An Efficient 4-Parallel Feed Forward FFT Architecture by using Multipath Delay Commutator

D.Nirmala[1], D.Linett Sophia[2]
PG Scholar[1], Assistant Professor[2],
St.Peter's University, TN, India

Abstract –Fast Fourier Transform (FFT) is widely used in the field of digital signal processing (DSP) such as filtering spectral analysis, etc, to compute discrete fourier transform. The appearance of radix-2^2 which had been a milestone in the design of pipelined FFT hardware architectures was later extended to radix-2^K.This project presents a new approach to develop parallel pipelined 16 point radix-2^2. Feed forward (MDC) FFT architectures along with data shuffling. Furthermore, decimation in frequency (DIF) decomposition has been used. The power and area consumption can be reduced in 4-parallel FFT architecture. As a result, the proposed radix-2^2 feed forward architectures not only offer an attractive solution for current applications, but also open up new research line on feed forward structures. The output samples are obtained, to a desired order to implement on XILINXSPARTAN-3E FPGA.

Index Terms — Fast Fourier Transform (FFT), multipath delay commutator (MDC), pipelined architecture, radix-2^k, FPGA.

1. INTRODUCTION

The present day real time applications demand, the FFT to be calculated at very high throughput rates, These high performance requirements appear in applications such as millimeter Wave wireless local area network , wireless personal area network systems and real time video streaming services in short range indoor environments. The FFT/IFFT processor has a high hardware complexity in the OFDM modulation of high rate WPAN systems. One OFDM symbol in the IEEE 802.11ad standards consists of a length of 512 subcarriers. Therefore, FFT processor conducts the FFT computation with 512-point arithmetic and should provide a high throughput rate of at least 2.115 GS/s [1]. The radix of the algorithm greatly influences the architecture of the FFT processor and the complexity of the implementation. A small radix is desirable because it results in a simple butterfly. Nevertheless, a high radix reduces the number of twiddle factor multiplications. The radix 2^k algorithms simultaneously achieve a simple butterfly and a reduced number of twiddle factor multiplications.

The radix-2 algorithm is a well-known simple algorithm for FFT processors, but it requires many complex Multipliers. Recently various radix 2^k FFT algorithms and architectures have been studied in order to reduce the number of complex multipliers.in this brief, a reconfigurable FFT architecture to compute 512-point FFT using radix-2^k algorithm with the high-speed is proposed. The key ideas for achieving high date throughput, reduced hardware complexity are described .The power consumption and hardware cost can be saved in our processor by using the higher radix FFT algorithm and less memory and complex multipliers.

2. RADIX -2^K ALGORITHM

The N-point DFT is formulated as

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}, k=0,1,...N-1 \quad (1)$$

Where the twiddle factors is defined as $W_N^{nk}=e^{-\frac{2\pi nk}{N}}$ .The n denotes the time index and the k denotes the frequency index. The radix 2^k algorithm can be derived by integrating twiddle factor decomposition through a divide and conquer approach [2].

2.1 Radix -2^2 Algorithm

Consider the first two steps of decomposition in radix-2 DIF FFT together. Applying a 3-dimensional linear index map as follows

$$n=\frac{N}{2}n_1+\frac{N}{4}n_2+n_3\{n_1,n_2=0,1 n_3=0\sim\frac{N}{4}-1\} \quad (2)$$
$$k=k_1+2k_2+4k_3\{k_1,k_2=0,1 k_3=0\sim\frac{N}{4}-1\}$$

The DFT has the form of

$$X(k_1+2k_2+4k_3)=\sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1}x(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)W_N^{nk}$$

$$=\sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\{B_{\frac{N}{2}}^{k_1}(\frac{N}{4}n_2+n_3)\}W_N^{(\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)} \quad (3)$$

Where the first butterfly structure has the form of

$$B_{\frac{N}{2}}^{k_1}(\frac{N}{4}n_2+n_3)=x(\frac{N}{4}n_2+n_3)+(-1)^{k_1}x(\frac{N}{4}n_2+n_3+\frac{N}{2}) \quad (4)$$

Decomposing the composite twiddle factor, it can be expressed in Eq.(5).

$$W_N^{(\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)}=(-j)^{n_2(k_1+2k_2)}W_N^{n_3(k_1+2k_2)}W_{N/4}^{n_3k_3} \quad (5)$$

Substituting the Eq.(5) into Eq.(3) and expanding the summation with regard to index n_2 ,we have a set of 4 DFTs of length $N/4$.

$$X(k_1+2k_2+4k_3)=\sum_{n_3=0}^{\frac{N}{4}-1}[H_{N/4}^{k_1k_2}(n_3)W_N^{n_3(k_1+2k_2)}]W_{N/4}^{n_3k_3} \quad (6)$$

Where a secondary butterfly structure $H_{N/4}^{k_1k_2}(n_3)$ is expressed as

$$H_{N/4}^{k_1k_2}(n_3)=B_{\frac{N}{2}}^{k_1}(n_3)+(-1)^{k_2}(-j)^{k_1}B_{\frac{N}{2}}^{k_1}(n_3+\frac{N}{4}) \quad (7)$$

After these two columns, full multiplications are used to apply the decomposed twiddle factor $W_N^{n_3(k_1+2k_2)}$ in Eq.(6).Applying this cascade decomposition recursively to the remaining DFTs of length $N/4$ in Eq.(6), the complete radix 2^2 FFT algorithm is obtained. Equation (7) represents the first two columns of butterflies with only trivial multiplication of (-j) which can be implemented using only real-imaginary swapping and sign inversion. The radix-2^2 algorithm is characterized according to the merit that it has the same multiplicative complexity and as the radix-4 algorithm, but still retains simple structures of the radix-2 butterfly.

3. RADIX-2^2 DIF-FFT

After these two stages, full multipliers are required to compute the product of decomposed twiddle factors. The complete radix-2^2 algorithm can be derived by applying recursively.Fig.1 shows the flow graph of an N = 16 point FFT decomposed according to decimation in frequency (DIF). The numbers at the inputs and outputs of the graph represent the index of input and output samples, respectively. The advantage of the algorithm is that it has the same multiplicative complexity as radix-4 algorithms, but still retains the radix-2 butterfly structures. We can observe that, only every other stage of the flow graph has non-trivial multiplications. The *–j* notion represents the trivial multiplication, which involves only real-imaginary swapping and sign inversion [3].

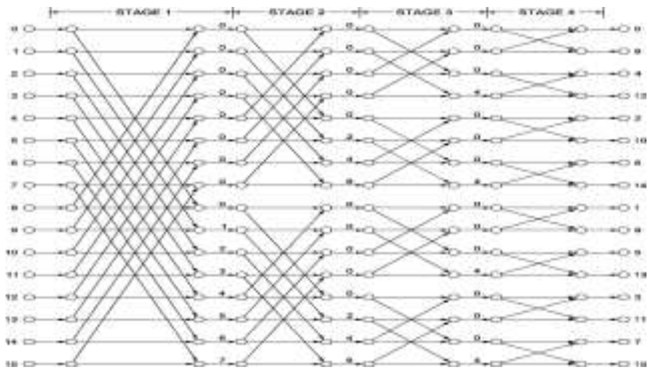


Fig 1: 16 Point radix2^2 DIF-FFT architecture

3.1 FEEDFORWARD ARCHITECTURE

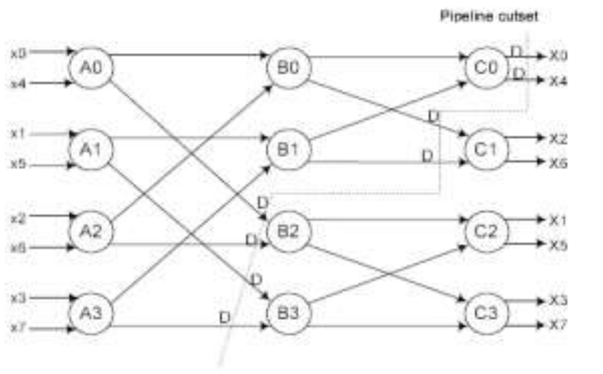


Fig 2: Pipelined Data Flow graph (DFG) of a 8-point DIF FFT as a pre-processing step for folding

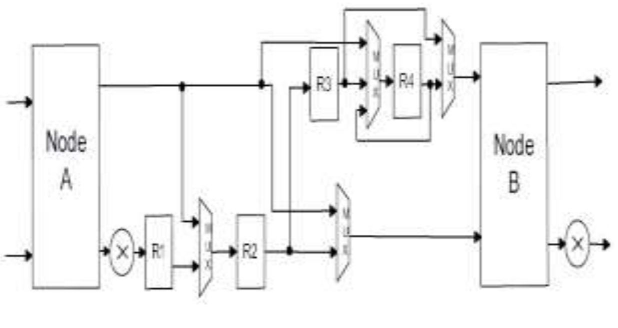


Fig 3.: Folded circuit between Node A and Node B

We derive the feed forward architecture using the same 8-point radix-2 DIT FFT example in Fig 3.

A = (φ, φ, φ, φ, A0, A1, A2, A3)

B = (B_2, B3, φ, φ, φ, φ, B0, B1)

C = (C1, C2, C3, φ, φ, φ, φ, C0)

The control complexity of the derived circuit is high. Four different signals are needed to control the multiplexers. A better register allocation is found such that the number of multiplexers is reduced in the final architecture. The more optimized register allocation for the same lifetime analysis chart is shown in Fig.10. Similarly, we can apply lifetime analysis and register allocation techniques for the variables x0, ..., x7 and z0, ..., z7, inputs to the DFG and the outputs from nodes B0,B1,B2,B3 respectively[4]. We can observe that the derived architecture is the same as R2MDC architecture. Similarly, the R2SDF architecture can be derived by minimizing the registers on all the variables at once. The linear lifetime chart for these variables is shown in Fig 3.

3.3 4-PARALLEL RADIX-2^2 FEED FORWARD ARCHITECTURE

The feed forward 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.

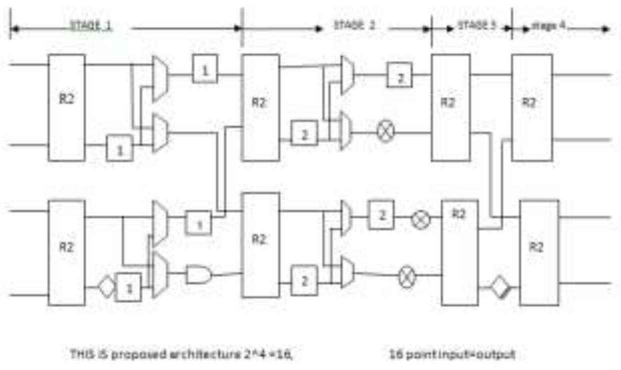
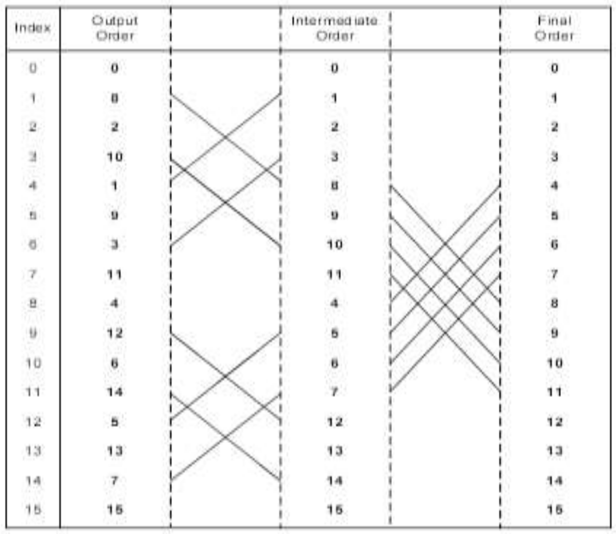


Fig 4: Proposed 4-parallel (Architecture 2) for the computation of 16-point radix-2 DIF FFT

The Radix-2^k was presented for the SDF FFT as an improvement on radix-2 and radix-4. Next, radix-2^3 and radix-2^4, which enable certain complex multipliers to be simplified, were also presented for the SDF FFT. An explanation of radix-2^k SDF architectures can be found in. Finally, the current need for high throughput has been met by the MDF, which includes multiple interconnected SDF paths in parallel. However, radix-2 had not been considered for feed forward architectures until the first radix-2^2 feed forward FFT architectures were proposed. The paper shows that radix-2 can be used for any number of parallel samples which is a power of two. Accordingly, radix-2^2 FFT architectures for 2, 4, and 8 parallel samples are presented. These architectures are shown to be more hardware-efficient than previous feed forward and parallel feedback designs in the literature. This makes them very attractive for the computation of the FFT in the most demanding applications.

3.4 REORDERING OF THE OUTPUT SAMPLES

Reordering of the output samples is an inherent problem in FFT computation. The outputs are obtained in the bit-reversal order in the serial architectures. In general the problem is solved using a memory of size *N*. Samples are stored in the memory in natural order using a counter for the addresses and then they are read in bit-reversal order by reversing the bits of the counter[5].



Index	Output Order	Intermediate Order	Final Order
0	0	0	0
1	8	1	1
2	2	2	2
3	10	3	3
4	1	8	4
5	9	9	5
6	3	10	6
7	11	11	7
8	4	4	8
9	12	5	9
10	6	6	10
11	14	7	11
12	5	12	12
13	13	13	13
14	7	14	14
15	15	15	15

Fig 5: Solution to the reordering of the output sample

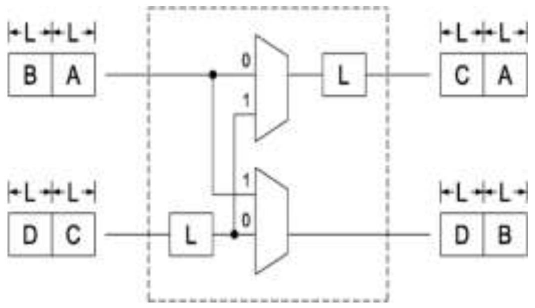


Fig 6: Basic circuit for the shuffling the data.

In embedded DSP systems, special memory addressing schemes are developed to solve this problem. But in case of real-time systems, this will lead to an increase in latency and area. The order of the output samples in the proposed architectures is not in the bit-reversed order. The output order change for different architectures because of different folding sets/scheduling schemes.

We need a general scheme for reordering these samples. One such approach is presented in this section. The approach is described using a 16-point radix-2 DIF FFT example and the corresponding architecture is shown in Fig.6. The order of output samples is shown in Fig 5.
The first column (index) shows the order of arrival of the output samples. The second column (output order) indicates the indices of the output frequencies. The goal is to obtain the frequencies in the desired order provided the order in the last column. We can observe that it is a type of de-interleaving from the output order the final order. Given the order of samples, the sorting can be performed in two stages. It can be seen that the first and the second half of the frequencies are interleaved. The intermediate order can be obtained by de-interleaving these samples as shown in the table. Next, the final order can be obtained by changing the order of the samples. It can be generalized for higher number of points; the reordering can be done by shuffling the samples in the respective positions according to the final order required[6].

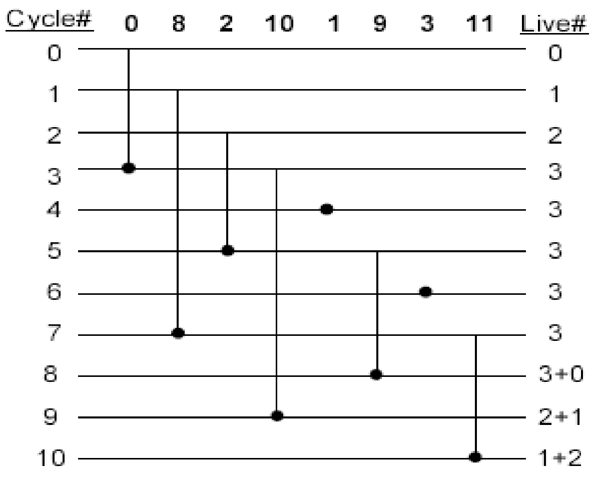


Fig 7 :Linear lifetime chart for the 1st stage shuffling of the data

A shuffling circuit is required to do the de-interleaving of the output data. Fig.6 shows a general circuit which can shuffle the data separated by R positions. If the multiplexer is set to ”1” the output will be in the same order as the input, whereas setting it to ”0” the input sample in that position is shuffled with the sample separated by R positions. The circuit can be obtained using lifetime analysis and forward-backward register allocation techniques. There is an inherent latency of R in this circuit. The life time analysis chart for the 1st stage shuffling in Fig 7. It uses 39 seven complex registers for a 16-point FFT. In general case, a N-point FFT requires a memory of 5N/8 - 3 complex data to obtain the outputs in natural order.

4 RESULT AND DISCUSSION

The simulation result of the proposed 4 parallel DIF –FFT architecture for all stages combined with the data shuffler is shown in Figure.

The inputs are given to the first stage of the 2^2, 16 point DIF FFT. The first stage calculates the 16 point DIF of the inputs. The output of the first stage is given to the second stage which calculates the 8 point of the given values. These values are given to the third stage which calculates the 4 point of the values and finally to the fourth stage which calculates the 2 point of the given values. The output values are bit reversed. The data shuffle is carried out in each stage of the 16 point DIF FFT operation.

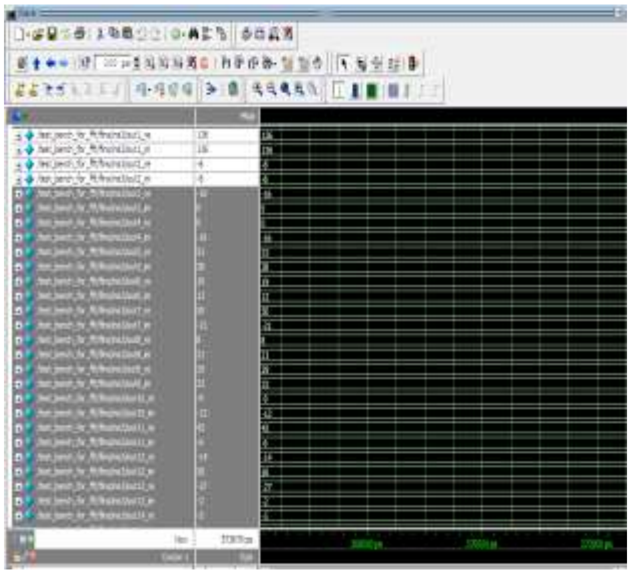


Fig 8: Simulated output.

The input real and imaginary values for the proposed DIF –FFT architecture is shown in Fig 9 . A sample value from the input is taken and theoretical calculations are carried out. The result of the calculations are shown in the below Fig 10. The result verify that the time domain signals are converted into frequency domain signals.

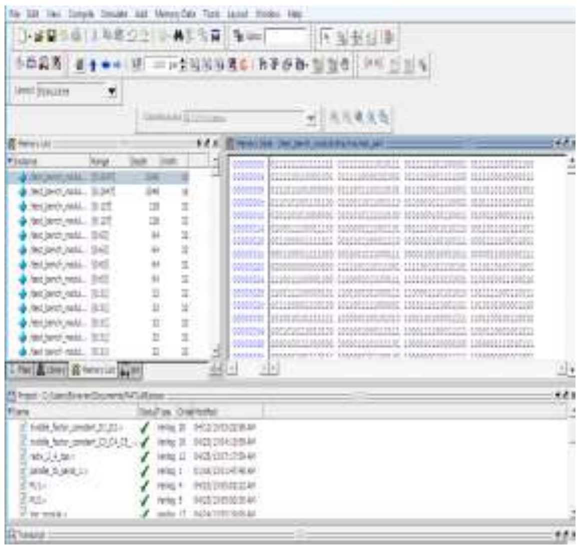


Fig 9: REAL & IMAGINARY VALUES

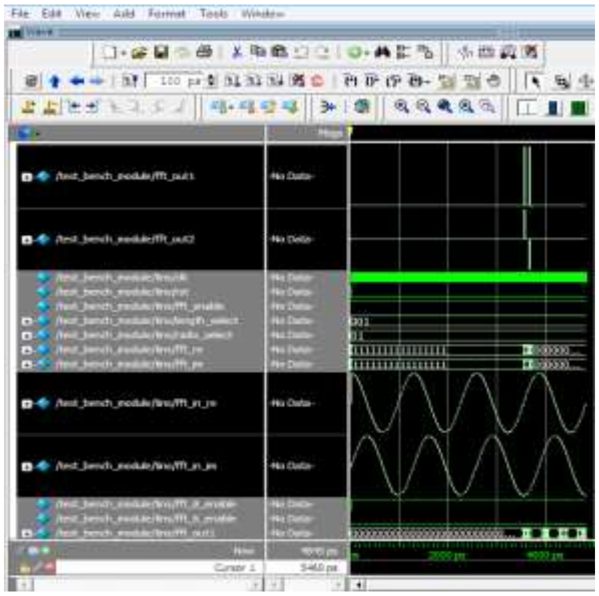


Fig 10: Domain verified simulated output.

5. CONCLUSION

In this paper, a radix 2^K algorithm and 512 point reconfigurable radix 2^k FFT architectures have been proposed for OFDM-based WPAN applications. The use of radix 2^k to feed forward (MDC) FFT architectures has shown that feed forward structures are more efficient than feedback ones when several samples in parallel must be processed. In feed forward architectures radix 2^k-can be used for any number of parallel samples which is a power of two. Indeed, the number of parallel samples can be chosen arbitrarily depending of the throughput that is required. Additionally, both DIF and DIT decompositions can be used. Additionally, data shuffling has also been implemented

REFERENCE

1. Mario Garrido, Member, IEEE, J. Grajal, M. A. Sánchez, and Oscar Gustafsson, Senior Member, IEEE, “Pipelined Radix-2^k Feedforward FFT Architectures”, IEEE transactions on very large scale integration (vlsi) systems,vol 21,No 1, January 2013.
2. Jes´us Garc´ıa, Juan A. Michell, and Angel M. Bur´on, “VLSI Configurable Delay Commutator for a Pipeline Split Radix FFT Architecture ”, ÏEEE transactions on signal processing, vol. 47, no. 11, november 1999.
3. Liang Yang, Kewei Zhang, Hongxia Liu, Jin Huang, and Shitan Huang , “An Efficient Locally Pipelined FFT Processor ”, IEEE transactions on circuits and systems—ii: ex.press briefs, vol. 53, no. 7, july 2006.
4. E. H. Wold and A. M. Despain, “Pipeline and parallel-pipeline FFT processors for VLSI implementation,” IEEE Trans. Comput., vol.C-33, no. 5, pp. 414–426, May 1984.
5. S. He and M. Torkelson, “A new approach to pipeline FFT processor,”in Proc. of IPPS, 1996, pp. 766–770.
6. P.Duhamel, “Implementation of split-radix FFT algorithms for complex,real, and real-symmetric data’’IEEE Trans.Acoust.,speech,Signal Process., Vol.34,no.2,pp.285-295,Apr.1986.