EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

Low power reconfigurable FP-FFT core with an array of folded DA butterflies

Augusta Sophy Beulet Paul[1*], Srinivasan Raju[2] and Raja Janakiraman[3]

Abstract

A variable length (32 ~ 2,048), low power, floating point fast Fourier transform (FP-FFT) processor is designed and implemented using energy-efficient butterfly elements. The butterfly elements are implemented using distributed arithmetic (DA) algorithm that eliminates the power-consuming complex multipliers. The FFT computations are scheduled in a quasi-parallel mode with an array of 16 butterflies. The nodes of the data flow graph (DFG) of the FFT are folded to these 16 butterflies for any value of N by the control unit. Register minimization is also applied after folding to decrease the number of scratch pad registers to $(log_2 N - 1) \times 16$. The real and imaginary parts of the samples are represented by 32-bit single-precision floating point notation to achieve high precision in the results. Thus, each sample is represented using 64 bits. Twiddle factor ROM size is reduced by 25% using the symmetry of the twiddle factors. Reconfigurability based on the sample size is achieved by the control unit. This distributed floating point arithmetic (DFPA)-based design of FFT processor implemented in 45-nm process occupies an area of 0.973 mm^2 and dissipates a power of 68 mW at an operating frequency of 100 MHz. When compared with FFT processor designed in the same technology with multiplier-based butterflies, this design shows 33% less area and 38% less power. The throughput for 2,048-point FFT is 222 KS/s and the energy spent per FFT is 7.4 to 14 nJ for 64 to 2,048 points being one among the most energy-efficient FFT processors.

Keywords: Fast Fourier transform (FFT); Distributed floating point arithmetic (DFPA); Twiddle factor; DIF FFT; Butterfly element; Array architecture

1 Introduction

Fast Fourier transforms (FFTs) efficiently compute the coefficients of a discrete Fourier series (DFS). Also, FFT is one of the most commonly used signal processing algorithms in any communication or multimedia system. Direct applications of FFT include spectral analysis, spectral estimation, image processing, interpolation, decimation, convolution, correlation, filtering, etc. FFT is also used in all wideband digital communication systems, which use orthogonal frequency division multiplexing (OFDM) as the modulation technique.

1.1 Need for reconfigurable FFT

In a multi-mode, multi-band, multi-functional wireless communication system like software-defined radio (SDR), OFDM is used for base band processing. FFTs of different size are required for different applications, which use OFDM. Table 1 tabulates the wired and wireless communication technologies that use OFDM as their modulation technique and their FFT size. The size of FFT varies from 64 to 2,048 in these applications and the need for a variable-length reconfigurable FFT processor is inevitable. This is the motivation for the researchers to propose as many methods and architectures for a reconfigurable FFT processor.

1.2 Need for low power FFT processor

While implementing FFT algorithm on hardware, the area, power, and speed are the major performance parameters. FFT algorithm is a computationally intensive algorithm and the large number of complex multiplications consumes a lot of power and area.

Implementing FFT and inverse fast Fourier transform (IFFT) blocks using digital signal processors (DSPs) is the method used in the initial years and it is followed even now, as reconfiguring the requirements can be done easily

* Correspondence: augustasophyt.p@vit.ac.in
[1]School of Electronics Engineering, VIT University, Kelambakkam Road, Chennai 600 127, India
Full list of author information is available at the end of the article

Table 1 Wired/wireless technologies which use OFDM

Applications	FFT points *N*
High-performance local area network (LAN)	64
Wireless LAN	64
Multiple-input and multiple-output (MIMO) OFDM system	64/128
Institute of Electrical and Electronics Engineers (IEEE) 802.16 based wireless systems	128 ~ 2,048
Digital audio broadcasting (DAB)	256 ~ 2,048
Very-high-bit-rate digital subscriber line (VDSL)	256 ~ 2,048
Asymmetric digital subscriber line (ADSL)	512
Worldwide interoperability for microwave access	2,048
Digital video broadcasting-terrestrial (DVB-T)	2,048/8,912

through software. But DSPs are power hungry and not suitable for battery-operated communications equipment. The FFT and IFFT can also be implemented on field-programmable gate array (FPGA) and other reusable IP cores, but the area and power consumption are not as low as for dedicated hard FFT cores.

Also as the technology node shrinks, with millions of switching transistors per μm^2, the total power dissipated by the high performing VLSI circuits greatly increases the temperature of devices and reduces its reliability. It needs higher efforts for cooling and increases the battery weight. In this scenario, a number of low power reconfigurable FFT processors with different architectures have been proposed in the literature and they are summarized in Section 2.

1.3 Review of FFT algorithm

Reviewing the basic discrete Fourier transform (DFT) equation of a N-point sequence $x(n)$ consisting of the samples, $\{x(0),x(1),x(2).....x(N\text{-}1)\}$, the DFT $X(k)$ is given by Equation 1.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \tag{1}$$

where the variables 'k' and 'n' vary from 0 to $N-1$. The transforming coefficient W_N^{kn}, commonly called as the 'twiddle factor', is defined as given by Equation 2.

$$W_N^{kn} = e^{-j2\pi kn/N}. \tag{2}$$

The direct computation or implementation of the DFT equation requires N^2 complex multiplications and $N(N-1)$ number of complex additions. Fast Fourier transforms (FFTs) compute the DFT efficiently with reduced number of multiplications and additions. The basic FFT algorithm was developed by Cooley and Tukey in 1965 [1]. The techniques used in developing the FFT algorithm are breaking down the DFT of a long sequence into small DFTs and exploiting the following properties of the twiddle factor. Those two properties are given in Equations 3 and 4.

- Symmetry property

$$W_N^{(k+N/2)} = -W_N^k. \tag{3}$$

- Periodicity property

$$W_N^{(k+N)} = W_N^k. \tag{4}$$

There are hundreds of different versions of the FFT algorithm. Decimation in time (DIT) and decimation in frequency (DIF) are the two methods in grouping the N samples. Radix-2 DIF FFT algorithm is applied in this work.

2 FFT processor architectures

2.1 General FFT architectures

Based on the FFT algorithm used, radix chosen, size of FFT and the number of channels, a variety of FFT architectures have been proposed in the literature [2-18]. While mapping the FFT algorithm into hardware generally, three [2] or more different architectures are followed [3].

2.1.1 Single PE architecture

A monoprocessor, i.e. a single processing element, is used to perform all the butterflies in the signal flow graph. As the single processing element is reused, usually a butterfly element of higher radix is preferred to reduce the latency. The advantage of single processing element (PE) architecture is high hardware utilization and the disadvantages are discontinuous input and output data streams [3].

2.1.2 Pipelined architecture

The pipelined architecture uses one PE for each stage and the speed of processing is increased. Thus, many concurrent processing elements are used to process different stages to achieve high throughput with less number of cycles [4,5]. Single-path delay feedback (SDF), single-path delay commutator (SDC) [6], and multi-path delay commutator (MDC) [7-9] are the common types of pipelined architectures.

2.1.3 Fully parallel FFT architecture

Parallel or column FFT processor maps the signal flow graph or a single stage of the signal flow graph, isomorphically, into a hardware structure. One stage of FFT computation is done using several processing elements and the same hardware is reused for the next stages. This architecture is hardware intensive.

2.1.4 Array architecture

Array-based architecture uses an array of processing elements to do the FFT computation. All the processing

elements can be enabled in parallel to increase the speed of operation. Thus, an area-speed trade-off is done. The scheduling logic of the processing elements makes the design complex and hence, this architecture is not commonly used.

2.2 Low power FFT architectures and techniques

Several low power FFT implementation approaches have been proposed in the literature over the past two decades, but still there is a continuing search for an ultra low power implementation of FFT. The research papers [19-21] on methods for motion estimation on a customizable reconfigurable hardware motivate the researchers to search for biologically inspired FFT architectures which might provide the best solution to design a low power FFT.

To achieve low power implementation of DSP circuits, pipelining, parallel processing, algebraic transformations, and algorithmic modifications are generally employed [10]. Reducing the physical switching capacitance either by reducing the physical capacitance or by reducing the switching activity is an appropriate solution to achieve low power. The physical capacitance can be reduced by reducing the complexity of the architecture, while the switching activity can be reduced by an appropriate data encoding method, by proper reordering of the operations, and by using point-to-point data buses [10]. The reduction in complexity and increase in throughput is depicted in [7] for MIMO OFDM system using 4-channel radix-2^3 ($R2^3$) and mixed radix architecture, as $R2^3$SDF needs the smallest number of non-trivial multiplications.

Pipelined FFT processor architecture is put into practice in [2,4,5,11]. Pipelining can be used either to increase the operating frequency or to lower the operating voltage, thus decreasing the power consumption. Radix-2, radix-4, and radix-8 butterflies are used in a pipeline to achieve the implementation of 64 to 2,048 point FFT in [11], and a high-speed radix-2^5 based processor is presented in [14]. A pipelined low power FFT/IFFT processor, along with optimized complex multiplier, is designed for up to 2,048 points for WiMax application in [15].

In [16], low power consumption is achieved by novel radix-2 and radix-4 butterfly elements, which share two complex multipliers. High throughput is also achieved in [16] using three distributed memories for loading the input data and for reading/writing data before and after computation. The low power FFT processor proposed in [5] uses radix-2^2 algorithm and power saving is achieved by using asynchronous memory instead of synchronous memory. The 64-point low power FFT processor of [4] has used radix-2, pipelined architecture. Twiddle factor ROM size is reduced by using a reconfigurable complex multiplier. Five types of twiddle factor multiplications are identified and thus the number of computations is reduced, achieving low power consumption [4]. There are many more architectures in the literature, which proposes low power design.

In [17], a 64 to 8,192 point FFT processor for low power applications is presented, by using dynamic data scaling scheme, thereby using a small word length of 11×2. To compensate for the signal-to-quantization-noise ratio (SQNR) of the reduced word length, 'trounding' (truncation and rounding) strategy is used instead of rounding/truncation. A power and area optimized reconfigurable FFT processor, employing radix factorization using the algorithmic, architectural, and also the circuit level optimization is proved to be highly energy-efficient in [18]. The possibility of achieving the most energy-efficient FFT processor architecture is investigated in all dimensions. An area and energy-efficient multimode processor proposed in [22] is also designed based on flexible-radix and multi-path delay feedback architecture and has achieved high throughput with good SQNR. Better SQNR and also 2.5 GS/s are reported in [14].

To summarize, the following methodologies are commonly employed in FFT processors to achieve low power.

- Reducing the load capacitance C or the switching frequency 'f'
- Pipelining
- Memory partitioning and reducing the twiddle ROM size
- Using higher radix, mixed radix algorithms, and radix factorization
- Using energy-efficient processing blocks

3 The proposed methodology

Three major approaches are used to achieve both low power and reconfigurability of the FFT core in our work. In a FFT core, the major portion of power consumption occurs in two blocks, namely the butterflies with complex twiddle factor multiplications and the internal data storage registers. These two issues are addressed in this design to achieve low power, and reconfigurability is also achieved with the following listed methodologies.

- Conventional butterfly with complex multipliers is replaced with distributed arithmetic-based butterfly, which reduces the dynamic power generated by the butterfly computation by 80% (at 20 MHz), thus the whole FFT computation consumes very less dynamic power.
- Reconfigurability of the processor to accommodate different lengths of FFT is made possible by the folded butterfly architecture done for an array of 16 coarse grain butterfly processing elements. This is an atypical architecture in contrast to the typical pipelined or parallel architectures.

- Using register minimization technique [23], the internal memory requirement is reduced to $(\log_2 N - 1) \times 16$, which further reduces the power.

These three features are explained in detail in the following sections.

3.1 The DAA-based butterfly design

Butterfly operation is the basic computation in the FFT algorithm. Distributed arithmetic algorithm is used for low-power finite impulse response (FIR) filter implementation without multipliers. [24] shows such an implementation of FIR filter with a combination of DA and common sub-expression elimination (CSE) and genetic algorithm on a reconfigurable hardware. Relating distributed arithmetic to a butterfly computation and constructing a FFT processor based on the bit serial butterfly with high latency are done for the first time in this project. The butterfly element is hand crafted using distributed floating point arithmetic (DFPA) for high-energy efficiency. The FFT processor design using distributed arithmetic was proposed as long back as in 1981 in the literature [25] but distributed arithmetic algorithm (DAA) is not applied to the butterfly operation, instead it is used directly for computing prime number DFTs. Combining many prime number FFTs, a larger length FFT is formed. In [25], a pipelined prime factor FFT algorithm is implemented for 504 points using shorter transforms of 8-point, 9-point, and 7-point which are implemented using DAA. But in this work, DA is applied in the butterfly level so that the design can be modularized for reusing it for variable-length, thus making reconfigurability possible.

Figure 1 shows the radix-2 DIF butterfly operation. Here x and y are the two complex inputs to the DIF butterfly and the two outputs X and Y are defined as in Equations 5 and 6. The real and imaginary parts of the inputs x and y and the outputs X and Y are represented as 32-bit single-precision floating point number in the IEEE 754 format. Thus, each sample is 64 bits wide.

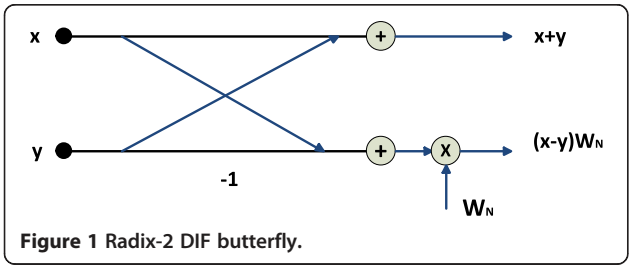

Figure 1 Radix-2 DIF butterfly.

The outputs X and Y of the butterfly in Figure 1 are given by Equations 5 and 6.

$$X = x + y, \tag{5}$$

$$Y = (x - y) W_N^k, \tag{6}$$

where x, y, X, and Y have real and imaginary parts and these complex values are represented in a 64-bit format.

Equations 5 and 6 can be expanded as given in Equations 7 and 8, respectively.

$$X_{\text{re}} + jX_{\text{img}} = (x_{\text{re}} + y_{\text{re}}) + j\left(x_{\text{img}} + y_{\text{img}}\right). \tag{7}$$

$$Y_{\text{re}} + jY_{\text{img}} = \left((x_{\text{re}} - y_{\text{re}}) + j\left(x_{\text{img}} - y_{\text{img}}\right)\right) \times \left(W_{\text{re}} + jW_{\text{img}}\right). \tag{8}$$

Thus in a conventional butterfly, to compute output X, two floating point adders/subtractors are required. To compute output Y, four floating point adders/subtractors and four floating point multipliers are required. These floating point multipliers consume more dynamic power and occupy more area.

DFPA algorithm-based butterfly (DFPABF) does not employ floating point complex multipliers. Instead, it uses two numbers of shift-accumulators and look-up tables (LUT) to generate the output Y. DA is a bit serial computation technique for finding the inner product of two vectors, when one of the vectors is known. The radix-2 butterfly, which is a two-point DFT, can be computed using DAA as the twiddle factors are known values. Design of a DAABF is described in [26] completely, by the same authors. Figure 2 shows the block diagram of a DFPABF. Table 2 shows the hardware requirement of a conventional butterfly and the DFPABF.

When implemented in 45-nm technology, the DFPA butterfly shows 80% less power and 44% less area compared with the conventional butterfly. As DAA is a bit serial operation, this design has a latency of 31 cycles to produce the output of the butterfly operation. But this latency is used efficiently to operate the butterflies in a quasi-parallel mode, in this design of the DFPABF-based FFT processor.

3.2 Mapping the DFG on the array of DA butterflies using folding transformation

Getting the insight from the FPGA architecture, 16 DFPA butterflies arranged in an array topology are used to compute FFT up to 2,048 points, instead of the pipelined architecture used in the conventional FFT processor. In pipelined architecture, one butterfly is employed for one stage of computation and the hardware utilization of a pipelined processor is not 100% except for the higher FFT points. But in this

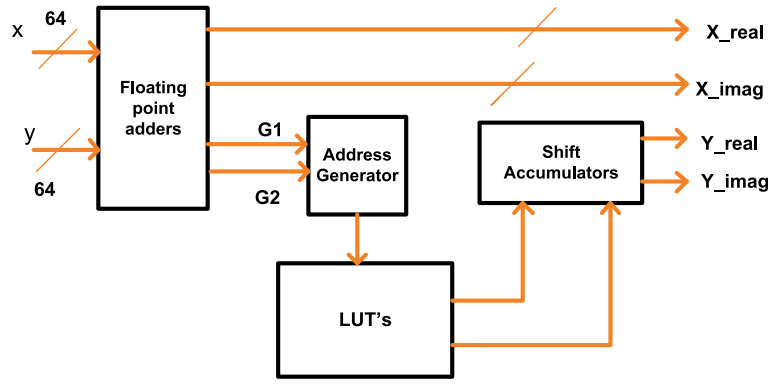

Figure 2 Block diagram of a DFPA-based butterfly.

methodology, all the butterflies are used even for small FFT size.

Pipeline FFT architecture can be derived systematically via folding transformation [27]. In this work, the folding transformation is used to arrive at array architecture and the scheduling of this array of butterflies is done. In other words, the nodes in the data flow graph (DFG) of FFT of any size can be folded on to the array of 16 butterfly elements available in the hardware. Such a folding transformation for a 32-point FFT is shown as an illustration to explain the reconfigurability of this design. DFG of the $N = 32$ point FFT is shown in Figure 3. These $(N/2)$ log2 N (=80) butterfly operations are mapped using folding transformation on to the butterfly block with 16 DFPABFs.

Note: The delays are shown only along the first set of edges and not shown for other edges for simplicity. They are present along the appropriate edges, and the pipelining delays used in the calculations are not shown.

For folding transformation, the folding set which is the set of nodes folded to a single computational unit and also the folding order should be determined. The folding set and the order of each node vary with the number of FFT points N. Null operations are not required in the folding set, as this is not a pipelined architecture. The instance at which $A0$ would fire is taken as $t = 0$, though it fires only after receiving both $x(0)$ and $x(N/2)$ which arrives after $N/2$ cycles with respect to the arrival of $x(0)$. Considering the first $N/2$ cycles as the initial latency, the orders of all 'A'-type nodes are '0'. The 80 butterfly nodes shown in the DFG for $N = 32$ are folded on to the 16 DFPABFs available in the hardware. Therefore, 16 folding sets each containing log2 N nodes are formed as shown in Equation 9.

$$\begin{aligned} S1 &= \mathrm{BF0} = \{A0, B0, C0, D0, E0\} \\ S2 &= \mathrm{BF1} = \{A1, B1, C1, D1, E1\} \\ &\vdots \\ S16 &= \mathrm{BF15} = \{A15, B15, C15, D15, E15\} \end{aligned} \quad . \tag{9}$$

Here the folding factor F is 5 for all the sets, the folding order is the time instance at which a particular node in the set fires, and the folding order varies from 0 to $F - 1$. For example, in the folding set BF0 containing five operations, the folding orders of $A0$ is 0, $B0$ is 1, $C0$ is 2, $D0$ is 3, and $E0$ is 4. The folding edges and switched inputs/outputs for set $S1$ folded to butterfly element BF0 = {$A0$,$B0$,$C0$,$D0$,$E0$} are derived as follows.

For an edge e from the node U whose l^{th} iteration is scheduled at $Fl + u$ time units to a node V whose l^{th} iteration is scheduled at $Fl + v$ time units in the original DFG with weight $w(e)$, with F as the folding factor, the new weight on the folded edge is calculated using the formula given in Equation 10. [23],

$$D_F(U \rightarrow V) = Fw(e) - P_U + v - u, \tag{10}$$

where P_U is the number of pipeline stages in the butterfly unit, which are 31 for DFPABF. The new weights of all the edges of the folded set for BF0 are calculated as

Table 2 Hardware used for conventional BF and DFPABF

Hardware	Conventional	DAABF
Floating point adder/subtractor	6	4
Floating point multiplier	4	0
Shift accumulators	0	2
4-input LUT	0	2

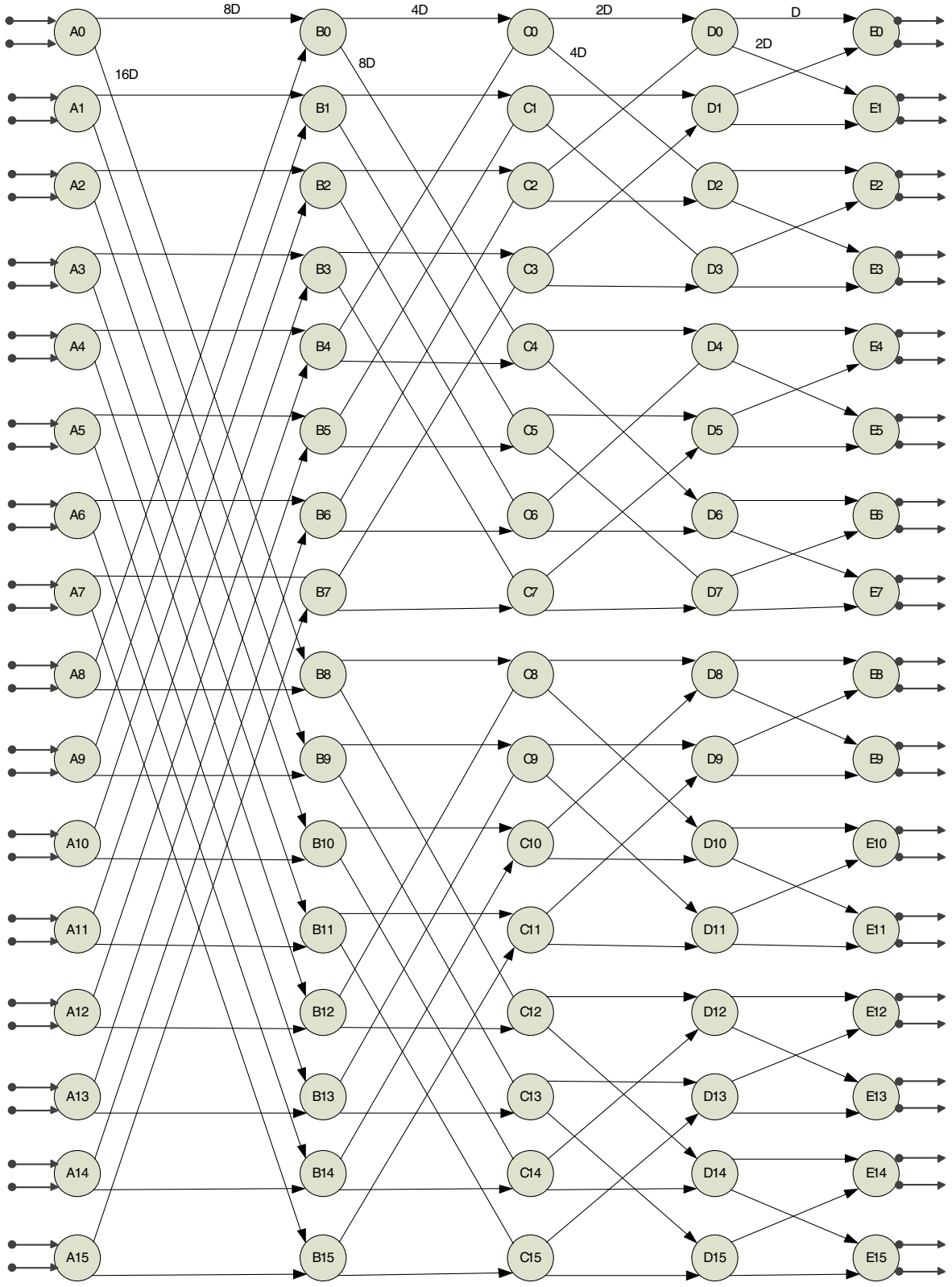

Figure 3 DFG of 32-point FFT.

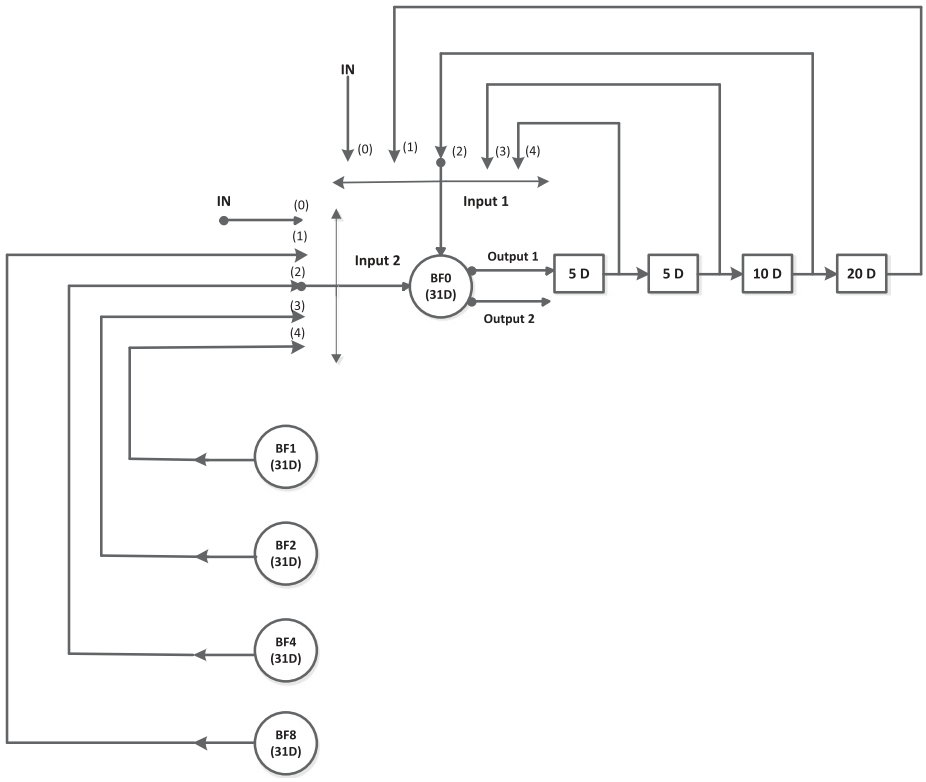

Figure 4 Folded butterfly BF0 for the nodes *A*0 to *E*0 for $N = 32$.

follows. Pipelining delays are added on the edges to get positive delays on the folded edges.

$$\begin{aligned}
D_F(A0 \rightarrow B0) &= 5 \times 14\text{–}31 + 1\text{–}0 = 40D \\
D_F(A8 \rightarrow B0) &= 5 \times 6\text{–}31 + 1\text{–}0 = 0D \\
D_F(B0 \rightarrow C0) &= 5 \times 10\text{–}31 + 2\text{–}1 = 20D \\
D_F(B4 \rightarrow C0) &= 5 \times 6\text{–}31 + 2\text{–}1 = 0D \\
D_F(C0 \rightarrow D0) &= 5 \times 8\text{–}31 + 3\text{–}2 = 10D \\
D_F(C2 \rightarrow D0) &= 5 \times 6\text{–}31 + 3\text{–}2 = 0D \\
D_F(D0 \rightarrow E0) &= 5 \times 7\text{–}31 + 4\text{–}3 = 5D \\
D_F(D1 \rightarrow E0) &= 5 \times 6\text{–}31 + 4\text{–}3 = 0D
\end{aligned} \tag{11}$$

Pipelining registers are added along the feed forward cut set, so that all the delays of the folded DFG are positive to make the block realizable. The delays added as pipelining delays along the edges are not shown in DFG diagram. Using the folding equations given in Equation 11, the set of nodes {*A*0,*B*0,*C*0,*D*0,*E*0,} is folded to one computational element BF0 as shown in Figure 4. The DAA-based BF takes 31 cycles by itself for a complete butterfly operation including the twiddle factor multiplication. This is considered as internal pipelining delay of the node.

Similar folding switches are added for 'output 2' terminal of the butterfly too. All the sets are folded to form BF1, BF2, etc. Thus, DFG for $N = 32$ with 80 nodes is folded/rolled over in the horizontal direction on to the 16 BFs, in contrast to vertical folding as done in the pipelined architecture. The same technique can be used for a FFT and the corresponding DFG of any size.

3.3 Register minimization

The number of internal registers required for storing the outputs of the nodes is determined systematically using the register minimization technique explained in [23]. The lifetime analysis is done for the five nodes in BF0 and the output variables produced by them. For each node, $T_{\text{input}} \rightarrow T_{\text{output}}$ is calculated. T_{input} is the time at

Table 3 Life time of nodes in the folding set BF0

Node	$T_{\text{input}} \rightarrow T_{\text{output1}}$	$T_{\text{input}} \rightarrow T_{\text{output2}}$
*A*0	31 → 71	31 → 71
*B*0	32 → 52	32 → 32
*C*0	33 → 43	33 → 33
*D*0	34 → 39	34 → 34
*E*0	35 → 35	35 → 35

Cycle	Node	A0	B0	C0	D0	E0	A0	B0	C0	D0	E0	# Live
0		Out1	Out1	Out1	Out1	Out1	Out2	Out2	Out2	Out2	Out2	
1												0
:												0
31							X					0
32								X				1
33									X			2
34										X		3
35						X					X	4
36												4
:												4
39												4
:												3
43												3
:												2
52												2
:												1
71												1
72												0

Figure 5 Life time chart.

which the node produces data and $T_{\text{input}} = u + P_U$, where u is the folding order and P_U is the pipelined delay of the node. $T_{\text{output}} = u + P_U + \max\ v\ D_F\ (U \rightarrow V)$, where $\max\ v\ D_F(U \rightarrow V)$ is the longest folded path delay for all the paths from node U. Here for every butterfly node there are two outputs, so the edges along both the outputs should be considered. The life time table for the set BF0 is given in Table 3.

The life time chart is given in Figure 5 and it can be seen that the number of registers required is only four, maximum number of live variables at a time instance. With 16 butterflies available in the hardware, each would require four registers and the total register array requirement is 64 for $N = 32$. When N increases by an order 2, the additional requirement is only 16 registers. The number of registers R required is given by $R = (\log 2\ N - 1) \times 16$. Thus for $N = 2{,}048$, we need only (10×16) 160 internal storage registers. Thus, the registers required are reduced drastically. It is mentioned here that the size of each register is 64 bits as 32-bit floating point numbers are used for both real and imaginary parts of the data.

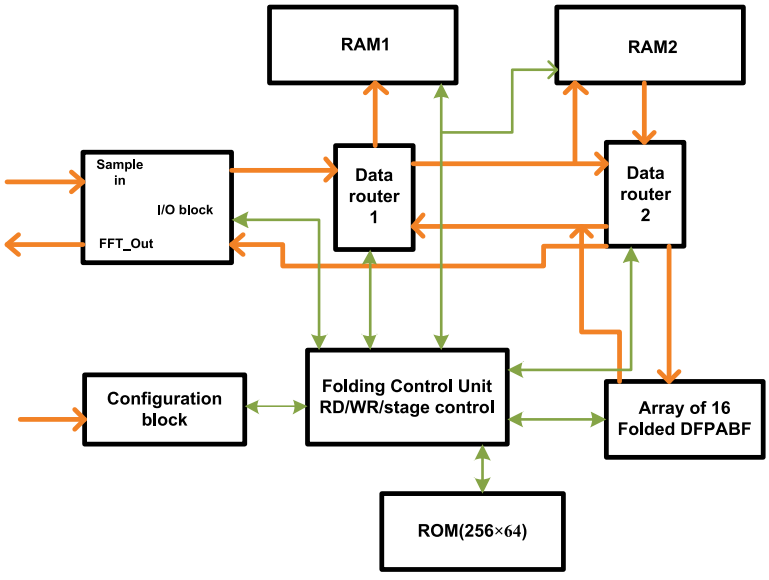

Figure 6 Architecture of proposed processor.

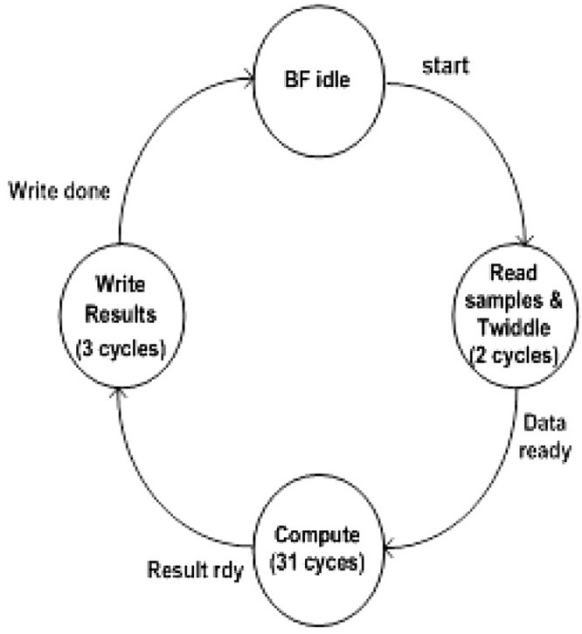

Figure 7 Compute FSM.

4 The adapted architecture

The FFT processor proposed in this work is reconfigurable for processing up to 2,048 input samples using an array of 16 low power DFPA butterflies on to which all the nodes in the DFG are folded. The delays along the folded edges differ with respect to the FFT size N and are configured by the stage control unit. Two register banks of size 64 words each (64×64) are fabricated in the FFT processor as a basic internal RAM, and they are alternatively used for storing the incoming data and as an internal register array. An additional register array of 32 registers is set aside to attain the maximum register size of 160 required for 2 K-point FFT. The butterflies are fired one after the other once in three clock cycles with its inputs, which process the FFT computation and are controlled by the control unit. This array and memory-based floating point FFT processor architecture as given in Figure 6 is presented in this section.

4.1 The IO block and butterfly block

The IO (input/output) block is the interface with the outside world. It receives the input samples in 64-bit format and writes the incoming samples to the RAM. The FFT output (64 bits), which is available in one of the RAMs after all the stages of processing, is transmitted out by the IO module.

The butterfly block consists of 16 DFPA-based butterfly elements. Each folded butterfly receives a set of two data from the read control block. The addresses of these data are also generated by the folding control block which is programmed with the address-generating algorithm for the different N values. Only one of the butterflies gets the inputs at a time and it ends the process after 31 cycles. The compute finite-state machine (FSM) is shown in Figure 7. In the meantime, the other butterflies receive the data sequentially once in every two clock cycles. Thus, there is an added latency of 1 cycle. Thus, outputs come sequentially once in every two clock cycles after the initial latency of 31 cycles from the butterfly block and get stored in the register array for the next stage of processing by one of the folded butterflies. The scheduling of the 16 butterflies is shown in Figure 8 and how the butterfly resources are allocated for the computation using the minor butterfly cycles is shown in Figure 9, for $N = 64$.

4.2 Reduced size twiddle ROM

A reduced twiddle factor ROM of size 256 × 64 bits (2 KB) is used in this processor. For an N-point FFT, there are $N/2$ distinct twiddle factors but there is inherent symmetry among the twiddle factors. The twiddle factor entry to the ROM can be further

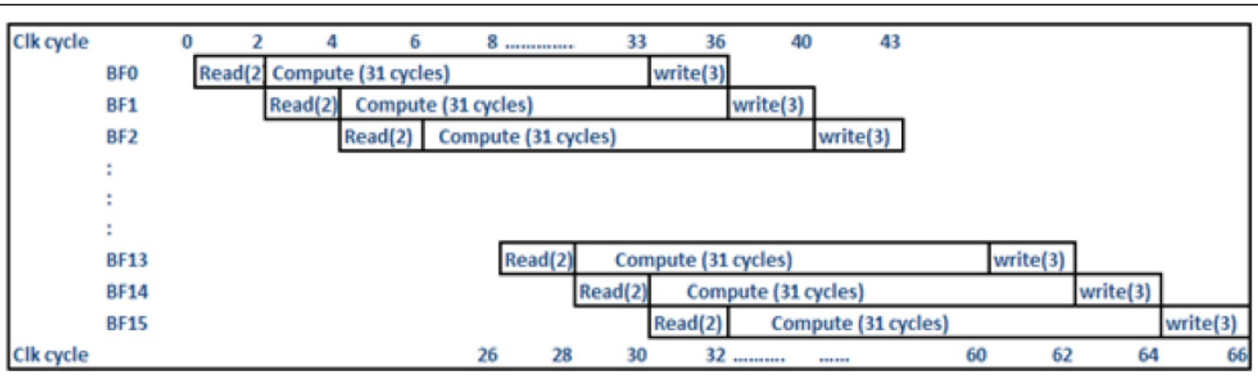

Figure 8 Scheduling of the 16 butterflies.

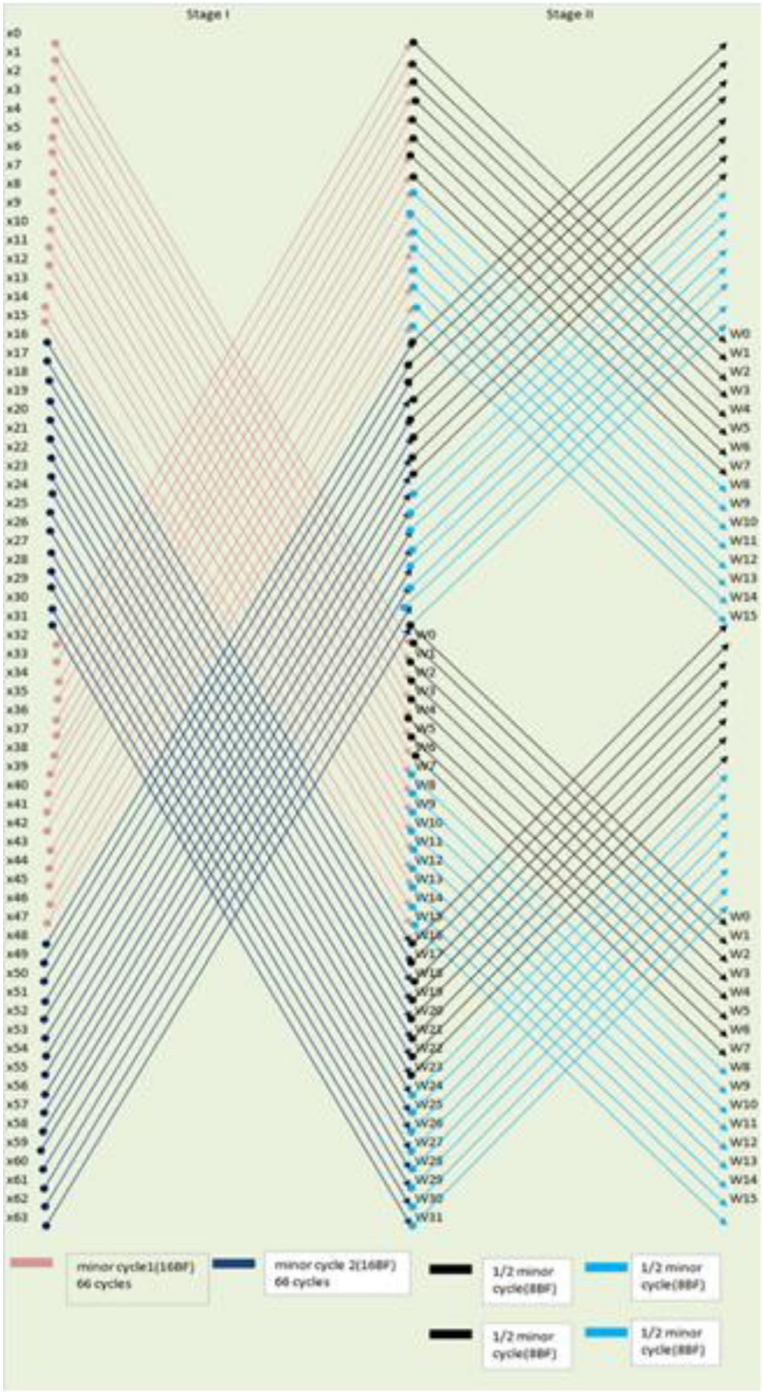

Figure 9 BF block in minor cycles-stages 1 and 2 computation for $N = 64$.

reduced with additional logic to either $N/4$ or $N/8$ using t $\pi/2$ symmetry or $\pi/4$ symmetry of the sine and cosine values [28]. In this design, the additional glue logic calculates the twiddle from the $N/8$ values. Thus for a 2,048 point FFT, the 1,024 distinct twiddle factors are obtained only with 256 values.

4.3 Configuration registers and control

FFT size N is given as the input to the configuration block. Aiding to the reconfiguration, the configuration registers configure the control unit for the required number of stages, number of butterflies per stage, number of times the BF block is used, etc. The stage

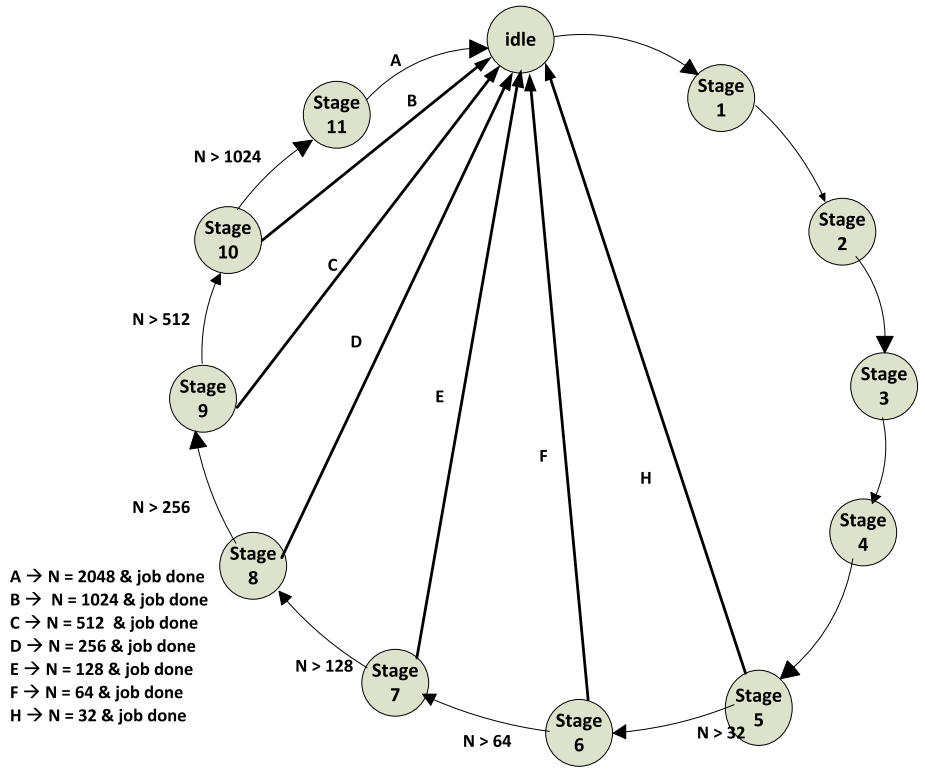

Figure 10 Stage control FSM.

control FSM shown in Figure 8 controls the whole computation process and reconfiguration process. On receiving the information from the configuration registers and other blocks, it controls the flow of data from the RAM and from the IO block. It also controls the data flow in the BF block and controls all the different stages of the FFT computation. The address generation for accessing the data and the twiddle factors from the RAM and ROM respectively are done by the read, write, and twiddle blocks but monitored and

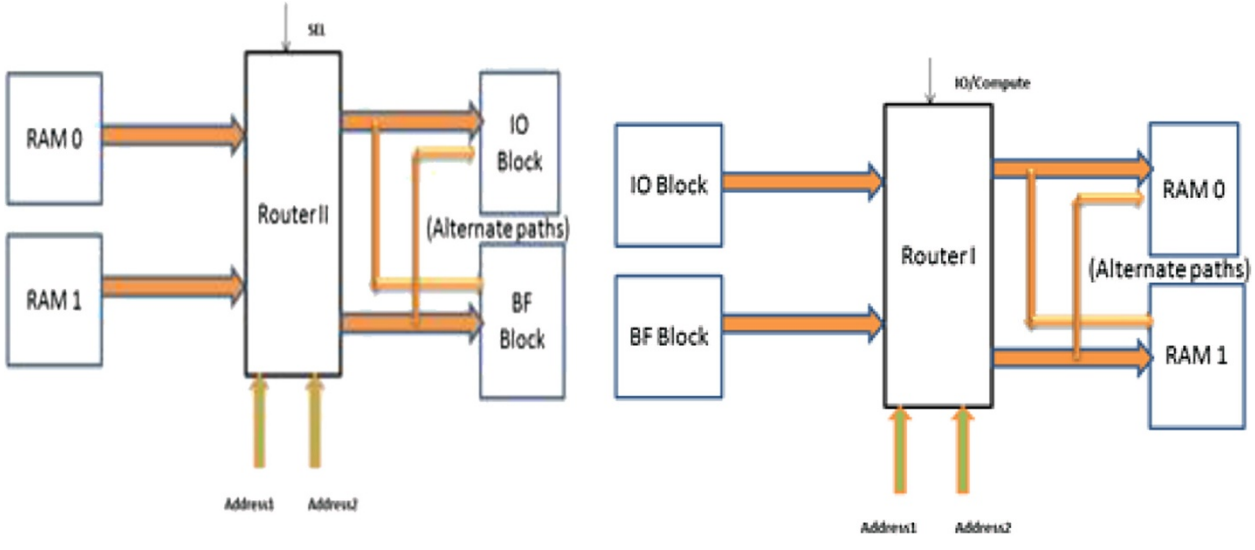

Figure 11 Data flow through routers I and II.

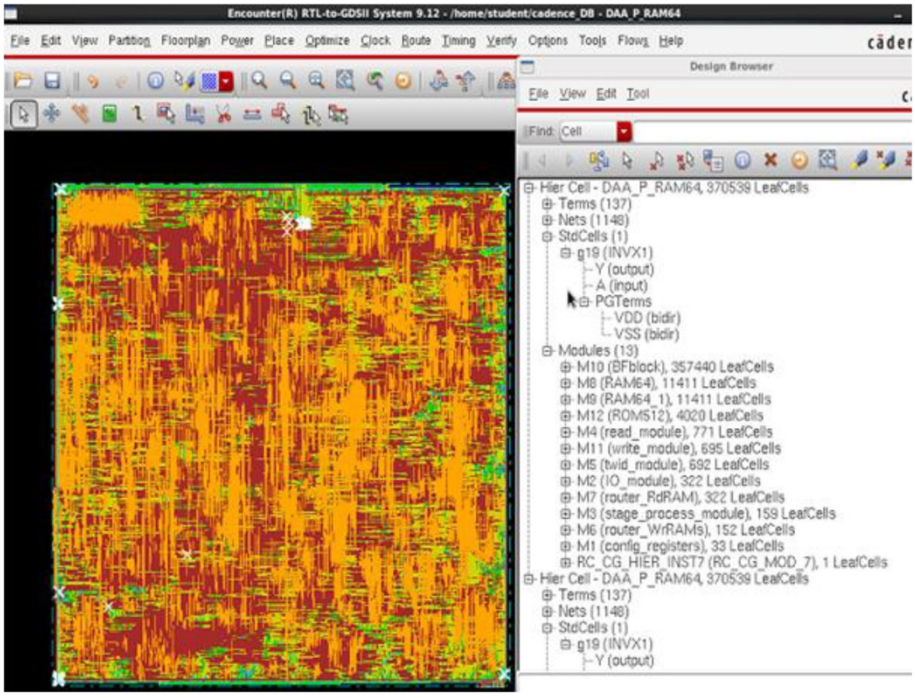

Figure 12 Layout of the distributed arithmetic-based processor.

controlled by the signals generated by the stage control block.

Once one of the RAMs is filled with the samples to be processed, the stage control FSM initiates the read module to read the pair of samples from RAM. The samples to be fed to a particular butterfly are read one after the other as a pair in three clock cycles. Then, one of the 16 butterflies is enabled and it starts processing. As the butterfly operation is based on DA algorithm, which is a bit serial operation, it takes 31 cycles to produce the output. In the meantime, the next butterfly receives the data samples and starts processing. For writing the two outputs to RAM, again three clock cycles are required (Figure 10).

Thus, the first butterfly finishes the whole process in $2 + 31 + 3 = 36$ clock cycles. When all the 16 butterflies are enabled, the first butterfly has finished the process and ready to process the next set of data. The scheduling of the 16 butterflies is shown in Figure 6. As shown in the Figure 6, BF0 produces its output at the 36th clock cycle and after that, for every two cycles, one set of outputs is produced and stored in the register array. Thus, there is an initial latency of 36 clock cycles, to get the first output of the first stage of FFT computation. One cycle of computation of all the 16 butterflies is called one minor cycle. One minor cycle gets completed in 66 clock cycles.

For N =32, one stage of computation is done in one minor cycle, and five minor cycles finish the computation. The inputs and outputs of all the 16 butterflies are fed back and forwarded from the register array using the folded architecture/switches. The data flow between the folded butterfly nodes is controlled by the stage control block. If $N = 64$, two minor cycles

Table 4 Chip implementation details

Technology	45-nm CMOS
Voltage	1.08 V
Process	1P6M
PVT conditions	Typical
Word length	64 bits
FFT size	32 to 2,048
Internal RAM	1.25 KB
ROM	2 KB
Maximum frequency	100 MHz
Core area	0.973 mm^2
Cell count	307,201
Leakage power	0.034 mW
Total power	68.17 mW
Energy per FFT	14 nJ for 2,048 points

Table 5 Comparison of conventional and DAA-based designs at 20 MHz

Parameter	DFPABF	Conventional BF	Percentage saving	Proposed FFT processor	Conventional BF-based processor	Percentage saving
Maximum frequency (MHz)	100	20	–	100	20	–
No. of cells	18,074	46,886	61.45	245,452	571,590	57.06
Area (mm^2)	0.031	0.055	43.64	0.694	1.04	33.27
Leakage power (nw)	1,318	2,711	51.38	26,574	48,679	45.41
Total power at 20 MHz (mW)	0.9878	4.937	79.99	28.9	46.85	38.31

are required to finish one stage of FFT computation. Then, the folding architecture is different. For N = 1,024, 32 minor cycles complete one stage. When one stage is completed, the next stage is carried out with another set of minor cycles. The design works with a clock frequency of 100 MHz with a clock period of 10 ns. Thus, final FFT output of 64-point FFT will be available after six stages, each stage consisting of two minor cycles of the BF block. Thus, for $N = 64$, the final FFT output will be available after an initial latency of $2 \times 66 \times 6 \times 10$ ns plus some stages over delays adding to it becomes 7.64 μs.

4.4 Data routers

There are two routers which routes the data to and from the two RAMs. Data router I receives data from the IO module and also the outputs from the BF block and sends them to RAM0 or RAM1 based on the control signals. Similarly, router II receives data from RAM0 and RAM1. It routes them to the BF block for processing and at the FFT output to the IO block. All the data are 64-bit wide as 32 bits are used for real part and 32 bits are used for imaginary part. The diagrams of the routers are given in Figure 11.

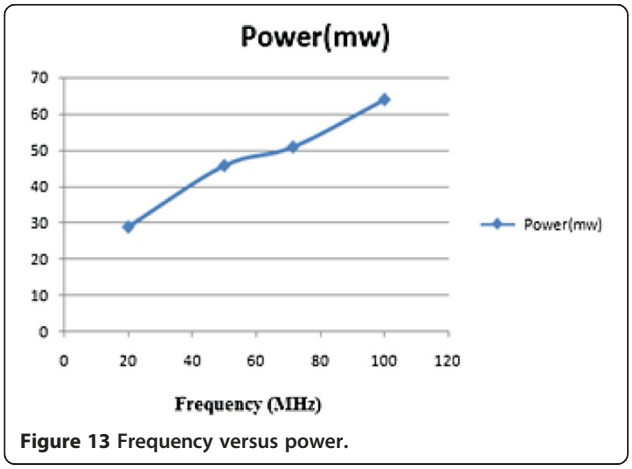

Figure 13 Frequency versus power.

5 Chip implementation and results

The proposed DFPABF-based reconfigurable processor core is implemented in Verilog hardware description language, synthesized using Cadence RTL compiler (Cadence Design Systems, San Jose, CA) using standard 45-nm technology library, with a V_{dd} supply of 1.08 V, for normal PVT conditions. The back end physical design up to layout of the chip is done using Cadence Encounter (Cadence Design Systems, San Jose, CA) for a six metal layer and one poly process. The layout is shown in Figure 12. This design runs with a maximum clock frequency of 100 MHz. A reconfigurable 64 to 2,048-point FFT processor using conventional multiplier-based butterflies with the same array architecture is also implemented in Verilog and implemented using the same technology, in order to compare and demonstrate the higher performance of the distributed arithmetic-based FFT processor.

5.1 Reduced area and power reports

The proposed distributed arithmetic-based FFT processor results in reduced area as well as power, as the computations are distributed over many clock cycles with less hardware. The latency created due to this bit serial distributed operations is exploited in the architecture of the processor, making this design area and power efficient. This reconfigurable FFT core is a coarse grain type, whose basic building blocks are the power- and area-efficient, radix-2 DIF butterflies.

Table 6 Latency as a function of *N*

FFT Size (*N*)	Latency	Throughput (KS/s)
64	7 μs	119.375
128	15.56 μs	139.375
256	38.36 μs	159.843
512	87.36 μs	180.625
1,024	0.196 ms	201.601
2,048	0.435 ms	222.625

Table 7 Comparison of design features and performance of various FFT processors

	This work	[Kai-Jiun]	[Chia]	[Song]	[Chu yu]	[Manish]
Technology	45 nm	90 nm	65 nm	180 nm	180 nm	180 nm
Voltage (V_{dd})	1.08 V	1 V	0.45 V	1.8 V	1.8 V	1.8 V
Architecture/algorithm	Array-based, DFPABF/radix-2	MDC, 4-stream, Radix-4/8	Mixed radix MDF	Flexible radix, MDF multiple stream	Pipelined, SDF	R2SDF
FFT size/modes	Variable 64 to 2,048	Variable 128 to 2,048	Variable 128 to 2,048	128/256/512/1024	Fixed 64	Variable 128 to 2,048
				1-4 streams		
Maximum frequency	100 MHz	40 MHz	20 MHz	300 MHz	20 MHz	40 MHz
Word length	64 bits	16 bits (input)	24	20	16	32
Memory	3.25 KB internal memory (RAM + ROM)	Dual port SRAM (10,224 × 16 bits)	48 KB of register file	Mixed SRAM	DL buffers	FIFO of varying sizes
Core area	0.973 mm^2	3.1 mm^2	1.375 mm^2	3.2 mm^2	0.88 mm^2	4.52 mm^2
Power consumption	68 mW	63.72 mW	4.05	507 mW at 512 points	9.79 mW	55.64 mW
Normalized area	0.475	1.51	0.858	1.25	3.45	0.275
Normalized power	0.332 μw	3.62 μw	1.51 μw	3.8 μw	11 μw	0.489 μw

The chip implementation detail of the proposed FFT core is given in Table 4. The proposed FFT processor performance is basically evaluated by comparing with performance of the FFT processor designed with the same configuration and architecture. The synthesis results of the DAA-based butterfly/conventional butterfly and the results of the DFPABF-based reconfigurable (64 to 2 K points) FFT processor and the conventional butterfly-based (64 to 2 K points) FFT processor with the same architecture are compared in Table 5 which shows 33% less area and 38% less power for the same architecture.

The power consumption of the proposed processor at various operating frequencies is observed by synthesizing the design at different frequencies. The processor consumes less power at lower frequencies and the frequency versus power graph is shown in Figure 13.

5.2 Latency and throughput of the design

The FFT output points are generated with an initial latency which depends on the FFT size. Each minor cycle takes 66 clock cycles, and change over delays are encountered at the end of the stages. The latency in getting the first output for different values of N is shown in Table 6. After the initial latency, output data is generated at the rate of one FFT point per 10 ns.

6 Comparison with prior low power FFT processors

Performance of the FFT processor, designed in this work, is compared against other existing processors with low power consumption. As the implemented technologies, frequencies, word sizes, FFT lengths, and their latencies different for these processors, they are ordered based on the normalized area and power. In [29], the concept of using normalized area/power for comparison of designs implemented in different technologies is first introduced. There are many variations of the formulae to calculate the normalized area and power with respect to the factors like FFT size. Operating frequency are found in the literature [4,8,18,22], etc. In this work, the word size of the complex data is 64 bits (as IEEE 754 standard single-precision floating point representation is used), whereas no other designs have used a long word size. The highest data width found for the complex data is 32, whereas most designs have used a data width of 16/20/22 bits. Thus, it is absolutely necessary to include the word size factor while normalizing the values with respect to this design. The formulae used for normalized area and power with respect to this implementation are given in Equations 12 and 13. Energy per FFT is calculated using the formula given in Equation 14. Normalized energy is not found as the execution times of other processors are not known.

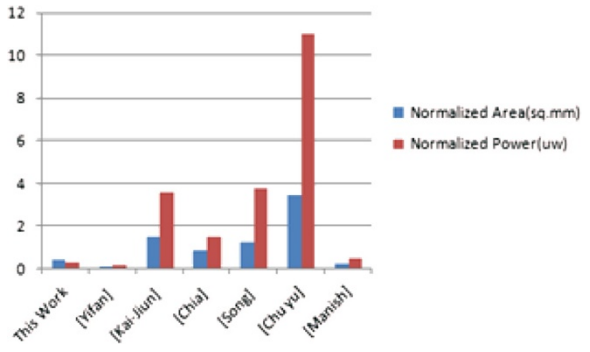

Figure 14 Power and area of various processors.

$$\frac{\text{Normalized area}}{\text{FFT}} = \frac{\text{Area} * 1000}{(\text{FFT Size}) * \left(\frac{L_{\text{mim}}}{45}\right)^2 * \left(\frac{\text{Wordlength}}{64^*}\right)}. \tag{12}$$

$$\frac{\text{Normalized power}}{\text{FFT}} = \frac{\text{Power*Clock period*1000}}{\left(V/(1.08)^{\$}\right)^2 * (\text{FFT size}) * \left(\frac{\text{Wordlength}}{64}\right)}. \tag{13}$$

Note: *Data width used in this design is 64
$^{\$}$Supply voltage in this work is 1.08 V

$$\frac{\text{Energy}}{\text{FFT}} = \frac{\text{Power} * \text{Execution time}}{\text{FFT size}}. \tag{14}$$

Table 7 shows the comparison of the FFT processor proposed in this work with six other processors on various parameters.

From the table, it can be observed that the FFT processor proposed in this work has less normalized area and power compared with five of the processors in the table as illustrated in Figure 14. All the processors have adapted a pipelined architecture with the variations like SDF or MDF with multiple streams and mixed radix algorithms. Only our work has used novel array architecture with 16 BF processing elements, each being fired one after the other, thus making them work in parallel with the required time delay. Thus, this

Table 8 Energy per FFT

FFT size (*N*)	Execution time (μs)	Energy per FFT (nJ)
64	7.64	7.43
128	17.84	8.79
256	40.92	10.18
512	92.48	11.60
1,024	206.44	13.01
2,048	456.08	14.44

architecture becomes suitable for the serial operation of the distributed arithmetic butterfly. The inherent advantage of the distributed arithmetic makes our processor both area and power efficient compared with most of the existing designs.

The throughput of the proposed processor is in the range of 119 to 222 KS/s. As the execution times of all the processors are not known, the normalized energy per FFT could not be calculated. The energy per FFT of this processor is calculated and it is proved with good results of 7.4 to 14.4 nJ for 64-point and 2,014-point FFT computation as shown in Table 8. Thus, it is more energy efficient than many other existing processors. This is achieved by the energy-efficient butterflies, register minimization, and the efficient scheduling of butterflies with folding transformation.

7 Conclusions

In this paper, we have presented an array architecture with folding transformation for a reconfigurable (32/64/128/256/512/1,024/2,048 points) FFT processor. The systematic folding transformation is illustrated for $N = 32$ and this approach is used for other FFT sizes also. The computational nodes are ultra low power and low-area distributed arithmetic-based FP butterflies, which accomplishes low power, less silicon processor, compared with existing low power FFT processors. The array of 16 folded butterfly elements works in a quasi-parallel mode. The number of butterflies is selected as 16 after analyzing different implementation factors and the control mechanism. Another new feature of this processor is it uses very low power butterfly elements whose design is based on DAA. The processor designed in this work occupies a silicon area of 0.973 mm^2 with a power dissipation of 68 mW at 100-MHz operating frequency. The throughput is also calculated to be in the higher range of 119 to 222 KS/s, where as one sample is 64 bits. The energy efficiency is also very high ranging from 7.4 to 14.4 nJ/FFT for the FFT size varying from 64 to 2,048. Thus, this design is one of the most energy-efficient processors designed so far.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

The authors express their heartfelt thanks to Mr. P. Radhakrishnan, Open Silicon Pvt Ltd, for his valuable inputs and constant guidance in completing this project. The authors also extend their gratitude to Dr. Anand Samuel, Dr. Menaka, Dr. Ravi Shankar, Prof. Reena, and Mr. Avinash of VIT University, Chennai, India for their valuable suggestions in the manuscript preparation and constant moral support in completing this work.

Author details

[1]School of Electronics Engineering, VIT University, Kelambakkam Road, Chennai 600 127, India. [2]SSN College of Engineering, OMR Road, Kalavakkam, Chennai 603 110, India. [3]Sri Sai Ram Engineering College, West Tambaram, Chennai 600 044, India.

Received: 7 May 2014 Accepted: 4 September 2014
Published: 17 September 2014

References

1. JW Cooley, JW Tukey, An algorithm for the machine calculation of complex Fourier series. Math Comp. **19**, 297–301 (1965)
2. L Weidong, W Lars, *A Pipeline FFT Processor, Signal Processing Systems*. SIPS, IEEE Workshop, 1999, pp. 654–662
3. S Cheng Han, L Kun Bin, J Chein Wei, *Design and Implementation of a Scalable Fast Fourier Transform Core. Proceedings of 2002 ASIA-Pacific Conference on ASIC*, 2002, pp. 295–298
4. Y Chu, Y Mao Hsu, H Pao Ann, C Sao Jie, A low power 64-point pipeline FFT/IFFT processor for OFDM applications. IEEE Trans. Consum. Electron **57**(1), 40–45 (2011)
5. W Gin Der, L Yi Ming, *Radix 2^2 Based Low Power Reconfigurable FFT Processor. Proc. Of IEEE International Symposium on Industrial Electronics (ISIE 2009)*, 2009, pp. 1134–1138
6. LIU Xue, YU Feng, NG Z k Wang, A pipelined architecture for normal I/O order FFT. Journal Zhejiang University-Science C (Computers & Electronics) **12**(1), 76–82 (2011)
7. K Byungcheol, K Jaeseok, *Low complexity multi-point 4-channel FFT Processor for IEEE 802.11n MIMO-OFDM WLAN system. International Conference on Green and Ubiquitous Technology (GUT)*, 2012, pp. 94–97. 7–8
8. K-J Yang, S-H Tsai, MDC FFT/IFFT processor with variable length for MIMO OFDM systems. IEEE Trans. Very Large Scale Integration (VLSI) Syst. **21**(4), 1188–1203 (2013)
9. M Garrido, J Grajal, MA Sanchez, O Gustafsson, Pipelined radix-2^k feed forward FFT architectures. IEEE Trans. VLSI **21**, 23–32 (2013)
10. T Arslan, DH Erdogan, AT Horrocks, Low power design for DSP methodologies and techniques. Microelectron J. **27**, 731–744 (1996). Elsevier Science Ltd
11. G Liu, Q Feng, *ASIC Design of Low Power Reconfigurable FFT Processor. ASIC,2007.ASICON '07.7th Internal Conference, IEEE*, 2007
12. W Shuenn Shyang, LI Chien Sung, An area-efficient design of variable-length fast Fourier transform processor. J. VLSI Signal Processing Systems **51**, 245–256 (2008). Springer Science
13. L Weidong, W Lars, *Low Power FFT Processors*. Swedish system-on-chip conference, SSoCC'01, Arild, Sweden, 2001, pp. 20–21
14. C Taesang, L Hanho, A high-speed low-complexity modified radix-2^5 FFT processor for high rate WPAN applications. IEEE Trans. Very Large Scale Integration (VLSI) Syst. **21**(1), 187–191 (2013)
15. S Manish, TD Patil, *Chhatbar and A.D.Darji, n area efficient and Low Power implementation of 2048 point, FFT/IFFT processor for mobile WiMax*. Proceedings of Signal Processing and Communication Conference (SPCOM), 2010, pp. 1–4
16. L Xiaojin, L ZongSheng, A low power and small area FFT processor for OFDM demodulator. IEEE Trans. Consum. Electron **53**(2), 274–277 (2007)
17. B Yifan, D Renfeng, H Jun, Z Xiaoyang, *A Hardware- efficient Variable-length FFT Processor for Low Power Applications. Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. Asia-Pacific (4, 4, 2013), pp. 1–4
18. Y Chia Hsiang, H Tsung, D Yu, Marković, Power and area minimization of reconfigurable FFT processors: A 3GPP-LTE Example. IEEE J. Solid State Circuits **47**, 3 (2012)
19. B Guillermo, B Uwe Meyer, G Antonio, R Manuel, Quantization analysis and enhancement of a VLSI gradient-based motion estimation architecture. Digital Signal Process **22**(6), 1174–1187 (2012). ISSN 1051–2004
20. G Botella, A Garcia, M Rodriguez-Alvarez, E Ros, U Meyer-Baese, MC Molina, Robust bioinspired architecture for optical-flow computation very large scale integration (VLSI) systems. IEEE Trans. **18**(4), 616–629 (2010)
21. G Botella, HJA Martín, M Santos, U Meyer, Baese, FPGA-based multimodal embedded sensor system integrating low- and mid-level vision. Sensors **11**, 8164–8179 (2011)
22. T Song, Nein, L Chi Hsang, C Tsin Yuan, An area efficient, multimode FFT processor for WPAN/WLAN/WMAN systems. IEEE Trans. Solid States **47**, 6 (2012)
23. KK Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation* (Wiley, India, 2007). Pvt. Limited
24. U Meyer Baese, G Botella, DE Romero, M Kumm, Optimization of high speed pipelining in FPGA-based FIR filter design using genetic algorithm, in *SPIE Defense, Security, and Sensing (pp 84010R-84010R). International Society for Optics and Photonics*, 2012

25. P Chow, ZG Vranesic, JL Yen, A pipelined distributed arithmetic PFFT processor. IEEE Trans. Comput. **C-32**(12), 1128–1136 (1983)
26. S Augusta, R Srinivasan, J Raja, Distributed arithmetic based butterfly element for FFT processor, in 45 nm technology. ARPN J. Eng. Appl. Sci. **8**, 1 (2013)
27. M Ayinila, M Brown, KK Parhi, Pipelined parallel FFT architectures via folding transformation. IEEE Trans. VLSI Syst. **20**, 6 (2012)
28. HJ Kang, JY Lee, JH Kim, Low-complexity twiddle factor generation for FFT processor. Electron Lett. **49**(23), 1443–1445 (2013)
29. M Bevan, Bass, A low power high performance 1024 point FFT processor. IEEE J. Solid State Circ. **34**(3), 380–387 (1999)

doi:10.1186/1687-6180-2014-144
Cite this article as: Beulet Paul *et al.*: **Low power reconfigurable FP-FFT core with an array of folded DA butterflies.** *EURASIP Journal on Advances in Signal Processing* 2014 **2014**:144.