Evaluative Comparator Hardware Implementation for State-of-The-Art High Performance Multipliers

Rithika Shyam Chari Tarek Elarabi

Electrical and Computer Engineering Department   School of Engineering, PennState Behrend

***Abstract* — The multiplier undoubtedly is one of the most critical digital logic components in computer architecture. To achieve a faster response from a system, digital logic components need to respond faster with negligible error rates. Important factors in the consideration for implementing high performance/ speed multipliers are reduction in time delay, power at maximum speed and power delay product. This paper implements and compares high performance multipliers using various algorithms and techniques. It also analyzes the performance of 2, 4 & 8-bit multipliers based on Vedic Algorithm and 4 & 8-bit multipliers based on the Modified Booth Algorithm, and Wallace Tree. The conclusion on the most preferred choice of algorithm across the product dimension is made based on the maximum delay. The multiplier algorithms minimize the delay by reducing the number of partial products. Implementation and analysis of the results have been carried out using Verilog on Xilinx Vivado IDE.**

*Index Terms*—High Performance Multipliers, Booth Algorithm, Wallace Tree, Vedic, Urdhva Triyakbhyam, Ripple Carry Adder, Carry Save Adder.

# **Introduction**

Multiplication which is calculated by hand involves three steps namely- computing partial products, shifting and summing the partial products. Originally, hardware multiplication also followed the same procedure. But this traditional method is cumbersome when there are many partial products. The final output will be delayed until every partial product is generated. There are certain algorithms that are used to reduce the number of partial products thereby increasing the speed of computation. Some of such high-performance algorithms implemented and discussed herein are Vedic Algorithm, Modified Booth’s algorithm and the Wallace tree. The concept of Vedic Algorithm finds its origin from the ancient Indian texts called the ‘Sutras’ in general. The Sutra that discusses multiplication is called the ‘Urdhva Triyakbhyam’ [1] [7] sutra which literally means vertically and crosswise. This method greatly reduces the number of partial products thereby increasing the speed of calculation. The Booth Algorithm is used for multiplication of signed and unsigned numbers. It reduces the number of partial products by half. The algorithm uses inversions and shifts to implement multiplication by a sequence of 1’s. The Wallace tree was proposed by C.S.Wallace wherein partial products were summed up in parallel using a tree of Carry Save Adders (Wallace Tree). The computational delay is proportional to the log of operand size.

# **II. VEDIC ALGORITHM**

This algorithm is called Urdhva Triyakbhyam in the ancient Indian texts. It is much faster as it eliminates redundant computational load and it generates parallel intermediate products. This unique method of computation is fast even on large numbers.

**Vedic2x2:**Figure 1 is the representation of the Vedic method of multiplication calculation.



Fig 1: Urdhva Triyakbhyam-vertical and crosswise [1]

The same concept is employed in the calculation of the Vedic 2x2 multiplier. The schematic implementation of the 2x2 Vedic multiplier is shown in Figure 2. The 2x2 multiplier uses 4 AND gates to generate the partial products (vertical and crosswise) and computes the sums using two half adders to generate the sum and carry, which together make 4-bit long product. The concept of vertical and crosswise multiplication pairs is achieved through the AND gates. The partial products are then added using two half adders in cascade. The 4-bit output product is a three-bit sum and one bit carry. The circuit has been implemented in Xilinx and the output screenshot is as shown below in Figure 3.



Fig 2: Block diagram of Vedic 2x2 multiplier [1]

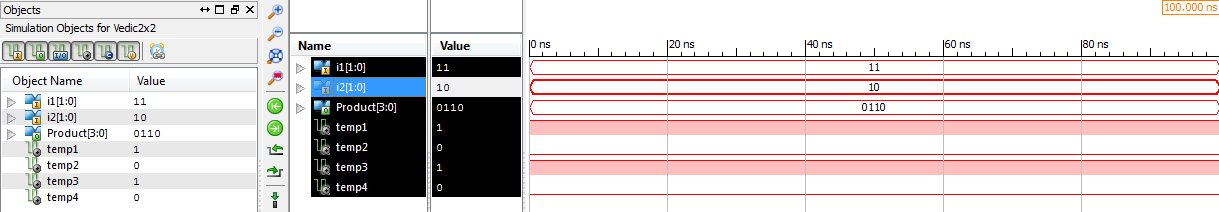


Fig 3: Output of Vedic 2x2 multiplier

**Vedic 4x4:**The 4x4 vedic multiplier [9] consists of four 2x2 Vedic multiplier blocks, a N-bit Carry Save Adder, a N+1-bit Ripple Carry adder and a N-bit Ripple carry adder. The block diagram for the same is realized as follows in Figure 4.



Fig 4: Schematic Vedic 4x4 multiplier [1]

The last two bits of the lower 4bit of 2x2 multiplier are the least two digits of the 8-bit product. The remaining two digits are appended with two zeros in front of them to convert them to a four-bit value. Then they are added with the partial products of the middle two 2x2 multipliers via a carry save adder. These adders compute at a higher speed as the sum and carry are computed in parallel. The carry bits are shifted to left by one bit and the sum bits are appended a zero at the MSB, and so two 5 bit numbers are formed. Then they are summed up using a ripple carry adder to form the 5-bit Sum. The last two digits of this sum are the second and third bits of the final 8-bit product. The remaining 3 bits are appended with a zero and added with the partial product of the last 2x2 vedic multiplier through a 4x4 ripple carry adder. Thus, the output is the 8-bit product. The circuit has been implemented in Xilinx and the output screenshot is as shown below in Figure 5.

**Vedic 8x8:**The 8x8 vedic multiplier [9] consists of four 4x4 Vedic multiplier blocks, two N-bit Ripple Carry Adders and a 12-bit Ripple Carry Adder. The block diagram for the same is realized as follows in Figure 6. The circuit has been implemented in Xilinx and the output screenshot is as shown below in Figure 7.

# **III. MODIFIED BOOTH ALGORITHM**

This algorithm is used for the multiplication of two signed binary numbers. It reduces the number of steps, in other words the number of partial products, required for the computation and thus faster. It compares the current bit and the right most bit and performs the following operations as per the following flowchart in Figure 8.

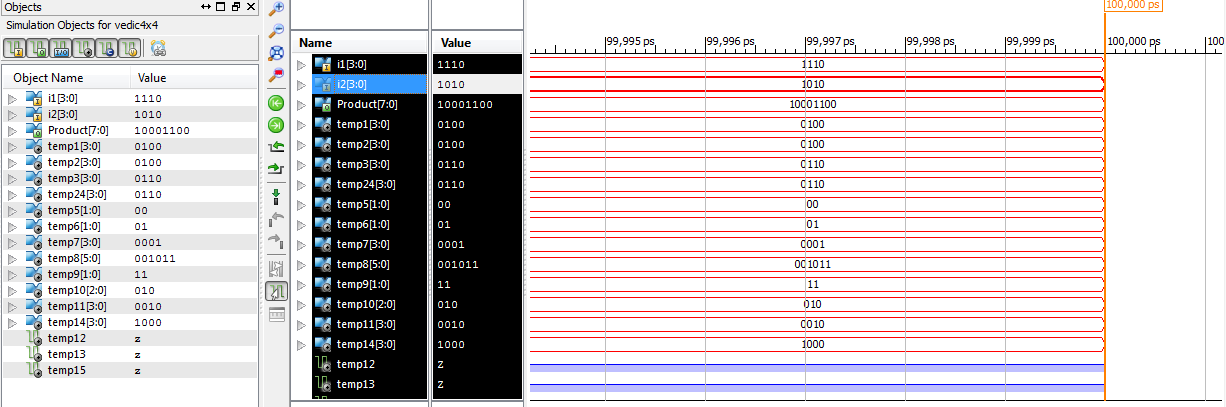


Fig 5: Output of Vedic 4x4 multiplier

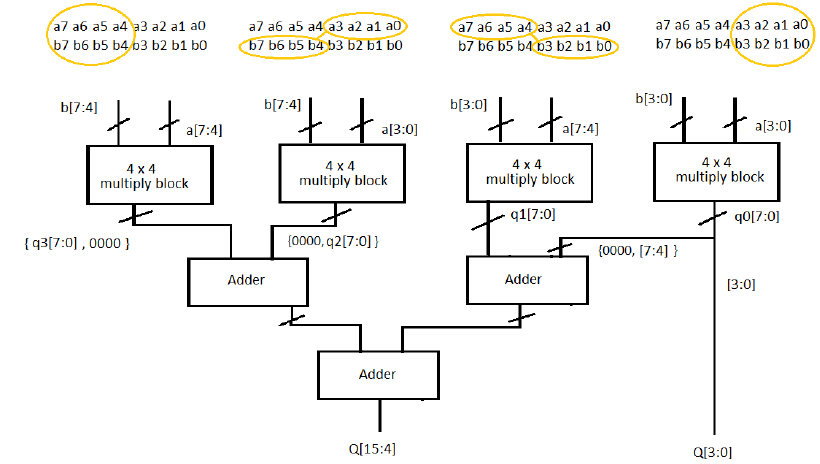
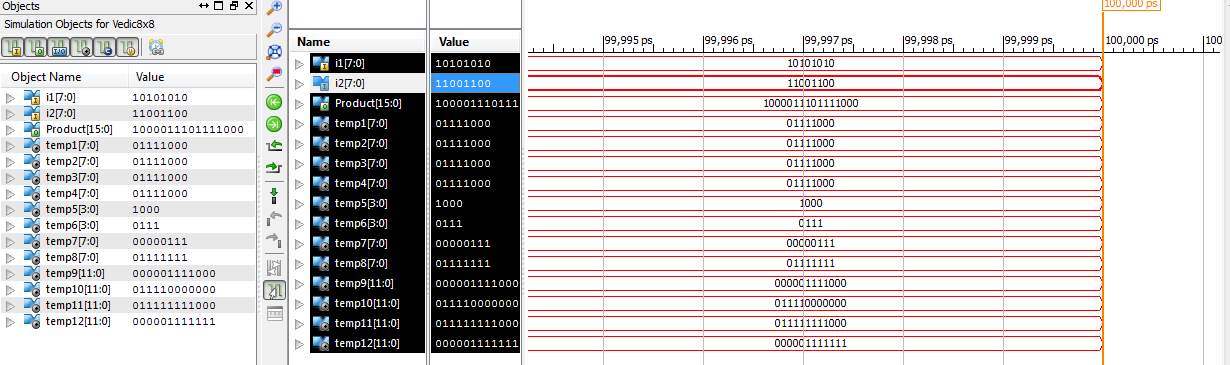


Fig 6: Schematic Vedic 8x8 multiplier [2]

Fig 7: Output of Vedic 8x8 multiplier

Practically, the Booth algorithm [8] is implemented by grouping the multiplier in sets of three bits with an overlapping bit from the previous triplet. For the first such triplet, a zero is appended as the LSB. Thus, an 8 bit multiplier will be split as M1, M0, 0; M1, M2, M3; M3, M4, M5 and so until the MSB is used. Every such input combination is encoded. As shown in Table I, these encodings are used to calculate the partial product when they are used as inputs (which are the 3 output bits of the Booth Encoder) to the Booth Decoder. The encodings are shown in Table I. The Booth Decoder generates partial products. These partial products are then sign extended, and appropriately the product is equal to the sum of these shifted partial products terms. Triplet grouping of the multiplier and the effect on the partial product:

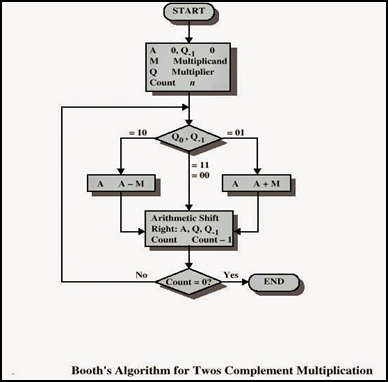


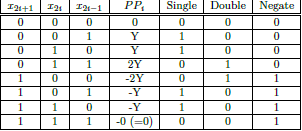
Fig 8: Booth Algorithm Flowchart [3]

Table I: Booth Encoding

|  |  |
| --- | --- |
| **Grouping** | **Effect on partial product** |
| 000 | 0\*Multiplicand |
| 001 | 1\*Multiplicand |
| 010 | 1\*Multiplicand |
| 011 | 2\*Multiplicand |
| 100 | (-2)\*Multiplicand |
| 101 | (-1)\*Multiplicand |
| 110 | (-1)\*Multiplicand |
| 111 | 0\*Multiplicand |

**Single Booth Encoder:**The truth table for a single Booth encoder is shown in Table II [4]. The encoder takes inputs *x*2*i*+1, *x*2*i*, and *x*2*i*−1 of the multiplier and produces three binary outputs for each triplet namely Single, Double, Negative. Figure 9 shows the schematic of the Booth encoder.

Table II: Truth table for Booth Encoder [4]



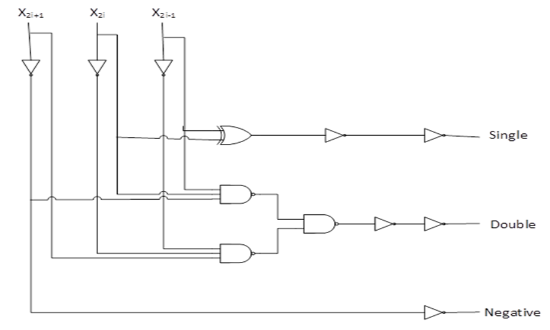
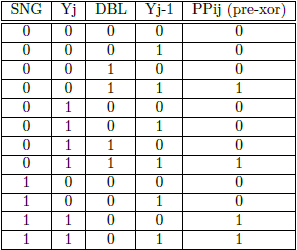


Fig 9: Schematic Booth Encoder [4]

**Single Booth Decoder:**The decoding block is (n+1) bits long. That is, for an 8-bit multiplicand, the decoder block is 9-bits wide. The three output signals from the Booth Encoder namely Single, Double and Negate are fed as inputs to the Booth decoder apart from two inputs (2-bits) of the multiplicand. Here, the LSB of the first decoding block is tied to ground. The remaining pairs of the multiplicand with an overlap from the previous pair are fed as multiplicand inputs to the decoder. The output of the decoder block is 1- bit of Partial Products (PP). The PPs are summed using a series of half adders that in turn sum the current partial product and the carry from the previous half adder. The output of every decoder block is a 9-bit wide PP[8:0]. The schematic of the single Booth decoder is represented in Figure 10. Table III show the truth table of the decoder.

Table III: Truth Table of single Booth decoder [4]



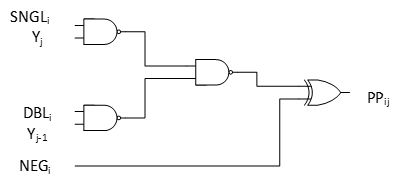


Fig 10: Schematic of Single Booth Decoder[4]

**Booth Multiplier 8x8:**The architecture of a booth algorithm [10] comprises of the four Booth Encoders and four 9-bit Booth Decoders, Sign extension for each of the partial products and the Carry Look Ahead adder blocks to sum the partial products. The schematic of an 8x8 booth algorithm implementation is shown in Figure 11. The circuit has been implemented in Xilinx and the output is shown below in Figure 12.

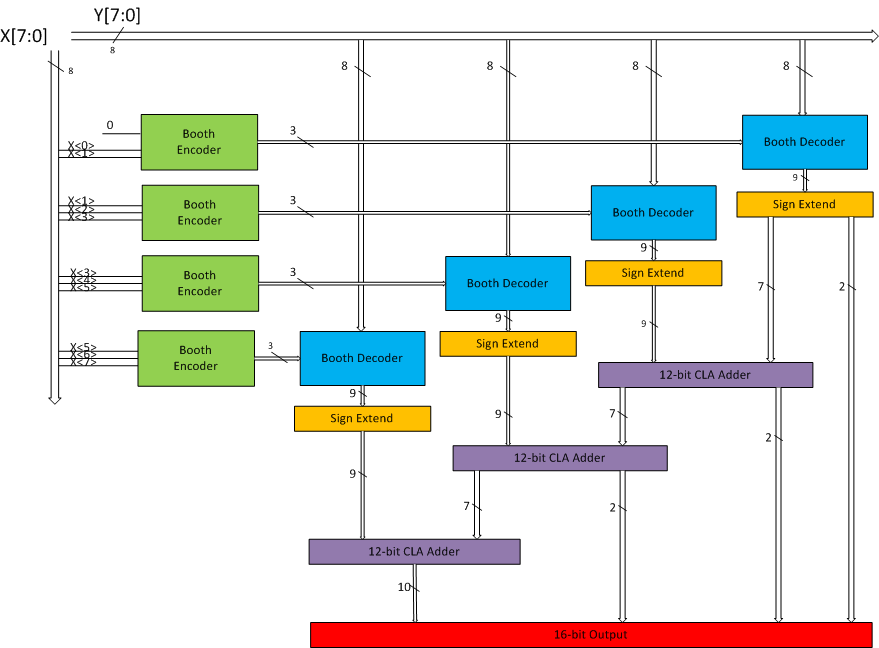
******

Fig 11: Schematic of 8x8 Booth algorithm [4]

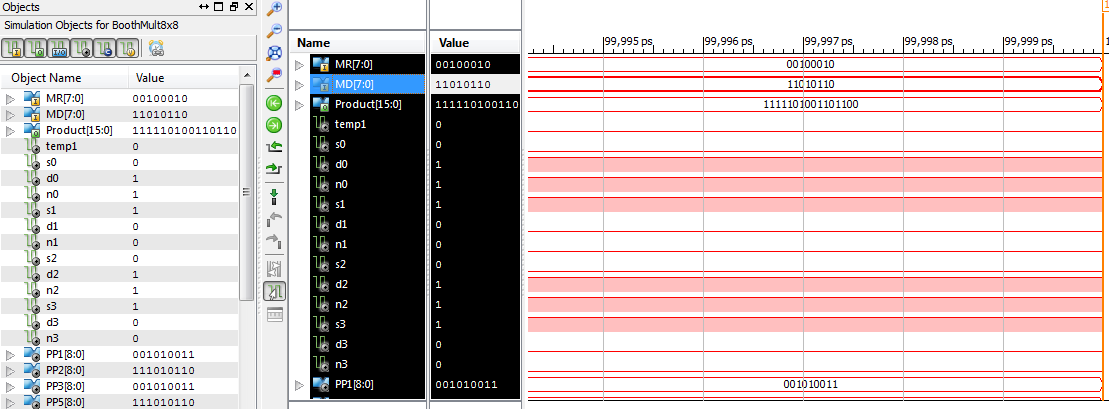
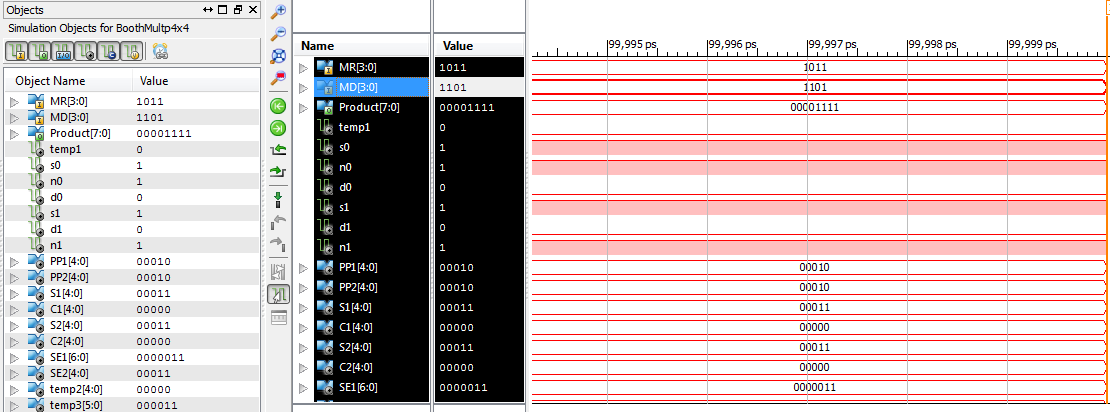


Fig 12: Booth 8x8 Xilinx Implementation

***Booth Multiplier 4x4:*** The architecture of the 4x4 Booth multiplier [10] is similar to that of the 8x8 Booth multiplier. The difference is being that the 4x4 multiplier uses two Booth Encoders and two 5-bit Booth Decoders and the corresponding sign extension and Carry Look Ahead adders. The circuit has been implemented in Xilinx and the output screenshot is as shown in Figure 13.

******Fig 13: Booth 4x4 Xilinx Implementation

**IV. WALLACE TREE MULTIPLIERS**

The Wallace tree multiplier is quite a popular one due to its speed of computation, ease of scalability, modularity and ease of fabrication. The Wallace Tree structure is designed to reduce the number of partial products at a rate of log3/2 (N/2). This type of multiplier is extremely efficient on high bit multiplications of the order of 16x16 bit and higher. The implementation starts with the grouping of partial products that have equal weight. The (m + n) of the partial products AmBn for various combinations of m and n are grouped and given as inputs to a series of half adders and full adders sequentially. Their sum and carry are further fed into full adders and half adders until you get all the bits of the final product.

**Wallace Tree Multiplier 4x4:**The concept of partial product generation and addition for a 4x4 multiplier is as shown in Figure 14. Note that partial products that have the same weights are summed up to give the final result. S30, S21, S12, S03 all have the same m + n = 3, where m and n are the bit subscript notations for the 4-bit long multiplier and multiplicand A and B. The architecture of the Wallace Tree Multiplier 4x4 is as shown in Figure 15. The circuit has been implemented in Xilinx and the output screenshot is shown in Figure 16.

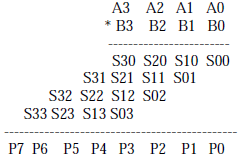


Fig 14: Partial Products for 4x4 Wallace design [5]

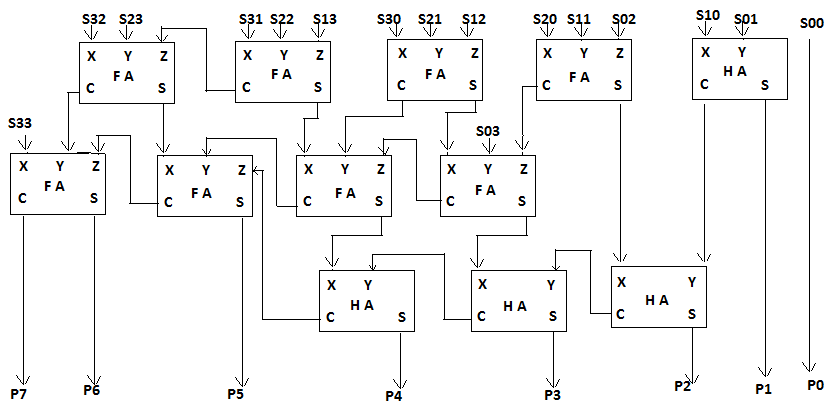


Fig 15: Circuit of Wallace multiplier 4x4 [5]

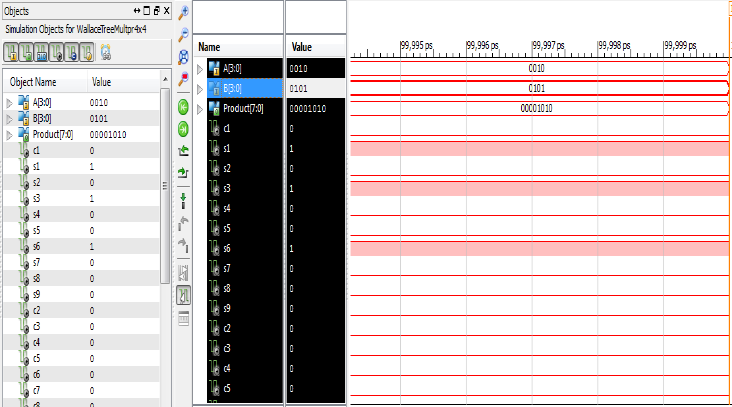


Fig 16: Wallace Tree 4x4 Xilinx implementation

***Wallace Tree Multiplier 8x8:*** The partial products are grouped in the same manner as explained for Wallace Tree multiplier 8x8. The summation of partial products for a 8x8 follows the Figure 17. The architectural implementation of the Wallace Tree structure of 8x8 is a little complicated. It works out costlier therefore, compared to conventional array multiplier, but nevertheless faster and more efficient. The structure for an 8x8 implementation is shown in Figure 19.

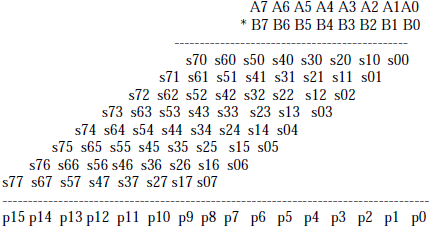
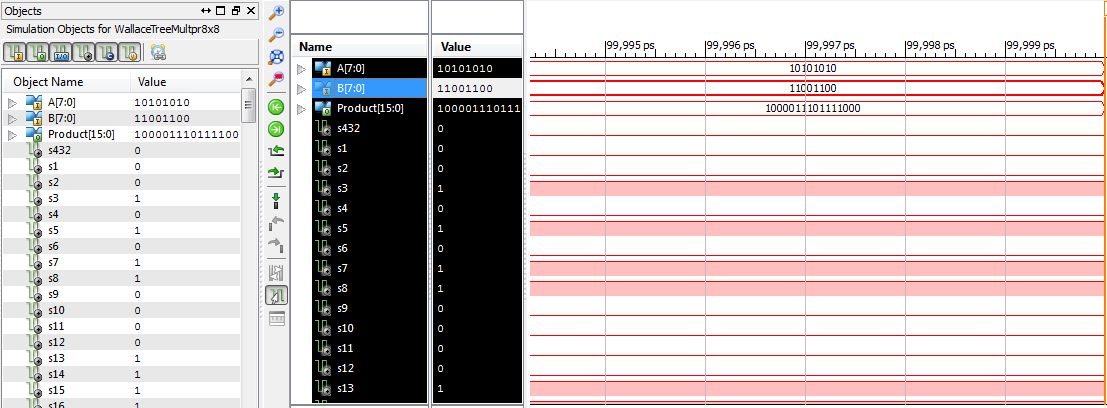


Fig 17: Partial Products for 8x8 Wallace design [5]

The circuit has been implemented in Xilinx and the output screenshot is as shown below in Figure 18.

Fig 18: Wallace Tree 8x8 Xilinx implementation

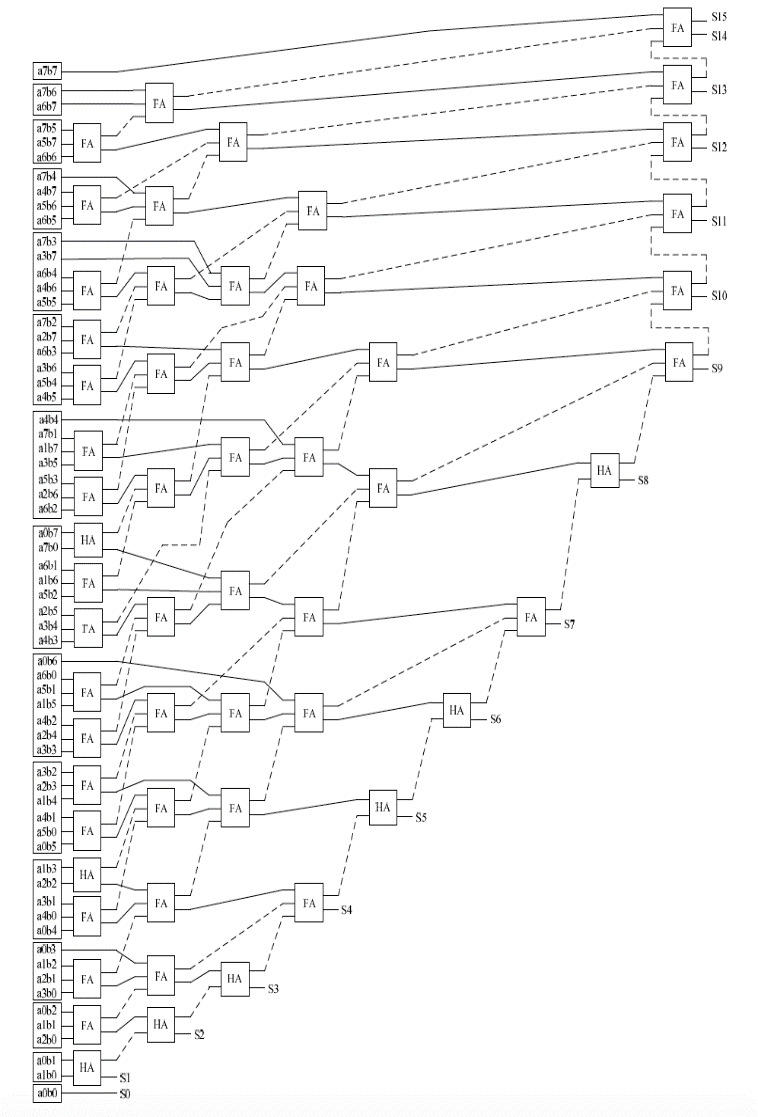


Fig19: Wallace Tree 8x8 multiplier [6]

Table IV: Delay (ns) comparison after the analysis

|  |  |  |  |
| --- | --- | --- | --- |
|  | **2x2** | **4x4** | **8x8** |
| **Vedic Algorithm** | 0.948 ns | 4.732 ns | 7.75 ns |
| **Booth Algorithm** | N.A | 3.295 ns | 6.83 ns |
| **Wallace Multiplier** | N.A | 3.163 ns | * 1. ns |

**V. CONCLUSIONS**

The implementations of the Vedic Algorithm, Modified Booth Algorithm, and Wallace Tree Multipliers can vary depending on the way the partial products were summed using adders. Summations herein have been implemented using Ripple Carry Adders, Carry Save Adders and Carry Look Ahead Adders. This paper also deals with methods of implementations that require the minimalistic use of hardware. All the circuit schematics have been implemented using Xilinx Vivado IDE. Table IV summarizes the conclusions derived from the various introduced implementations. The Booth Algorithm has been proven to be faster for an 8x8 multiplication while the Wallace Multiplier was a better choice for a 4x4 multiplication.

# **VI. REFERENCES**

[1] Jaina, D., Sethi, K., & Panda, R. “Vedic mathematics based multiply accumulate unit.” Computational Intelligence and Communication Networks (CICN), 2011 International Conference on 754-757. IEEE

[2] Anon,”Design and Implementation of 16 Bit Vedic Arithmetic Unit” [online] Available: http://verilogcode.blogspot.com/2014/01/design-and-implementation-of-16-bit.html.

[3] Nguyen Thi Hoang Lan, Computer Arithmetic"OpenStax CNX", Cnx.org, [Online]. Available: http://cnx.org/contents/ET2nPmjR@1/Computer-Arithmetic.

[4] Robbie D’Angelo, Scott Smith “Design of an 8x8 Booth Multiplier”. [Online]. Available: http://www.eecs.tufts.edu/~rjdang/index2.htm.

[5] Girma, T. “Designing and Synthesizing a Wallace Tree Multiplier for High Speed Performance”. [Online]. Available: http://ijaim.org/administrator/components/com\_jresearch/files/publications/IJAIM\_193\_Final.pdf.

[6] Li, X. “Implementation of a CMOS Wallace-tree Multiplier.” In Northeast ASEE (The American Society for Engineering Education) Conference, 2009.

[7] Panda, Siba Kumar, Ritisnigdha Das, and Tapasa Ranjan Sahoo. "VLSI Implementation of Vedic Multiplier Using Urdhva–Tiryakbhyam Sutra in VHDL Environment: A Novelty." In IOSR Journal of VLSI and Signal Processing (IOSR-JVSP), 2015, 17-24.

[8] Ramteke, S. K., Alok Dubey, and Yogeshwar Khandagare. "VLSI designing of low power radix4 booths multiplier." International Journal of Electrical, Electronics and Computer Systems 2.2 (2014), 48-52.

[9] Akhter, Shamim. "VHDL implementation of fast NxN multiplier based on vedic mathematic." In Circuit Theory and Design, 2007. ECCTD 2007. 18th European Conference on, 472-475. IEEE.

[10] Biswas, Barun, and Bidyut B Chaudhuri. "Generalization of Booth's Algorithm for Efficient Multiplication." Procedia Technology, 10/2013, 304-310.