Emerging Trends in Electrical, Electronics & Instrumentation Engineering: An international Journal (EEIEJ), Vol. 1, No. 1, February 2014

DESIGN OF RADIX-8 BOOTH MULTIPLIER USING KOGGESTONE ADDER FOR HIGH SPEED ARITHMETIC APPLICATIONS

Paladugu Srinivas Teja

MTech,Department of Electronics and Communication Engineering, CVSR College Of Engineering,JNTU University, Hyderabad, A.P,India.

ABSTRACT

This paper presents the design and implementation of radix-8 booth Multiplier .The number of partial products are reduced to n/2 in radix-4We can reduce the number of partial products even further to n/3 by using a higher radix-8 in the multiplier encoding, thereby obtaining a simpler CSA tree .This implies less delay and a smaller area size .Since this multiplication operation is for both signed and unsigned numbers,cost of the system can also be reduced. The carry save adder (CSA) tree and the final adder can speed up the operation of multiplier. Koggestone adder is a parallel prefix form carry look ahead adder .We determine that by replacing carry save adder(CSA) and final two operand parallel prefix adder with parallel prefix adders of koggestone algorithm reduces delay furthur more resulting in substantial increase in speed of circuits.

KEYWORDS

Booth algorithm , Radix-8 , carry save adder , Koggestone adder , hard multiples.

1. INTRODUCTION

Multipliers play an important part in digital signal processing (DSP) systems. They are used in implementations of recursive and transverse filters, discrete Fourier transforms, correlation, range measurement[1]-[3].Regular advances in technology allowed to design multipliers which are both high-speed and has regularity in layout suitable for VLSI implementation. In any multiplication algorithm, the operation is reduced to a partial product summation. Every partial product denotes a multiple of the multiplicand which should be added to the final result[4]. In radix-2 algorithm,we form a series of products inbetween the multiplicand, Y, and each and every bit of the multiplier, X, resulting in partial products[5]. After that, all the partial products are added. We use some redundant arithmetic to get the additions as fast as possible. Usually the speed can be increased by a CSA tree.

In the conventional CSA tree, partial product bits with many inputs residing at the same bit position, are successively reduced to a final sum and carry pair with the help of a series of full adders which are single bit each. At the output, we will be left with sum and carry which has to be added by a carry-propagate adder (CPA).Where as radix-8 recoding provides gain in time while summing up the partial products as partial products are reduced to n/3 for n bits of multiplier and multiplicand compared to n/2 in radix-4[5].

However our multiplier is designed such that to modification in the previous adder stages. In this way, generation of odd multiple is speeded up even further. Another interesting point in the use of radix-8 recoding is the less number of transistors resulting in a reduced power dissipation and area size compared to radix-4[6].We will also reduce number of partial products using a higher radix-8 booth technique in the multiplier encoding and by replacing CSA tree with koggestone adder ,a parallel prefix form of carry look ahead adder we obtain a even lesser delay.

2. PROPOSED RADIX-8 ENCODING TO OBTAIN PARTIAL PRODUCTS

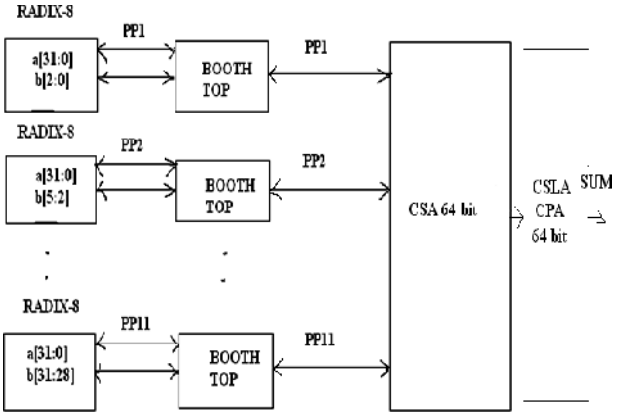
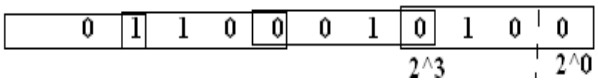


Fig 1. Proposed radix-8 booth multiplier

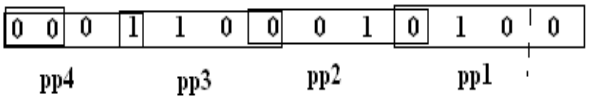
Radix-8 encoding applies the similar algorithm to radix-4, but here we take quartets of bits instead of triplets. Each quartet is coded as a signed-digit using the table 1

1) Consider two inputs of 10 bits each, x=0010010100(148) and y=0110001010(394)

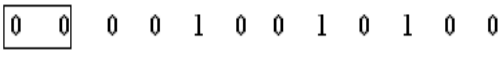
2) Append a 0 to the lsb of the y and group the bits according to radix-8



3) Denote each group as a partial product and add necessary bits to complete group of y i.e., if msb is 1 add 1's and if msb is 0 add 0's.Denote it as multiplier b



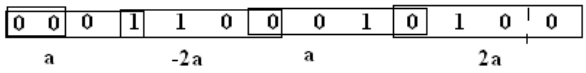
4) Append two 0's or two 1's to msb of the x by checking msb of x and denote it as multiplicand



5) Denote partial products groups of b according to radix-8 encoding table given below

PP bits of b	Partial Products
0 0 0 0	0a
0 0 0 1	+1a
0 0 1 0	+1a
0 0 1 1	+2a
0 1 0 0	+2a
0 1 0 1	+3a
0 1 1 0	+3a
0 1 1 1	+4a
1 0 0 0	-4a
1 0 0 1	-3a
1 0 1 0	-3a
1 0 1 1	-2a
1 1 0 0	-2a
1 1 0 1	-1a
1 1 1 0	-1a
1 1 1 1	0a

Table 1 Radix-8 encoding



6) Obtain partial products by applying radix-8 encoding on multiplicand a

0 0 0 0 1 0 0 1 0 1 0 0 = a

7) Obtain 2a by shifting a to left once.4a is obtained by shifting a left twice

0 0 0 1 0 0 1 0 1 0 0 0 = 2a

8) To obtain –a we need to perform 2's complement on a

1 1 1 1 0 1 1 0 1 1 0 0 = -a(2's complement)

9) Obtain -2a by shifting –a once to left.-4a is obtained by shifting –a left twice

1 1 1 0 1 1 0 1 1 0 0 0 = -2a

10) To generate 3a we only have to add 2a and a and similarly -3a is obtained by adding-2a and –a.Here 3a,-3a are denoted as hard multiples.

11) pp2 is placed under pp1 after leaving three places from lsb of pp1,pp3 is placed after leaving six places from lsb of pp1 and so on.All remaining locations are filled with 0's.

12) Extend sign bits of all the partial products according to corresponding msb's i.e.,extend 1's for msb 1 and 0's for msb 0.

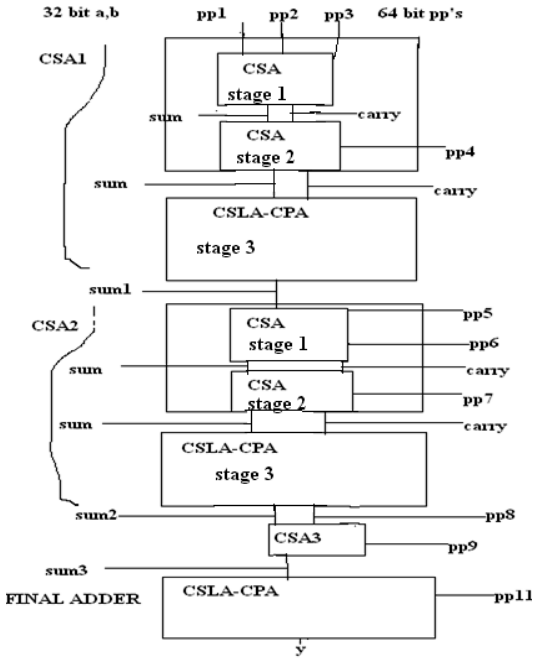


Fig 2. Reduced partial product accumulation using CSA tree for 64 bit

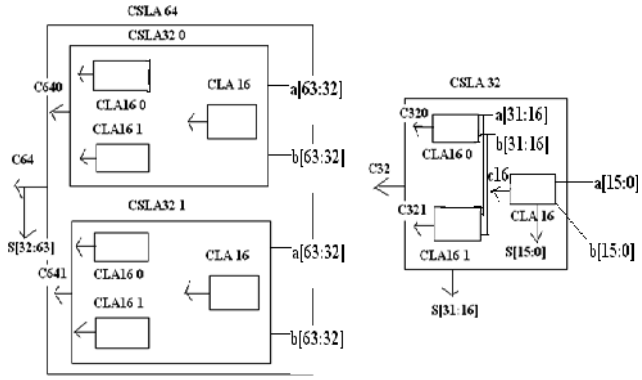


Fig 3 CSLA-CPA

3. REDUCED PARTIAL PRODUCT ACCUMULATION USING CSA

The first set of four partial products are given to CSA1 which is of 64 bits as shown in fig 2.It consists of two CSA stages.CSA stage 1 computes sum and carry from the partial products pp1,pp2,pp3 each of 64 bit.The obtained sum and carry along with pp4 forms the input for CSA stage2.This inturn produces final sum and carry of CSA1 which is given to CSLA-CPA to produce sum1.This process continues for CSA2.Here CSA stage1 has only pp5 and pp6 as inputs where as the the third input is the previous sum1 of CSA1.Finally CSLA-CPA of CSA2 produces sum2. Similarly the inputs for CSA3 are sum2,pp8 and pp9 which finally results in the sum3.The

complete partial product accumulation for 64 bit pp's is shown in fig 2.Now we are left with sum3 and pp11 which are given as inputs to final adder to produce the result y.

4. CSA TREE

A carry-save adder calculates the sum of three or more *n*-bit numbers in binary as shown in fig 4..It outputs two numbers of the same length as the inputs, one is the sum bits and other is the carry bits.

Consider the sum:12345678+87654322=100000000.

Using the arithmetic we learned as children, we go from right to left i.e., "8+2=0, carry 1", "7+2+1=0, carry 1", "6+3+1=0, carry 1", and so on until the end of the sum. Therefore adding two *n*-digit numbers has to take a time proportional to *n*, even if the machinery we are using would be capable of performing many calculations simultaneously.

Using binary bits in the above case implies if we have *n* one-bit adders, we still have to allot a time proportional to *n* to permit a possible carry propagation from one end to the other end of a number. Till then,

1) Result of the addition is unknown.

2) Result of the addition whether it is larger or smaller than a given number also cannot be determined.

The CSA tree however propogates the carries obtained in all the first stages immediately as inputs to the next consecutive adders in the second stage without waiting for entire column to be computed at a time. As a result of it next columns need not wait for carries until all stages in previous columns are computed. Similarly carries obtained in all second stages are propogated as inputs to next successive adders stage i.e., CSLA-CPA as shown in Fig 3.Therefore this mechanism reduces the computation time during partial product summation.

5. CSLA-CPA

The carry select mechanism is employed in order to parallelise the operation of a 16 bit carry propagate adder for 64 bits. a[31:16],b[31:16] are given to two 16 bit cpa's assuming carry as 1 and 0 respectively. This is done while a single cpa is calculating carry C16 from a[15:0] and b[15:0].Thus once the carry C16 is computed multiplexer chooses required appropriate carry to end up with carry C32.This is the first module. Similarly next 32 bits that is a[63:32],b[63:32] are computed parallely with the first module to obtain carry C64 as shown in fig 3.The sum is finally calculated by performing xor operation between propagate terms Pi and the carries Ci.

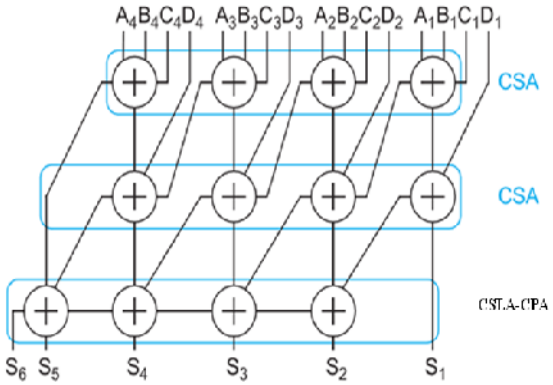


Fig 4 Carry save adder

6. KOGGESTONE ADDER FOR CSA

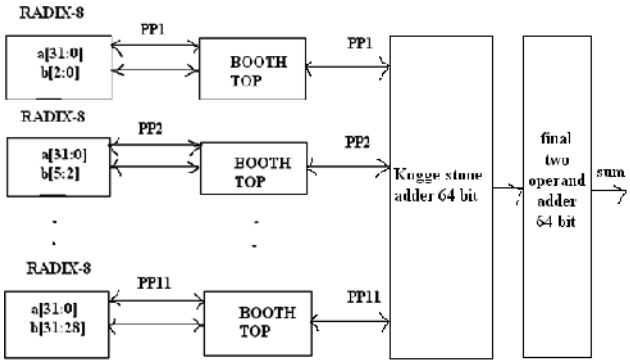


Fig 5. Basic block diagram

6.1. Parallel prefix adder basics

Suppose that A=A(n-1)…A_0 and B=B(n-1) … B_0 denotes the numbers which is to be added. S=S(n-1)… S_0 denotes their sum. An adder can be considered as a three-stage circuit as shown in fig 6.

The pre processing stage computes the carry-generate bits G_i, the carry-propagate bits P_i, according to

G(i)=ai*bi and P(i)=ai^bi

where pi and gi denote logical AND and exclusive- OR, respectively. The second stage also known as the carry computation stage, calculates the carries C_i with the help of generate and propagate bits G_i and P_i. The third stage which is the final adder stage calculates the sum using the formula,

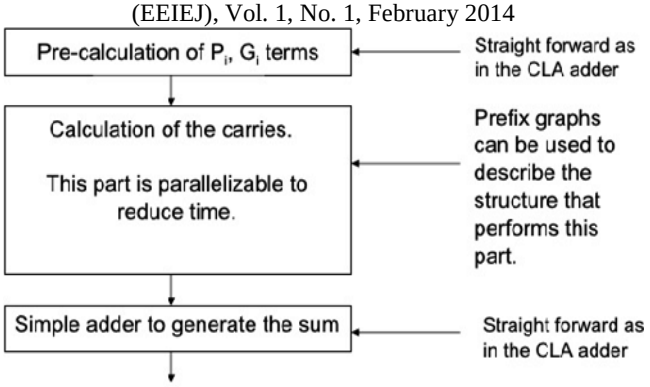


Fig 6. A block diagram of prefix adder

$$S(i)=p(i)\hat{}c(i)$$

As $C_i = G_i$ for every bit, many different algorithms have been developed for calculating all the carries [1][3].

6.2. The prefix carry tree

The enhancement is seen in the carry computation stage shown in fig 6. Koggestone algortihm is employed in this stage because it has log2N stages and also a fan-out of 2 at every stage[1][5]. For this an example of Kogge-Stone adder of 16 bit is shown in Fig 7.

This comes at the cost of long wires that has to be routed between stages. The adder tree also has more PG cells which may not affect the area if the layout of the adder tree is on a regular grid. Kogge-Stone adder is used in large word length adders because it has the minimum delay when compared to all other adders.

6.3 Grey cell:

$$G=cin*pi+gi$$

6.4 Black cell:

$$P=p(i+1)*pi$$

$$G=g(i+1)+p(i+1)*gi$$

The above shown koggestone structure is extended for addition of 64 bit partial products considering two partial products at a time.pp1 and pp2 are given as inputs to first koggestone adder ka1 inorder to obtain sum1.This sum1 along with pp3 is given to second koggestone adder ka2 to obtain sum2.This process continues until pp11 is computed to obtain final result y from ka10 as shown in fig 8.

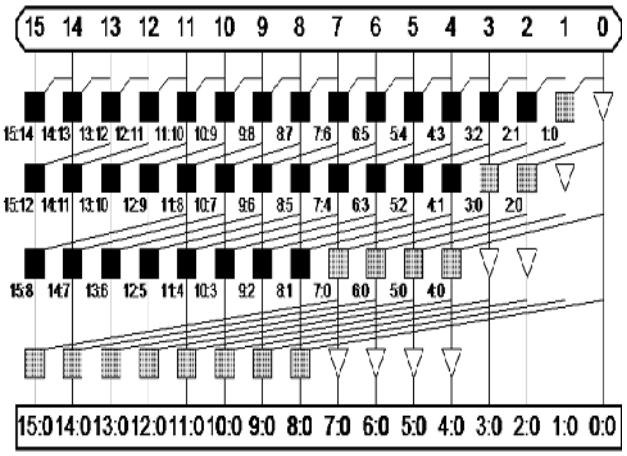


Fig 7. Kogge-Stone adder of 16 bit.

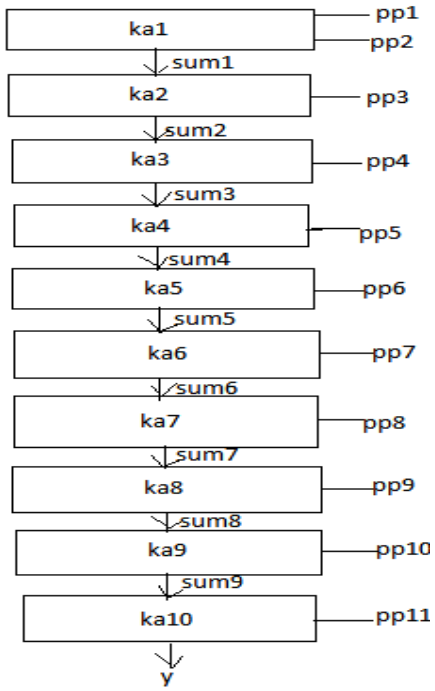


Fig 8. Partial product accumulation using koggestone adder of 64 bit

7. SIMULATION RESULTS

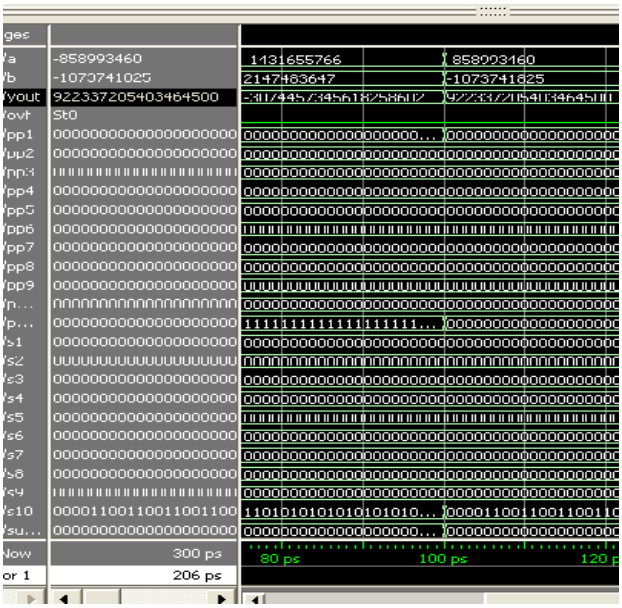


Fig. 9. Radix-8 booth multiplier using CSA adder

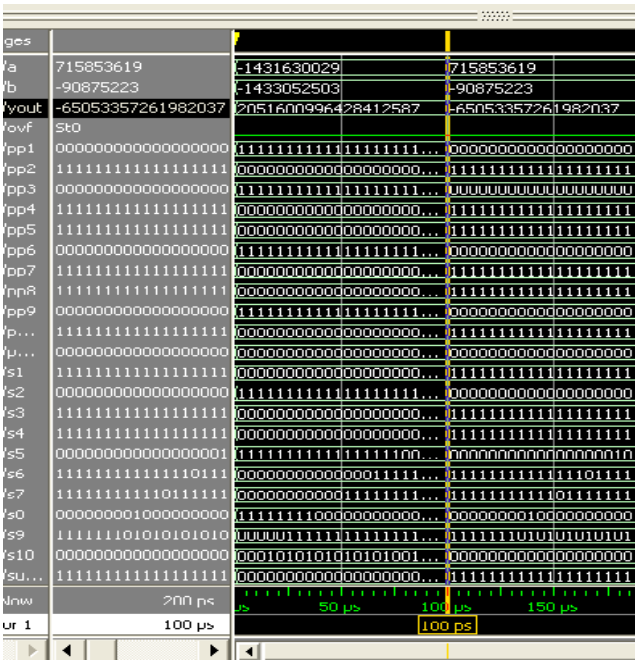


Fig. 10. Radix-8 booth multiplier using koggestone adder

8. PERFORMANCE COMPARISON

Logic utilization	Used	Available	Utilization
No of 4 input LUTs	2820	6144	45%
No of occupied slices	5178	12288	42%
No of bonded IOBs	129	240	53%

Table 2. Device utilisation Summary for Radix-8 booth multipler using CSA.

Delay	Radix-8 booth multiplier using CSA	Radix-8 booth multiplier using Kogge Stone adder
(ns)	53.294 ns	49.274 ns

Table 3. Delay comparison between Radix-8 booth multiplier using koggestone adder and CSA.

9. CONCLUSION

It has been performed the design and implementation of a 32 bit radix-8 booth multiplier. It has been proved that it can be useful to apply a radix-8 architecture in high speed multipliers because of the gain in time obtained due to reduction of partial products to n/3. The use of a radix-8 recoding is the less number of transistors resulting in a reduced power dissipation and area size, compared to a radix-4 architecture. Delay has been furthur reduced by replacing CSA with koggestone parallel prefix adder in the summation stage. Due to this overall multiplication time has been reduced with our radix-8 architecture.

REFERENCES

[1] "Parallel-prefix structures for binary and modulo {2^n -1, 2^n, 2^n + 1} adders", September 2011 by Jun Chen.
[2] "Embedded Cryptographic Hardware: Methodologies and Architectures", august 2012 by Nadia Nedjah, Luiza de Macedo mourelle.
[3] Shivanand pariwar and Raj singh "Efficient Floating Point 32-bit single Precision Multipliers Design using VHDL", Pilani, Engineering and Technology 2011.
[4] J.A. Hidalgo, V. Moreno-Vergara, O. Oballe, A. Daza, M.J. Martín-Vázquez, A.Gago ,"A Radix-8 multiplier design for specific purpose"@2011.
[5] lakshmanan, m. othman, m.a.m. ali, "design and characterization of parallel prefix adders using fpgas," journal of computers, vol. 5, no. 10, october 2012.
[6] L.P. Rubinfield, "A Proof of the Modified Booth's Algorithm for Multiplication," IEEE Transaction on computers, vol.39.

Author

Paladugu Srinivasteja received the B.Tech degree in Electronics & Communication Engineering from Netaji institute of engineering and technology (JNTU), AP, India, in 2011 and pursuing Masters in VLSI system design at Anurag Group of Institutions formerly known CVSR College of Engineering HYD, AP, India where he is engaged in designing of Radix-8 booth multiplier using koggestone adder for high speed arithmetic applications.