Pipelined Parallel FFT Architectures via Folding Transformation

Manohar Ayinala, *Student Member, IEEE*, Michael Brown, and Keshab K. Parhi, *Fellow, IEEE*

***Abstract*—This paper presents a novel approach to develop parallel pipelined architectures for the fast Fourier transform (FFT). A formal procedure for designing FFT architectures using folding transformation and register minimization techniques is proposed. Novel parallel-pipelined architectures for the computation of complex and real valued fast Fourier transform are derived. For complex valued Fourier transform (CFFT), the proposed architecture takes advantage of under utilized hardware in the serial architecture to derive L-parallel architectures without increasing the hardware complexity by a factor of L. The operating frequency of the proposed architecture can be decreased which in turn reduces the power consumption. Further, this paper presents new parallel-pipelined architectures for the computation of real-valued fast Fourier transform (RFFT). The proposed architectures exploit redundancy in the computation of FFT samples to reduce the hardware complexity. A comparison is drawn between the proposed designs and the previous architectures. The power consumption can be reduced up to 37% and 50% in 2-parallel CFFT and RFFT architectures, respectively. The output samples are obtained in a scrambled order in the proposed architectures. Circuits to reorder these scrambled output sequences to a desired order are presented.**

***Index Terms*—Fast Fourier transform (FFT), folding, parallel processing, pipelining, radix-2^2, radix-2^3, real signals, register minimization, reordering circuit.**

I. Introduction

FAST FOURIER TRANSFORM (FFT) is widely used in the field of digital signal processing (DSP) such as filtering, spectral analysis, etc., to compute the discrete Fourier transform (DFT). FFT plays a critical role in modern digital communications such as digital video broadcasting and orthogonal frequency division multiplexing (OFDM) systems. Much research has been carried out on designing pipelined architectures for computation of FFT of complex valued signals (CFFT). Various algorithms have been developed to reduce the computational complexity, of which Cooley-Tukey radix-2 FFT [1] is very popular.

Algorithms including radix-4 [2], split-radix [3], radix-2^2 [4] have been developed based on the basic radix-2 FFT approach. The architectures based on these algorithms are some of the standard FFT architectures in the literature [5]–[10]. Radix-2 multi-path delay commutator (R2MDC) [5] is one of the most classical approaches for pipelined implementation of radix-2 FFT. Efficient usage of the storage buffer in R2MDC leads to the Radix-2 Single-path delay feedback (R2SDF) architecture with reduced memory [6]. R4SDF [7] and R4MDC [8], [9] have been proposed as radix-4 versions of R2SDF and R4MDC, respectively. Radix-4 single-path delay commutator (R4SDC) [10] is proposed using a modified radix-4 algorithm to reduce the complexity of the R4MDC architecture.

Manuscript received August 13, 2010; revised February 07, 2011; accepted April 02, 2011. Date of publication May 27, 2011; date of current version May 05, 2012. This work was carried out at Leanics Corporation.

The authors are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: ayina004@umn.edu; brow1283@umn.edu; parhi@umn.edu).

Digital Object Identifier 10.1109/TVLSI.2011.2147338

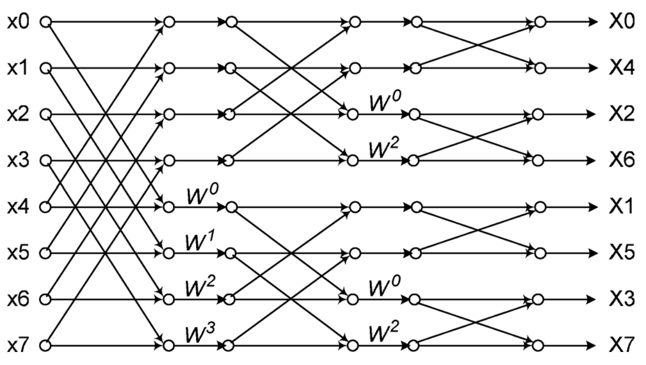


Fig. 1. Flow graph of a radix-2 8-point DIF FFT.

Many parallel architectures for FFT have been proposed in the literature [11]–[15]. In [11] and [12], architectures are developed for a specific N-point FFT, whereas hypercube theory is used to derive the architectures in [15]. A formal method of developing these architectures from the algorithms is not well established. Further, most of these hardware architectures are not fully utilized and require high hardware complexity. In the era of high speed digital communications, high throughput and low power designs are required to meet the speed and power requirements while keeping the hardware overhead to a minimum. In this paper, we present a new approach to design these architectures from the FFT flow graphs. Folding transformation [16] and register minimization techniques [17], [18] are used to derive several known FFT architectures. Several novel architectures are also developed using the proposed methodology which have not been presented in the literature before.

When the input samples are real, the spectrum is symmetric and approximately half of the operations are redundant. In applications such as speech, audio, image, radar, and biomedical signal processing [19]–[22], a specialized hardware implementation is best suited to meet the real-time constraints. This type of implementation also saves power in implantable or portable devices which is a key constraint. Few pipelined architectures for real valued signals have been proposed [23], [24] based on the Brunn algorithm. However, these are not widely used. In [23]–[25] different algorithms have been proposed for computation of RFFT. These approaches are based on removing the redundancies of the CFFT when the input is real. These can be

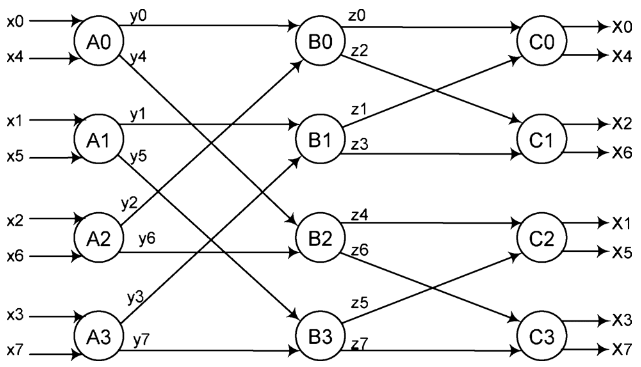


Fig. 2. Data Flow graph (DFG) of a radix-2 8-point DIF FFT.

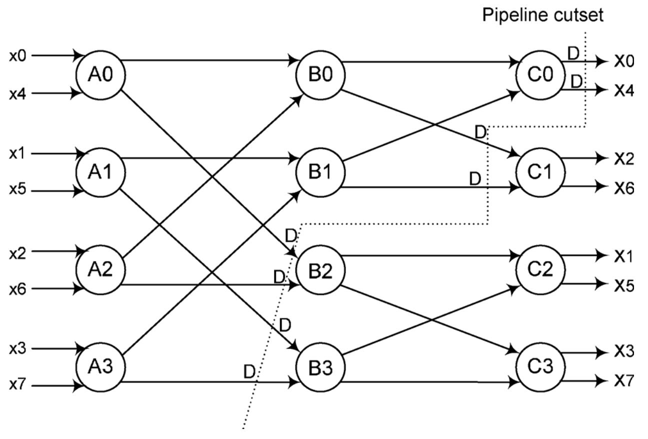


Fig. 3. Pipelined DFG of a 8-point DIF FFT as a preprocessing step for folding.

efficiently used in a digital signal processor compared to a specialized hardware implementation. In [26], RFFT is calculated using the CFFT architecture in an efficient manner. One such algorithm is the doubling algorithm, where an existing CFFT is used to calculate two RFFTs simultaneously. Another one is the packing algorithm, which forms a complex sequence of length $N/2$ taking the even and odd indexed samples of a real input sequence of length N, and computes the $N/2$-point CFFT of the complex sequence. In these algorithms, additional operations are necessary to obtain the final results, i.e., the outputs of the CFFT. In [27], a pipelined architecture has been proposed for the RFFT using the symmetric property of the RFFT based on the radix-2 algorithm.

This paper presents a novel approach to design of FFT architectures based on the folding transformation. In the folding transformation, many butterflies in the same column can be mapped to one butterfly unit. If the FFT size is N, a folding factor of $N/2$ leads to a 2-parallel architecture. In another design, we can choose a folding factor of $N/4$ to design a 4-parallel architectures, where four samples are processed in the same clock cycle. Different folding sets lead to a family of FFT architectures. Alternatively, known FFT architectures can also be described by the proposed methodology by selecting the appropriate folding set. Folding sets are designed intuitively to reduce latency and to reduce the number of storage elements. It may be noted that prior FFT architectures were derived in an *adhoc* way, and their derivations were not explained in a systematic way. This is the first attempt to generalize design of FFT architectures for arbitrary level of parallelism in a systematic manner via the folding transformation. In this paper,

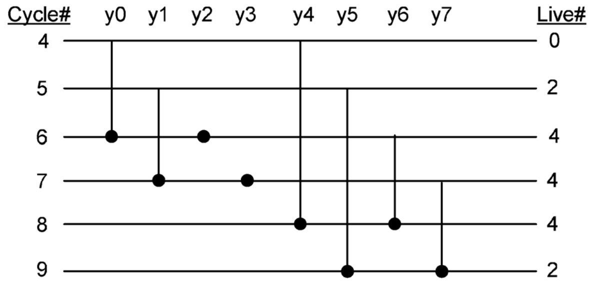
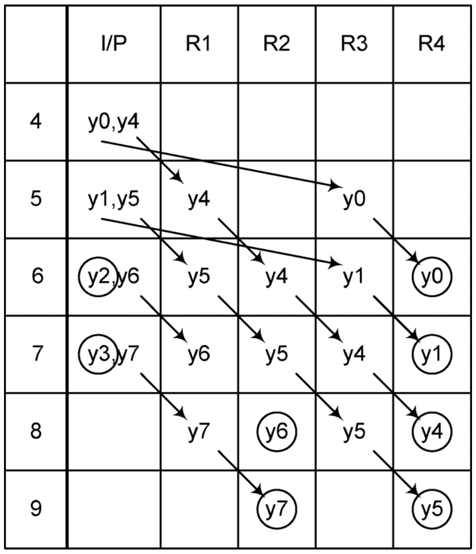


Fig. 4. Linear lifetime chart for the variables $y0, y1, \ldots, y7$.



	I/P	R1	R2	R3	R4
4	y0,y4				
5	y1,y5	y4		y0	
6	y2,y6	y5	y4	y1	y0
7	y3,y7	y6	y5	y4	y1
8		y7	y6	y5	y4
9			y7		y5

Fig. 5. Register allocation table for the data represented in Fig. 4.

design of prior architectures is first explained by constructing specific folding sets. Then, several new architecures are derived for various radices, and various levels of parallelism, and for either the decimation-in-time (DIT) or the decimation-in-frequency (DIF) flow graphs. All new architectures achieve full hardware utilization. It may be noted that all prior parallel FFT architectures did not achieve full hardware utilization.

Various new real FFT architectures are also presented based on higher radices. It may be noted that the approach in [27] cannot be generalized for radices higher than 2. This drawback is removed by the proposed methodology. In this paper, we propose several novel parallel-pipelined architectures for the computation of RFFT based on radix-2^2 [4] and radix-2^3 algorithms [28]. It combines the advantages of radix-2^n algorithms, which require fewer complex multipliers when compared to radix-2 algorithm, with the reduction of operations using redundancy.

This paper is organized as follows. The folding transformation-based FFT architectures design is presented in Section II. In Section III, we describe the proposed architectures for complex FFT. The architectures which are specific to RFFT are explained in Section IV. In Section V, the proposed architectures are compared with the previous approaches and some conclusions are drawn in Section VI. Reordering of the output samples for the proposed architectures is presented in the appendix.

II. FFT Architectures via Folding

In this section, we illustrate the folding transformation method to derive several known FFT architectures in general.

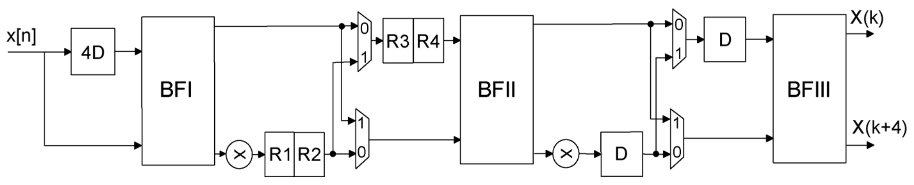


Fig. 6. Folded architecture for the DFG in Fig. 3. This corresponds to the well known R2MDC architecture.

TABLE I
DATAFLOW AND MUX CONTROL OF THE ARCHITECTURE SHOWN IN FIG. 6

#Cycle	Dataflow	Control
4	y0 → R3	0
5	y1 → R3	0
6	y2 → i/p R2 → R3	1
7	y3 → i/p R2 → R3	1
8	R2 → i/p	0
9	R2 → i/p	0

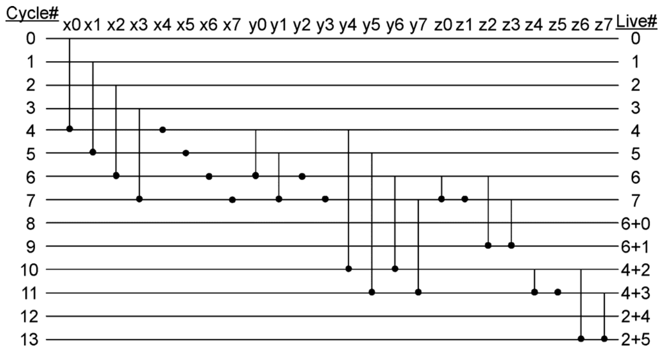
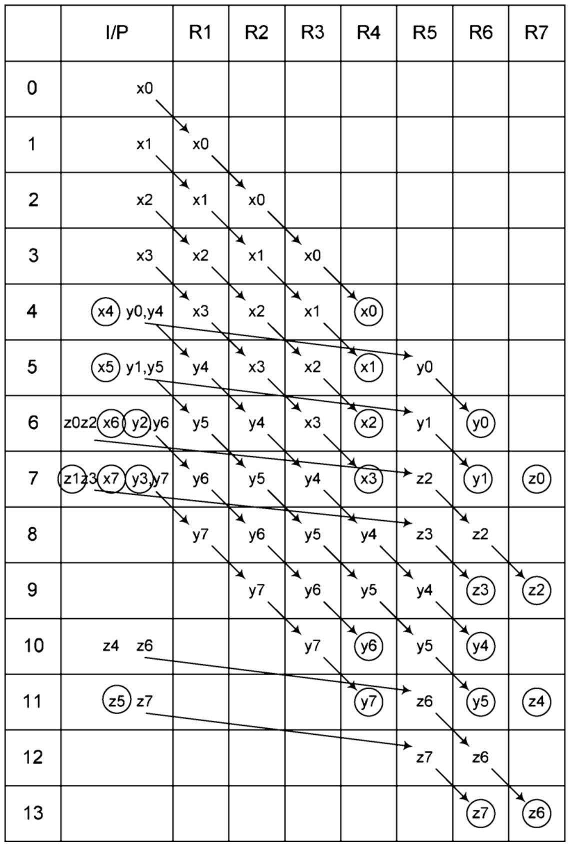


Fig. 7. Linear lifetime chart for variables for a 8-point FFT architecture.



	I/P	R1	R2	R3	R4	R5	R6	R7
0	x0							
1	x1	x0						
2	x2	x1	x0					
3	x3	x2	x1	x0				
4	x4 y0,y4	x3	x2	x1	x0			
5	x5 y1,y5	y4	x3	x2	x1	y0		
6	z0z2 x6 y2 y6	y5	y4	x3	x2	y1	y0	
7	z1z3 x7 y3 y7	y6	y5	y4	x3	z2	y1	z0
8		y7	y6	y5	y4	z3	z2	
9			y7	y6	y5	y4	z3	z2
10	z4 z6			y7	y6	y5	y4	
11	z5 z7				y7	z6	y5	z4
12						z7	z6	
13							z7	z6

Fig. 8. Register allocation table for the data represented in Fig. 7.

The process is described using an 8-point radix-2 DIF FFT as an example. It can be extended to other radices in a similar fashion. Fig. 1 shows the flow graph of a radix-2 8-point DIF FFT. The graph is divided into three stages and each of them consists of a set of butterflies and multipliers. The twiddle factor in between the stages indicates a multiplication by W_N^k, where W_N denotes the Nth root of unity, with its exponent evaluated modulo N. This algorithm can be represented as a data flow graph (DFG) as shown in Fig. 2. The nodes in the DFG represent tasks or computations. In this case, all the nodes represent the butterfly computations of the radix-2 FFT algorithm. In particular, assume nodes A and B have the multiplier operation on the bottom edge of the butterfly.

The folding transformation [16], [29] is used on the DFG in Fig. 2 to derive a pipelined architecture. To transform the DFG, we require a folding set, which is an ordered set of operations executed by the same functional unit. Each folding set contains K entries some of which may be null operations. K is called the folding factor, i.e., the number of operations folded into a single functional unit. The operation in the jth position within the folding set (where j goes from 0 to $K-1$) is executed by the functional unit during the time partition j. The term j is the folding order, i.e., the time instance to which the node is scheduled to be executed in hardware. For example, consider the folding set $A = \{\phi, \phi, \phi, \phi, A0, A1, A2, A3\}$ for $K = 8$. The operation $A0$ belongs to the folding set A with the folding order 4. The functional unit A executes the operations $A0, A1, A2, A3$ at the respective time instances and will be idle during the null operations. We use the systematic folding techniques to derive the 8-point FFT architecture. Consider an edge e connecting the nodes U and V with $w(e)$ delays. Let the executions of the lth iteration of the nodes U and V be scheduled at the time units $Kl + u$ and $Kl + v$, respectively, where u and v are the folding orders of the nodes U and V, respectively. The folding equation for the edge e is

$$D_F(U \rightarrow V) = Kw(e) - P_U + v - u \tag{1}$$

where P_U is the number of pipeline stages in the hardware unit which executes the node U [16].

A. Feed-Forward Architecture

Consider folding of the DFG in Fig. 2 with the folding sets

$$A = \{\phi, \phi, \phi, \phi, A0, A1, A2, A3\}$$
$$B = \{B2, B3, \phi, \phi, \phi, \phi, B0, B1\}$$
$$C = \{C1, C2, C3, \phi, \phi, \phi, \phi, C0\}.$$

Assume that the butterfly operations do not have any pipeline stages, i.e., $P_A = 0$, $P_B = 0$, $P_C = 0$. The folded architecture

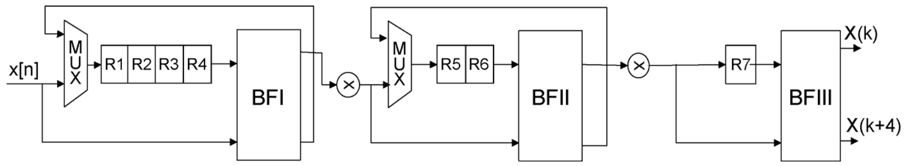


Fig. 9. Folded architecture for the DFG in Fig. 3. This corresponds to the well known R2SDF architecture.

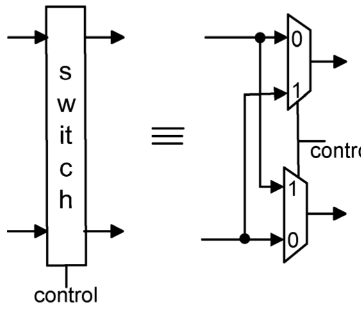


Fig. 10. Switch equivalent circuit.

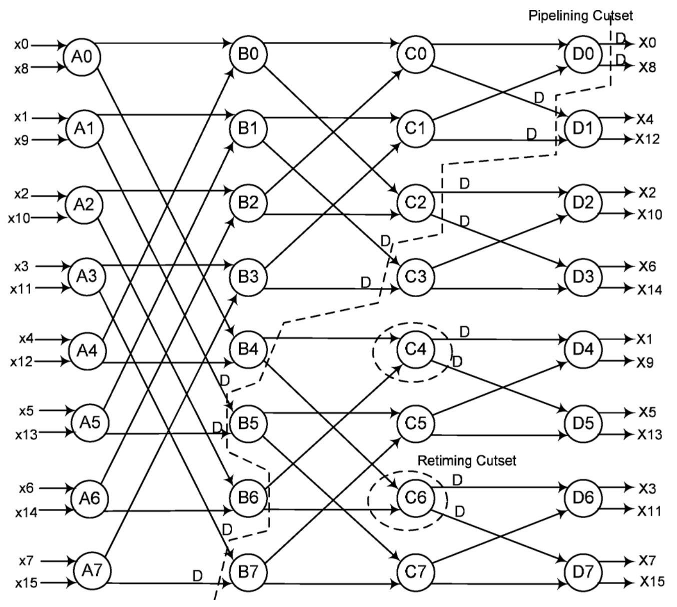


Fig. 11. DFG of a Radix-2 16-point DIF FFT with retiming fr folding.

can be derived by writing the folding equation in (1) for all the edges in the DFG. These equations are

$$\begin{aligned}
&D_F(A0 \to B0) = 2 && D_F(B0 \to C0) = 1\\
&D_F(A0 \to B2) = -4 && D_F(B0 \to C1) = -6\\
&D_F(A1 \to B1) = 2 && D_F(B1 \to C0) = 0\\
&D_F(A1 \to B1) = -4 && D_F(B1 \to C1) = -7\\
&D_F(A2 \to B0) = 0 && D_F(B2 \to C2) = 1\\
&D_F(A2 \to B2) = -6 && D_F(B2 \to C3) = 2\\
&D_F(A3 \to B1) = 0 && D_F(B3 \to C2) = 0\\
&D_F(A3 \to B3) = -6 && D_F(B3 \to C3) = 1. \qquad (2)
\end{aligned}$$

For example, $D_F(A0 \to B0) = 2$ means that there is an edge from the butterfly node A to node B in the folded DFG with two delays. For the folded system to be realizable, $D_F(U \to V) \geq 0$ must hold for all the edges in the DFG. Retiming and/or pipelining can be used to either satisfy this property or determine that the folding sets are not feasible [16], [29]. We can observe the negative delays on some edges in (2). The DFG can be pipelined as shown in Fig. 3 to ensure that folded hardware has non-negative number of delays. The folded delays for the pipelined DFG are given by

$$\begin{aligned}
&D_F(A0 \to B0) = 2 && D_F(B0 \to C0) = 1\\
&D_F(A0 \to B2) = 4 && D_F(B0 \to C1) = 2\\
&D_F(A1 \to B1) = 2 && D_F(B1 \to C0) = 0\\
&D_F(A1 \to B1) = 4 && D_F(B1 \to C1) = 1\\
&D_F(A2 \to B0) = 0 && D_F(B2 \to C2) = 1\\
&D_F(A2 \to B2) = 2 && D_F(B2 \to C3) = 2\\
&D_F(A3 \to B1) = 0 && D_F(B3 \to C2) = 0\\
&D_F(A3 \to B3) = 2 && D_F(B3 \to C3). = 1 \qquad (3)
\end{aligned}$$

From (3), we can observe that 24 registers are required to implement the folded architecture. Lifetime analysis technique [18] is used to design the folded architecture with minimum possible registers. For example, in the current 8-point FFT design, consider the variables $y0, y1, \ldots y7$, i.e., the outputs at the nodes $A0, A1, A2, A3$, respectively. It takes 16 registers to synthesize these edges in the folded architecture. The linear lifetime chart for these variables is shown in Fig. 4. From the lifetime chart, it can be seen that the folded architecture requires 4 registers as opposed to 16 registers in a straightforward implementation. The next step is to perform forward-backward register allocation. The allocation table is shown in Fig. 5. Similarly, we can apply lifetime analysis and register allocation techniques for the variables $x0, \ldots, x7$ and $z0, \ldots, z7$, inputs to the DFG and the outputs from nodes $B0$, $B1$, $B2$, $B3$, respectively, as shown in Fig. 2. From the allocation table in Fig. 5 and the folding equations in (3), the final architecture in Fig. 6 can be synthesized. The dataflow through the 1st stage multiplexers is shown in Table I. The control signals for all the MUXes can be easily generated using a 2-bit counter. The control signals for the 1st stage and 2nd stage muxes are obtained by dividing clock signal by 4 and by 2, respectively. We can observe that the derived architecture is the same as the R2MDC architecture [5]. Similarly, the R2SDF architecture can be derived by minimizing the registers on all the variables at once.

B. Feedback Architecture

We derive the feedback architecture using the same 8-point radix-2 DIT FFT example in Fig. 1. Consider the following folding sets:

$$\begin{aligned}
A &= \{\phi, \phi, \phi, \phi, A0, A1, A2, A3\}\\
B &= \{\phi, \phi, B2, B3, \phi, \phi, B0, B1\}\\
C &= \{\phi, C1, \phi, C2, \phi, C3, \phi, C0\}.
\end{aligned}$$

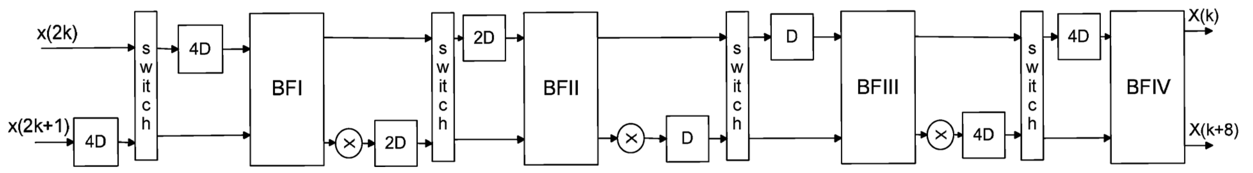


Fig. 12. Proposed 2-parallel (Architecture 1) for the computation of a radix-2 16-point DIF complex FFT.

Assume that the butterfly operations do not have any pipeline stages, i.e., $P_A = 0$, $P_B = 0$, $P_C = 0$. The folded architecture can be derived by writing the folding equation in (1) for all the edges in the DFG. The folding equations are

$$\begin{aligned} D_F(A0 \to B0) &= 2 & D_F(B0 \to C0) &= 1\\ D_F(A0 \to B2) &= -2 & D_F(B0 \to C1) &= -5\\ D_F(A1 \to B1) &= 2 & D_F(B1 \to C0) &= 0\\ D_F(A1 \to B1) &= -2 & D_F(B1 \to C1) &= -6\\ D_F(A2 \to B0) &= 0 & D_F(B2 \to C2) &= 1\\ D_F(A2 \to B2) &= -4 & D_F(B2 \to C3) &= 3\\ D_F(A3 \to B1) &= 0 & D_F(B3 \to C2) &= 0\\ D_F(A3 \to B3) &= -4 & D_F(B3 \to C3) &= 2. \end{aligned} \tag{4}$$

It can be seen from the folding equations in (4) that some edges contain negative delays. Retiming is used to make sure that the folded hardware has non-negative number of delays. The pipelined DFG is the same as the one in the feed-forward example and is shown in Fig. 3. The updated folding equations are shown in (5)

$$\begin{aligned} D_F(A0 \to B0) &= 2 & D_F(B0 \to C0) &= 1\\ D_F(A0 \to B2) &= 6 & D_F(B0 \to C1) &= 3\\ D_F(A1 \to B1) &= 2 & D_F(B1 \to C0) &= 0\\ D_F(A1 \to B1) &= 6 & D_F(B1 \to C1) &= 2\\ D_F(A2 \to B0) &= 0 & D_F(B2 \to C2) &= 1\\ D_F(A2 \to B2) &= 4 & D_F(B2 \to C3) &= 3\\ D_F(A3 \to B1) &= 0 & D_F(B3 \to C2) &= 0\\ D_F(A3 \to B3) &= 4 & D_F(B3 \to C3) &= 2. \end{aligned} \tag{5}$$

The number of registers required to implement the folding equations in (5) is 40. The linear life time chart is shown in Fig. 7 and the register allocation table is shown in Fig. 8. We can implement the same equations using seven registers by using the register minimization techniques. The folded architecture in Fig. 9 is synthesized using the folding equations in (5) and register allocation table.

The hardware utilization is only 50% in the derived architecture. This can also be observed from the folding sets where half of the time null operations are being executed, i.e., hardware is idle. Using this methodology, we now present new low-power parallel FFT architectures in the next two sections.

III. Proposed Architectures With Complex Inputs (CFFT)

In this section, we present parallel architectures for complex valued signals based on radix-2 and radix-2^3 algorithms. These architectures are derived using the approach presented in the previous section. The same approach can be extended to radix-2^2, radix-2^4 and other radices as well. Due to space constraints, only folding sets are presented for different architectures. The folding equations and register allocation tables can be obtained easily. For the rest of the paper, MUX control circuit is replaced by a switch block as shown in Fig. 10. The control signals for these switches can be generated by using a log_2N-bit counter. Different output bits of the counter will control the switches in different stages of the FFT.

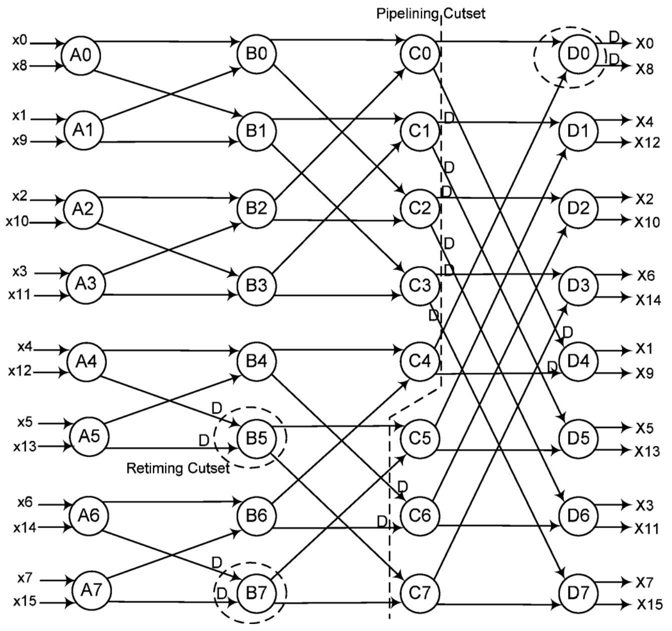


Fig. 13. DFG of a Radix-2 16-point DIT FFT with retiming for folding.

A. 2-Parallel Radix-2 FFT Architecture

The utilization of hardware components in the feed-forward architecture is only 50%. This can also be observed from the folding sets of the DFG where half of the time null operations are being executed. We can derive new architectures by changing the folding sets which can lead to efficient architectures in terms of hardware utilization and power consumption. We present here one such example of a 2-parallel architecture which leads to 100% hardware utilization and consumes less power.

Fig. 11 shows the DFG of radix-2 DIF FFT for $N = 16$. All the nodes in this figure represent radix-2 butterfly operations. Assume the nodes A, B, and C contain the multiplier operation at the bottom output of the butterfly. Consider the folding sets

$$\begin{aligned} A &= \{A0, A2, A4, A6, A1, A3, A5, A7\}\\ B &= \{B5, B7, B0, B2, B4, B6, B1, B3\},\\ C &= \{C3, C5, C7, C0, C2, C4, C6, C1\}\\ D &= \{D2, D4, D6, D1, D3, D5, D7, D0\}. \end{aligned} \tag{6}$$

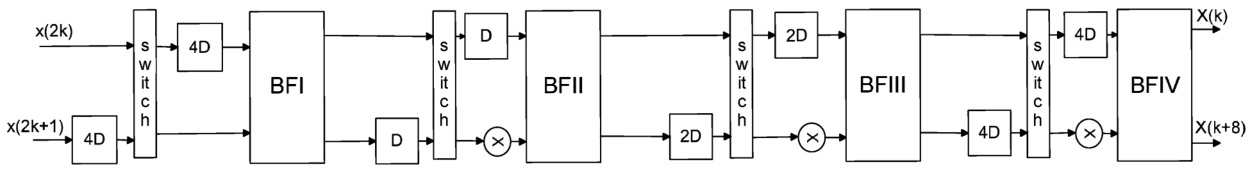


Fig. 14. Proposed 2-parallel (Architecture 1) for the computation of a radix-2 16-point DIT Complex FFT.

We can derive the folded architecture by writing the folding equation in (1) for all the edges. Pipelining and retiming are required to get non-negative delays in the folded architecture. The DFG in Fig. 11 also shows the retimed delays on some of the edges of the graph. The final folded architecture is shown in Fig. 12. The register minimization techniques and forward-backward register allocation are also applied in deriving this architecture as described in Section II. Note the similarity of the datapath to R2MDC. This architecture processes two input samples at the same time instead of one sample in R2MDC. The implementation uses regular radix-2 butterflies. Due to the spatial regularity of the radix-2 algorithm, the synchronization control of the design is very simple. A $\log_2 N$-bit counter serves two purposes: synchronization controller, i.e., the control input to the switches, and address counter for twiddle factor selection in each stage.

We can observe that the hardware utilization is 100% in this architecture. In a general case of N-point FFT, with N power of 2, the architecture requires $\log_2(N)$ complex butterflies, $\log_2(N) - 1$ complex multipliers and $3N/2 - 2$ delay elements or buffers.

In a similar manner, we can derive the 2-parallel architecture for Radix-2 DIT FFT using the following folding sets. Assume that multiplier is at the bottom input of the nodes B, C, D.

$$\begin{aligned} A &= \{A0, A2, A1, A3, A4, A6, A5, A7\} \\ B &= \{B5, B7, B0, B2, B1, B3, B4, B6\}, \\ C &= \{C6, C5, C7, C0, C2, C1, C3, C4\} \\ D &= \{D2, D1, D3, D4, D6, D5, D7, D0\}. \end{aligned}$$

The pipelined/retimed version of the DFG is shown in Fig. 13, and the 2-parallel architecture is in Fig. 14. The main difference in the two architectures (see Figs. 12 and 14) is the position of the multiplier in between the butterflies.

B. 4-Parallel Radix-2 FFT Architecture

A 4-parallel architecture can be derived using the following folding sets:

$$\begin{aligned} A &= \{A0, A1, A2, A3\} \quad A' = \{A'0, A'1, A'2, A'3\} \\ B &= \{B1, B3, B0, B2\} \quad B' = \{B'1, B'3, B'0, B'2\} \\ C &= \{C2, C1, C3, C0\} \quad C' = \{C'2, C'1, C'3, C'0\} \\ D &= \{D3, D0, D2, D1\} \quad D' = \{D'3, D'0, D'2, D'1\}. \end{aligned}$$

The DFG shown in Fig. 15 is retimed to get non-negative folded delays. The final architecture in Fig. 16 can be obtained following the proposed approach. For a N-point FFT, the architecture takes $4(log_4 N - 1)$ complex multipliers and $2N - 4$ delay elements. We can observe that hardware complexity is almost double that of the serial architecture and processes four-samples

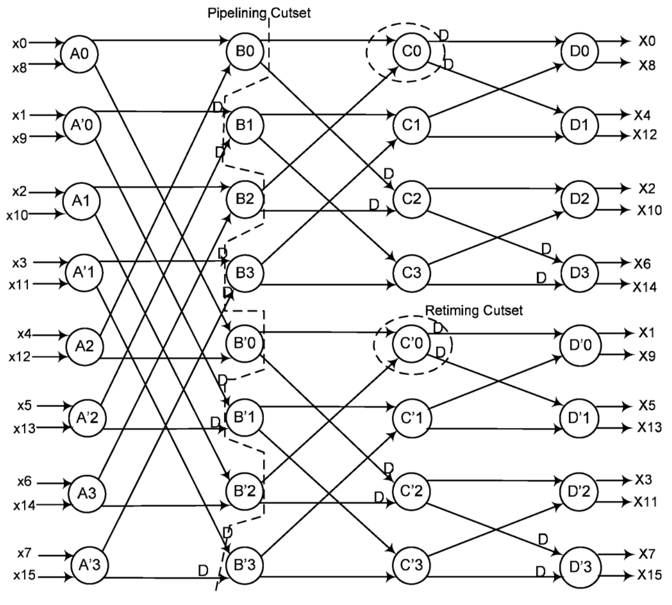


Fig. 15. DFG of a Radix-2 16-point DIF FFT with retiming for folding for 4-parallel architecture.

in parallel. The power consumption can be reduced by 50% (see Section V) by lowering the operational frequency of the circuit.

C. Radix-2^3 FFT Architecture

The hardware complexity in the parallel architectures can be further reduced by using radix-2^n FFT algorithms. We consider the example of a 64-point radix-2^3 FFT algorithm [28]. The advantage of radix-2^3 over radix-2 algorithm is its multiplicative complexity reduction. A 2-parallel architecture is derived using folding sets in (6). Here the DFG contains 32 nodes instead of 8 in 16-point FFT. The folded architecture is shown in Fig. 17. The design contains only two full multipliers and two constant multipliers. The constant multiplier can be implemented using canonic signed digit (CSD) format with much less hardware compared to a full multiplier. For an N-point FFT, where N is a power of 2^3, the proposed architecture requires $2(\log_8 N - 1)$ multipliers and $3N/2 - 2$ delays. The multiplication complexity can be halved by computing the two operations using one multiplier. This can be seen in the modified architecture shown in Fig. 18. The only disadvantage of this design is that two different clocks are needed. The multiplier has to be operated at double the frequency compared to the rest of the design. The architecture requires only $\log_8 N - 1$ multipliers.

A 4-parallel radix-2^3 architecture can be derived similar to the 4-parallel radix-2 FFT architecture. A large number of architectures can be derived using the approach presented in Section II. Using the folding sets of same pattern, 2-parallel and 4-parallel architectures can be derived for radix-2^2 and radix-2^4 algorithms. We show that changing the folding sets can lead to different parallel architectures. Further DIT and DIF

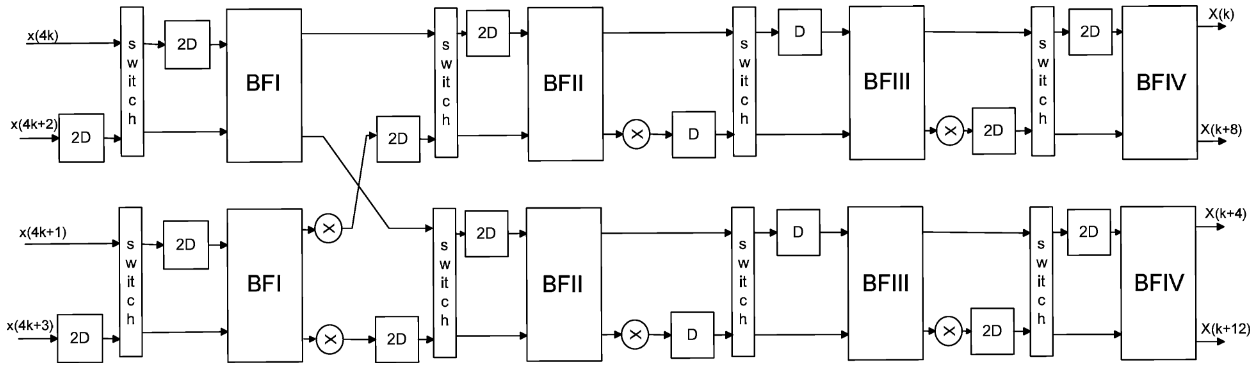


Fig. 16. Proposed 4-parallel (Architecture 2) for the computation of 16-point radix-2 DIF Complex FFT.

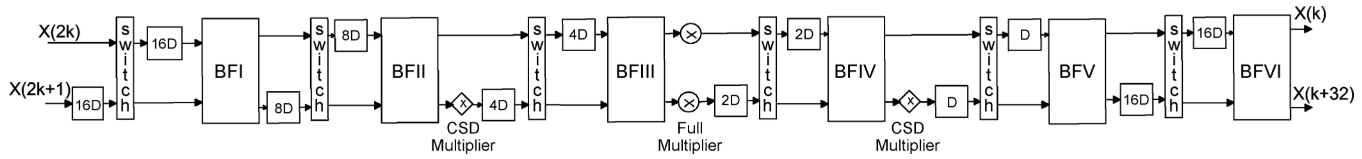


Fig. 17. Proposed 2-parallel (Architecture 3) for the computation of 64-point radix-2^3 DIF complex FFT.

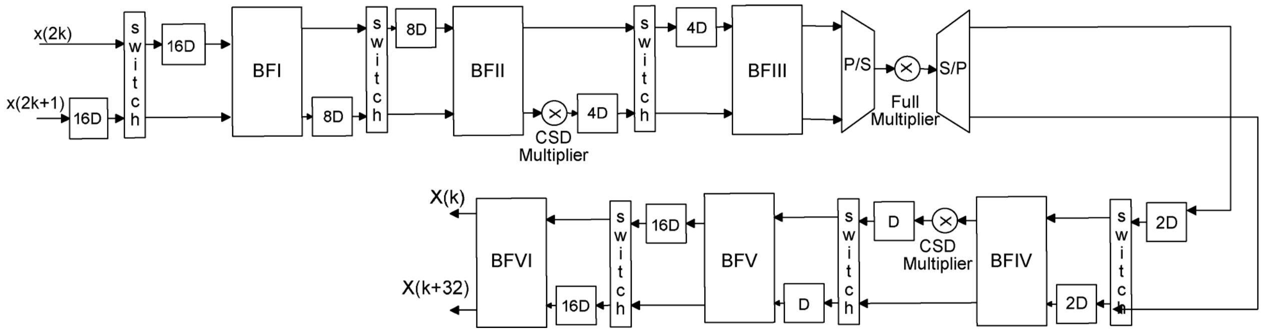


Fig. 18. Proposed 2-parallel (Architecture 4) for the computation of 64-point radix-2^3 DIF complex FFT.

algorithms will lead to similar architectures except the position of the multiplier operation.

D. Deriving Folding Sets

Folding sets for a given DFG are determined based on scheduling, i.e., assigning execution times to the operations in the DFG such that the precedence constraints of the DFG are not violated. Scheduling in the context of high level synthesis for ASIC design for DSP applications has been studied in the past [30]–[33]. The scheduling algorithms are either based on integer linear programming (ILP) or other heuristic approaches. Further, [34] presents algorithms for exhaustively generating all of the solutions (folding sets) to the scheduling problem. However in this paper, the folding sets are derived manually based on simple observations.

IV. Proposed Architectures with Real Inputs (RFFT)

If the inputs are complex, we need all the internal computations and outputs of the flow graph to obtain the CFFT. On the other hand, if inputs are real, we can simplify the graph using the RFFT properties as follows.

For RFFT, the input sequence is assumed to be real, i.e., $\forall n, x[n] \in \Re$. It is easy to show that, if $x[n]$ is real, then the output $X[k]$ is symmetric, i.e.,

$$X[N-k] = X^*[k].$$

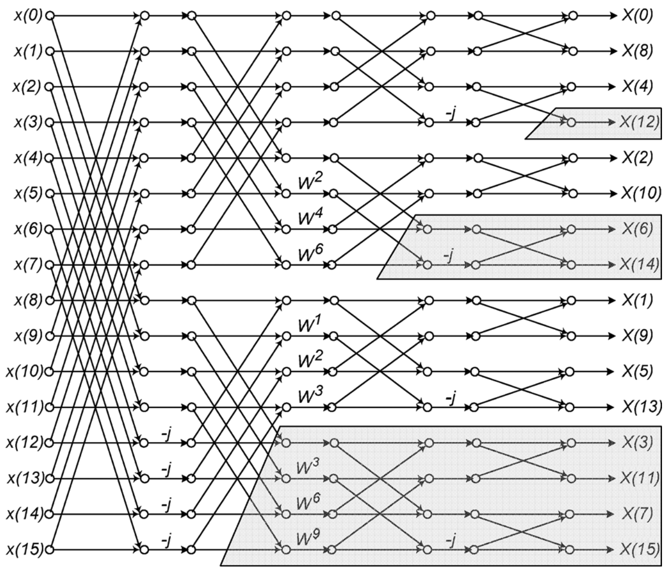


Fig. 19. Flow graph of a radix-2^2 16-point DIF FFT. The boxed regions are redundant operations for RFFT.

Using this property, $(N/2) - 1$ outputs can be removed which are redundant. Most of the approaches in literature obtain the frequencies with indices $k = [0, N/2]$ or $k = [0, N/4] \bigcup [N/2, 3N/4]$. A new approach to find the redundant samples, which simplifies the hardware implementation has been proposed in [27]. We use this approach [27],

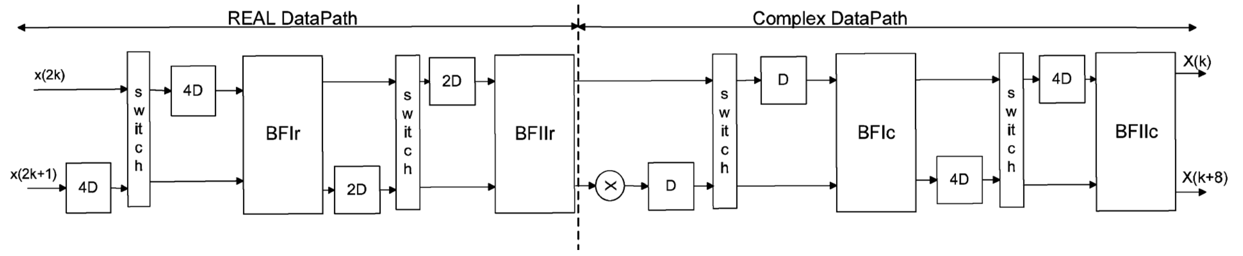


Fig. 20. Proposed 2-parallel (Architecture 5) for the computation of 16-point radix-2^2 DIF RFFT.

to find the redundant samples in the flow graph. The shaded regions in Fig. 19 can be removed and only $N/2+1$ outputs of the FFT are needed.

The other simplification is that $Im(x[n])=0$, since inputs are real. According to this, every piece of data is real until it is multiplied by a complex number. Thus, the additions performed on this data are real. Thus we can save few adders required in implementing the FFT in hardware. We propose novel pipelined architectures based on these modifications which can process two samples in parallel. Two of these architectures are proposed in [35], which are derived using the new approach. Further this idea is extended to radix-2^3 algorithm.

A. 2-Parallel Radix-2 Architecture

The DFG of the radix-2 DIF FFT is shown in Fig. 11. The pipelined architecture for RFFT can be derived with the following folding sets similar to the CFFT architecture

$$\begin{aligned} A &= \{A0, A2, A4, A6, A1, A3, A5, A7\} \\ B &= \{B5, B7, B0, B2, B4, B6, B1, B3\}, \\ C &= \{C3, C5, \phi, C0, C2, C4, \phi, C1\} \\ D &= \{D2, D4, \phi, D1, \phi, D5, \phi, D0\}. \end{aligned}$$

The architecture will be similar to the radix-2 DIF CFFT architecture shown in Fig. 12 except that the first two stages of the butterfly will contain a real datapath. The hardware complexity of this architecture is similar to the CFFT architecture. Although the proposed architecture and the architecture in [27] have same hardware complexity, the proposed architecture is more regular. By extending the same approach to higher radices, multiplier complexity can be reduced.

B. 2-Parallel Radix-2^2 Architecture

The direct mapping of the radix-2^2 DIF FFT algorithm to a hardware architecture is simple. We can simply use the Radix-2^2 single-path delay feedback ($\mathrm{R2^2SDF}$) approach presented in [4], with just modifying the first complex butterfly stage into real. But, this cannot exploit the properties of the RFFT, where almost half of the output samples are redundant. In $\mathrm{R2^2SDF}$ architecture, we can also observe that the utilization of multipliers will be 50% after the redundant operations are removed. Therefore, we propose a novel architecture for computing real FFT based on the flow graph shown in Fig. 19. The proposed architecture can process two input samples in parallel as opposed to serial architectures proposed in the literature. Two different architectures are derived using two different scheduling approaches, i.e., changing the folding order of the butterfly nodes.

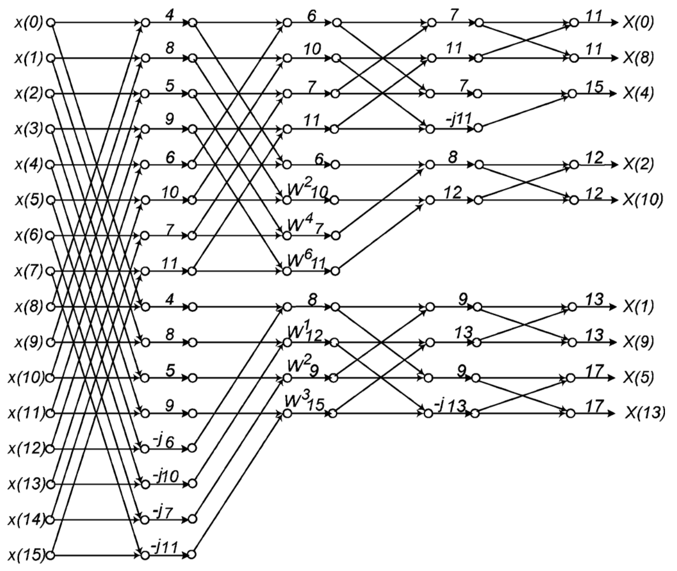


Fig. 21. Simplified flow graph of a 16-point radix-2^2 DIF RFFT along with the proposed scheduling 1.

1) Scheduling Method 1: Fig. 20 shows the proposed parallel-pipelined architecture for a 16-point DIF RFFT obtained from the flow graph in Fig. 19. The architecture can be derived using the following folding sets:

$$\begin{aligned} A &= \{A0, A2, A4, A6, A1, A3, A5, A7\} \\ B &= \{B5, B7, B0, B2, B4, B6, B1, B3\}, \\ C &= \{C3, C5, \phi, C0, C2, C4, \phi, C1\} \\ D &= \{D2, D4, \phi, D1, \phi, D5, \phi, D0\}. \end{aligned}$$

The nodes from $A0, \ldots, A7$ represent the eight butterflies in the first stage of the FFT and $B0, \ldots, B7$ represent the butterflies in the second stage and so on. Assume the butterflies $B0, \ldots, B7$ have only one multiplier at the bottom output instead of on both outputs. This assumption is valid only for the RFFT case due to the redundant operations. This radix-2^2 feed-forward pipelined architecture maximizes the utilization of multipliers and processes two input samples per clock cycle. As shown in the flow graph, all the edges in the first two stages carry real samples and later stages carry complex samples. Therefore, the radix-2 butterflies in the first two stages process two real inputs and they consist of only a real adder and a real subtractor. The butterflies and the multipliers in the remaining stages operate on complex inputs. In a general case of N-point RFFT, with N power of 2, the architecture requires $\log_2(N)-1$ real butterflies, $\log_4(N)-1$ complex multipliers and $9N/8-2$ delay elements or buffers.

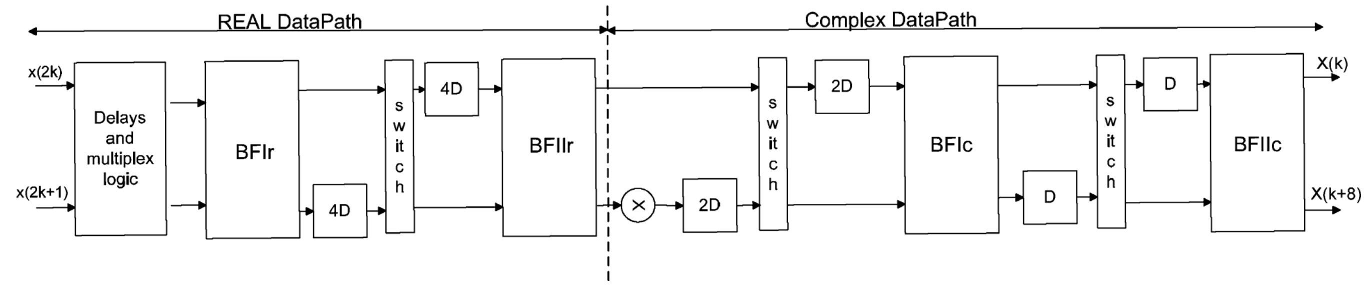
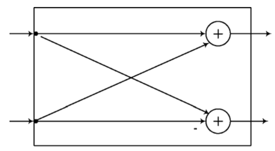
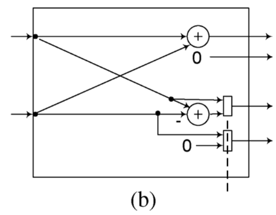


Fig. 22. Proposed 2-parallel (Architecture 6) for the computation of 16-point radix-2^2 DIF RFFT.

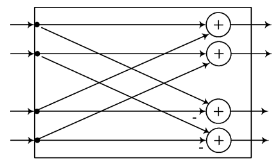


(a)

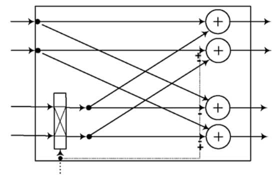


(b)

Fig. 23. Butterfly structure for the proposed FFT architecture in the real datapath. (a) BF2Ir. (b) BF2IIr.



(a)



(b)

Fig. 24. Butterfly structure for the proposed FFT architecture in the complex datapath. (a) BF2Ic. (b) BF2IIc.

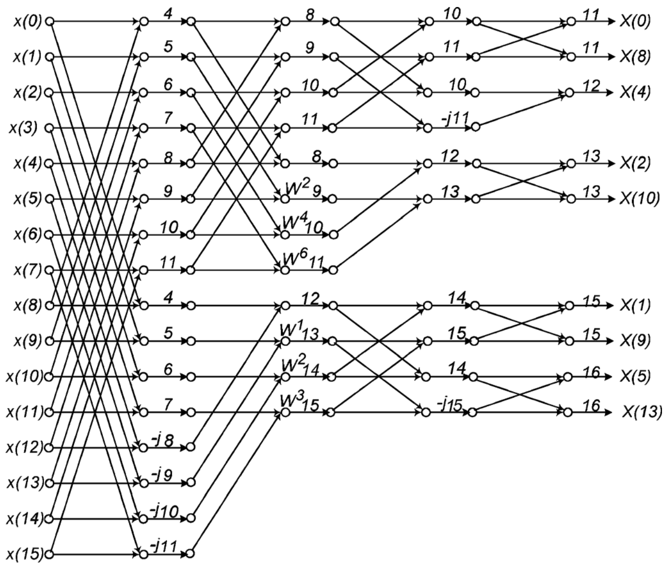


Fig. 25. Simplified flow graph of a 16-point radix-2^2 DIF RFFT along with the proposed scheduling 2.

The scheduling for the proposed architecture in Fig. 20 is shown in Fig. 21. The numbers on the edges indicate the clock cycle numbers in which those intermediate samples are computed. The first two stages compute only real butterflies. According to the input order of the data, the first butterfly computes the pairs of samples in the following order: (0,8), (2,10), (4,12), (6,14), (1,9), (3,11), (5,13), (7,15). This circuit first processes the even samples and then the odd samples.

We can observe the similarity of the data-path to Radix-2 Multipath delay commutator (R2MDC) and the reduced number of multipliers. The implementation uses four types of butterflies as shown in the architecture. BF2Ir and BF2Ic are regular butterflies which handle real and complex data, respectively, as shown in Figs. 23 and 24. Although there is a multiplicative factor "$-j$" after the first stage, the first two stages consists of only real-valued datapath. We need to just combine the real and imaginary parts and send it as an input to the multiplier in the next stage. For this, we do not need a full complex butterfly stage. The factor $-j$ is handled in the second butterfly stage using a bypass logic which forwards the two samples as real and imaginary parts to the input of the multiplier. The adder and subtractor in the butterfly remains inactive during that time. Fig. 23(b) shows BF2IIr, a regular butterfly which handles real data and also contains logic to implement the twiddle factor "$-j$" multiplication after the first stage. Fig. 24(b) shows the complex butterfly structure for BF2IIc and contains logic to implement the twiddle factor $(-j)$ multiplication.

2) Scheduling Method 2: Another way of scheduling is proposed which modifies the architecture slightly and also reduces the required number of delay elements. In this scheduling, the input samples are processed sequentially, instead of processing the even and odd samples separately. This can be derived using the following folding sets:

$$\begin{aligned}A &= \{A0, A1, A2, A3, A4, A5, A6, A7\}\\ B &= \{B4, B5, B6, B7, B0, B1, B2, B3\},\\ C &= \{C2, C3, C4, C5, \phi, \phi, C0, C1\}\\ D &= \{D1, D2, \phi, D4, D5, \phi, \phi, D0\}.\end{aligned}$$

The nodes from $A0, \ldots, A7$ represent the eight butterflies in the first stage of the FFT and $B0, \ldots, B7$ represent the butterflies in the second stage. Assume the butterflies $B0, \ldots, B7$ have only one multiplier at the bottom output instead of both outputs.

TABLE II
COMPARISON OF PIPELINED HARDWARE ARCHITECTURES FOR THE COMPUTATION OF N-POINT CFFT

Architecture	# Multipliers	# Adders	# Delays	Control	Throughput
R2MDC	$2(log_4N-1)$	$4log_4N$	$3N/2-2$	simple	1
R2SDF	$2(log_4N-1)$	$4log_4N$	$N-1$	simple	1
R4SDC	(log_4N-1)	$3log_4N$	$2N-2$	complex	1
R2^2SDF	(log_4N-1)	$4log_4N$	$N-1$	simple	1
R2^3SDF*	(log_8N-1)	$4log_4N$	$N-1$	simple	1
Proposed Architectures					
Arch 1 (radix-2)	$2(log_4N-1)$	$4log_4N$	$3N/2-2$	simple	2
Arch 2 (radix-2)	$4(log_4N-1)$	$8log_4N$	$2N-4$	simple	4
Arch 3 (radix-2^3)*	$2(log_8N-1)$	$4log_4N$	$3N/2-2$	simple	2
Arch 4 (radix-2^3)*	log_8N-1	$4log_4N$	$3N/2-2$	simple	2

$*$ These architectures need two constant multipliers as described in Radix-2^3 *algorithm*

Fig. 22 shows the modified architecture and the corresponding scheduling is shown in Fig. 25. In a general case of N-point RFFT, with N power of 2, the architecture requires $\log_2(N)-1$ real butterflies, $\log_4(N)-1$ complex multipliers and $N-2$ delay elements or buffers. We can observe that the savings in number of delay elements is due to the last stage. In the last stage, we need to store complex samples, i.e., double delay elements are required to store both real and imaginary parts. By decreasing the number of delay elements in this stage, we are able to reduce the total number of delay elements required. The control logic becomes slightly complex for this scheduling as shown in the first stage. Register minimization techniques [29] are used to find the optimal number of registers required for this scheduling. $N/2$ registers are needed in the first stage to achieve the required scheduling.

C. Radix-2^3

The approach of radix-2^2 RFFT architecture can be extended to radix-2^3 algorithm which has low multiplier complexity. The radix-2^3 FFT algorithm is shown in Fig. 26. The shaded regions show the redundant operations for RFFT. We can observe that only one multiplier is required at the end of the third butterfly column due to redundancy. Two multipliers are required for a complex FFT as shown in Fig. 17.

The parallel architecture can be derived using the same kind of folding sets described in Scheduling Method I. The folded architecture is shown in Fig. 27. For an N-point RFFT with N power of 2^3, the architecture requires $\log_8(N)-1$ complex multipliers and $N-2$ delay elements or buffers. It also requires constant multipliers to perform the trivial multiplication operations. The advantage of this architecture is its reduced multiplier complexity.

Similar to 4-parallel radix-2 CFFT, 4-parallel radix-2^2, and radix-2^3 architectures for RFFT can be derived using the folding sets of the same pattern.

V. COMPARISON AND ANALYSIS

A comparison is made between the previous pipelined architectures and the proposed ones for the case of computing an N-point complex FFT in Table II and real FFT in Table III. The comparison is made in terms of required number of complex multipliers, adders, delay elements, throughput and control complexity. The control complexity is defined as simple if only a counter can generate the required control signals and as complex if the control circuitry needs complex logic circuit as in the case of R4SDC [10]. For RFFT, the proposed designs and the design in [27] are the only specific approaches for the computation of RFFT. The other approaches [4] are not specific for the RFFT and can be used to calculate the CFFT.

The proposed architectures are all feed-forward which can process either 2 or 4 samples in parallel, thereby achieving a higher performance than previous designs which are serial in nature. When compared to some previous architectures, the proposed design doubles the throughput and halves the

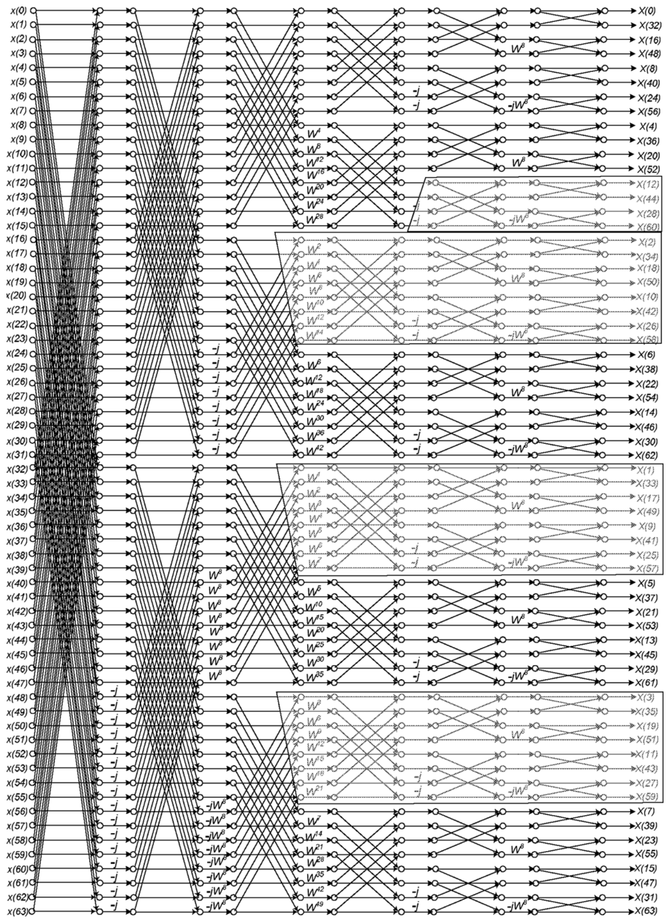


Fig. 26. Flow graph of a 64-point radix-2^3 DIF RFFT.

TABLE III
COMPARISON OF PIPELINED HARDWARE ARCHITECTURES FOR THE COMPUTATION OF N-POINT RFFT

Architecture	# Multipliers	# Adders	# Delays	Control	Throughput
R2MDC	$2(log_4 N-1)$	$4log_4 N$	$3N/2-2$	simple	1
R2SDF	$2(log_4 N-1)$	$4log_4 N$	$N-1$	simple	1
R4SDC	$(log_4 N-1)$	$3log_4 N$	$2N-2$	simple	1
R2^{2}SDF	$(log_4 N-1)$	$4log_4 N$	$N-1$	simple	1
radix-2 [27]	$2(log_4 N-1)$	$4log_4 N$	$N-1$	complex	4
Proposed Architectures					
radix-2	$2(log_4 N-1)$	$4log_4 N-2$	$9N/8-2$	simple	2
Arch 5 (radix-2^2)	$log_4 N-1$	$4log_4 N-2$	$9N/8-2$	simple	2
Arch 6 (radix-2^2)	$log_4 N-1$	$4log_4 N-2$	$N-2$	simple	2
Arch 7 (radix-2^3)*	$log_8 N-1$	$4log_4 N-2$	$N-2$	simple	2

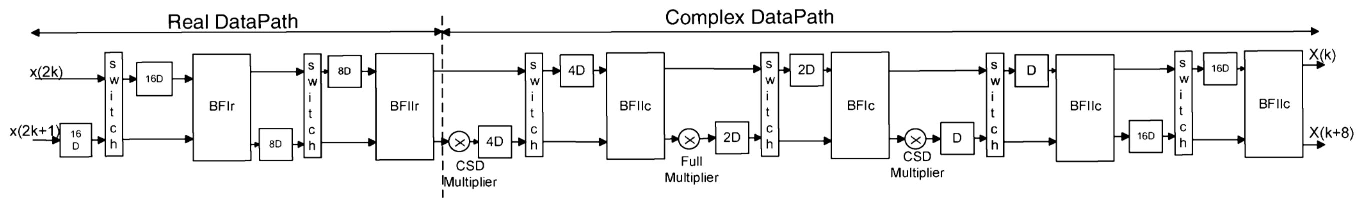


Fig. 27. Proposed 2-parallel (Architecture 7) for the computation of 64-point radix-2^3 DIF RFFT.

TABLE IV
COMPARISON OF PIPELINED ARCHITECTURES FOR THE COMPUTATION OF 128-POINT CFFT

	# Multipliers	# Adders	# Delays	Control
2-parallel [12]	2+0.41	28	190	medium
2-parallel Proposed	2+0.41	14	190	simple
4-parallel [14]	4 + 0.41	52	316	medium
4-parallel [11]	2+(4*0.62)	48	380	complex
4-parallel Proposed	4+0.41	28	254	simple

TABLE V
FGPA RESOURCE USAGE

Point size	Slice Register	Slice LUTs	Slices	Throughput
16	1499	1174	523	720Ms/s
32	1909	1544	645	720Ms/s
64	2320	1916	780	720Ms/s

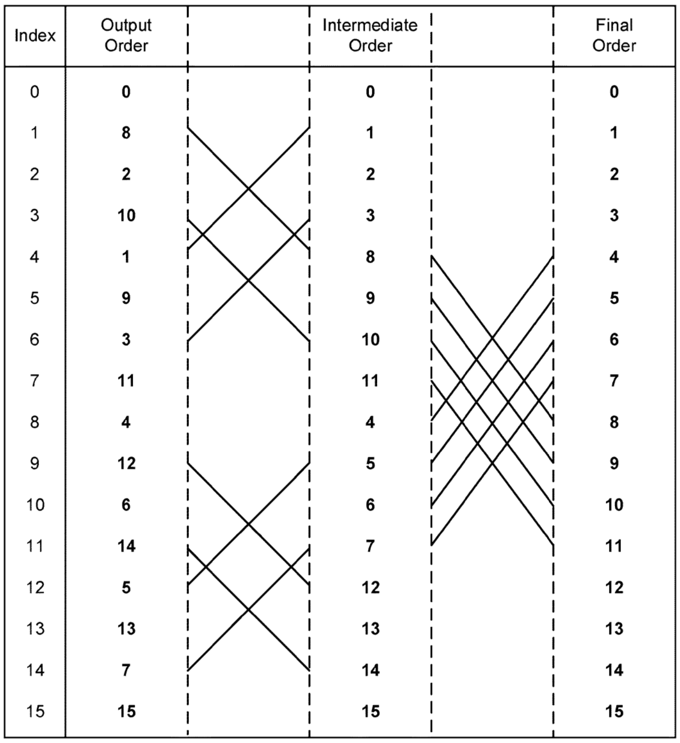
latency while maintaining the same hardware complexity. The proposed architectures maintain hardware regularity compared to previous designs.

Further we compare the hardware complexity of the proposed architectures with the specific ($N = 128$) existing designs. Table IV presents the performance comparison between the proposed parallel architectures and the existing 128-point FFT designs. For fair comparison, radix-2^4 algorithm is considered to implement the proposed 2-parallel and 4-parallel architectures.

A. Power Consumption

We compare power consumption of the serial feedback architecture with the proposed parallel feedforward architectures of same radix. The dynamic power consumption of a CMOS circuit can be estimated using the following equation:

$$P_{\text{ser}} = C_{\text{ser}} V^2 f_{\text{ser}} \tag{7}$$



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. 28. Solution to the reordering of the output samples for the architecture in Fig. 12.

where C_{ser} denotes the total capacitance of the serial circuit, V is the supply voltage, and f_{ser} is the clock frequency of the circuit. Let P_{ser} denotes the power consumption of the serial architecture.

In an L-parallel system, to maintain the same sample rate, the clock frequency must be decreased to f_{ser}/L. The power consumption in the L-parallel system can be calculated as

$$P_{\text{par}} = C_{\text{par}} V^2 \frac{f_{\text{ser}}}{L} \tag{8}$$

where C_{par} is the total capacitance of the L-parallel system.

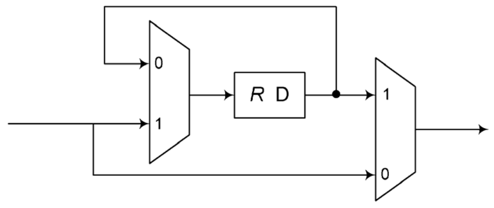


Fig. 29. Basic circuit for the shuffling the data.

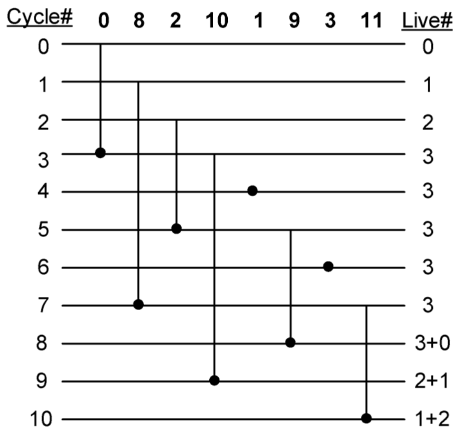


Fig. 30. Linear lifetime chart for the first stage shuffling of the data.

Complex FFT: For example, consider the proposed architecture in Fig. 12 and R2SDF architecture [6]. The hardware overhead of the proposed architecture is 50% increase in the number of delays. Assume the delays account for half of the circuit complexity in serial architecture. Then $C_{\text{par}} = 1.25C_{\text{ser}}$ which leads to

$$P_{\text{par}} = 1.25C_{\text{ser}}V^2\frac{f_{\text{ser}}}{2} = 0.625P_{\text{ser}}. \tag{9}$$

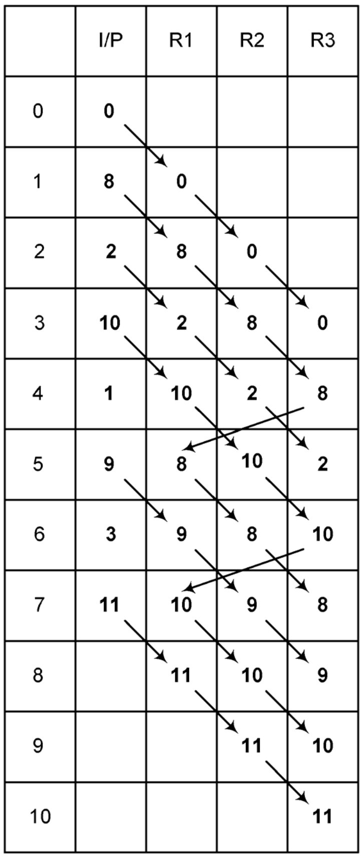
Therefore, the power consumption in a 2-parallel architecture has been reduced by 37% compared to the serial architecture.

Similarly, for the proposed 4-parallel architecture in Fig. 16, the hardware complexity doubles compared to R2SDF architecture. This leads to a 50% reduction in power compared to a serial architecture.

Real FFT: In the case of RFFT, the hardware complexity remains the same compared to its complex counterpart. Therefore, almost 50% power reduction can be achieved. To maintain the regularity, redundant samples (incorrect values) are still being computed in the proposed architectures. We can use clock gating to further reduce power consumption during the computation of redundant samples.

B. Hardware Implementation

The complex FFT for different word lengths were implemented on Xilinx Virtex-5 FPGA (XC5VLX20T) using 16-bit fixed-point implementation based on the proposed 2-parallel radix-2 FFT. Table V shows the resource usage of the FPGA device. The design achieves a throughput of 720 Msamples/s. As the design takes two inputs at a time, power consumption can be reduced by operating the circuit only at 370 MHz. Throughput is constant regardless of FFT length due to extra pipelining, and other resource usage increases linearly as the FFT length is doubled.



	I/P	R1	R2	R3
0	0			
1	8	0		
2	2	8	0	
3	10	2	8	0
4	1	10	2	8
5	9	8	10	2
6	3	9	8	10
7	11	10	9	8
8		11	10	9
9			11	10
10				11

Fig. 31. Register allocation table for the first stage shuffling of the data.

VI. Conclusion

This paper has presented a novel approach to derive the FFT architectures for a given algorithm. The proposed approach can be used to derive partly parallel architectures of any arbitrary parallelism level. Using this approach parallel architectures have been proposed for the computation of complex FFT based on radix-2^n algorithms. The power consumption can be reduced by 37% and 50% in proposed 2-parallel and 4-parallel architectures, respectively. Further these architectures are optimized for the case when inputs are real-valued. For RFFT, two different scheduling approaches have been proposed with one having less complexity in control logic while the other has fewer delay elements. Due to the capability of processing two input samples in parallel, the frequency of operation can be reduced by 2, which in turn reduces the power consumption up to 50%. These are very suitable for applications in implantable or portable devices due to their low area and power consumption. The real FFT architectures are not fully utilized. Future work will be directed towards design of FFT architectures for real-valued signals with full hardware utilization.

Appendix
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 [5] 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

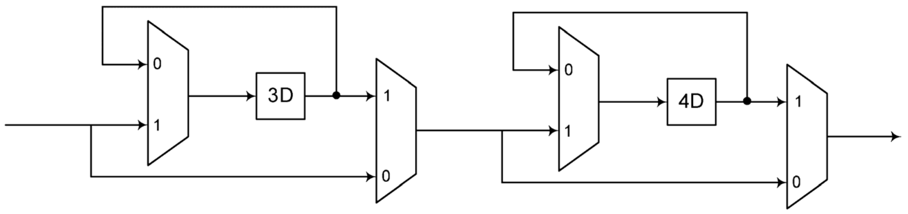


Fig. 32. Structure for reordering the output data of 16-point DIF FFT.

the counter. 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 changes for different architectures because of different folding sets/scheduling schemes. We need a general scheme for reordering these samples. Inspired by [27], 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. 12. The order of output samples is shown in Fig. 28. 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 and 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.

A shuffling circuit is required to do the de-interleaving of the output data. Fig. 29 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 first stage shuffling in Fig. 28 is shown in Fig. 30 and the register allocation table is in Fig. 31. Similar analysis can be done for the second stage too. Combining the two stages of reordering in Fig. 28, the circuit in Fig. 32 performs the shuffling of the outputs to obtain them in the natural order. It uses 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.

References

[1] J. W. Cooley and J. Tukey, "An algorithm for machine calculation of complex fourier series," *Math. Comput.*, vol. 19, pp. 297–301, Apr. 1965.

[2] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Singal Processing*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1998.

[3] 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.

[4] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proc. of IPPS*, 1996, pp. 766–770.

[5] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[6] 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.

[7] A. M. Despain, "Fourier transfom using CORDIC iterations," *IEEE Trans. Comput.*, vol. C-233, no. 10, pp. 993–1001, Oct. 1974.

[8] 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. SC-19, no. 5, pp. 702–709, Oct. 1984.

[9] E. E. Swartzlander, V. K. Jain, and H. Hikawa, "A radix-8 wafer scale FFT processor," *J. VLSI Signal Process.*, vol. 4, no. 2/3, pp. 165–176, May 1992.

[10] G. Bi and E. V. Jones, "A pipelined FFT processor for word-sequential data," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 12, pp. 1982–1985, Dec. 1989.

[11] Y. W. Lin, H. Y. Liu, and C. Y. Lee, "A 1-GS/s FFT/IFFT processor for UWB applications," *IEEE J. Solid-State Circuits*, vol. 40, no. 8, pp. 1726–1735, Aug. 2005.

[12] J. Lee, H. Lee, S. I. Cho, and S. S. Choi, "A High-Speed two parallel radix- 2^4 FFT/IFFT processor for MB-OFDM UWB systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2006, pp. 4719–4722.

[13] J. Palmer and B. Nelson, "A parallel FFT architecture for FPGAs," *Lecture Notes Comput. Sci.*, vol. 3203, pp. 948–953, 2004.

[14] M. Shin and H. Lee, "A high-speed four parallel radix- 2^4 FFT/IFFT processor for UWB applications," in *Proc. IEEE ISCAS*, 2008, pp. 960–963.

[15] M. Garrido, "Efficient hardware architectures for the computation of the FFT and other related signal processing algorithms in real time," Ph.D. dissertation, Dept. Signal, Syst., Radiocommun., Univ. Politecnica Madrid, Madrid, Spain, 2009.

[16] K. K. Parhi, C. Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE J. Solid-State Circuits*, vol. 27, no. 1, pp. 29–43, Jan. 1992.

[17] K. K. Parhi, "Systematic synthesis of DSP data format converters using lifetime analysis and forward-backward register allocation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 39, no. 7, pp. 423–440, Jul. 1992.

[18] K. K. Parhi, "Calculation of minimum number of registers in arbitrary life time chart," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 41, no. 6, pp. 434–436, Jun. 1995.

[19] D. Sinha and A. H. Tewfik, "TDCS, OFDM, and MC-CDMA: A brief tutorial," *IEEE Commun. Mag.*, vol. 43, no. 9, pp. S11–S16, Sep. 2005.

[20] J. W. Picone, "Signal modeling techniques in speech recognition," *Proc. IEEE*, vol. 81, no. 9, pp. 1215–1247, Sep. 1993.

[21] M. H. Cheng, L. C. Chen, Y. C. Hung, and C. M. Yang, "A real-time maximum likelihood heart-rate estimator for wearable textile sensors," in *Proc. IEEE 30th Annu. Int. Conf. EMBS*, 2008, pp. 254–257.

[22] T. Netoff, Y. Park, and K. K. Parhi, "Seizure prediction using cost-sensitive support vector machine," in *Annu. Int. Conf. EMBS*, 2009, pp. 3322–3325.

[23] R. Storn, "A novel radix-2 pipeline architecture for the computation of the DFT," in *Proc. IEEE ISCAS*, 1988, pp. 1899–1902.

[24] Y. Wu, "New FFT structures based on the Bruun algorithm," *IEEE Trans. Acoust., Speech Signal Process.*, vol. 38, no. 1, pp. 188–191, Jan. 1990.

[25] B. R. Sekhar and K. M. M. Prabhu, "Radix-2 decimation in frequency algorithm for the computation of he real-valued FFT," *IEEE Trans. Signal Process.*, vol. 47, no. 4, pp. 1181–1184, Apr. 1999.

[26] W. W. Smith and J. M. Smith, *Handbook of Real-Time Fast Fourier Transforms*. Piscataway, NJ: Wiley-IEEE Press, 1995.
[27] M. Garrido, K. K. Parhi, and J. Grajal, "A pipelined FFT architecture for real-valued signals," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 12, pp. 2634–2643, Dec. 2009.
[28] S. He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," in *Proc. Int. Symp. Signals, Syst.*, 1998, pp. 257–262.
[29] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. Hoboken, NJ: Wiley, 1999.
[30] C. Wang and K. K. Parhi, "High-level DSP synthesis using concurrent transformations, scheduling, and allocation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 14, no. 3, pp. 274–295, Mar 1995.
[31] K. Ito, L. E. Lucke, and K. K. Parhi, "ILP-based cost-optimal DSP synthesis with module selection and data format conversion," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 6, no. 4, pp. 582–594, Dec. 1998.
[32] H. De Man, F. Catthoor, G. Goossens, J. Vanhoof, J. Van Meerbergen, and J. Huisken, "Architecture driven synthesis techniques for VLSI implementation of DSP algorithms," *Proc. IEEE*, vol. 78, no. 2, pp. 319–335, Feb. 1990.
[33] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of data-path intensive architectures," *IEEE Design Test*, vol. 8, no. 2, pp. 40–51, Jun. 1991.
[34] T. C. Denk and K. K. Parhi, "Exhaustive scheduling and retiming of digital signal processing systems," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 45, no. 7, pp. 821–838, Jul. 1998.
[35] M. Ayinala and K. K. Parhi, "Parallel-Pipelined radix- 2^2 FFT architecture for real valued signals," in *Proc. Asilomar Conf. Signals, Syst. Comput.*, 2010, pp. 1274–1278.

Manohar Ayinala (S'10) received the Bachelor's degree in electronics and communication engineering from Indian Institute of Technology, Guwahati, India, in 2006, and the M.S. degree in electrical and computer engineering from University of Minnesota, Minneapolis, in 2010, where he is currently pursuing the Ph.D. degree.

His research interests include digital signal processing and classification algorithms, very low-power and cost efficient VLSI architectures, and implementation for biomedical applications.

Michael Brown was born in 1982. He received the B.E.E. and M.S.E.E. degrees from the University of Minnesota, Minneapolis, in 2009 and 2011, respectively.

His research interests include embedded systems design and biomedical signals analysis. He is currently with Leanics Corporation as Principal Investigator of an embedded device research project.

Keshab K. Parhi (S'85–M'88–SM'91–F'96) received the B.Tech., M.S.E.E., and Ph.D. degrees from the Indian Institute of Technology, Kharagpur, India, the University of Pennsylvania, Philadelphia, and the University of California at Berkeley, in 1982, 1984, and 1988, respectively.

He has been with the University of Minnesota, Minneapolis, since 1988, where he is currently a Distinguished McKnight University Professor with the Department of Electrical and Computer Engineering. His research interests address VLSI architecture design and implementation of physical layer aspects of broadband communications systems, error control coders and cryptography architectures, high-speed transceivers, and ultra wideband systems. He is also currently working on intelligent classification of biomedical signals and images, for applications such as seizure prediction, lung sound analysis, and diabetic retinopathy screening. He has published over 450 papers, has authored the text book VLSI Digital Signal Processing Systems (Wiley, 1999) and coedited the reference book *Digital Signal Processing for Multimedia Systems* (Marcel Dekker, 1999).

Dr. Parhi was a recipient of numerous awards including the 2004 F. E. Terman Award by the American Society of Engineering Education, the 2003 IEEE Kiyo Tomiyasu Technical Field Award, the 2001 IEEE W.R.G. Baker Prize Paper Award, and a Golden Jubilee Award from the IEEE Circuits and Systems Society in 1999. He has served on the editorial boards of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: EXPRESS BRIEFS, *VLSI Systems, Signal Processing, Signal Processing Letters,* and *Signal Processing Magazine*, and served as the Editor-in-Chief of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS (2004-2005), and currently serves on the Editorial Board of the *Journal of VLSI Signal Processing*. He has served as technical program cochair of the 1995 IEEE VLSI Signal Processing Workshop and the 1996 ASAP Conference, and as the general chair of the 2002 IEEE Workshop on Signal Processing Systems. He was a distinguished lecturer for the IEEE Circuits and Systems society during 1996-1998. He was an elected member of the Board of Governors of the IEEE Circuits and Systems Society from 2005 to 2007.