Implementation of Dadda and Array Multiplier Architectures Using Tanner Tool

Addanki Purna Ramesh
Associate Professor, Department of ECE
Sri Vasavi Engg College, Tadepalliguem.

Abstract: **The heart of the MAC unit is the multiplier. Multipliers are the fundamental components in all digital processing systems. Many research efforts have been devoted to reducing the power dissipation of different multipliers. The largest contribution to the power consumption in a multiplier is due to generation and reduction of partial products. Among multipliers, tree multipliers are used in high speed applications such as filters, but these require large area. The carry-select-adder (CSA)-based radix multipliers, which have lower area overhead, employ a greater number of active transistors for the multiplication operation and hence consume more power. Hence in this work, proposing a new power aware VLSI architecture for 16 bit multiplication process for DADDA multiplier in a schematic editor using tanner tool, T-spice is used as simulator and w-editor is used for formal verification of the multiplier.**

Key words: **Dadda Multiplier, Tanner Tool, Array Multiplier**

I. INTRODUCTION

Multipliers are among the fundamental components of many digital systems and, hence, their power dissipation and speed are of primary concern. For portable applications where the power consumption is the most important parameter, one should reduce the power dissipation as much as possible. One of the best ways to reduce the dynamic power dissipation, henceforth referred to as power dissipation in this paper, is to minimize the total switching activity, i.e., the total number of signal transitions of the system.

Multiplication plays an essential role in computer arithmetic operations for both general purpose and digital signal processors. For computational extensive algorithms required by multimedia functions such as finite impulse response (FIR) filters, infinite impulse response (IIR) filters and fast Fourier transform (FFT), the percentage of power consumption occupied by multiplication shows the importance itself.

(a) Array multiplier

The composition of an array multiplier is shown in Fig 1 There is a one-to-one topological correspondence between this hardware structure The generation of N partial products requires $N \times M$ two-bit AND gates most of the area of the multiplier is devoted to the adding of the N partial products, which requires N -1 M-bit adders. The shifting of the partial products for their proper alignment is performed by simple routing and does not require any logic. The overall structure can easily be compacted into a rectangle, resulting in a very efficient layout.

Fig 1: 4 × 4 bit-array multiplier

Due to the array organization, determining the propagation delay of this circuit is not straightforward. Consider the implementation of the partial sum adders are implemented as ripple-carry structures. Performance optimization requires that the critical timing path be identified first. This turns out to be nontrivial. In fact, a large number of paths of almost identical length can be identified. A closer look at those critical paths yields an approximate expression for the propagation delay.

$t_{mult}= [(M-1) +(N-2)]t_{carry} + (N-1)t_{sum}+ t_{and}$

where $\mathbf{t_{carry}}$ is the propagation delay between input and output carry, $\mathbf{t_{sum}}$ is the delay between the input carry and sum bit of the full adder, and $\mathbf{t_{and}}$ is the delay of the AND gate**.**

Since all critical paths have the same length, speeding up just one of them—for instance, by replacing one adder by a faster one such as a carry-select adder—does not make much sense from a design standpoint. AH critical paths have to be attacked at the same time. From the above equation, it can be deduced that the minimization of $\mathbf{t_{mult}}$ requires the minimization of both $\mathbf{t_{carry}}$.

(b)Dadda multiplier

In a popular multiplication scheme the array, the summation proceeds in a more regular, but slower manner, to obtaining the summation of the partial products .Using this scheme only one row of bits in the matrix is eliminated at each stage of the summation.

In a parallel multiplier the partial products are generated by using array of AND gates. The main problem is the summation of the partial products, and it is the time taken to perform this summation which determines the maximum speed at which a multiplier may operate. The Dadda scheme essentially minimizes 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**.**

Dadda multipliers are a refinement of the parallel multipliers presented by Wallace. Dadda multiplier consists of three stages. The partial product matrix is formed in the first stage by N^2 AND stages. In the second stage, the partial product matrix is reduced to a height of two. Dadda replaced Wallace Pseudo adders with parallel (n, m) counters. A Parallel (n, m) counter is a circuit which has n inputs and produce m outputs which provide a binary count of the ONEs present at the inputs. A full adder is an implementation of a (3, 2) counter which takes 3 inputs and produces 2 outputs. Similarly a half adder is an implementation of a (2, 2) counter which takes 2 inputs and produces 2 outputs.

In Dadda multipliers that reduce the number of rows as much as possible on each layer, Dadda multipliers do as few reductions as possible. Because of this, Dadda multipliers have less expensive reduction phase, but the numbers may be a few bits longer, thus requiring slightly bigger adders.

In general, the product, p, of two n-bit unsigned binary numbers x and y may be expressed as follows:

$$(p_{(2n-1)}\, p_{(2n-2)} \ldots\ldots p_2\, p_1\, p_0) = \sum_{i=0}^{n-1} \{y_i \wedge (x_{n-1} \ldots\ldots x_0)\} \cdot 2^i$$

In a parallel multiplier, the terms $y_i \wedge (x_{n-1} - \ldots x_0)$ are known as the partial products and are generated using an array of AND gates. For a parallel multiplier, the shifting term 2^i is inherent in the wiring and does not require any explicit hardware. Thus the main problem is the summation of the partial products, and it is the time taken to perform this summation which determines the maximum speed at which a multiplier may operate.

The realization of a parallel multiplier for digital computers has been considered in [7] by C.S. Wallace, who proposed a tree of pseudo-adders (that means adders without carry propagation) producing two numbers, whose sum equals the product. This sum can be obtained by applying the two numbers to a carry-propagating adder.

Consider the process of multiplication of two binary numbers, each composed of n bit, as been based on obtaining the sum of v summands.These summands are obtained, in the simplest schemes, by shifting left the multiplicand by 1, 2, 3,….(n-1) places, and multiplying it by the corresponding bits of the multiplier. In this situation v = n. Now the number of summands can be made less than n by using some multiples of the multiplicand, on the basis of two or more multiplier digits[7]. Hence a proposed architecture can be developed by L Dadda, which works on the principle of reducing the number of summands. This architecture is based on the use of logical blocks called it as parallel (n, m) counters, these are combinational networks with m outputs and $n(\leq 2^m)$ inputs. The m outputs, considered as a binary number, codify the number of « ones» present at the inputs.

II Proposed architecture: Dadda multiplier

The Dadda multiplier is a hardware multiplier design, invented by computer scientist Luigi Dadda in 1965. It is slightly faster (for all operand sizes) and requires fewer gates (for all but the smallest operand sizes) than array multiplier.

Dadda multipliers have the same 3 steps:

1. Multiply (that is - AND) each bit of one of the arguments, by each bit of the other, yielding N^2 results. Depending on position of the multiplied bits, the wires carry different weights, for example wire of bit carrying result of a2b3 is 32.

2. Reduce the number of partial products to two layers of full and half adders.

3. Group the wires in two numbers, and add them with a conventional adder.

Dadda multipliers perform few reductions only when compared to Wallace multiplier. Because of this, Dadda multipliers have less expensive reduction phase, but the numbers may be a few bits longer, thus requiring slightly bigger adders

To achieve this, the structure of the second step is governed by slightly more complex rules than in the wallace multipliers. The reduction rules however are as follows:

Take any 3 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 3 input wires.

If there are 2 wires of the same weight left, and the current number of output wires with that weight is equal to 2 (modulo 3), input them into a half adder. Otherwise, pass them through to the next layer.

If there is just 1 wire left, connect it to the next layer.

This step does only as many adds as necessary, so that the number of output weights stays close to a multiple of 3, which is the ideal number of weights when using full adders as (3, 2) counters.

However, when a layer carries at most 3 input wires for any weight, that layer will be the last one. In this case, the Dadda tree will use half adder more aggressively to ensure that there are only two outputs for any weight. Then, the second rule is above changes as follows

If there are 2 wires of the same weight left, and the current number of output wires with that weight is equal to 1 or 2 (modulo 3), input them into a half adder. Otherwise, pass them through to the next layer.

III Implementation of multiplier

In order to make the most effective use of the processing elements, the multiplier was implemented as a linear pipeline [9]. It was important to ensure that the delay of each processing stage in the pipeline was approximately equal so that a 'bottleneck' was not introduced by any individual processing stage.

The multiplication of an M-bit multiplicand by an N-bit multiplier yields an N by M matrix of partial products. The reduction of this partial product matrix through the parallel application of (3, 2) and (2, 2) counters results in a matrix with a height of two. Each (3, 2) counter (full adder) accepts three inputs from a given column and produces a sum bit which remains in that column and a carry bit which goes into the next more significant column. A (2, 2) counter (half adder) accepts two inputs from a column and produces a sum bit in the same column and a carry bit in the next more significant column.

The implemented 16 × 16 Dadda multiplier with the help of dot diagram is shown in Fig 2 (The notation is taken from [8][10] in which the outputs from a full adder are joined by a solid line, and those from half adders are joined by a line with a dash through the centre). The Dadda scheme essentially minimizes the number of adder stages required to perform the summation of the partial products. This is achieved by using full and half adders to reduce the number of rows in the matrix of bits at each summation stage by a factor of 3/2. This results in a final matrix consisting of two rows of bits which must be summed using a multiple-bit adder (e.g. a ripple-carry or carry look-ahead adder). The corresponding circuit for a multiplier using this scheme is shown in Fig 3.2. By way of contrast, in a popular multiplication scheme the array, the summation proceeds in a more regular, but slower manner, to obtaining the summation of the partial products .Using this scheme only one row of bits in the matrix is eliminated at each stage of the summation.

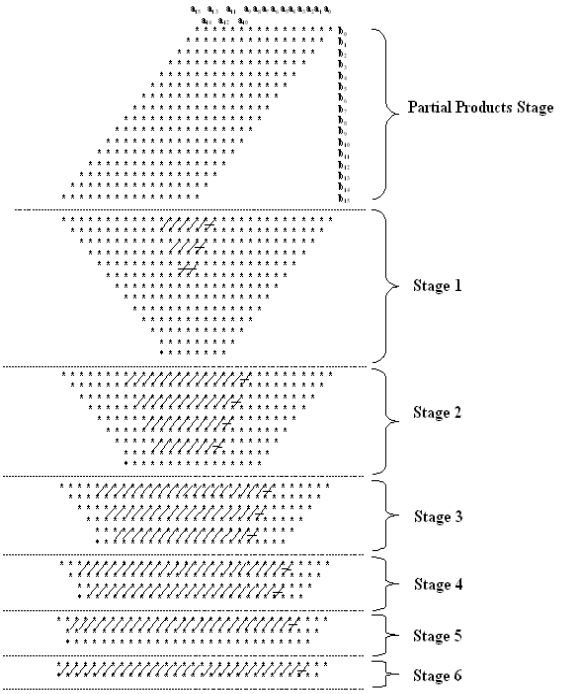

Fig 2: Dot diagram of proposed 16 × 16 Dadda multiplier

The process of Dadda multiplication is as follows: The entire 16 × 16 multiplication requires six stages. Always the first stage is partial products stage, which is obtained by simple multiplication of multiplicand with multiplier. The number of rows (height) present at this stage is 16. Now reduce the number of rows further in such a way that final stage contains only two rows. For this, Dadda [8] [10] introduces a sequence of intermediate matrix heights that provides the minimum number of reduction stages for a given size multiplier. This sequence determined by working back from the final two row matrix, limit the height of each intermediate matrix to the largest integer that is no more than 1.5 times the height of its successor. The proposed multiplier 16x16 Dadda multiplier requires six reduction stages with intermediate matrix heights of 13, 9,6,4,3 and finally 2.

The single bit in 1^{st} column of the first stage represents the least significant bit of the product. From the dot diagram, 2 – row stage can be derived from 3 – row stage, and 3 – row stage can be derived from 4 – row stage with the help of (3, 2) and (2, 2) counters. This is $(S-1)^{th}$ stage, where S is the number of stages to implement the multiplier.

The 4 – row stage can be derived from 6 – row stage. This is $(S-2)^{th}$ stage. The 6 – row stage can be derived from 9 – row stage. This can be $(S-3)^{th}$ stage. The 9 – row stage can be derived from 13 – row stage. This is $(S-4)^{th}$ stage and then finally 13 – row stage can be derived from partial product stage.

In passing from partial products stage to stage 1, columns are partially reduced, so that no more than 13 rows are obtained. From the dot diagram, column 14(14^{th} bit) of partial products stage will be transformed in a 13 –bits column in stage 1 by reproducing 12 bits without transformation and transforming only 2 bits by (2, 2) counter. Consequently, column 15 (15^{th} bit and 14^{th} bit) of the partial products stage will be transformed in a 13 – bits column in stage 1 by reproducing 12 bits without transformation and transforming only 2 bits by a (3, 2) counter with the help of the carry generated from the previous column. Consequently, only some columns in the central portion of partial products stage are actually transformed.

In passing from stage 1 to stage 2, columns having no more than 9 bits are obtained by means of applying (2, 2) and (3,2) counters. In succeeding transformations, columns with no more than 6, 4, 3 and 2 bits respectively are obtained.

In this Dadda implementation, in general, the number of full adders required is N^2-4N+3 and the number of half adders is always N-1.

The below table 1 shows the number of reduction stages required to implement Dadda architecture for various number of bits.

Table1: Number of reduction stages for Dadda multiplier

Bits in Multiplier(N)	Number of Stages
3	1
4	2
$5 \leq N \leq 6$	3
$7 \leq N \leq 9$	4
$10 \leq N \leq 13$	5
$14 \leq N \leq 19$	6
$20 \leq N \leq 28$	7
$29 \leq N \leq 42$	8
$43 \leq N \leq 63$	9
$63 \leq N \leq 94$	10

IV Algorithm

1.Multiply (that is - AND) each bit of one of the arguments, by each bit of the other, yielding N^2 results.

2.Reduce the number of partial products to two layers of full and half adders. For this, Dadda reduction scheme uses the following algorithm.

a)Let $d_1 = 2$ and $d_{j+1} = [3.d_j / 2]$, where d_j is the matrix height for the j-th stage from the end. Find the largest j such that at least one column of the matrix has more than d_j bits.

b)Employ (3, 2) and (2, 2) counters to obtain a reduced matrix with no more than d_j elements in any column.

c)Until a matrix with only two rows is generated. Let j = j-1 and repeat step b

3.Group the wires in two numbers, and add them with a conventional adder.

V Flow Chart

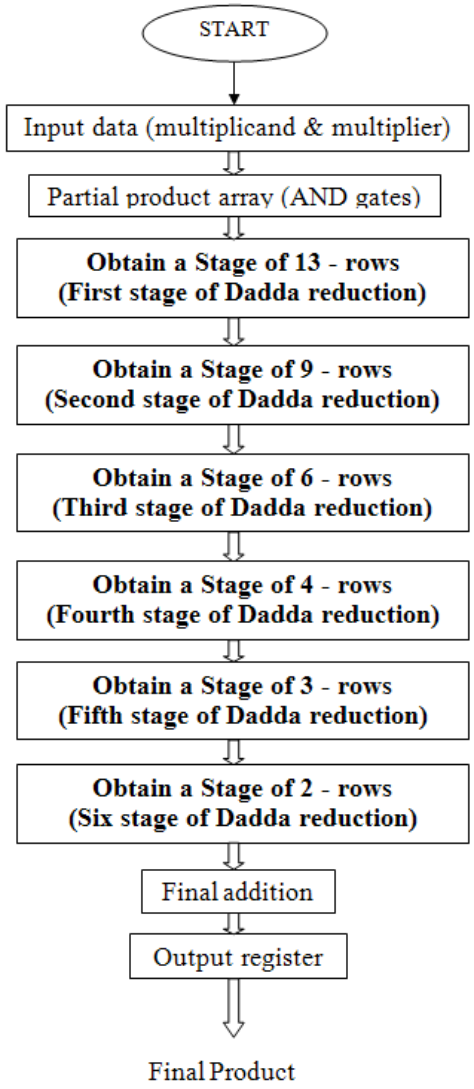

Fig 3: Flow Chart of Proposed 16 × 16 Dadda multiplier

VI Schematic editor

For this project we used TANNER software tools (T-spice) because it is designed to solve a wide variety of circuit problems. Its flexibility is due to robust algorithms which can be optimized by means of user-adjustable parameters. T-Spice uses Kirchhoff's Current Law (KCL) to solve circuit problems. To T-Spice, a circuit is a set of devices attached to nodes. The circuit's state is represented by the voltages at all the nodes. T-Spice solves for a set of node voltages that satisfies KCL (implying that the sum of the currents flowing into each node is zero).

In order to evaluate whether a set of node voltages is a solution, T-Spice computes and sums all the currents flowing out of each device into the nodes connected to it (its terminals). The relationship between the voltages at a device's terminals and the currents through the terminals is determined by the device model. For example, the device model for a resistor of resistance *R* is $I = \Delta v / R$, where Δv represents the voltage difference across the device.

Most T-Spice simulations start with a DC operating point calculation. A circuit's *DC* operating point is its steady state, which would in principle be reached after an infinite amount of time if all inputs were held constant. In DC analysis, capacitors are treated as open circuits and inductors as short circuits. Because many devices, such as transistors, are described by nonlinear device models, the KCL equations that T-Spice solves in DC analysis are nonlinear and must therefore be solved by iteration.

VII Implementation of basic multiplier

Inverter

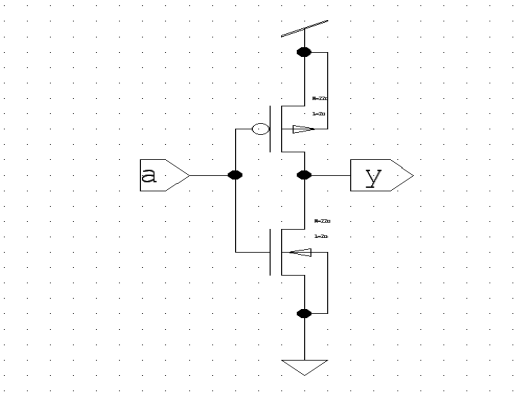

Fig 4: Schematic diagram of inverter

AND gate

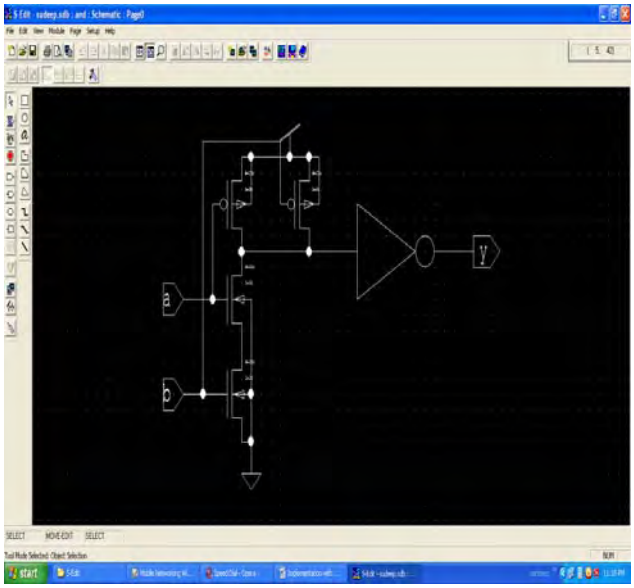

Fig 5: Schematic diagram of AND gate

OR gate

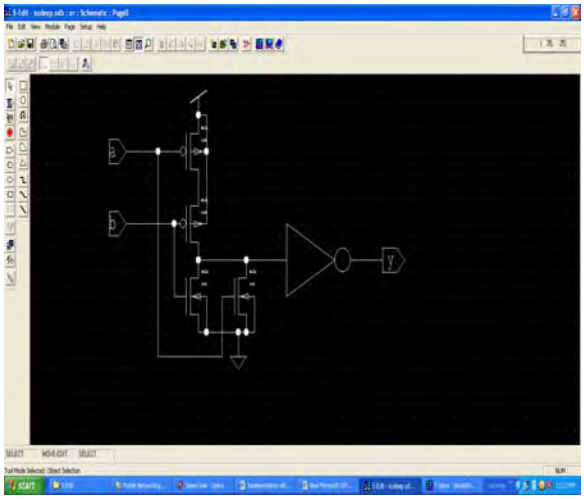

Fig 6: Schematic diagram of OR gate

Half adder

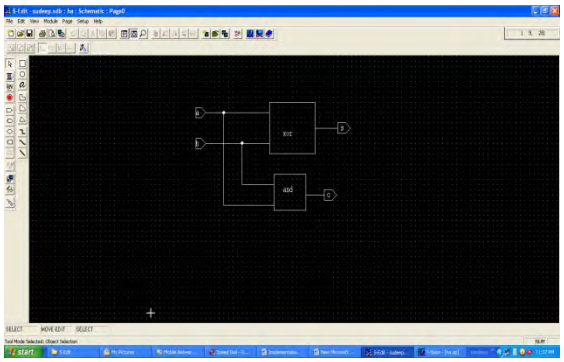

Fig 7: Schematic diagram of half adder

Full adder

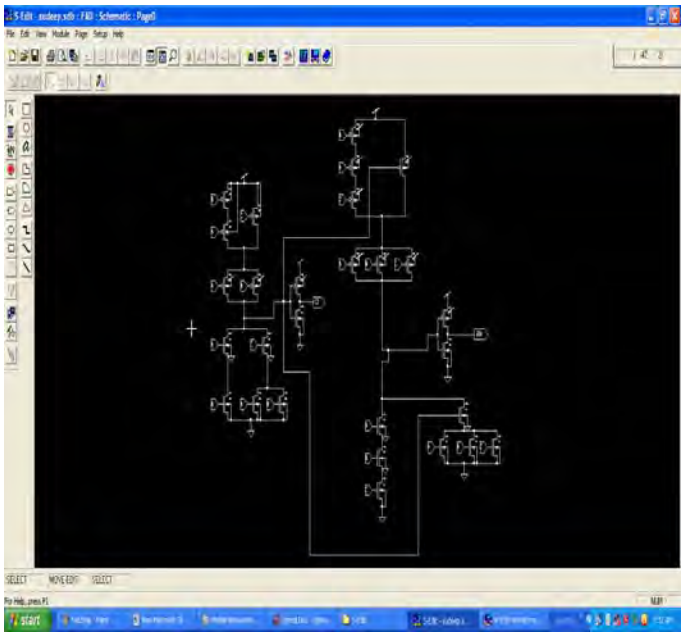

Fig 8: Schematic diagram of Full adder

8 × 8 Array multiplier

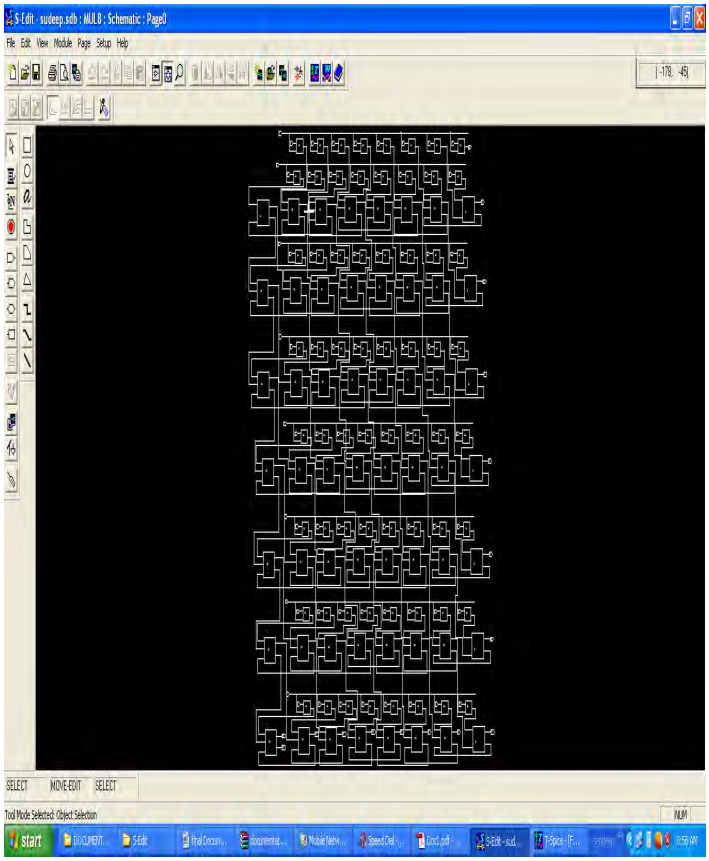

Fig 9: Schematic diagram of 8 × 8 array Multiplier

8 × 8 array multiplier waveform

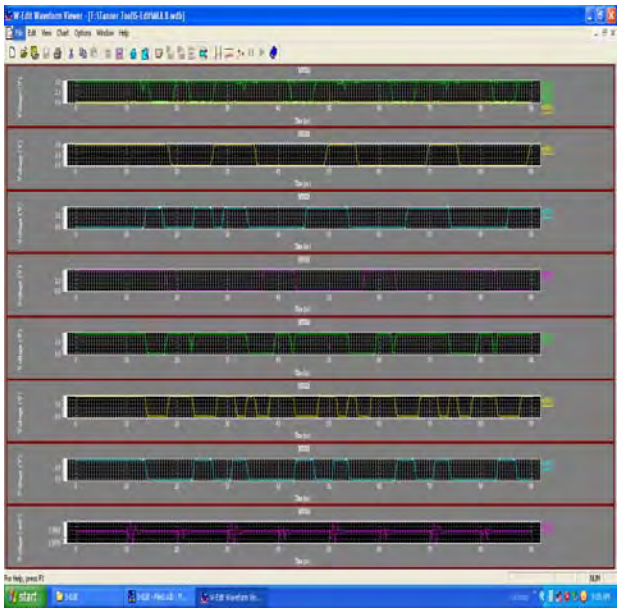

Fig 10: Output waveform of 8 × 8 array multiplier

16 × 16 Array multiplier

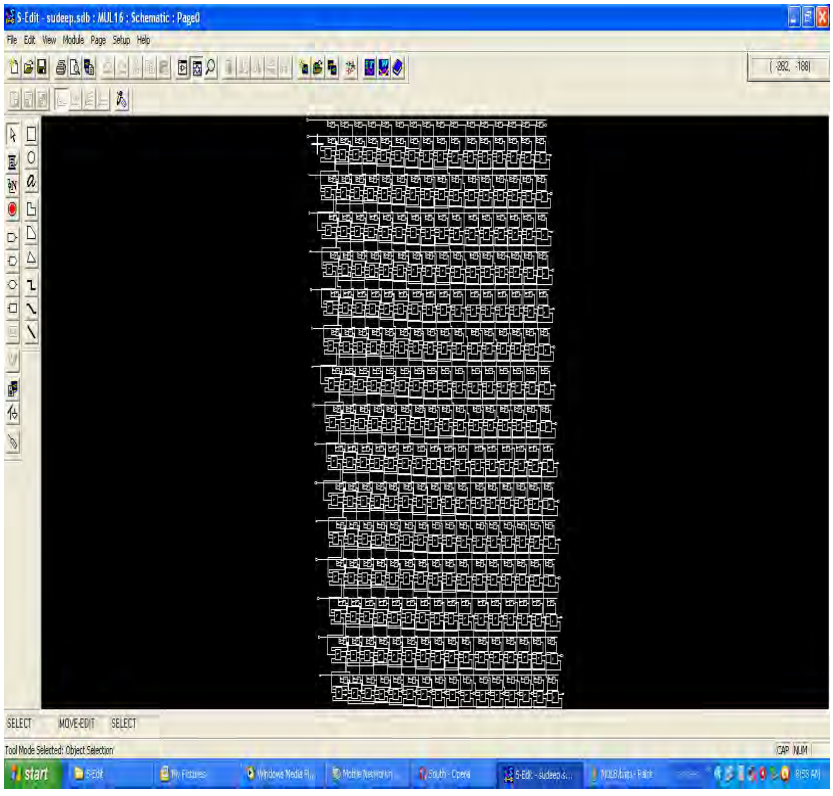

Fig 11: Schematic diagram of 16 × 16 array multiplier

16 × 16 array multiplier waveform

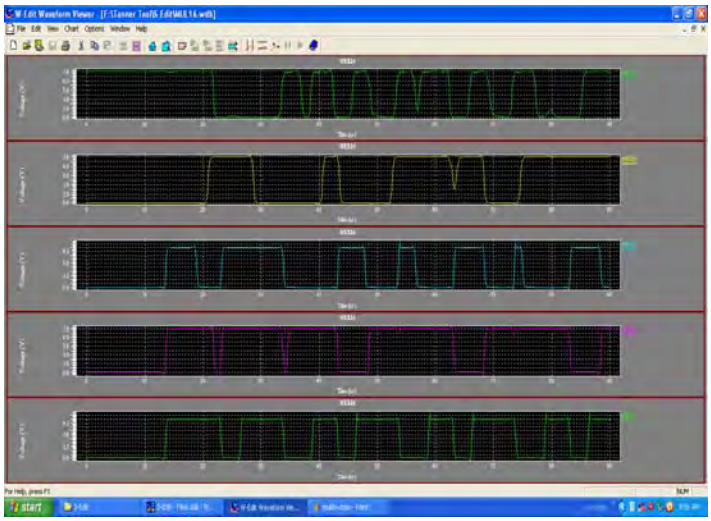

Fig 12: Output waveform of 16 × 16 array multiplier

8 × 8 Dadda multiplier

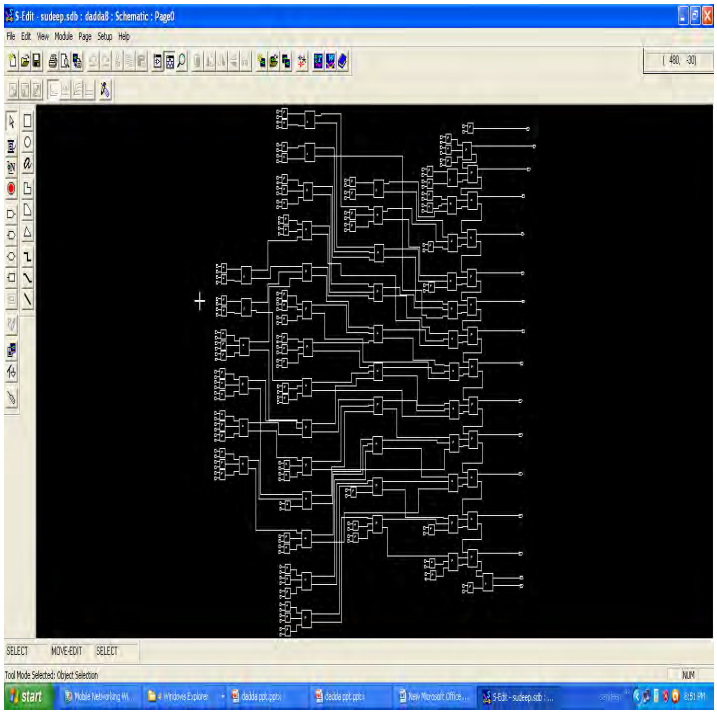

Fig 13: Schematic Diagram of 8 × 8 Dadda multiplier

8 × 8 Dadda multiplier wave form

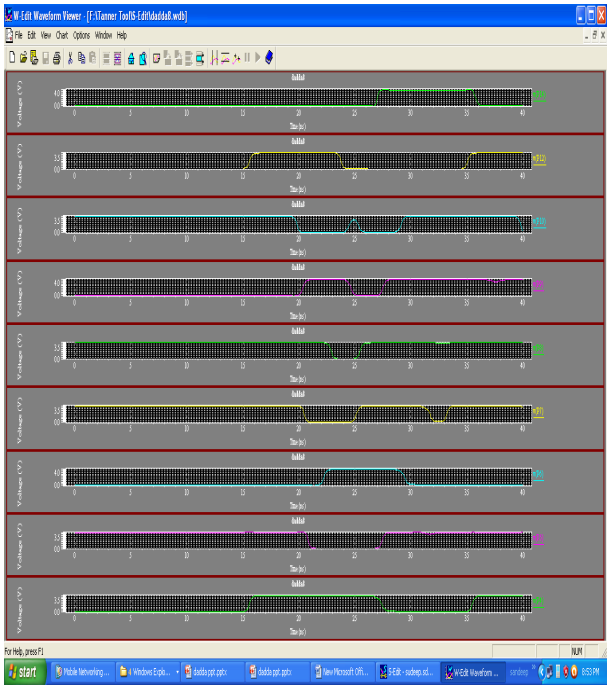

Fig13: Output waveform of 8 × 8 Dadda multiplier

16 × 16 Dadda multiplier

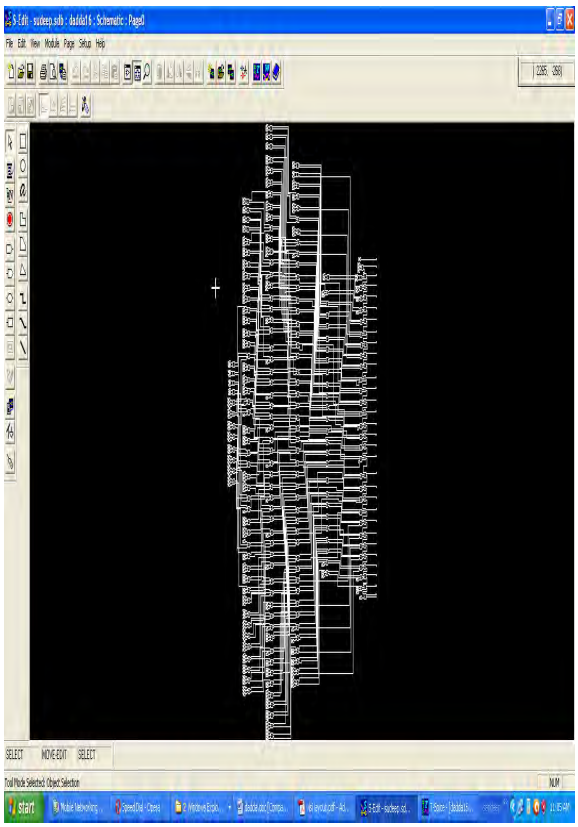

Fig14: Schematic Diagram of 16 × 16 Dadda multiplier

Schematic wave form of 16x16 Dadda multiplier

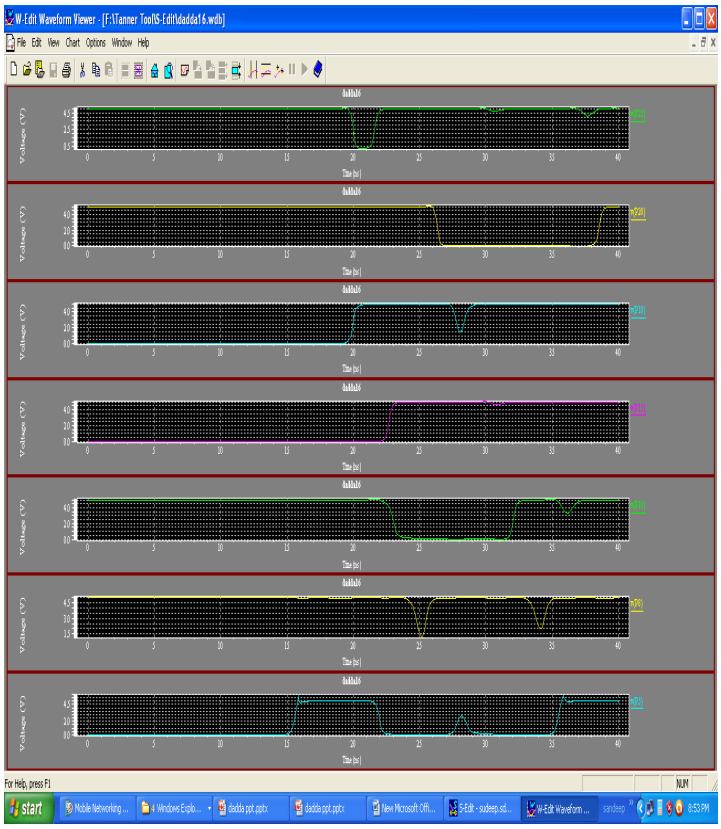

Fig15: Output waveform of 16 × 16 Dadda multiplier

VIII Results

In this work, Dadda multiplier is implemented by schematic editor using tanner tool, T-spice is used as simulator and w-editor is used for formal verification of the multiplier

In conventional 16 × 16 array multiplier architecture, 240 adders are required to implement the multiplier, where as in the proposed Dadda multiplier, the total number of adders required are 210. Hence the proposed Dadda multiplier saving of 30 adders, then it reduces the total switching activity of circuit design.

The below table 2 shows the comparison between conventional array multiplier and Dadda multiplier (for both 16 × 16 and 8 × 8 operations).

Table2: Comparison between array and Dadda multiplier

Parameter	**8 × 8 array multiplier**	**8 × 8 Dadda multiplier**
Hardware requirement	Adders – 56 (Full adders –48 Half adders – 8)	Adders –42 (Full adders – 35 Half adders – 7)
Time(delay)	104.65 Seconds	33.61 Seconds

Voltage(v)	8 × 8 array multiplier	8 × 8 Dadda multiplier
5	2.33e-003w	1.83e-003W
4	1.306e-001w	1.085e-001w
3	6.268e-002w	5.109e-002w
2	2.144e-002w	1.775e-002w
1	2.703e-002w	2.552e-003w

Voltage(v)	Power 16 × 16 array multiplier	16 × 16 Dadda multiplier
5	3.76 e-003 W	3.16 e-003
4	4.129e-001w	3.046e-001w
3	2.087e-001w	2.054e-001w
2	7.295e-002w	6.883e-002w
1	1.0074e-002w	1.012e-002w

IX Conclusions

In this project, a proposed Dadda multiplication scheme is implemented for a 16 bit × 16 bit multiplication. With respect to the parameters power consumption, area estimate and hardware requirement, this Dadda multiplication technique is better than the conventional array multiplication schemes. Hence in this work, saving of 84% of power consumption, reduction of 30 adders and saving of 61.18% of time can be done, when compared with array multiplication techniques.

X Future Scope of Work

As can be seen from the results obtained by Dadda multiplication scheme, this approach is further extended to perform the multiplication of higher bits (i.e., 32 bit × 32 bit, 64 bit × 64 bit and so on). The power consumption and area estimate are further reduced by implementing the final adder with look ahead carry generation logic (look ahead carry adder).

XI References

[1] HENLIN, D.A., FERTSCH, M.T., MAZIN, M., and LEWIS, E.T.: *"A 16 × 16 bit pipelined multiplier macrocell"*, IEEE, J. Solid-State Circuits, 1985, SC-20, pp. 542-547.

[2] HATAMIAN, M., and CASH G L: *"A 70-MHz 8-bit × 8-bit parallel pipelined multiplier in 2.5-μm CMOS"* IEEE J Solid-state circuits, 1986, SC-21, pp.505 – 513.

[3] SCHMITT-LANDSIEDEL, D., NOLL, T.G., KLAR, H., and ENDERS, G.: *"A pipelined 330 MHz multiplier*". ESSCIRC '85, 11th European Solid State Circuits Conf. 16 – 18 September 1985.

[4] LEE, F S., KAELIN, G R., WELCH, B M., ZUCCA, R., SHEN, E., ASBECK, P., LEE, C.P., KIRKPATRICK, C. G., LONG, S.I., and EDEN R. C.:*" A High-Speed LSI GaAs 8 × 8 bit parallel multiplier"*, IEEE J. Solid – state Circuits, 1982, SC – 17, pp.638 – 645.

[5] YUNG, H.C., and ALLEN, C.R.: *"Part 1: VLSI implementation of an optimized hierarchical multiplier"*, IEE Proc. G, Electron. Circuits & Syst., 1984,131, (2), pp. 56-60.

[6] J. V. MCCARMY, D. PHIL, and J. G. MCWHIRTER, "*Completely iterative, pipelined multiplier array suitable for VLSI*," Proc. Inst. Elec. Eng., vol. 129, Pt. G, no. 2, pp. 40–46,Apr.1982.

[7] WALLACE, C.S.: *"A Suggestion for a fast multiplier"*, IEEE Trans. on Electronic Computers, vol. EC – 13, pp 14 – 17, February 1964.

[8] DADDA, L.: 'Some Schemes for Parallel Multipliers', *Alta Freq.*, 34, 1965, pp. 349-356

[9] D. G. CRAWLEY and G. A. J. AMARATUNGA, *"8×8 bit pipelined Dadda multiplier in CMOS" in* IEE Proceedings-Circuits, Device and Systems, vol. 135, no. 6, December 1988, pp. 231–240.

[10] L. DADDA, "*On Parallel Digital Multipliers*," Alta Frequenza, vol. 45, pp. 574 – 580, 1976.