Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2009, Article ID 219140, 9 pages
doi:10.1155/2009/219140

Research Article

Pipeline FFT Architectures Optimized for FPGAs

Bin Zhou,[1,2] Yingning Peng,[1] and David Hwang[2]

[1] *Department of Electronic Engineering, Tsinghua University, Beijing 100084, China*
[2] *Department of Electrical and Computer Engineering, George Mason University, 4400 University Drive, Fairfax, VA 22030, USA*

Correspondence should be addressed to David Hwang, dhwang@gmu.edu

Received 28 February 2009; Accepted 23 June 2009

Recommended by Cesar Torres

This paper presents optimized implementations of two different pipeline FFT processors on Xilinx Spartan-3 and Virtex-4 FPGAs. Different optimization techniques and rounding schemes were explored. The implementation results achieved better performance with lower resource usage than prior art. The 16-bit 1024-point FFT with the R2^2SDF architecture had a maximum clock frequency of 95.2 MHz and used 2802 slices on the Spartan-3, a throughput per area ratio of 0.034 Msamples/s/slice. The R4SDC architecture ran at 123.8 MHz and used 4409 slices on the Spartan-3, a throughput per area ratio of 0.028 Msamples/s/slice. On Virtex-4, the 16-bit 1024-point R2^2SDF architecture ran at 235.6 MHz and used 2256 slice, giving a 0.104 Msamples/s/slice ratio; the 16-bit 1024-point R4SDC architecture ran at 219.2 MHz and used 3064 slices, giving a 0.072 Msamples/s/slice ratio. The R2^2SDF was more efficient than the R4SDC in terms of throughput per area due to a simpler controller and an easier balanced rounding scheme. This paper also shows that balanced stage rounding is an appropriate rounding scheme for pipeline FFT processors.

1. Introduction

The Fast Fourier Transform (FFT), as an efficient algorithm to compute the Discrete Fourier Transform (DFT), is one of the most important operations in modern digital signal processing and communication systems. The pipeline FFT is a special class of FFT algorithms which can compute the FFT in a sequential manner; it achieves real-time behavior with nonstop processing when data is continually fed through the processor. Pipeline FFT architectures have been studied since the 1970's when real-time large scale signal processing requirements became prevalent. Several different architectures have been proposed, based on different decomposition methods, such as the Radix-2 Multipath Delay Commutator (R2MDC) [1], Radix-2 Single-Path Delay Feedback (R2SDF) [2], Radix-4 Single-Path Delay Commutator (R4SDC) [3], and Radix-2^2 Single-Path Delay Feedback (R2^2SDF) [4]. More recently, Radix-2^2 to Radix-2^4 SDF FFTs were studied and compared in [5]; in [6] an R2^3SDF was implemented and shown to be area efficient for 2 or 3 multipath channels. Each of these architectures can be classified as multipath or single-path. Multipath approaches can process M data inputs simultaneously, though they have limitations on the number of parallel data-paths, FFT points, and radix. This paper focuses on single-path architectures.

From the hardware perspective, Field Programmable Gate Array (FPGA) devices are increasingly being used for hardware implementations in communications applications. FPGAs at advanced technology nodes can achieve high performance, while having more flexibility, faster design time, and lower cost. As such, FPGAs are becoming more attractive for FFT processing applications and are the target platform of this paper.

The primary goal of this research is to optimize pipeline FFT processors to achieve better performance and lower cost than prior art implementations. In this paper, two comparative implementations (R4SDC and R2^2SDF) of pipeline FFT processors targeted towards Xilinx Spartan-3 and Virtex-4 FPGAs are presented. Different parameters such as throughput, area, and SQNR are compared.

The rest of the paper is organized as follows. Section 2 discusses the methodology used to select the two architectures. Section 3 describes the implementation tools and optimization methods used to improve performance and reduce resource utilization. Section 4 explains the balanced rounding schemes that were implemented and their impact on the signal-to-quantization noise ratio (SQNR). Section 5 presents the results, and Section 6 presents some brief conclusions.

Table 1: Hardware resource requirements comparison of pipeline FFT architectures (based on [4]).

	Complex multipliers	Complex adders	Memory size	Control logic	Comp. Utilization	
					add/sub	Multiplier
R2SDF	$\log_2 N - 2$	$2\log_2 N$	$N - 1$	simple	50%	50%
R4SDF	$\log_4 N - 1$	$8\log_4 N$	$N - 1$	medium	25%	75%
R4SDC	$\log_4 N - 1$	$3\log_4 N$	$2N - 2$	complex	100%	75%
R2^{2}SDF	$\log_4 N - 1$	$4\log_4 N$	$N - 1$	simple	75%	75%
R2MDC	$\log_2 N - 2$	$2\log_2 N$	$3N/2 - 2$	simple	50%	50%
R4MDC	$3(\log_4 N - 1)$	$8\log_4 N$	$5N/2 - 4$	medium	25%	25%

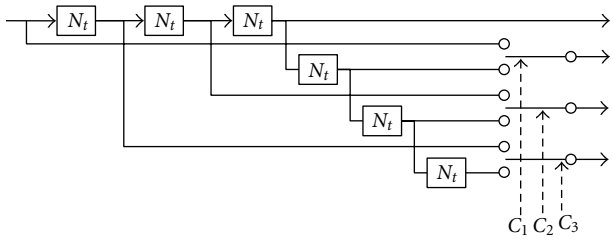


Figure 1: R4SDC commutator of stage t.

Table 2: Implementation tools.

Design step	Tool
VHDL simulation	ModelSim SE 6.2b
FPGA synthesis	Synplicity Synplify Pro; Xilinx XST
FPGA implementation	Xilinx ISE 9.1
Target FPGA	Spartan-3 Family; Virtex-E Family; Virtex-4 Family
Verification	Matlab R2006a

2. Pipeline FFT Architectures

2.1. Architecture Selection. The major characteristics and resource requirements of several pipeline FFT architectures are listed in Table 1. Computational efficiency is measured by resource utilization percentage—how often the resources are in an active state versus an idle state. As shown in the table, the radix-4 Single-Path Delay-Commutator (R4SDC) and radix-2^2 Single Path Delay Feedback (R2^{2}SDF) architectures provide the highest computational efficiency and were selected for implementation. The R4SDC architecture is appealing due to the computational efficiency of its addition; however the controller design is complex. The R2^{2}SDF architecture has a simple controller but a less efficient addition scheme. These designs are both radix-4 and scalable to an arbitrary FFT size N (N is a power of 4).

2.2. R4SDC Architecture.

R4SDC Algorithms. The R4SDC was proposed by Bi and Jones [3] and uses an iterative architecture to calculate the radix-4 FFT. The key to the algorithm is splitting the FFT into different stages by using different radices. In this paper, the radix is always 4.

The derivation starts from the fundamental DFT equation for an N-point FFT:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N{}^{nk} \quad \left(k = 0, 1 \cdots N-1; W_N = e^{-j(2\pi/N)}\right). \tag{1}$$

N can be represented as composite of v numbers $N = r_1 r_2 \cdots r_v$ and defined as

$$N_t = \frac{N}{r_1 r_2 \cdots r_t}, \quad 1 \le t \le v-1, \tag{2}$$

where t is the stage, and r_t is the stage radix. After putting (2) into (1) and applying the relationship $W_{N_i N_j}^{N_j k} = W_{N_i}{}^{k}$, (1) becomes

$$X(k) = \sum_{q_1=0}^{N_1-1} W_N{}^{q_1 k} \sum_{p=0}^{r_1-1} x(N_1 p + q_1) W_{r_1}{}^{pk}. \tag{3}$$

Indeces k_1 and m_1 can be defined by $k = r_1 k_1 + m_1$, where $0 \le k_1 \le N_1 - 1$, $0 \le m_1 \le r_1 - 1$. Equation (3) becomes

$$\begin{aligned} X(r_1 k_1 + m_1) &= \sum_{q_1=0}^{N_1-1} x_1(q_1, m_1) W_{N_1}{}^{q_1 k_1}, \\ x_1(q_1, m_1) &= W_N{}^{q_1 m_1} \sum_{p=0}^{r_1-1} x(N_1 p + q_1) W_{r_1}{}^{p m_1}. \end{aligned} \tag{4}$$

Therefore the complete N-point DFT can be written as $v - 1$ different stages with intermediate stages in a recursive equation:

$$x_t(q_t, m_t) = W_{N_{t-1}}^{q_t m_t} \sum_{p=0}^{r_t-1} x_{t-1}(N_t p + q_t, m_{t-1}) W_{r_t}^{p m_t}. \tag{5}$$

$W_{N_{t-1}}^{q_t m_t}$ is the twiddle factor. For radix-4, the equations become

$$X(4k_1 + m_1) = \sum_{q_1=0}^{N/4-1} x_1(q_1, m_1) W_{N/4}{}^{q_1 k_1},$$
$$x_t(q_t, m_t) = W_{N_{t-1}}^{q_t m_t} \sum_{p=0}^{3} x_{t-1}(N_t p + q_t, m_{t-1}) W_4{}^{p m_t} \quad (6)$$

The R4SDC architecture is presented in Figures 1–3. An N-point radix-4 pipeline FFT is decomposed to $\log_4 N$ stages. Each stage consists of a commutator, a butterfly, and a complex multiplier. Figure 1 outlines the commutator for the R4SDC. Its six shift registers provide N_t delays. The control signals are generated by logic functions. The butterfly element, shown in Figure 2, performs the summation, where trivial multiplication is replaced by add/sub and imaginary/real part swapping. Figure 3 shows the overall architecture.

2.3. R2²SDF Architecture. The R2²SDF architecture was proposed by He and Torkelson [4] and also begins from (1). He applies a 3-dimensional index map:

$$n = \left\langle \frac{N}{2} n_1 + \frac{N}{4} n_2 + n_3 \right\rangle_N; \qquad k = \langle k_1 + 2k_2 + 4k_3 \rangle_N. \quad (7)$$

Using the Common Factor Algorithm (CFA) to decompose the twiddle factor, the FFT can be reconstructed as a set of 4 DFTs of length $N/4$:

$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{N/4-1} \left[H(k_1, k_2, n_3) W_N^{n_3(k_1+2k_2)} \right] W_{N/4}^{n_3 k_3} \quad (8)$$

$H(k_1, k_2, n_3)$ can be expressed as

$$H(k_1, k_2, n_3) = \overbrace{[x(n_3) + (-1)^{k_1} x(n_3 + (N/2))]}^{BF\,2I} + (-j)^{(k_1+2k_2)} \underbrace{\overbrace{[x(n_3 + (N/4)) + (-1)^{k_1} x(n_3 + (3/4)N)]}^{BF\,2I}}_{BF\,2II}, \quad (9)$$

The R2²SDF algorithm can be mapped to the architecture shown in Figures 4–6. The number of stages is $\log_4 N$. Every stage contains two butterfly elements, each associated by an N_t feedback shift register. A simple counter creates the control signals. Pipeline registers can be added between butterfly elements and between stages. Registers are also added inside the complex multipliers to reduce the critical path through the summation to the multiplier. The total latency is approximately $N + 4(\log_4 N - 1)$ cycles.

3. FPGA-Based Implementations and Optimizations

3.1. Specifications, Tool Flow, and Verification. Both of these FFT architectures were implemented with generic synthesizable VHDL code and verified with simulation against Matlab scripts using Modelsim. Synplify or XST was used to perform the synthesis, and ISE was used for place and route and implementation. The architectures were optimized to achieve maximum throughput with minimal area (slices). The tools and development environment used are shown in Table 2.

3.2. General Optimization Methods. Some general optimization measures were performed, including FSM encoding, retiming, and CAD-related optimizations. Since the FFT processors were targeted to Xilinx Spartan-3 and Virtex-4 FPGAs (as well as synthesized for Virtex-E FPGAs), the SRL16 component, which can implement a 16-bit shift register within a single LUT, was inferred as much as possible to preserve LUTs. This particularly helped the R2²SDF architecture because of the large number of shift registers. R4SDC also benefited from SRL16 components in its commutator registers. Block RAMs were used to store twiddle factors, which dramatically reduced the combinational logic utilization.

3.3. Architecture-Specific Optimization. A number of architecture-specific optimizations were used. For both architectures, a complex multiplication technique was used. Usually, a complex multiplication is computed as:

$$(a + bi) \times (c + di) = a \times c - b \times d + (a \times d + b \times c)i. \quad (10)$$

This requires 4 multiplications and 2 add/suboperations. As is well known, the equation is simplified to save one multiplier:

$$(a + bi) \times (c + di) = [a \times (c + d) - (a + b) \times d] + [a \times (c + d) + (a - b) \times c]i. \quad (11)$$

This requires only 3 multiplications and 5 add/suboperations. Pipeline registers were also added in order to avoid the long critical path brought by the connection of real adders and multipliers. Figure 7 shows the pipeline stages inserted which were effective to reduce the critical path (REG means pipeline register).

3.3.1. R4SDC Optimization. The R4SDC has a complex controller, which creates a long critical path. By observing that all stages have the same control bits but have different sequences, using a ROM with an incremental address was

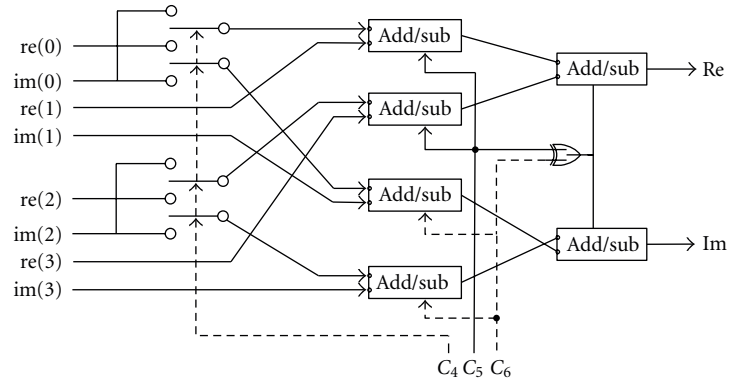


Figure 2: R4SDC butterfly element of stage t.

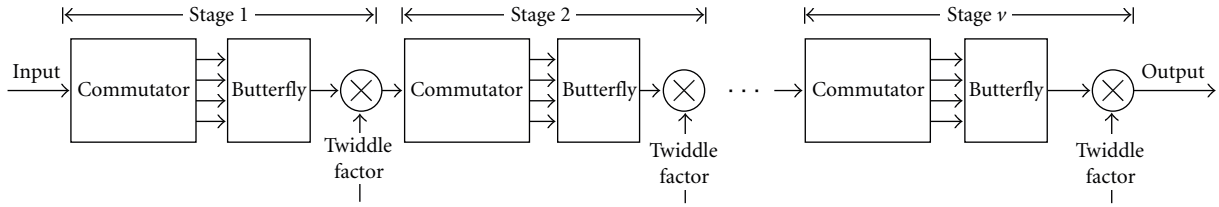


Figure 3: N-point R4SDC pipeline FFT processor architecture.

a simpler solution than using a complex FSM. Pipeline registers were also added to the butterfly elements, multipliers, and between stages. Figure 8 illustrates the addition of pipeline registers to cut the critical path efficiently within the butterfly element. Since two continuous add/subelements bring about a long propagation path, they were split using pipelining. Figure 9 shows the addition of pipeline registers between majority elements and between stages. For timing purposes, the applicable control signals were also buffered.

There were some special measures taken into account within controller in order to keep proper timing of signals. Twiddle factors should also be delayed to cope with the delayed sequence.

3.3.2. R2²SDF Optimization. Due to its simple control requirements, a simple counter was sufficient as the entire controller for the R2²SDF. To speed up the controller, a fast adder could potentially be faster than a simple ripple-carry adder. However, due to the small number of stages ($\log_4 N$), no substantial savings were found for a fast adder. Pipeline registers were added between major elements and also between stages. Note that the R2²SDF is not suited for adding pipeline registers within individual butterfly elements, because this would break the timing for the data feedback path. Figure 10 presents the pipelined stages. Note that registers were only added between element units; in addition, registers were added as necessary to keep the control signals properly timed.

4. Rounding Scheme and SQNR

Due to finite wordlength effects, the implemented FFTs always scaled by $1/N$ at the output of the design. This scaling factor was distributed as divide-by-two operations throughout each stage to reduce error propagation. As is well known, truncation or conventional rounding (which is denoted as round-half-up) will bring a notable quantization error bias in divide-by-two operations, and this bias will accumulate throughout the processing chain [5]. To alleviate the bias, three unbiased rounding methods are investigated for division by two.

Sign Bit-Based Rounding. In this scenario, if the MSB of the number to be divided is 0 (i.e., positive number) it is rounded-half-up. This will have a positive bias. On the other hand, if the MSB is 1 (i.e., negative number) it is truncated, leaving a negative bias. Assuming that the positive and negative numbers are uniformly distributed, this approach will lead to an unbiased rounding scheme. However, selecting the bias based on the MSB implies that these two rounding methods coexist in a single rounding position, which requires extra hardware. This increases the critical path, harming the performance. So it is not chosen.

Randomized [7]. In this scenario, if the bit to be rounded is 1, a random up or down rounding is performed. If it is 0, the same rounding scheme as done previously is performed. From the statistical point of view, no bias exists. But this method requires a random bit generator and a long accumulation time, requiring big extra hardware resources and significantly affects the performance. So it is not implemented.

Balanced Stages Rounding [11]. This rounding method explores balancing between stages. Round-half-up and truncation are used in an interlaced fashion, as shown in Figure 11.

Table 3: SQNR with different FFT sizes.

	FFT size	Input data width	Twiddle factor width	Stage number	SQNR (dB)
R4SDC	16	16	16	2	82.29
	64	16	16	3	73.49
	256	16	16	4	67.47
	1024	16	16	5	61.25
R2^{2}SDF	16	16	16	2	81.82
	64	16	16	3	74.47
	256	16	16	4	68.22
	1024	16	16	5	62.68

Table 4: Implementation results on Spartan-3 devices.

	Point size	Input data width	Twiddle factor width	Slices	Block RAM	Max. speed (MHz)	Latency (cycles)	Transform time Cycles	Transform time Time (μs)	Throughput (MS/s)	Throughput/area (MS/s/slice)
R4SDC	16	16	16	468	2	108.20	21	16	0.15	108.20	0.231
	64	16	16	952	2	107.23	73	64	0.60	107.23	0.113
	256	16	16	1990	3	111.98	269	256	2.76	111.98	0.056
	1024	16	16	4409	8	123.84	1041	1024	8.27	123.84	0.028
R2^{2}SDF	16	16	16	427	2	121.24	22	16	0.13	121.24	0.284
	64	16	16	810	2	98.14	74	64	0.65	98.14	0.121
	256	16	16	1303	3	98.73	270	256	2.59	98.73	0.076
	1024	16	16	2802	8	95.25	1042	1024	10.75	95.25	0.034

Table 5: Implementation results on Virtex-4 devices.

	Point size	Input data width	DSP48	Slices	Block RAM	Max. speed (MHz)	Latency (cycles)	Transform time Cycles	Transform time Time (μs)	Throughput (MS/s)	Throughput/area (MS/s/slice)
R4SDC	16	16	4	530	1	236.7	21	16	0.07	236.7	0.447
	64	16	8	803	2	236.4	73	64	0.27	236.4	0.294
	256	16	12	1370	3	218.9	269	256	1.17	218.9	0.160
	1024	16	16	3064	8	219.2	1041	1024	4.67	219.2	0.072
R2^{2}SDF	16	16	4	517	1	237.9	22	16	0.07	237.9	0.460
	64	16	8	779	2	236.7	74	64	0.27	236.7	0.304
	256	16	12	1234	3	236.7	270	256	1.08	236.7	0.192
	1024	16	16	2256	8	235.6	1042	1024	4.35	235.6	0.104

Table 6: Performance comparison versus prior art on Virtex-E devices.

FFT Design	Point size	Input data width	Twiddle factor width	Slices	Block RAM	Max. speed (MH)	Latency (Cycle)	Transform time Cycles	Transform time Time (μs)	Throughput (MS/s)	Throughput/area (MS/s/slice)
Amphion [8]	1024	13	13	1639	9	57	5097	4096	71.86	14.25	0.009
Xilinx [8, 9]	1024	16	16	1968	24	83	4096	4096	49.35	20.75	0.011
Sundance [10]	1024	16	10	8031	20	49	1320	1320	27.00	49.00	0.006
Suksawas R2^{2}SDF [8]	1024	16	16	7365	28	82	1099	1024	12.49	82.00	0.011
Our R2^{2}SDF	1024	16	16	5008	32	95.0	1042	1024	10.78	95.00	0.019
Our R4SDC	1024	16	16	7052	32	94.2	1041	1024	10.87	94.20	0.013

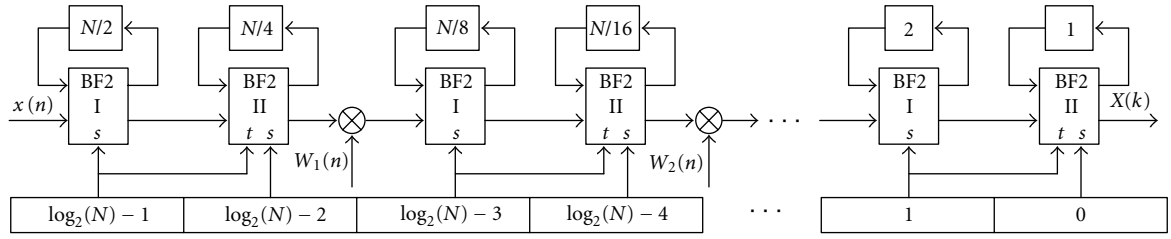


Figure 4: N-point R2^{2}SDF pipeline FFT processor architecture.

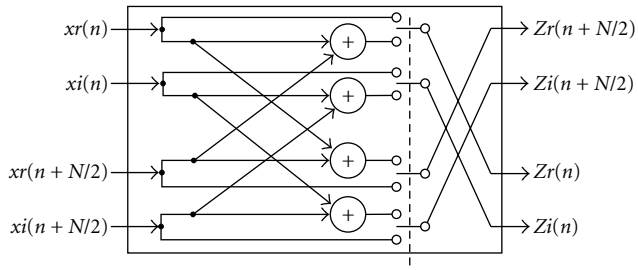


Figure 5: R2^{2}SDF BF2 I structure.

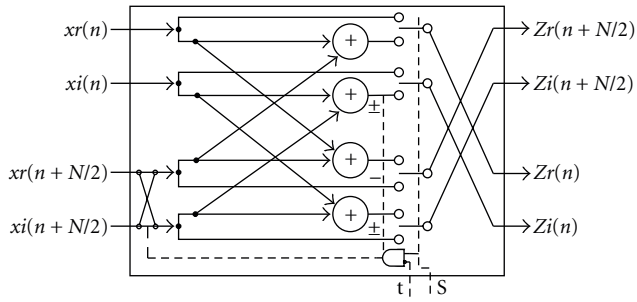


Figure 6: R2^{2}SDF BF2 II structure.

For an even number of stages, this will achieve the same result as the randomized approach, while having a smaller resource usage and simpler control. This scheme fits the R2^{2}SDF architecture particularly well, because the two butterfly elements within same stage of R2^{2}SDF can be naturally balanced. This method was chosen for the designs presented in the paper.

In order to compute the signal-to-quantization noise ratio (SQNR), random generated noise was used as the input to the pipeline FFT. A Matlab script generated double precision floating point FFT results, which were used as the true values. Figure 12 shows how they are compared with the fixed-point implementations. Random experiments were run several times and averaged to get a better error approximation.

5. Results and Analysis

5.1. SQNR Results. Figure 13 shows the SQNR results with different rounding schemes (balanced stages, truncation, and round-half-up), for R4SDC and R2^{2}SDF, respectively, for a 16-bit data width (input data, twiddle factors, and output data are 16 bits). The balanced stage rounding typically improved the SQNR by 1-2 dB. The balanced stages scheme gives better SQNR, because it leverages the randomness between stages. The truncation and round-half-up only reserve half of the information.

Table 3 presents the SQNR results as they vary with FFT size. The larger the FFT, the worse the SQNR due to the longer processing chain. Both architectures gave comparable results in terms of SQNR. It is clear that larger data widths will also give better SQNR but will increase area and critical path. A 16-bit wordlength is a sufficient choice for many signal processing applications.

The FFT architectures with smaller wordlengths than 16 bits are also implemented. The example in the Figure 14 shows the R4SDC architecture for an $N = 1024$ point FFT. Every bit of word length increment brings about 6 dB of SQNR gain.

5.2. Implementation Results. Table 4 gives the performance results of both architectures with different FFT sizes on Spartan-3 FPGAs (90 nm) [12]. The R2^{2}SDF achieved a smaller area and better throughput per area than the R4SDC. Due to the pipeline design, the maximum clock frequency did not change drastically with FFT size for either design. As expected the throughput per area decreases for larger FFT sizes, which require more stages and area.

Table 5 presents the results on Virtex-4 FPGAs (90 nm) using a 16-bit wordlength. Virtex-4 FPGAs have hardwired DSP modules called DSP48 blocks, which are high-speed modules optimized for signal processing operations such as multiply-accumulate, and FIR filtering. By utilizing these DSP48 blocks, the maximum clock frequency increased substantially over the Spartan-3 devices.

Comparisons with prior art are shown in Table 6, which shows publicly available pipeline FFT implementations on FPGAs from literature. For fair comparison, since many of the prior art implementations were implemented on Virtex-E FPGAs (180 nm), the designs are also implemented on Virtex-E. The best performance for the R2^{2}SDF method for a 16-bit 1024-point FFT was published by Sukhsawas and Benkrid in [8]. They used Handel-C as a rapid prototype language and implemented the design on Virtex-E FPGAs. They achieved 82 MHz maximum clock frequency and 7365 slices, giving a throughput per area ratio of 0.011 Msamples/s/slice.

On the Virtex-E, our R2^{2}SDF achieved better performance of 95 MHz and a smaller area of 5008 slices, giving a superior throughput per area ratio of 0.019 Msamples/s/slice.

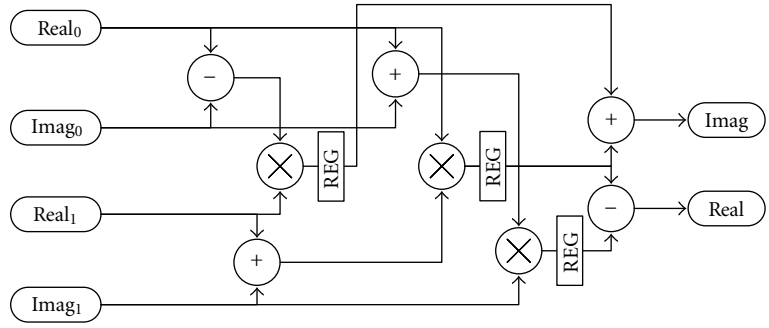


Figure 7: The pipelined complex multiplier.

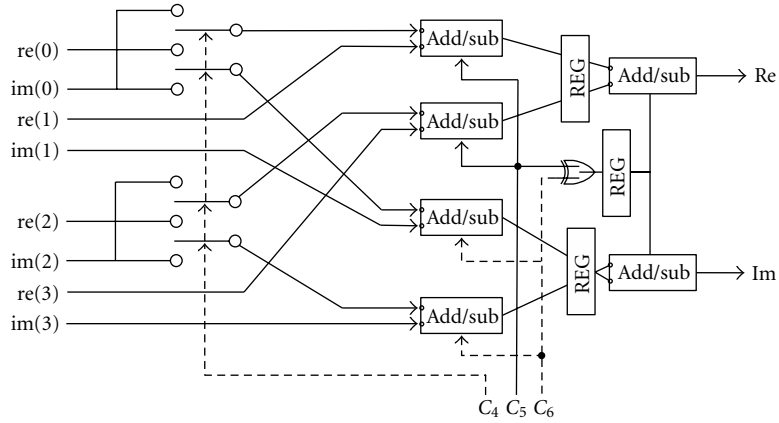


Figure 8: Adding pipeline registers to R4SDC butterfly element.

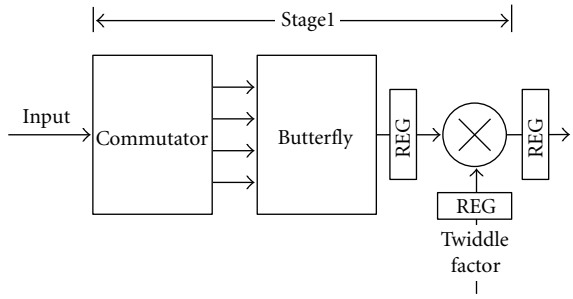


Figure 9: Adding pipeline registers between elements and stages.

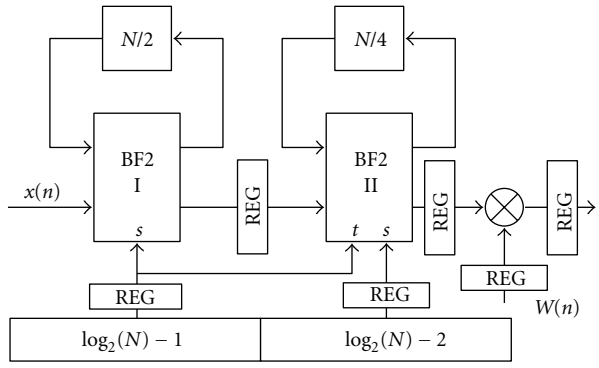


Figure 10: Adding pipeline registers for R2²SDF.

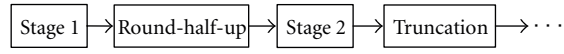


Figure 11: Balanced stages rounding.

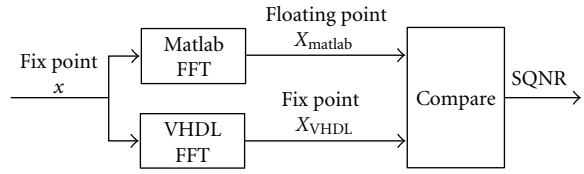
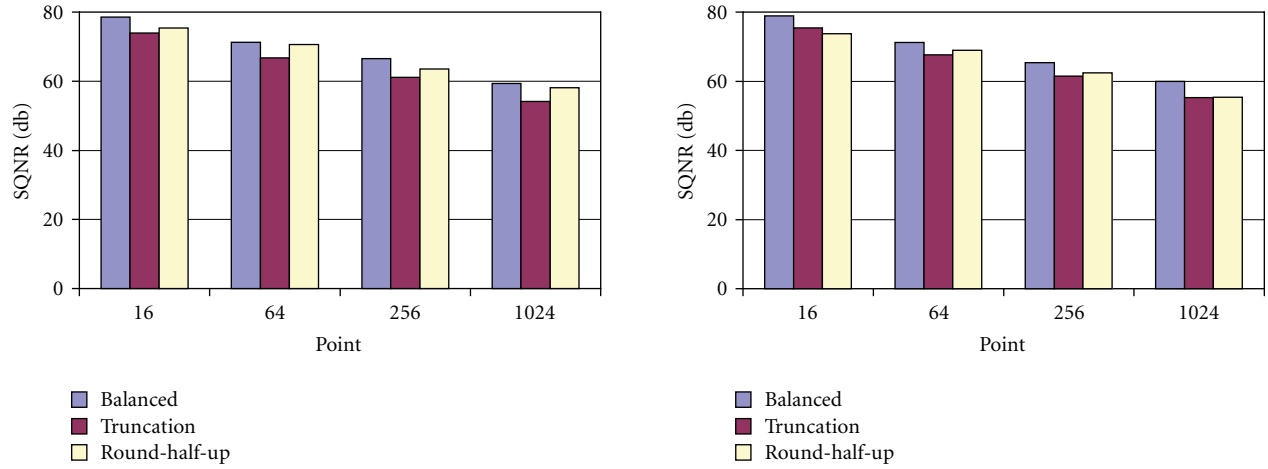


Figure 12: SQNR calculation.

Our R4SDC architecture was also superior to prior art, running at 94.2 MHz and using 7052 slices, a throughput per area ratio of 0.013 Msamples/s/slice.

Another point of reference is the Xilinx FFT IP core. For comparison sake, the IP core for Virtex-E is shown in the table. The Virtex-E core shows four times the latency (4096) in cycles due to its internal architecture. Its throughput per area ratio is also only 0.011 Msamples/s/slice. Note that all comparisons for throughput per area do not take into account block RAMs, though each of the designs had a similar number of required block RAMs. However, the Xilinx FFT DSP core could perform better with new Xtreme technology: on Virtex 4 device 4vsx25-10, 1024-point FFT



(a) R4SDC (b) R2^{2}SDF

Figure 13: Rounding effects on SQNR.

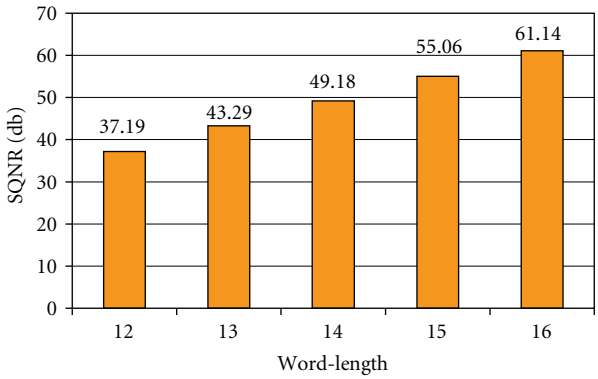


Figure 14: SQNR variation with different word lengths.

could be finished within 2.85 nanoseconds in best case, while cost 2141 Slices, 7 block RAMs, and 46 Xtreme DSP slices [13].

6. Conclusions

In this paper, optimized implementations of R4SDC and R2^{2}SDF pipeline FFT processors on Spartan-3, Virtex-4, and Virtex-E FPGAs are presented. The 16-bit 1024-point FFT with the R2^{2}SDF architecture had a maximum clock frequency of 95.2 MHz and used 2802 slices on the Spartan-3. The R4SDC ran at 123.8 MHz and used 4409 slices on the Spartan-3. On Virtex-4 device, the numbers became 235.6 MHz and 2256 slices for R2^{2}SDF and 219.2 MHz and 3064 slices for R2^{2}SDF, respectively. Different rounding schemes were analyzed and compared. SQNR analysis showed the balanced stages rounding scheme gave high SQNR with small overhead. The SQNR will gain around 6 dB with every bit increment of word length.

The R2^{2}SDF architecture outperformed the R4SDC architecture in terms of throughput per area, a measure of efficiency, for the 1024-point FFT. This is due to its simpler controller and compatibility with pipelining insertion. Both architectures have comparable maximum clock frequency and SQNR with the balanced stages rounding scheme.

References

[1] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall, Upper Saddle River, NJ, USA, 1975.

[2] E. H. Wold and A. M. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementation," *IEEE Transactions on Computers*, vol. 33, no. 5, pp. 414–426, 1984.

[3] G. Bi and E. V. Jones, "A pipelined FFT processor for word-sequential data," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 12, pp. 1982–1985, 1989.

[4] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)*, pp. 766–770, Honolulu, Hawaii, USA, April 1996.

[5] J.-Y. Oh and M.-S. Lim, "New radix-2 to the 4th power pipeline FFT processor," *IEICE Transactions on Electronics*, vol. E88-C, no. 8, pp. 1740–1746, 2005.

[6] T. Sansaloni, A. Pérez-Pascual, V. Torres, and J. Valls, "Efficient pipeline FFT processors for WLAN MIMO-OFDM systems," *Electronics Letters*, vol. 41, no. 19, pp. 1043–1044, 2005.

[7] S. Johansson, S. He, and P. Nilsson, "Wordlength optimization of a pipelined FFT processor," in *Proceedings of the 42nd Midwest Symposium on Circuits and Systems*, vol. 1, pp. 501–503, 1999.

[8] S. Sukhsawas and K. Benkrid, "A high-level implementation of a high performance pipeline FFT on Virtex-E FPGAs," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '04)*, pp. 229–232, February 2004.

[9] Xilinx, Inc., "High-Performance 1024-Point Complex FFT/IFFT V2.0," San Jose, Calif, USA, July 2000, http://www.xilinx.com/ipcenter.

[10] Sundance Multiprocessor Technology Ltd., 1024-Point Fixed Point FFT Processor, July 2008, http://www.sundance.com/web/files/productpage.asp?STRFilter=FC200.

[11] P. Kabal and B. Sayar, "Performance of fixed-point FFT's: rounding and scaling considerations," in *Proceedings of the*

IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '86), pp. 221–224, 1986.

[12] B. Zhou and D. Hwang, "Implementations and optimizations of pipeline FFTs on Xilinx FPGAs," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig '08)*, pp. 325–330, 2008.

[13] Xilinx, Inc., "Xilinx Fast Fourier Transform V3.2 Product Specification," San Jose, Calif, USA, January 2006.