FeedForward FFT Hardware Architecture Based On Rotator Allocation Using Modified Booth Algorithm

[1]N.Yamini Swetha ,[2]Y.Rakesh
[1]M.Tech scholar, [2]Associate professor
[1,2]UshaRama College of Engineering &Technology, Vijayawada, India

Abstract: The Fast Fourier transform (FFT) is one of the most important algorithms in the field of Digital signal processing. It is used to calculate the Discrete Fourier transform (DFT) efficiently. In order to meet the high performance and real time requirements of modern applications, hardware designers have always tried to implement efficient architectures for the computation of the FFT. A new feedforward FFT hardware architectures based on rotator allocation is presented. It consists in finding an efficient distribution of FFT rotations that reduces the number of rotators and their complexity. Radix-2 feedforward architectures based on rotator allocation are presented along with MDC methodology. In rotators, general multipliers and general adders are used for implementation. The Booth algorithm consists of repeatedly adding one of two predetermined values to a product P and then performing an arithmetic shift to the right on P. The multiplier architecture consists of two architectures, i.e., Modified Booth and Booth algorithm. By the study of different multiplier architectures, we find that Modified Booth increases the speed because it reduces the partial products by half. Also, the delay in the multiplier can be reduced by using Wallace tree. The energy consumption of the Wallace Tree Multiplier is also lower than the Booth and the array. The characteristics of the two multipliers can be combined to produce a high-speed and low-power multiplier. The modified stand-alone multiplier consists of a modified recorder (MBR). MBR has two parts, i.e., Booth Encoder (BE) and Booth Selector (BS).

Index Terms - Fast Fourier transform (FFT), discrete Fourier transform (DFT), a Modified recorder (MBR), Booth Encoder (BE), Butter fly unit (BU).

1. INTRODUCTION

The Fast Fourier transform (FFT) is one of the most important algorithms in the field of digital signal processing. It is used to calculate the discrete Fourier transform (DFT) efficiently. In order to meet the high performance and real-time requirements of modern applications, hardware designers have always tried to implement efficient architectures for the computation of the FFT. In this context, pipelined hardware architectures[1][3] [4] are widely used, because they provide high throughputs and low latencies suitable for real time, as well as a reasonably low area and power consumption. There are two main types of pipelined architectures: feedback (FB) and feedforward (FF)[6]. On the one hand, feedback architectures are characterized by their feedback loops, i.e., some outputs of the butterflies are fed back to the memories at the same stage. Feedback architectures can be divided into single-path delay feedback (SDF)[2][9] , which process a continuous flow of one sample per clock cycle, and multi-path delay feedback (MDF) or parallel feedback, which process several samples in parallel. On the other hand, feedforward architectures also known as multi-path delay commutator (MDC), do not have feedback loops and each stage passes the processed data to the next stage. These architectures can also process several samples in parallel. In current real-time applications, the FFT has to be calculated at very high throughput rates, even in the range of Giga samples per second. These high-performance requirements appear in applications such as orthogonal frequency division multiplexing (OFDM and ultra wideband (UWB). In this context two main challenges can be distinguished. The first one is to calculate the FFT of multiple independent data sequences. In this case, all the FFT processors can share the rotation memory in order to reduce the hardware. Designs that manage a variable number of sequences can also be obtained. The second challenge is to calculate the FFT when several samples of the same sequence are received in parallel. This must be done when the required throughput is higher than the clock frequency of the device. In this case it is necessary to resort to FFT architectures that can manage several samples in parallel. As a result, parallel feedback architectures, which had not been considered for several decades, have become very popular in the last few years. Conversely, not very much attention has been paid to feedforward (MDC) architectures.

2. FAST FOURIER TRANSFORM

FFT circuitry consists of several consecutive multipliers and adders over complex numbers. Until recently, most FFT architectures used fixed-point arithmetic only, before FFTs based on floating-point operations became prominent. Using the IEEE-754-2008 standard for floating-point arithmetic allows FFT co-processors to collaborate with general purpose processors. Despite the fact that binary computer arithmetic improves processing speed and reduces hardware complexity, decimal computer arithmetic has recently been revived. The advantage of decimal computer arithmetic over its binary counterpart is that decimal arithmetic is capable of mirroring human computations (i.e., radix-10) and representing fractions precisely where binary 2 cannot (e.g., 0.2). Some applications, such as finance and banking, cannot tolerate a loss of precision; this is where decimal computer arithmetic is useful.

Decimal computer arithmetic can be implemented in hardware or software. The software implementation of decimal arithmetic operations with binary logic devices was widely used until IBM revealed an all-hardware implementation of decimal processors such as the POWER6 decimal processor. Additionally, the IEEE 754-2008 standard for floating-point arithmetic now supports the decimal hardware implementation. Hardware decimal arithmetic is used where high-speed computations are performed on large amounts of data.

2.1 Fast Fourier Transform (FFT)

In present several methods are present for computing the DFT efficiently. In view of the importance of the DFT in various digital signal processing applications, such as linear filtering, correlation analysis, and spectrum analysis, its efficient computation is a topic that has received considerable attention by many mathematicians, engineers, and applied scientists. From this point, we change the notation that *X (k)*, instead of *y (k)* in previous sections, represents the Fourier coefficients of *x (n)*. Basically, the computational problem for the DFT is to compute the sequence {*X* (*k*)} of *N* complex-valued numbers given another sequence of data {*x* (*n*)} of length *N*, according to the formula in equation 1&2.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad 0 \le k \le N-1$$
$$W_N = e^{-j2\pi/N} \qquad (1)$$

In general, the data sequence *x*(*n*) is also assumed to be complex valued. Similarly, The IDFT becomes

$$x(n) = \frac{1}{N}\sum_{n=0}^{N-1} X(k) W_N^{-nk}, \qquad 0 \le n \le N-1 \qquad (2)$$

Since DFT and IDFT involve basically the same type of computations, our discussion of efficient computational algorithms for the DFT applies as well to the efficient computation of the IDFT.

2.2. Radix-2 FFT Algorithm

Consider the computation of the $N = 2^v$ point DFT by the divide-and conquer approach. We split the *N*-point data sequence into two *N*/2-point data sequences $f_1(n)$ and $f_2(n)$, corresponding to the even-numbered and odd-numbered samples of *x*(*n*), respectively, that is represented in equation (3)

$$f_1(n) = x(2n)$$
$$f_2(n) = x(2n+1), \qquad n = 0,1,\ldots,N/2-1 \qquad (3)$$

Thus $f_1(n)$ and $f_2(n)$ are obtained by decimating *x*(*n*) by a factor of 2, and hence the resulting FFT algorithm[6][7] is called a *decimation in time algorithm.*

Now the N-point DFT can be expressed in terms of the DFT's of the decimated sequences as follows in equation (4):

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad k = 0,1,\ldots,N-1$$
$$= \sum_{n\ \text{even}} x(n) W_N^{kn} + \sum_{n\ \text{odd}} x(n) W_N^{kn}$$
$$= \sum_{m=0}^{(N/2)-1} x(2m) W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1) W_N^{k(2m+1)} \qquad (4)$$

But $W_N^2 = W_{N/2}$. With this substitution, the equation can be expressed as

$$X(k) = \sum_{m=0}^{(N/2)-1} f_1(m) W_{N/2}^{km} + W_N^k \sum_{m=0}^{(N/2)-1} f_2(m) W_{N/2}^{km}$$
$$= F_1(k) + W_N^k F_2(k), \qquad k = 0,1,\ldots,N-1 \qquad (5)$$

Where $F_1(k)$ and $F_2(k)$ are the *N*/2-point DFTs of the sequences $f_1(m)$ and $f_2(m)$, respectively in equation (5). Since $F_1(k)$ and $F_2(k)$ are periodic, with period *N*/2, we have $F_1(k+N/2) = F_1(k)$ and $F_2(k+N/2) = F_2(k)$. In addition, the factor $W_N^{k+N/2} = -W_N^k$. Hence the equation may be expressed as shown in below equation (6).

$$X(k) = F_1(k) + W_N^k F_2(k), \qquad k = 0,1,\ldots,\frac{N}{2}-1$$
$$X(k+\frac{N}{2}) = F_1(k) - W_N^k F_2(k), \qquad k = 0,1,\ldots,\frac{N}{2}-1 \qquad (6)$$

3. FFT Hardware Architecture

In order to design FFT hardware architecture, we have to be aware of the FFT properties. The first property, which is general for any FFT architecture and any N, is that at any FFT stage, butterflies operate on data whose index I differ in bn−s, where n is the number of FFT stages and s is the specific stage that we are considering. This fact can be observed in the flow graph of Fig. 1. In this flow graph, the index has n = 4 bits, i.e., I ≡ b3 b2 b1 b0. At the first stage, the butterflies operate on samples whose indexes differ in bn−s = b4−1 = b3. This happens for samples with indexes 0 and 8, 1 and 9, etc. For the second stage, the different bit is bn−s = b4−2 = b2. Note for instance, that the data with indexes 0 and 4 are operated together in the butterfly at stage 2. For the third and fourth stages, the corresponding bits are b1 and b0, respectively. Therefore, if we want to design an FFT hardware architecture, we have to assure that at each stage s, the indexes of the inputs to any butterfly at any time instant differ in bn−s. Note that the term butterfly refers now to a hardware component of the architecture, not to the mathematical operation of the algorithm in the flow graph. We observe that this property is fulfilled at all the stages. In this figure bn−s is at the lowest parallel dimension, which corresponds to the pair of samples that flow into the butterflies at the same clock cycle. The property of bn−s is the only requirement set by the butterflies in FFT hardware architecture. As long as this property is met, we can have any data order at the different FFT stages. This allows for exploring a variety of data orders at the FFT stages. This is what is done in the current paper, as explained later in Section V. The second FFT property refers to the rotations at the FFT stages. At each stage, any sample with index I must be rotated according to equation by a value φ that depends on the index I and on the stage, s. We can represent it as φs(I). The work explains how to calculate φs(I), which only depends on the FFT algorithm that is used. By combining these ideas, we lead to the following conclusions: On the one hand, any order at any stage of any FFT hardware architecture is possible as long as the property of bn−s is met at the input of the butterflies. On the other hand, to any index I at any FFT stage corresponds a specific rotation φs(I). As a result, we can play with the data order at different FFT stages to look for patterns that allow for a more optimized distribution of rotations. This is the idea behind the proposed rotator allocation approach.

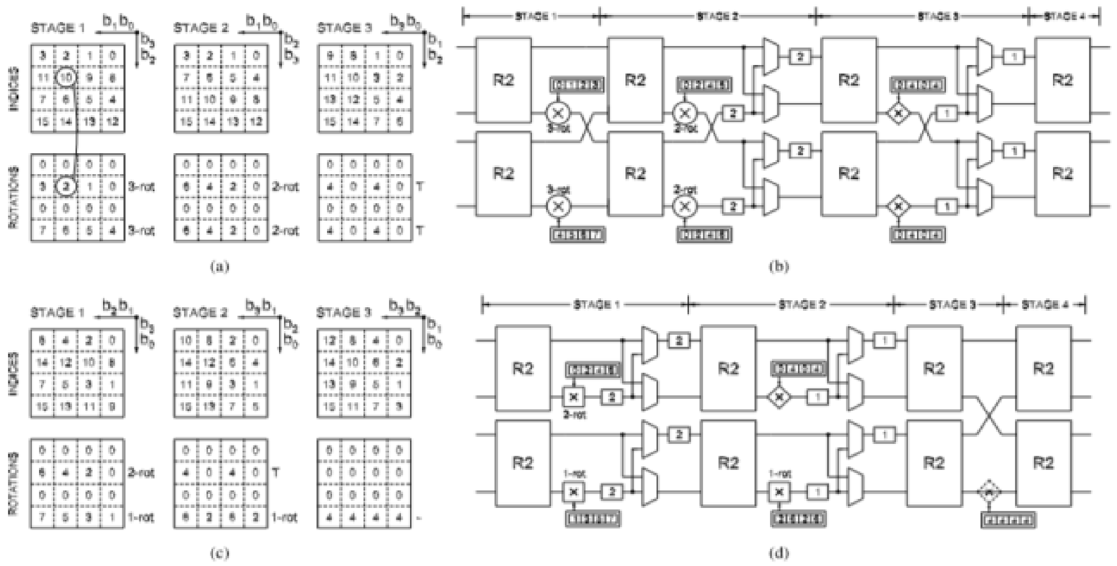


Fig 1. Rotator allocation of a 16-point 4-parallel FFT.
(a) Layout not aware of the rotator allocation
(b) FFT architecture for the layout in Fig. 1(a)
(c) Layout aware of the rotator allocation.
(d) FFT architecture for the layout in Fig. 1(c).

The purpose of the FFT design using rotator allocation is to distribute the rotations of the FFT in such a way that the number and complexity of the required rotators is reduced. Fig. 1(a) shows an example of a layout for the first three stages of a 16-point 4-parallel FFT. The indexes in the figure show how data flows at the different stages. Each element in the matrices of indexes is the index value according to Fig. 1. Data flows from left to right. Thus, values in the same column are data that flow in parallel and values in the same row flow through the same path in consecutive clock cycles. The matrices of rotations show the value φs(I) that corresponds to each of the indexes according to the flow graph in Fig. 1. For instance, the rotation corresponding to the index 10 at stage 1 is φs(I) = φ1(10) = 2 according to Fig. 1. This case is highlighted in Fig. 1(a). As for the indexes, each column in the matrices of rotations is rotations that are calculated in parallel at the same clock cycle, whereas rotations in the same row are calculated in consecutive clock cycles by the same rotator. According to this, each row of the matrices of rotations are the rotations that must be calculated by a single rotator in consecutive clock cycles. For instance, stage 1 needs a rotator by φ = {0, 1, 2, 3} and a rotator by φ = {4, 5, 6, 7}, which are 3-rots according to Table I. Stage 2 includes 2 2-rots and stage 3 includes 2 trivial rotators (T). The layout in Fig. 1(a) translates into the FFT architecture in Fig. 1(b), which shows the rotators at each stage, as well as the content of the rotation memories. By applying rotator allocation we aim to reduce the number of rotators and their complexity. Rotator allocation simply consists of reorganizing the matrices of indexes and, therefore, the matrices of rotations, in such a way that the matrices of rotations have less rotator (if possible) and the complexity of the rotators is lower. On the one hand, fewer rotators are achieved when there are more rows in the matrices of rotations whose elements are φ = 0. On the other hand, the complexity of the rotators is reduced when the rotations in the same row are in less SAS. The procedure of rotator allocation consists in distributing the bits bn−1 ... b0 of the index I into serial and parallel dimensions. Serial dimensions correspond to data arriving at the same input terminal in series and parallel dimensions refer to data arriving at parallel terminals. Depending on the FFT size N = 2n and the number of parallel data in the FFT P = 2p, the number of serial bits is n−p = log2(N)−log2(P) and the number of parallel ones is p = log2(P). For the example , N = 16 and P = 4, so there are n − p = log2(16) − log2(4) = 2 serial dimensions and p = log2(P) = 2 parallel ones. The alternatives to allocate the bits correspond to all the possible permutations, i.e., Serial Parallel

b3b2|b1b0 b2b3|b1b0 b3b2|b0b1 b2b3|b0b1 b3b1|b2b0 b1b3|b2b0 b3b1|b0b2 b1b3|b0b2 b3b0|b2b1 b0b3|b2b1 b3b0|b1b2 b0b3|b1b2 b0b1|b2b3 b1b0|b2b3 b0b1|b3b2 b1b0|b3b2 b0b2|b1b3 b2b0|b1b3 b0b2|b3b1 b2b0|b3b1 b1b2|b0b3 b2b1|b0b3 b1b2|b3b0 b2b1|b3b0

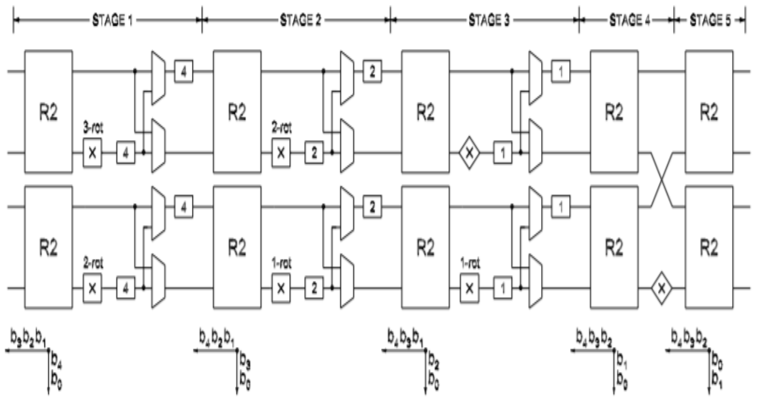


Fig. 2. Proposed 32-point 4-parallel radix-2 MDC DIF FFT architecture.

Fig. 2 represents the proposed 32-point 4-parallel radix-2 MDC DIF FFT architecture. The MDC FFT Architecture consists of 5 stages. A different order of the serial bits changes the order of the rotations, but the same rotations are calculated by the rotators. A different order of the parallel bits changes the edges in which the rotators are placed, but the rotators are the same. Therefore, the alternatives in each row have the same complexity, so only one alternative per row needs to be evaluated. According to this, the number of alternatives that need to be evaluated at each FFT stage.

4. RADIX-8 MODIFIED BOOTH ALGORITHM

The Booth algorithm consists of repeatedly adding one of two predetermined values to a product P and then performing an arithmetic shift to the right on P.

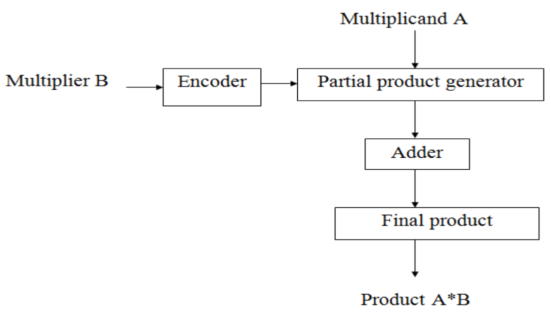


Fig 3. Booth algorithm

Fig.3 shows the block diagram of booth algorithm. The Booth algorithm consists of encoder, partial product generator ,adder And result to store i.e. final product. The multiplier architecture consists of two architectures, i.e., Modified Booth and Booth algorithm. By the study of different multiplier architectures, we find that Modified Booth increases the speed because it reduces the partial products by half. Also, the delay in the multiplier can be reduced by using Wallace tree. The energy consumption of the Wallace Tree multiplier is also lower than the Booth and the array[5][7][8]. The characteristics of the two multipliers can be combined to produce a high-speed and low-power multiplier.

The modified stand-alone multiplier consists of a modified recorder (MBR). MBR has two parts, i.e., Booth Encoder (BE) and Booth Selector (BS). The operation of BE is to decode the multiplier signal, and the output is used by BS to produce the partial product. Then, the partial products are added to the Wallace tree adders, similar to the carry-save-adder approach. The last transfer and sum output line are added by a carry look- ahead adder, the carry being stretched to the left by positioning.

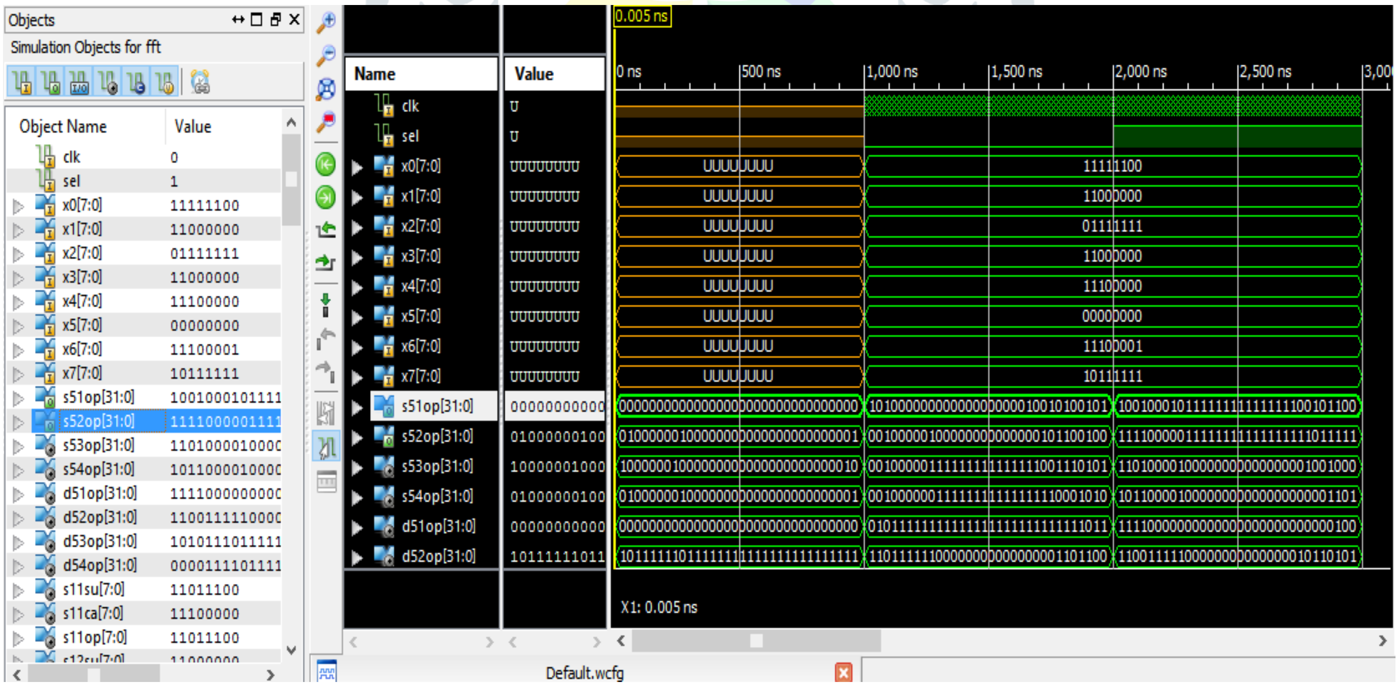
Table. 1. Quartet coded signed-digit table

Quartet value	Signed-digit value
0000	0
0001	+1
0010	+1
0011	+2
0100	+2
0101	+3
0110	+3
0111	+4
1000	-4
1001	-3
1010	-3
1011	-2
1100	-2
1101	-1
1110	-1
1111	0

Here we have a multiplication multiplier, 3Y, which is not immediately available. To Generate it, we must run the previous addition operation: 2Y + Y = 3Y. But we are designing a multiplier for specific purposes and then the multiplier belongs to a set of previously known numbers stored in a memory chip. We have tried to take advantage of this fact, to relieve the radix-8 bottleneck, that is, 3Y generation. In this way, we try to obtain a better overall multiplication time or at least comparable to the time, we can obtain using radix-4 architecture (with the added benefit of using fewer transistors). To generate 3Y with 21-bit words you just have to add 2Y + Y i.e. add the number with the same number moved to a left position.

A product formed by multiplying it with a multiplier digit when the multiplier has many digits. Partial products are calculated as intermediate steps in the calculation of larger products .The partial product generator is designed to produce the product multiplying by multiplying A by 0, 1, -1, 2, -2, -3, -4, 3, 4. Multiply by zero implies that the product is "0 ". Multiply by" 1 "means that the product remains the same as the multiplier. Multiply by "-1" means that the product is the complementary form of the number of two. Multiplying with "-2" is to move left one as this rest as per Table.1

5. RESULT



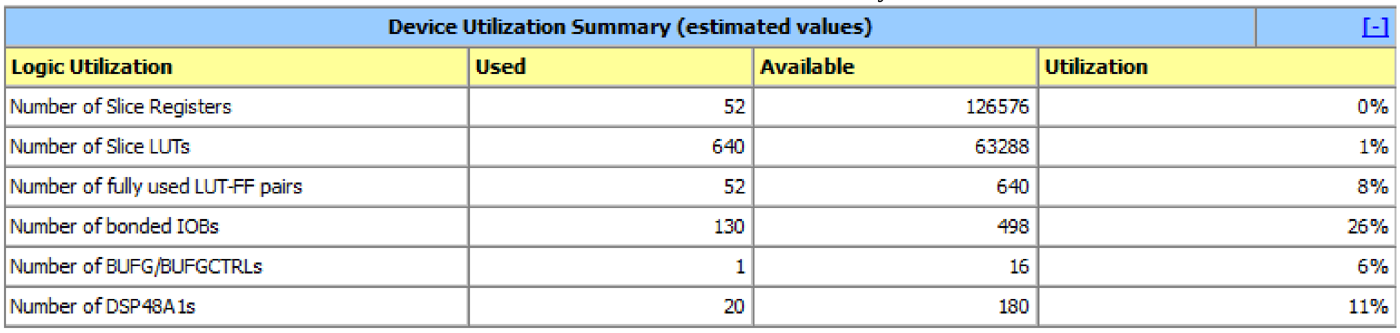
Object Name	Value
clk	0
sel	1
x0[7:0]	11111100
x1[7:0]	11000000
x2[7:0]	01111111
x3[7:0]	11000000
x4[7:0]	11100000
x5[7:0]	00000000
x6[7:0]	11100001
x7[7:0]	10111111
s51op[31:0]	1001000101111
s52op[31:0]	1111000001111
s53op[31:0]	1101000010000
s54op[31:0]	1011000010000
d51op[31:0]	1111000000000
d52op[31:0]	1100111110000
d53op[31:0]	1010111011111
d54op[31:0]	0000111101111
s11su[7:0]	11011100
s11ca[7:0]	11100000
s11op[7:0]	11011100

```
Name          Value
clk           U
sel           U
x0[7:0]       UUUUUUUU   UUUUUUUU                          11111100
x1[7:0]       UUUUUUUU   UUUUUUUU                          11000000
x2[7:0]       UUUUUUUU   UUUUUUUU                          01111111
x3[7:0]       UUUUUUUU   UUUUUUUU                          11000000
x4[7:0]       UUUUUUUU   UUUUUUUU                          11100000
x5[7:0]       UUUUUUUU   UUUUUUUU                          00000000
x6[7:0]       UUUUUUUU   UUUUUUUU                          11100001
x7[7:0]       UUUUUUUU   UUUUUUUU                          10111111
s51op[31:0]   00000000000 00000000000000000000000000000000 10100000000000000000010010100101 10010001011111111111111100101100
s52op[31:0]   01000000100 01000000100000000000000000000001 00100000100000000000000101100100 11110000011111111111111111011111
s53op[31:0]   10000001000 10000001000000000000000000000010 00100000111111111111111001110101 11010000100000000000000001001000
s54op[31:0]   01000000100 01000000100000000000000000000001 00100000011111111111111110001010 10110000100000000000000000001101
d51op[31:0]   00000000000 00000000000000000000000000000000 01011111111111111111111111111011 11110000000000000000000000000100
d52op[31:0]   10111111011 10111111011111111111111111111111 11011111100000000000000001101100 11001111100000000000000010110101
```

Fig.4 Simulation output

Fig.4 shows the simulation output of modified booth algorithm. The inputs are x1, x2,x3,x4,x4,x5,x6,x7 which are 7 down 0 [7:0]. The outputs are s51 and s52 which are 31 down to 0[31:0].

Table.2 Device utilization summary



Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	52	126576	0%
Number of Slice LUTs	640	63288	1%
Number of fully used LUT-FF pairs	52	640	8%
Number of bonded IOBs	130	498	26%
Number of BUFG/BUFGCTRLs	1	16	6%
Number of DSP48A1s	20	180	11%

Table.2 shows the device utilization summary of booth algorithm. The table consists of number of slice registers, number of slice LUTs, number of fully used LUT-FF Pairs, number of bonded IOBs, number of BUFG/BUFGCTRLs, number of DSP481s that are available and used.

6. CONCLUSION

Finally this paper presents an FFT rotation that reduces the number of rotators and their complexity. This leads to new radix-2 and radix-2k MDC FFT architectures. These architectures require the same memory and butterflies as previous MDC FFTs, but fewer and/or simpler rotators. Modified booth encoding insertion plays vital role to reduce parameters like area and time.

7. REFERENCES

[1] M. Garrido, J. Grajal, M. A. Sánchez, and O. Gustafsson, "Pipelined radix-2 k feedforward FFT architectures," IEEE Trans. VLSI Syst., vol. 21, no. 1, pp. 23–32, Jan. 2013.
[2] J.Wang, C.Xiong, K.Zhang, and J.Wei, ''A mixed –decimation MDF architecture for raix-2^k parallel FFT,'' ''IEEE Trans. Very Large Scale Integer(VLSI) Syst., vol. 24 , no. 1 ,pp.67-68, Jan. 2016
[3] S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, "A 2.4-GS/s FFT processor for OFDM-based WPAN applications," IEEE Trans. Circuits Syst. II, vol. 57, no. 6, pp. 451–455, Jun. 2010.
[4] C.-H. Yang, T.-H. Yu, and D. Markovic, "Power and area minimization of reconfigurable FFT processors: A 3GPP-LTE example," IEEE J. Solid-State Circuits, vol. 47, no. 3, pp. 757–768, Mar. 2012.
[5] W. Han, A. T. Erdogan, T. Arslan, and M. Hasan, "High-performance low-power FFT cores," ETRI J., vol. 30, no. 3, pp. 451–460, Jun. 2008.
[6] H. Liu and H. Lee, "A high performance four-parallel 128/64-point radix-2 4 FFT/IFFT processor for MIMO-OFDM systems," in Proc. IEEE Asia-Pacific Conf. Circuits Syst., Nov. 2008, pp. 834–837.
[7] J.-Y. Oh and M.-S. Lim, "New radix-2 to the 4th power pipeline FFT processor," IEICE Trans. Electron., vol. E88-C, no. 8, pp. 1740–1746, Aug. 2005.
[8] Z. Wu, J. Sha, Z. Wang, L. Li, and M. Gao, "An improved scaled DCT architecture," IEEE Trans. Consum. Electron. vol. 55, no. 2, pp. 685– 689, May 2009.
[9] M.Garrido, R.Anderson, F.Qureshi and O.Gustafsson, ''Multiplier less unity-gain SDF FFTs,''IEEE Trans. Very Large Scale Integer(VLSI) Syst., vol. 24 , no. 9 ,pp. 3003-3007, sep.2016