An Evaluation of High-Throughput Scalable Radix-4 FFT Processor Architecture Using Fixed-Point Arithmetic

Tomotaka Kawabata and Hiroshi Tsutsui
Graduate School of Information Science and Technology, Hokkaido University

***Abstract*—In this paper, we propose a scalable and high-throughput pipeline FFT processor architecture and evaluate its variations. To achieve high-throughput with reasonable operating frequency, the proposed architecture utilizes radix-4, where four points are processed every clock cycle. Like IP core generators, our architecture can be reconfigured by changing the number of FFT stages to support various numbers of FFT points. Our architecture is based on fixed-point arithmetic to relieve the complexity but might be extended to support floating-point implementation to keep high dynamic ranges. The proposed architecture achieves four times the throughput of the operating frequency. For example, 492 M samples/sec of throughput can be achieved when the operating frequency is 123 MHz, which may be a reasonable performance for 5G OFDM implementation. In this case, the gate count of our 4K-point FFT is 443,419, excluding SRAMs for pipeline buffers.**

I. Introduction

In recent years, the demands for multimedia services including video streaming has increased in the communication field. To communicate such a large amount of information, communications over wide frequency bands are required. In wideband wireless digital communications, OFDM (orthogonal frequency division multiplexing) is widely used [1] due to its high efficiency and robustness to the multipath fading effects. The OFDM is used in a wide range of applications such as digital broadcasting and broadband mobile communications.

The fast Fourier transform (FFT) is one of the essential digital processing techniques, required for OFDM modulation and demodulation. Furthermore, the number of OFDM subcarriers is increasing due to the demand for high-speed, large-capacity communications, which is obvious from the wireless communication trend towards 5G systems, so there is a demand for high-throughput FFT processors with a large number of points. In addition, since the required number of FFT points varies along with communication standards and their applications, scalable FFT processor designs are highly required. Motivated by this, we propose a high-throughput and scalable FFT processor design using a radix-4 butterfly operation circuit with stage-by-stage pipeline operation. Like IP core generators, our architecture can be reconfigured changing the number of FFT stages to support various numbers of FFT points. Our architecture is based on fixed-point arithmetic to relieve the complexity. The proposed architecture achieves four times the throughput of the operating frequency. For example, 492 M samples/sec of throughput can be achieved when the operating frequency is 123 MHz, which may be a reasonable performance for 5G OFDM implementation. In this case, the gate count of our 4K-point FFT is 443,419, excluding SRAMs for pipeline buffers.

The rest of this paper is organized as follows. In Sec. II, we briefly describe a theoretical background with our radix-4 approach. In Sec. III, we present our proposed high-throughput scalable radix-4 FFT architecture. In Sec. IV, we show logic synthesis results and evaluate the proposed approach. Finally, Sec. V concludes this paper.

II. Theoretical background

This section describes the theoretical background of FFT and twiddle factor (TF) processing for hardware implementations focusing on radix-4 FFT. The discrete Fourier transform (DFT) of $f(x)$, $x = 0, 1, 2, \ldots, N-1$ is expressed by

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) W^{ux}, \quad W = \exp\left(-j2\pi \frac{1}{N}\right) \tag{1}$$

where W^{ux} is called *twiddle factors* (TFs). The input for DFT, $f(x)$, is a time domain signal, while the output of DFT, $F(u)$, is the frequency domain representation of $f(x)$. When the DFT is directly calculated using the above formula, N^2 multiplications are required resulting a lot of processing time, that is, $O(N^2)$. When N is a power of 2, FFT can be applied to reduce the computational cost from $O(N^2)$ to $O(N \log N)$. In this paper, we utilize the Cooley-Tukey algorithm [2], which is the most common FFT algorithm. This algorithm utilizes a divide-and-conquer approach, where the processing is divided into stages depending on the number of FFT points. At each stage, inputs are multiplied with TFs, and so-called butterfly operations are repeatedly applied.

Here, we describe the butterfly operation. In the case of the smallest operation unit, that is, radix-2, the butterfly operation is expressed by,

$$\begin{cases} x'_0 = x_0 + x_1 W^n, \\ x'_1 = x_0 - x_1 W^n, \end{cases} \tag{2}$$

where n is an integer, x_0 and x_1 are inputs, and x'_0 and x'_1 are outputs. The flow of this operation is shown in Fig. 1. It is called a butterfly because of the shape of the symmetrical addition and subtraction. TF W^n rotates the complex value x_1

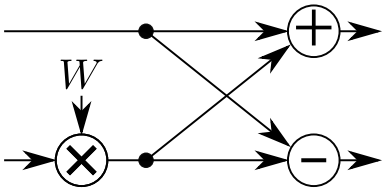

Fig. 1. Butterfly of radix-2 FFT

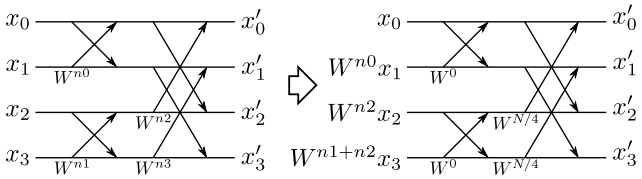

Fig. 2. Separation of TF multipliers and radix-4 butterfly core

with the angle of n/N. When n equals 0, 1, and $N/2$, W^n has special values, that is, $W^0 = W^N = 1$, $W^{N/2} = -1$. The butterfly operation utilizes this property.

The FFT repeats the butterfly operation changing the order of processing data for each stage. When the radix number is given by a, which has generally a power of 2, the number of stages is $\log_a N$, and butterfly operations are performed N/a times for each stage. When the FFT algorithm is decomposed with a higher radix, the number of stages decreases. In this case, however, the structure of butterfly operations may become complicated. Therefore, the radix number of 2 or 4 is generally used [3]. Since we are aiming at high-throughput FFT processors in this paper, we use radix-4. In the case of radix-4, assuming that four inputs can be processed in one cycle, the throughput can be four times of the operating frequency. In this sense, radix-4 can achieve the throughput twice that of radix-2, when the operating frequencies are the same. Therefore, by using radix-4, a high-throughput can be realized while keeping a low operating frequency, resulting in a low power consumption. Besides, in the case of the radix-4 butterfly operation, we can divide the operation into (a) the multiplication part of the input data and the TFs, and (b) the addition/subtraction part [4]. This approach can reduce the number of multiplications compared to simple radix-2 implementations, resulting an efficient hardware implementation.

Here, we describe the radix-4 butterfly operation and its design. The number of multiplications for an N-point FFT is $\frac{N}{2}\log_2 N$. The multiplication requires a lot of computational cost. Moreover, since the TFs also varies depending on the data input order, the implementation becomes complicated and the throughput may decrease. Therefore, we change the radix-4 butterfly operations.

Generally, the radix-4 butterfly operation is given by,

$$\begin{cases} x'_0 = x_0 + W^{n0}x_1 + W^{n2}(x_2 + W^{n1}x_3), \\ x'_1 = x_0 - W^{n0}x_1 + W^{n3}(x_2 - W^{n1}x_3), \\ x'_2 = x_0 + W^{n0}x_1 - W^{n2}(x_2 - W^{n1}x_3), \\ x'_3 = x_0 - W^{n0}x_1 - W^{n3}(x_2 + W^{n1}x_3), \end{cases} \tag{3}$$

or

$$\begin{cases} x'_0 = x_0 + W^{n0}x_1 + W^{n2}x_2 + W^{n1+n2}x_3, \\ x'_1 = x_0 - W^{n0}x_1 + W^{n3}x_2 - W^{n1+n3}x_3, \\ x'_2 = x_0 + W^{n0}x_1 - W^{n2}x_2 - W^{n1+n2}x_3, \\ x'_3 = x_0 - W^{n0}x_1 - W^{n3}x_2 + W^{n1+n3}x_3, \end{cases} \tag{4}$$

where $n0$, $n1$, $n2$, and $n3$ are integers, x_0, x_1, x_2, and x_3 are inputs, and x'_0, x'_1, x'_2, and x'_3 are outputs. The TF values, W^{n0}, W^{n1}, W^{n2}, and W^{n3}, depend on the number of times the butterfly operations are called. Therefore, we separate this variable part, and we call the remaining fixed part a *butterfly core*. The same calculation can be realized by multiplying the input values with the corresponding TFs before the butterfly core [5]. Assuming $n3 = n2 + N/4$, (4) can be written as

$$\begin{cases} x'_0 = y_0 + y_1 + y_2 + y_3, \\ x'_1 = y_0 - y_1 - jy_2 + jy_3, \\ x'_2 = y_0 + y_1 - y_2 - y_3, \\ x'_3 = y_0 - y_1 + jy_2 - jy_3, \end{cases} \tag{5}$$

$$\begin{cases} y_0 = x_0, \\ y_1 = W^{n0}x_1, \\ y_2 = W^{n2}x_2, \\ y_3 = W^{n1+n2}x_3, \end{cases} \tag{6}$$

where $W^{N/4} = -j$ is used. As shown in Fig. 2, in the case of (3), four multiplications are required. On the other hand, in the case of (5) and (6), using three multiplications in (6), the radix-4 butterfly calculation is realized owing to $n3 = n2 + N/4$, and the variable TF multiplication part can be successfully separated from the butterfly core.

III. High-Throughput Scalable Radix-4 FFT Processor Architecture

Based on the discussions described in the previous section, we designed a high-throughput and scalable Radix-4 FFT processor. The current design utilizes fixed-point arithmetic to reduce complexity. We believe that this architecture can be extended to use floating-point arithmetic when a high dynamic range is required. The designed FFT processor consists of $N_{\text{stages}} = \log_4 N$ stages. The number of stages may change depending on the number of FFT points the implementation should support. We assume that the designed FFT processor will be used as a semi-conductor intellectual property (IP) core, which can generate N-point FFT processor implementation when N is given. In other words, it is possible to support any number of FFT points by simply changing the number of stages and memory capacity.

A similar approach can be found in [6], where radix-2^k multi-path delay commutator (MDC) architectures are proposed. In [6], FIFOs are used between stages for data shuffling while, in our architecture, SRAMs are used as pipeline buffers.

The designed FFT processor uses the radix-4 butterfly operation circuit repeatedly in each stage. Figure 3 shows the

block diagram of one stage. The input and output SRAMs store all input/output data for the stage, whose number is N. Since the proposed architecture is based on radix-4, each SRAM has 4 banks so that 4 point data can be fed to the calculation part simultaneously. After reading 4 point data from the input SRAM, the radix-4 butterfly operation is performed, whose output is stored into the output SRAM, which is regarded as the input SRAM of the next stage. As mentioned before, the radix-4 butterfly operation is divided to a twiddle factor multiplier (multi) and a radix-4 butterfly operation circuit (butterfly core). Note that the butterfly core does not include any multiplication. Since the TF values are different for each calculation, so the appropriate TF values are read from ROMs. The proposed architecture utilizes FFT-stage based pipeline operation as shown in Fig. 4, to achieve a high-throughput. Fig. 4 assumes $N = 4096$ and $N_{\text{stages}} = 6$. In this case, the output SRAM in Fig. 3 cannot be used as the input SRAM for the next stage simultaneously. Therefore, we utilize a double buffer configuration shown in Fig. 4. In this case the total number of SRAMs is $N_{\text{SRAM}} = 2(N_{\text{stages}} + 1)$.

A. SRAM Access

In order to operate at the throughput of 4 sample/cycle, each SRAM has 4 banks. The FFT requires rearrangement of samples for each stage. The control module controls the data rearrangement utilizing unpack/pack operations. To realize this, we pack 4 samples into one word. Therefore, the depth of each SRAM bank is

$$\frac{N \text{ samples}}{4 \text{ banks} \times 4 \text{ samples/word}} \text{ (words/bank)}. \tag{7}$$

The outputs from the butterfly core are 4 samples in one cycle. These 4 outputs are packed into one word, which will be stored into one corresponding SRAM bank. The set of each four output words are stored into the 1st, 2nd, 3rd, and 4th SRAM banks, respectively. Therefore, in the write/output side, just one bank from four banks is selected to be written in each cycle. As for the inputs, four words, which include 16 samples, are read from four SRAM banks in one cycle. Each sample in each word is fed into the multi/(butterfly core) module in each cycle. In other words, four samples each of which is a part of a different word are fed into the processing butterfly core in each cycle. Unused 12 samples are not used. By utilizing such approach, we unpack the word stored in the input SRAM, and pack four samples as the input to the multi/(butterfly core) module. Therefore, in the read/input side, four banks are accessed in one cycle and every cycle.

B. Twiddle factor multiplier

Fig. 5 shows the twiddle factor multiplier (multi) module where the inputs are multiplied with TFs. The TFs are stored in ROMs whose depth depends on the number of FFT points. Considering the symmetric property of trigonometric functions, we can reduce the number of TF values to be stored in the ROMs to $N/4$ TF values. Since it is necessary to read the corresponding TF values, we use counters (`cnt` in the figure) to control the addresses for the ROMs. In our design,

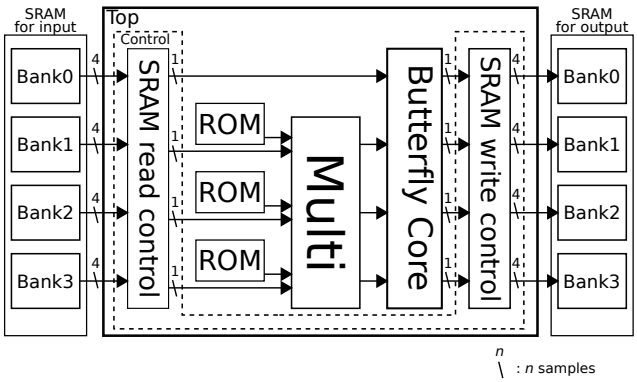

Fig. 3. Block diagram of one stage

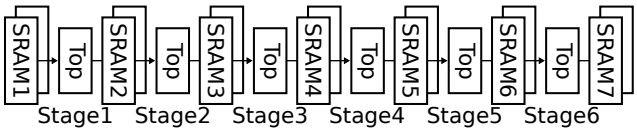

Fig. 4. Pipeline structure using a double buffer scheme ($N = 4096$, 6 stages)

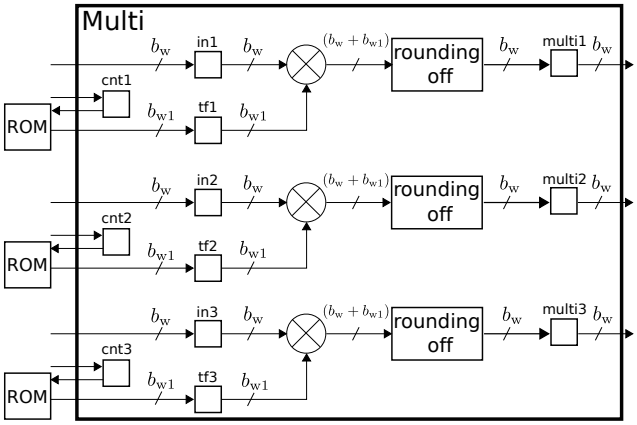

Fig. 5. The twiddle factor multiplier (multi) module where the inputs are multiplied with TFs. The bit widths are parameterized as b_{w} bits for samples and b_{w1} bits for TF values.

the bit widths are parameterized as b_{w} bits for samples and b_{w1} bits for TF values. In the current design, the resulted values of multiplication are rounded again to the same bit width to the sample bit width b_{w} as shown in Fig. 5.

IV. Circuit Area Evaluation

Using a 0.13 μm CMOS cell library, we performed a logic synthesis with an operating frequency of 123 MHz. As for the bit width, we use a configuration where the number of bits for both integer and fractional parts is 12 bits, resulting that one complex value is represented by 48 bits. The ROMs were implemented as combinational circuits. The number of TFs stored in one ROM is 1/4 of the number of FFT points N. The total number of butterfly cores is N_{stages}, the total number of ROMs is $3 \times N_{\text{stages}}$, and the total bit number of SRAMs

TABLE I
TOTAL NUMBER OF GATES OF EACH MODULE

FFT points (N)	64	256	1,024	4,096
The number of stages (N_{stages})	3	4	5	6
Top	217,460	293,036	370,031	443,419
Control	7,792	10,779	13,862	17,024
Multi	186,520	245,590	309,535	369,422
Core	2,792×3	2,792×4	2,792×5	2,792×6
ROMs	124×3×3	399×3×4	1,222×3×5	4,045×3×6
SRAMs (number of total bits)	48×64×8 24,576	48×256×10 122,880	48×1,024×12 589,824	48×4,096×14 2,752,512

The gate counts are NAND gate equivalents. The gate counts show the total number of gates for each module including all stages. The top module includes all modules.

is $48 \times N \times N_{\mathrm{SRAM}}$. Table I shows the logic synthesis result when the FFT points are 64, 256, 1,024, and 4,096. Note that SRAMs are not included as gate counts. The gate counts in the tables show the total number of gates for each module, including all stages. Note that the top module includes all modules. Since the butterfly core has a simple structure by separating the TF multiplier, the size of butterfly core is relatively small, that is, 2,792 gates/stage. The control modules which controls SRAM access are not so large. The TF multiplier module occupies a large percentage of the whole area because the multiplications of the inputs and the TFs is performed. Note that the size of this TF multiplier modules can be controlled by changing the bit widths b_{w} and b_{w1}. The throughput can be estimated as 4 sample/cycle $\times$ 123 MHz = 492 M sample/sec regardless of the number of FFT points, which is a significantly high-throughput. Considering the FFT implementation for 5G OFDM which requires 4,096-point FFT, we believe that this throughput is reasonable.

V. CONCLUSION

We designed a high-throughput and scalable FFT processor and evaluated its hardware scale. By synthesizing using a CMOS cell library, we demonstrated the proposed FFT processors with different numbers of FFT points. Owing to our scalable design, the circuit scale increases in proportion to the number of FFT points, that is, the number of stages. The current implementations show that the gate count of our 4K-point FFT is 443,419 excluding SRAMs for pipeline buffers. Most parts are by the TF multiplier, whose size can be controlled by changing the bit widths to represent samples. Further optimizations such as to realize the FFT processor design optimized for a given target SNR constraint, and floating-point designs to support a given high dynamic range remain as future works.

In [6], FIFOs are used between stages for data shuffling while, in our architecture, SRAMs are used as pipeline buffers. The approach in [6] is scalable in terms of both throughput and number of stages. Also, the architectures in [6] achieve lower latency compared to our architecture. Further optimizations referring MDC approaches to archive a low latency and efficient implementation also remain as future works.

ACKNOWLEDGMENTS

The authors would like to thank the GI-CoRE GSB, Hokkaido University for fruitful discussions. This work is supported in parts by the Ministry of Internal Affairs and Communications for SCOPE Program (185001003). This work is also supported through the activities of VDEC, the University of Tokyo, in collaboration with Synopsys, Inc., Cadence Design Systems, Inc., and Mentor Graphics, Inc.

REFERENCES

[1] B. Kang and J. Kim, "Low complexity multi-point 4-channel FFT processor for IEEE 802.11n MIMO-OFDM WLAN system," in *Proc. International Conference on Green and Ubiquitous Technology*, 2012, pp. 94–97.

[2] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965. [Online]. Available: `http://www.jstor.org/stable/2003354`

[3] Z. A. Abbas, N. B. Sulaiman, N. A. M. Yunus, W. Z. Wan Hasan, and M. K. Ahmed, "An FPGA implementation and performance analysis between radix-2 and radix-4 of 4096 point FFT," in *Proc. International Conference on Smart Instrumentation, Measurement and Application (ICSIMA)*, 2018, pp. 1–4.

[4] J. Takala and K. Punkka, "Butterfly unit supporting radix-4 and radix-2 FFT," in *Proc. International TICSP Workshop on Spectral Methods and Multirate Signal Processing (SMMSP)*, vol. 30, 2005, pp. 47–54.

[5] R. Neuenfeld, M. Fonseca, and E. Costa, "Design of optimized radix-2 and radix-4 butterflies from FFT with decimation in time," in *Proc. Latin American Symposium on Circuits Systems (LASCAS)*, 2016, pp. 171–174.

[6] M. Garrido, J. Grajal, M. A. Sanchez, and O. Gustafsson, "Pipelined radix-2^k feedforward FFT architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 1, pp. 23–32, 2013.