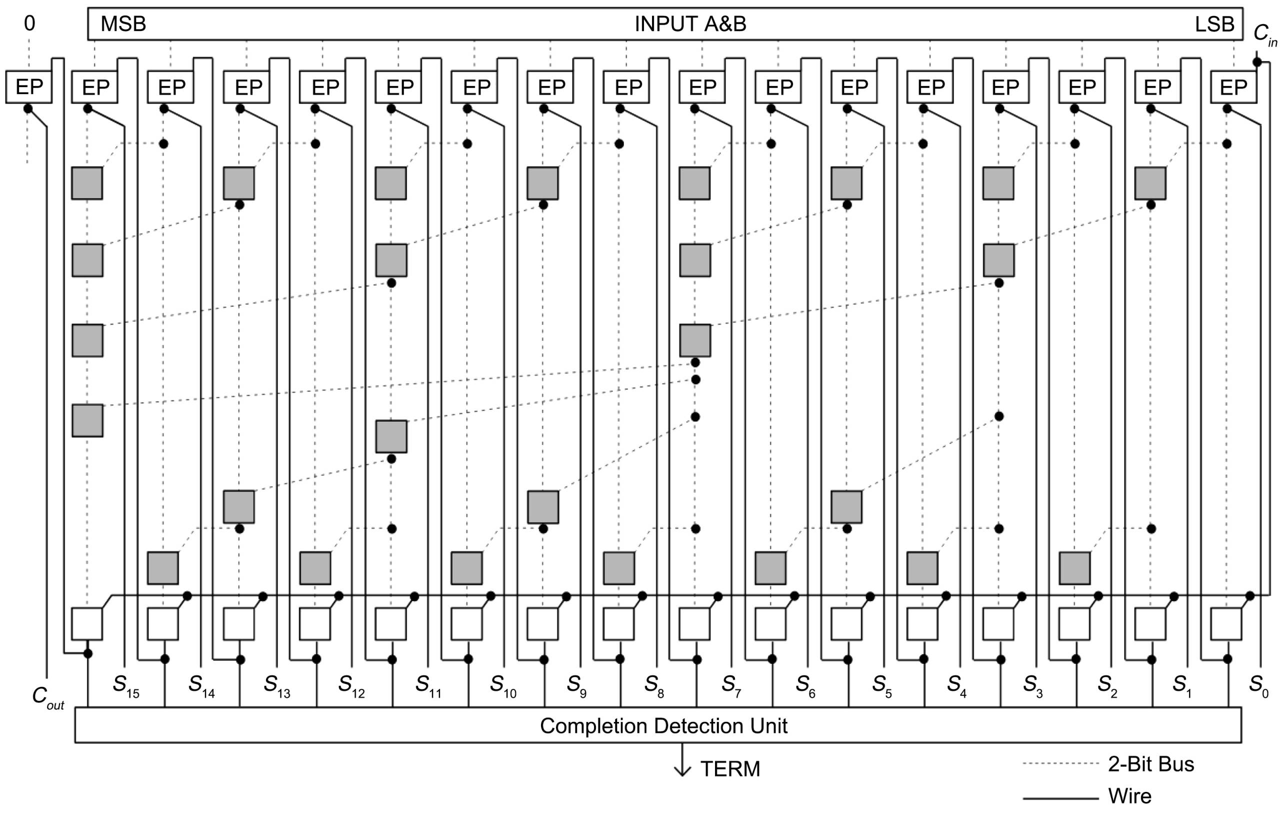

Figure 6. Architecture of a 16-bit EPASTA with lookahead-carry generator of BKA.

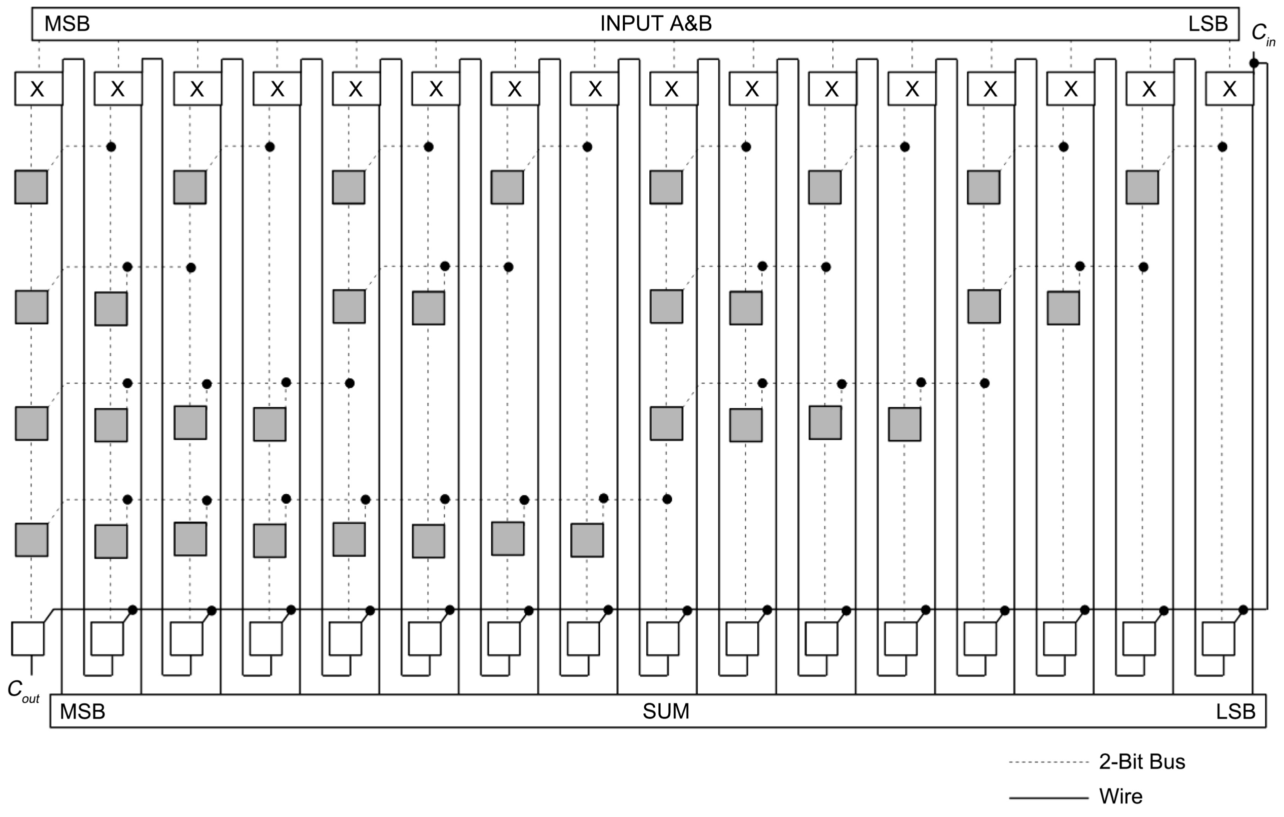

Figure 7. Architecture of a 16-bit EPASTA with lookahead-carry generator of SCSA.

3. Results

In this section the simulation results are presented for all the adders that are discussed in this paper. Though the illustrated adders are 16-bit adders, actually 32-bit versions of those adders are implemented for simulation. All the simulation are done by using an industry standard software tool and executed on 64 bit Linux platform. Simulation is performed for three different TSMC processes.

Three different types of adder operation are analyzed and those are: worst-case, best-case and average case. The best-case addition does not involve any carry propagation and hence incurs only a single bit adder delay for producing the result. The worst-case involves maximum carry propagation cascaded delays due to the propagation length of 32-bits. The average case shows how the separate carry propagation is limited within their individual propagation chain and can progress simultaneously with the other carry propagation chains. Some test-cases, representative of these cases, are chosen. The dataset used for this experiment is summarized in **Table 1**. The expected carry chain length for n-bit binary numbers is established in [22]. A carry chain length indicates the maximum number of consecutive PASTA-blocks that propagate a carry bit (1). While adding two n-bit binary numbers, a carry chain length m indicates that a carry bit will propagate through maximum of m consecutive PASTA-blocks for at least once in the whole n-bit addition operation. Since carry chain length is a factor, five different random numbers are chosen. Among these numbers, three have maximum carry chain length of 5 and two have maximum carry chain length of 6. Thus they represent an average carry chain length of 5.4. The delay is measured at 70% transition point for the related signals.

The delay performances of different adders are shown in **Table 2**. It is divided into three parts to differentiate between conventional synchronous parallel adders, basic architecture of PASTA and four versions of EPASTA. The top part shows the results for conventional adders (*i.e.* BKA, BCLA, KSA and SCSA). Worst-case delay is important for these adders because they do not have any completion sensing mechanism. The adders in the middle and the bottom part

Table 1. Dataset for comparing different adders.

Test Case	Operand A	Operand B	Maximum Carry Chain Length
Worst Case	FFFF FFFF	0000 0001	32
Average Case 1	0501 6A44	FC3F 0499	6
Average Case 2	3F05 0FC0	0130 0041	6
Average Case 3	0902 6A44	F83E 0499	5
Average Case 4	3E05 0F80	0230 0081	5
Average Case 5	0052 40A2	57C5 0F84	5
Bast Case	55E1 9D5C	AA1E 62A3	0

Table 2. Spice timing report for different 32-bit adders.

Process	TSMC 0.35μ (V_{dd} = 3.3V)			TSMC 0.25μ (V_{dd} = 2.5V)			TSMC 0.18μ (V_{dd} = 1.8V)		
	Best (ns)	**Avg.** (ns)	**Worst** (ns)	**Best** (ns)	**Avg.** (ns)	**Worst** (ns)	**Best** (ns)	**Avg.** (ns)	**Worst** (ns)
BKA	0.3090	1.4493	2.8693	0.2251	1.2158	2.4743	0.1342	0.8497	1.7939
BCLA	0.2039	1.3224	2.6980	0.1432	1.1601	2.3761	0.0796	0.8207	1.7342
KSA	0.3245	1.3423	1.7786	0.2383	1.1238	1.5254	0.1468	0.7823	1.0931
SCSA	0.3094	1.4563	2.5772	0.2254	1.2232	2.2399	0.1346	0.8580	1.6168
PASTA	**0.6313**	**2.0387**	9.0872	**0.9593**	**1.9673**	8.1748	**1.8610**	2.8601	7.6258
EPASTA-BKA	1.1919	3.1586	4.4193	1.2188	2.8962	3.9819	2.9017	2.8594	4.1784
EPASTA-BCLA	1.0021	2.8991	4.0468	1.0444	2.6826	3.6860	2.5684	**2.6333**	3.7517
EPASTA-KSA	1.1957	3.2059	**3.6728**	1.2244	2.9455	**3.3933**	2.9138	2.8426	**2.9672**
EPASTA-SCSA	1.1903	3.2221	4.5280	1.2270	2.9578	4.0840	2.8942	2.9153	3.8388

are asynchronous adders which have completion detection mechanism and whose complete operation is divided into two phase (initial phase and recursive phase). So for computing the delay of these adders the following relation is used:

$$t_{total} = t_{initial} + t_{recursive}$$

Here $t_{initial}$ represents the time required for the state transition of the initial phase and $t_{recursive}$ represents the delay between *SEL* and the *Terminate* signals. The value of t_{total} is listed in Table 2. Since the major concern of this study is to analyze the performance of the proposed adders with respect to PASTA, the minimum delays of the asynchronous adders are shown in bold face.

Table 2 shows that all of the four proposed architectures of EPASTA give better worst-case delay compared to that of PASTA and the amount of improvement is minimum 3.45 nS and maximum 4.66 nS (*i.e.* minimum 45.25% and maximum 61.09%). Though, with respect to PASTA, the best-case and average case delays of EPASTA increase most of the time, the amount of this increment is not so high (maximum 1.18 nS).

Among the four proposed architectures, EPASTA with lookahead-carry generator of KSA gives best result for the worst case. For the other two cases (best case and average case), EPASTA with lookahead-carry generator of CLA performs better than the other architectures of EPASTA.

4. Conclusion

The major objective of this paper was to analyze the practicality of using the lookahead-carry generator with the newly introduced self-timed adder PASTA. Moreover, this study also investigates a probable solution to improve the worst case performance of PASTA. The results show that the proposed architecture gives better worst case performance than the basic architecture of PASTA without major compromise of the best and average case performances. Moreover, these results also support the practicality of the proposed architecture of self-timed adders.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Hennessy, J.L. and Patterson, D.A. (1990) Computer Architecture: A Quantitative Approach. Morgan Kaufmann, Waltham.

[2] Franklin, M.A. and Pan, T. (1994) Performance Comparison of Asynchronous Adders. *Proceedings of IEEE Symposium on Advanced Research in Asynchronous Circuits and Systems*, Salt Lake City, 3-5 November 1994, 117-125. https://doi.org/10.1109/ASYNC.1994.656299

[3] Garside, J.D. (1993) A CMOS VLSI Implementation of an Asynchronous ALU. In: Furber, S. and Edwards, M., Eds., *Asynchronous Design Methodologies*, IFIP Transactions, North Holland, 181-192.

[4] Patterson, D.A. and Hennessy, J.L. (2014) Computer Organization and Design: The Hardware/Software Interface. 5th Edition, Morgan Kaufmann, Waltham.

[5] Zimmermann, R. (1997) Binary Adder Architectures for Cell-based VLSI and Their Synthesis. Ph.D. Dissertation, Swiss Federal Institute of Technology, Zurich.

[6] Rahman, M.Z., Kleeman, L. and Habib, M.A. (2014) Recursive Approach to the Design of a Parallel Self-Timed Adder. *IEEE Transactions on Very Large Scale Integration* (*VLSI*) *Systems*, **23**, 213-217. https://doi.org/10.1109/TVLSI.2014.2303809

[7] Johnson, D. and Akella, V. (1998) Design and Analysis of Asynchronous Adders. IEE Proceedings—Computers and Digital Techniques, **145**, 1-8. https://doi.org/10.1049/ip-cdt:19981770

[8] Jayanthi, A.N. (2019) Performance Improvement in VLSI Adders. *International Journal of Research in Arts and Science*, **5**, 76-87. https://doi.org/10.9756/BP2019.1002/07

[9] Sivakumar, M. and Omkumar, S. (2018) Design and FPGA Implementation of FBMC Transmitter by Using Clock Gating Technique based QAM, In verse FFT and Filter Bank for Low Power and High Speed Applications. *Journal of Electrical Engineering & Technology*, **13**, 2479-2484.

[10] Jhamb, M. (2017) Efficient Adders for Assistive Devices. *Engineering Science and Technology*, **20**, 95-104. https://doi.org/10.1016/j.jestch.2016.09.007

[11] Vigneshwari, R., Jayasimha, T. and Sasikumar, P. (2017) Power Analysis by Combining the Modules PASTA Using DGMOSFET. *Advances in Natural and Applied Sciences*, **11**, 691-698.

[12] Sivakumar, M. and Omkumar, S. (2016) Integration of Optimized GDI Logic Based NOR Gate and Half Adder into PASTA for Low Power & Low Area Applications. *International Journal of Applied Engineering Research*, **11**, 2629-2633.

[13] Chapiro, D.M. (1985) Globally-Asynchronous Locally-Synchronous Systems. Ph.D. Dissertation, Stanford University, California.

[14] Kogge, P.M. and Stone, H.S. (1973) A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations. *IEEE Transactions on Computers*, **100**,

786-793. https://doi.org/10.1109/TC.1973.5009159

[15] Rabaey, J.M., Chandrakasan, A.P. and Nikolić, B. (2003) Digital Integrated Circuits: A Design Perspective. Second Edition, Prentice Hall, New Jersey.

[16] Brent, R.P. and Kung, H.T. (1982) A Regular Layout for Parallel Adders. *IEEE Transactions on Computers*, **C-31**, 260-264. https://doi.org/10.1109/TC.1982.1675982

[17] Sklansky, J. (1960) Conditional-Sum Addition Logic. *IRE Transactions on Electronic Computers*, **EC-9**, 226-231. https://doi.org/10.1109/TEC.1960.5219822

[18] Ngai, T.F., Irwin, M.J. and Rawat, S. (1986) Regular Area-Time Efficient Carry-Lookahead Adders. *Journal of Parallel and Distributed Computing*, **3**, 92-105. https://doi.org/10.1016/0743-7315(86)90029-8

[19] Flores, I. (1963) The Logic of Computer Arithmetic. Prentice Hall, New Jersey.

[20] Unger, S.H. (1977) Tree Realizations of Iterative Circuits. *IEEE Transactions on Computers*, **C-26**, 365-383. https://doi.org/10.1109/TC.1977.1674846

[21] Cheng, F.C., Unger, S.H. and Theobald, M. (2000) Self-Timed Carry-Lookahead Adders. *IEEE Transactions on Computers*, **49**, 659-672. https://doi.org/10.1109/12.863035

[22] Reitwiesner, G.W. (1960) The Determination of Carry Propagation Length for Binary Addition. *IRE Transactions on Electronic Computers*, **EC-9**, 35-38. https://doi.org/10.1109/TEC.1960.5221602