INTERNATIONAL JOURNAL OF RESEARCH
In Advanced Engineering Technologies

Volume 6, Issue 1 JAN 2017

RECURSIVE APPROACH TO THE DESIGN OF A PARALLEL SELF-TIMED ADDER

N.Rakesh Kumar [1] **M. Govind Raj [2]** **P. Suresh Kumar[3]**

nemalirakesh100@gmail.com[1] mgovindraj415@gmail.com[2]

[1]PG Scholar, Dept of ECE, Avanthi Institute of engineering and technology, Gunthapally, Hayath Nagar, Ranga Reddy ,Telangana,

[2]Assistant Professor, Dept of ECE, Avanthi Institute of engineering and technology, Gunthapally, Hayath Nagar, Ranga Reddy ,Telangana,

[3] Associate Professor, HOD, Dept of ECE, Avanthi Institute of engineering and technology, Gunthapally, Hayath Nagar, Ranga Reddy ,Telangana

Abstract:- As the scale of integration keeps growing, more and more sophisticated signal processing systems are being implemented on a VLSI chip. These signal processing applications not only demand great computation capacity but also consume considerable amounts of energy. While performance and area remain to be two major design goals, power consumption has become a critical concern in today's VLSI system design. Multiplication is a fundamental operation in most arithmetic computing systems. Multipliers have large area, long latency and consume considerable power. Multiplication is a basic arithmetic operation which is present in any part of the digital computer especially in signal processing systems. Different techniques are used for multiplication. Some of the techniques are CSA, CSD, Booth's, Grid, Lattice, Combinational, Sequential, Array, Vedic, Wallace-tree etc

Keywords: Multiplier, VHDL, FPGA.

I. INTRODUCTION

Over the last two decades, 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 systems 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 [4], [5] 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 selftimed 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) [6] and power management of an embedded, single-issue processor [7]. In pipelined adaptive signal processing systems, the pipeline depth of the adaptation feedback loops is the key to tune the inherent tradeoff 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 tradeoff. 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. Although the basic idea of the above design approach is simple and intuitive, how to implement it in the real systems involves the following three critical design issues:

1) What type of self-timed pipeline structure should be used? Clearly, to justify the practicality of this design approach, the employed self-timed pipeline must be able to support the same (or comparable) throughput as its synchronous counterpart when they have the same pipeline depth. This means that the recursive self-timed pipeline datapath should have the same (or comparable) propagation delay as its synchronous counterpart. This is a very strict requirement since most self-timed pipeline design schemes involve extra delay overhead for realizing self-timed handshake and have the longer latency than their synchronous counterparts, although they can support very fine-grain pipeline to realize high throughput. In this work, we propose to use the well-known Ted William's high-speed self-timed pipeline [4], [8] because of its zerodelay-overhead feature (i.e., no extra handshake delay is incurred when data propagate through the pipeline). Hence the zero-delay-overhead pipeline can achieve the same latency performance as its synchronous counterpart.

2) How to realize the self-timed data flow synchronization in the recursive adaptation loop? In self-timed data path, synchronization of parallel computational threads relies on forks and joins, where fork refers to a stage with one input channel and multiple output channels and join refers to a stage with multiple input channels and a single output channel. The recursive adaptation loop of adaptive signal processing algorithms contains many forks and joins. However, like many other self-timed pipeline styles, the zero-delay-overhead self-timed pipeline was initially proposed for linear datapath (i.e., without forks and joins). Therefore, it must be appropriately modified to support forks and joins.

3) How to realize run-time addition/removal of tokens in order to change the pipeline depth? In a feed forward only datapath, the pipeline depth can be readily changed by adjusting the input data rate. However, as we will show later, it is not trivial to change the pipeline depth in recursive adaptation loops. We have to design some special circuit elements that can be placed on the recursive adaptation loop to realize run-time addition/removal of tokens.

II. Literature Survey

Power is a problem primarily when cooling is a concern. The maximum power at any time, peak power, is often used for power and ground wiring design, signal noise margin and reliability analysis. Energy per operation or task is a better metric of the energy efficiency of a system, especially in the domain of maximizing battery lifetime. In digital CMOS design, the well-known power-delay product is commonly used to assess the merits of designs. Generally multiplication consists of three steps: generation of partial products or PPs (PPG), reduction of partial products (PPR), and final carry-propagate addition (CPA). Different multiplication

algorithms vary in the approaches of PPG, PPR, and CA. For PPG, radix-2 digit-vector multiplication is the simplest form because the digit-vector multiplication is produced by a set of AND gates. To reduce the number of PPs and consequently reduce the area/delay of PP reduction, one operand is usually recoded into high -radix digit sets. The most popular one is the radix-4 digit set {−2, −1, 0, 1, 2}. For PPR, two alternatives exist: reduction by rows, performed by an array of adders, and reduction by columns, performed by an array of counters. In reduction by rows, there are two extreme classes: linear array and tree array. Linear array has the delay of O (n) while both tree array and column reduction have the delay of O (log n), where n is the number of PPs. The final CPA requires a fast adder scheme because it is on the critical path. Some low-level techniques that has been studied for multipliers include using voltage scaling, layout optimization, transistor reordering and sizing, using pass-transistor logic and swing limited logic, signal polarity optimization, delay balancing and input synchronization. However, these techniques have only achieved moderate improvement on power consumption in multipliers with much design effort or considerable area/delay overhead. The difficulty of low-power multiplier design lies in three aspects. Multiplication is basically a shift add operation. There are, however, many variations on how to do it. Some are more suitable for FPGA use than others; some of them may be efficient for a system like CPU. This section explores various verities and attracting features of multiplication hardware. The multiplier area is quadratically related to the operand precision. Second, parallel multipliers have many logic levels that introduce spurious transitions or glitches. Third, the structure of parallel multipliers could be very complex in order to achieve high speed, which deteriorates the efficiency of layout and circuit level optimization. As a fundamental arithmetic operation, multiplication has many algorithm-level and bit-level computation features in which it differs from random logic. These features have not been considered well in low-level power optimization. It is also difficult to consider input data characteristics at low levels. Therefore, it is desirable to develop algorithm and architecture level power optimization techniques.

Array Multiplier

Array multiplier is well known due to its regular structure. Multiplier circuit is based on add and shift algorithm. Each partial product is generated by the multiplication of the multiplicand with one multiplier bit. The partial product are shifted according to their bit orders and then added. The addition can be performed with normal carry save adder.

Advantages

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 section, the architecture and theory behind PASTA is presented. The adder first accepts two input operands to perform half additions for each bit. Subsequently, it iterates using earlier generated carry and sums to perform half-additions repeatedly until all carry bits are consumed and settled at zero level.

A. Architecture of PASTA

The general architecture of the adder is shown in Fig. 1. The selection input for two-input multiplexers corresponds to the Req handshake signal and will be a single 0 to 1 transition denoted by SEL. It will initially select the actual operands during SEL=0and will switch to feedback/carry paths for subsequent iterations using SEL=1. The feedback path from the HAs enables the multiple iterations to continue until the completion when all carry signals will assume zero values

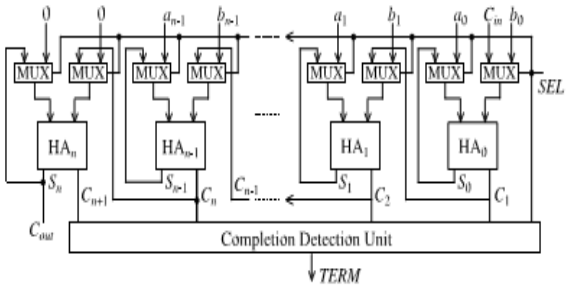

Fig. 1. General block diagram of PASTA

B. State Diagrams

In Fig. 2, two state diagrams are drawn for the initial phase and the iterative phase of the proposed architecture. Each state is represented by (Ci+1Si) pairwhereCi+1, Si represent carry out and sum values, respectively, from the ith bit adder block. During the initial phase, the circuit merely works as a combinational HA operating in fundamental mode. It is apparent that due to the use of HAs instead of FAs, state (11) cannot appear.

During the iterative phase (SEL=1), the feedback path through multiplexer block is activated. The carry transitions (Ci) are allowed as many times as needed to complete the recursion. From the definition of fundamental mode circuits, the present design cannot be considered as a fundamental mode circuit as the input–outputs will go through several transitions before producing the final output. It is not a Muller circuit working outside the fundamental mode either as internally; several transitions will take place, as shown in the state diagram. This is analogous to cyclic sequential circuits where gate delays are utilized to separate individual states.

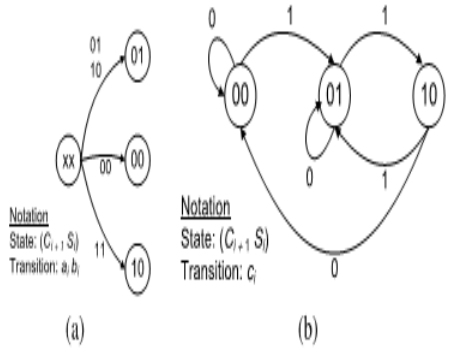

Fig. 2. State diagrams for PASTA. (a) Initial phase. (b) Iterative phase

C. Recursive Formula for Binary Addition

Let S ji andC j i+1 denote the sum and carry, respectively, for ith bit at the jth iteration. The initial condition (j =0) for addition is formulated as follows

$$S_i^0 = a_i \oplus b_i$$
$$C_{i+1}^0 = a_i b_i. \quad (1)$$

Thejth iteration for the recursive addition is formulated by

$$S_i^j = S_i^{j-1} \oplus C_i^{j-1}, \quad 0 \le i < n \quad (2)$$
$$C_{i+1}^j = S_i^{j-1} C_i^{j-1}, \quad 0 \le i \le n. \quad (3)$$

The recursion is terminated atkth iteration when the following condition is met:

$$C_n^k + C_{n-1}^k + \cdots + C_1^k = 0, \quad 0 \le k \le n. \quad (4)$$

Now, the correctness of the recursive formulation is inductively proved as follows.

Theorem 1:The recursive formulation of (1)–(4) will produce correct sum for any number of bits and will terminate within a finite time.

Proof: We prove the correctness of the algorithm by induction on the required number of iterations for completing the addition (meeting the terminating condition).

Basis: Consider the operand choices for which no carry propagation is required, i.e., $C^0_i = 0$ for ∀i,i ∈[0..n]. The proposed formulation will produce the correct result by a single-bit computation time and terminate instantly as (4) is met.

Induction: Assume that $C^k_{i+1} \neq 0$ forsomeith bit at kth iteration. Let l be such a bit for which C^k_{l+1} =1. We show that it will be successfully transmitted to next higher bit in the (k+1)th iteration. As shown in the state diagram, the kth iteration of lth bit state (C^k_{l+1},S^k_l) and (l +1)th bit state C^k_{l+2},S^k_{l+1}) could be in any of (0,0), (0,1),or(1,0) states. As C^k_{l+1} =1, it implies that S^k_l =0. hence, from (3),C^{k+1}_{l+1} =0 for any input condition between 0to l bits.

We now consider the (l +1)th bit state (C^k_{l+2},S^k_{l+1}) for kth iteration. It could also 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 kth iteration will correctly produce output of(0,1) following (2) and (3). For(0,1) state, the carry successfully propagates through this bit level following (3).

Thus, all the single-bit adders will successfully kill or propagate the carries until all carries are zero fulfilling the terminating condition. The mathematical form presented above is valid under the condition that the iterations progress synchronously for all bit levels and the required input and outputs for a specific iteration will also be in synchrony with the progress of one iteration. In the next section, we present an implementation of the proposed architecture which is subsequently verified using simulations.

IV Design of CSA Multiplier

Generally multiplication consists of three steps: generation of partial products or PPs (PPG), reduction of partial products (PPR), and final carry-propagate addition (CPA). Different multiplication algorithms vary in the approaches of PPG, PPR, and CA.

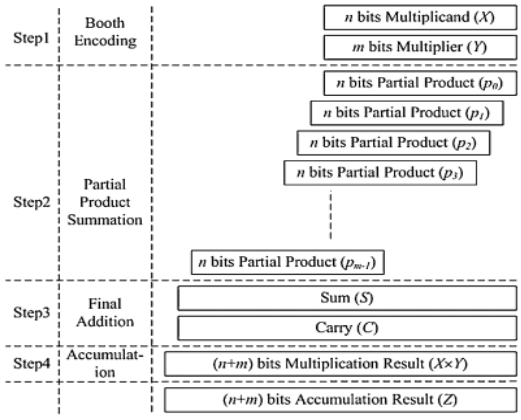

Fig 3:Basic arithmetic steps of multiplication and accumulation.

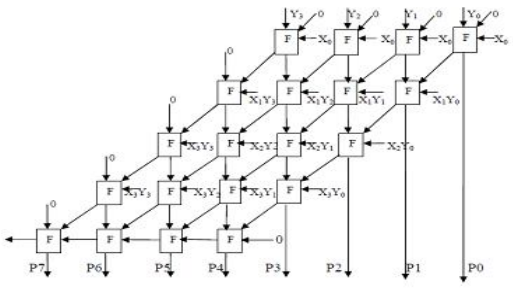

Fig.4.Multiplier with Carry saves Adder Architecture

In the Carry Save Addition method, the first row will be either Half-Adders or Full-Adders. If the first row of the partial products is implemented with Full-Adders, Cin will be considered „0". Then the carries of each Full-Adder can be diagonally forwarded to the next row of the adder. The resulting multiplier is said to be Carry Save Multiplier, because the carry bits are not immediately added, but rather are saved for the next stage. In the design if the full adders have two input data the third input is considered as zero. In the final stage, carries and sums are merged in a fast carry-propagate (e.g. ripple carry or carry look ahead) adder stage

V. SIMULATION RESULTs

The written Verilog HDL Modules have successfully simulated and verified using Modelsim6.4b and synthesized using Xilinxise10.1.

Synthesis Results:

RTL schematic

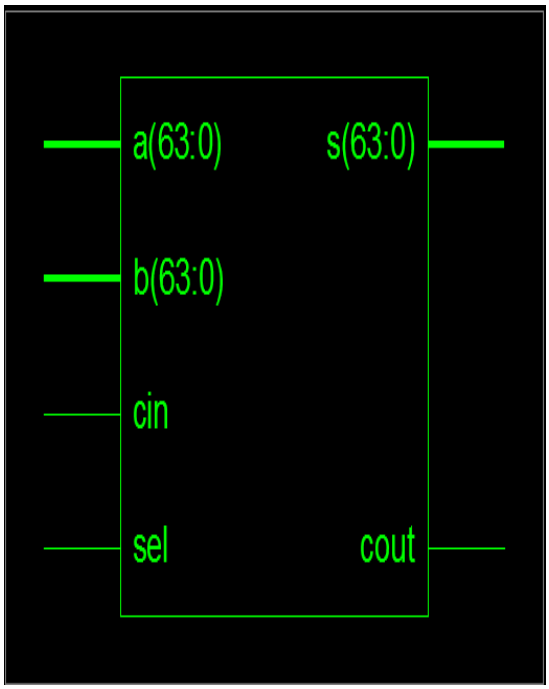

Fig 5: Proposed PASTA

Fig.6.Schematic of the proposed architecture

Design Summary;
PASTA:

jrjueue Project Status (10/03/2016 - 16:44:01)			
Project File:	jrjueue.ise	Current State:	Synthesized
Module Name:	pasta64	• Errors:	No Errors
Target Device:	xc3s500e-4fg320	• Warnings:	21 Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	

jrjueue Partition Summary
No partition information was found.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	80	4656	1%
Number of 4 input LUTs	144	9312	1%
Number of bonded IOBs	195	232	84%

Simulation results

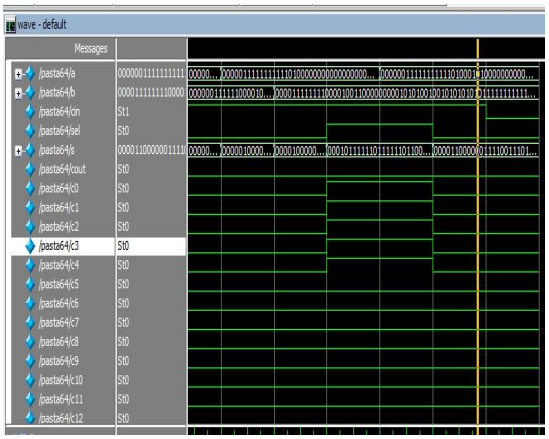

Fig 7:Simulation results

VI. Conclusion

These days speed of the multiplier has become an asset or constraint due to the importance of multiplier circuit in a wide variety of microelectronic systems. In this paper we analyzed different multiplier techniques taking speed as the main criteria. Carry save adder is proved to be more efficient in terms of speed compared to conventional multiplication techniques generated the output in 2.06 sec, whereas the booth multiplier generated the output in 3.09sec while the shift &add multiplier produce d the output in 2.31 sec. However the array multiplier generated the output in 2.75sec and modified booth multiplier

in around 3.08 sec.The carry save adder on the other hand consumes less hardware than other multiplication techniques.

REFERENCES

[1] GarimaTiwari "Analysis, Verification and FPGA Implementation of Low Power Multiplier".

[2] Kripa Mathew, S.AshaLatha, T.Ravi, E.Logashanmugam "design and analysis of an Array Multiplier using an Area Efficient full adder cell in 32 nm CMOS Technology".

[3] ChakibAlaoui "Design and Simulation of a Modified Architecture of Carrysave Adder".

[4]Deepali Chandel,Gagan Kumawat, Pranay Lahoty, VidhiVartChandrodaya, Shailendra Sharma.International Journal of Emerging Technology and Advanced Engineering Volume 3, Issue 3, March 2013"Booth Multiplier: Ease of multiplication".

[5] International Journal of Engineering Science InventionShaik.Kalisha Baba, D.Rajaramesh "Design and Implementation of Advanced Modified Booth Encoding Multiplier".

[6] G.W. Bewick, "Fast Multiplication: Algorithms and Implementation."Ph.D. dissertation, Stanford University, Feb. 1994

[7] Shiann-RongKuang, Jiun-Ping Wang, and Cang-Yuan Guo, "Modified Booth multipliers with a Regular Partial Product Array," IEEE Transactions on circuits and systems-II, vol 56, No 5, May 2009.

[8] 8.M. Zamin Ali Khan1, Hussain Saleem2, Shiraz Afzal3 and Jawed Naseem4, ― An Efficient 16-Bit Multiplier based on Booth Algorithm, international Journal of Advancements in Research & Technology, Volume 1, Issue 6, November-2012 ISSN 2278-7763

[9] Dr. Ravi Shankar Mishra,Prof. PuranGour,BrajBihariSoni, ―Design and Implements of Booth and Robertson„s multipliers algorithm on FPGA.‖ International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622.

[10] F.C Cheng, S. H. Unger, "Self-Timed Carry-Look Ahead Adders", IEEE Transactions on Computers, Vol. 49, No. 7, July 2000