International Journal of VLSI System Design and Communication Systems

ISSN 2322-0929
Vol.04, Issue.06,
June-2016,
Pages:0442-0448

www.ijvdcs.org

Design and Implementation of Wallace Tree Multiplier using Higher Order Compressors

DHANYA M RAVI
Assistant Professor, Dept of ECE, Indo American Institutions Technical Campus, Anakapalli, AP, India,
E-mail: dhanu037@gmail.com.

Abstract: The Wallace tree multiplier is considered as faster than a simple array multiplier and is an efficient implementation of a digital circuit which is multiplies two integers. A Wallace tree multiplier is a parallel multiplier which uses the carry save addition algorithm to reduce the latency. There are many researchers have been worked on the design of increasingly more efficient multipliers. They aim at achieve higher speed and lower power consumption even while occupying reduced silicon area. The Wallace tree basically multiplies two unsigned integers. The new architecture enhances the speed performance of the widely acknowledged WTM. The synthesis is carried out by Xilinx ISE tool.

Keywords: WTM (Wallace Tree Multiplier), Xilinx ISE.

I. INTRODUCTION

Multiplication is one of the most area consuming arithmetic operations in high-performance circuits. As a consequence many research works deal with low power design of high speed multipliers. Multiplication involves two basic operations, the generation of the partial products and their sum, performed using two kinds of multiplication algorithms, serial and parallel. Serial multiplication algorithms use sequential circuits with feedbacks: inner products are sequentially produced and computed. Parallel multiplication algorithms often use combinational circuits and do not contain feedback structures. Multiplication of two bits produces an output which is twice that of the original bit. It is usually needed to truncate the partial product bits to the required precision to reduce area cost. Fixed-width multipliers, a subset of truncated multipliers, compute only n most significant bits (MSBs) of the 2n-bit product for n × n multiplication and use extra correction/compensation circuits to reduce truncation errors. In previous related papers, to reduce the truncation error by adding error compensation circuits. So that the output will be précised. In this approach jointly considers the tree reduction, truncation, and rounding of the PP bits during the design of fast parallel truncated multipliers so that the final truncated product satisfies the precision requirement. In our approach truncation error is not more than 1ulp (unit of least position), so there is no need of error compensation circuits, and the final output will be précised. The figure below shows how a Wallace Tree Multiplier can be realized for the 8-bit i.e. an 8x8 multiplier.

Advantages:

- Compare to dada multiplication delay is very high in wallace multiplication.
- Power consumption in the Wallace multiplier is high.
- Speed in Wallace multipier is normal compare to dada multiplier i.e delay and power is inversely proposals to each other. (delay is high).

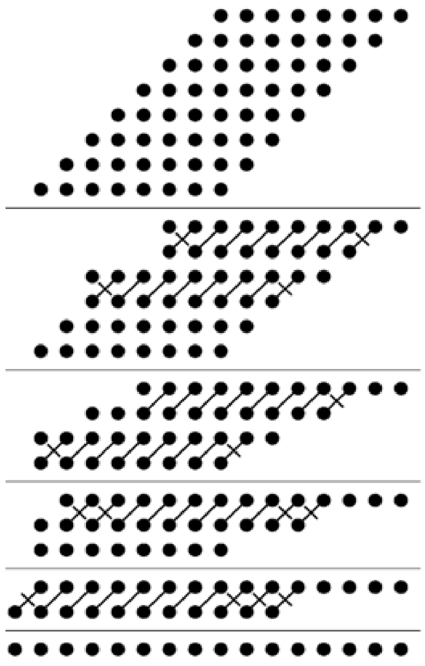
Fig.1.Method of Reduction On 8x8 Multiplier.

Disadvantages:

- Memory occupation in wallace multiplier is high copare to dada multiplier that is huge number of gates fabricated on a chip.
- Compare to dada multiplier Wallace multiplier gives low performance.i.e dada is modified version of the Wallace multiplier.

The multiplier is one of the key hardware blocks in most of the digital and high performance systems such as digital signal processors and microprocessors. With the recent advances in technology, many researchers have worked on the design of increasingly more efficient multipliers as shown in Fig.1. They aim at offering higher speed and lower power consumption even while occupying reduced silicon area. This makes them compatible for various complex and portable VLSI circuit implementations. However, the fact remains that the area and speed are two conflicting performance constraints. Hence, innovating increased speed always results in larger area. In this paper, we arrive at a better trade-off between the two, by realizing a marginally decreased delay which proportionally increases the speed performance through a small rise in the number of transistors. The new architecture enhances the speed performance of the widely acknowledged Wallace tree multiplier. The structural optimization is performed on the conventional Wallace multiplier, in such a way that the latency of the total circuit reduces considerably.

II. INTRODUCTION TO DIFFERENT ADDERS

A. Binary Adder Notations and Operations

As mentioned previously, adders in VLSI digital systems use binary notation. In that case, add is done bit by bit using Boolean equations.

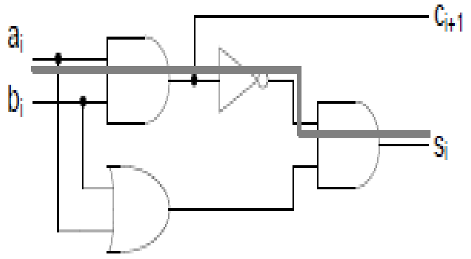

Fig.1. 1-bit Half Adder.

Consider a simple binary add with two n-bit inputs A;B and a one-bit carry-in c_{in} along with n-bit output S.

$$S = A + B + C_{in}$$

Where $A = a_{n-1}, a_{n-2}\ldots\ldots a_0$; $B = b_{n-1}, b_{n-2}\ldots\ldots b_0$.

The + in the above equation is the regular add operation. However, in the binary world, only Boolean algebra works. For add related operations, AND, OR and Exclusive-OR (XOR) are required. In the following documentation, a dot between two variables (each with single bit), e.g. a _ b denotes 'a AND b'. Similarly, a + b denotes 'a OR b' and a _ b denotes 'a XOR b'. Considering the situation of adding two bits, the sum s and carry c can be expressed using Boolean operations mentioned above.

$$S_i = a_i \wedge b_i$$
$$C_i + 1 = a_i \,.\, b_i$$

The Equation of C_i+1 can be implemented as shown in Fig.2. In the figure, there is a Half adder, which takes only 2 input bits. The solid line highlights the critical path, which indicates the longest path from the input to the output. Equation of c_i+1 can be extended to perform full add operation, where there is a carry input.

$$S_i = a_i \wedge b_i \wedge c_i$$
$$C_i + 1 = a_i \,.\, b_i + ai \,.\, c_i + b_i \,.\, c_i$$

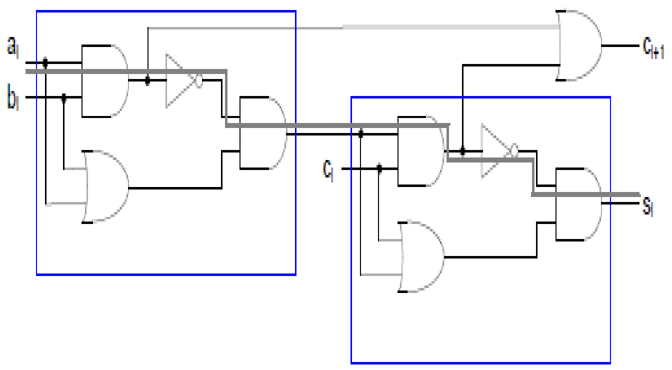

Fig.2. 1-bit Full Adder.

A Full adder can be built based on Equation above. The block diagram of a 1-bit full adder is shown in Fig.2.2. The full adder is composed of 2 half adders and an OR gate for computing carry-out. Using Boolean algebra, the equivalence can be easily proven. To help the computation of the carry for each bit, two binary literals are introduced. They are called carry generate and carry propagate, denoted by gi and pi. Another literal called temporary sum ti is employed as well. There is relation between the inputs and these literals.

$$G_i = a_i \,.b_i$$
$$P_i = a_i + b_i$$
$$T_i = a_i \wedge\ b_i$$

Where i is an integer and 0i < n.

With the help of the literals above, output carry and sum at each bit can be written as:

$$C_i + 1 = g_i + p_i \,.c_i$$
$$S_i = t_i \wedge c_i$$

In some literatures, carry-propagate pi can be replaced with temporary sum ti in order to save the number of logic gates. Here these two terms are separated in order to clarify the concepts. For example, for Ling adders, only pi is used as carry-propagate. The single bit carry generate/propagate can be extended to group version G and P. The following equations show the inherent relations.

$$G_i : k = G_i : j \;+\; P_i : j \,.\, G_j - 1 : k$$
$$P_i : k = P_i : j \,.\; P_j\text{-}1 : k$$

Where i : k denotes the group term from i through k.

Using group carry generate/propagate, carry can be expressed as expressed in the following equation.

$$C_i + 1 = G_i : j \;+\; P_i : j \,.\, C_j$$

B. Ripple Carry Adder

Ripple carry adder is an n-bit adder built from full adders. Fig.3 shows a 4-bit ripple carry adder. One full adder is responsible for the addition of two binary digits at any stage of the ripple carry. The carryout of one stage is fed directly to the carry-in of the next stage. Even though this is a simple adder and can be used to add unrestricted bit length numbers, it is however not very efficient when large bit numbers are used.

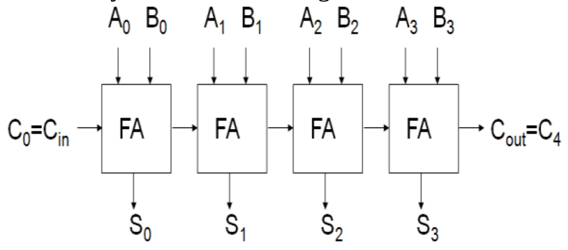

Fig.3. 4-bRippleCarryAdder.

One of the most serious drawbacks of this adder is that the delay increases linearly with the bit length. The worst-case delay of the RCA is when a carry signal transition ripples through all stages of adder chain from the least significant bit to the most significant bit, which is approximated by:

$$T = (n-1)\ t_c + t_s$$

Delay: The latency of a 4 bit ripplecarry adder can be derived by considering the worst-case signal propagation path. We can thus write the following expressions:

$$TRCA\text{-}4bit = TFA(A_0,B_0 \rightarrow Co)+T\ FA\ (C_{in} \rightarrow C_1)+TFA\ (C_{in} \rightarrow C_2)+\ TFA\ (C_{in} \rightarrow S_3)$$

And, it is easy to exten d to k-bit RCA:

$$TRCA\text{-}4bit = TFA(A_0,B_0 \rightarrow C_o)+(K-2)^*\ TFA\ (C_{in} \rightarrow C_i)+\ TFA\ (C_{in} \rightarrow Sk-1).$$

Drawbacks: Delay increases linearly with the bit length and Not very efficient when large bit numbers are used.

C. Carry Look-Ahead Adder

Lookahead carry algorithm speed up the operation to perform addition, because in this algorithm carry for the next stages is calculated in advance based on input signals. In CLA, the carry propagation time is reduced to O(log2(Wd)) by using a tree like circuit to compute the carry rapidly. Fig.4 shows the 4-bit Carry Look-Ahead Adder.

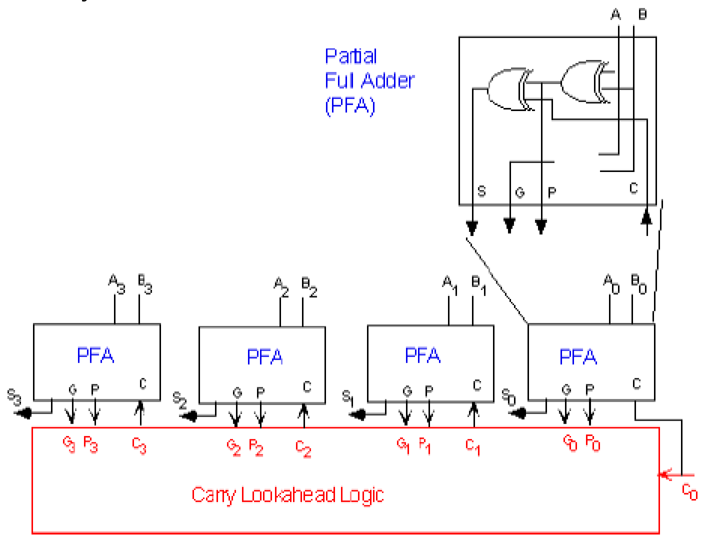

Fig.4. 4-bit Carry Look Ahead Adder.

The CLA exploits the fact that the carry generated by a bit-position depends on the three inputs to that position. If ‘X’ and ‘Y‘ are two inputs then if X=Y=1, a carry is generated independently of the carry from the previous bit position and if X=Y= 0, no carry is generated. Similarly if X ≠ Y, a carry is generated if and only if the previous bit-position generates a carry. ‘C’ is initial carry, “S” and “C_{out}” are output sum and carry respectively, then Boolean expression for calculating next carry and addition is:

P_i = X_i xor Y_i -- Carry Propagation
G_i = X_i and Y_i -- Carry Generation
$C_i + 1$ =G_i or (P_i and C_i) -- Next Carry
S_i = X_i xor Yi xor C_i -- Sum Generation

Thus, for 4-bit adder, we can extend the carry, as shown below:

$$C_1 = G_0 + P_0 \cdot C_0$$
$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$
$$C_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$
$$C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

As with many design problems in digital logic, we can make tradeoffs between area and performance(delay). In the case of adders,wecancreatefaster(butlarger)designsthantheRCA.TheCarryLookaheadAdder(CLA)isoneofthese designs (there are others too, but we will only look at the CLA).

Drawbacks: For long bit length, a carry look-ahead adder is not practical, but a hierarchical structure one can improve much. The disadvantage of CLA is that the carry logic block gets very complicated for more than 4-bits. For that reason, CLA’s are usual implemented as 4-bit modules and are used in a hierarchical structure to realize adders that have multiples of 4-bits.

D. Carry Save Adder

The carry-save adder reduces the addition of 3 numbers to the addition of 2 numbers. The propagation delay is 3 gates regardless of the number of bits. The carry-save unit consists of n full adders, each of which computes a single sum and carries bit based solely on the corresponding bits of the three input numbers. The entire sum can then be computed by shifting the carry sequence left by one place and appending a 0 to the front (most significant bit) of the partial sum sequence and adding this sequence with RCA produces the resulting n+1-bit value. This process can be continued indefinitely, adding an input for each stage of full adders, without any intermediate carry propagation. These stages can be arranged in a binary tree structure, with cumulative delay logarithmic in the number of inputs to be added, and invariant of the number of bits per input. The main application of carry save algorithm is, well known for multiplier architecture is used for efficient CMOS implementation of much wider variety of algorithms for high speed digital signal processing .CSA applied in the partial product line of array multipliers will speed up the carry propagation in the array.

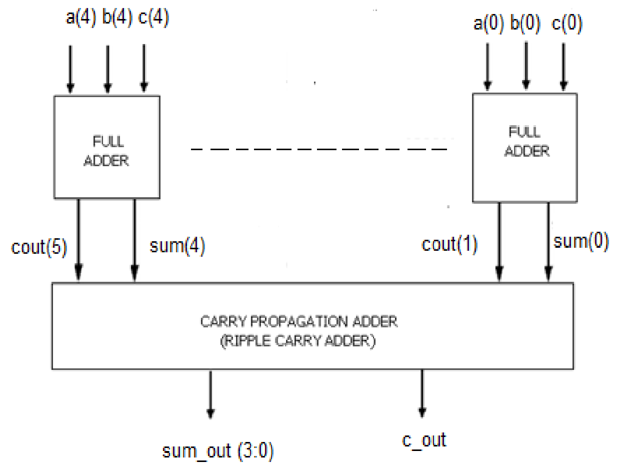

Fig.5. 4-bit Carry Save Adder.

Basically, carry save adder is used to compute sum of three or more n-bit binary numbers. Carry save adder is same as a full adder. As shown in the Fig.5, here we are computing sum of two 4-bit binary numbers, so we take 4 full adders at first stage. Carry save unit consists of 4 full adders, each of which computes single sum and carry bit based only on the corresponding bits of the two input numbers. Let X and Y are two 4-bit numbers and produces partial sum and carry as S and C as shown in the below :

$S_i = X_i$ xor Y_i ; $C_i = X_i$ and Y_i

The final addition is then computed as:

- Shifting the carry sequence C left by one place.
- Placing a 0 to the front (MSB) of the partial sum sequence S.
- Finally, a ripple carry adder is used to add these two together and computing the resulting sum.

Carry Save Adder Computataion:

X:	1 0 0 1 1
Y:	1 1 0 0 1
Z: +	0 1 0 1 1
S:	0 0 0 0 1
C: +	1 1 0 1 1
SUM:	1 1 0 1 1 1

In this design 126 bit carry save adder is used since the output of the multiplier is 126 bits (2N). The carry save adder minirnize the addition from 3numbers to 2 numbers. The propagation delay is 3gates despite of the number of bits. The carry save adder contains n full adders, computing a single sum and carries bit based mainly on the respective bits of the three input numbers. The entire sum can be calculated by shifting the carry sequence left by one place and then appending a 0 to most significant bit of the partial sum sequence. Now the partial sum sequence is added with ripple carry unit resulting in n + 1 bit value. The ripple carry unit refers to the process where the carryout of one stage is fed directly to the carry in of the next stage. This process is continued without adding any intermediate carry propagation. Since the representation of 126 bit carry save adder is infeasible, hence a typical 6 bit carry save adder is shown in the fig.6. Here we are computing the sum of two 126 bit binary numbers, then 126 half adders at the first stage instead of 126 full adder. Therefore , carry save unit comprises of 126 half adders, each of which computes single sum and carry bit based only on the corresponding bits of the two input numbers.

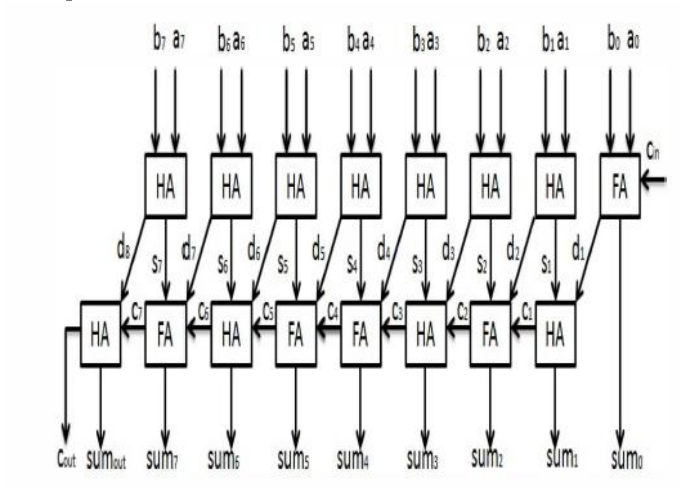

Fig.6. bit carry save adder.

If x and y are supposed to be two 126 bit numbers then it produces the partial products and carry as S and C respectively.

$$S_i = x_i \wedge y_i \quad (1)$$

$$C_i = x_i \,\&\, y_i \quad (2)$$

During the addition of two numbers using a half adder, two ripple carry adder is used. This is due the fact that ripple carry adder cannot compute a sum bit without waiting for the previous carry bit to be produced, and hence the delay will be equal to that of n full adders. However a carry-save adder produces all the output values in parallel, resulting in the total computation time less than ripple carry adders. So, Parallel In Parallel Out (PIPO) is used as an accumulator in the final stage.

III. INTRODUCTION OF WALLACE MULTIPLIER

Luigi WALLACE, the computer scientist has invented the WALLACE hardware multiplier during 1965. WALLACE multiplier is extracted form of parallel multiplier [5]. It is slightly faster and requires fewer gates. Different types of schemes are used in parallel multiplier. The WALLACE scheme is one of the parallel multiplier schemes that essentially minimize the number of adder stages required to perform the summation of partial products. This is achieved by using full and half adders to reduce the number of rows in the matrix number of bits at each summation stage. Even though the WALLACE multiplication has regular and less complex structure, the process is slower in manner due to serial multiplication process. Further, WALLACE multiplier is less expensive compared to that of Wallace tree multiplier. Hence, in this paper, WALLACE multiplier is designed and analysed by considering different methods using full adders involving different logic styles.

A. Implementation of Wallace Multiplier

The algorithm of WALLACE multiplier is based on the below matrix form shown in Fig.2. The partial product matrix is formed in the first stage by AND stages which is illustrated in Fig.7.

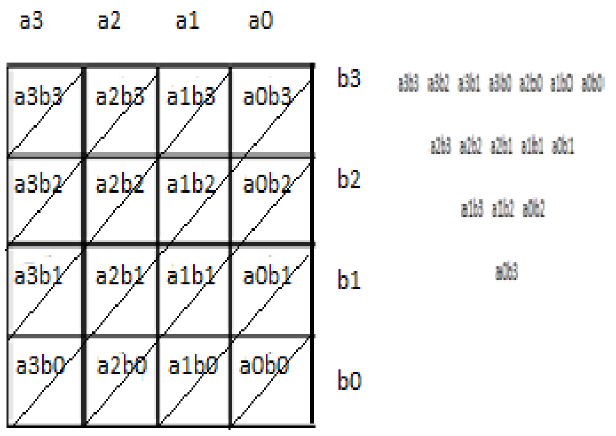

Fig.7.1-4x4 WALLACE Algorithm.

Steps involved in WALLACE TREE multipliers Algorithm:

- Multiply (that is - AND) each bit of one of the arguments, by each bit of the other, yielding N results. Depending on position of the multiplied bits, the wires carry different weights.
- Reduce the number of partial products to two layers of full adders as shown in Fig.8.
- Group the wires in two numbers, and add them with a conventional adder.

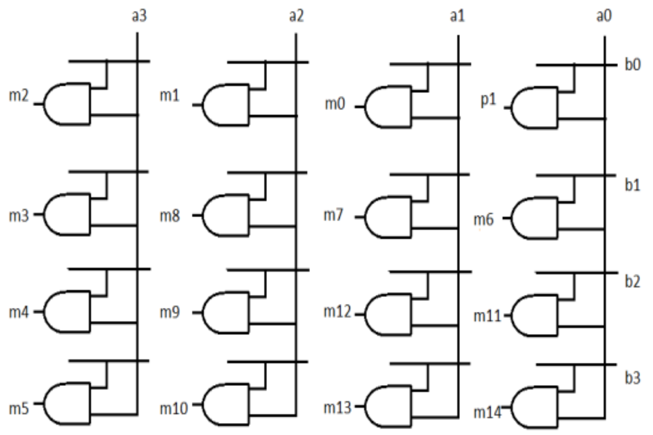

Fig.8. Product terms generated by a collection of AND gates.

B. Wallace Tree Multiplier Using Ripple Carry Adder

Ripple Carry Adder is the method used to add more number of additions to be performed with the carry in sand carry outs that is to be chained. Thus multiple adders are used in ripple carry adder. It is possible to create a logical circuit using several full adders to add multiple-bit numbers. Each full adder inputs a Cin, which is the C_{out} of the previous adder. This kind of adder is a ripple carry adder, since each carry bit "ripples" to the next full adder. The proposed architecture of WALLACE multiplier algorithm using RCA is shown in Figs.9 to 11 Take any 3 values with the same weights and gives them as input into a full adder. The result will be an output wire of the same weight.

- Partial product obtained after multiplication is taken at the first stage. The data's are taken with 3 wires and added using adders and the carry of each stage is added with next two data's in the same stage.
- Partial products reduced to two layers of full adders with same procedure.
- At the final stage, same method of ripple carry adder method is performed and thus product terms p1 to p8 is obtained.

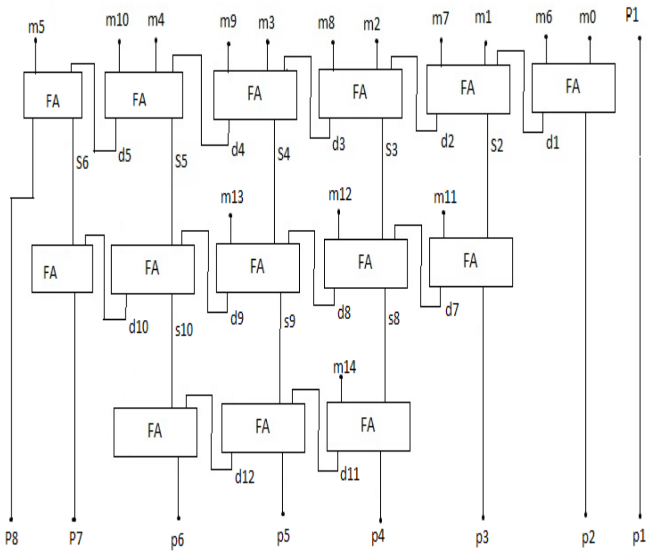

Fig.9.4x4 Wallace Multiplier Implementation.

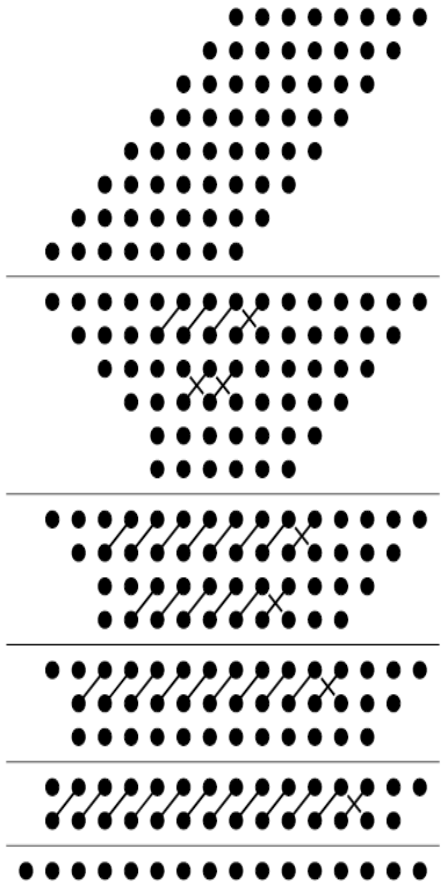

Fig.10. Method 8x8 Wallace Multiplier.

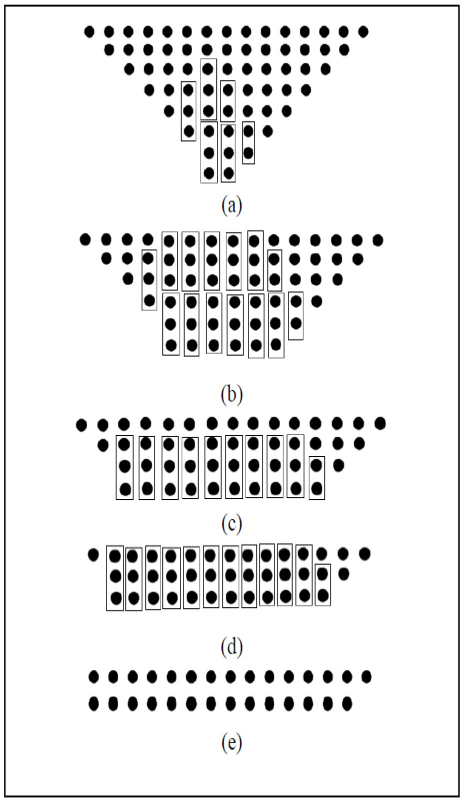

Fig.11. Column Compression scheme for 8x8 wallace multiplier.

Advantages:

- Compare to normal multiplication dalay is very low in Wallace multiplication.
- Power consumption in the Wallace multiplier is low.
- Speed is very fast i.e delay and power is inversely proposals to each other.

Disadvantages: Memory occupation in Wallace tree multiplier is high, that is huge number of gates are used. A Wallace tree is an efficient implementation of a digital circuit that multiplies two integers, devised by an Australian Computer Scientist. The Wallace tree has three steps:

- Multiply (that is - AND) each bit of one of the arguments, by each bit of the other, yielding results. Depending on position of the multiplied bits, the wires carry different weights, for example wire of bit carrying result is 32.
- Reduce the number of partial products to two by layers of full and half adders.
- Group the wires in two numbers, and add them with a conventional adder.
- The second phase works as long as there are three or more wires with the same weight add a following layer:
  - Take any three wires with the same weights and input them into a full adder. The result will be an output wire of the same weight and an output wire with a higher weight for each three input wires.
  - If there are two wires of the same weight left, input them into a half adder.
  - If there is just one wire left, connect it to the next layer.

The benefit of the Wallace tree is that there are only reduction layers, and each layer has propagation delay. As making the partial products is and the final addition is, the multiplication is only, not much slower than addition (however, much more expensive in the gate count). Naively adding partial products with regular adders would require time. These computations only consider gate delays and don't deal with wire delays, which can also be very substantial. The Wallace tree can be also represented by a tree of 3/2 or 4/2 adders. Generally it is combined with Booth encoding.

IV. RESULTS

Results of this paper is as shown in bellow Figs.12 to 14.

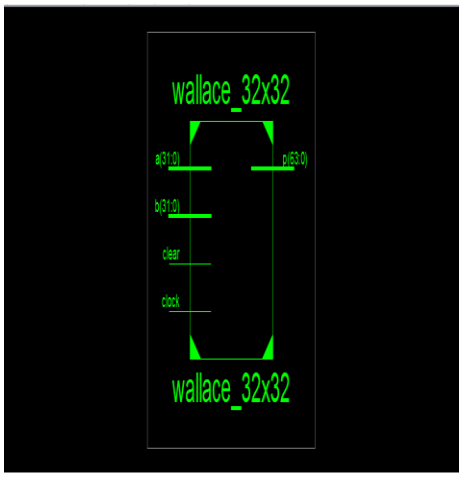

Fig. 12. Schematics.

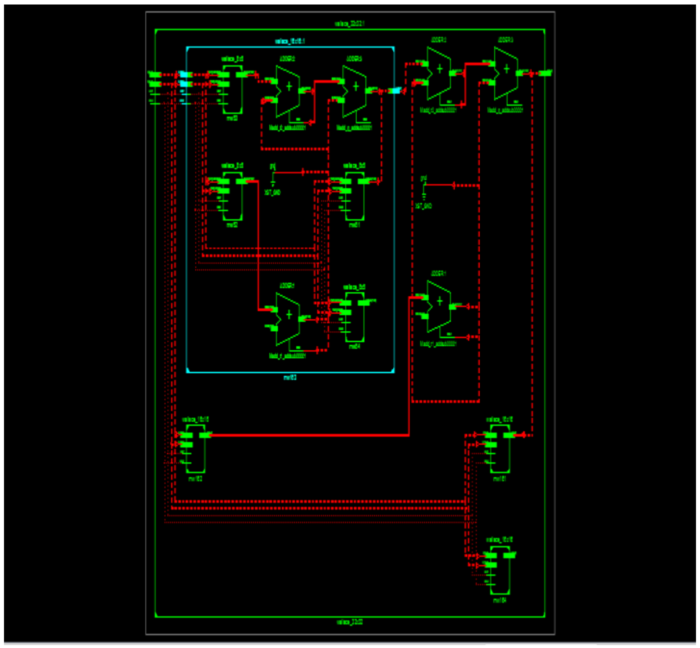

Fig.13. RTL Schematics.

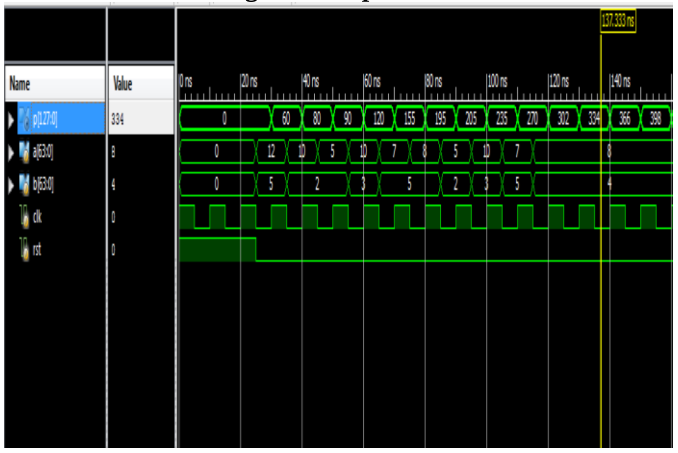

Fig.14. Waveforms.

V. CONCLUSION

The Design of high performance 32-bit Multiplier was implemented in this paper. The total unit operates at a frequency of 215MHz's with a total power dissipation of 155.532mW. Since the delay of 32-bit Multiplication is less, this design can be used in the system which requires high performance in processors involving large number of bits of the operation. The functionality of the Multiplication is verified using XILINX ISE 12.3i and synthesized using XILINX synthesizer.

VI. REFERENCES

[1] B. Cope, P. Cheung, W. Luk, and L. Howes, "Performance Comparison of Graphics Processors to Reconfigurable Logic: ACase Study," IEEE Trans. Computers, vol. 57, no. 4, pp. 433-446,Apr. 2010.
[2] S. Dikmese, A. Kavak, K. Kucuk, S. Sahin, A. Tangel, and H.Dincer,"Digital Signal Processor against Field Programmable Gate ArrayImplementations of Space-Code Correlator Beam former for SmartAntennas," IET Microwaves, Antennas Propagation, vol. 4, no. 5,pp. 573-577, May 2010.
[3] S. Roy and P. Banerjee, "An Algorithm for Trading off QuantizationError with Hardware Resources for MATLAB-based FPGA Design,"IEEE Trans. Computers, vol. 54, no. 5, pp. 666-676, July 2005.
[4] F. Schneider, A. Agarwal, Y.M. Yoo, T. Fukuoka, and Y. Kim,"A Fully Programmable Computing Architecture for MedicalUltrasound Machines," IEEE Trans. Information Technology inBiomedicine, vol. 14, no. 2, pp. 536-540, Mar. 2010.
[5] J. Hill, "The Soft-Core Discrete-Time Signal Processor Peripheral[Applications Corner]," IEEE Signal Processing Magazine, vol. 26,no. 2, pp. 112-115, Mar. 2007.
[6] J.S. Kim, L. Deng, P. Mangalagiri, K. Irick, K. Sobti, M. Kandemir,V. Narayanan, C. Chakrabarti, N. Pitsianis, and X. Sun, "AnAutomated Framework for Accelerating Numerical Algorithmson Reconfigurable Platforms Using Algorithmic/ Architectural Optimization," IEEE Trans. Computers, vol. 56, no. 12, pp. 1654-1665, Dec. 2007.
[5] H. Lange and A. Koch, "Architectures and Execution Modelsfor Hardware/Software Compilation and their System-LevelRealization," IEEE Trans. Computers, vol. 57, no. 10, pp. 1363-1355, Oct. 2010.
[6] L. Zhuo and V. Prasanna, "High-Performance Designs for LinearAlgebra Operations on Reconfigurable Hardware," IEEE Trans.Computers, vol. 55, no. 6, pp. 1055-1051, Aug. 2006.
[7] C. Mancillas-Lopez, D. Chakraborty, and F.R. Henriquez, "ReconfigurableHardware Implementations of Tweakable EncipheringSchemes," IEEE Trans. Computers,, vol. 57, no. 11, pp. 1545-1561, Nov. 2010.
[10] T. Guneysu, T. Kasper, M. Novotny, C. Paar, and A. Rupp,"Cryptanalysis with COPACOBANA," IEEE Trans. Computers,vol. 55, no. 11, pp. 1476-1513, Nov. 2006.