International Journal For Technological Research In Engineering
Volume 4, Issue 4, December-2016 *ISSN (Online): 2347 - 4718*

DESIGN OF AN EFFICIENT PARALLEL SELF-TIMED ADDER USING MULTIPLIER

Biruda Prasanthi[1], Jhansi Rani Kaka[2]
[1]P.G.Scholar, VLSI, [2]Assistant Professor
Electronics and communication engineering department, University College of engineering Kakinada, Kakinada

Abstract: ***In contemporary world there is a great need for low power consumption circuit design, which is very compatible with emphasis on high performance in DIP (Digital Image Processing) system. In this paper the proposed method presents a parallel single-rail self-timed adder which uses recursive method for performing multi bit binary addition and multipication. This design obtains good performance without any special speedup circuitry. A practical enactment is provided along with a completion detection unit. The implementation is extensive and does not have any practical limitations of high fan-outs. A high fan-in gate is essential even though this is unavoidable for asynchronous logic and is managed by connecting the transistors in parallel. Integer addition is one of the most significant operations in digital computer systems because the performance of processors is outstandingly influenced by the speed of their adders. PASTA design uses multiplexers along with half adders. The absence of clock generation and distribution units also results in less power dissipation. Clock less chips offer power efficiency, robustness and reliability. A multiplier is designed and synthesized using existing adder and proposed adder to show the superiority of the proposed approach.***

Keywords: Asynchronous circuits, Array Multiplier, CSA, PASTA

I. INTRODUCTION

Integer addition is one of the most essential operations in digital computer systems. In addition to explicit arithmetic (such as addition, subtraction, multiplication, and division) performed in a program, additions are performed to increment program counters and calculate effective addresses. Adders are considered to be the heart of computational circuits and addition has been the core for many complex arithmetic circuits. Most of the adders have been designed for asynchronous circuits as asynchronous circuits do not assume any quantization of time and some still use synchronous circuits as well. Thus, having free from various problems of clocked (synchronous) circuits. Here we have generated the design of binary adders and concentrated on asynchronous self-timed adders. Self timed relate the logic circuits that depend on timing assumptions for the accurate operation of the circuit. Self-timed adders have the capacity to run faster averaged for dynamic data, that early completion sensing can avoid delay mechanism of synchronous circuits. Adaptive signal processing has developed into a self-contained field that finds wide range of real-life applications such as adaptive equalization, noise and echo cancellation, linear predictive coding, and adaptive beam-forming. Adaptive signal processing algorithms are characterized by their recursive operations for realizing algorithmic self-designing/adaptation. To realize high-throughput VLSI implementation of adaptive signal processing algorithms, architecture-level technique pipelining is typically used. Pipelined adaptive signal processing systems are essentially subject to a trade-off between system throughput and signal processing performance, i.e., deeper pipelined adaptation feedback loop can realize higher throughput, but the delayed feedback will incur larger performance degradation. It should be pointed out that, for other recursive algorithms such as infinite impulse response (IIR) filtering and Viterbi algorithm, direct pipelining may simply ruin their functionality and appropriate algorithm-level modification is required for the use of pipelining. A pipelined adaptive signal processing algorithm implemented using the conventional synchronous pipeline typically has a fixed pipeline depth that is determined in the design phase to accommodate the highest run-time throughput requirement. Although it is possible to on-the-fly configure the pipeline depth of synchronous pipeline by selectively bypassing certain levels of registers, this is very inflexible and cannot realize fine-grain graceful configuration on the throughput/performance trade-offs. For example, consider an 8-stage pipelined recursive adaptation loop in which the registers are almost evenly placed along the loop for maximizing the throughput. If we bypass one level of registers to realize a 7-stage pipeline, the delay of the critical path may double and the throughput will reduce almost by half. Self-timed pipeline works in a different way from its synchronous counterpart. Without a common and discrete notion of time, self-timed pipeline relies on the handshake between components to perform the synchronization and communication. Each distinct data propagating through a self timed pipeline is conventionally called a token. The pipeline depth of a self-timed pipeline simply equals the number of tokens present in the pipeline at the same time. Hence, we can dynamically configure the pipeline depth by controlling the number of tokens present in the pipeline. This property of self-timed pipeline has been exploited in the design of a mixed synchronous-asynchronous FIR filter that can support variable latency (in terms of clock cycles) and power management of an embedded, single-issue processor. In pipelined adaptive signal processing systems, the pipeline depth of the adaptation feedback loops is the key to tune the inherent trade-off between throughput and signal processing performance. This directly motivates us to apply self-timed

pipeline for the implementation of adaptive signal processing systems to realize gracefully configurable throughput/performance trade-off. This can be leveraged to improve the overall system performance in many circumstances. For example, for adaptive signal processing systems with variable data rate, we can dynamically adjust the pipeline depth to the minimum allowable value according to the current data rate to realize the best signal processing performance. These Self-timed adders have the capacity to run faster than the conventional circuit design. This paper presents an asynchronous parallel self-timed adder (PASTA). This design uses half-adders (HAs) with the multiplexers requiring minimum interconnections. Thus, making way for many VLSI implementations. For independent carry chain blocks PASTA works in a parallel manner. There are four levels of minimization of power dissipation in CMOS based system designs: technology, circuit, architecture and algorithm. Generally choosing appropriate circuit design style 20% to 30 % power can be saved in circuit level.

II. BACKGROUND

Addition is the most common and often used arithmetic operation on microprocessor, digital signal processor, especially digital computers. Also, it serves as a building block for synthesis all other arithmetic operations. Therefore, regarding the efficient implementation of an arithmetic unit, the binary adder structures become a very critical hardware unit. A carry-save adder is a type of digital adder, used in computer micro architecture to compute the sum of three or more n bit numbers in binary. It differs from other digital adders in that it outputs two numbers of the same dimensions as the inputs, one which is a sequence of partial sum bits and another which is a sequence of carry bits.

Consider the sum: 12345678+87654322=100000000. Using basic arithmetic, calculate right to left, "8+2=0, carry 1", "7+2+1=0, carry 1", "6+3+1=0, carry 1", and so on to the end of the sum. Although, the last digit of the result at once, we cannot know the first digit until every digit in the calculation, passing the carry from each digit to the one on its left. Thus adding two n-digit numbers has to take a time proportional to n, even if the machinery is capable of performing many calculations simultaneously. The carry-save unit consists of n full adders, each of which computes a single sum and carry bit based solely on the corresponding bits of the three input numbers. Given the three n bit numbers a, b, and c, it produces a partial sum ps_i and a shift-carry sc_i:

$$Psi = ai \oplus bi \oplus ci$$

$$Sci = (ai \wedge bi) \vee (ai \wedge ci) \vee (bi \wedge ci)$$

The entire sum can then be computed by:

- Shifting the carry sequence sc left by one place.
- Appending a 0 to the front (most significant bit) of the partial sum sequence ps.
- Using a ripple carry adder to add these two together and produce the resulting n + 1-bit value.

Array Multiplier Array number is renowned thanks to its regular structure. Number circuit relies on add and shift rule. Every partial product is generated by the multiplication of the number with one number bit. The partial product is shifted per their bit orders then additional. The addition will be performed with traditional carry save adder. First advantage of the array multiplier is that it has a regular structure. Since it is regular, it is easy to layout and has a small size. . A second advantage of the array multiplier is its ease of design for a pipelined architecture.

III. DESIGN OF PASTA

In this segment, the design and theory behind nourishment is presented. The adder initial accepts 2 input operands to perform [Fig.1.] additions for every bit. Subsequently, it iterates exploitation earlier generated carry and sums to perform half-additions repeatedly till all carry bits are consumed and settled at zero level. The general design of the adder is shown in Fig. 1. The choice input for two-input multiplexers corresponds to the Request handclasp signal and can be one zero to one transition denoted by SEL. can it'll at the start choose the particular operands throughout SEL=0and will switch to feedback/carry methods for consequent iterations exploitation SEL=1. The feedback path from the HAs allows the multiple iterations to continue till the completion once all carry signals can assume zero values.

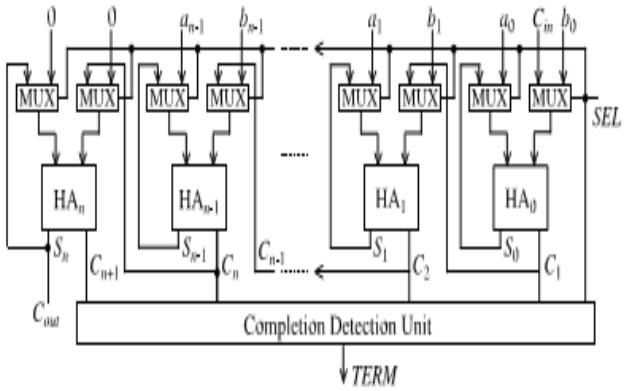


Fig .1. General block diagram of PASTA

3.1 State Diagrams

In Fig. 3, 2 state diagrams are drawn for the initial section and therefore the repetitious section of the projected design. Every state is delineate by (Ci+1Si) pairwhereCi+1, Si represent do and add values, severally, from the i^{th} bit adder block. Throughout the initial section, the circuit just works as a combinatory hour angle operational in elementary mode. It's evident that because of the utilization of HAs rather than FAs, state (11) cannot seem. During the repetitious section (SEL=1), the feedback path through electronic device block is activated. The carry transitions (Ci) are allowed as again and again PRN to complete the formula. From the definition of elementary mode circuits, the current style cannot be thought-about as a elementary mode circuit because the input–outputs can undergo many transitions before manufacturing the ultimate output. It's not a Muller circuit operating outside the basic mode either as internally; many transitions can occur, as shown within the state diagram. this can be analogous to cyclic consecutive circuits wherever gate delays are utilised to separate individual states.

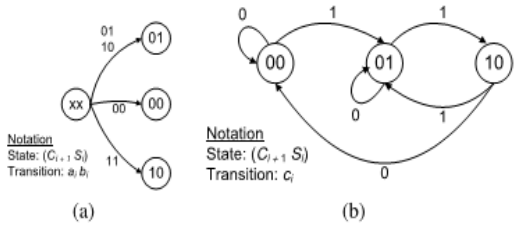


Fig. 2. State diagrams for PASTA. (b) Iterative phase

3.2 Recursive Formula for Binary Addition

Let s_i^j and C_{i+1}^j denote the sum and carry, respectively, for i^{th} bit at the j^{th} iteration. The initial condition (j =0) for addition is formulated as follows

$S_i^0 = a_i \oplus b_i$

$C^0_{i+1} = a_i b_i$ [1]

The j^{th} iteration for the recursive addition is formulated by

$S_i^j = Sij^{-1} \oplus C_i^{j-1}$,$0 \le i < n$ [2]

$C^j_{i+1} = S_i^{j-1} C_i^{j-1}$, $0 \le i \le n$. [3]

The recursion is terminated at k^{th} iteration when the following condition is met

$C_n^k + C_{n-1}^k + \ldots\ldots\ldots\ldots + C_1^k = 0$, $0 \le k \le n$. [4]

Now, the correctness of the algorithmic formulation is inductively evidenced as follows.

Theorem 1: The algorithmic formulation of (1)–(4) can turn out correct total for any range of bits and can terminate inside a finite time.

Proof: A tendency to prove the correctness of the algorithmic rule by induction on the specified range of iterations for finishing the addition (meeting the terminating condition).

Basis: contemplate the quantity decisions that no carry propagation is needed, i.e., C_i^0 = zero for $\forall i, i \in [0..n]$. The projected formulation can turn out the right result by a single-bit computation time and terminate instantly as (4) is met.

Induction: Assume that $C^k_{i+1} \neq 0$ for some i^{th} bit at k^{th} iteration. Let l be such somewhat that $C^k_{l+1} = 1$. We have a tendency to show that it'll be with success transmitted to next higher bit within the (k+1) th iteration. As shown within the state diagram, the k^{th} iteration of l^{th} bit state (C^k_{l+1}, S_l^k) and (l +1)th bit state C^k_{l+2}, S^k_{l+1}) can be in any of (0,0), (0,1),or(1,0) states. As $C^k_{l+1} = 1$, it implies that $S_l^k = 0$. Hence, from (3), $C_{l+1}^{k+1} = 0$ for any input condition between 0to l bits.

Currently contemplate the $(l+1)^{th}$ bit state (C^k_{l+2}, S^k_{l+1}) for kth iteration. It may even be in any of (0, 0), (0, 1), or (1,0) states. In(k+1)th iteration, the(0,0)and(1,0)states from the k^{th} iteration can properly turn out output of(0,1) following (2) and (3). For (0, 1) state, the carry with success propagates through this bit level following (3). Thus, all the single-bit adders can with success kill or propagate the carries till all carries square measure zero fulfilling the terminating condition. The mathematical kind bestowed higher than is valid below the condition that the iterations progress synchronously for all bit levels and also the needed input and outputs for a selected iteration will be in temporal relation with the progress of 1 iteration. Within the next section, we have a tendency to gift associate degree implementation of the projected design that is later on verified

IV. DESIGN OF CSA MULTIPLIER

Generally multiplication consists of 3 steps: generation of partial merchandise or PPs (PPG), reduction of partial merchandise (PPR), and final carry-propagate addition (CPA). Totally different multiplication algorithms vary within the approaches of PPG, PPR, and CA.

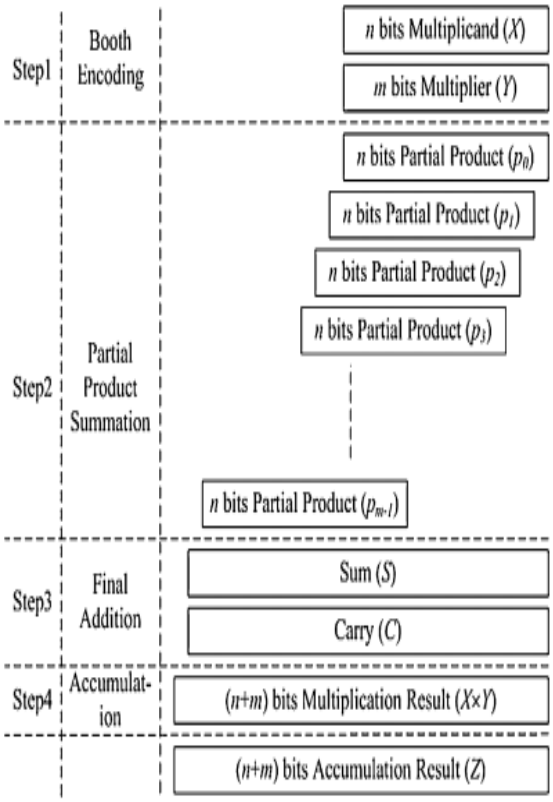


Fig 3.Basic arithmetic steps of multiplication and accumulation.

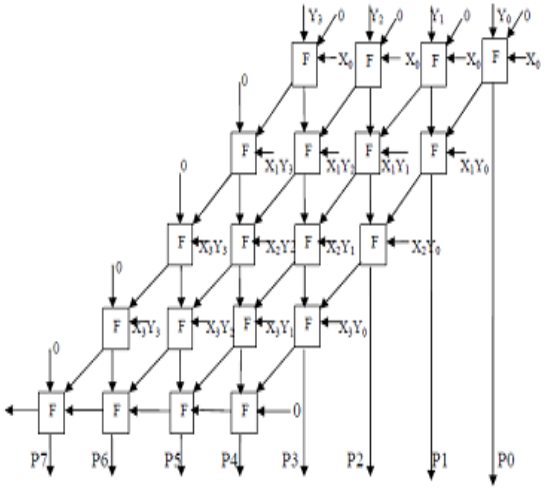


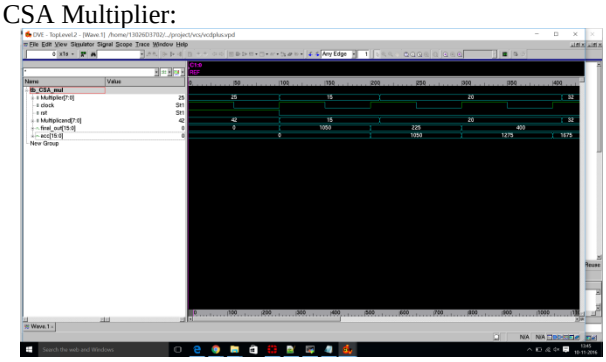
Fig.4.Multiplier with Carry saves Adder Architecture

In the Carry Save Addition methodology, the primary row is either Half-Adders or Full-Adders. If the primary row of the partial product is enforced with Full-Adders, C_{in} are thought of "0". Then the carries of every Full- Adder is diagonally forwarded to consequent row of the adder. The ensuing

multiplier factor is claimed to be Carry Save multiplier factor, as a result of the carry bits aren't straightaway another, however rather are saved for consequent stage. Within the style if the complete adders have 2 input file the third input is taken into account as zero. Within the finish, carries and sums are integrated in an exceedingly quick carry-propagate (e.g. ripple carry or carry look ahead) adder stage

V. SIMULATION RESULTS

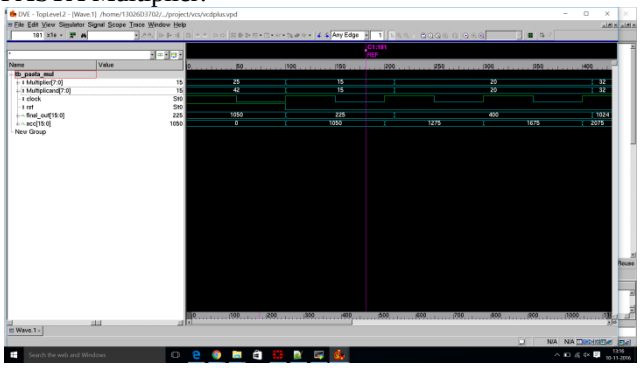
CSA Multiplier:



Design Summary

Number of ports	50
Number of nets	168
Number of cells	68
Number of combinational cells	49
Number of sequential cells	16
Number of macros/black boxes	0
Number of buf/inv:	49

PASTA Multiplier:



Design Summary:

Number of ports	50
Number of nets	331
Number of cells	59
Number of combinational cells	34
Number of sequential cells	16
Number of macros/black boxes	0
Number of buf/inv	34

Timing Summary:

CSA	PASTA
Data arrival time=13.88	Data arrival time= 10.33
Data required time= 9.93	Data required time= 5.0

VI. CONCLUSION

Nowadays speed of the multiplier has turned into an advantage or requirement because of the significance of multiplier circuit in a wide variety of microelectronic systems. In this paper analyzation of different multiplier techniques taking area, timing as the main criteria. Carry save adder is proved to be more efficient in terms of speed compared to conventional multiplication techniques. However the carry save adder design generated the number of cells is 68 and modified parallel self-timed adder using multiplier in around 59 cells. Hence the area is reduced compared with carry save adder; on the other hand it consumes less hardware than other multiplication techniques.

REFERENCES

[1]. Mohammed Ziaur Rahman, Lindsay Kleeman, and Mohammad Ashfak Habib, “Recursive Approach to the Design of a Parallel Self-Timed Adder

[2]. D. Geer, “Is it time for clockless chips? [Asynchronous processor chips],” IEEE Comput., vol. 38, no. 3, pp. 18–19, Mar. 2005.

[3]. J. Sparsø and S. Furber, Principles of Asynchronous Circuit Design. Boston, MA, USA: Kluwer Academic, 2001.

[4]. P. Choudhury, S. Sahoo, and M. Chakraborty, “Implementation of basic arithmetic operations using cellular automaton,” in Proc. ICIT, 2008, pp. 79–80.

[5]. M. Z. Rahman and L. Kleeman, “A delay matched approach for the design of asynchronous sequential circuits,” Dept. Comput. Syst. Technol., Univ. Malaya, Kuala Lumpur, Malaysia, Tech. Rep. 05042013, 2013.

[6]. M. D. Riedel, “Cyclic combinational circuits,” Ph.D. dissertation, Dept. Comput. Sci., California Inst. Technol., Pasadena, CA, USA, May 2004.

[7]. R. F. Tinder, Asynchronous Sequential Machine Design and Analysis: A Comprehensive Development of the Design and Analysis of Clock-Independent State Machines and Systems. San Mateo, CA, USA: Morgan, 2009.

[8]. W. Liu, C. T. Gray, D. Fan, and W. J. Farlow, “A 250-MHz wave pipelined adder in 2-µm CMOS,”IEEE J. Solid-State Circuits, vol. 29, no. 9, pp. 1117–1128, Sep. 1994.

[9]. F.-C. Cheng, S. H. Unger, and M. Theobald, “Self-timed carry-lookahead adders,” IEEE Trans. Comput., vol. 49, no. 7, pp. 659–672, Jul. 2000.

[10]. S. Nowick, “Design of a low-latency asynchronous adder using speculative completion,” IEE Proc. Comput. Digital Tech., vol. 143, no. 5, pp. 301–307, Sep. 1996.

[11]. N. Weste and D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective. Reading, MA, USA: Addison-Wesley, 2005

[12]. C. Cornelius, S. Koppe, and D. Timmermann, “Dynamic circuit techniques in deep submicron technologies: Domino logic reconsidered,” in Proc. IEEE ICICDT, Feb. 2006, pp. 1–4

[13]. M. Anis, S. Member, M. Allam, and M. Elmasry, "Impact of technology scaling on CMOS logic styles," IEEE Trans. Circuits Syst., Analog Digital Signal Process., vol. 49, no. 8, pp. 577–588, Aug. 2002.

