DESIGN AND IMPLEMENTATION OF COMPUTATIONALLY EFFICIENT ARITHMETIC UNIT

A.Azhagu Jaisudhan Pazhani,
Assistant Professor,
Ramco Institute of Technology, Rajapalayam.
alagujaisudhan@gmail.com

Dr.C.Vasanthanayaki,
Associate Professor,
Government College of Technology, Coimbatore.
vasanthi@gct.ac.in

Abstract*— Approximate circuit designs empower us to swap over computation quality, for instance, exactness and computational attempt by abusing the characteristic oversight adaptability of various applications. As the computation excellence need of an application typically varies at runtime, it is attractive over to include the ability to reconfigure approximate circuits to satisfy such prerequisites and extra worthless computational attempt. These fresh structures are well-organized as far as area, speed, and power utilization concerning their exact adversaries. Multiplier and adder play a vital part in the present signal handling and different applications. With progresses in innovation, numerous specialists have attempted and are endeavoring to plan adder and multipliers which offer either of the following design targets – high speed, low power consumption, regularity of layout for VLSI implementation. In this paper approximate adders and splitting based multipliers are designed and the adder produce erroneous result. On the other hand, it generates erroneous outcomes, deterministically, in favor of a little portion of input combinations. In view of the fact that mistakes happen with enormously low likelihood, this new sort of adder and splitting based multiplier is significantly faster than state-of-the-art adders and multipliers at the point when the general latency is found the middle value of over numerous additions.

Index Terms— Approximate computing, Carry look-Ahead Adder, Reconfigurable adder and multiplier, splitting concept

I. INTRODUCTION

Power exploitation and performance are fundamental parameters in the arrangement of electronic circuit. Because of advanced handling frameworks, inferable from confined power spending designs and steadfastness concerns, achieving a desired execution level can be troublesome. Adders are the significant building blocks in Arithmetic and logic units (ALUs). Adders are utilized for expansion as well as for different tasks like subtraction, multiplication and division. Adders within arithmetic units are the most power devouring segments in the processor and are frequently hot-spot areas [1]. Decreasing the postponement and power exploitation of these blocks is an imperative and demanding task.

To enhance the exactness, Ripple Carry Adder (RCA) is utilized as a part of which the carry will get spread from the earlier stage to the following stage. In spite of the fact that it has more exactness than different adders, its postponement is expanded directly with expanding stages. With a specific end goal to lessen delay, Carry Select Adder (CSA) is utilized which is the cascaded form of Ripple Carry Adder. CSA prompts higher power exploitation than RCA. To meet up the area, power and speed design necessities, a variety of procedures at different arrangement reflection levels have been prescribed.

Approximate computing which is a rising model oversees request execution and power efficiency restrictions in favor of adders at the cost of diminishing the computational accuracy of the result [2]. This kind of design is especially sensible for applications where the suitable reaction is not noteworthy and an arrangement of genuinely exact answers are satisfactory. These applications consolidate multimedia processing, machine learning, signal processing, and other error adaptable computations [2]. Approximate arithmetic units are on a very basic level in light of the unraveling of the arithmetic unit circuits. Unmistakable structures for approximate adders have been anticipated in [3], [5]. In approximate arithmetic units however the outcome is not equivalent to correct value, it has low power exploitation and fast constrains (i.e.) minor delay which is considered as the primary necessity in the present electronics.

In Reconfigurable Error Tolerant Carry Look-Ahead Adder (RET-CLA) where the LSB part of the adder is composed in an approach to deliver rough outcome. The structure of the snake, which relies upon the traditional CLA, does not require an external amendment unit for the right include process. In this kind of adder having a goal that the deferral and power usage are should be extensively lighter. This is in light of the fact that, in the approximate made by abusing the power gating framework, the power usage is essentially decreased.

In this paper [7] they had used the basic array tree for the multiplication of two numbers. This architecture is simple so that the cost of the multiplier is reduced [7]. Wallace algorithm [8] is used to design Modified Wallace Multiplier to reduce the number of adders which reduces the complexity of the multiplier. The modified Wallace reduction reduces the number of half adders required by at least 80 percent compared to the conventional Wallace reduction with only a very slight increase in number of full adders. Both the conventional Wallace multiplier and modified Wallace multiplier have the same number of stages and consequently the delay is expected to be the same. The partial product reduction concept for integer multiplication is used in [9]. In this paper[10] High Performance splitting based Multiplier has been designed which has the ease of layout of a simple carry save reduction array, but in case of a high speed low power Dadda-style tree it has worst delay which depends on the logarithm of the word length N. The energy assignment of probabilistic (array) multipliers is studied in [10]. In particular, they had devised an energy assignment scheme that aims at minimizing the computational error caused by probabilistic multipliers. Low power and high performance multiplier [11] proposed different signed 16 x 16 bit approximate radix-8 Booth Multiplier and employed 2-bit adder to implement the lower part of an approximate recoding adder to overcome the issue commonly found in a radix-8 scheme. Further a truncated technique has been employed to save power and time.

Array based approximate computing [12] has designed a 16-bit fixed width Booth Multiplier based on the conventional array multiplier which consumes less energy and area than the conventional Booth Multiplier. Parallel reduced area multiplier [13] had used partial product reduction scheme to design a high speed, parallel multiplier which offer a modest speed advantage over Dadda and Wallace Multiplier because they employ a smaller carry-propagate adder.

This paper is organized as follows. Section II describes the introduction about arithmetic unit. Section III & IV explains the design of the proposed approximate arithmetic unit which includes reconfigurable error tolerant adder and splitting based multiplier. Section V & VI describes the implementation and performance analysis of the splitting based multiplier and proposed RET adder in terms of power, area and delay. Finally the work is concluded in section VII.

II. INTRODUCTION TO ARITHMETIC LOGIC UNIT

The arithmetic logic unit (ALU) is the core of a processor. In computing, an arithmetic and logic unit (ALU) is a digital circuit that performs integer arithmetic and logical operations. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs. ALUs are designed to perform integer calculations.

ALUs often handle the multiplication of two integers, since the result is also an integer. However, ALUs typically do not perform division operations, since the result may be a fraction, or a "floating point" number. Instead, division operations are usually handled by the floating-point unit (FPU), which also performs other non-integer calculations.

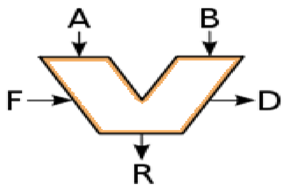

Fig.1. Arithmetic logic unit

The external control unit tells the ALU what operation to perform on that data, and then the ALU stores its result into an output register. The control unit is responsible for moving the processed data between these registers, ALU and memory. The inputs to the ALU are the data to be operated on (called operands) and a code from the control unit indicating which operation to perform.

The adder cell is the important functional unit of a multiplier. The constraints of the adder have to satisfy are area, power and speed requirements. A portion of the ordinary adders are ripple-carry adder, carry look ahead adder, carry-skip adder and Manchester carry chain adder. The delay in an adder is dominated by the carry propagation. Carry propagation analysis must consider transistor count and path delay. Partial products of multiplier are added with the help of adders which decides the area, delay and power consumption of the multiplier unit. Error tolerant systems play a very prominent role in many digital systems. Error tolerance will also be necessary in image and speech processing applications. Error tolerant arithmetic logic circuits play a vital role in all digital signal processing systems.

III. PROPOSED APPROXIMATE RET ADDER UNIT

In ordinary digital VLSI plan, one for the most part accepts that a usable circuit/framework ought to dependably give positive and exact outcomes. Be that as it may, truth be told, such impeccable tasks are rarely required in our non-computerized common encounters. In numerous applications, for example, a communication structure, the analog signal beginning from the external world should first be analyzed preceding being changed over to cutting edge data. The automated data are then arranged and transmitted in a hysterical channel ahead of altering back to a analog signal. In the midst of this technique, blunders may take place wherever. Truncation and round off mistakes in adders has turned out to be inevitable in present day VLSI technology. Another sort of adder i.e. error tolerant adder (ETA) is executed to persevere through those botches and to accomplish little power utilization and rapid execution in DSP frameworks. In ordinary adder circuit, delay is in a general sense attested to the carry proliferation chain along the fundamental route, from the LSB to MSB. On the off chance that the carry propagation can be removed by the procedure projected in this thesis, an awesome change in speed execution and power utilization is accomplished. The objective of this project is to design an Error tolerant ‘n’ bit adder using approximation logic and improved approximation logic. The Error Tolerant Adder can be used in the applications where there is no need to produce exactly correct numerical outputs like multimedia applications.

In the proposed adder, error tolerance is obtained by approximation logic in order to reduce the complexity at the gate level hence low power will be achieved. The error tolerant adder is outlined utilizing approximate full adder cells with decreased unpredictability at the door level. 16 bit traditional Ripple Carry Adder (RCA) and planned 16 bit Error Tolerant Adder utilizing approximation logic is actualized utilizing Xilinx 14.7. For each of the 2^{16} input mixes, examination is made amongst traditional and planned adders and the average error is calculated. The average error of the Error Tolerant Adder can be further reduced by using Improved Approximation Logic. In approximation logic, error tolerance is applied to whole data and in improved approximation logic, error tolerance is applied to the least significant part of input data and hence the average error will be reduced.

In the ordinary CLA, the carry outcome of the ith stage is resolved from:

$$C_{i+1} = G_i + G_{i-1} P_i + \ldots + G_0 \prod_{j=1}^{i} P_j + C_{in} \prod_{j=0}^{i} P_j$$

Here C_{in} is the input carry and Pi and Gi are the propagate $(A_i \oplus B_i)$ and generate $(A_i . B_i)$ signals of the ith phase, respectively. By expanding the width of CLA, the delay, area usage and power exploitation of the carry generator units increment. Here, we intend to utilize a window whose size is W, the smallest significant bit utilizes approximate adders whose output carry can be composed as,

$$C_{i+1} = (A_i + B_i).C_{in}$$

Here, C_{in} is the input carry and A_i, B_i are the relating inputs specified and C_{i+1} is the output carry at all stage. The outcome of (2) is known as the approximate value whose carry is provided as the contribution to the following stage which is carry look ahead adder. Obviously processing uncertain C_{i+1} is speedier and expends less power contrasted with computing C_{i+1} decisively. In this manner, when contrasted with the approximate estimated carry look ahead adder the postponement is decreased and furthermore area and power utilization are additionally diminished.

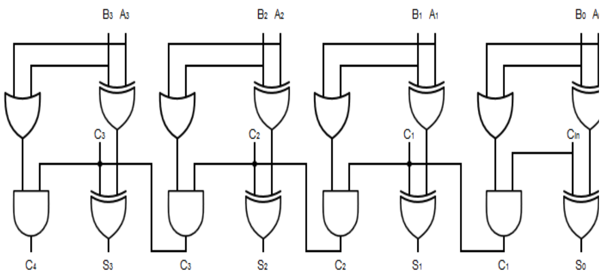

Fig.2 Approximate adder part of RET-CLA.

Fig.2 is the minimum noteworthy piece of reconfigurable error tolerant adder when the window dimension is four. The acquired carry C_4 is then provided to the following stage whose expansion is finished utilizing carry look ahead adder. In reconfigurable error tolerant adder since the slightest considerable part is altered the region is lessened fundamentally, thus power utilization is additionally decreased.

In the past adders anticipated, each piece has numerous quantities of stages to create carry C_i which expands region however in the planned paper the slightest noteworthy bits just uses 4 logic gates which need little zone. For instance if a 8 bit inputs are included by utilizing window dimension 4 then the slightest 4 bits utilizes Fig.3 and the most noteworthy bits utilizes carry look ahead adder. Also, if two 16 bit inputs are included utilizing window dimension 4 then slightest 4 bits utilizes Fig.2 and for the following three four bits the carry look ahead adder gets rehashed so the deferral will get extensively decreased.

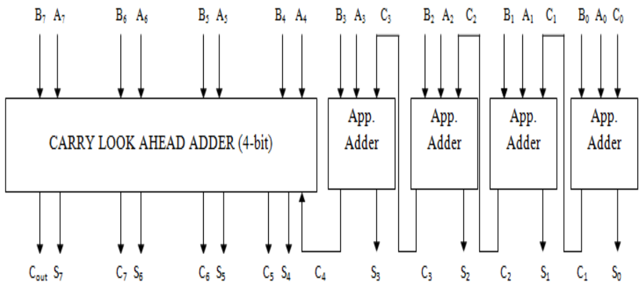

Fig.3. RET CLA block diagram

In this area, initially the error metric utilized to revise the precision of the approximate squares is error significance. Error significance is the arithmetical complexity among right and yield results; this assesses the measure of error. The precisions of the approximate adder of this effort alongside some different adders are looked at. To put side by side the precision of the proposed adder with those of the modern past works, the parameters talked about in the past subsection are measured. The similar examination incorporates the planned reconfigurable error tolerant adder and the already proposed reconfigurable approximate adder (RAP-CLA) [6] with the regular carry look ahead adder. Here the error measurements are contemplated on account of an 8-bit adder. The table I comprises of a portion of the example values for which the outcomes of approximate adder and error tolerant adder are contrasted with the yield of carry look ahead adder and the error percentage of both approximate adder and the projected adder are computed.

Table I Sample Inputs and Outputs of RET CLA

C_{in}	SAMPLE INPUT A	SAMPLE INPUT B	CLA (A+B)	RAPCLA (A+B)	Error% in RAPCLA	RETCLA (A+B)	Error% in RETCLA
0	147	198	345	345	0	341	1.16
0	43	178	221	221	0	221	0
0	195	60	255	255	0	255	0
0	87	202	289	257	11.07	285	0.35
0	255	255	510	510	0	480	5.88
1	147	198	346	346	0	346	0
1	43	178	222	220	0.9	222	0
1	195	60	256	254	0.78	256	0
1	87	202	290	256	11.72	290	0
1	255	255	511	511	0	511	0

From this it can be recognized that the error percentage of approximate adder differs from 0– 12% while in error tolerant adder it changes from 0-6% as it were. On observing the normal error percentage, already anticipated adder has 2.805% and the reconfigurable error tolerant adder has 1.375% which implies that the error rate lessens to half of the beforehand anticipated adder.

Table II Sample Inputs and Outputs of RET CLA

C_{in}	SAMPLE INPUT A	SAMPLE INPUT B	CLA (A+B)	RAPCLA (A+B)	RET (A-
0	10010011	11000110	101011001	101011001	1010
0	00101011	10110010	011011101	011011101	0110
0	11000011	00111100	011111111	011111111	0111
0	01010111	11001010	100100001	100000001	1000
0	11111111	11111111	111111110	111111110	1111
1	10010011	11000110	101011010	101011010	1010
1	00101011	10110010	011011110	011011100	0110
1	11000011	00111100	100000000	011111110	1000
1	01010111	11001010	100100010	100000000	1001
1	11111111	11111111	111111111	111111111	1111

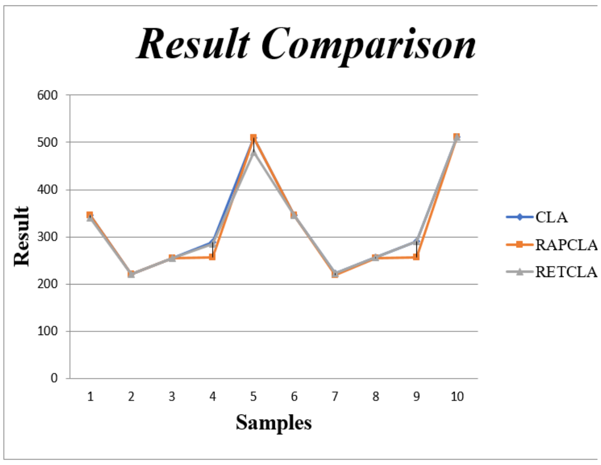

Fig.4.Graph plotted between number of samples and sum
value for CLA, RAPCLA and RETCLA for 8 bit inputs.

The graph shown in Fig.4 illustrates the correlation of results between traditional carry look ahead adder, reconfigurable approximate adder and reconfigurable error tolerant adder. The graph is plotted among numbers of test values to the total values. It can be recognized from the graph that both the outcomes of carry look ahead adder and error tolerant adder overlaps with each other which implies that the in spite of the fact that error tolerant adder is an approximate one, its outcome is equivalent to the exact value (i.e.) outcome of carry look ahead adder. The approximate adder marginally changes from the carry look ahead adder and the error level of approximate adder is elevated when contrasted with reconfigurable error tolerant adder.

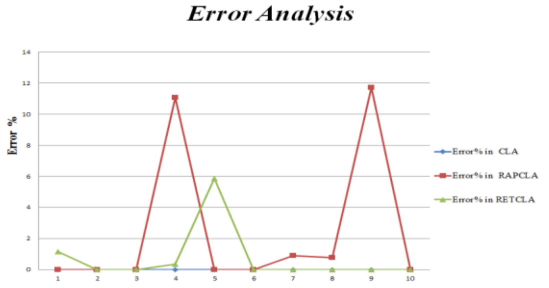

Fig. 5 Error analysis plot between CLA, RAPCLA and RETCLA.

The error analysis graph is exposed in Fig.5. It is plotted between the number of tests to the error percentage taken for those tests. At this point, carry look ahead adder which is the ordinary one has zero error percentage. The reconfigurable approximate adder has high error percentage (12%) when contrasted with every one of the adders taken here. As examined before the error percentage of reconfigurable error tolerant carry look ahead adder (6%) is definitely diminished when contrasted with every one of the adders talked about here. It can be noted down that after a maximum value the error percentage of error tolerant adder is diminished to zero.

IV. PROPOSED SPLITTING BASED MULTIPLIER

The traditional method for multiplication is done by using array multiplier. Array multiplier is popular due to its regular structure. It is based on add and shift algorithm. In parallel multiplication operation, number of partial products to be added is the main parameter that determines the execution of the multiplier. Every fractional item is produced by the duplication of the multiplicand with one multiplier bit. The limited products are then shifted according to their bit order and then added. Steps involved in the proposed multiplier:

1. Partitioning the Multiplicands
2. Splitting the Multiplicand
3. Computing the Products
4. Addition the partial products using carry skip adder

A. *Partitioning the Multiplicands*

Let A and B be numbers of b bits such that multiplication of the inputs A and B gives the 2b bits product. Here A and B are split into t numbers as A_0, A_1, $A_{2,\ldots,}A_{t-1}$ andB_0, B_1, $B_{2,\ldots,}$ B_{t-1} each consisting of r bits where

$$t = b/r \qquad (1)$$

Consider the value of b = 8, r =2
Then

$$t=4 \tag{2}$$

A and B can therefore be expressed as

$$A = A_0 2^0+A_1 2^2+A_2 2^4+A_3 2^6 \tag{3}$$

$$B = B_0 2^0+B_1 2^2+B_2 2^4+B_3 2^6 \tag{4}$$

A. *Splitting the Multiplicand*

Now, let A be split into even and odd partitions A_e and $A_{o,}$ with the end goal that

$$A_e = A_0 2^0+A_2 2^4 \tag{5}$$

$$A_o = A_1 2^2+A_3 2^6 \tag{6}$$

In this manner, A_e consists of the constant partitions of A with the unusual partitions supplanted by zeros. It is clear that

$$AB = A_eB+A_oB \tag{7}$$

B. *Computing the Products*

The products A_eB and A_oB are formed by partitioning and adding the zeros which is shown in Fig.6 and by concatenating we avoid carries between the partitions. Here the computation is done in parallel to reduce delay.

The product AB is obtained by summing the A_eB and A_oB which is given by

$$A_eB = A_e\, B_0 2^0+ A_e\, B_1 2^2+ A_e\, B_2 2^4+ A_e\, B_3 2^6 \tag{8}$$

$$A_oB = A_0 B_0 2^2+ A_o\, B_1 2^4+ A_o\, B_2 2^6+ A_o\, B_3 2^8 \tag{9}$$

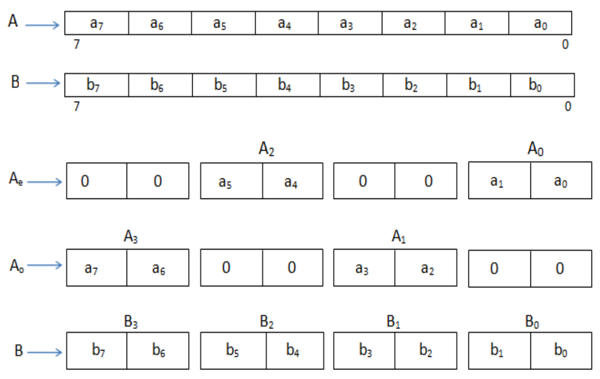

Fig. 6. Partitioning and adding zeros in the multiplicand

FORMATION OF PRODUCTS

The product A_eB is framed by including every summands from (8), and the item A_oB is framed by including each of the summands from (9). The final product AB can be obtained by adding A_eB and A_oB.

A. *Formation of A_eB*

The Least Significant Bit (LSB) of the even partition of A is multiplied with LSB of B and the product is concatenated with the product formed by the multiplication of Most Significant Bit (MSB) of the even partition of A and LSB of B. The above step is repeated for the subsequent significant of B and the each pair is added in parallel to get A_eB. The addition tree can be given by

$$\begin{array}{r} A_2\ \ B_0\ \ A_0\ \ B_0 \\ A_2\ \ B_1\ \ A_0\ \ B_1\quad\ \ \\ A_2\ \ B_2\ \ A_0\ \ B_2\quad\quad\ \ \ \\ A_2\ \ B_3\ \ A_0\ \ B_3\quad\quad\quad\quad \\ \hline \end{array}$$

$$A_eB_{\text{MSB}}\ldots\ldots\ldots\ldots\ldots\ldots A_eB_{\text{LSB}}$$

B. *Formation of A_eB*

The Least Significant Bit (LSB) of the even partition of A is multiplied with LSB of B and the product is concatenated with the product formed by the multiplication of Most Significant Bit (MSB) of the even partition of A and LSB of B. The above step is repeated for the subsequent significant of B and the each pair is added in parallel to get A_oB. The addition tree can be given by

$$\begin{array}{r} A_3\ \ B_0\ \ A_1\ \ B_0 \\ A_3\ \ B_1\ \ A_1\ \ B_1|\quad\ \ \\ A_3\ \ B_2\ \ A_1\ \ B_2\quad\quad\ \ \ \\ A_3\ \ B_3\ \ A_1\ \ B_3\quad\quad\quad\quad \\ \hline \end{array}$$

$$A_oB_{\text{MSB}}\ldots\ldots\ldots\ldots\ldots\ldots A_oB_{\text{LSB}}$$

Illustration of proposed splitting based multiplier

Let us consider an example for a manual calculation of the modified multiplier. The steps for the multiplication area as follows: Let A be the multiplicand and B be the multiplier.

A. *Step 1 (Partitioning)*

Let the values of A and B be

A -> 0 1 0 1 1 0 1 1
B -> 1 0 0 1 0 1 0 1

Here the input A is splitted into odd(A_o) and even(A_e) partitions which is given by

A_e-> 0 0 0 1 0 0 1 1
A_2 A_0
A_o-> 0 1 0 0 1 0 0 0
A_3 A_1
B -> 1 0 0 1 0 1 0 1
B_3 B_2 B_1 B_0

B. *Step 2 (Multiplication)*
The splitted partitions are increased as given below

A_0B_0 -> 0 0 1 1
A_2B_0 -> 0 0 0 1
A_0B_1 -> 0 0 1 1
A_2B_1 -> 0 0 0 1

C. *Step 3 (Concatenation)*
The multiplied partitions are connected as beneath

A_0B_0&A_2B_0 ->0 0 0 1 0 0 1 1
A_0B_1&A_2B_1 ->0 0 0 1 0 0 1 1
A_0B_2&A_2B_2 -> 0 0 0 1 0 0 1 1
A_0B_3&A_2B_3 -> 0 0 1 0 0 1 1 0

A_1B_0&A_3B_0 ->0 0 0 1 0 0 1 0
A_1B_1&A_3B_1 ->0 0 0 1 0 0 1 0
A_1B_2&A_3B_2-> 0 0 0 1 0 0 1 0
A_1B_3&A_3B_3 -> 0 0 1 0 0 1 0 0

D. *Step 4 (Shifting)*

Subsequent to the connection, answers are left moved in relation to the architecture to shape the beneath result

A_0B_0&A_2B_0<<0 -> 0 0 0 1 0 0 1 1
A_0B_1& A_2B_1<<2 -> 0 0 0 1 0 0 1 1 0 0
A_0B_2& A_2B_2<<4 -> 0 0 0 1 0 0 1 1 0 0 0 0
A_0B_3 & A_2B_3<<6 -> 0 0 1 0 0 1 1 0 0 0 0 0 0 0

A_1B_0&A_3B_0<<2 0 0 0 1 0 0 1 0 0
0
A_1B_1& A_3B_1<<4 -> 0 0 0 1 0 0 1 0 0 0 0 0
A_1B_2& A_3B_2<<6 -> 0 0 0 1 0 0 1 0 0 0 0 0 0 0
A_1B_3 & A_3B_3<<8 -> 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0

Step 5(Addition)

Finally the shifted answers are added to give the final result.

A_0B_0&A_2B_0<<0 -> 0 0 0 1 0
0 1 1
A_0B_1& A_2B_1<<2 -> 0 0 0 1 0 0 1
1 0 0
A_0B_2& A_2B_2<<4 -> 0 0 0 1 0 0 1 1 0
0 0 0
A_0B_3 & A_2B_3<<6 -> 0 0 1 0 0 1 1 0 0 0 0
0 0 0

0 0 1 0 1 1 0 0 0
0 1 1 1 1

A_1B_0&A_3B_0<<2 -> 0 0 0 1 0 0 1 0 0 0
A_1B_1& A_3B_1<<4 -> 0 0 0 1 0 0 1 0 0 0 0 0
A_1B_2& A_3B_2<<6 -> 0 0 0 1 0 0 1 0 0 0 0 0 0 0
A_1B_3 & A_3B_3<<8 -> 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0

0 0 1 0 1 0 0 1 1 1 1 0 1 0 0 0

In the proposed splitting based multiplier for the multiplication of each component the existing multiplier circuits such as traditional array multipliers, or even higher performance multipliers can be used, in order to trade off performance and area. It is also possible to use smaller splitting based multiplier to compound the speed improvements.Similarly, for the addition the existing adders such as ripple carry adder, carry save adder,prefix adder can be used. The repeated utilization of the splitting based multiplier likely be much more profitable at higher numbers of bits.

Multiplication Algorithm of RoBA Multiplier:

A Rounding-Based Approximate Multiplier designed for High-Speed however Energy-Efficient Digital Signal Processing and splitting based multiplier have been presented here. In RoBA multiplier, the operands are rounded to the nearest exponent of two. In splitting based multiplier the inputs are riving into odd and even partitions and adding the zeros in odd positions of the even partition and in even positions of the odd partition.

The operation of the approximate multiplier is first round off the values of inputs of A and B by Ar and Br, respectively. The multiplication of A by B may be rewritten as

$$A\times B = (Ar - A) \times (Br - B) + (Ar \times B) + (Br \times A) - (Ar \times Br)$$

The multiplications of Ar × Br, Ar × B , and Br ×A is implemented just by the shift operation

(Ar−A) × (Br−B) → The hardware implementation is complex.

This term in the final result, depends on differences of the exact numbers from their rounded ones, is typically small. Hence, we omit this part from the above equation and simplify the multiplication operation. To perform the multiplication process, the following expression is used:

$$A\times B \sim= (Ar \times B) + (Br \times A) – (Ar \times Br)$$

Thus, the multiplication operation can be performed using three shift and two addition/subtraction operation. Here the nearest values for *A* and *B* in the form of 2^n should be determined. When the value of *A* (or *B)* is equal to the $3 \times 2^{p-2}$ (where p is an arbitrary positive integer larger than one), it has two nearest values in the form of 2n with equal absolute differences that are 2^p and 2^{p-1}. While both values lead to the same effect on the accuracy of the modified multiplier, selecting the larger one (except for the case of p= 2) leads to a smaller hardware implementation for determining the nearest rounded value, and hence, it is considered in this paper.

It originates from the fact that the numbers in the form of $3 \times 2^{p-2}$ are considered as do not care in both rounding up and down simplifying the process, and smaller logic expressions may be achieved if they are used in the rounding up. Finally, it should be noted the advantage of the RoBA multiplier exists only for positive inputs because in the two's complement representation, the rounded values of negative inputs are not in the form of 2n. Hence, we suggest that, before the multiplication operation starts, the absolute values of both inputs and the output sign of the multiplication result based on the inputs signs be determined and then the operation be performed for unsigned numbers and, at the last stage, the proper sign be applied to the unsigned result. But in the case of splitting based multiplier produces accurate result when compared to RoBA and it take less time when compared to array and RoBA multiplier.

Multipliers	Bit size	Delay (ns)	No. of LUTs	No. of Latches
Array	4 bit	3.395	17	0
	8 bit	10.264	94	0
	16 bit	73.349	327	0
RoBA	4 bit	4.133	48	7
	8 bit	6.180	147	13
	16 bit	10.737	332	22
Splitting based Multiplier	4 bit	7.114	21	0
	8 bit	7.967	91	0
	16 bit	6.838	335	0

V. IMPLEMENTATION OF SPLITTING BASED MULTIPLIER

The Interlaced Partition multiplier was implemented for several values of b on Xilinx 14.7. The maximum combinational path delay in the design was compared to the delay of a typical array multiplier and RoBA implemented on the same process for each value of b. Additionally, the total transistor count in each design is used to quantify the area consumption of each multiplier. Each implemented multiplier has been extensively simulated with multiple input patterns to verify proper operation. The recreation outcome which has been taken by providing a quantity of the sample inputs to array multiplier, RoBA multiplier and splitting based multiplier.

TABLE III Analysis and Comparison of Delay Area of array, RoBA and Splitting based Multiplier

Table IV consists of two inputs whose lengths are 16-bits.The exact result is calculated using array, approximate output of RoBA multiplier and splitting based multiplier has been calculated and compared with the results of modified multiplier. The simulation result of Splitting based Multiplier is shown in the Fig.7.

Table IV Sample Inputs and outputs for 16-bit Array, RoBA and splitting based Multiplier

Sample input A	Sample input B	Array multiplier	RoBA multiplier	Splitting based multiplier
256	333	85248	75392	85248
312	400	124800	65536	124800
512	454	232448	181760	232448
720	907	653040	262144	653040
2048	678	1388544	1218560	1388544
555	1112	617160	524288	617160

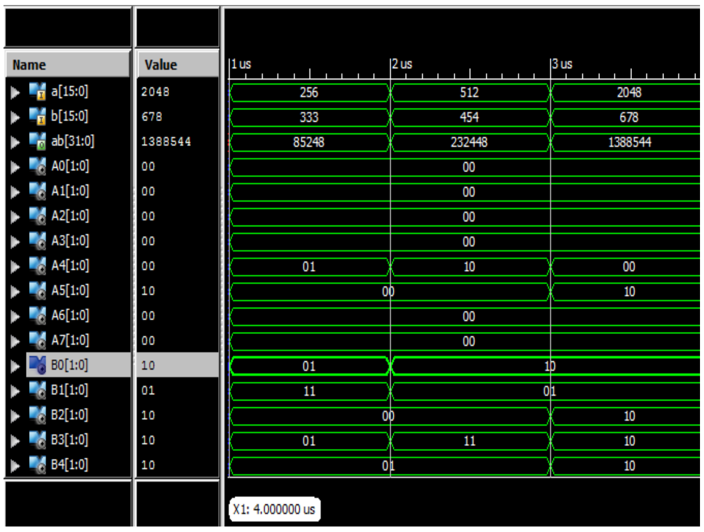

Fig.7.Simulation Result of Splitting based Multiplier

VI. RESULTS AND DISCUSSION OF RET CLA

In this area the outline parameters of the projected 8 bit reconfigurable error tolerant adder will be considered and the results of utilizing the projected adder in picture handling applications are displayed. The results for the delay, area and power of the 8-bit ordinary adder, approximate adder and error tolerant adder have been accounted for in Table V.

TABLE V Analysis and Comparison of Delay Area and Power of CLA, RAPCLA and RETCLA.

Parameter	CLA	RAP-CLA	RET-CLA
Delay (ns)	3.21	2.737	3.560
Number of slice LUTs	17	17	13
Power Used (mW)	104	113	84

In Table V the postponement has expanded in view of the approximate adder part (LSB part) of reconfigurable error tolerant adder in which the carry gets spread from one phase to other. In spite of the fact that it has higher deferral than ordinary and approximate adders, the zone have been diminished which thus lessens the power utilization of the segments and the error of proposed adder has been decreased incredibly. A standout amongst the most critical property of the proposed adder (RET-CLA) is that it possesses less measure of territory and furthermore delivers approximate outcomes having fewer blunders than approximate adder (RAP-CLA).

Table VI comprises of two data sources whose length is 8 bits. The correct outcome is computed utilizing CLA which has zero blunders and after that the approximate outcome of RAP-CLA has been figured and contrasted and the outcomes of RET-CLA. For various information tests the carry esteem is changed to 1 and 0 and the simulated outcomes are watched.

TABLE VI Sample Inputs and Corresponding Outputs for CLA, RAPCLA and RETCLA

C_{in}	SAMPLE INPUT A	SAMPLE INPUT B	CLA (A+B)	RAPCLA (A+B)	RETCLA (A+B)
0	10010011	11000110	101011001	101011001	101010101
0	00101011	10110010	011011101	011011101	011011001
0	11000011	00111100	011111111	011111111	011111111
0	01010111	11001010	100100001	100000001	100011101
0	11111111	11111111	111111110	111111110	111100000
1	10010011	11000110	101011010	101011010	101011010
1	00101011	10110010	011011110	011011100	011011110
1	11000011	00111100	100000000	011111110	100000000
1	01010111	11001010	100100010	100000000	100100010
1	11111111	11111111	111111111	111111111	111111111

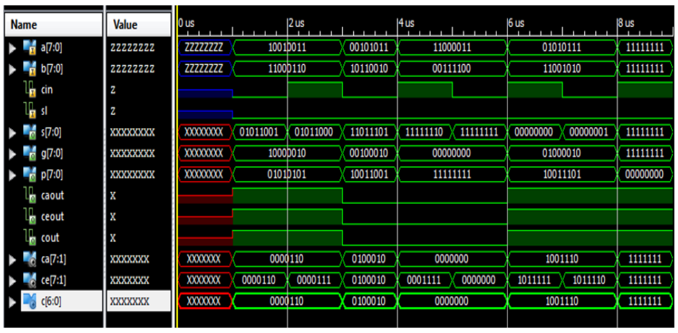

Fig.8 .Simulation result of RAPCLA

In Fig.8. The simulation outcomes of reconfigurable approximate carry look ahead adder (RAP-CLA) is appeared. A similar example esteems utilized for CLA are given to approximate adder whose outcome gets stray for the greater part of the qualities which produces error.

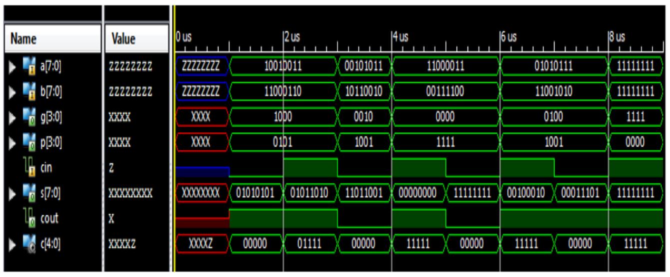

Fig.9.Simulation result of RETCLA.

The simulated outcome for reconfigurable error tolerant adder (RET-CLA) is appeared in fig.9. The projected reconfigurable error tolerant carry look ahead adder and multiplier are actualized in Spartan 6 FPGA unit. The execution is improved the situation 8 bit expansion

VII.CONCLUSION

In this paper, a reconfigurable error tolerant carry look-ahead adder and high speed multiplier was suggested. The Splitting based multiplier indeed provides an effective tradeoff between high speed. The domain in which it is a good choice becomes more pronounced as the number of bits increases. The Splitting based Multiplier has a place and represents a unique tradeoff for 32 and 64 bit cases. By using different component adders and multipliers, and through recursive implementations at higher numbers of bits, the

Splitting based multiplier can be optimized for a given application in terms of the trade-off between area and speed.

To survey the productivity of the projected design, its outline parameters were appeared differently in relation to those of a few proposed reconfigurable approximate adders and Splitting based multipliers. The parameters which added area, power and delay were assessed utilizing Xilinx 14.7 apparatus. The outcomes showed up moderately high performance and computationally efficient arithmetic unit. This proposed RET Carry look ahead adder and Splitting based multiplier finds application in image processing such as discrete cosine transforms.

REFERENCES

[1] B. K. Mohanty and S. K. Patel, "Area–Delay - Power Efficient Carry-Select Adder," *IEEE Transactions on Circuits and Systems II: ExpressBriefs*, vol. 61, no. 6, pp. 418-422, June 2014.

[2] B. Shao and P. Li, "Array-Based Approximate Arithmetic Computing: A General Model and Applications to Multiplier and Squarer Design," *IEEE Transactions on Circuits and Systems I:Regular Papers*,vol. 62, no. 4, pp. 1081-1090,April2015.

[3] A. Raha, H. Jayakumar, and V. Raghunathan,"Input- Based Dynamic Reconfiguration of Approximate Arithmetic Units for Video Encoding," *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, vol. 24, no. 99,pp.1-1,May2015.

[4] M. S. Khairy, A. Khajeh, A. M. Eltawil and F. J.Kurdahi, "Equi-Noise: A Statistical Model that Combines Embedded Memory Failures and Channel Noise," *IEEE Transactions on Circuits and Systems I:Regular Papers*, vol. 61, no. 2, pp. 407-419,Feb.2014.

[5] R. Ye, T. Wang, F. Yuan, R. Kumar and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," *Proceedings of IEEE/ACM International Conference on Computer Aided Design (ICCAD)*,2013,pp.48-54.

[6] OmidAkbari, Mehdi Kamal, Ali Afzali-Kusha and MassoudPedram, "RAP-CLA: A Reconfigurable Approximate Carry Look-Ahead Adder", IEEE Transactions on Circuits and Systems II: Express Briefs, 2016.

[7] C.S. Wallace,"A suggestion for a fast multiplier," IEEE Trans. Electron Comput., vol.EC-13, no.1, pp.14-17,Feb. 1964.

[8] L.Dadda,"Some schemes for parallel multipliers," Alta Frequency, vol.34,pp.349-356,1965.

[9] R.Waters and E. Swartzlander,"A reduced Complexity Wallace mutliplier reduction,"IEEE Trans.Comput., vol.59, no. 8, pp. 1134–1137,Aug. 2010.

[10] M. Schulte and E. Swartzlander, "Parallel reduced Area multipliers,"J.VLSI Signal Process.,vol.9,pp. 181–191, 1995.

[11]H.Eriksson,P.Larsson-Edefors,M.Sheeran,M.Sjalander, D. Johansson, and M. Scholin,"Multiplier reduction tree with logarithmic logic depth and regular connectivity," in Proc. IEEE Int.Symp. Circuits Syst.,2006, pp.4–8.

[12] MOSIS ON-Semiconductor (Formerly AMIS) Process [Online]. semiconductor/c5, 2015.

[13] P. Kogge and H. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," IEEE Trans. Comput., vol.C-22, no. 8, pp.783 791, Aug. 1973.

[14] W. J. Townsend, E. E. Swartzlander Jr., and J. A. Abraham "A comparison of Dadda and Wallace multiplier delays," in Proc. SPIE, Adv. Signal Process.Algorithms, vol. 5205, pp. 552–560, 2003.

[15] L.Dadda, "Some Schemes for Parallel Multipliers," Alta Frequenza, vol. 34, pp. 349-356, 1965.

[16] K. C. Bickerstaff, M. J. Schulte, and E.E. Swartzlander Jr.,"Parallel Reduced Area Multipliers,"J. VLSI Signal Processing Systems, vol. 9, no. 3, pp. 181191, Apr.1995.

[17] Shiang-RongKuang and Jiun-Ping Wang, "Design of Power Efficient Configurable Booth Multiplier", 2010.

[18] Li-Rong Wang, Shyh-JyeJou and Chung-Len Lee,"A Well Structured Modified BoothMultipierDesign",2011.