Study of 8 Bits Fast Multipliers for Low Power Applications

Vasudeva G, Cyril Prasanna Raj P

Abstract- High–speed multiplication has always been a fundamental requirement of high performance processors and systems. With MOS scaling and technological advances there is a need for design and development of high speed data path operators such as adders and multipliers to perform signal processing operations at very high speed supporting higher data rates. In Digital signal Processing applications, multiplication is one of the most utilized arithmetic operations as part of filters, convolves and transforms processors. It is found in the literature that improving multipliers design directly benefits the high performance embedded processors used in consumer and industrial electronic products. Also significant increase in the bit length increases the critical path affecting the frequency of operations. It is also found that the regular structure required for each processing elements also increases and hence consumes area and power. Hence there is a need for design and development of high-speed architectures for N-bit multipliers supporting high speed and power. In this paper we review the architecture reported in the literature for multipliers and critical issues degrading the speed and power. Based on the literature review suitable modifications are suggested in the design for high speed and low power multipliers. The multipliers Booth, Wallace tree and Dadda are implemented and the constraints Area, Power and Timing are optimized using software resources NC SIM and VC SIM.

Keywords: DSP, microprocessor, NC SIM, VC SIM

I. INTRODUCTION

Multiplication is a less common operation than addition, but is still essential for microprocessors, digital signal processors and graphics engines. Multiplication algorithms will be used to illustrate methods of designing different cells so that they fit into a large structure. The most basic form of multiplication consists of forming the product of two unsigned binary numbers, simplified to base 2. M x N -bit multiplication can be viewed as forming N partial products of M bits each, and then summing the appropriately shifted partial products to produce on M + N -bit result P. Binary multiplication is equivalent to a logical AND operation. Therefore, generating partial product consists of logical ANDing of the appropriate bits of the multiplier and multiplicand. Each column of partial products must then be added and if necessary, any carry values passed to the next columns. In the 1960's two classes of parallel multipliers were defined. The first class [6] of parallel multipliers uses a rectangular array of identical combinational cells to generate and sum the partial product bits. Multipliers of this type are called array multipliers. They have a delay that is generally proportional to the word length of the multiplier input. Due to the regularity of their structures, array multipliers are carrying to layout and have been implemented frequently. The second class of parallel multipliers reduces a matrix of partial product bits to two words through the strategic application of counters or compressors. These two words are then summed using a fast carry-propagate adder to generate the product. This class of parallel multiplier is known as column compression multiplier. Since the delay is proportional to the logarithm of the multiplier, word length, these are also the fastest multipliers. In array multiplier, the two basic functions of partial product generation and summation are combined. For unsigned N X N multiplication, N2+N-1 cells, where N2 contain an AND gate for partial product generation and a full adder for summing and N-1 cells containing a full adder, are connected to produce a multiplier. The array generates N lower product bits directly and uses a carry-propagate adder, in this case a ripple carry adder, to form the upper N bits of the product. Column compression multiplier continued to be studied due to their high speed performance. With total delays that are proportional to the logarithm of the operand word length. Column compression multipliers are faster than array multipliers whose delay grows linearly with operand word length. According to Thomas Ko Callaway et al. [11] column compression multipliers are more power efficient than array multipliers. In 1964, Wallace [12] introduced a scheme for fast multiplication based on summing the partial product bits on parallel using a tree of carry save adders which became generally known as the Wallace tree. Dadda [13] later refined Wallace's method by defining a counter placement strategy that required fewer counters in the partial product reduction stage at the cost of a larger carry-propagate adder. For both methods, the total delay is proportional to the logarithm of the operand word-length. Other partial product reduction methods have been proposed since the work of Wallace and Dadda. The reduced area [7] and the Windsor methods are based on strategic utilization of (3, 2) and (2, 2) counters to improve area and layout, while maintaining the fast speed of the Wallace and Dadda designs. In this paper we identify techniques for optimal computer aided designs of column compression multipliers by analyzing area, power and timing characteristics with particular emphasis on low power.

II. DESIGN AND ANALYSIS

The major works in this paper are study of multiplier architectures for high speed signal processing applications, identifying the specifications for the multiplier design, modeling the architecture, functional verification, and developing the test bench to verify the design for all possible input combinations. We also do FPGA implementation of the proposed multiplier to meet the

Manuscript Received on February 16, 2015.

Vasudeva G, Asst. Prof., Department of ECE, Rashtreeya Vidyalaya College of Engineering, Bangalore-56, India.

Dr. Cyril Prasanna Raj P, M.S. Engineering College, Bangalore, India.

specification identified. Synopsys tool flow is used for ASIC synthesis, physical design and implementation of multipliers. GDSII Generated and Report prepared. Following are the technical specification of experimental work carried out to Design and implement Booth, Wallace, Dadda Multipliers using 130nm technology.

Input bit width: 8-bit, signed, unsigned, Integer, Decimal
Input arrival: Parallel with 100 Mbytes / sec.
Expected output: 16-bit output, supporting all formats.
Output data rate: 100 Mbytes /sec.
Tech: 130 nm, Lib: TSMC
Power: $\leq 10\mu$ watts
Area: 400 sq. mm

The power analysis is the process of calculating the power consumption of the chip. It also consists of the calculation of voltage, current drop (IR drop) and electro migration analysis due to high current density of the metal. Table 1 gives the power consumption of the multipliers.

Table 1. Comparison of Area, power consumption and Timing

8 bit	Booth Multiplier	Wallace tree	Dadda Multiplier
Area (μm)	5115.963379	1330.7615	1330.761597
Power(μw)	324.5302	655.5517	655.8073
Timings (ns)	3.75	1.56	1.56

Area wise, Dadda multiplier consumes less area as compared to Wallace tree and Booth Multiplier. Power wise, Booth multiplier consumes less power as compared to Wallace tree and Dadda multiplier. And delay wise, Wallace tree has less delay as compared to Booth and Dadda multiplier. Booth multiplier has maximum number of ROMs, Macros and BELS. Wallace tree has minimum number of BELS and Macros compared to Booth and Dadda multiplier. Also Wallace tree multiplier and Dadda multiplier have no flip-flops and Booth multiplier has maximum Flip-flops. The multipliers have been synthesized setting a constraint on speed to a maximum of 130MHz. Based on this constraint the Table 2 depicts the synthesis result. The results clearly indicate the performance of the multipliers that are compared in the graphs.

Table 2. Design compiler output

8 bit	Booth Multiplier	Wallace tree Multiplier	Dadda Multiplier
Frequency	130MHZ	130MHZ	130MHZ
Number of ports	38	34	32
Number of Nets	265	302	348
Number of Cells	214	248	266
References	35	5	5
Combinational Area (μm)	1868.8373	1470.7612	1330.7615
Sequential Area (μm)	3247.1262	0.0000	0.0000
Total Cell Area (μm)	5115.96337	1470.7672	1330.7615
Cell internal Power (μw)	303.1691	501.8940	562.0927
Net Switching power (μw)	21.3691	153.6577	153.7151
Total Dynamic Power (μw)	324.5302	655.5517	655.8073
Cell Leakage Power (μw)	5.6910	5.6263	5.6258

We denote the multiplicand as

$$Y = (y_{M-1}, y_{M-2}. . . y_1, y_0) \quad (1)$$

and multiplier as

$$X = (x_{N-1}, x_{N-2}. . . x_1, x_0) \quad (2)$$

For unsigned multiplication, the product is given in (3).

$$P = (\sum_{j=0}^{M-1} y_j 2^j)(\sum_{i=0}^{N-1} x_i 2^i) = \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_i y_j 2^{i+j} \quad (3)$$

There are a number of techniques that can be used to perform multiplication. In general, the choice is based up on factors such as latency, throughput, area, and design complexity. An obvious approach is to use an M+1 – bit carry propagate adder (CPA) to add the first two partial products, then another CPA to add the third partial product to the running sum, and so forth. Such an approach requires N-1 CPAs and is slow, even if a fast CPA is employed. More efficient parallel approaches use some sort of array or tree of full adders to sum the partial products. In the early 1950's, multiplier performance was significantly improved with the introduction of Booth multiplier [4] and the development of faster adders [5] and memory components. Booth's method and the modified Booth's method do not require a correction of the product when either (or both) of the operands is negative for two's complement numbers. During the 1950's, adders designs moved away from the slow sequential formation of carried executed by ripple carry adders carry look ahead, carry select, and conditional sum adders yielded speedy sums through the faster simultaneous or parallel generation of carriers. In the 1960's two classes of parallel multipliers were defined .The first class [6] of parallel multipliers uses a rectangular array of identical combinational cells to generate and sum the partial product bits. Multipliers of this type are called array multipliers. They have a delay that is generally proportional to the word length of the multiplier input. Due to the regularity of their structures, array multipliers are carrying to layout and have been implemented frequently. The second class of parallel multipliers reduces a matrix of partial product bits to two words through the strategic application of counters or compressors. These two words are then summed using a fast carry-propagate adder to generate the product. This class of parallel multiplier is known as column compression multiplier. These are also the fastest multipliers because of delay is proportional to the logarithm of the multiplier and word length.

III. ARRAY MULTIPLIER

In array multiplier, the two basic functions, partial product generation and summation are combined. For unsigned N by N multiplication, N2+N-1 cells are connected to produce a multiplier, where N2 contain an AND gate for partial product generation, a full adder for summing and N-1 cells containing a full adder. The array generates N lower product bits directly and uses a carry-propagate adder, in this case a ripple carry adder, to form the upper N bits of the product. Replacing full adder with half adders, possibly reduces the complexity to N2 AND gates, N half adders, and N(N-2) full adders as shown in Fig. 1.

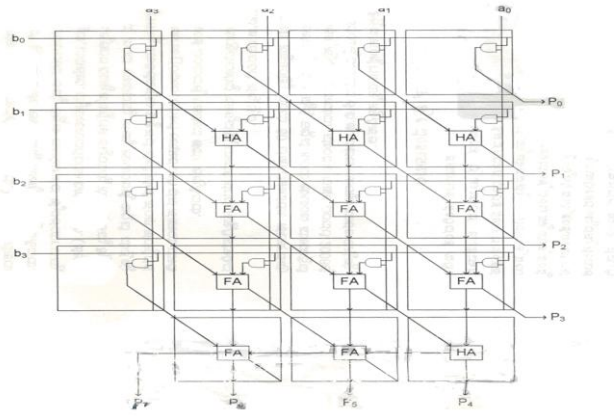


Fig. 1. 4x4 Square array multiplier

This 4x4 multiplier is shown as a square array with modifications to the first two rows. Since the carry-in bits and the previous partial products bits are zero for the first row and the left column, only the AND gates are needed. With only two switching inputs, the second row employs half adders instead of full adders. The worst case delay is $(2N-2)\Delta_c$, where Δ_c is the adder delay.

In order to design an array multiplier for two's complement operands, Booth algorithm [8] can be employed. The Booth's algorithm array multiplier computes the partial products by examining two multiplicand bits at a time. Except for enabling usage of two's complement operands, this Booth's algorithm array multiplier offers no performance or area advantage in comparison to the basic array multiplier. Better delays, though can be achieved by implementing a higher radix modified Booth algorithm. Another method for building an array multiplier that handles two's complement operands was presented by Baugh et al. [9] as shown in Fig. 2. This method increases the maximum column height by two. This may lead to an additional stage of partial product reduction, thereby increasing overall delays. A modified form of the Baugh et al. strategy is more commonly used because it does not increase the maximum column height.

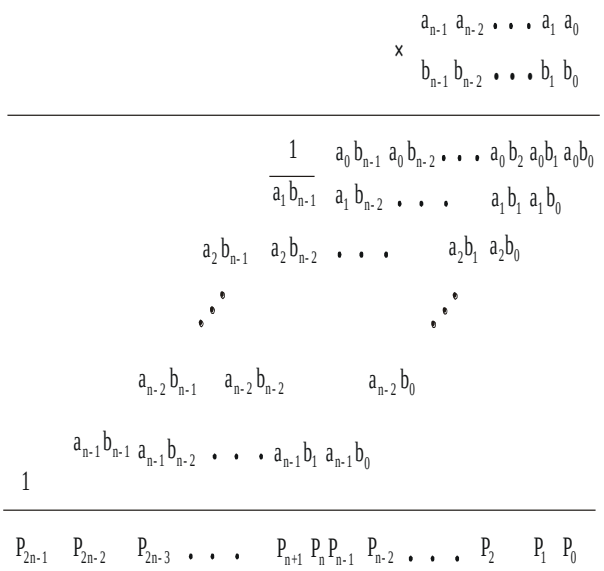


Fig. 2. Two's complement by modified Baugh-Wooley method

IV. COLUMN COMPRESSION MULTIPLIER

Column compression multiplier continued to be studied due to their high speed performance. With total delays that are proportional to the logarithm of the operand word length, where as other array multipliers delay grows linearly with operand word length. According to Thomas Ko Callaway et al. [11] column compression multipliers are more power efficient than array multipliers. In 1964, Wallace [12] introduced a scheme for fast multiplication based on summing the partial product bits on parallel using a tree of carry save adders which became generally known as the Wallace tree. Dadda [13] later refined Wallace's method by defining a counter placement strategy that required fewer counters in the partial product reduction stage at the cost of a larger carry-propagate adder. For both methods, the total delay is proportional to the logarithm of the operand word-length. Other partial product reduction methods have been proposed since the work of Wallace and Dadda. The reduced area [7] and the Windsor methods are based on strategic utilization of (3, 2) and (2, 2) counters to improve area and layout, while maintaining the fast speed of the Wallace and Dadda designs. This research identifies techniques for optimal computer aided designs of column compression multipliers by analyzing area, power and timing characteristics with particular emphasis on low power.

V. TOOLS

5.1 Design Compiler and Design Vision

The Synopsys Design compiler (DC) and Design Vision (DV) comprise a powerful suite of logic synthesis products, designed to provide an optimal gate-level synthesized net list based on the design specifications, and timing constraints.

5.2 Primetime (Static Timing Analysis tool)

Primetime (PT) is the Synopsys sign-off quality, full chip, and gate level static timing analysis tool. It allows comprehensive modeling capabilities often required by large designs. Primetime is faster compared to design compilers internal static timing analysis engine. It provides enhanced analysis capabilities to other Synopsys tools, this tool is based on TCL language, thus providing powerful features of that language to promote the analysis and debugging of the design.

5.3 Standard Delay format (SDF) generation

5.3.1 SDF file

The SDF file is used to perform exhaustively throughout the ASIC world to perform dynamic timing simulations. It contains timing information of each cell in the design. The basic timing data comprises of the following.

- IOPATH delay- specifies the cell delay.
- INTERCONNECT delay- specifies point to point delay.
- SETUP timing check- contains the required setup of each sequential cell.
- HOLD timing check- hold time of each sequential cell.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

5.3.2 SDF file generation

The SDF file may be generated for pre-layout or post-layout simulations. The post-layout SDF is generated from DC or PT, after back annotating the extracted RC delay values and parasitic capacitances, to DC or PT. The post-layout values represent the actual delays associated with the design. The pre-layout numbers contain delay values that are based upon the wire-load models. It does not contain the clock tree. Therefore it is necessary to approximate the post-route clock trees delays while generating the pre-layout SDF. The post-layout design contains the clock tree information. Therefore all the steps that were needed to fix the clock latency, skew and clock transition time, during pre-layout phase are not required for post-layout SDF file generation.

5.4 DESIGN FOR TEST (DFT)

The design-for-test or DFT techniques are increasingly gaining momentum among ASIC designers. These techniques provide measures to test the manufactured device for quality and coverage.

Types of DFT

The main DFT techniques that are currently in use are

1) Scan insertion
2) Memory BIST insertion
3) Logic BIST insertion
4) Boundary scan insertion

Scan insertion is one of the most widespread DFT techniques used by design engineers to test the chip for defects such as stuck-at faults. The scan insertion technique involves replacing all the flip-flops in the design with special flops that contain built-in logic, solely for testability. The most commonly used architecture is the multiplexed flip-flop. Scan can also be used to test the DUT for any possible timing violations. The memory BIST is comprised of controller logic that uses various algorithms to generate input patterns that are used to exercise the memory elements of a design. The BIST logic is automatically generated based upon the size and configuration of the memory element. It is in the form of synthesizable verilog or VHDL which is inserted in the RTL source with hookups, leading to the memory elements. Boundary scan is used for testing the board connections, without unplugging the chip from the board.

5.5 Synopsys Technology Library and Delay Calculation

5.5.1 Wire load models

The physical library is a text file and is compiled by LC to generate a binary format with a "pdb" extension. Synopsys have provided a useful utility called "lef2pdb" that takes the standard library Exchange Format(LEF) file and the process technology file as input and converts it to the "pdb" format .The **wire_load** group contains information that DC utilizes to estimate interconnect wiring delays during the pre layout phase of the design. Usually several models appropriate to different sizes of the logic are included in the technology library. These models define the **capacitance**, **resistance** and **area** factors. Also the **wire_load** group specifies slope and **fanout_length** for the logic under consideration.

The **capacitance, resistance** and **area** factors represent the wire resistance; the capacitance and area respectively per unit length of interconnect wire.

The **fanout_length** attribute specifies values for the length of the wire associated with the number of fan outs. This may also contain values for other parameters such as **average_capacitance**, **standard_deviation** and **number_of_nets**.

VI. REVIEW OF MULTIPLIERS

6.1 Description of Modified Booth's Multiplier

Booth's algorithm is a powerful direct algorithm to perform signed number multiplication. It involves repeatedly adding one of two predetermined values A and S to a Product P, then performing a rightward arithmetic shift on P. Let **x** and **y** be the multiplicand and multiplier respectively. Let x and y represent the number of bits in **x** and **y**.

1) Determine the values of A, S and the initial value of P. All of these numbers should have a length equal to x+y+1.
- A: Fill the most significant (leftmost) bits with the value of **x**. Fill the remaining (y+1) bits with zeros.
- S: Fill the most significant bits with the value of (-**x**) in two's complement notation. Fill the remaining (y+1) bits with zeros.
- P: Fill the most significant x bits with zeros. To the right of this append the value of **y**. Fill the least significant (rightmost) bits with a zero

2) Determine the two least significant (rightmost) bits of P.
- If they are 01, find the value of P+A, ignore any overflow.
- If they are 10, find the value of P+S, ignore any overflow.
- If they are 00 or 11, do nothing, use P directly in the next step

3) Arithmetically shift the value obtained in the previous step by a single place to the right. Let P now equal to this new value.

4) Repeat steps 2 and 3 until they have been done y times.

5) Drop the least significant (rightmost) bit from P, resultant is the product of x and y.

The following example 1 demonstrates the Booth's multiplier algorithm

Example1: Find 3* -4 with x=3 and y=4.

Solution: step 1 A= 0011 0000 0
S= 1101 0000 0
P= 0000 1100 0
Perform the loop 4 times

Step 2: P= 0000 110**0 0**.The last two bits are 00.
P= 0000 0110 0.Arithmetic right shift.

Step 3: P= 0000 011**0 0**.The last two bits are 00.
P= 0000 0011 0.Arithmetic right shift

Step 4: P=0000 001**10**. The last two bits are 10
P= 1101 00110. P=P+S
P= 1110 10011. Arithmetic right shift

Step 5: P= 1110 100**11**. The last two bits are 11
P=1111 0100. Arithmetic right shift

The product is 1111 0100 which is -12.

6.2 Wallace tree Multiplier

In 1964 C.S.Wallace introduced a scheme for the multiplication based on

summing the partial product bits in parallel using a tree of carry save adders which became generally known as the Wallace tree. This method has a three step process is used to multiply two numbers.

Step 1: The bit products are formed

Step 2: The bit product matrix is reduced to a two row matrix by using carry save adders known as Wallace tree.

Step 3: The remaining two rows are summed using a fast carry –propagate adder to produce the product.

Though the process seems to be complex it yields multipliers with delay proportional to the logarithm of the operand size n. The Wallace tree multiplier belongs to a family of multipliers called column compression multipliers. The principle in this family of multipliers is to achieve partial product accumulated by successively reducing the number of bits of information in each column using full adders or half adders. The full adder is known as (3:2) compressor because of its ability to add three bits from a single column of the partial product matrix and output two bits, one bit in the same column and one bit in the next column of the output matrix. The half adder is known as (2:2) compressor because of its ability to take two bits from a single column of the partial product matrix and output two bits, one bit in the next column of the output matrix. Fig. 3 gives dot diagram of Wallace tree multiplier.

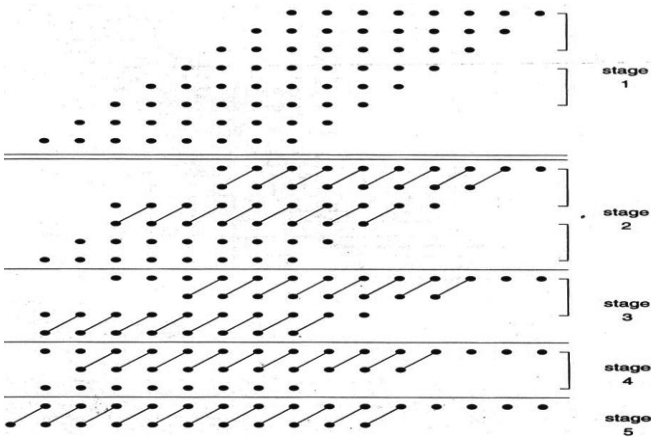


Fig. 3. Dot diagram of Wallace tree multiplier

Example2: 128_{10} x 128_{10} (using Wallace tree multiplication)

```
10000000 x 10000000

                0 0 0 0 0 0 0 0
                0 0 0 0 0 0 0 0        step1
              0 0 0 0 0 0 0 0
            0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
      1 0 0 0 0 0 0 0
  ---------------------------------
          0 0 0 0 0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0        step2
        0 0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0
    1 0 0 0 0 0 0 0
  ---------------------------------
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
                0 0 0 0 0 0 0 0        step3
      1 0 0 0 0 0 0 0 0
  ---------------------------------
        0 0 0 0 0 0 0 0

    1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    step4
  0 0 0 0 0 0 0 0 0 0 0 0
  ---------------------------------
  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0(16384)10
```

The Wallace tree consists of numerous levels of such column compressor structures until finally only two full width operands remain. These two operands can then be added using regular 2N-bit adders to obtain the product result. The difference between the Wallace tree multiplier from column compression multiplier is that, in the Wallace tree every possible bit in every column is covered by the (3:2) or (2:2) Compressors respectively. Until finally the partial product matrix has a depth of only two. Thus the Wallace tree multiplier uses as much hardware as possible to compress the partial product matrix as quickly as possible into the final product.

6.3 Dadda Multiplier

Dadda refined Wallace's method by defining a counter placement strategy that required fewer counters in the partial product reduction stage at the cost of a larger carry propagate adder. Dadda has introduced a number of ways to compress the partial product bits using such a counter which later became known as Dadda's Counter. This process is shown for an 8 by 8 Dadda multiplier in Fig. 4.

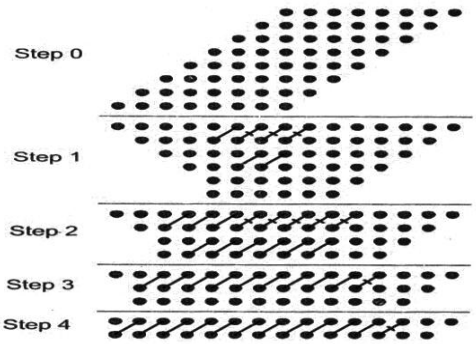


Fig. 4. Operation 8X8 bits Dadda Multiplier

An input 8 by 8 matrix of dots (each dot represents a bit product) is shown as matrix 0.Columns having more than six dots are reduced by the use of half adders, each half adder takes in two dots and outputs one in the same column and one in the next more significant column and full adders, each full adder takes in three dots and outputs one in the same column and one in the next more significant column so that no column in matrix 1 will have more than six dots. Half adders are shown by a crossed line in the succeeding matrix and full adders are shown by a line in the succeeding matrix. In each case the rightmost dot of the pair that is connected by a line is in the column from which the inputs were taken from the adder. In the succeeding steps reduction to matrix two with no more than four dots per column, matrix three with no more than three dots per column, and finally matrix four with no more than two dots per column is performed. The height of the matrices is determined by working back from the final two row matrix and limiting the

height of the each matrix to the largest integer that is no more than 1.5 times the height of its successor. Each matrix is produced from its predecessor in one adder delay. Since the number of bits in the words to be multiplied, the delay of the matrix reduction process that reduces is proportional to log (n).Since the adder that reduces the final two row matrix can be implemented as a carry look ahead adder which also has logarithmic delay, the total delay for this multiplier is proportional to the logarithm of the word size proportional to the logarithm of the word size.

VII. COMPARISON BETWEEN 8X8-BIT DADDA AND WALLACE TREE MULTIPLIERS

1. Wallace tree multiplier uses 38 full adders and 15 half adders.
2. Dadda multiplier uses 35 full adders and 7 half adders.
3. Wallace tree multiplier requires a carry-propagate adder of 10 bits wide
4. Dadda multiplier requires a carry propagate adder of 14 bits wide.
5. The other disadvantage of Dadda multiplier is that it is less regular than the Wallace tree multiplier, making it more difficult to layout in VLSI design.

Complexities of Multipliers:

The multiplier being one of the major complex arithmetic building blocks for VLSI design has its own sets of complexities in terms of area, power, speed, and cost and design methodology. The tables 2.12 and 2.12(a) below shows the complexity involved in multiplier design. With bit width being increased, the number of stages also increases, and this introduces complexity. Wide bit width is required for accuracy and high sampling rate. Hence there is a need for design and development of an IP that can be easily adopted for any high speed applications by just using the basic building block of the multiplier design.

Table 3. Number of stages in multipliers

Bitwidth of Multiplier (N)	Number of Stages (S)
2	0
3	1
4	2
5 to 6	3
7 to 9	4
10 to 13	5
14 to 19	6
20 to 28	7
29 to 42	8
43 to 63	9
64 to 94	10

Table 4. Comparison of Different Multipliers with area, speed and power

Multipliers	Delay (ns)	Area (μm^2)	Power (mw)
Dadda using carry look ahead adder	2.56	878	5.65
Dadda using Ripple carry adder	2.73	853	5.23
Array 8	3.02	979.7	5.16
Wallace 8	2.81	910	5.39

Fig. Comparison of Different Multipliers with area, speed and power

VIII. COMPARISON OF FPGA AND ASIC

Multiplier	**Booth**	**Wallace tree**	**Dadda**
No. of ports	38	34	32

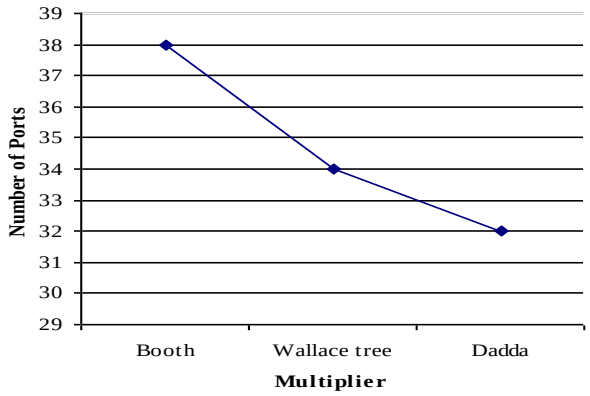


Fig. 5. Graphical representation of design compiler output with frequency 130 MHz of Multipliers verses No. of ports.

Table 5. Comparison of ASIC and FPGA

Sl No.	FPGA	ASIC
1	Complexity of multipliers is more	Complexity of multipliers is less
2	Area occupied by the multipliers is less.	Area occupied by the multipliers is more.
3	Power consumption by the multipliers is more.	Power consumption by the multipliers is less.
4	Delay is more and hence speed is less.	Delay is less and hence speed is high.

Hence ASIC physical design is preferred compared to FPGA.

IX. APPLICATIONS, CONCLUSION AND FUTURE WORK

9.1 Applications

1. High Speed Signal Processing that includes DSP based applications.
2. Used in DWT and DCT transforms used for image and wide signal processing.
3. Used in FIR and IIR Filters for high speed, low power filtering applications.

Used in Multirate signal processing applications such as digital down converters and up converters.

9.2 Conclusion

In this project we have identified the techniques for optimal computer aided designs of selected three 8-bit multipliers namely Booth, Wallace tree and Dadda by analyzing delay, area and power characteristics with particular emphasis on designing the cells for optimum power using layout design techniques. The three multipliers Booth, Wallace and Dadda are implemented and the constraints area, power and timing are optimized using Verilog codes based on software resources NC SIM and VC SIM.

The main results of the project are

1. After carrying out the literature review on the existing high speed serial and parallel multipliers available, identified the specification requirements for the multipliers.
2. Modeled the multipliers using HDL and verified the functionality using test vectors.
3. Implemented the design on FPGA and verified its functionality and identified the hardware requirements.
4. Carried out ASIC design on the synthesized net list by appropriately providing the constraints based on the first cut information obtained from FPGA synthesis.
5. Compared the performance of multiplier design and optimized the design for area, speed and power.

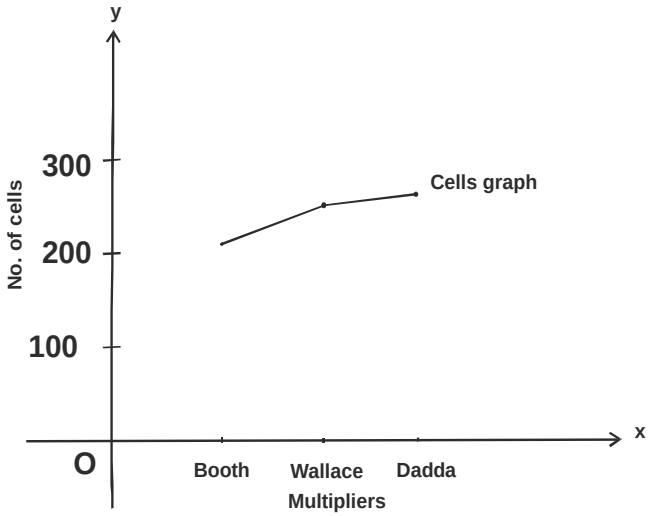
So the final conclusion of this project is performance wise, Dadda multiplier consumes less area as compared to Wallace tree and Booth multiplier. Power wise Booth multiplier consumes less power compared to Wallace tree and Dadda multiplier. Delay wise Wallace tree has less delay as compared to Booth and Dadda multiplier.

From the graph we observed that the Dadda Multipliers requires more nets and consumes lesser references than Wallace tree multiplier and Booth multiplier. The graphs indicate that, as the number of intermediate stages increases in multipliers, the interconnection between the building blocks also increases. As Dadda multiplier has more number of intermediate stages it has more number of interconnections.

$\frac{265}{348} \times 100 = 76.5\%$ Increase in the number of interconnections.

3.3. Graph of multipliers verses no. of cells

Multiplier	Booth	Wallace tree	Dadda
No. of cells	214	248	266



9.3 Scope for Future work

As this project was limited to design of only the multiplier as an IP using TSMC 130nm CMOS technology, it would be better to incorporate the Multiplier into a MAC unit that can perform multiplication and accumulation. As MAC forms the major block for any filtering application. During the design of MAC block, redundancy in filter coefficients can be exploited to minimize the filter structure and optimize the performances of MAC unit. Also, there is possibility in developing a hybrid multiplier that takes into consideration both Dadda and Wallace multiplier architecture combined with booths multiplier.

ACKNOWLEDGMENT

I express my deep sense of gratitude to my project guide, Mr. Cyril Prasanna Raj P, Assistant Professor, Course Manager, VSD, MSRSAS. His willingness to teach and unfailing patience have been a source of great motivation for me to excel in my work. Without his guidance and invaluable time spent with me in this project, this thesis would not have been completed successfully. I express my sincere thanks to my internal project guide, Ms.B G Shivaleelavati, Assistant Professor, Department of Electronics and Communication Engineering, JSSATE, Bangalore for her continuous encouragement and suggestions at every stage of this work.. Finally I offer my sincere pranamas to my parents and sisters for their blessings in all my intellectual pursuits.

REFERENCES

1. Keshab K.Parhi, "VLSI DIGITAL SIGNAL PROCESSING SYSTEMS", Design and implementation John Wiley and sons (ASIA), 1999.
2. Neil H.E.Weste, David Harris, Ayan Banerjee, "CMOS VLSI DESIGN", A circuits and systems perspective. Pearson Education, Third edition 2007, pp345-356.
3. Robert F.Shaw, "Arithmetic operations in a binary computer", Review of scientific instruments; vol 21, pp 687-693, 1950.
4. O.L.Mac Sorley,"High-Speed Arithmetic in Binary Computers", Proceedings of the IRE, Vol 49, pp.67-91, 1961.
5. Bruce Gilchrist, J.H. Pomerene and S.Y.Wong, "Fast Carry logic for Digital Computers", IRE Transactions on Electronic Computers, Vol.EC-4, PP.133-136, 1955.
6. R. De Mori, "Suggestions for an TC Fast Parallel Multiplier", Electronics letters, Vol.5, pp 50-51,1965
7. K'Adrea C.Bickerstaff,Michael J.Schulte,and Earl E. Swartz Lander, Jr., "Reduced Area Multipliers", Proceedings of the 1993 International Conference on Application Specific Array Processors, pp.478-489,1993.
8. Andrew D.Booth, "A Signed Binary Multiplication Technique", Quarterly Journal of Mechanics and Applied Mathematics, Vol .4, pp.236-240, 1951.
9. Charles R. Baugh and Bruce .A.Wooley, "A Two's Complement Parallel Array Multiplication Algorithm", IEEE Transactions on Computers, Vol. C-22, pp.1045-47, 1973.
10. Behrooz Parhami, "Computer Arithmetic: Algorithms and Hardware Designs, Newyork: Oxford University press, 2000.
11. Thomas K.Callaway and Earl E. Swatzlander,JR., "Optimizing Multipliers for WSI", Proceedings of the 1993 International Conference on Wafer Scale Integration, pp.85-94,1993.
12. C.S.Wallace," A suggestion for a Fast Multiplier", IEE Transactions on Electronic Computers, Vol.EC-13, pp.14-17, 1964.
13. Luigi Dadda," Some Schemes for Parallel Multipliers", Alta Frequenza, Vol.34, pp.349-356, August 1965.
14. Advanced Asic Chip Synthesis by Himanshu Bhatnagar Second Edition pp-183-256, 2002.
15. Wey C.L and Chang T.Y., "Design and analysis of VLSI-based parallel multipliers", IEEE proceedings computers and Digital Techniques, vol.137, no.4,pp,328-336, July 1990.(Journal paper)

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication