International Journal of Electrical and Electronic Engineering & Telecommunications

ISSN 2319 – 2518 www.ijeetc.com
Vol. 3, No. 4, October 2014

Research Paper

A NEW APPORACH FOR DESIGNING EFFECTIVE FEEDFORWARD FFT ARCHITECTURE

Maruthi Sai Bharadwaj T[1]* and N Suresh Babu[2]

Corresponding Author: **Maruthi Sai Bharadwaj T ✉ tmsbharadwaj@gmail.com*

The paper present radix radix-2^k module which is proposed for single-path delay feedback (SDF) architectures, but not for feed forward ones,and even this paper deal with presents the radix 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 very high throughputs, 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. 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 by implemetanting the radix which is highly reducing the complexity.

Keywords: Fast Fourier transform (FFT), Multipath delay commutator (MDC), Pipelined architecture, Radix-2^k, VLSI

INTRODUCTION

The Fourier series is a trigonometric series. Specifically, it is a series of sinusoids (plus a constant term), whose amplitudes may be determined by a certain process (to be described in the following chapters). this compact notation cannot reveal the feedback incredible mathematical subtlety contained within. The Fourier series, like the Taylor/ Maclauren series shown earlier functions, but it has a different derivation and a different purpose. Rather than being a means of evaluating sines, cosines, etc., at a single point, it serves as a “transformation” for the whole of a given, arbitrary, This, then, is the general pool that we have thrown our Fourier

[1] M.Tech. Student, Department of ECE, Chirala Engineering College, Chirala 523155, Prakasam Dt., AP.

[2] Associate Professor, Department of ECE, Chirala Engineering College, Chirala 523155, Prakasam Dt., AP.

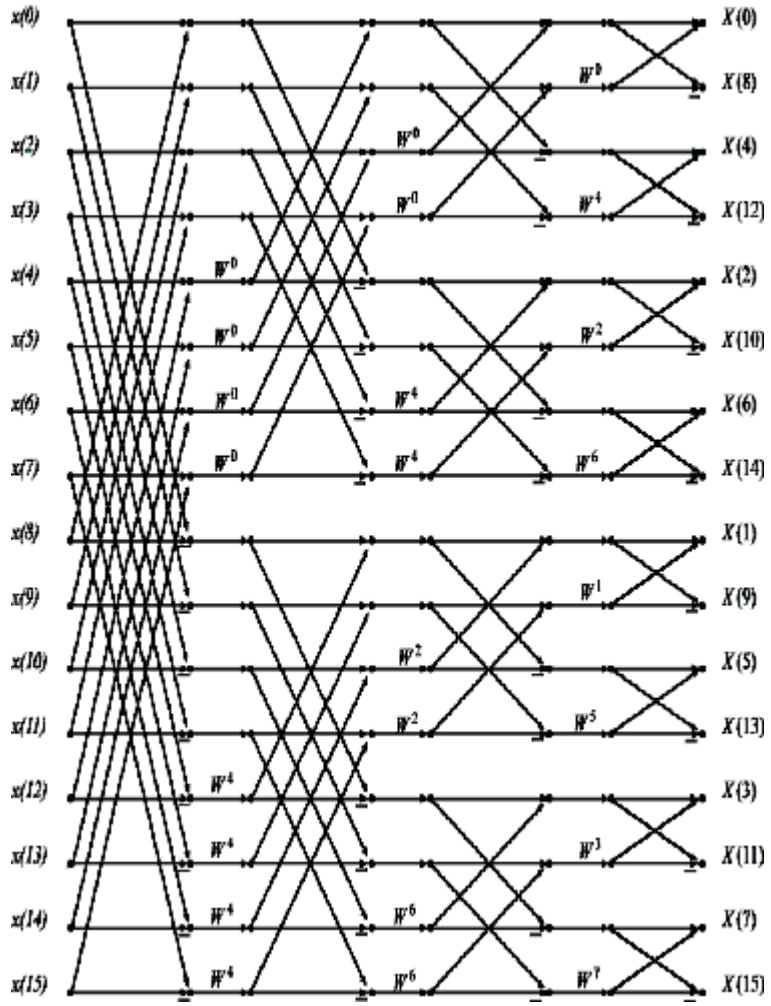
Transform into, but we are at risk here of making the pool so obscure it will require more definition than our definition itself. The newcomer may well ask; "What is this transformation you speak of?" Apparently we are going to transform the original function into another, different, function—but what is the new function and why da we bother? Does the transformed function have some special mathematical properties? Can we still obtain the same information provided by the original function? The answer is yes to both of these questions but we will come to all of that later; for now we may say that the transform we are referring to, in its digital form, provides a mathematical tool of such power and scope that it can hardly be exceeded by any other development of applied mathematics in the twentieth century. Function (FB) and feedforward (FF). 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) 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 Gigasamples 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. This paradoxical fact, however, has a simple explanation. Originally, SDF and MDC architectures were proposed for radix- and radix- Some years later, radix-2^2 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 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 proposed a few years ago.

Figure 1: Flow Graph of the 16-point Radix-2 DIF FFT



In this work we present the radix-2^k feedforward FFT architectures. The proposed designs include radix-2^2 , radix-2^3 and radix-2^4 architectures. The paper shows that radix-2^k can be used for any number of parallel samples which is a power of two. Accordingly, radix-2^k FFT architectures for 2, 4, and 8 parallel samples are presented. These architectures are shown to be more hardware-efficient than previous feedforward and parallel feedback designs in the literature. This makes them very attractive for the computation of the FFT in the most demanding applications.

This paper is organized as follows. Section II explains the radix-2^2 FFT algorithm and Section III shows how to design radix-2^2 FFT architectures. As a result, the pipelined radix-2^2 feedforward FFT architectures are presented in Section IV, where architectures for different number of parallel samples using DIF and DIT decompositions are proposed. In Section V, the results are extended to radix-2^k and feedforward FFT architectures for radix-2^3 and radix-2^4 are presented. In Section VI, the proposed designs are compared to previous ones and in Section VII experimental

results are provided. Finally, the main contributions of this work are summarized in Section VIII.

I. RADIX2²- FFT ALGORITHM

The -point DFT of an input sequence x[n]is defined as where

$$X[K] = \sum_{n=0}^{N-1} X[n] W_N^K \qquad k = 0, 1, N-1$$

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

The first designed chip is an FFT processor. The FFT processor has a central position both in the OFDM transmitter and receiver. The FFT is a computationally demanding operation that requires an ASIC implementation to reach high performance, i.e. high throughput combined with low energy consumption

The FFT and IFFT Equations has the property that, if

FFT(Re(xi)+ jIm(xi)) = Re(Xi)+ jIm(Xi)

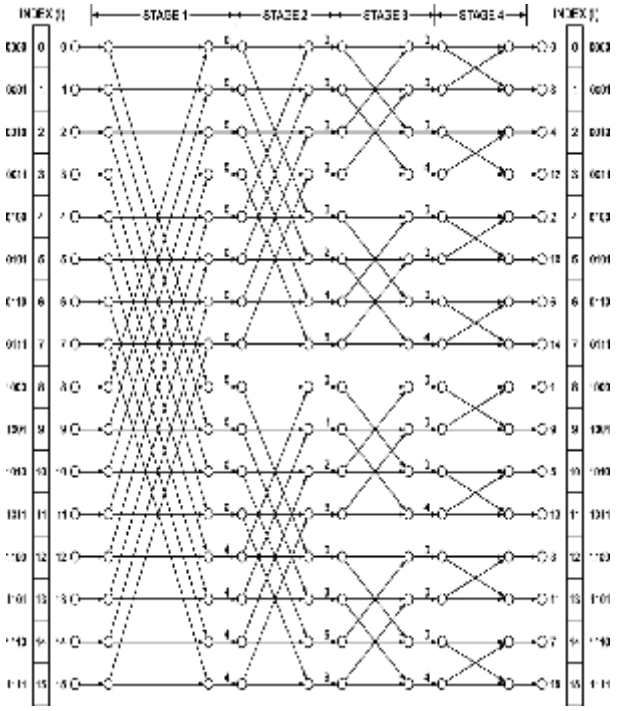
IFFT(Re(Xi)+ jIm(Xi)) = Re(xi)+ jIm(xi),

where xi and Xi are N words long sequences of complex valued, samples and sub-carriers respectively, then

1/N * FFT(Im(Xi)+ jRe(Xi)) = Im(xi)+ jRe(xi).

Thus, it is only necessary to discuss and implement the FFT equation. To calculate the inverse transform, the real and imaginary part of the input and output are swapped. Since N is a power of two, scaling with 1/N is the same as right shift the binary word Log2(N) bits. Even simpler, is to just remember that the binary point has moved log2(N) bits to the left. Not performing the bit shift until, if ever, it is

Figure 2: Flow Graph of the 16-point Radix-2² DIF FFT



necessary, which depends on how the output from the IFFT will be used.

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, Where as 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

$$w_N^{\Phi} = e^{-j(2\pi/N)\Phi}$$

As a consequence, ϕ = 0, samples for which do not need to be rotated. Likewise, if $\phi \varepsilon$ {N/4,N/2,3N/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.

Radix-2^2 is based on radix-2 and the flow graph of a radix-2^2 DIF FFT can be obtained from the graph of a radix-2 DIF one. This can be done by breaking down each angle, ϕ, at odd stages into a trivial rotation and a non-trivial one, ϕ, where ϕ' = ϕ mod N/4, and moving the latter to the following stage. This is possible thanks to the fact that in the radix-2 DIF FFT the rotation angles at the two inputs of every butterfly, ϕ_A and ϕ_B, only differ by 0 or N/4. Thus, if ϕ_A= ϕ' and ϕ_B= ϕ'+N/4 , the rotation ϕ' is moved to the following stage in accordance with

where the first side

$$Ae^{-j\frac{2x}{N}\phi'} + Be^{\left(-j\frac{2x}{N}\phi'+N/4\right)} = \left[A \pm (-j)B\right].e^{-j\frac{2x}{N}\phi'}$$

of (3) represents the computations using radix-2 and the second one using radix-2^2 , A and B being the input data of the butterfly. In radix-2, and are rotated before the butterfly is computed, whereas in radix-2^2 B is rotated by the trivial rotation –j before the butterfly, and the remaining rotation is carried out after the butterfly. Consequently, rotations by ϕ' can be combined with those rotations of the following stage. This derivation of radix- 2^2 from radix-2 can be observed in Figures 1 and 2 for the particular case of N=16.

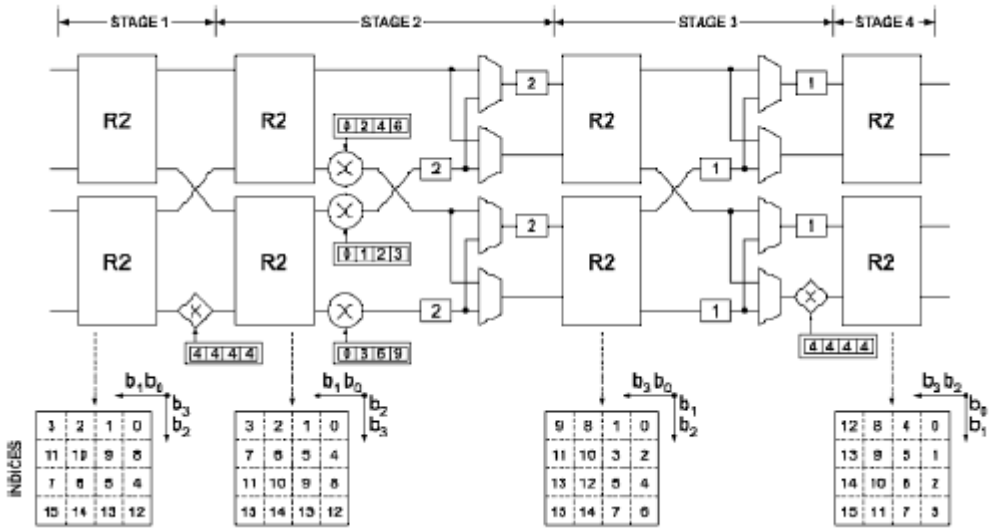
Analogously, the radix-2^2 DIT FFT can be derived from the radix-2 DIT FFT. Contrary to DIF, for DIT the non-trivial rotations ϕ' are moved to the previous stage instead of the following one.

DESIGNING RADIX- FFT ARCHITECTURES

The proposed architectures have been derived using the framework presented by M Garrido (2009). 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 following paragraphs explain these properties and how they are obtained.

The properties depend on the index of the data, I a" b_{n-1},.....b_1,b_0, where (a")will be used throughout the paper to relate both the decimal and the binary representations of a number. This index is included in Figure 2 both in decimal and in binary. On the one hand, the

Figure 3: Proposed 4-parallel Radix-2^2 feed Forward Architecture for the Computation of the 16-point DIF FFT



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, 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 Figure 2 it can be observed that at the third stage, s = 3 , data with indices I=12 ≡ 1100 and I' =14 ≡ 1110 are processed together by a butterfly. These indices differ in bit b_1 , which meets b_{n-s} , since $n = \log_2^N = n = \log_2^{16}=4$ and, thus, $b_{n-s.}$

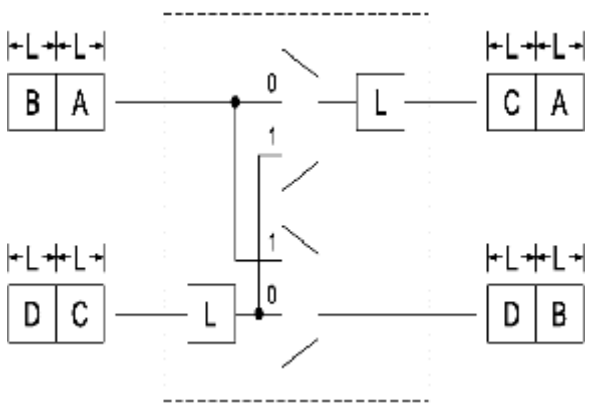
RADIX-2^2 FEEDFORWARD FFT ARCHITECTURES

This section presents the radix- feedforward architectures. First, a 16-point 4-parallel radix- feedforward FFT architecture is explained in depth in order to clarify the approach and show how to analyze the architectures. Then, radix- feedforward architectures for different number of parallel samples are presented.

When N is a power of 4, i.e. N =4p, a radix-4 FFT can be used instead of a radix-2 FFT. With a radix-4 the computational complexity is reduced, i.e. the numbers of complex multiplications are reduced compared to a radix-2 FFT. The drawback with a radix-4 is that the butterfly structure is more complicated. Another, better type of radix-4 is the radix-2, there the butterflies still look like radix-2 butterflies but the number of complex multipliers are reduced as if it were a radix-4. Each stage in aradix-2 FFT consists of two butterflies, one trivial multiplier, i.e. multiplication with “j”, and one complex multiplier that multiply data with the twiddle factor that butterflies always operate in pairs of samples. 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 b3 as n=4, and s=1 at the first stage , so the condition is fulfilled. This property can also be checked for the rest of the butterflies in a similar way. By particularizing this condition for the first stage, is obtained. In the architecture shown in Figure 3 the indices that full fill 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, being for the second stage. As 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, so they include non-trivial rotators .The rotation memories of the circuit store the coefficients 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 Figure 2. For instance, at the flow graph the sample with index has to be rotated by at the second stage. In the architecture shown in Figure 3 the sample with index 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 .Thirdly, the buffers and multiplexers carry out data shuffling. These circuits have already been used in previous pipelined FFT architectures, shows how they work .For the first clock cycles the multiplexers are set to “0”, being the length of the buffers. Thus, the first samples from the upper path (set) are stored in the output buffer and the first samples from the lower path (set) are stored in the input buffer. Next, the multiplexer changes to “1”, so set passes to the output buffer and set is stored in the input buffer. At the same time, sets and are provided in parallel at the output .When the multiplexer commutes again to “0”, sets and are provided in parallel. As a result, sets and are interchanged. Finally, the control of the circuit is very simple: As the multiplexers commute every clock cycles and is a power of two, the control signals of the multiplexers are directly obtained from the bits of a counter. Figure 5 shows the proposed radix- feedforward architectures for the computation of the 64-point DIF FFT. The cases of 2-parallel, 4-parallel, and 8-parallel samples, respectively. For this purpose, the order of the samples at every stage has been added at the bottom of the architectures. As can be seen in the proposed architectures the number of butterflies depends on to the number of samples in parallel. The throughput in samples per clock cycle is equal to the number of samples in parallel , whereas the latency is proportional to the size of the FFT divided by the number of parallel samples. Thus, the most suitable architecture for a given

Figure 4: Data Shuffling

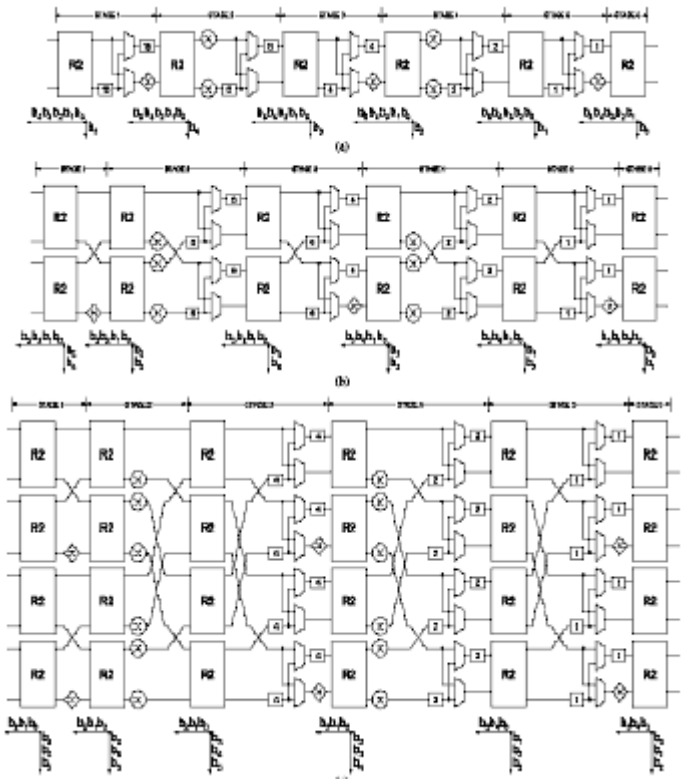


application can be selected by considering the throughput and latency that the application demands. Indeed, the number of parallel samples can be increased arbitrarily, which assures that the most demanding requirements are met. Finally, the memory size does not increase with the number of parallel samples. For the architectures the shuffling structure at any stage requires buffers of length. According to this, the total sample memory of the architectures is

$$\sum_{n=p}^{n-1} 2^{p,L} = \sum_{n=p}^{\log_2 N-1} 2^{p}, \frac{N}{2^{n+1}} = N - 2^{P} = N - P$$

Therefore, a total sample memory of addresses is enough for the computation of an -point FFT independently of the degree of parallelism of the FFT. Indeed, the total memory of addresses that the proposed architectures require is the minimum amount of memory for the point 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-feedforward FFTs the memory requirements for natural I/O depend on the FFT size and on the number of parallel samples. For a parallel point FFT a total memory of size is enough to carry out the input reordering, whereas a total memory of size is enough for the output reordering The

Figure 5: Proposed Radix-2^2 Feedforward Architectures for the Computation of the 64-point DIF FFT. (a) 2-parallel radix-2^2 Feedforward FFT, (b) 4-parallel Radix Feedforward FFT, (c) 8-parallel Radix-2^2 Feedforward FFT



proposed approach can also be used to derive radix feedforward architectures FFT for DIT. Accordingly, parallel radix-feedforward architecture for the computation of the 64-point DIT FFT. This architecture can be compared with the DIF version and 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.

EXTENSION TO RADIX-2^K

As for radix-2^2, these properties have been obtained directly from the flow graphs of the algorithms. The conditions for butterflies are the same for all stages of the FFT, whereas the conditions for rotations depend on the stage. Rotations are classified into trivial (T), non-trivial (NT), and rotations by W_8 or W_{16}. Rotations W_8 by W_{16} and are not-trivial, but include a reduced set of angles. According to (2), rotations by W_8 only consider angles that are multiples of $\Pi/4$, whereas W_{16} only includes multiples of $\Pi/8$. This allows for the simplification of the rotators that carry out the rotations. For this purpose, different techniques have been proposed in the literature. They include the use of trigonometric identities, the representation of the coefficients in canonical signed digit (CSD) and the scaling of the coefficients. Finally, in the table $i \in z$ and, thus, for radix-2^k the type of rotation repeats every k stages. Figure 7(a) and (b) show the proposed radix-2^3 feedforward architectures, respectively for 2 and 4 samples in parallel. It can be observed that radix-2^3 feedforward architectures only require general non-trivial rotators every three stages. Additionally, the architectures must calculate rotations by W_8, which are represented by squared-shaped rotators. Compared to the 2-parallel radix-2^2 feedforward architecture in Figure 5(a), the 2-parallel radix-2^3 feedforward FFT in Figure 7(a) has the same number of butterflies, rotators and total memory. However, some of the rotators for radix-2^3 calculate rotations by W_8, which can be simplified. Likewise, the 4-parallel radix-2^3 feedforward FFT in Figure 7(b) includes fewer general rotators than the radix-2^2 one in Figure 5(b). The proposed radix-2^4 feedforward FFT architectures for N=256 are shown in Figure 8. The architectures also include square-shaped rotators, which carry out the rotations by W_{16}. Note that the 2-parallel radix-2^4 feedforward FFT is very similar to the 2-parallel radix-2^2 one, with the difference that general rotators every four stages in radix-2^2 are substituted by W_{16} rotators in radix-2^4. For 4-parallel samples, radix-2^4 also needs fewer general rotators than radix-2^2 and radix-2^3. Architectures for a higher number of samples in parallel can also be obtained using radix-2^k. For a general case of a P-parallel radix-2^k N-point

$$P.\log_2 N$$

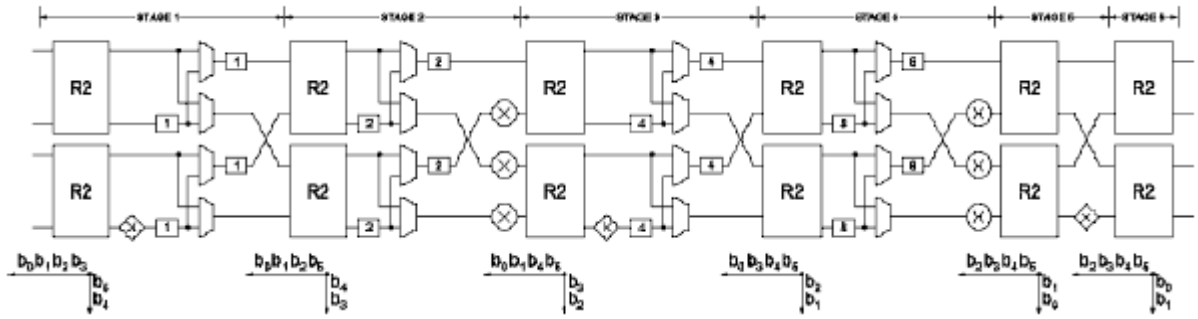
feedforward FFT, the number of complex adders is equal tothe number of general rotators can be calculated as

$$P.\left(\frac{\log_2 N}{k}-1\right), if\ P<2^k$$

$$\frac{2^k-1}{2^k}P.\left(\frac{\log_2 N}{k}-1\right), if\ P<2^k$$

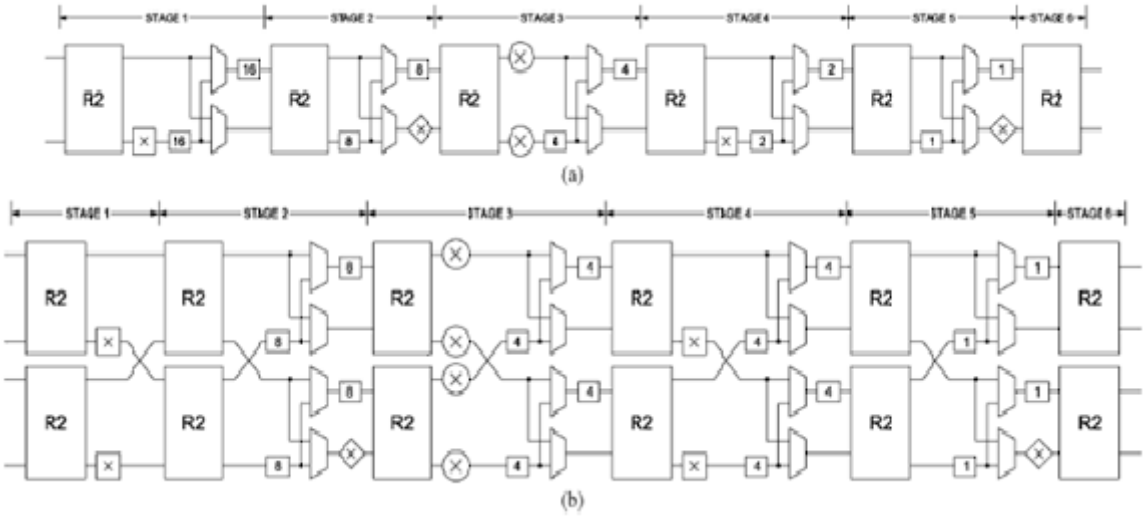
and the total memory is N-P. Likewise, the

Figure 6: Proposed 4-parallel radix- 2^2 Feedforward Architecture for the Computation of the 64-point DIT FFT



throughput is always equal to the number of parallel samples, P, and the latency is N/P. Note that apart from general rotators, the architectures must include rotators that calculate the simpler non-trivial rotations by W_L, where $L=2^k$ is the number of angles of the kernel. These kernels are W_8 and W_{16}, respectively for radix-2^3 and radix-2^4, which allow for efficient hardware implementations. Nevertheless, if k is larger, radix-2^k architectures include W_L kernels with larger number of angles. As a result, the implementation of these rotators becomes more complicated, being necessary to resort to general rotators in most cases. Note also that the proposed radix-2^k feedforward FFT architectures can be used for any number of parallel samples, $P=2^P$. Conversely, conventional feedforward architectures based on radix-r are only for $r \leq P$.

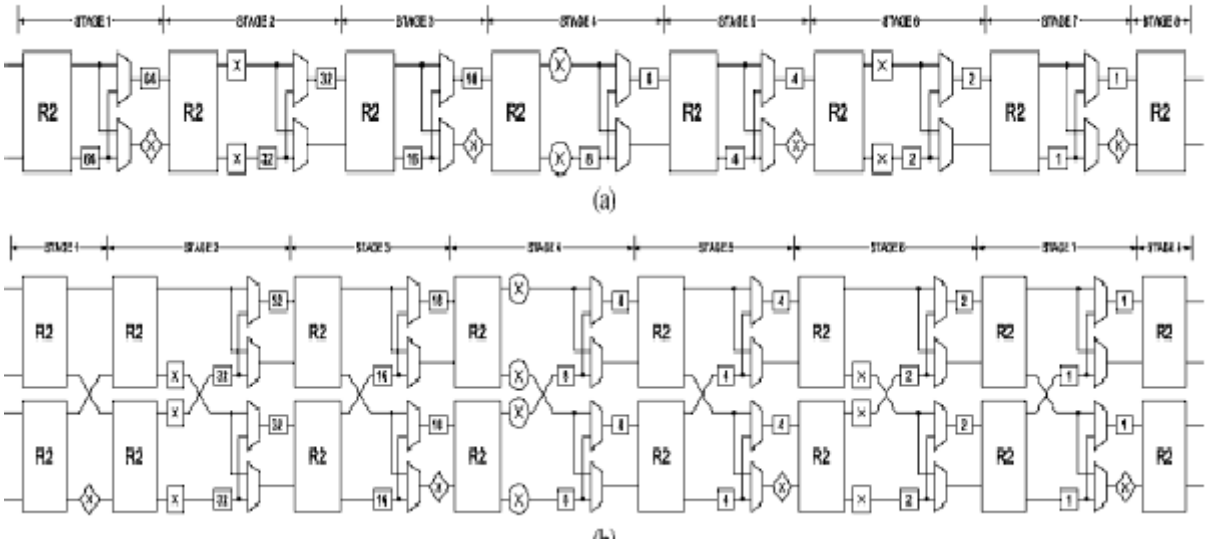
Figure 7: Proposed Radix-2^3 Feedforward Architectures for the Computation of the 64-point DIF FFT
(a) 2-Parallel Radix-2^3 Feedforward FFT (b) 4-Parallel Radix-2^3 Feedforward FFT



COMPARISON AND ANALYSIS

One of the more important decisions designing a fixed-point pipelined FFT processor regards the word length in the architecture, since it will affect the precision, number of gates, and power consumption. The most straightforward implementation is to have the same word length throughout the whole FFT processor. However, this will give a poor performance per kbit memory since the data has to be shifted down before each butterfly, to avoid overflow. As there are ten butterflies in a 1024 point FFT, the ten least significant bits are lost if a fixed word length data path is used. The proposed structures to other efficient pipelined architectures for the computation of an N-point FFT. The architectures are classified into 2-parallel, 4-parallel, and 8-parallel ones. The first two columns indicate the type of architecture and the radix. The rest of the table shows the trade-off between area and performance. On the other hand, area is measured in terms of the number of rotators, adders and memory. As different applications demand different input and output orders, circuits for data reordering before and after the FFTs are not considered in the comparison. Rotators are required for non-trivial rotations. In Table 1 they are classified into rotators for W_8 and W_{16}, and general rotators for other non-trivial rotations. The total number of rotators is also included. On the other hand, performance is represented by throughput and latency. The latency is defined as the number of clock cycles that the architecture needs to process an input sequence, considering that it receives a continuous flow of data. Meanwhile, the throughput indicates the number of samples per clock cycle that are processed. In all architectures this throughput is equal to the number of samples that are processed in parallel. Among 2-parallel architectures, the proposed radix-2^k feedforward FFTs require

Figure 8: Proposed Radix-2^4 Feedforward Architectures for the Computation of the 256-point DIF FFT, (a) 2-parallel Radix-2^4 Feedforward FFT, (b) 4-parallel Radix-2^4 Feedforward FFT



the same number of rotators, adders and memory as the radix-2 feedforward FFT. However, some of the rotators in radix-2^3 and radix-2^4 FFTs can be simplified, as they only have to calculate rotations by W_8 and W_{16}. Compared to previous radix-2^4 parallel feedback architectures, the proposed radix-2^4 designs save 50% of the adders and reduce the memory requirements, and proposed architectures with respect to parallel feedback ones. Specifically the proposed radix-2^2 architecture saves 25% of the total number of rotators. Furthermore, the proposed 4-parallel radix-2^4 feedforward FFT saves 50% of the adders and 25% of the W_{16} rotators with respect to radix-2^4 parallel feedback architectures. Finally, the proposed 8-parallel radix-2^k architectures improve on all previous designs in the literature. The proposed 8-parallel radix-2^2 feedforward FFT saves 25% of the rotators with respect to radix-2 feedforward FFTs, and 50% of the adders and 25% of the rotators with respect to feedback architectures. The proposed 8-parallel radix-2^3 feedforward FFT reduces the memory requirements and latency of previous radix-8 feedforward FFTs, and the proposed 8-parallel radix-2^4 feedforward FFT saves 12% of the W_{16} rotators and 50% of the adders with respect to radix-2^4 parallel feedback designs.

EXPERIMENTAL RESULTS

The presented architectures have been programmed for the use in field-programmable gate arrays (FPGAs). The designs are parameterizable in the number of points , word length, and number of The results for 4-parallel pipelined architectures are shown in Figure 10(a). In the figure, the numbers next to the lines indicate the amount of DSP48E slices that

Figure 9: RTL Schematic of Radix DFFT

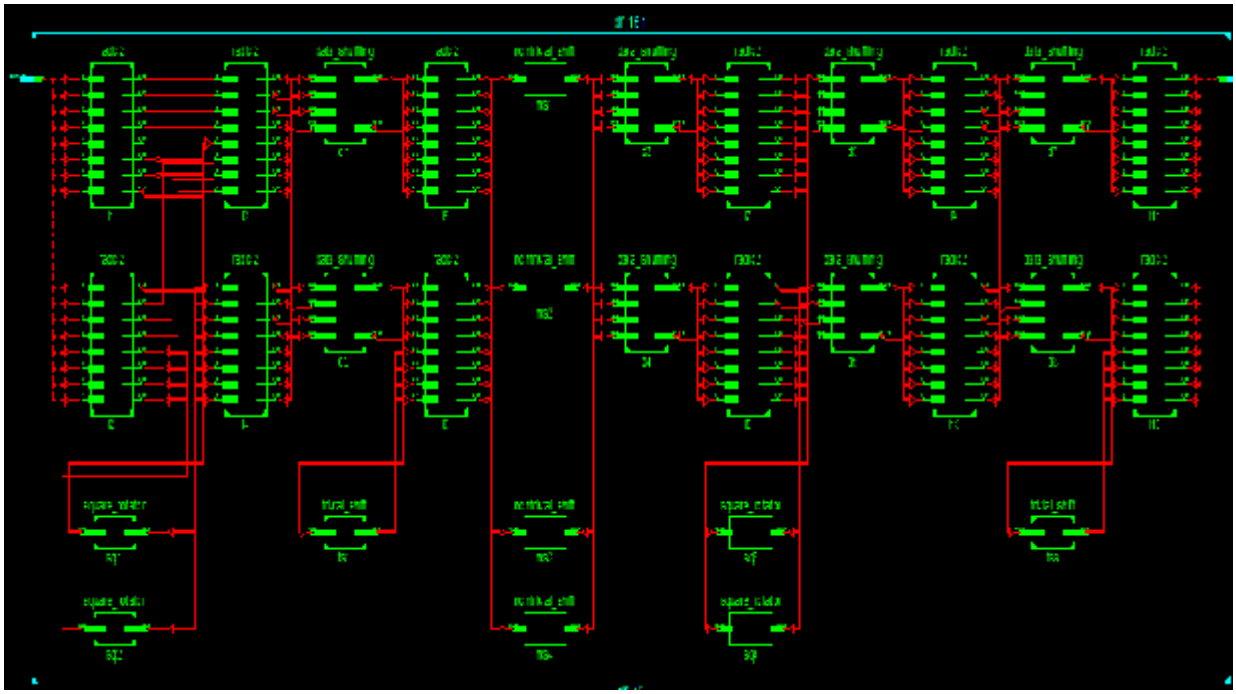


Table 1: Comparison of the Proposed Radix-2^k Feedforward Architectures

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of 4 input LUTs	33	66,560	1%
Number of occupied Slices	20	33,280	1%
Number of Slices containing only related logic	20	20	100%
Number of Slices containing unrelated logic	0	20	0%
Total Number of 4 input LUTs	33	66,560	1%
Number of bonded IOBs	64	633	10%
Average Fanout of Non-Clock Nets	2.34		

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	10	768	1%
Number of 4 input LUTs	17	1536	1%
Number of bonded IOBs	32	124	25%

each architecture requires. It can be observed that the area and performance of the proposed p-parallel n-pointradix-2^2 feedforward fft architectures for 16 bits. proposed radix-2^2 architectures require less area than previous designs for any FFT size, N. This improvement increases with the size of the FFT. For 8-parallel samples, Figure 9(b) shows that the proposed designs also improve over radix-2 and radix-4 architectures, and the larger N the larger the savings. Architectures that use radix-8 need less DSP48E blocks at the cost of a significant increase in the number of slices. The throughput of the proposed designs to other 4-parallel and 8-parallel pipelined FFTs. It can be observed that, the proposed designs achieve the highest throughputs both for 4-parallel and 8-parallel designs. Indeed, even higher throughput can be achieved by resorting to 16-parallel radix-2^2 feedforward architectures.

Figure 10: Area of 4-parallel and 8-parallel pipelined FFT architectures (a) 4-parallel pipelined FFT architectures (b) 8-parallel pipelined FFT architectures (c) 8 bit result

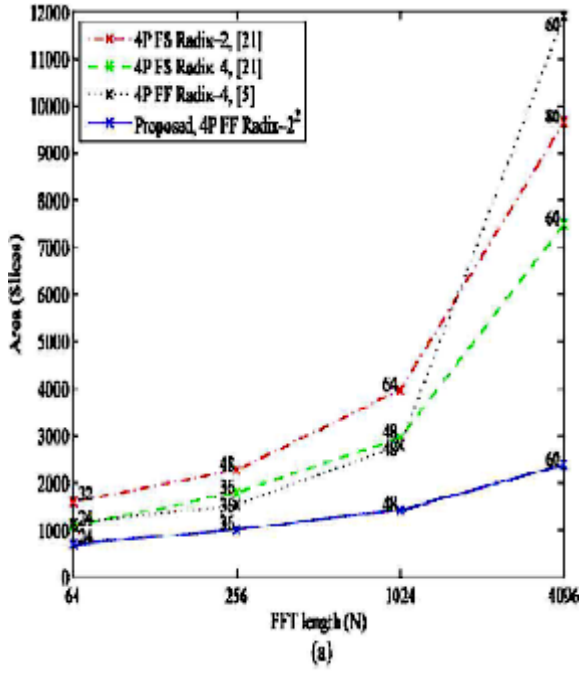
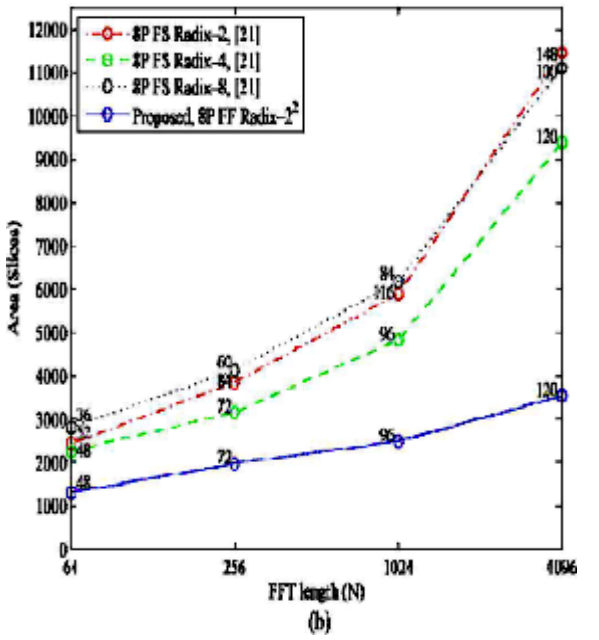


Figure 10 (Cont.)



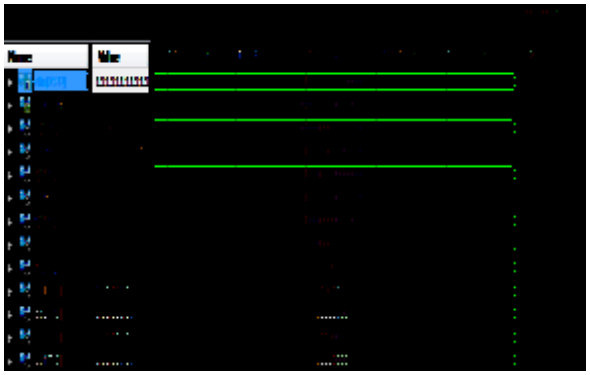
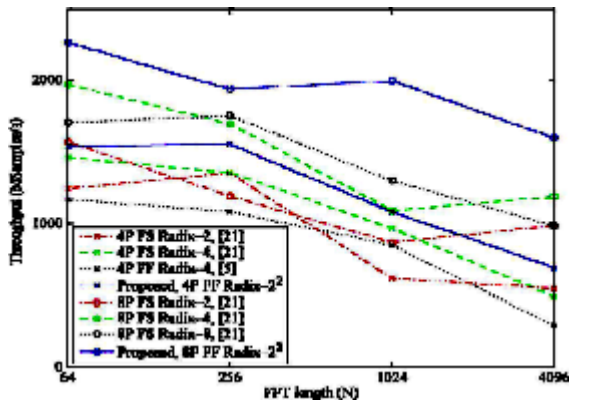


Figure 11: Throughput of 4-parallel and 8-parallel Pipelined FFT Architectures



CONCLUSION

In the VLSI design the power delay and area are grave concern, here in this projects also depends based up on these factors, that is instead of feedback, feedforward (MDC) architectures are used. 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, the number of parallel samples can be chosen arbitrarily depending of the throughput that is required. Internally both DIF and DIT decompositions can be used. finally compare to feedback architectures, feedforward(MDC) architecture takes less samples , low area consumption and delay. With the existing one it is more speed. It possible to obtain throughputs of the order of G Samples(s) as well as very low latencies. below results shows. Throughput of 4-parallel and 8-parallel pipelined FFT architectures

ACKNOWLEDGMENT

The authors would like to thank Dr. R. Conway for his valuable suggestions about the presentation of this work.

REFERENCES

1. Chang Y N (2008), "An efficient VLSI architecture for normal I/O order pipeline FFT design", *IEEE Trans. Circuits Syst. II, Exp. Briefs*, Vol.55, No. 12, pp. 1234–1238.
2. Cheng C and Parhi K K (2007), "High-throughputVLSI architecture for

FFTcomputation", *IEEE Trans. Circuits Syst. II, Exp. Briefs*, Vol. 54, No. 10, pp. 863–867

3. Cho S I, Kang K M and Choi S S (2008), "Implemention of 128-pointfast Fourier transform processor for UWB systems", in *Proc. Int.Wirel. Commun. Mobile Comput. Conf.*, pp. 210–213.
4. Cooley J W and Tukey J W (1965), "An algorithm for the machine calculation of complex Fourier series", *Math. Comput.*, Vol. 19, pp. 297–301.
5. Cortés A, Vélez I and Sevillano J F (2009), "Radix FFTs: Matricial representation and SDC/SDF pipeline implementation", *IEEE Trans. Signal Process.*, Vol. 57, No. 7, pp. 2824–2839.
6. Despain A M (1974), "Fourier transform computers using CORDIC iterations", *EEE Trans. Comput.*, Vol. C-23, pp. 993-1001.
7. Garrido M (2009), "Efficient hardware architectures for the computation ofthe FFT and other related signal processing algorithms in real time", Ph.D. dissertation, Dept. Signals, Syst., Radiocommun., Univ. PolitécnicaMadrid, Madrid, Spain.
8. Garrido M, Grajal J, and Gustafsson O (2011), "Optimum circuits for bit reversal," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, Vol. 58, No. 10, pp.657–661.
9. Garrido M, Gustafsson O, and Grajal J (2011), "Accurate rotations based oncoefficient scaling", *IEEE Trans. Circuits Syst. II, Exp. Briefs*, Vol. 58, No. 10, pp. 662–666.
10. Garrido, Parhi K K, and Grajal J (2009), "A pipelined FFT architecturefor real-valued signals", *IEEE Trans. Circuits Syst. I, Reg. Papers*, Vol. 56, No. 12, pp. 2634–2643.
11. Gold B and Bially T (1973), "Parallelism in fast Fourier transformhardware"*, IEEE Trans. Audio Electroacoust.*, Vol. 21, No. 1, pp. 5–16.
12. Groginsky H L and Works G A (1970), "A pipeline fast Fourier transform", *IEEE Trans. Comput.*, Vol. C-19, No. 11, pp. 1015-1019.
13. He S and Torkelson M (1998), "Design and implementation of a 1024-pointpipeline FFT processor", in *Proc. IEEE Custom Integr. Circuits Conf.*, pp. 131-134.
14. Johnston J A (1983), "Parallel pipeline fast Fourier transformer", *IEE Proc.F Commun. Radar Signal Process.*, Vol. 130, No. 6, pp. 564–572.
15. Lee J, Lee H, Cho S I, and Choi S S (2006), "A high-speed, low-complexity-radix-FFT processor for MB-OFDM UWB systems", in*Proc. IEEE Int. Symp. Circuits Syst.*, pp. 210–213.
16. Li N and van der Meijs N P (2009), "A radix based parallel pipeline FFT processor for MB-OFDM UWB system", in *Proc. IEEE Int. SOCConf.*, pp. 383–386.
17. Li Y W , Xu H, Fan W, Chen Y, and Zeng W (2010), "A 128/256-point pipelineFFT/IFFT processor for MIMO OFDM system IEEE 802.16e", in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 1488–1491.

18. Lin Y W and Lee C Y (2007), "Design of an FFT/IFFT processor forMIMO OFDM systems", *IEEE Trans. Circuits Syst. I, Reg. Papers*, Vol. 54, No. 4, pp. 807–815.

19. Liu H and Lee H (2008), "A high performance four-parallel 128/64-point radix- FFT/IFFT processor for MIMO-OFDM systems", in *Proc.IEEE Asia Pacific Conf. Circuits Syst.*, pp. 834–837.

20. Liu L, Ren J, Wang X, and Ye F (2007), "Design of low-power, 1 GS/sthroughput FFT processor for MIMO-OFDM UWB communication system", in *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 2594–2597.

21. McClellan J H and Purdy P J (1978), "Applications of digital signal processingto radar", in *Applications of Digital Signal Processing*, Englewood Cliffs, NJ, Prentice-Hall, Ch. 5.

22. Milder P A , Franchetti F, Hoe J C, and Püschel M (2008), "Formal datapath representation and manipulation for implementing DSP transforms", in*Proc. IEEE Design Autom. Conf.*, pp. 385–390.

23. Oppenheim A V and Schafer R W (1989), *Discrete-Time Signal Processing*, Englewood Cliffs, NJ, Prentice-Hall.

24. Qureshi F and Gustafsson O (2011), "Low-complexity constant multiplicand based on trigonometric identities with applications to FFTs", *IEICE Trans. Fundamentals*, Vol. E94-A, No. 11, pp. 324–326.

25. Sánchez M A, Garrido M, López M L and Grajal J (2008), "ImplementingFFT-based digital channelized receivers on FPGA platforms", *IEEE Trans. Aerosp. Electron. Syst.*, Vol. 44, No. 4, pp. 1567–1585.

26. Swartzlander E E, Young W K W, and Joseph S J (1984), "A radix 4delay commutator for fast Fourier transform processor implementation", *IEEE J. Solid-State Circuits*, Vol. 19, No. 5, pp. 702–709.

27. Tang S N, Tsai J W and Chang T Y (2010), "A 2.4-GS/s FFT processor for OFDM-based WPAN applications", *IEEE Trans. Circuits Syst. I, Reg. Papers*, Vol. 57, No. 6, pp. 451–455.

28. Wold E H and Despain A M (1984), "Pipeline and parallel-pipeline FFT processors for VLSI implementations", *IEEE Trans. Comput.*, Vol. C-33, No. 5, pp. 414-426.

29. Xudong W and Yu L (2009), "Special-purpose computer for 64-point FFTbased on FPGA", in *Proc. Int. Conf. Wirel. Commun. Signal Process.*, pp. 1–3.

30. Yang L, Zhang K, Liu H, Huang J and Huang S (2006), "An efficient locally pipelined FFT processor", *IEEE Trans. Circuits Syst. II, Exp. Briefs*, Vol. 53, No. 7, pp. 585-589.