International Journal of VLSI System Design and Communication Systems

ISSN 2322-0929
Vol.02, Issue.08,
October-2014,
Pages:0691-0698

www.ijvdcs.org

Area Efficient Pipelined Radix-2k Feedforward FFT Architectures using Booth Multiplier

JAMPA SHRAVANI[1], K.SIVA NAGI REDDY[2]

[1]PG Scholar, Dept of ECE, Sridevi Women's Engineering College, JNTUH, Telangana, India, Email: shravanijampa27@gmail.com.
[2]Assoc Prof, Dept of ECE, Sridevi Women's Engineering College, JNTUH, Telangana, India, Email: sivanagireddykalli@gmail.com.

Abstract: The appearance of radix-2^2 was a milestone in the design of pipelined FFT hardware architectures. Later, radix-2^2 was extended to radix-2^k. However, radix-2^k was only proposed for single-path delay feedback (SDF) architectures, but not for feed forward ones, also called multi-path delay commentator (MDC). This paper presents the radix-2^k feed forward (MDC) FFT architectures. In feed forward architectures radix-2^k can be used for any number of parallel samples which is a power of two. Furthermore, both decimation in frequency (DIF) and decimation in time (DIT) decompositions can be used. In addition to this, the designs can achieve a very high throughput, which makes them suitable for the most demanding applications. Indeed, the proposed radix-2^k feed forward architectures require fewer hardware resources than parallel feedback ones, also called multi-path delay feedback (MDF), when several samples in parallel must be processed. By using booth multiplier the number of computational steps can be reduced as compared with basic multiplier when processing several samples in parallel. As the number of computational steps gets reduces hardware utilization reduces which in turn reduces area and also speed increases. As a result, the proposed radix-2^k feed forward architectures not only offer an attractive solution for current applications, but also open up a new research line on feed forward structures.

Keywords: Fast Fourier Transform (FFT), Radix-2k, Multipath Delay Commutator (MDC), Pipelined Architecture, Very large- scale integration (VLSI).

I. INTRODUCTION

Many digital signal processing (DSP) algorithms handle data on a frame basis. As the stream enters the DSP system, sets of data samples are taken together and an algorithm is computed on them. The Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT) algorithms are usually the first step in this kind of processing. Both of them, transform frames of a signal into Fourier's domain and the resulting coefficients are then analyzed or processed depending on a particular application. The better execution time of the FFT over the classical DFT algorithm places it as the standard selection for high performance signal processing applications. The relevance of this mathematical method impulse the development of special purpose VLSI systems to compute it and many different approaches were followed looking forward to maximize the speed of operation, minimize area or minimize power consumption. Fast Fourier Transforms (FFT) is the fast implementation of the Discrete Fourier Transform (DFT) which relies on mathematical simplification and classification of the input sequence to achieve their performance gain. The fast Fourier transform (FFT) was proposed by Cooley and Turkey [1] to efficiently reduce the time complexity to O (Nlog 2N) operations in comparison to the straight forward DFT that requires O(N2) operations, where N denotes the FFT size.

In current real-time applications, the FFT has to be calculated at very high throughput rates, even in the range of G Samples/s. These high-performance requirements appear in applications such as Orthogonal Frequency Division Multiplexing (OFDM) [9]–[12] and Ultra Wideband (UWB) [10]–[13]. 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 [20]. 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 [8]–[14]. Conversely, not very much attention has been paid to feedforward (MDC) architectures. This paradoxical fact, however, has a simple explanation. Originally, SDF and MDC architectures were proposed for radix-2 [2], [17] and radix-4 [3], [17]. Some years later, radix- 2^2 was presented for the SDF FFT [4] 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 [6]. Finally, the current need for high throughput has been meet by the MDF, which includes multiple interconnected SDF paths in parallel. However, radix- 2^k had not been considered for feedforward architectures until the first radix-2^2 feedforward FFT architectures were introduced a few years ago.

The single-path delay feedback architecture has a feedback loop at every stage [14, 20]. The butterfly processing element stores the input samples in feedback memory, until they are required for butterfly computation. These butterflies have 50% utilization ratio. This architecture has one continuous data stream of one sample per clock cycle, which means that at every clock cycle one sample is fed and one sample is received as output. However, higher throughput can be achieved by increasing the clock frequency. Multi-path delay feedback pipelined FFT architectures are also known as parallel feedback architectures. These architectures consist of parallel SDF pipelined FFT architectures for processing several samples. Like, the SDF pipelined FFT, the MDC architecture has 50% butterfly utilization ratio. Thus, the parallelism does not improve the utilization ratio. MDF architectures can process samples in parallel, so it has high throughput at the expense of hardware cost. The Multi-path delay commutator pipelined FFT architecture is the most straight forward implementation of FFT algorithms, also referred to as the feedforward FFT architecture [16]. This architecture does not have any feedback loop so data is processed by the butterflies and the rotators then fed to the next stage. As a result, they have 100% utilization ratio of butterflies [16].

Generally, it can compute several samples in parallel, so they can provide higher throughput than SDF pipelined architecture at the cost of hardware. The MDC pipelined architecture can be used for several butterflies, such as radix-2, radix-4 and radix-8. These architectures can be improved by using radix-2^2, radix-2^3, radix-2^4 and radix-2^k [16]. Radix-*r* MDC pipelined FFT architecture consists of radix-*r* butterflies, rotators, buffers and commutators. After the radix-*r* butterfly computes the *r*-point DFTs, the result is then fed to the buffer and commutator after twiddle factor multiplication. The purpose of the buffer and commutator block is to store the samples coming from one stage and reorder them in the right order for the next stage. Hence, the buffer and commutator block is used after twiddle factor multiplication. Each buffer and commutator block has a different size of buffer determined by the stage number.
The importance of pipelined architectures and also feedback and feedforward architectures has explained. When compared with feedback, in feedforward the butterfly utilization is 100%. Since butterfly utilization is 100% in feedforward, it can compute several samples in parallel and they can provide higher throughput than feedback architectures.

II. RADIX-2^2 FFT ALGORITHM

FFT algorithms are used to compute DFTs efficiently. These algorithms are not based on any approximation, so they have the same results as the direct computation of the DFT. The DFT is defined as:

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

Where N represent the size of the DFT, *n* is the time index, *k* is the frequency index and W_N^{nk} is the twiddle factor. The twiddle factor W_N^{nk} is defined as:

$$W_N^{nk} = e^{-j\frac{2\pi}{N}nk} \qquad (2)$$

When *N* is a power of two, the FFT based on the Cooley-Tukey algorithm [15] is most commonly used in order to compute the DFT efficiently.

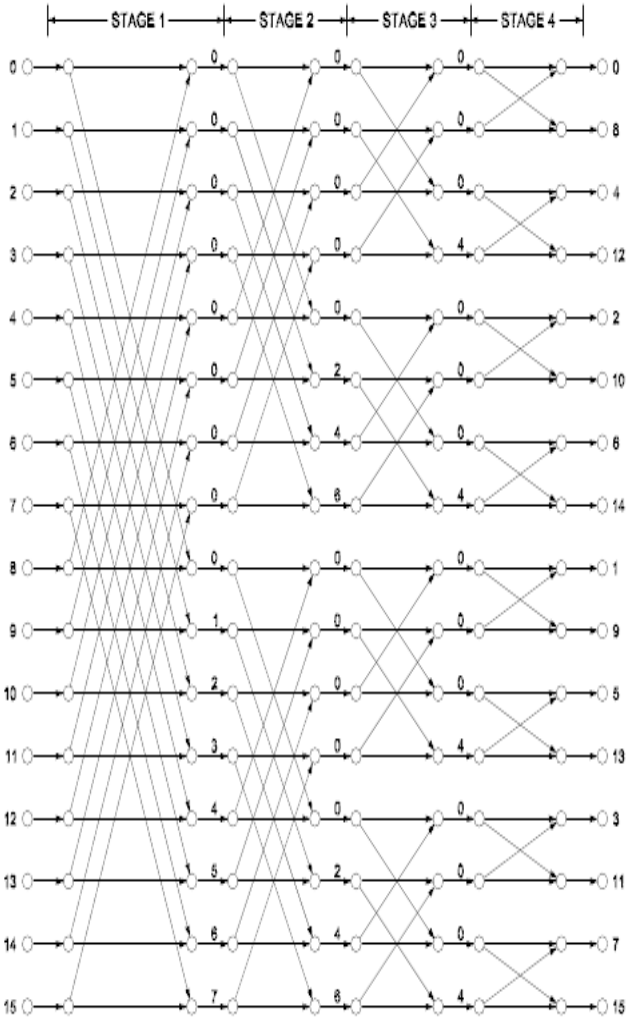


Fig1. Flow graph of the 16-point radix-2 DIF FFT.

The Cooley-Tukey algorithm reduces the number of operations from $O(N^2)$ for the DFT to $O(N \log_2 N)$ for the FFT1.Radix-2 decimation-in-time is the most common form of the Cooley-Tukey algorithm, for any arbitrary size N; Radix-2 DIT divides the size N DFT into two interleaved DFT's of size N/2. Radix-2 divides the DFT into two equal

parts, the first part calculates the Fourier Transform of the even index numbers and other part calculates the Fourier Transform of the odd index numbers and then finally merges them to get the Fourier Transform for the whole sequence. This will reduce the overall time to O(N log N).

Fig3 and2 show the flow graphs of 16-point radix-2 and radix-2^2 FFTs, respectively, decomposed using decimation in frequency (DIF) [16]. At each stage of the graphs, s ϵ {1, . . . , n} butterflies and rotations have to be calculated. The lower edges of the butterflies are always multiplied by -1. These -1 are not depicted in order to simplify the graphs. The numbers at the input represent the index of the input sequence, whereas those at the output are the frequencies, *k*, of the output signal *X*[*k*]. Finally, each number, ϕ, in between the stages indicates a rotation by:

$$W_N^{\phi} = e^{-j\frac{2\pi}{N}\phi} \qquad (3)$$

As a consequence, samples for which $\phi = 0$ do not need to be rotated. Likewise, if ϕ ϵ *2* [0,*N*/4,*N*/2, 3*N*/4] the samples must be rotated by 0°, 270°, 180° and 90°, which correspond to complex multiplications by 1, *-j*, -1 and *j*, respectively. These rotations are considered trivial, because they can be performed by interchanging the real and imaginary components and/or changing the sign of the data.

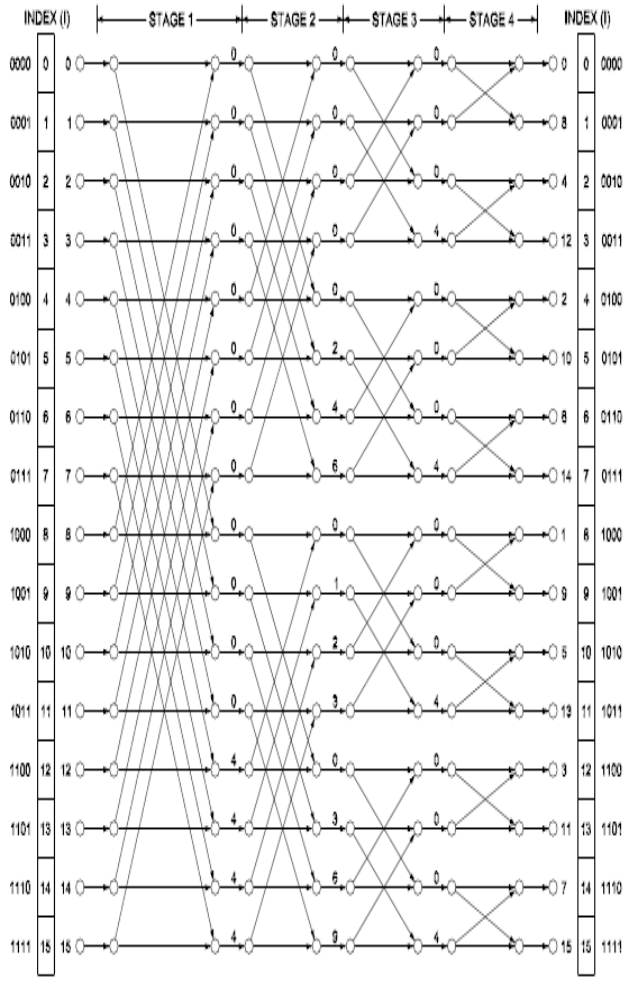


Fig2. Flow graph of the 16-point radix-2^2 DIF FFT.

A. Designing Radix-2^2 FFT Architectures:

The design is based on analyzing the flow graph of the FFT and extracting the properties of the algorithm. These properties are requirements that any hardware architecture that calculates the algorithm must fulfill. The properties of the radix-2^2 FFT are shown in Table1. The following paragraphs explain these properties and how they are obtained. The properties depend on the index of the data, $I \Xi b_{n-1}, \ldots, b_1, b_0$, where ($\Xi$) will be used throughout the paper to relate both the decimal and the binary representations of a number. This index is included in Fig.2 both in decimal and in binary.

Table1: Properties of the Radix-2^2 FFT Algorithm for DIF and DIT

Properties Radix-2^2	DIF	DIT
Butterflies	b_{n-s}	b_{n-s}
Trivial rotations (odd *s*)	$b_{n-s} \cdot b_{n-s-1} = 1$	$b_{n-s} \cdot b_{n-s-1} = 1$
Non-trivial rotations (even *s*)	$b_{n-s+1} + b_{n-s} = 1$	$b_{n-s-1} + b_{n-s-2} = 1$

On the one hand, the properties related to the butterfly indicate which samples must be operated together in the butterflies. This condition is b_{n-s} both for DIF and DIT decompositions and means that at any stage of the FFT, *s*, butterflies operate in pairs of data whose indices differ only in bit b_{n-s}, where $n = \log 2\ N$ is the number of stages of the FFT. In Fig. 2.2 it can be observed that at the third stage, $s = 3$, data with indices $I = 12\ \Xi\ 1100$ and $I' = 14\ \Xi\ 1110$ are processed together by a butterfly. These indices differ in bit *b*1, which meets b_{n-s}, since $n = \log 2\ N = \log 2\ 16 = 4$ and, thus, $b_{n-s} = b_{4-3} = b_1$. On the other hand, there are two properties for rotations. At odd stages of the radix-2^2 DIF FFT only those samples whose index fulfills $b_{n-s} \cdot b_{n-s-1} = 1$ have to be rotated. These rotations are trivial and the symbol (.) indicates the logic AND function. For the 16-point radix-2^2 FFT in Figure 2 only samples with indices 12, 13, 14 and 15 must be rotated at the first stage. For these indices $b_3 . b_2 = 1$ is fulfilled, meeting the property $b_{n-s} \cdot b_{n-s-1} = 1$, since $n = 4$ and $s = 1$. Conversely, at even stages rotations are non-trivial and they are calculated over indexed data for which $b_{n-s+1} + b_{n-s} = 1$, where the symbol (+) indicates the logic OR function.

III. PROPOSED SYSTEM

A. Radix-2^2 Feedforward FFT Architectures:

This section presents the radix-2^2 feedforward architectures. First, 16-point 4-parallel radix-2^2 feedforward FFT architecture is explained in depth in order to clarify the approach and show how to analyze the architectures. Then, radix-2^2 feedforward architectures for different number of parallel samples are presented. Fig3 shows a 16-point 4-parallel radix-2^2 feedforward FFT architecture. The architecture is made up of radix-2 butterflies (R2), non-trivial rotators (⊗), trivial rotators, which are diamond-

shaped, and shuffling structures, which consist of buffers and multiplexers. The lengths of the buffers are indicated by a number.

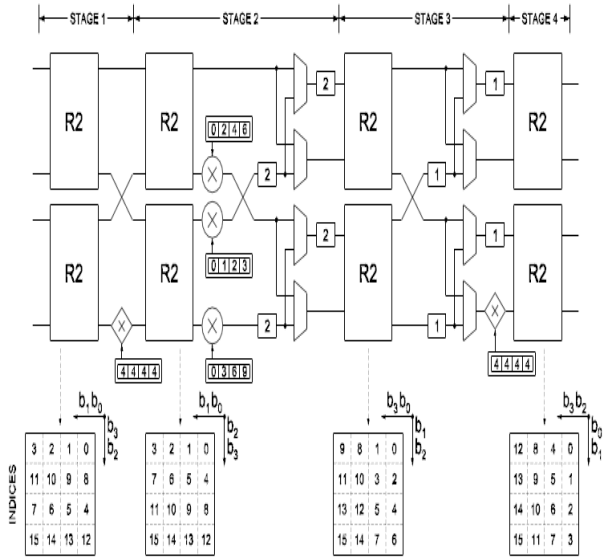


Fig3. Proposed 4-parallel radix-2^2 feedforward architecture for the computation of the 16-point DIF FFT.

The architecture processes four samples in parallel in a continuous flow. The order of the data at the different stages is shown at the bottom of the figure by their indices, together with the bits b_i that correspond to these indices. In the horizontal, indexed samples arrive at the same terminal at different time instants, whereas samples in the vertical arrive at the same time at different terminals. Finally, samples flow from left to right. Thus, indexed samples (0, 8, 4, 12) arrive in parallel at the inputs of the circuit at the first clock cycle, whereas indexed samples (12, 13, 14, 15) arrive at consecutive clock cycles at the lower input terminal.

Taking the previous considerations into account, the architecture can be analyzed as follows. Firstly, it can be observed that butterflies always operate in pairs of samples whose indices differ in bit b_{n-s} , meeting the property in Table 2.1. For instance, the pairs of data that arrive at the upper butterfly of the first stage are: (0, 8), (1, 9), (2, 10) and (3, 11). The binary representation of these pairs of numbers only differ in b_3. As, $n = 4$ and $s = 1$ at the first stage, $b_{n-s} = b_{4-1} = b_3$, so the condition is fulfilled. This property can also be checked for the rest of the butterflies in a similar way.

Secondly, that rotations at odd stages are trivial and only affect samples whose indices fulfill $b_{n-s} \cdot b_{n-s-1} = 1$. By particularizing this condition for the first stage, $b_3 \cdot b_2 = 1$ is obtained. In the architecture shown in Figure 3.3 the indices that fulfill this condition are those of the lower edge and, thus, a trivial rotator is included at that edge. On the other hand, the condition for non-trivial rotations at even stages is $b_{n-s+1} + b_{n-s} = 1$, $b_3 + b_2 = 1$ being for the second stage. As $b_3 + b_2 = 0$ for all indexed samples at the upper edge of the second stage, this edge does not need any rotator. Conversely, for the rest of edges $b_3 + b_2 = 1$, so they include non-trivial rotators. The rotation memories of the circuit store the coefficients ϕ of the flow graph. It can be seen that the coefficient associated to each index is the same as that in the flow graph of Fig4. For instance, at the flow graph the sample with index $I = 14$ has to be rotated by $\phi = 6$ at the second stage. In the architecture shown in Fig5 the sample with index $I = 14$ is the third one that arrives at the lower edge of the second stage. Thus, the third position of the rotation memory of the lower rotator stores the coefficient for the angle $\phi = 6$. Thirdly, the buffers and multiplexers carry out data shuffling. These circuits have already been used in previous pipelined FFT architectures and Fig4. shows how they work.

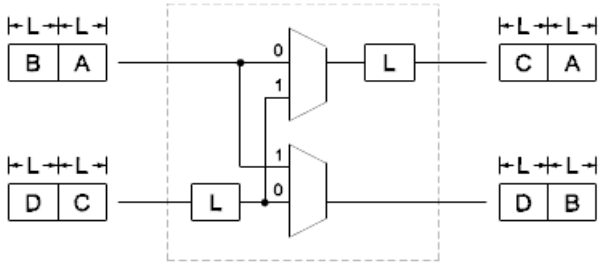
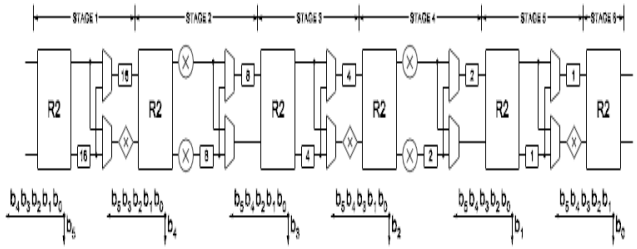
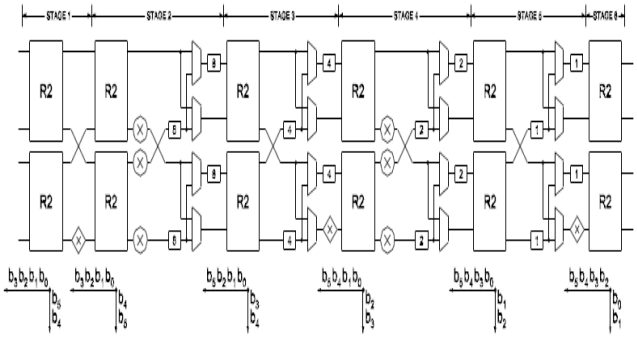


Fig4. Circuit for data shuffling

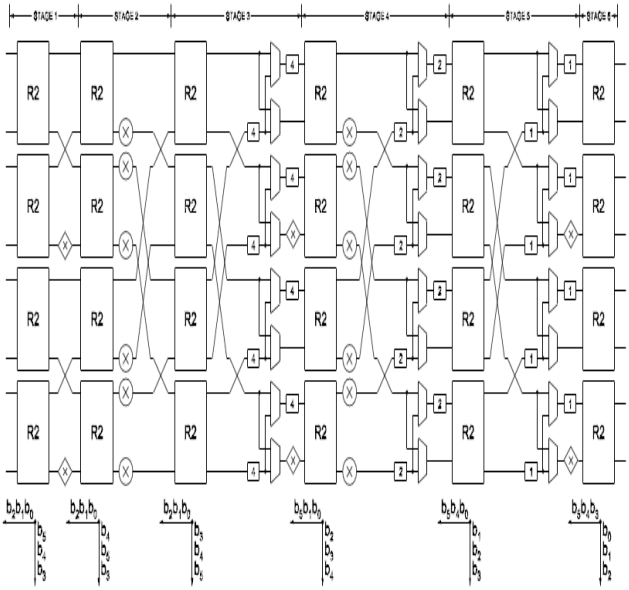
For the first L clock cycles the multiplexers are set to ”0”, L being the length of the buffers. Thus, the first L samples from the upper path (set A) are stored in the output buffer and the first L samples from the lower path (set C) are stored in the input buffer. Next, the multiplexer changes to ”1”, so set C passes to the output buffer and set D is stored in the input buffer. At the same time, sets A and B are provided in parallel at the output. When the multiplexer commutes again to ”0”, sets C and D are provided in parallel. As a result, sets B and C are interchanged. Finally, the control of the circuit is very simple: As the multiplexers commute every L clock cycles and L is a power of two, the control signals of the multiplexers are directly obtained from the bits of a counter. Fig5 shows the proposed radix-2^2 feedforward architectures for the computation of the 64-point DIF FFT. Fig5(a), 5(b) and 5(c) show the cases of 2-parallel, 4- parallel and 8-parallel samples, respectively. These circuits can be analyzed as has been done for the architecture in Figure 3.3. For this purpose, the order of the samples at every stage has been added at the bottom of the architectures.



(a) 2-parallel radix-2^2 feedforward FFT.



(b) 4-parallel radix-2^2 feedforward FFT.



(c) 8-parallel radix-2^2 feedforward FFT.

Fig5. Proposed radix-2^2 feedforward architectures for the computation of the 64-point DIF FFT

As can be seen in Fig. 3.3, in the proposed architectures the number of butterflies depends on to the number of samples in parallel, $P = 2^p$. For any *P*-parallel *N*-point FFT the number of butterflies is $P/2 \cdot \log_2 N = P \cdot \log_4 N$. Therefore, the number of complex adders is $2P \cdot \log_4 N$. Likewise, the number of rotators is $3P/4 \cdot (\log_4 N - 1)$. The only exception is for $P = 2$. In this case, the number of rotators is $2 \cdot (\log_4 N - 1)$. The proposed architectures can process a continuous flow of data. The throughput in samples per clock cycle is equal to the number of samples in parallel, $P = 2^p$, whereas the latency is proportional to the size of the FFT divided by the number of parallel samples, i.e., *N/P*. Thus, the most suitable architecture for a given application can be selected by considering the throughput and latency that the application demands. Indeed, the number of parallel samples can be increased arbitrarily, which assures that the most demanding requirements are met. Finally, the memory size does not increase with the number of parallel samples. For the architectures shown in Fig. 5, the shuffling structure at any stage $s \in [p, n-1]$ requires $P = 2^p$ buffers of length $L = N/2^{s+1}$. According to this, the total sample memory of the architectures is:

$$\sum_{s=p}^{n-1} 2^p . L = \sum_{s=p}^{\log_2 N-1} 2^p . \frac{N}{2^{s+1}} = N - 2^p = N - P \qquad (4)$$

Therefore, a total sample memory of *N* addresses is enough for the computation of an *N*-point FFT independently of the degree of parallelism of the FFT. Indeed, the total memory of *N-P* addresses that the proposed architectures require is the minimum amount of memory for an *N*-point *P*-parallel FFT. Sometimes input samples are provided to the FFT in natural order and output frequencies are also required in natural order. Under these circumstances, reordering circuits are required before and after the FFT to adapt the input and output orders. For the proposed radix-2^2 feedforward FFTs the memory requirements for natural I/O depend on the FFT size and on the number of parallel samples. For a *P*-parallel *N*-point FFT a total memory of size *N - N/P* is enough to carry out the input reordering, whereas a total memory of size *N* is enough for the output reordering.

The proposed approach can also be used to derive radix-2^2 feedforward architectures FFT for DIT. In this case, the properties for DIT in Table 2.1 must be considered. Accordingly, Figure 6 shows a 4-parallel radix-2^2 feedforward architecture for the computation of the 64-point DIT FFT. This architecture can be compared with the DIF version in Figure 3.3(b). It can be noted that both DIF and DIT architectures use the same number of hardware components. Nevertheless, the layout of the components is different. For any number of parallel samples, DIF and DIT architectures also require the same number of components.

IV. BOOTH ALGORITHM

Booth's multiplication algorithm is an algorithm which multiplies 2 signed integers in 2's complement. The algorithm is depicted in the following figure with a brief description. This approach uses fewer additions and subtractions than more straightforward algorithms. Booth's algorithm can be implemented by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values *A* and *S* to a product *P*, then performing a rightward arithmetic shift on *P*. Let M and Q be the multiplicand and multiplier, respectively; and let *x* and *y* represent the number of bits in M and Q.

1. Determine the values of *A* and *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 M. Fill the remaining $(y + 1)$ bits with zeros.
   - S: Fill the most significant bits with the value of (−M) 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 Q. Fill the least significant (rightmost) bit 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, do nothing. Use *P* directly in the next step.
   - If they are 11, do nothing. Use *P* directly in the next step.
   - Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let *P* now equal this new value.
   - Repeat steps 2 and 3 until they have been done *y* times.
   - Drop the least significant (rightmost) bit from *P*. This is the product of M and Q.

A. Flowchart for Booth Algorithm:

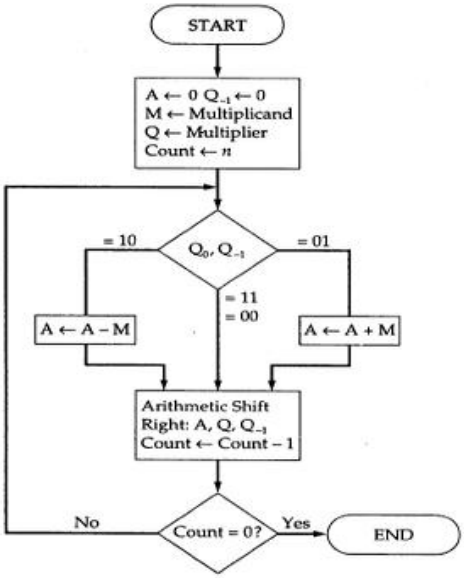


Fig6. Flowchart for Booth Algorithm

Fig6 shows the flowchart for booth multiplier which is followed by the algorithmic steps as written above. Designed Radix-2^2 feedforward FFT architecture 4-parallel 16point DIF-FFT. In order to reduce the area consumption, booth multiplier is used as the number of computational steps reduces in Booth algorithm. Thus this approach uses fewer additions and subtractions than more straightforward algorithms.

V.RESULTS

The proposed Algorithm is designed with individual modules by writing behavioral Verilog code and hierarchically designed its top module by writing the structural modeling of Verilog code. The functional verification of all the modules Architecture is done using active HDL Tool and its results are captured in the form of waveforms. The design is synthesized and its final RTL Level net-list files are captured using XILINX ISE Tool.

A. Radix 2^2 4 parallel 16 point FFT Simulation and synthesis results

1.Simulation Result:

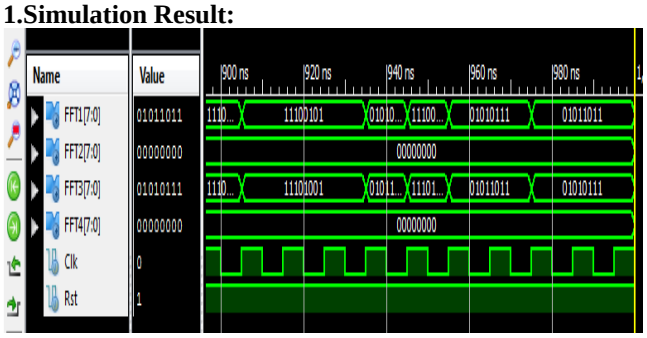


Fig7. Simulation result of 4 parallel 16 point FFT

2. Synthesis Report:

Device utilization summary:

Selected Device : 3s100evq100-5

Number of Slices:	201	out of	960	20%
Number of 4 input LUTs:	349	out of	1920	18%
Number of IOs:	34			
Number of bonded IOBs:	34	out of	66	51%

Timing summary:

Speed Grade: -5

Minimum period: 8.455ns (Maximum Frequency: 118.273MHz)

Minimum input arrival time before clock: 7.063ns

Maximum output required time after clock: 4.040ns

Maximum combinational path delay: No path found

Total memory usage is 253176 kilobytes

B. Radix 2^2 2 parallel 64 point FFT:

1. Simulation Result

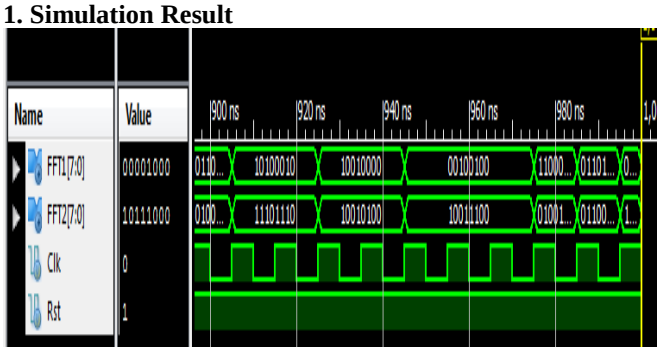


Fig8. Simulation result of 2 parallel 16 point FFT.

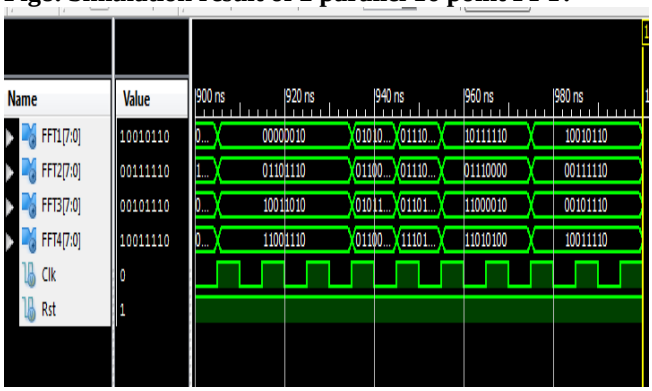


Fig9. Simulation result of 4 parallel 16 point FFT.

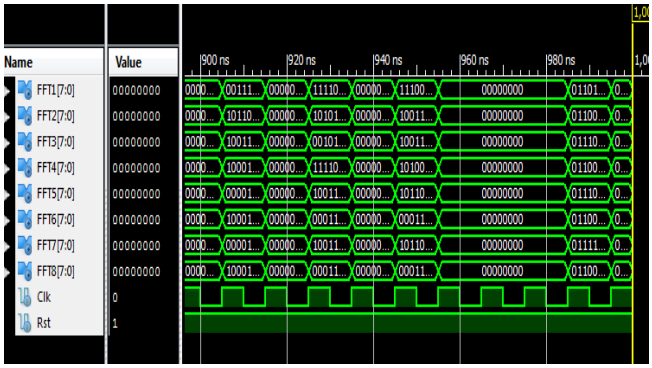


Fig10. Simulation result of 8 parallel 16 point FFT.

2. Synthesis Report:

Device utilization summary:

Selected Device : 3s100evq100-5
Number of Slices: 185 out of 960 19%
Number of 4 input LUTs: 329 out of 1920 17%
Number of IOs: 18
Number of bonded IOBs: 18 out of 66 27%
Speed Grade: -5
Minimum period: 8.587ns (Maximum Frequency: 116.461MHz)
Minimum input arrival time before clock: 7.009ns
Maximum output required time after clock: 4.040ns
Maximum combinational path delay: No path found
Total memory usage is 251960 kilobytes

C. Radix 2^2 4 parallel 64 point:

1. Synthesis Report:

1. Device utilization summary:

Selected Device : 3s100evq100-5
Number of Slices: 201 out of 960 20%
Number of 4 input LUTs: 349 out of 1920 18%
Number of IOs: 34
Number of bonded IOBs: 34 out of 66 51%
Timing Summary:
Speed Grade: -5
Minimum period: 8.455ns (Maximum Frequency: 118.273MHz)
Minimum input arrival time before clock: 7.063ns
Maximum output required time after clock: 4.040ns
Maximum combinational path delay: No path found
Total memory usage is 253496 kilobytes

D. Radix 2^2 8 parallel 64 point FFT:

1. Synthesis Report:

Device utilization summary:

Selected Device : 3s100evq100-5
Number of Slices: 383 out of 960 39%
Number of 4 input LUTs: 648 out of 1920 33%
Number of IOs: 66
Number of bonded IOBs: 66 out of 66 100%
Timing Summary:
Speed Grade: -5
Minimum period: 8.151ns (Maximum Frequency: 122.679MHz)
Minimum input arrival time before clock: 7.344ns
Maximum output required time after clock: 4.040ns
Maximum combinational path delay: No path found
Total memory usage is 264888 kilobytes.

E. Radix 2^2 4 parallel 16 point FFT:

Table2. Comparison Results

Function	Number of slices	Number of slice Flip Flops	Number of 4 input LUTs	Number of IO's	Number of bonded IOBs	Number of GCLKs
Existing work	239	214	440	66	66	1
Proposed work	201	186	349	34	34	1

Table3. Memory utilization

Function	Memory (kilobytes)
Existing work	261304
Proposed work	253176

F. Radix 2^2 parallel 64 point FFT:

Table4: Memory utilization

Function	**2 parallel (KB)**	**4 parallel (KB)**	**8 parallel (KB)**
Existing system	26450	265272	231160
Proposed system	227348	228052	227732

VI. CONCLUSION

The design is simulated by Xilinx 13.2. This paper extends the use of radix-2^k to feedforward (MDC) FFT architectures. Indeed, it is shown that feedforward structures are more efficient than feedback ones when several samples in parallel must be processed. In feedforward 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. By using Booth multiplier the number of computations is reduced which in turn minimizes area. Finally, experimental results show that the designs are efficient both in area and performance, being possible to obtain throughputs of the order of G Samples/s.

VII. REFERENCES

[1] L. Yang, K. Zhang, H. Liu, J. Huang, and S. Huang, "An efficient locally pipelined FFT processor," IEEE Trans. Circuits Syst. II, vol. 53, no. 7, pp. 585–589, Jul. 2006.
[2] H. L. Groginsky and G. A. Works, "A pipeline fast Fourier transform," IEEE Trans. Comput., vol. C-19, no. 11, pp. 1015–1019, Oct. 1970.
[3] A. M. Despain, "Fourier transform computers using CORDIC iterations," IEEE Trans. Comput., vol. C-23, pp. 993–1001, Oct. 1974.
[4] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," in Proc. IEEE Custom Integrated Circuits Conf., May 1998, pp. 131–134.

[5] M. A. S´anchez, M. Garrido, M. L. L´opez, and J. Grajal, "Implementing FFT-based digital channelized receivers on FPGA platforms," IEEE Trans. Aerosp. Electron. Syst., vol. 44, no. 4, pp. 1567–1585, Oct. 2008.
[6] A. Cort´es, I. V´elez, and J. F. Sevillano, "Radix rk FFTs: Matricial representation and SDC/SDF pipeline implementation," IEEE Trans. Signal Process., vol. 57, no. 7, pp. 2824–2839, Jul. 2009.
[7] E. H. Wold and A. M. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementations," IEEE Trans. Comput., no. 5, pp. 414–426, May 1984.
[8] 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. I, vol. 57, no. 6, pp. 451–455, Jun. 2010.
[9] H. Liu and H. Lee, "A high performance four-parallel 128/64-point radix-24 FFT/IFFT processor for MIMO-OFDM systems," in Proc. IEEE Asia Pacific Conf. Circuits Syst., 2008, pp. 834–837.
[10] L. Liu, J. Ren, X. Wang, and F. Ye, "Design of low-power, 1GS/s throughput FFT processor for MIMO-OFDM UWB communication system," in Proc. IEEE Int. Symp. Circuits Syst., May 2007, pp. 2594– 2597.
[11] J. Lee, H. Lee, S. in Cho, and S.-S. Choi, "A high-speed, low-complexity radix-24 FFT processor for MB-OFDM UWB systems," in Proc. IEEE Int. Symp. Circuits Syst., 2006, pp. 210–213.
[12] N. Li and N. P. van der Meijs, "A radix 22 based parallel pipeline FFT processor for MB-OFDM UWB system," in Proc. IEEE Int. SOC Conf., 2009, pp. 383–386.
[13] S.-I. Cho, K.-M.Kang, and S.-S. Choi, "Implementation of 128-point fast Fourier transform processor for UWB systems," in Proc. Int. Wireless Comm. Mobile Comp. Conf., 2008, pp. 210–213.
[14] W. Xudong and L. Yu, "Special-purpose computer for 64-point FFT based on FPGA," in Proc. Int. Conf. Wireless Comm. Signal Process., 2009, pp. 1–3.
[15] C. Cheng and K. K. Parhi, "High-throughput VLSI architecture for FFT computation," IEEE Trans. Circuits Syst. II, vol. 54, no. 10, pp. 863–867, Oct. 2007.
[16] J. A. Johnston, "Parallel pipeline fast Fourier transformer," in IEE Proc. F Comm. Radar Signal Process., vol. 130, no. 6, Oct. 1983, pp. 564– 572.
[17] B. Gold and T. Bially, "Parallelism in fast Fourier transform hardware," IEEE Trans. Audio Electroacoust., vol. 21, no. 1, pp. 5–16, Feb. 1973.
[18] E. E. Swartzlander, W. K. W. Young, and S. J. Joseph, "A radix 4 delay commutator for fast Fourier transform processor implementation," IEEE J. Solid-State Circuits, vol. 19, no. 5, pp. 702–709, Oct. 1984.
[19] J. H. McClellan and R. J. Purdy, Applications of Digital Signal Processing. Prentice-Hall, 1978, ch. 5, Applications of Digital Signal Processing to Radar.
[20] M. Garrido, K. K. Parhi, and J. Grajal, "A pipelined FFT architecture for real-valued signals," IEEE Trans. Circuits Syst. I, vol. 56, no. 12, pp. 2634–2643, Dec. 2009.