ISSN:0975-9646

Highly Accelerated Advanced Multiplier Design for An Efficient Bandwidth Utilized FFT Computation

M.Sahithi[#1],Naseema Shaik[#2], A.Jhansi Rani[#3],J.Poornima[#4],M.Jyothi[#5],K.Purnima[#6]

[#1]*M.Tech student, Department of ECE, K L University*
Vijayawada, INDIA

[#2, 3, 4, 5,6]*Department of ECE, K L University*
Vijayawada,INDIA

[*]*Naseema Shaik, Department of ECE, KL University*
Vijayawada, INDIA

***Abstract*— Fast multipliers are essential parts of digital signal processing systems. The speed of multiply operation is of great importance in digital signal processing as well as in general purpose processors today especially since the media processing took off. We present a Fast fourier transform implementation using Twin precision technique. The twin precision technique can reduce the power dissipation by adapting a multiplier to the bit width of the operands being computed. The algorithm used here is Baugh-Wooley algorithm. By adapting to actual multiplication bit-width using twin precision technique, it is possible to save power, increase speed, double computation throughput and highly efficient. By using this the execution time of a Fast fourier transform is reduced with 15% at a 14% reduction in datapath energy dissipation.**

***Keywords*— Fast Fourier Transform, Highly efficient, Baugh-Wooley Algorithm.**

I. INTRODUCTION

During the last decade of integrated electronic design ever more functionality has been integrated on to the same chip paving the way for having a system on single chip. The strive for ever more functionality increase the demands on circuit designers that have to provide the foundations for all these functionality. With an increased interest and use of reconfigurable [1] architectures there is a need for flexible and reconfigurable computational units that can meet the demand of high speed, high throughput, low speed and area efficiency. Multiplications are complex to implement and they continue to give the designers headaches when trying to efficiently implement multipliers in hardware. Multipliers are therefore interesting to study, when investigating how to design flexible and reconfigurable computations.

Today complex circuits are described in hardware descriptive languages like Vhdl and verilog and are synthesized to gate level. A core operation in actual circuits, especially in Digital signal processing like Filtering, Modulation, Video processing, Neural networks, Satellite Communication, Graphics or Control systems etc is multiplication. In past multiplication was generally implemented via addition, subtraction and shift operations. Multiplication can be considered as a series of repeated additions. The repeated addition method suggested by arithmetic definition is slow that is almost always replaced by algorithm that's make use of positional representation. It is possible to decompose multipliers into two parts. The first part is dedicated to generate partial products and the second one collects and adds them.

Multiplication is therefore a multi operand operation. To extend multiplication to both signed and unsigned numbers, a convenient number system would be the representation of numbers in two's complement format. In this we present the Fast fourier transform implementation using Twin precision technique. By using the number of multiplications has been reduced. This can be achieved in less amount of time, therefore we get high throughput, high efficient, high speed. The algorithm used here is Baugh-Wooley algorithm.

II. FAST FOURIER TRANSFORM

Fast fourier transform is an efficient algorithm to compute the Discrete fourier transform and its inverse. A DFT decomposes a sequence of values into components of different frequencies. This operation is useful in many fields but often computing is too slow to be practical. An FFT is a way to compute same result more quickly; computing the DFT of N points is the naïve way it takes $O(N^2)$ arithmetical operations, while FFT computes it in only in O(N log N) operations.

The DFT is defined by the formula

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k \frac{n}{N}} \qquad k = 0, \ldots, N-1.$$

Evaluating this definition directly require $O(N^2)$ operations. To illustrate the savings of FFT consider the count of complex multiplications and additions. Evaluating the sum of DFT's involves N^2 complex multiplications and N(N-1) addition operations. The well known Cooley-Tucky algorithm [2] for N a power of 2 can compute the same result with only (N/2) $\log_2$ N complex multiplier and N $\log_2$ N complex additions. A basic element of Butterfly model is shown in Fig.1.

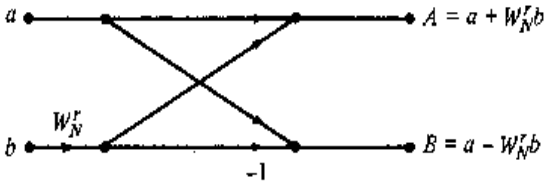

Fig . 1 Basic butterfly computation in the decimation-in-time FFT algorithm.

The butterfly element is used to construct larger structures to perform larger transformations. A structure of 8-point FFT using butterfly model is shown in Fig.2. where it also shown that there are three stages in 8 –point FFT calculation.

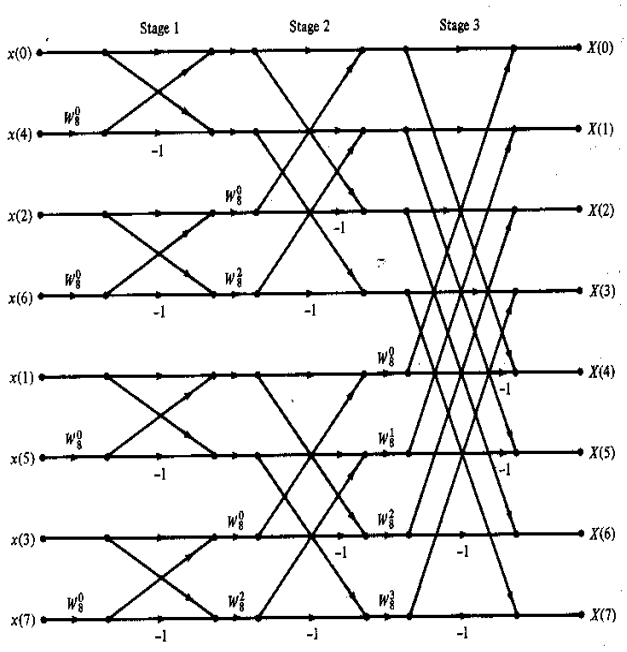

Fig.2. Eight-point decimation-in-time FFT algorithm.

From the fig.2. it mainly shows the basic butterfly model with complex multiplication and addition. So in the coming topics we present our twin precision technique and we can show how it easily performed.

III. PROPOSED MODEL

In this we propose a Twin precision technique for the fast implementation of Fast Fourier transform using Baugh – Wooley algorithm.Thia can be seen in the next topics.

A. Twin Precision technique

For a first analysis of the twin-precision [3] technique, the discussion will be based on an illustration of an unsigned binary multiplication. In an unsigned binary multiplication each bit of one of the operands, called the multiplier, is multiplied with the second operand, called multiplicand. That way one row of partial products is generated. Each row of partial products is shifted according to the position of the bit of the multiplier, forming what is commonly called the partial-product array. Finally, partial products that are in the same column are summed together, forming the final result. An illustration of an 8-bit multiplication is shown in fig.3

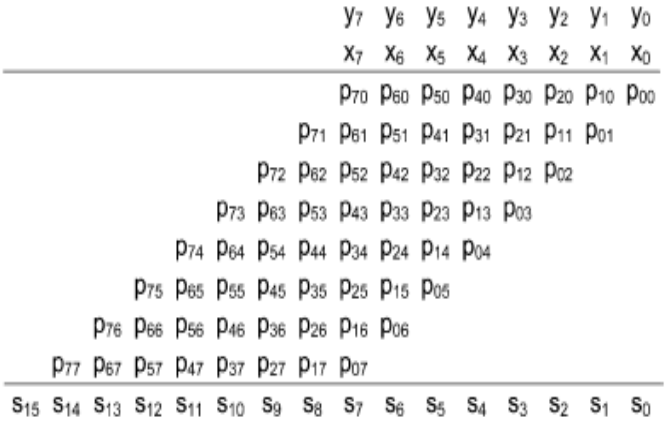

Fig. 3: Illustration of an unsigned 8-bit multiplication

Let us look at what happens when the precision of the operands is smaller than the multiplier we intend to use. In this case, the most significant bits of the operands will only contain zeros, thus large parts of the partial-product array will consist of zeros. Further, the summation of the most significant part of the partial-product array and the most significant bits of the final result will only consist of zeros. An illustration of an 8-bit multiplication, where the precision of the operands is four bits, is shown in Fig.4.

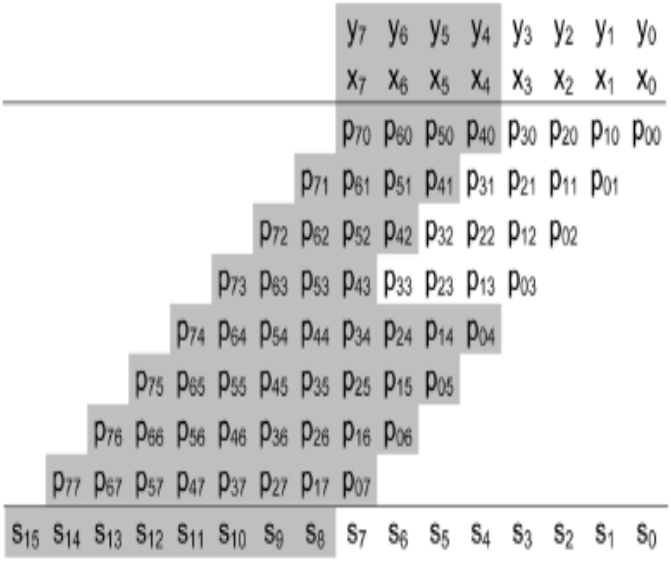

Fig .4. Illustration of an unsigned multiplication, where the precision of operands is smaller than precision of multiplication. Unused bits of operands and products, as well as unused partial products are shown in gray

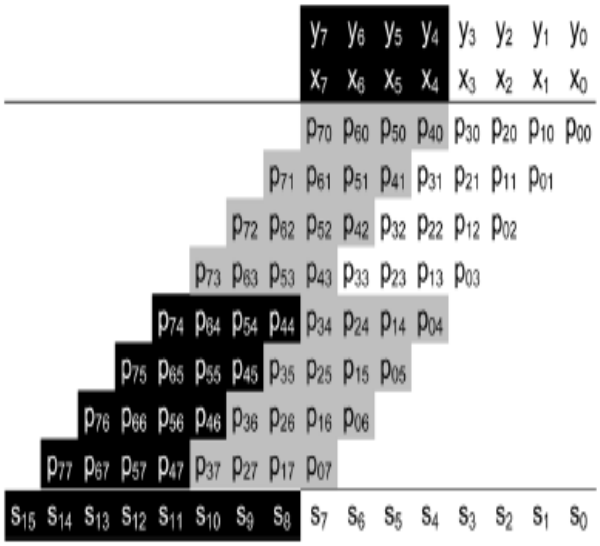

Fig .5.Illustration of an unsigned 8-bit multiplication, where a 4-bit multiplication is shown is white is computed in parallel with a second 4-bit multiplication shown in black.

Fig. 4.shows that large parts of the partial products are only containing zeros and are, thus, not contributing with any useful information for the final result. What if these partial products could be utilized for a second, concurrent multiplication? Since partial products of the same column are summed together, it would not be wise to use any of the partial products that are in the same column as the multiplication that is already computed. Looking closer at the 4-bit multiplication marked in white in Fig. 4, one can also observe that the column at position S7 should not be used either. This is because that column might have a carry from the active part of the partial-product array that will constitute the final S7.

Altogether this makes only the partial products in the most significant part of the partial-product array available for a second multiplication. In order to be able to use the partial products in the most significant part, there has to be a way of setting their values. For this we can use the most significant bits of the operands, since these are not carrying any useful information. If we are only looking at the upper half of the operands, the partial products generated from these bits are the ones shown in black in Fig. 5. By setting the other partial products to zero, it is then possible to perform two multiplications within the same partial-product array, without changing the way the summation of the partial-product array is done. How the partial products, shown in gray, can be set to zero will be investigated in the implementation section later on.

Assume, for now, that there is a way of setting unwanted partial products to zero, then it suddenly becomes possible to partition the multiplier into two smaller multipliers that can compute multiplications in parallel. In the above illustrations the two smaller multiplications have been chosen such that they are of equal size. This is not necessary for the technique to work. Any size of the two smaller multiplications can be chosen, as long as the precision of the two smaller multiplications together are equal or smaller than the full precision [4] (NFULL) of the multiplication, equation below. To be able to distinguish between the two smaller multiplications, they are referred to as the multiplication in the Least Significant Part (LSP) of the partial-product array with size NLSP, shown in white, and the multiplication in the Most Significant Part (MSP) with size NMSP , shown in black.

NFULL >= **N**LSP + **N**MSP

It is functionally possible to partition the multiplier into even more multiplications. For example, it would be possible to partition a 64-bit multiplier into four 16-bit multiplications. Given a number K of low precision multiplications their total size need to be smaller or equal to the full precision multiplication.

$$N \geq \sum_{i=1}^{K} N_i.$$

For the rest of this investigation, the precision of the two smaller multiplications will be equal and half the precision (N=2) of the full precision (N) of the multiplier. This is the main twin precision concept which is generally

used to do the multiplication very easily. So by using this we can get the high throughput, high speed. The Fast Fourier transform can be implemented using Bugh-Wooley algorithm. This can be explained in the next section.

B. Baugh-Wooley Algorithm

In the previous section, the concept of twin precision was introduced by looking at an unsigned multiplication. However, for many applications signed multiplications are needed and consequently an unsigned multiplier is of limited use. In this section a twin-precision multiplier based on the Baugh-Wooley (BW) algorithm will be presented.The Baugh-Wooley algorithm [5] is an efficient way to handle sign bits. The BW algorithm is a relative straightforward way of doing signed multiplications. Fig. 6 illustrates the algorithm for an 8-bit case, where the partial product array has been reorganized according the the scheme of Hatamian[6]. The creation of the reorganized partial-product array comprises three steps:

*i)*the most significant partial product of the first (N – 1) rows and the last row of partial

products except the most significant has to be negated,

ii) a constant one is added to the Nth column,

iii) the most significant bit (MSB) of the final result is negated.

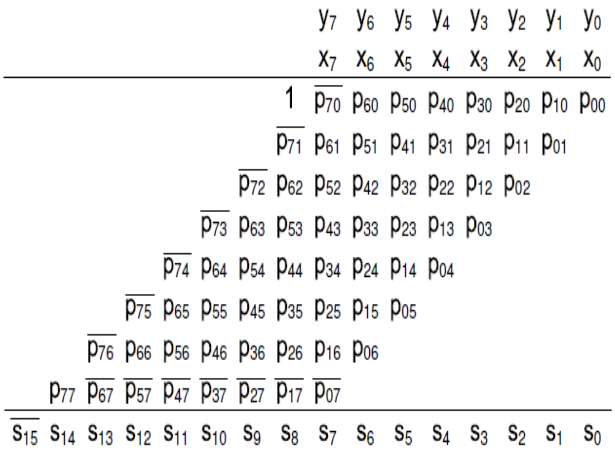

Fig. 6. Illustration of an 8-bit Baugh-Wooley multiplication

To combine twin-precision with BW is not as simple as for the unsigned multiplication,where only parts of the partial products needed to be set to zero. To be able to compute two signed N=2 multiplications, it is necessary to make a more sophisticated modification of the partial-product array. Fig. 6 shows an illustration of an 8-bit BW multiplication, where two 4-bit multiplications have been depicted in white and black. When comparing the illustration of Fig. 6 with that of Fig. 7 one can see that the only modification needed to compute the 4-bit multiplication in the MSP of the array is an extra sign bit '1' in column S12. For the 4-bit multiplication in the LSP of the array, there is a need for some more modifications. Looking at the active partial-product array of the 4-bit LSP multiplication (shown in white), we see that the most significant partial product of all rows, except the last, needs to be negated. For the last row it is the opposite, here all partial products, except the most significant, are negated. Also for this multiplication a sign bit '1' is needed, but this time in column S4. Finally the MSB of the result needs to be negated to get the correct result of the two 4-bit multiplications.

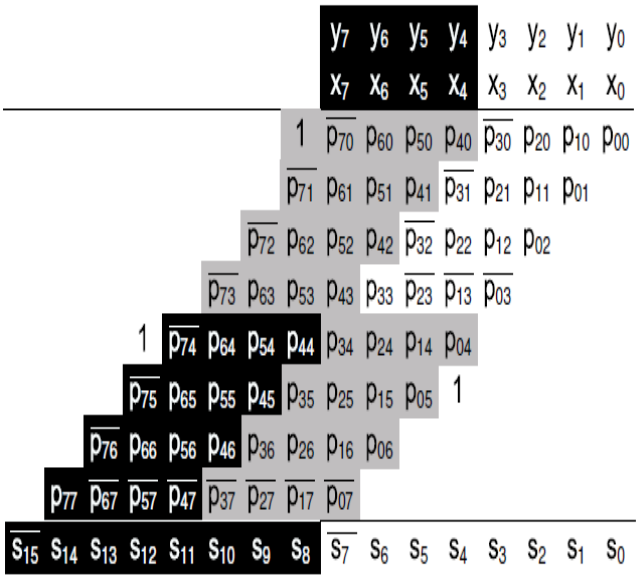

Fig .7.Illustration of signed 8-bit multiplication using Baugh-Wooley algorithm, where 4-bit multiplication shown in white is computed in parallel with a second 4-bit multiplication shown in black.

To allow for the full-precision multiplication of size N to coexist with two multiplications of size N=2 in the same multiplier, it is necessary to modify the partial-product generation and the reduction tree. For the N=2-bit multiplication in the MSP of the array all that is needed is to add a control signal that can be set to high, when the N=2-bit multiplication is to be computed and to low, when the full precision N multiplication is to be computed. To compute the N=2-bit multiplication in the LSP of the array, certain partial products need to be negated. This can easily be accomplished by changing the 2-input AND gate that generates the partial product to a 2-input NAND gate followed by an XOR gate. The second input of the XOR gate can then be used to invert the output of the NAND gate. When computing the N=2-bit LSP multiplication, the control input to the XOR gate is set to low making it work as a buffer. When computing a full-precision N multiplication the same signal is set to high making the XOR work as an inverter.

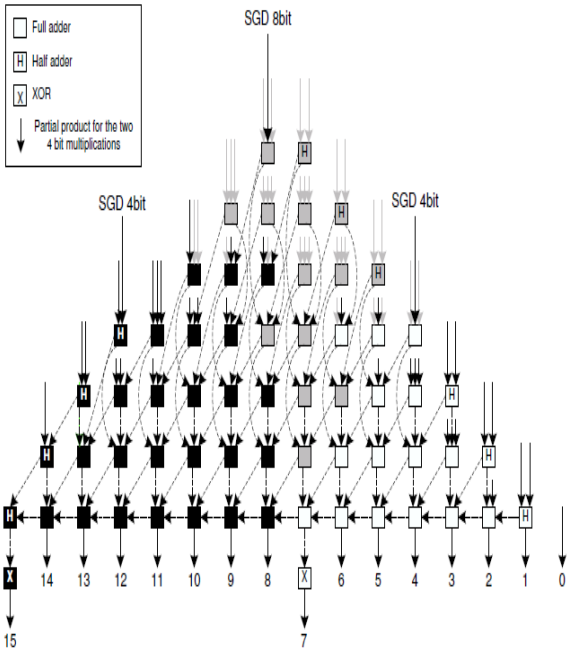

Fig .8. Block diagram of signed 8-bit multiplier using Baugh Wooley algorithm, where 4-bit multiplication shown in white is computed in parallel with a second 4-bit multiplication shown in black.

Fig. 8 shows an implementation of a twin-precision 8-bit BW multiplier. The modifications of the reduction tree compared to the unsigned 8-bit multiplier in Fig. 8 consist of three things; *i)* the half adders in column 4 and 8 have been changed to full adders in order to fit the extra sign bits that are needed, *ii)* for the sign bit of the 4-bit MSP multiplication there is no half adder that can be changed in column 12, so here an extra half adder has been added which makes it necessary to also add half adders for the following columns of higher precision, and *iii)* finally XOR gates have been added at the output of column 7 and 15 so that they can be inverted. The simplicity of the BW implementation makes it easy to also compute unsigned multiplications. All that is needed is to set the control signals accordingly, such that none of the partial products are negated, the XOR gates are set to not negate the final result and all the sign bits are set to zero.

From the above sessions we have seen the Twin precision technique fundamentals and also discussed about Baugh-Wooley algorithm and its implementation using Twin precision technique. The main aim is to implement Fast Fourier transform using Twin precision technique. The Fast Fourier transform butterfly model is shown in Fig.9.

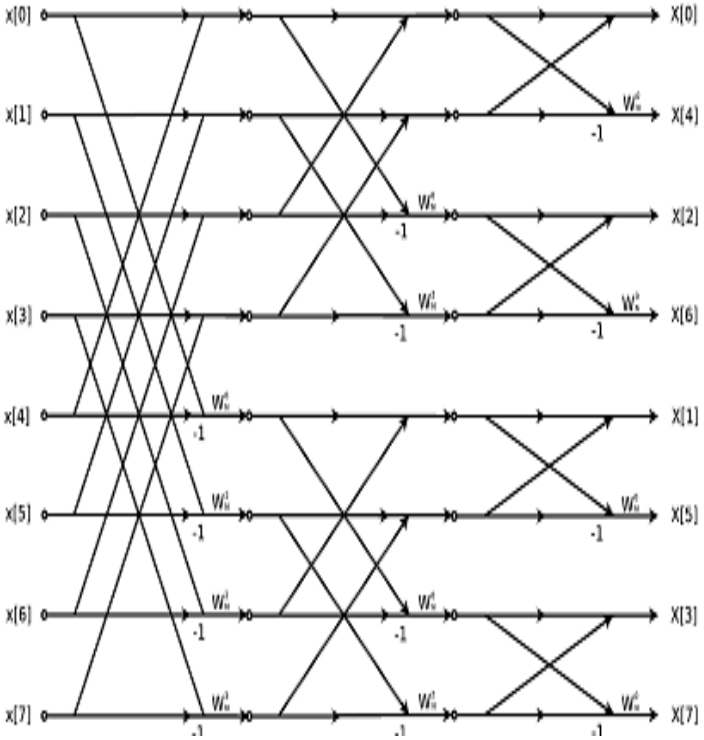

Fig.9. Butterfly model of Fast Fourier transform

From the above Fig.9. thesis can be taken in 3 stages. The Fast Fourier transform consists of multiplication and addition. The schematic of the butterfly consists of a complex multiplier, complex adder and complex subtractor.The butterfly operation processor section performs the butterfly operation, with each 8-bit input data width. A complete butterfly operation requires complex multiplier, complex adder and complex subtractor.This consists of four real multipliers, three real adders and three real subtractors. In butterfly operation when first cycle is finished then the first result bit of each real multiplication is ready, then the additions and subtractions are operated. This takes a long time to execute. So for that reason we present a Twin precision technique. In this the multiplication is done parallely that for example if we want to execute 64-bit operation we can execute the two 8-bit multipliers parellaly. By using this we can get high throughput. It is highly efficient, low power, high speed. The coding is done by using vhdl and simulation is done in Xilinx ISE simulator. Multiplications in above block diagram are performed by using twin precision technique. Implies, efficient utilization can be done throughout entire process.

In this paper we present Fast Fourier transform implementation using Twin precision technique. We select Baugh –Wooley because compared to Booth algorithm and Modified Booth algorithms [7] it is highly efficient and high speed. It can be done in less amount of time. So by this algoritm and the Twin precision technique we can get high throughput which is highly efficient, low power, high speed.

IV. SIMULATION RESULTS

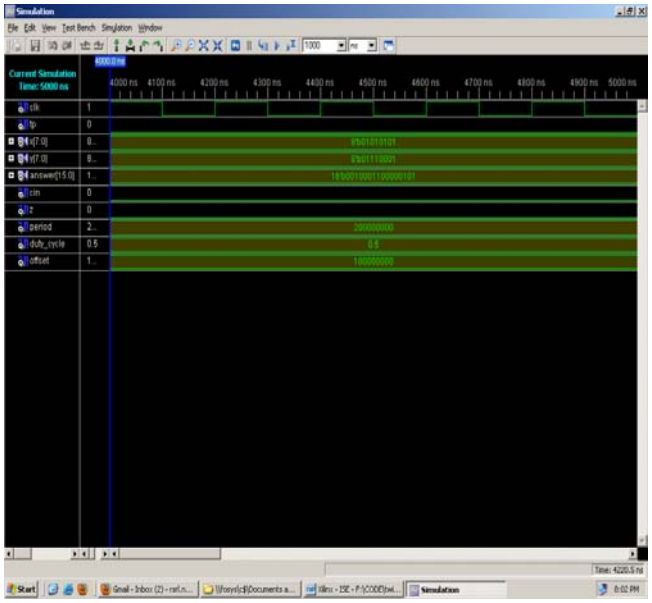

Fig. 10.Simulation Results of Baugh-Wooley using Twin Precision

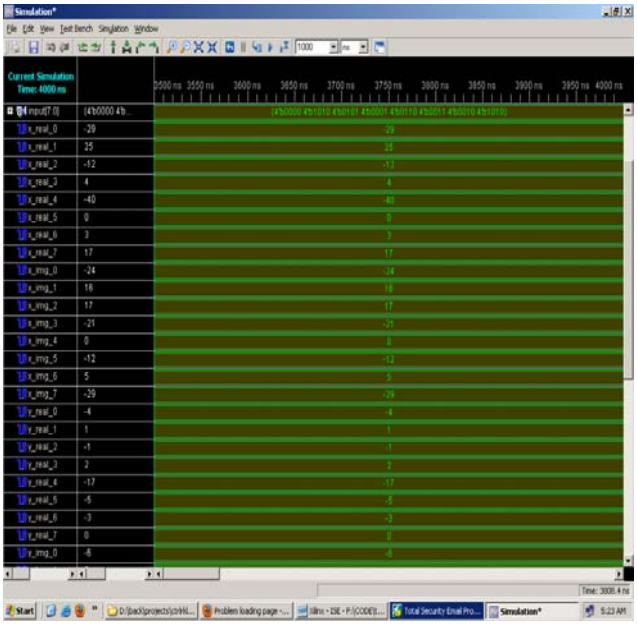

Fig. 11.Simulation Results of Fast Fourier Transform using Twin Precision technique and Baugh-Wooley Algorithm.

Here Fig. 10. shows the result for Baugh-Wooley algorithm using Twin precision technique. Here in the main aim of this technique is to get the result in less amount of time. This reduces the number of multipliers. In Fig.10 two 8-bits has been taken and the result is 16-bit.This result is divided into two 8-bits in less amount time.

In Fig.11.this shows the result of Fast Fourier Transform using Twin Precision technique.Here the algorithm used is Baugh-Wooley. We used Baugh-Wooley algorithm Because this is very efficient algorithm when compared to General multipliers,Booth algorithm and Modified Booth algorithms. This Baugh-Wooley algorithm with Twin precision technique is the best for the multipliers to execute in less amount of time. In this we generate the Baugh-Wooley algorithm using Twin precision and Fast Fourier transform using Twin precision.

V. CONCLUSIONS

The Fast Fourier transform implementation using Twin precision technique has been implemented. The twin-precision technique, which offers flexibility at a low implementation overhead, makes it possible to efficiently deploy these flexible architectures. We present the Baugh-Wooley algorithm. This makes the multiplier to run fastly that is high throughput, high speed, high efficient. It is executed in less amount of time. The main applications are DSP processors, Communication applications etc. By adapting to actual multiplication bit-width using twin precision technique, it is possible to save power, increase speed, double computation throughput and highly efficient. By using this execution time of a Fast Fourier transform is reduced with 15% at a 14% reduction in datapath energy dissipation.

REFERENCES

[1] G.Venkataramana Sagar and Dr.K.Srinivasa Rao "Reconfigurable FFT System On Chip (SOC) in International Journal of Computer Applications (0975-8887) Volume 11-No.5,December 2010.

[2] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Comput*, vol. 19, pp. 297–301, 1965.

[3] M. Själander, H. Eriksson, and P. Larsson-Edefors, "An efficient twin precision multiplier," in *Proc. 22nd IEEE Int. Conf. Comput. Des.*, Oct.

[4] C Magnus Själander and Per Larsson-Edefors," Multiplication Acceleration Through Twin Precision". In IEEE Transactions on VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 17, NO. 9, SEPTEMBER 2009.

[5] R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. Comput.*, vol. 22, pp. 1045–1047, Dec. 1973.

[6] M.Hatamain, " A 70MHZ 8 bit*8 bit parallel pipelined multiplier in 2.5µm CMOS," *IEEE Solid state circuits*, vol 21, no.4, pp.505- 513, Aug.1986.

[7] A. D. Booth, "A signed binary multiplication technique," *Quarterly J. Mechan. Appl. Math.*, vol. 4, no. 2, pp. 236–240, 1951.

[8] Xilinx, Inc. Xilinx Libraries Guide, 1999