International Journal of Electronics Communication and Computer Engineering
Volume 7, Issue 2, ISSN (Online): 2249–071X

# Computational Time, Latency, Throughput Improvement Using Tabulation Method Multiplier

**Vinay Patel**
Student, Department of ECE
Lakshmi Narain College of Technology
Bhopal (M.P), India
vinaypatel28290@gmail.com

**Vinod Pathak**
Asst. Prof. Department of ECE
Lakshmi Narain College of Technology
Bhopal(M.P), India
vpathak65@gmail.com

***Abstract* – Multiplier operation is depends on the speed of adder. Speed of adder is affected to its path propagation delay. Thus the multiplier is usually slowest and area consuming element in the system. Our designs offer tradeoffs between Computational time area, latency and throughput for performing multiplication. Our designs offer tradeoffs between Computational time area, latency and throughput for performing multiplication. A fast computation of multiplication operations of two finite length sequences implemented with the help of this propose algorithm is possible using VHDL language. A parallel architecture for multiplication using tabulation method is design for this purpose. Since it gives faster addition in multiplication process with less number of transistor. The carries of this adder are computed in parallel by two independent 4-bit carry chains which reduces the carry chain length. The circuit is propose for 8-bit adder module with significant operating speed improvement compared to the corresponding adders.**

***Keywords* – Area Delay Product (ADP), Redundant, Tabulation Method, Partial Product.**

## I. INTRODUCTION

Multiplication is the most time consuming process in various signal processing operations like Convolution, Circular Convolution, Cross Correlation, and auto-correlation, Image processing applications such as edge detection, microprocessors arithmetic and logical units etc. The performance of microprocessor is determine by performance of the multiplier. Multiplier operation is depends on the speed of adder. speed of adder is affected to its path propagation delay. Thus the multiplier is usually slowest and area consuming element in the system. Our designs offer tradeoffs between Computational time area, latency and throughput for performing multiplication. So to increase the speed of multiplier, it is require to improve the speed of addition. In this approach it is required to find the longest critical paths in the multi-bit adders and then shortening the path to reduce the total critical path delay. A bit-serial word-parallel (BSWP) architecture of RB multiplier is use to reduce the complexity of implementation and for high-speed realization. It is observed that the hardware utilization efficiency and throughput of existing structures can be improved by efficient design of algorithm and architecture [1]. A low power and low area array multiplier with using transmission gate with carry save adder bypasses the final addition stage of the multiplier as compare to the conventional parallel array multiplier [2].

## II. METHODOLOGY

To illustrate this methodology, let us consider the multiplication of different length of two decimal numbers. Here we show the implementation between input sequence length two to eight. In this technique, the numbers to be multiplied let us say bi and ai are written on the consecutive sides of the square table. On partitioning the square into rows and columns, each row/column belongs to one of the digit of the two numbers to be multiplied such that every digit of one number has a small square common to the digit of other number. These small squares are further divided into two equal parts by crosswise lines. Now the each digit of one number is multiplied with every digit of second number and two digit products are placed in their corresponding square.

Multiplicand = [a0, a1, a2, a3]
Multiplier = [b0, b1, b2, b3]

Let us consider the two finite input sequence is of length three as 9999X9999 the result is 99980001.

| | 9 (a0) | 9 (a1) | 9 (a2) | 9 (a3) | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 (b0) | 81 (b0a0) | 81 (b0a1) | 81 (b0a2) | 81 (b0a3) | | | | |
| 9( b1) | 81 (b1a0) | 81 (b1a1) | 81 (b1a2) | 81 (b1a3) | | | | |
| 9 (b2) | 81 (b2a0) | 81 (b2a1) | 81 (b2a2 ) | 81 (b2a3) | | | | |
| 9 (b3) | 81 (b3a0) | 81 (b3a1) | 81 (b3a2) | 81 (b3a3) | | | | |
| s0 | 81 (S00) | 162 (S01) | 243 (S02) | 324 (S03) | 243 (S04) | 162 (S05) | 81 (S06) | |
| s1 | 8 (S10) | 17 (S11) | 26 (S12 ) | 35 (S13) | 28 (S14) | 19( S15) | 10 (S16) | 1 (S17) |
| s2 | 9 (S20) | 9 (S21) | 9 (S22) | 7 (S23) | 9(S24) | 10 (S25) | 0 (S26) | 1 (S27) |
| s3 | 9 (S30) | 9 (S31) | 9 (S32) | 7 (S33) | 10(S34) | 0 (S35) | 0 (S36) | 1 (S37) |
| s4 | 9 (S40) | 9 (S41) | 9 (S42) | 8 (S43) | 0 (S44) | 0 (S45) | 0 (S46) | 1 (S47) |

Then their product is evaluated as shown

s00=b0a0
s01=b1a0+b0a1
s02=b2a0+b1a1+b0a2
s03=b3a0+b2a1+b1a2+b0a3
s04=b3a1+b2a2+b1a3
s05=b3a2+b2a3
s06=b3a3

As in the result of row s1, least significant bit of 81,62,43,24,43,62,81 of row s0 acts as the result bit and all the other bits act as carry. In row s2 least significant bit of 8,17,26,35,28,19,10,1of row s1 acts as the result bit and all the other bits act as carry. In row s3 least significant bit of 9,9,9,7,9,10,0,1 of row s2 acts as the result bit and all the other bits act as carry. In row s4 least significant bit of 9,9,9,7,10,0,0,1 of row s3 acts as the result bit and all the other bits act as carry. The final result of multiplication is shown in row s4.

For four bit number multiplication total five rows are evaluated in tabulation method.

At row So the product of input ai and bi is evaluate.

At row S1, the LSB bit of element of row S0 is consider and the its MSB bit is added to the next element of ROW S0.

At row S2, the LSB bit of element of row S1 is consider and the its MSB bit is added to the next element of ROW S1.

At row S3, the LSB bit of element of row S2 is consider and the its MSB bit is added to the next element of ROW S2.

At row S4, the LSB bit of element of row S3 is consider and the its MSB bit is added to the next element of ROW S3.

The final result of multiplication is shown in row s4.

## III. Modules in Row Computation of Tabulation Multiplier

The multiplier module are divided in smaller sub modules . Initially a partial product generation is done using AND logics. After that these partial products are added to form the resultant multiplication data. The multiplier module are divided in smaller sub modules. Initially a partial product generation is done using AND logics. After that these partial products are added to form the resultant multiplication data. The multiplier module are divided in smaller sub modules. Initially a partial product generation is done using AND logics. After that these partial products are added to form the resultant multiplication data. The process is shown in given block Diagram.

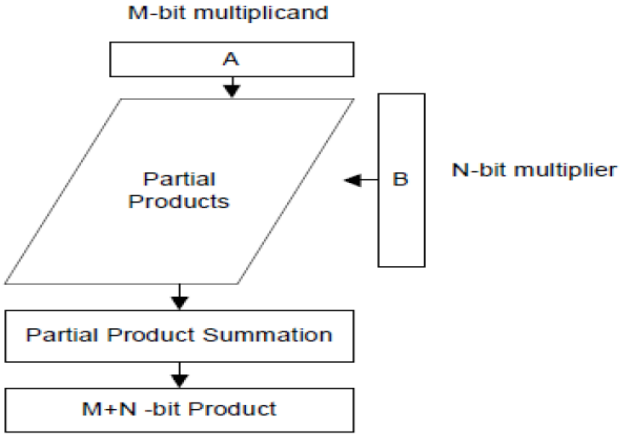


Fig. 1. Multiplier Block Diagram

This multiplier of two four bit multiplicand i.e a3a2a1a0 and b3b1b2b0 uses 16 AND logic gates to design partial product of b0a0, b0a1,b0a2, b0a3, b1a0, b1a1, ,b1a2, b1a3, b2a0,b2a1, b2a2, b2a3, b3a0, b3a1, b3a2 and b3a3. These partial products are added diagonally to generate the multiplier output P0 to P6.as:

c<= a(3) and b(3); d<= a(3) and b(2); e<= a(3) and b(1); f<= a(3) and b(0);

g<= a(2) and b(3); h<= a(2) and b(2); i<= a(2) and b(1); j<= a(2) and b(0);

k<= a(1) and b(3); l<= a(1) and b(2); m<= a(1) and b(1); n<= a(1) and b(0);

o<= a(0) and b(3); p<= a(0) and b(2); q<= a(0) and b(1); r<= a(0) and b(0);

These partial products are the added to for the result of first row of this methodology.

s1(0)<= g xor d; s1(1)<= g and d;

s2(0)<=k xor h xor e; s2(1)<= ((k and h) or (k and e) or (h and e));

s3(0)<=(o xor l xor i xor f); s3(1) <= ((o and l)and (not i)) OR ((l and f) and (not i)) OR ((i and f) and (not o)) OR ((i and l) and (not f))OR ((i and o) and (not l))OR ((o and f) and (not l));

s4(0)<= ((p xor m) xor j); s4(1)<= ((p and m) or (p and j) or (m and j));

s5(0)<= q xor n; s5(1)<= q and n; s6<=r;

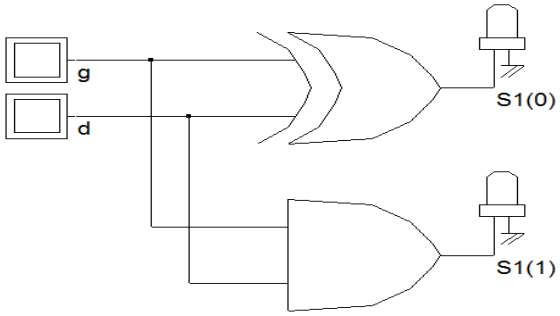


Fig. 2. Schematic Design of S1 element in First Row

Fig. 2 shows the addition of second array in tabulation table.

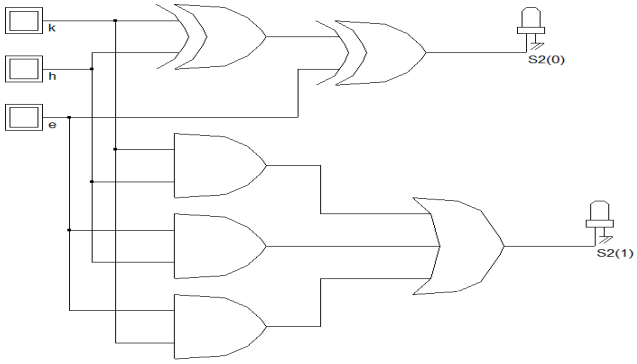


Fig. 3. Schematic Design of S2 element in First Row

Fig. 3 shows the addition of third array in tabulation table.

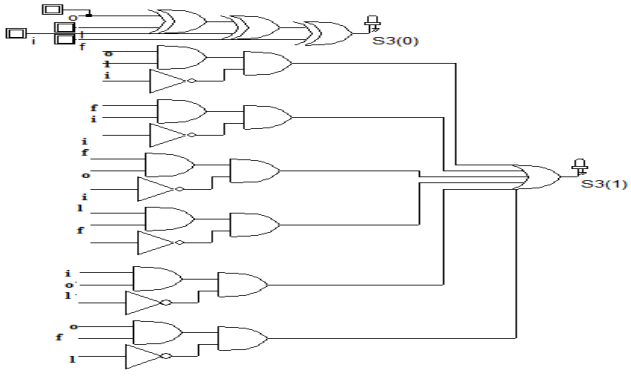


Fig. 4. Schematic Design of S3 element in First Row

Fig 4 shows the addition of fourth array in tabulation table.

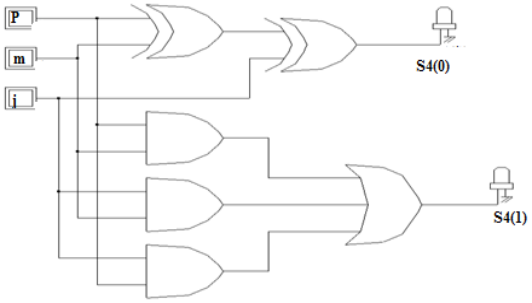


Fig. 5. Schematic Design of S4 element in First Row

Fig. 5. shows the addition of fifth array in tabulation table.

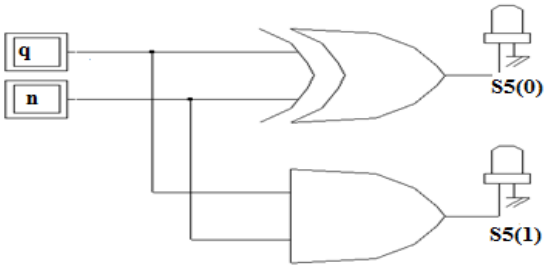


Fig. 6. Schematic Design of S5 element in First Row

Fig. 6. shows the addition of sixth array in tabulation table.

The multiplicants are of 4 bit each and the result generates is of 8 bits. Here both the multiplicand number is of 4 bits i.e. a3a2a1a0 and b3b2b1b0. The inputs are break to two bits numbers i.e. multiplier 1 for input bits (a1a0 & b1b0), multiplier 2 for input bits (a3a2 &b1b0), multiplier 3 for input bits (a1a0 &b3b2), and multiplier 4 for input bits (a3a2 & b3b2).

In step 1, the same two bit multiplication steps are followed. The multiplication of (a1a0 & b1b0) generates the four bit result in which the LSB and next bit to LSB generates the first two LSB bits of final result.

In step 2, the 4 bit result generated by each 2X2 multiplier is added to 4 bit parallel adder with two zeros are padded to LSB of first 2X2 multiplier 1 and last 2X2 multiplier 4. The result of both adder is of 5 bit.

In step 3, the adder tree is use which added the results of addition in step 2.

This hardware design is very similar to that of the famous array multiplier where an array of adders is required to arrive at the final product. All the partial products are calculated in parallel and the delay associated is mainly the time taken by the carry to propagate through the adders which form the multiplication array.

In the computation of second Row of tabulation method all the elements of first row is compare with binary 00,01,10, and 11. If these elements are equal to binary 00 or 01 the its binary value is retain to the LSB of next row. If these elements are equal to binary 10 or 11 the its binary value is reduce by 10 and a carry 01 is added to the LSB of next element.

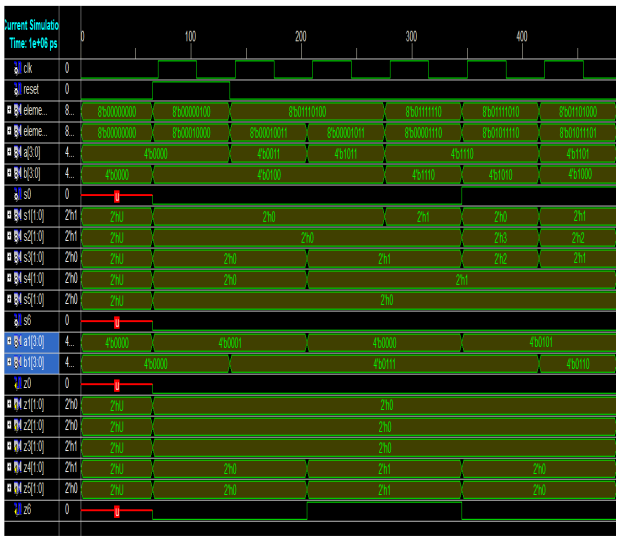

Fig. 7. Timing simulation for row 1 of tabulation method on timing scale of 1 to 500ns.

Fig. 7 shows the timing simulation for first row of tabulation algorithm on the timing scale of 1 to 500ns. The seven arrays multiplication bits are added to form the row S bits. The S1 element of row 1 is form by addition of {g.,d}. The S2 element of row 1 is form by addition of {k,h,e}. }. The S3 element of row 1 is form by addition of {o,l,I,f}. The S4 element of row 1 is form by addition of {p,m,j}. The S5 element of row 1 is form by addition of {q,n}.

Table 1: Row 1 of tabulation method for input a and b on timing scale of 1 to 500ns for S elements.

| Timing Scale | a | b | S0 | S1 | S2 | S3 | S4 | S5 | S6 |
|---|---|---|---|---|---|---|---|---|---|
| 100ns | 0011 | 0100 | 0 | 00 | 00 | 01 | 01 | 00 | 0 |
| 200ns | 1011 | 0100 | 0 | 01 | 00 | 01 | 01 | 00 | 0 |
| 300ns | 1110 | 1110 | 1 | 10 | 11 | 10 | 01 | 00 | 0 |
| 350ns | 1110 | 1010 | 1 | 01 | 10 | 01 | 01 | 00 | 0 |
| 450ns | 1101 | 1000 | 1 | 01 | 00 | 01 | 00 | 00 | 0 |

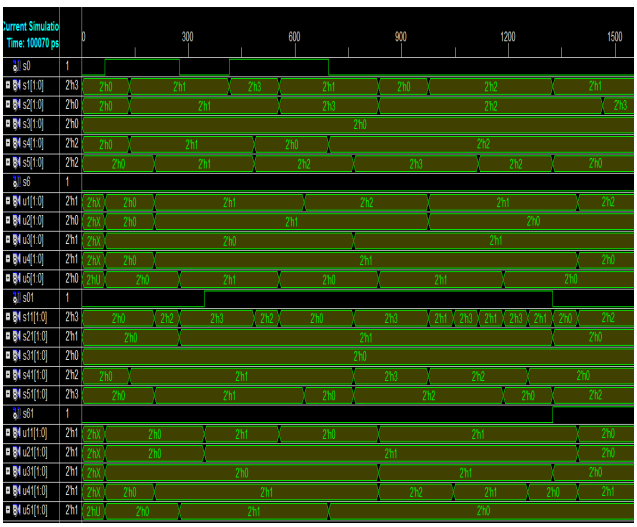

Fig. 8. Timing simulation for row 2 of tabulation method on timing scale of 0 to 1500ns.

Fig. 8 shows the timing simulation for row 2 of tabulation method on timing scale of 0 to 1500ns. The seven elements of row 1are added to form the row 2 element bits. The U1 element of row 2 is form by assigning the LSB bit of element S1and its MSB bit are added to next elements LSB of element S2. The U2 element of row 2 is form by assigning the LSB bit of element S2 and the MSB bit are added to next elements LSB of S3. By similar way all the elements of row 2 are computed.

Fig. 9. Timing Simulation for 32 bit tabulation Method of Multiplication

Fig. 9 shows the timing simulation for 32 bit tabulation Method of Multiplication. The timing delay of multiplication process is 5ns. The multiplication of two 32 bit number generates the result in 64 bit numbers.

Table 2 Comparative Analysis

| Design | P | Q | Area | Delay (ns) | ADP |
|---|---|---|---|---|---|
| This Work (Tabulation Method) | 32 | 63 | 2222 | 5 ns | 1.111 |
| | 16 | 31 | 586 | 5 ns | 0.293 |
| | 8 | 15 | 161 | 5 ns | 0.0805 |
| [1] Redundant Basis | 32 | 9 | 2966 | 0.142 ms | 0.42 |
| | 16 | 17 | 2959 | 0.297 ms | 0.88 |
| | 8 | 34 | 2421 | 0.669 ms | 1.62 |

Where, Q is parallel arrays.

Each array consist of : P number of bits and M multiplication nodes.

## IV. CONCLUSION

In this propose work, the different type multiplication operation use in convolution, circular convolution, cross correlation, auto correlation is possible. A fast computation of multiplication operations of two finite length sequences implemented with the help of this propose algorithm is possible using VHDL language. By using the propose flow diagram a high-throughput digit-serial RB multipliers are derived using hardware description language (VHDL) to achieve significantly less gate counts than the existing ones.

## REFERENCES

[1] Jiafeng Xie, Pramod Kumar Meher, and Zhi-Hong Mao "High-Throughput Finite Field Multipliers Using Redundant Basis for FPGA and ASIC Implementations " IEEE Transactions On Circuits And Systems—I: Regular Papers, Vol. 62, No. 1, January 2015 pp no 110.

[2] Gustavo D. Sutter, Jean-Pierre Deschamps, and José Luis Imaña "Efficient Elliptic Curve Point Multiplication Using Digit-Serial Binary Field Operations" IEEE Transactions On Industrial Electronics, Vol. 60, No. 1, January 2013 pp no. 217.

[3] Diptendu Kumar Kundu1, Supriyo Srimani2, Saradindu Panda3, Prof. Bansibadan Maji4 " Implementation of Optimized High Performance 4x4 Multiplier using Ancient Vedic Sutra in 45 nm Technology" IEEE Second International Conference on Devices, Circuits and Systems (ICDCS) 2014.

[4] Aravind E Vijayan, Arlene John, Deepak Sen "Efficient Implementation of 8-bit Vedic Multipliers for Image Processing Application" IEEE International Conference on Contemporary Computing and Informatics (IC3I) 2014 pp no. 544-550.

[5] Alex Pappachen James, Dinesh S. Kumar, and Arun Ajayan "Threshold Logic Computing: Memristive-CMOS Circuits for Fast Fourier Transform and Vedic Multiplication" IEEE Transactions On Very Large Scale Integration (VLSI) Systems Jan 2015.

[6] P. E. Allen and D. R. Holberg, CMOS Analog Circuit Design. New York, NY, USA: Oxford Univ. Press, 2011.