Microelectronics Journal 47 (2016) 19–30

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

A pipelined area-efficient and high-speed reconfigurable processor for floating-point FFT/IFFT and DCT/IDCT computations

Mingyu Wang [a,*], Fang Wang [b], Shaojun Wei [a], Zhaolin Li [b]

[a] *Institute of Microelectronics, Tsinghua University, Beijing 100084, China*
[b] *Research Institute of Information Technology, Tsinghua University, Beijing 100084, China*

A R T I C L E   I N F O

Article history:
Received 21 September 2014
Received in revised form
23 June 2015
Accepted 13 November 2015

Keywords:
Reconfigurable processor
Floating-point
Fast Fourier transform (FFT)
Discrete cosine transform (DCT)
Area-efficient
High-speed

A B S T R A C T

For scientific computing and high-resolution imaging applications, this paper presents a pipelined reconfigurable processor to implement variable-length single-precision floating-point FFT/IFFT and DCT/IDCT computations compatible with the IEEE 754 standard. In order to minimize the total hardware overhead and power consumption, a reconfigurable radix-4 butterfly (RR4BF) is proposed to reduce 75% adders in comparison to the conventional parallel radix-4 butterfly, and the partially shared Ping-Pong structured register bank (PSPPRB) provides an efficient and specific intermediate data caching mechanism to realize the maximized adder resource utilization ratio in RR4BF and to guarantee the high throughput for the pipelined design. Moreover, fused floating-point 4-input adder and fused floating-point 2-term dot product unit are proposed, which can not only improve about 3 dB signal-to-quantization-noise ratio (SQNR), but also save 28% and 19% hardware overhead compared with discrete implementations and previous state-of-the-art design, respectively. Simulation results show that the latency for FFT computations is about 25% of the R4SDF design without any throughput loss, and over 139 dB SQNR is achieved. Logic synthesis results in a 65 nm CMOS technology show that the power consumption ranges from 43.5 mW to 372.3 mW for 16- to 1024-point FFTs at 500 MHz, and the total hardware overhead is equivalent to 543k NAND2 gates.

1. Introduction

Floating-point fast Fourier transform (FFT) and discrete cosine transform (DCT) computations have attractive advantages on wide dynamic range, high processing precision and complete overflow/underflow concerns in the fields of scientific computing and high-resolution imaging applications. For example, floating-point FFT kernels were employed to accelerate the applications for scientific computing [1]. Also, floating-point FFT/IFFT modules were implemented for pulse compression computations in ultra-high performance radar systems [2], and floating-point DCT was required for image compressing in high dynamic range sensors [3].

Most of the existing floating-point FFT/IFFT or DCT/IDCT computations were implemented by digital signal processors (DSPs) and field programmable gate arrays (FPGAs) [4,5], but they had significant drawbacks on power consumption and processing speed compared with application specific integrated circuit (ASIC) designs [6,7]. However, ASICs lack algorithm flexibility. Hence, coarse-grained reconfigurable processors are increasingly demanded to achieve a tradeoff between these advantages of ASICs and DSPs/FPGAs. Varieties of reconfigurable designs have been presented in [8–12]. For instance, a multi-delay feedback (MDF) based reconfigurable processor was proposed to implement 128- to 2048-point FFT computations [8], and a ring-structured reconfigurable architecture was proposed to implement 8- to 4096-point FFT/IFFT computations [12]. Unfortunately, all of them were implemented in fixed-point based methods. Although the fixed-point based designs have been successfully applied to process the baseband data in wireless communication systems, such as 3GPP-LTE and IEEE 802.16e [8,9], the SQNR and dynamic range of those fixed-point designs cannot satisfy the ever-increasing requirements of scientific computing and high-resolution imaging applications. For example, floating-point processor was required to process the data with large dynamic range for digital signal processing and scientific computing [13].

Compared with floating-point designs, the SQNR of fixed-point FFTs/DCTs is limited by the dynamic range because overflow occurs easily with the increasing of the input magnitudes [14–17]. Therefore, in order to achieve both the wide dynamic range and high SQNR for scientific computing and high-resolution imaging applications, floating-point FFT/DCT processors are more attractive than fixed-point designs. However, few literatures mentioned

* Corresponding author.
E-mail address: wang-my12@mails.tsinghua.edu.cn (M. Wang).

http://dx.doi.org/10.1016/j.mejo.2015.11.004
0026-2692/© 2015 Elsevier Ltd. All rights reserved.

reconfigurable processors to implement complete floating-point FFT/IFFT or DCT/IDCT computations. The authors in [18–20] only mentioned floating-point butterflies. Although floating-point units suffer from higher hardware overhead and power consumption than fixed-point implementations, reasonable total hardware overhead and power consumption can be achieved by optimizing the required storage and computational resources.

In this paper, a pipelined area-efficient and high-speed reconfigurable processor is presented for variable-length single-precision floating-point FFT/IFFT and DCT/IDCT computations compatible with the IEEE 754 standard [21]. By modifying the data flow of conventional parallel radix-4 butterflies, a novel reconfigurable radix-4 butterfly (RR4BF) is proposed to save 75% adders compared with the parallel radix-4 butterflies that are employed in traditional R4SDF designs [22], and the partially shared Ping-Pong structured register bank (PSPPRB) provides an efficient and specific intermediate data caching mechanism to realize the maximized adder resource utilization ratio in RR4BF and to guarantee the high throughput for the pipelined design. Moreover, fused floating-point 4-input adder and fused floating-point 2-term dot product unit are also proposed, which can not only improve about 3 dB SQNR, but also save 28% and 19% hardware overhead compared with discrete implementations and previous state-of-the-art design [18], respectively. Experimental results demonstrate the area-efficiency, the high processing speed and the high SQNR of the proposed processor.

The rest of this paper is organized as follows. Section 2 reviews the radix-4 FFT algorithm and presents a derivation of DCT computations based on FFT. Section 3 introduces the proposed reconfigurable processor. Section 4 describes the fused 4-input adder and the fused 2-term dot product unit. Section 5 gives the results and comparisons on SQNR, hardware utilization and performance. Finally, Section 6 concludes this work.

2. Preliminaries

2.1. Radix-4 FFT algorithm

The N-point discrete Fourier transform (DFT) is defined as

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

where W_N^{kn} is the twiddle factor and equals to $e^{-j(2\pi kn/N)}$. Let

$$n=4n_1+n_0,\ \begin{array}{l}n_1=0,1,2,\dots,N/4-1\\ n_0=0,1,2,3\end{array}$$
$$k=\frac{N}{4}k_1+k_0,\ \begin{array}{l}k_1=0,1,2,3\\ k_0=0,1,2,\dots,N/4-1\end{array} \tag{2}$$

Then, the N-point DFT can be decomposed as follows [11]:

$$\begin{aligned}X[k]=X\left[\frac{N}{4}k_1+k_0\right]&=\sum_{n_0=0}^{3}\sum_{n_1=0}^{N/4-1}x(4n_1+n_0)W_N^{(4n_1+n_0)((N/4)k_1+k_0)}\\&=\sum_{n_0=0}^{3}\sum_{n_1=0}^{N/4-1}x(4n_1+n_0)W_N^{(Nn_1k_1+4n_1k_0+(N/4)n_0k_1+n_0k_0)}\\&=\underbrace{\sum_{n_0=0}^{3}\left\{\left[\underbrace{\sum_{n_1=0}^{N/4-1}x(4n_1+n_0)W_{N/4}^{n_1k_0}}_{N/4-point\ DFT}\right]\cdot\underbrace{W_N^{n_0k_0}}_{twiddle\ factor}\right\}W_4^{n_0k_1}}_{radix-4\ butterfly}\end{aligned} \tag{3}$$

If we define a new $N/4$-point DFT as

$$G_n(k)=W_N^{nk}\left[\sum_{n_1=0}^{N/4-1}x(4n_1+n)W_{N/4}^{nk}\right],\ \begin{array}{l}n=0,1,2,3\\ k=0,1,2,\dots,N/4-1\end{array} \tag{4}$$

Then, the N-point DFT in Eq. (1) can be rewritten as follows:

$$X[k]=X\left[\frac{N}{4}k_1+k_0\right]=\sum_{n_0=0}^{3}\left\{W_4^{n_0k_1}\cdot G_{n_0}(k_0)\right\} \tag{5}$$

Substituting n_0, k_1 into Eq. (5), the radix-4 butterfly computation can be expressed as Eq. (6), where multiplying $\pm j$ are implemented by real-imaginary swappings easily. The data flow graph of the radix-4 butterfly is given in Fig. 1. One of the four output data corresponding to the left side in Eq. (6) is named as an output branch of the butterfly.

$$\begin{bmatrix}X[k_0]\\X[k_0+N/4]\\X[k_0+N/2]\\X[k_0+3N/4]\end{bmatrix}=\begin{bmatrix}1&1&1&1\\1&-j&-1&j\\1&-1&1&-1\\1&j&-1&-j\end{bmatrix}\cdot\begin{bmatrix}G_0(k_0)\\G_1(k_0)\\G_2(k_0)\\G_3(k_0)\end{bmatrix} \tag{6}$$

Therefore, the N-point DFT has been decomposed into four $N/4$-point DFTs by using the radix-4 butterfly computations. In order to compute the remaining $N/4$-point DFTs, each of them can be further decomposed into four $(N/4)/4$-point DFTs in a similar way as Eqs. (3)– (6). Particularly, if N is a power of 4, such as 4^m, the N-point DFT can be decomposed into m stages and each stage contains $N/4$ radix-4 butterfly computations. As an example, the data flow graph of a 16-point radix-4 FFT is shown in Fig. 2. The corresponding equations are given in Eq. (7). This decomposition thus corresponds to a decimation-in-frequency (DIF) FFT computation [23], which has been widely implemented by some basic FFT architectures, such as R4SDF [22], R4SDC [24] and R4MDC [8]. The detailed comparisons with the prior architectures and the advantages of the proposed

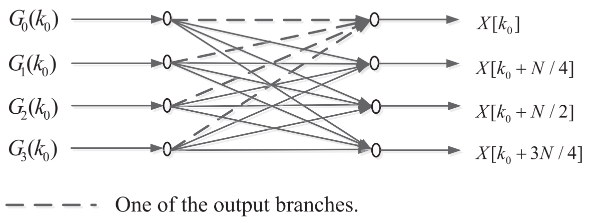

Fig. 1. Data flow graph of the radix-4 butterfly.

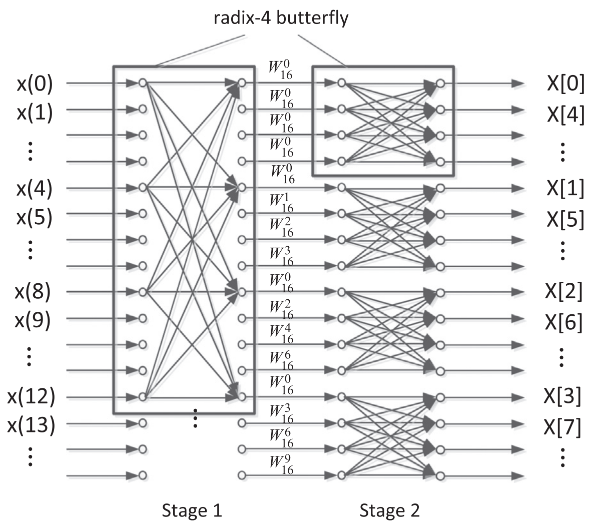

Fig. 2. Data flow graph of the 16-point radix-4 FFT.

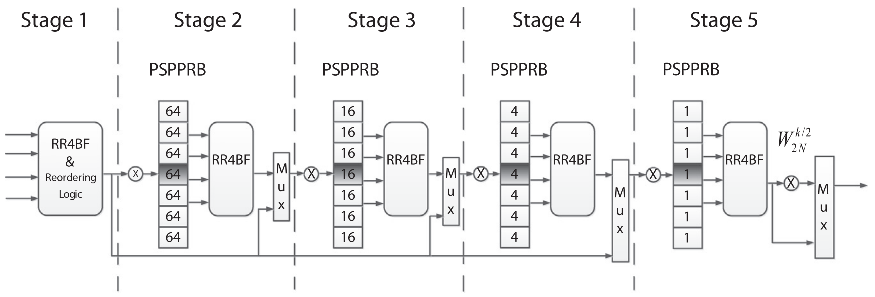

Fig. 3. Block diagram of the proposed pipelined reconfigurable processor.

processor will be presented in Section 5.

$$X[k] = X[4k_1 + k_0] = \sum_{n_0=0}^{3}\sum_{n_1=0}^{3} x(4n_1+n_0) W_{16}^{(4n_1+n_0)(4k_1+k_0)}$$

$$= \underbrace{\sum_{n_0=0}^{3}\left\{\left[\underbrace{\sum_{n_1=0}^{3} x(4n_1+n_0)W_4^{n_1k_0}}_{stage\ 1}\right]\cdot \underbrace{W_{16}^{n_0k_0}}_{twiddle\ factor}\right\}W_4^{n_0k_1}}_{stage\ 2} \tag{7}$$

2.2. DCT computations using FFTs

The definition of N-point DCT-II is given as

$$X[k] = \alpha_k \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} x(n) \cos\frac{(2n+1)k\pi}{2N}, \quad k = 0, 1, \ldots, N-1$$

$$\alpha_k = \begin{cases} 1/\sqrt{2}, & k=0 \\ 1, & k = 1, 2, \ldots, N-1 \end{cases} \tag{8}$$

In order to realize DCT computations with simple modifications based on the existing FFT algorithm, a $2N$-point sequence $y(n)$ is defined as follows [25]:

$$y(n) = \begin{cases} x(n), & n = 0, 1, \ldots, N-1 \\ x(2N-n-1), & n = N, N+1, \ldots, 2N-1 \end{cases} \tag{9}$$

And the $2N$-point DFT of $y(n)$ is expressed as

$$\begin{aligned} Y(k) &= \sum_{n=0}^{2N-1} y(n) W_{2N}^{kn}, 0 \le k \le 2N-1 = \sum_{n=0}^{N-1} x(n) W_{2N}^{kn} + \sum_{n=N}^{2N-1} x(2N \\ &\quad -n-1) W_{2N}^{kn} = \sum_{n=0}^{N-1} x(n) W_{2N}^{kn} + \sum_{n=0}^{N-1} x(n) W_{2N}^{k(2N-n-1)} \\ &= \sum_{n=0}^{N-1} x(n) \cdot [W_{2N}^{kn} + W_{2N}^{-k(n+1)}] \end{aligned} \tag{10}$$

Let $Y[k]$ be multiplied by another twiddle factor $W_{2N}^{k/2}$.

$$W_{2N}^{k/2} \cdot Y[k] = \sum_{n=0}^{N-1} x(n) \cdot [W_{2N}^{k(n+1/2)} + W_{2N}^{-k(n+1/2)}] = 2 \cdot \sum_{n=0}^{N-1} x(n) \cdot \cos\frac{(2n+1)k\pi}{2N} \tag{11}$$

Then, combining Eqs. (8) and (11), an $N/2$-point complex DCT can be easily realized by reordering the input data of an N-point FFT and attaching a complex multiplier after the FFT arithmetic unit. Since the difference between DCT and IDCT is the scaling constant α_k and IFFT is computed by conjugating the twiddle factors of FFT, only the detailed implementations of variable-length FFTs and DCTs are discussed in this paper.

Table 1
Configurations for variable-length FFTs and DCTs.

FFT/IFFT	DCT/ IDCT	Enabled stages					Reordering	The last multiplier
		1	2	3	4	5		
	8-point	√	–	–	–	√	√	√
16-point		√	–	–	–	√	–	–
	32-point	√	–	–	√	√	√	√
64-point		√	–	–	√	√	–	–
	128-point	√	–	√	√	√	√	√
256-point		√	–	√	√	√	–	–
	512-point	√	√	√	√	√	√	√
1024-point		√	√	√	√	√	–	–

[a] Symbol √ indicates that the corresponding unit is enabled.
[b] Symbol – indicates that the corresponding unit is bypassed and clock gated.

3. Design of the reconfigurable processor

3.1. Overview

Based on the decompositions in Section 2, in order to realize 16- to 1024-point FFT/IFFT and 8- to 512-point DCT/IDCT computations, a total of five cascaded stages are required in the proposed processor shown in Fig. 3. Each stage consists of two key components. One of the key components is RR4BF, which performs radix-4 butterfly computations serially by mapping the modified butterfly data flow. Another key component is PSPPRB, which stores the intermediate data in a partially shared Ping-Pong mechanism and feeds the input data to RR4BF continuously to guarantee the high throughput. Besides, the inter-stage multiplexers are used to realize variable-length FFT or DCT computations. By correctly configuring these multiplexers and employing an efficient clock gating strategy for small-size FFT or DCT computations, the reconfigurability and power-scalability of the proposed processor are achieved easily. The detailed configuration information and the clock gating strategy are shown in Table 1. The inter-stage multipliers are designed to complete the twiddle factor multiplications, which are implemented by fused 2-term dot product units, and all the required floating-point complex twiddle factors are stored in the coefficient ROM as a lookup table to achieve high processing speed and computational accuracy. In particular, the multiplier of the last stage is designed to realize the attached twiddle factor multiplications for DCT/IDCT computations. The external data RAM reading address generating logic are embedded in the first stage, which provides a 4-way parallel input data path to reduce the latency and realizes the reading address reordering for DCT/IDCT computations efficiently.

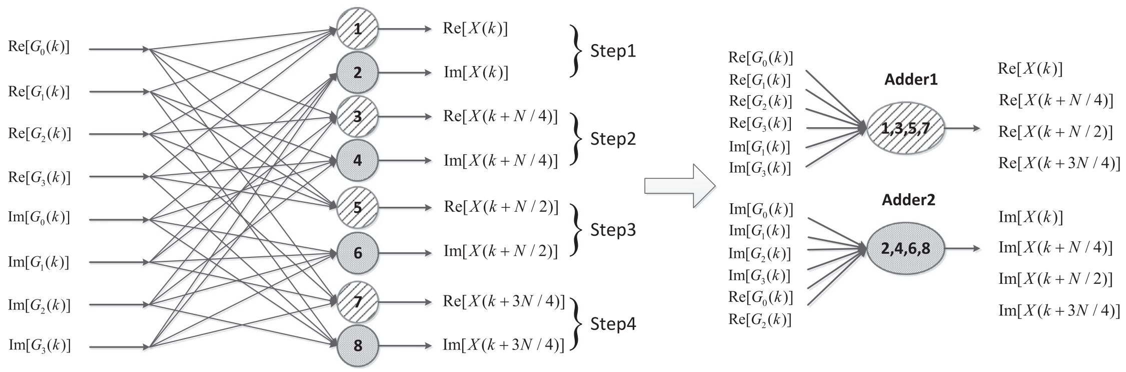

Fig. 4. Addition operator scheduling of the radix-4 butterfly.
Re[·] represents the real part of the complex data.
Im[·] represents the imaginary part of the complex data.

3.2. Reconfigurable radix-4 butterfly

As floating-point implementations, the hardware overhead and power consumption of the computational resources become serious. In order to address this problem, we optimize the computational resources by reusing the adders in a single butterfly unit as much as possible in this paper. Based on the observations that the resource utilization ratio of the adders in R4SDF-based processors is only 25%, we modify the conventional parallel radix-4 butterfly to be performed in four serial steps. Each step of the modified butterfly corresponds to one of the four output branches of the parallel radix-4 butterfly as shown in Fig. 1. To explain the reasons of saving adders, Fig. 4 gives an addition operator scheduling scheme, which is obtained by merging the addition operators that are not performed in the same step. Addition operators ①～⑧ represents 8 real adders with four input ports used in the conventional parallel radix-4 butterfly. By using the proposed operator scheduling scheme, 8 adders in the conventional design are compressed to 2 adders and 75% adders are saved. To further present the mechanism of adders reusing, Fig. 5 shows the detailed hardware implementation of RR4BF. Four 1-bit opcodes (Op00,Op01,Op10 and Op11) are used to control the input multiplexers to implement real-imaginary swappings while multiplying the imaginary factor $\pm j$ are performed for the computations of the second or fourth branches according to Eq. (6), and two 3-bit opcodes (Op0 and Op1) are used to make the addition–subtraction selections for Adder1 and Adder2 which are designed to implement the computation of $(A \pm B) \pm (C \pm D)$.

Fig. 6 shows the modified data flow graph of a 16-point FFT based on the modified radix-4 butterfly, which is taken as an example to illustrate the process of completing radix-4 FFT computations by using RR4BF. First, during the period of step 1 in stage 1, RR4BF performs the computations for the first branches of all the four different butterflies with the input data sets from $x(0), x(4), x(8), x(12)$ to $x(3), x(7), x(11), x(15)$ serially. Second, during the period of step 2 in stage 1, RR4BF performs the computations for the second branches of all the four different butterflies with the same input data sets from $x(0), x(4), x(8), x(12)$ to $x(3), x(7), x(11), x(15)$ serially. Similarly, during the period of step 3 and step 4, the computations for the remaining two branches of these four butterflies are completed with the same input data sets as the prior two steps. During the period of the same steps, the configurations of RR4BF are kept unchanged because the arithmetic expressions for the same branches are identical according to Eq. (6). Otherwise, the configurations of RR4BF will be changed by setting the opcodes if it performs the computations for different branches. Consequently, RR4BF computes the four branches serially to achieve a 100% adder resource utilization ratio, and produces the

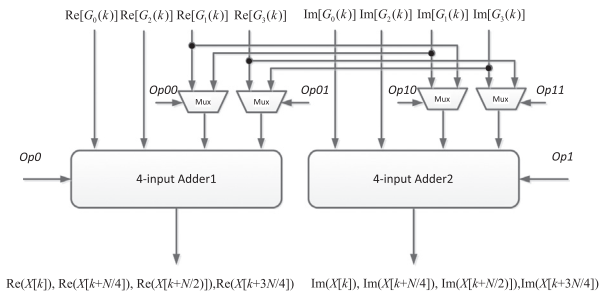
Re[$G_0(k)$] Re[$G_2(k)$] Re[$G_1(k)$] Re[$G_3(k)$]
Im[$G_0(k)$] Im[$G_2(k)$] Im[$G_1(k)$] Im[$G_3(k)$]
Op00
Op01
Op10
Op11
Mux
Op0
Op1
4-input Adder1
4-input Adder2
Re($X[k]$), Re($X[k+N/4]$), Re($X[k+N/2]$)),Re($X[k+3N/4]$)
Im($X[k]$), Im($X[k+N/4]$), Im($X[k+N/2]$)),Im($X[k+3N/4]$)

Fig. 5. Hardware implementation of RR4BF.

16 outputs consecutively over 16 cycles in contrast to the parallel radix-4 butterfly in R4SDF-based designs which must produce the 16 outputs within 4 cycles leaving 12 cycles unused. Thus, no any throughput loss is introduced in the proposed RR4BF while the number of adders is reduced greatly.

3.3. Intermediate data caching mechanism

As described above, in order to realize the maximized adder resource utilization ratio in RR4BF and to guarantee the high throughput for the pipelined design, the timing diagram of the input data for RR4BF should be accordingly modified compared with the conventional parallel radix-4 butterfly. Thus, an efficient and specific intermediate data caching mechanism is urgently demanded in the proposed processor. Fig. 7 shows the detailed structure of PSPPRB to achieve this goal, which is used to store the input data in a partially shared Ping-Pong mechanism and to feed the input data for RR4BF continuously. PSPPRB consists of seven FIFO-based register banks (RBs) and each RB can be configured to two types (normal and feedback). The normal type RB works as a general FIFO (cascaded D flip-flops), and the feedback type RB means that the output data of the last flip-flop are exactly loop-backed to the input data of the first flip-flop in this RB, denoted by the dashed lines in Fig. 7. In order to correctly store the output data of the previous stage and feed the input data to RR4BF of the current stage for continuous radix-4 butterfly computations, PSPPRB is controlled by a finite state machine (FSM) with six states (IDLE, PREPARE, CALC0, CALC1, CALC2, CALC3) as shown in Fig. 8, in which state transitions are controlled by an internal clock counter. The state IDLE means that the system is idle after system reset or all the computations are finished. The state PREPARE is entered once the input data is valid. To explain the details of the data caching mechanism, the computation process for stage 1 of two continuous 16-point DFTs, which is the first stage decomposition of a 64-point radix-4 FFT, is taken as an example here.

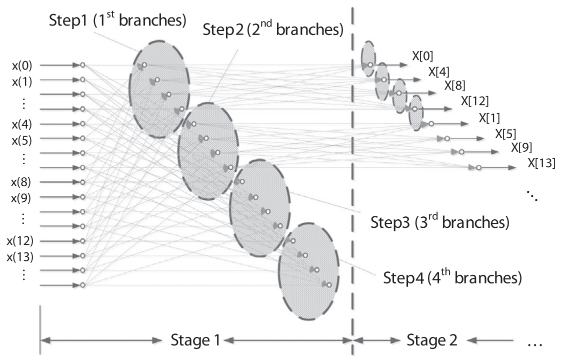

Fig. 6. Modified data flow graph of the 16-point radix-4 FFT.

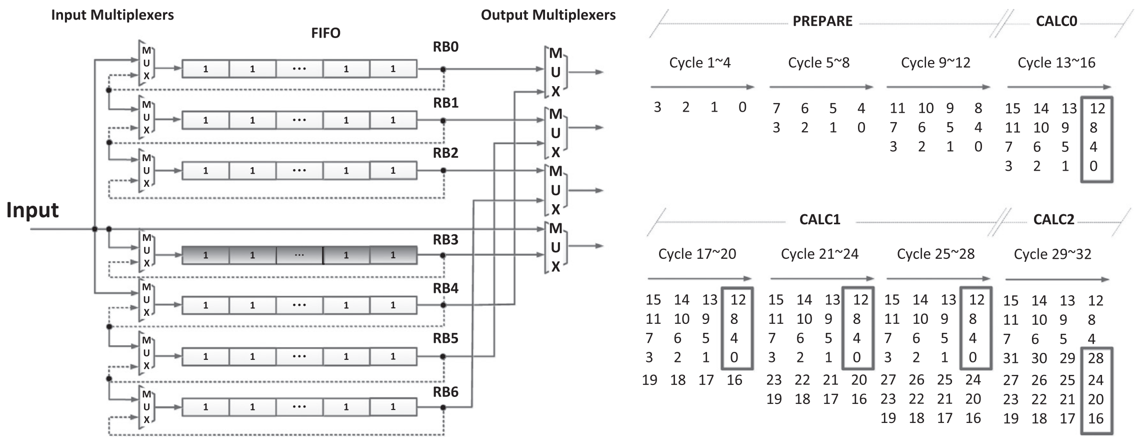

Fig. 7. Detailed structure of PSPPRB and a timing diagram example.

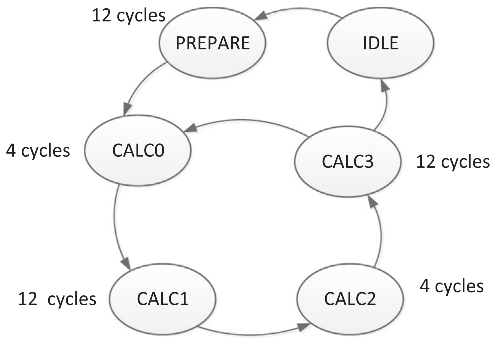

Fig. 8. Finite state machine (FSM) of PSPPRB (for 16-point DFTs).

In the state PREPARE from cycle 1 to cycle 12, RB0–RB2 are configured to the normal type and work as 12 cascaded D flip-flops (DFFs) to store the input data of $x(0), x(1), \ldots, x(11)$ sequentially. In the state CALC0 from cycle 13 to cycle 16, the input data sets for step 1 in stage 1 of the first 16-point DFT are ready to be fed to RR4BF by RB0–RB2 and the input port, such as $x(0), x(4), x(8), x(12)$ illustrated as the black box. During this period, RB0–RB2 are configured to the feedback type and RB3 is configured to the normal type to store the last four input data of $x(12)$–$x(15)$ for the first 16-point DFT computations. Then, in the state CALC1 from cycle 17 to cycle 28, the types of RB0–RB2 are kept unchanged and RB3 is changed to the feedback type for 12 clock cycles. The input data sets of $x(0), x(4), x(8), x(12)$ to $x(3), x(7), x(11), x(15)$ are fed to RR4BF repetitively and periodically by these four feedback type RBs to complete the computations from step 2 to step 4 in stage 1 of the first 16-point DFT. At the same time, RB4–RB6 are configured to the normal type to store the input data of $x(16), x(17), \ldots, x(31)$ for the second 16-point DFT computations sequentially. Once the computations from step 1 to step 4 of the first 16-point DFT are finished, the input data sets for step 1 in stage 1 of the second 16-point DFT are ready to be fed to RR4BF, such as $x(16), x(20), x(24), x(28)$ illustrated as the black box in the state CALC2. Similarly, RB4–RB6 in the state CALC2 and CALC3 work in the same way as RB0–RB2 in the state CALC0 and CALC1 to store and feed the input data to RR4BF for the second 16-point DFT computations.

Consequently, by sharing one of the register banks (RB3) and the input port, RB0–RB2 and RB4–RB6 work in a Ping-Pong mechanism to guarantee that multiple DFTs of the same size can be computed continuously by RR4BF with a 100% adder resource utilization ratio and no any throughput loss. In addition, to provide convincible basis to ignore the extra hardware complexity introduced by the control logic (FSM) and interconnection (MUX, etc.) of PSPPRB, Fig. 9 gives the VLSI implementation results of variable-length PSPPRBs (the length represents the number of DFFs in a single RB). It is obvious that the extra FSM and MUX hold a very low hardware ratio (1–5%) if the length becomes more than 16. Therefore, nearly 1/8 of the storage resources are saved compared with the symmetrical Ping-Pong structured implementations.

3.4. Reading address generating and input data reordering

The proposed pipelined processor not only holds a high throughput, but also reduces the latency significantly due to the 4-way parallel input data path. And no swapping buffers are required to realize the input data reordering for DCT computations by employing the efficient reading address generating scheme.

Fig. 10 shows the detailed 4-way parallel RAM reading address generating logic for both FFTs and DCTs of variable length, which consists of an 8-bit counter, two 10-bit integer subtractors and a controller. By merging the low $\log_2 N-2$ bits of the counter with the $[10-(\log_2 N-2)]$–bit decimal number 0, 1, 2 or 3 provided by the controller, the address generating logic can easily generate four 10-bit reading addresses for FFT or DCT computations. Furthermore, based on Eq. (9), it is obvious that only the second half of the input data should be reordered to implement DCT computations. Thus, two 10-bit integer subtractors are inserted in the last two address generating paths corresponding to Addr2 and Addr3. The two subtractors are used to subtract the address values that are generated for FFT computations with the decimal number 1023, 255, 63 or 15 accordingly. For example, the four addresses for 256-point FFT computations are generated by merging the 4-bit decimal number 0, 1, 2 or 3 with the low 6 bits of the counter illustrated by the timing diagram example in Fig. 10. Then, by subtracting the values of Addr2 and Addr3 for the previous 256-point FFT with the decimal number 255, the reading addresses for 128-point DCT computations are obtained and the timing diagram is also shown in Fig. 10. This method is very suitable for large-size DCT computations, and only small modifications are needed to support different reordering for other FFT-based orthogonal transforms.

4. Design of the fused floating-point operators

4.1. Fused floating-point 4-input adder

As mentioned in Section 3.2, fused floating-point 4-input adders are the main components of RR4BF. However, few literatures mentioned related work. In order to efficiently implement the proposed RR4BF, a novel fused floating-point 4-input adder is proposed in this paper. The 4-input adder is designed to implement the computation of $(A\pm B)\pm(C\pm D)$ compatible with the IEEE 754 single-precision floating point standard [21]. Compared with the equivalent discrete 2-input implementations, the proposed adder optimizes one of the four shifters in the first pipeline stage, and also reuses the computational resources about the normalization shift operation, the exponent adjustment and the special cases processing in the last two pipeline stages, respectively. In order to illustrate the details of resources reusing in the proposed design, the six pipeline stages of the adder are shown in Fig. 11, and the function of each pipeline stage is described as follows.

The first stage: This stage mainly completes the exponent processing, special case detection, bit inversion and alignment shift operations. The exponent processing module computes all the exponent differences between each two operands by six subtractors in parallel, and then determines the shift amounts of the three smaller operands relative to the biggest operand, respectively. Since only three of the four operands should be shifted here, one of the four shifters can be reused to save resources, such as the shaded shifter of stage 1 in Fig. 11. In addition, if A and B (or C and D) have opposite signs, the smaller one (absolute value) is required to perform a bit-inversion operation and the sign of $A\pm B$ (or $C\pm D$) is determined by the bigger one.

The second stage: The main function of this stage is to perform the computations of $(A\pm B)$ and $(C\pm D)$. Two 52-bit carry save adders (CSAs) are employed for these two additions in parallel.

The third stage: The function of this stage is similar to the first stage. Since all shift operations have already been done in the first stage, shifters and bit-inversions are no longer needed here. If the results of $(A\pm B)$ and $(C\pm D)$ have opposite signs, the smaller result (absolute value) is needed to perform another bit-inversion operation, and the final sign of $(A\pm B)\pm(C\pm D)$ is determined based on the difference between these two results in this stage as well.

The fourth stage: This stage completes the final addition of $(A\pm B)\pm(C\pm D)$. It has the similar function as the second stage, but only one 52-bit CSA is needed here.

The fifth stage: The main function of this stage is to prepare the result for the final output. The normalization shift operation and the exponent adjustment are performed in parallel, according to the shift amount computed by the leading zero anticipator (LZA).

The sixth stage: The main functions of this stage are to process the special cases and to output the final result. All the abnormal processing principles are compatible with the IEEE 754 standard.

In order to illustrate the advantages on resources reusing of the proposed 4-input adder, Table 2 shows the comparisons of the worst time delay and area between the conventional discrete 2-input implementations and the proposed design synthesized in a TSMC 65 nm CMOS technology. It is clear that the proposed design not only saves nearly 28% hardware overhead, but also shortens the critical path and reduces the rounding error significantly.

4.2. Fused floating-point 2-term dot product unit

The 2-term dot product computation is defined as the computation of $(A\cdot B)\pm(C\cdot D)$, which has been widely used in complex multiplications. Some useful hardware implementations for

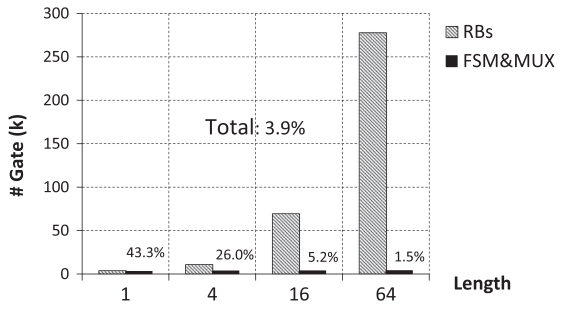

Fig. 9. Hardware overhead of PSPPRBs vs. variable-length.

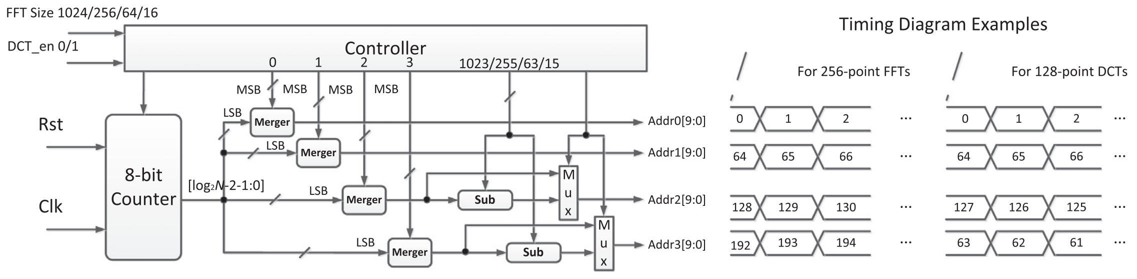

Fig. 10. Reading address generating and timing diagram examples.

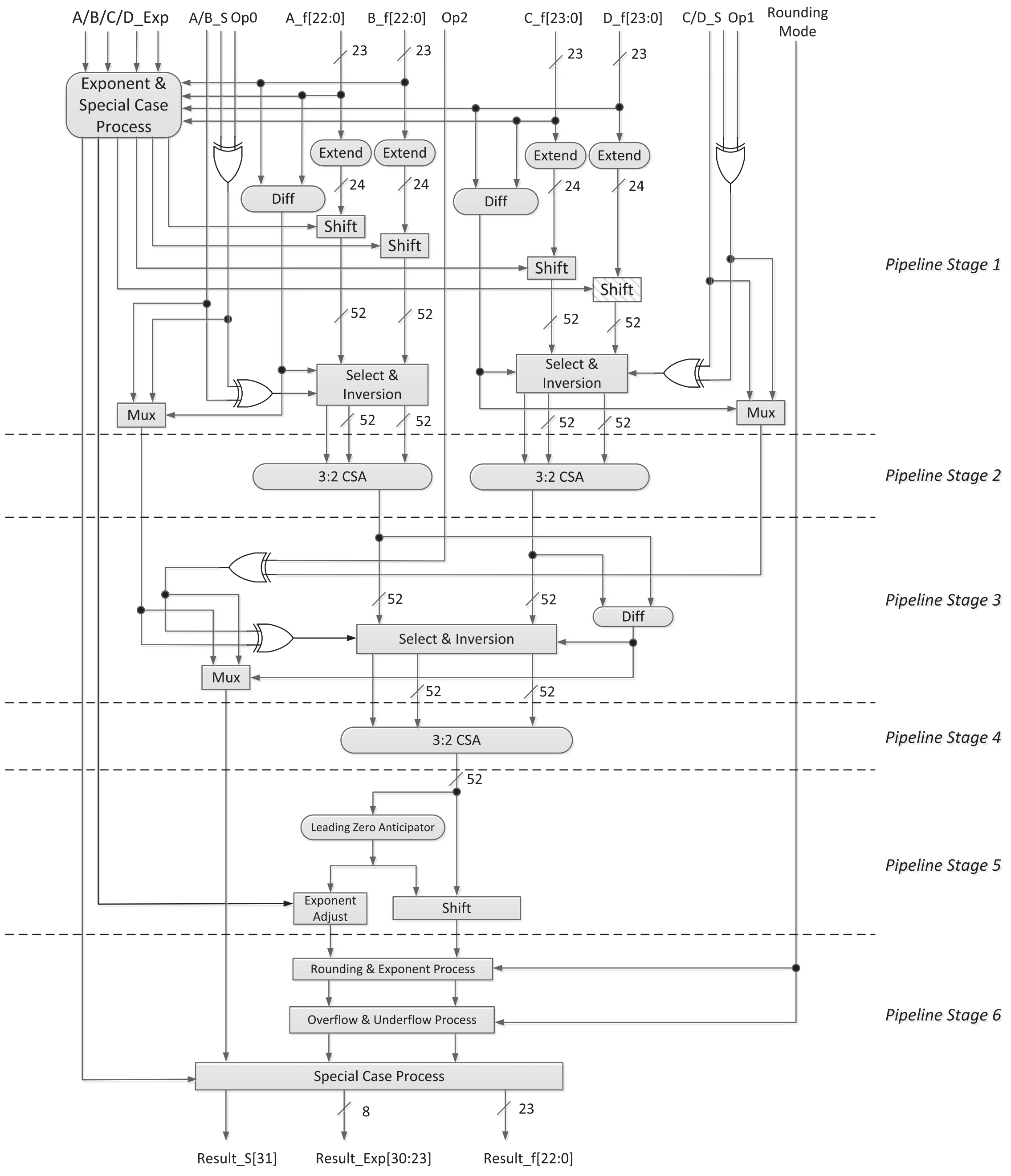

Fig. 11. Detailed structure of the fused 4-input adder.

the 2-term dot product computations have been presented in the existing literatures. For example, the authors in [18] have presented a fused floating-point 2-term dot product unit for FFT butterfly computations recently. Although the state-of-the-art design in [18] shows obvious advantages on hardware overhead saving, worst delay reducing and rounding improvement, some further optimization opportunities can be achieved in this paper. In order to illustrate the details of the proposed design, the six pipeline stages of the 2-term dot product unit are shown in Fig. 12, and the function of each pipeline stage is described as follows.

The first stage: The main functions of this stage are to generate two 48-bit partial products of $A \cdot B$ and $C \cdot D$ by two radix-4 booth encoding modules and to compute the exponents of these two products as Eq. (12). Also, special case detection and sign XOR operations for $A \cdot B$ and $C \cdot D$ are performed.

$$\exp(A \cdot B) = \exp(A) + \exp(B) - 127$$
$$\exp(C \cdot D) = \exp(C) + \exp(D) - 127 \quad (12)$$

The second stage: In this stage, final additions of the two 48-bit partial products are performed in parallel.

Table 2
Comparisons between discrete implementations and the proposed 4-input adder.

Design	Worst delay (ns)	Area (μm^2)	Roundings	Technology (nm)
Discrete	0.87 (100%)	26252.4 (100%)	2	65
Proposed	0.60 (69%)	19005.2 (72.4%)	1	65

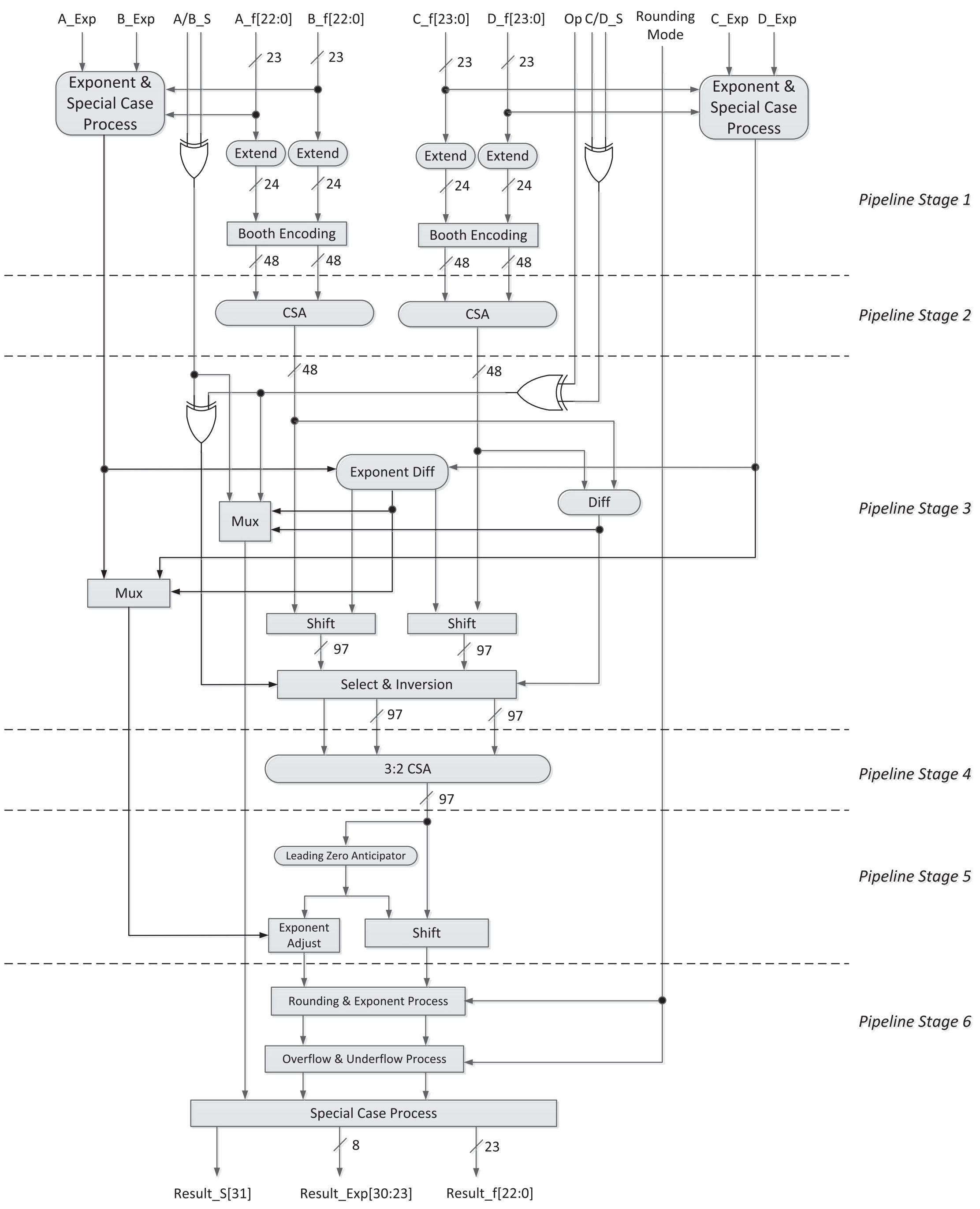

Fig. 12. Detailed structure of the fused 2-term dot product unit.

The third stage: Two subtractions of $\exp(A\cdot B)-\exp(C\cdot D)$ and $A\cdot B-C\cdot D$ are computed in parallel. Then the shift amount for the smaller one of $A\cdot B$ and $C\cdot D$ is determined by the difference between these two exponents, and the final result of the 2-term dot product has the same sign as the bigger one of $A\cdot B$ and $C\cdot D$. Moreover, it is similar to the third stage of the proposed 4-input adder that the smaller product (absolute value) is required to perform a bit-inversion operation if $A\cdot B$ and $C\cdot D$ have opposite signs. And the width of the bit-inversion results is extended to 97 bits to guarantee the computational accuracy.

The fourth stage: The function of this stage is similar to the fourth stage of the 4-input adder, which is the final addition of the two 97-bit shifted results.

The fifth stage: The function of this stage is also similar to the fifth stage of the 4-input adder, LZA computes the shift amount for normalization, and the final exponent adjustment is performed in parallel in this stage.

The sixth stage: The main functions of this stage are to process the special cases and to output the final result. All the abnormal processing principles are compatible with the IEEE 754 standard.

Table 3 shows the comparisons of the worst time delay and hardware overhead between the previous state-of-the-art design [18] and the proposed 2-term dot product unit. For fair comparisons in different technologies, the normalized area (w.r.t. 65nm) is used to evaluate the hardware overhead. It is obvious that better area-efficiency and lower worst delay are achieved in the

Table 3
Comparisons between previous design and the proposed 2-term dot product unit.

Design	Worst delay (ns)	Normalized area (μm^2)	Roundings	Technology (nm)
Swartzlander's [18]	2.72 (100%)	33599.7 (100%)	1	45
Proposed	0.96 (35.3%)	27401.2 (81.5%)	1	65

proposed design, even though the design in [18] uses a faster technology (45 nm).

5. Results and comparisons

5.1. Numerical simulations

As presented in [14–17], detailed theoretical analysis had been done about the advantages of floating-point FFTs over the fixed-point case. As a summary, due to the floating-point representations, floating-point FFTs not only enjoy the almost unlimited input signal magnitude without any concerns about the overflow or underflow problems, but also provide a high SQNR due to the relatively long mantissa (24 bits). In order to verify these advantages of the proposed floating-point design, plenty of numerical simulations have been performed. Then, SQNR can be calculated by comparing the simulation outputs with the double precision floating-point outputs of Matlab, and the formula of the calculation is given by Eq. (13) [18,26]:

$$\text{SQNR} = 10\log\frac{\|X[k]_{\text{ref}}\|^2}{\|Q[X[k]] - X[k]_{\text{ref}}\|^2} \tag{13}$$

where $X[k]_{\text{ref}}$ represents the double precision floating-point reference results of Matlab, and $Q[X[k]]$ is the quantized results obtained from the numerical simulations. The symbol $\|\cdot\|$ is the two-norm operator.

Random experimental environment was established, and the uniformly distributed pseudo-random single-precision floating-point numbers were taken as the input data sets. Random experiments had been run several times to get a better error approximation. The averaged SQNR results of the 16- to 1024-point FFTs and 8- to 512-point DCTs are depicted in Fig. 13, respectively. In order to illustrate the improvement on reducing the rounding error by employing the proposed fused operators, the SQNR of an equivalent reconfigurable FFT processor, where RR4BF and complex multipliers were implemented by discrete 2-input adders and 2-input multipliers, was measured at the same time. Also, the SQNR of a normal radix-2 floating-point design [27] had been simulated to show the advantages of the proposed radix-4 design which utilized high-radix computations combined with operators fusing technique.

Obviously, nearly 140 dB SQNR is achieved for 1024-point FFT computations in the proposed processor. Compared with the equivalent discrete 2-input implementations, the proposed 4-input adder and 2-term dot product unit can improve about 3 dB SQNR on average. Additionally, by combining the high-radix computations and operators fusing, the proposed processor reduces the rounding error significantly, and more than 10 dB SQNR improvement has been achieved over the radix-2 design.

5.2. Hardware resources comparisons

Table 4 shows the hardware resource comparisons among several typical FFT architectures. It is obvious that the proposed design has the highest resource utilization ratio on both the

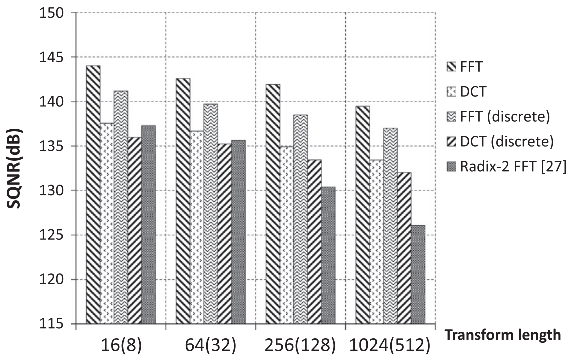

Fig. 13. SQNR results vs. variable-length.

computational resources and storage resources. The number of required adders has been minimized which is especially meaningful in terms of the proposed floating-point processor. The total required storage resources (complex words) is calculated as Eq. (14), while the control logic and the interconnection are ignored based on the experimental results mentioned in Section 3.3:

$$\text{Storage} = \sum_{i=1}^{\log_4^N - 1} 7\cdot 4^i = 7\cdot(N/4-1)/3 \approx 0.58N - 2.33 \tag{14}$$

Compared with the radix-4 multi-path delay commutator (R4MDC) processors, the proposed design reduces both the storage resources and computational resources greatly, and has a significant improvement on the resource utilization ratio of adders and multipliers. Although R4MDC provides a four times throughput of the proposed design, the high requirements on adders and storage of R4MDC is unacceptable in terms of floating-point implementations. In fact, if the RR4BF in the last stage of the proposed processor is replaced by a parallel radix-4 butterfly, the same throughput as R4MDC can be achieved with some increasing of hardware overhead and power consumption. On the other hand, compared with the R4SDF design, which is known as the high memory utilization ratio and high processing speed in pipelined processors [11], the proposed design reduces 75% latency due to the 4-way input data path, and reduces 5/8 adders (a 4-input adder is equivalent to three 2-input adders) due to the proposed RR4BFs. To evaluate the control complexity of the proposed design, the VLSI implementation results of the extra control logic used for intermediate data caching in PSPPRB and R4SDF is presented in Table 5, respectively. The extra control complexity of PSPPRB is nearly 1.5 times of R4SDF, but both of them can be ignored in medium- or large-size FFT computations based on the descriptions in Section 3.3. To provide convincible basis of the area-efficiency due to reducing adders and storage resources in this paper, an equivalent R4SDF processor was implemented, where butterflies and multipliers were realized by 4-input adders and 2-term dot product units for fair comparisons. As a result, Fig. 14 shows both the VLSI implementation results of the proposed design and the R4SDF design. Apparently, compared with the R4SDF design, although little control complexity increases in the proposed design, up to 40% total hardware overhead can be saved without any throughput loss, which is efficient for floating-point implementations.

5.3. Performance comparisons

The proposed processor has been modeled by Verilog HDL and synthesized in a TSMC 65 nm CMOS technology. In order to quantitatively compare the performance with prior typical designs, the latency for N-point FFT computations is calculated as Eq. (15), while the clock cycles spent for filling the pipeline

Table 4
Hardware comparisons between the proposed processor and typical designs for N-point FFTs.

Architecture	Proposed	R4SDF	R4MDC	2^2RSDF	R2MDC	R2SDF
Multipliers	$\log_4 N-1$	$\log_4 N-1$	$3\log_4 N-3$	$\log_4 N-1$	$2\log_4 N-2$	$2\log_4 N-2$
Adders	$\log_4 N$[a]	$8\log_4 N$	$8\log_4 N$	$4\log_4 N$	$4\log_4 N$	$4\log_4 N$
Storage (complex words)	$0.58N-2.33$	$N-1$	$2.5N-4$	$N-1$	$1.5N-2$	$N-1$
Adder utilization	100%	25%	25%	50%	50%	50%
Multiplier utilization	75–100%[b]	75%	25%	75%	50%	50%
Storage utilization	100%	100%	25%	100%	50%	100%
Throughput	1	1	4	1	2	1
Latency[c] (cycles)	$N/4-1$	$N-1$	$5N/4-2$	$N-1$	$3N/2-2$	$N-1$

[a] A 4-input adder is equivalent to three 2-input adders.
[b] The utilization of the multiplier in the first stage is 100% and the others are 75%.
[c] The latency is from the first input samples to the first output samples, and the cycles spent for filling the pipeline registers of adders and multipliers are ignored here.

Table 5
Comparisons of control complexity between R4SDF and the proposed design in a single stage.

Design	No. of 2:1 multiplexers	No. of States in the control FSM	Averaged area (Gate)
R4SDF	4	4	2366.8
PSPPRB	11	6	3639.7

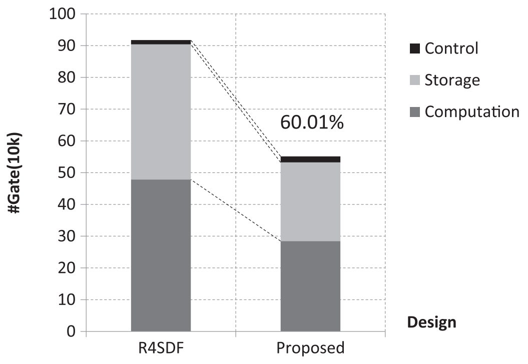

Fig. 14. VLSI implementations of R4SDF and the proposed design.

registers in the fused floating-point operators (adders and dot product units) is 6 based on the descriptions in Section 4:

$$\text{Latency} = \frac{3}{4} \times \sum_{i=1}^{\log_4^N - 1} 4^i + \log_4^N \times 6 + (\log_4^N - 1) \times 6 = (N/4-1) + (2\log_4^N - 1) \times 6 \quad (15)$$

Then, the latency for 1024/256/64/16-point FFT is 309, 105, 45 and 21 cycles, respectively. And, the latency for DCT computations is only 6 cycles more than that of the corresponding FFTs due to the attached floating-point multiplications in the last stage.

Table 6 gives the performance comparisons with previous FFT processors. Since few floating-point FFT/IFFT or DCT/IDCT processors have been reported in existing literatures, we compare our proposed design to some state-of-the-art fixed-point or block floating-point (BFP) processors. Due to the floating-point representations and the proposed resources reusing strategy, the main advantages of the proposed design can be concluded as the large dynamic range, the high SQNR and the area-efficiency. Compared with the design in [6], the proposed floating-point processor shows significant advantages on SQNR because both the computational accuracy and input magnitudes are relatively limited by the 12-bit fixed-point numbers employed in [6]. On the other hand, the authors in [7,10] have made some SQNR improvements due to the data scaling method of BFP technique. For example, over 20 dB SQNR improvement can be achieved in [7] with the same wordlength as [6]. However, it introduces extra hardware complexity for data scaling circuits and is also difficult to be integrated in the systems compatible with the IEEE 754 standard for scientific computing.

Another advantage of the proposed processor is the area-efficiency. Although floating-point units suffer from higher hardware overhead and power consumption than fixed-point implementations, reasonable total hardware overhead and power consumption can be achieved by optimizing the required storage and computational resources in this paper. As a result, the total hardware overhead of the proposed design is equivalent to 543k gates which can be comparable with the fixed-point implementations in [8] and [9]. In order to make fair comparisons among the designs with different technologies and different wordlength, "normalized FFTs per energy" shown in Eq. (16) is employed in Table 6 to reflect the energy efficiency. To alleviate the influences of different technologies, some terms such as MOS sizes (Tech.), the supply voltage and wordlength (WL) are introduced to Eq. (16) as referred to [8,10].

$$\text{Nor. FFTs per Energy} = \frac{\left(\frac{\text{Tech.}}{65\ \text{nm}}\right) \times \left(\frac{\text{Vdd}}{1.0\ \text{V}}\right)^2 \times \left[\frac{2}{3}\left(\frac{\text{WL}}{32}\right) + \frac{1}{3}\left(\frac{\text{WL}}{32}\right)^2\right]}{\text{Power} \times \text{Execution Time} \times 10^6} \quad (16)$$

The detailed power breakdown for variable-length FFT/FFT and DCT/DCT computations are shown in Figs. 15 and 16, respectively. It is obvious that up to 88.3% total power computation can be saved for small-size FFT/IFFT computations by the clock gating strategy. Furthermore, by carefully dividing the pipelines of the internal floating-point arithmetic units, the clock frequency of the processor reaches over 500 MHz. Although it is not efficient to improve the throughput by using a highly parallel butterfly based on the considerations of hardware overhead and power consumption for floating-point implementations, the relatively high clock frequency can also guarantee the high processing speed requirement with the 0.5 GS/s throughput achieved.

6. Conclusion

In this paper, a pipelined reconfigurable processor for variable-length floating-point FFT/IFFT and DCT/IDCT computations is presented. By using the proposed RR4BF and PSPPRB, 75% floating-point adders are saved compared with the conventional parallel radix-4 butterfly, and efficient intermediate data caching mechanism is achieved to guarantee the high throughput for the pipelined design. Moreover, the floating-point 4-input adder and floating-point 2-term dot product unit can further reduce 28% and 19% the computational resources compared with the three discrete 2-input adders and previous 2-term dot product unit, respectively. Due to the floating-point representations, the SQNR of the proposed processor reaches over 139 dB without the limitation of

Table 6
Performance comparisons between the proposed processor and previous designs.

Design	Proposed	Cho's [6]	Huang's [7]	Yang's [8]	Yang's [9]	Tang's [10]	Xilinx's[b] [27]
Technology/supply voltage	65 nm/1.0 V	90 nm/1.2 V	90 nm	90 nm/1.0 V	65 nm/0.45 V	0.18 μm/1.8 V	–
FFT size	16–1024	512	512	128–2048	128–2048	64–1024	64–1024
Data type	Float	Fixed	Fixed	Fixed	Fixed	Fixed	Float
BFP employment	–	No	Yes	No	No	Yes	–
Wordlength	32-bit	12-bit	12-bit	10-bit	12-bit	10-bit	32-bit
IEEE 754 compatibility	Yes	No	No	No	No	No	Yes
SQNR	139 dB	35 dB	57 dB	–	–	40.3 dB	133 dB[c]
DCT support	Yes	No	No	No	No	No	No
Gate count	~543k	290k	–	–	1100k	–	–
Area (mm^2)	0.87+0.133[a]	0.78	0.93	3.1	1.375	3.2	–
Nor. Area (mm^2)	1.003	0.407	0.485	1.617	1.375	0.417	–
Clock frequency	500 MHz	310 MHz	324 MHz	40 MHz	1.25–20 MHz	300 MHz	395 MHz
Throughput (GS/s)	0.5	2.5	2.59	0.078	0.08	2.4	~0.4
Execution time/Latency for 1024-point FFT (μs)	2.05/0.62	–	–	25.6	12.8	–	2.6/11.4
Power (mW)	43.5–372.3	92.8	42	51.69–63.72	4.05, 8.55	382~507	–
Nor. FFTs per energy (1024-point)	1.31	–	–	0.207	1.507	–	–

[a] The area of external memory is about 0.133 mm^2.
[b] The selected FFT IP type is radix-4, burst I/O and single-precision floating-point with Bit/Digit reversed output ordering.
[c] The value is obtained from the simulation experiments.

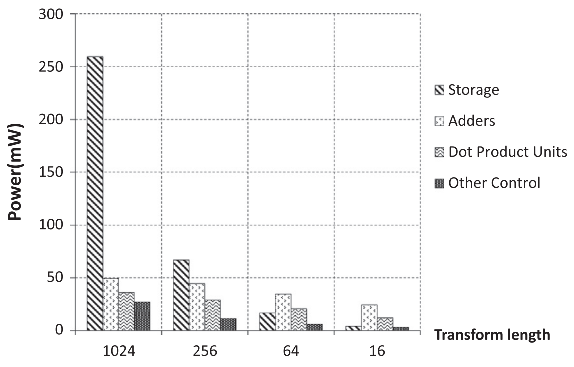

Fig. 15. Power breakdown of variable-length FFTs.

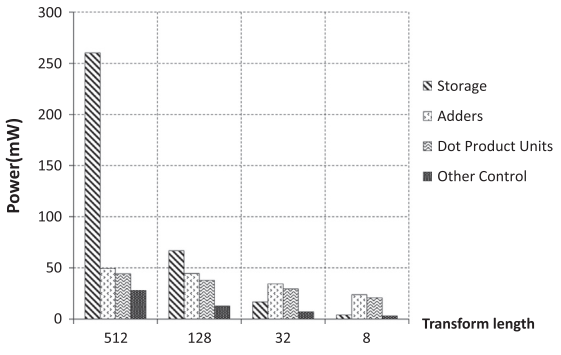

Fig. 16. Power breakdown of variable-length DCTs.

dynamic range. Compared with previous work, improvements on area-efficiency, resource-utilization, power-scalability and high SQNR are exhibited, relatively. The latency of the processor is about 1/4 of the R4SDF design for FFT computations without any throughput loss. Logic synthesis results show that the power consumption ranges from 43.5 mW to 372.3 mW for 16- to 1024-point FFTs at 500 MHz, and the total hardware overhead is equivalent to 543k NAND2 gates. Consequently, reasonable total hardware overhead and power consumption are achieved in terms of the proposed floating-point design. In addition, the proposed reconfigurable processor can also be modified and be extended to support longer-size FFTs/DCTs or other FFT-based orthogonal transformations easily. Hence, the proposed reconfigurable processor has good potentiality in scientific computing and high-resolution imaging applications.

Acknowledgments

This work is supported by the National Key Science and Technology Projects of China, and special thanks are due to the referees that have made several suggestions to significantly improve the paper.

References

[1] Y. Ogata, T. Endo, N. Maruyama, S. Matsuoka, An efficient, model-based CPU-GPU heterogeneous FFT library, in: Proceedings of the International Symposium on Parallel and Distributed Processing (IPDPS), 2008, pp. 1–10.
[2] S. Reddaway, P. Bruno, P. Rogina, R. Pancoast, Ultra-high performance, data parallel radar implementations, IEEE Aerosp. Electron. Syst. Mag. 21 (4) (2006) 3–7.
[3] F. Messaoud, A. Peizerat, A. Dupret, Y. Blanchard, On-chip compression for HDR image sensors, in: Proceedings of the Conference Design and Architectures for Signal and Image Processing (DASIP), 2010, pp. 176–182.
[4] K. Kloker, B. Lindsley, N. Baron, G. Sohie, Efficient FFT implementation on an IEEE floating-point digital signal processor, in: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 2, 1989, pp. 1302–1305.
[5] B. Duan, W. Wang, X. Li, C. Zhang, P. Zhang, N. Sun, Floating-point mixed-radix FFT core generation for FPGA and comparison with GPU and CPU, in: Proceedings of the International Conference on Field-Programmable Technology (FPT), 2011, pp. 1–6.
[6] T. Cho, H. Lee, A high-speed low-complexity modified radix-2^5 FFT processor for high rate WPAN applications, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 21 (1) (2013) 187–191.
[7] S.-J. Huang, S.-G. Chen, A high-throughput radix-16 FFT processor with parallel and normal input/output ordering for IEEE 802.15.3c systems, IEEE Trans. Circuits Syst. I: Regul. Pap. 59 (8) (2012) 1752–1765.
[8] K.-J. Yang, S.-H. Tsai, G. Chuang, MDC FFT/IFFT processor with variable length for MIMO-OFDM systems, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 21 (4) (2013) 720–731.
[9] C.-H. Yang, T.-H. Yu, D. Markovic, Power and area minimization of reconfigurable FFT processors: a 3GPP-LTE example, IEEE J. Solid-State Circuits (JSSC) 47 (3) (2012) 757–768.
[10] S.-N. Tang, C.-H. Liao, T.-Y. Chang, An area- and energy-efficient multimode FFT processor for WPAN/WLAN/WMAN systems, IEEE J. Solid-State Circuits (JSSC) 47 (6) (2012) 1419–1435.
[11] C.-T. Lin, Y.-C. Yu, L.-D. Van, Cost-effective triple-mode reconfigurable pipeline FFT/IFFT/2-D DCT processor, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 16 (8) (2008) 1058–1071.
[12] G. Zhong, F. Xu, A.N. Willson Jr., A power-scalable reconfigurable FFT/IFFT IC based on a multi-processor ring, IEEE J. Solid-State Circuits (JSSC) 41 (2) (2006) 483–495.

[13] D. Spaderna, P. Green, K. Tam, T. Datta, M. Kumar, An integrated floating point vector processor for DSP and scientific computing, in: Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors (ICCD), 1989, pp. 8–13.

[14] A. Oppenheim, C. Weinstein, Effects of finite register length in digital filtering and the fast Fourier transform, Proc. IEEE 60 (8) (1972) 957–976.

[15] C.J. Weinstein, Quantization effects in digital filters, Technical Report, DTIC Document, 1969.

[16] G.U. Ramos, Roundoff error analysis of the fast Fourier transform, Math. Comput. 25 (116) (1971) 757–768.

[17] C. Weinstein, Roundoff noise in floating point fast Fourier transform computation, IEEE Trans. Audio Electroacoust. 17 (3) (1969) 209–215.

[18] E. Swartzlander, H. Saleh, FFT implementation with fused floating-point operations, IEEE Trans. Comput. 61 (2) (2012) 284–288.

[19] Z. Zhang, D. Wang, Y. Pan, D. Wang, X. Zhou, G. Sobelman, FFT implementation with multi-operand floating point units, in: Proceedings of the IEEE International Conference on ASIC (ASICON), 2011, pp. 216–219.

[20] J. Sohn, E. Swartzlander, Improved architectures for a fused floating-point add-subtract unit, IEEE Trans. Circuits Syst. I: Regul. Pap. 59 (10) (2012) 2285–2291.

[21] IEEE standard for floating-point arithmetic, IEEE Std 754-2008.

[22] A. Despain, Fourier transform computers using CORDIC iterations, IEEE Trans. Comput. C-23 (10) (1974) 993–1001.

[23] L.R. Rabiner, B. Gold, Theory and Application of Digital Signal Processing, vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[24] G. Bi, E. Jones, A pipelined FFT processor for word-sequential data, IEEE Trans. Acoust. Speech Signal Process. 37 (12) (1989) 1982–1985.

[25] N. Ahmed, T. Natarajan, K. Rao, Discrete cosine transform, IEEE Trans. Comput. C-23 (1) (1974) 90–93.

[26] C.-Y. Wang, C.-B. Kuo, J.-Y. Jou, Hybrid wordlength optimization methods of pipelined FFT processors, IEEE Trans. Comput. 56 (8) (2007) 1105–1118.

[27] Xilinx, LogiCORE IP Fast Fourier Transform v7.1. ⟨http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf⟩.