2016 IEEE 46th International Symposium on Multiple-Valued Logic

An FFT Circuit Using Nested RNS in a Digital Spectrometer for a Radio Telescope

Hiroki Nakahara, Ehime University, Japan
Tsutomu Sasao, Meiji University, Japan
Hiroyuki Nakanishi, Kagoshima University, Japan
Kazumasa Iwai, National Institute of Information and Communications Technology, NICT, Japan
Tohru Nagao, Ehime University, Japan
Naoya Ogawa, Ehime University, Japan

***Abstract*—A radio telescope analyzes radio frequency (RF) signal received from celestial objects. It consists of an antenna, a receiver, and a spectrometer. The spectrometer converts the time domain signal into the frequency domain signal by an FFT operation. This paper proposes an FFT circuit based on nested residue number system (NRNS). Since the FFT operation is the most computationally intensive part, parallel implementation is necessary to realize a high-speed FFT. We used an FPGA to implement the circuit. The FPGA consists of look-up tables (LUTs) and block RAMs (BRAMs). For direct parallel FFT realization using an existing FPGA library, the number of LUTs for the complex multipliers is the bottleneck. To reduce the number of LUTs in an FPGA, we increase the dynamic range stage by stage. In this case, NRNS2NRNS converters that convert smaller dynamic range to larger dynamic range are necessary. We implemented the proposed NRNS FFT on the Xilinx Corp. Virtex 7 FPGA. Compared with a conventional binary FFT, although the number of block RAMs (BRAMs) was increased by 20.0-156.5%, in the RNS FFT, the number of LUTs was decreased by 42.4-47.8% and the maximum clock frequency was increased by 9.3-41.7%. With this technique, we successfully implemented an FFT that satisfied the required size and speed specifications on an available FPGA, since the excessive number of LUTs was the bottleneck of the binary FFT.**

I. Introduction

A. Digital Spectrometer for a Radio Telescope

A radio telescope analyzes the radio frequency (RF) signal received from celestial objects. Fig. 1 shows the operations in a typical radio telescope. First, the antenna receives the RF signal coming from the celestial objects. Second, the feed horn sends the electromagnetic wave to the receiver through the waveguide. Third, the receiver transforms it to the intermediate frequency (IF) signal by using the amplifier and mixer. Finally, **the digital spectrometer** converts the analog signal to the power spectrum. Fig. 2 shows the digital spectrometer. First, the analog-to-digital converter (ADC) converts the IF analog signal into a digital signal. Then, the Fourier transform unit performs **the fast Fourier transform (FFT)** [3] to obtain the spectrum. Next, the magnitude unit calculates the power spectrum. Finally, to increase the dynamic range of the obtained power spectrum, they are accumulated. Since the FPGAs have a dedicated I/O to connect to the high-speed ADCs and can be reconfigured for a desired specification, they are widely used in the digital spectrometer [1], [2], [6], [8], [9], [12].

We are developing the next generation radio telescope

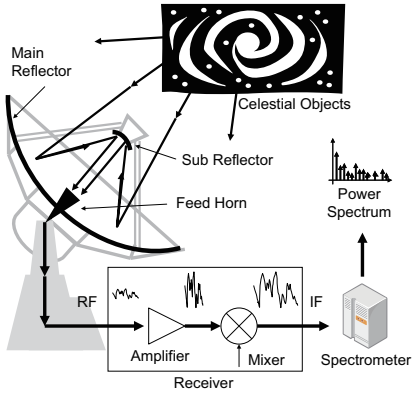

Fig. 1. Radio Telescope.

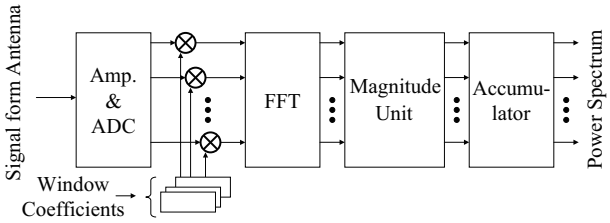

Fig. 2. Digital spectrometer

which is called SKA (square kilometers array). It requires wide band (up to 24 GHz) FFT operations [14]. Since the available FPGA operates at only up to a megahertz order, more parallel operations are necessary. The FPGA consists of look-up tables (LUTs) and block RAMs (BRAMs). As for a direct parallel FFT realization using the existing FPGA library [18], the number of LUTs for the complex multipliers becomes the bottleneck [8], [9]. Thus, the reduction of the number of LUTs is essential. To implement an n-bit parallel multiplier for the FFT circuit, LUTs of $O(n \cdot 2^n)$ is necessary. In this paper, to reduce the number of LUTs, we decompose large parallel multipliers into smaller ones by the residue number system (RNS).

B. Contributions of the Paper

We use a residue number system (RNS) [15], [19] to decompose n-input multipliers into smaller ones. With this

0195-623X/16 $31.00 © 2016 IEEE
DOI 10.1109/ISMVL.2016.35

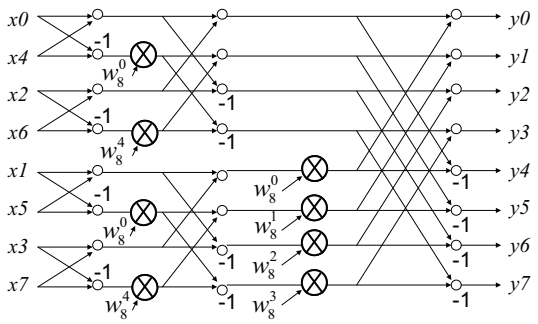

Fig. 3. Signal flow graph.

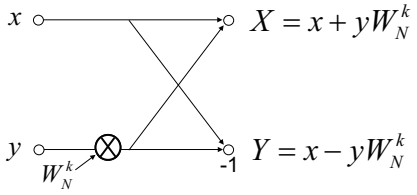

Fig. 4. Radix-2 butterfly operator.

technique, we reduce the LUT count to implement the FFT. The RNS decomposes an integer into a set of L integers by residues of moduli set. In a conventional RNS, since no pair of moduli have a common factor with any other, the resulting multipliers have different sizes. This means that RNS-based design requires LUTs with different sizes. To solve this problem, we use **the nested RNS (NRNS)** [11], which recursively decompose the numbers in RNS. By using the NRNS, we decompose the multiplier on the FFT circuit into smaller ones so that they can be efficiently implemented by LUT of an FPGA. As a result, the proposed method can realize more FFTs in parallel than the existing FPGA library. Additionally, we compare the NRNS with the conventional RNS for different numbers of inputs n for the arithmetic circuits. Finally, we show the condition that the NRNS realization is smaller than the RNS realization.

C. Organization of the Paper

The rest of the paper is organized as follows: Chapter 2 shows the pipelined binary FFT circuit; Chapter 3 introduces the conventional residue number system (RNS); Chapter 4 shows the FFT circuit based on the RNS (RNS FFT); Chapter 5 proposes the FFT based on the Nested RNS (NRNS FFT); Chapter 6 compares the NRNS FFT with the RNS one; Chapter 7 shows the experimental results; and Chapter 8 concludes the paper.

II. Binary FFT

A. Fast Fourier Transform (FFT)

Let $(x_0, x_1, \ldots, x_{N-1})$ be an input consisting of N complex numbers. **The discrete Fourier Transform (N point DFT)** is $(c_0, c_1, \ldots, c_{N-1})$, where

$$c_k = \sum_{j=0}^{N-1} a_j w_N^{jk}, \quad (1)$$

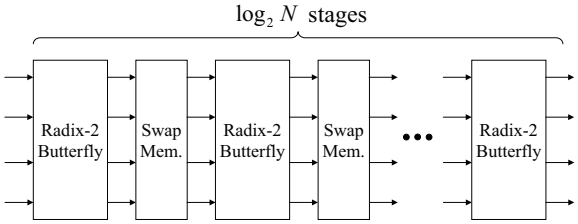

Fig. 5. Pipelined radix-2 binary FFT.

and $w_N^{jk} = exp(-2\pi i \frac{jk}{N})$ is a **twiddle factor**. A time complexity of a direct computation for Expr. (1) is $O(N^2)$. By applying decompositions to the N point DFT $s = \lceil \log_r N \rceil$ times recursively, we have a **Cooley-Tukey Fast Fourier Transform (N point FFT)** [3]. Let $s = \lceil log_r N \rceil$ be the number of stages, and r be the radix of the FFT. In this paper, we assume that $r = 2$. Fig. 3 shows a signal flow graph obtained by the FFT algorithm, where $N = 8$ and $r = 2$.

B. Pipeline Radix-2 Binary FFT

As shown in Fig. 3, different stages handle points with different distances. By applying an index swap operation replacing indices between adjacent stages, we can adjust the points of the inputs for the butterfly operations. **The swap memory** performs an index swap operation. Let w be a **precision** of the FFT. Then, the amount of memory for each swap is wN, and the total amount of memory for the swap memory is $wN\lceil log_r N \rceil$. Fig. 4 shows a **radix-2 butterfly operator** for $r = 2$, which consists of two complex multipliers. Fig. 5 shows a pipelined radix-2 FFT [4], which allows continuous data processing. Unfortunately, the radix-2 binary FFT requires very large multipliers, since the dynamic range of the latter stages are very large.

III. Residue Number System (RNS)

A **residue number system (RNS)** [19], [15] is defined by a set of L integer constants as follows:

$$\langle m_1, m_2, \ldots, m_L \rangle,$$

where no pair of moduli have a common factor with any other. An arbitrary integer Z can be uniquely represented by the RNS as a tuple of L integers as follows:

$$(z_1, z_2, \ldots, z_L),$$

where $z_i \equiv Z \ (mod\ m_i)$.

$M = \prod_{i=1}^{L} m_i$ is a **dynamic range** of the RNS. In the RNS, the addition, the subtraction, and the multiplication are performed in digit-wise. Let X and Y be integers, x_i and y_i be integers in the RNS defined by m_i $(1 \leq i \leq L)$. Let $\circ$ denote $+$ (addition), $-$ (subtraction), or $*$ (multiplication). Then, $Z = X \circ Y$ can be represented as

$$Z = (z_1, z_2, \ldots, z_L),$$

where $z_i \equiv (x_i \circ y_i) \ (mod\ m_i)$. Note that, the division is not included in the operations.

Example 3.1: Let $\langle m_1, m_2, m_3 \rangle = \langle 3, 4, 5 \rangle$ be the moduli set. Consider the multiplication $X \times Y$, where $X = 8$ and

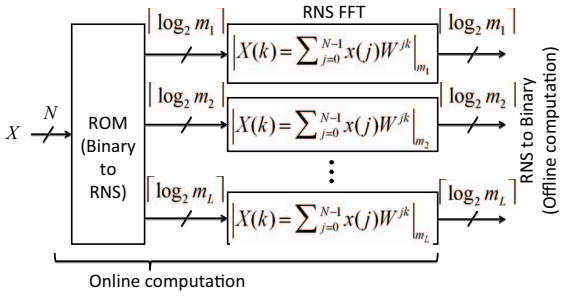

Fig. 6. Radix-2 RNS FFT [16].

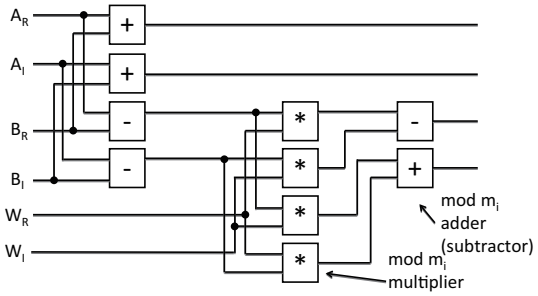

Fig. 7. Modulo m_i RNS butterfly operator.

$Y = 2$. Since $X \times Y = 16$, the product is represented by $(1, 0, 1)$ in the RNS. X and Y are represented by $(2, 0, 3)$ and $(2, 2, 2)$ in the RNS, respectively. In this case, $X \times Y$ in the RNS is computed as follows:

$$\begin{aligned} X \times Y &\equiv (4 \ mod \ 3, 0 \ mod \ 4, 6 \ mod \ 5) \\ &\equiv (1, 0, 1). \end{aligned}$$

■

In the RNS, the arithmetic operation is performed in digit-wise. This means that we can decompose large multipliers into smaller ones. In this way, we can reduce the sizes of LUTs for the FFT.

IV. Radix-2 RNS FFT Circuit

Expr. (1) shows that the FFT operation consists of the addition, the subtraction, and the multiplication. By applying the RNS to the FFT, we have the **FFT circuit based on the RNS (RNS FFT)** shown in Fig. 6. In this circuit, first, the binary input signal is converted into the RNS by read only memories (ROMs). Typically, the input signals from analog-digital converters (ADCs) are 8-14 bits. The binary to RNS converter can be implemented by 18Kb BRAMs on the FPGA. Next, the RNS FFT circuit computes each signal in the digit-wise manner. In this paper, we assume that the conversion from the RNS to the binary is done off-line.

Fig. 7 shows the **modulo m_i RNS butterfly**, which is derived from the binary butterfly operator shown in Fig. 4. In Fig. 7, $A = (A_R, A_I)$ and $B = (B_R, B_I)$ denote the complex inputs, $W = (W_R, W_I)$ denotes the complex twiddle factor, R denotes the real part, and I denotes the imaginary part. The m_i RNS butterfly module can be realized by small LUTs [16]. Let m_i be the modulo in the RNS. Then, the total amount of

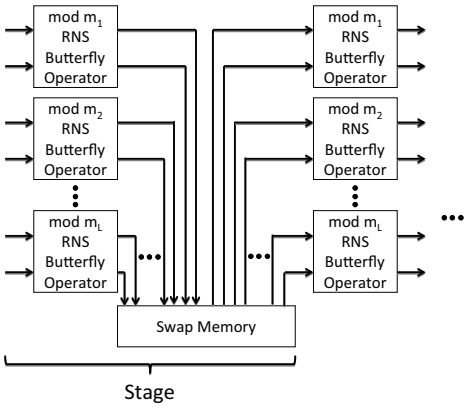

Fig. 8. Swap memory on the radix-2 RNS FFT.

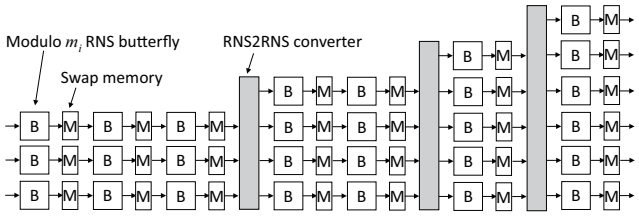

Fig. 9. RNS FFT using RNS2RNS converters.

memory for the modulo m_i butterfly is $10 \times (m_i)^2 \lceil log_2 m_i \rceil$ bits, since the butterfly operator uses 10 arithmetic circuits. Fig. 7 shows that the N points RNS FFT requires memories with $\sum_{i=1}^{L} 10 \times (m_i)^2 \lceil log_2 m_i \rceil \times log_2 N$ bits. In other words, the number of LUTs is $O((m_i)^2 \cdot (log \ m_i) \cdot (log N))$. Therefore, we can decreases the number of LUTs by decreasing m_i .

Fig. 8 shows the swap memory on the radix-2 RNS FFT. As shown in Fig. 6, the RNS FFT consists of L moduli FFTs. Swap values for L butterfly operators are stored in the swap memory. When $r = 2$, the RNS butterfly operator swaps $\frac{N}{2}$ signals. Also, each RNS butterfly operator produces $\lceil log_2 m_i \rceil$ bits. Thus, the amount of swap memory Mem for each stage is

$$Mem = \frac{N}{2} \times (\sum_{i=1}^{L} \lceil log_2 m_i \rceil).$$

Since the number of stages is $\lceil log_2 N \rceil$, the total amount of swap memories is $Mem \lceil log_2 N \rceil$ bits.

V. FFT Circuit Based on Nested RNS

A. RNS FFT using RNS2RNS Converters

A direct realization of the RNS FFT shown in Fig. 6 requires an excessive number of LUTs, since the dynamic range is too large for the first half stages of the RNS butterflies. In this paper, we increase the dynamic range stage by stage. We insert **RNS2RNS converters** [10] that converts a small dynamic range to a large dynamic range. Fig. 9 shows the RNS FFT inserted RNS2RNS converters. In this circuit, since large moduli are removed, the number of LUTs is reduced. However, it requires additional RNS2RNS converters. We

(Bin)	m_1	m_2	m_1	m_2	m_3
0	0	0	0	0	0
1	1	1	1	1	1
2	0	2	0	2	2
3	1	0	1	0	3
4	0	1	0	1	4
5	1	2	1	2	0

Fig. 10. Example of an RNS2RNS converter.

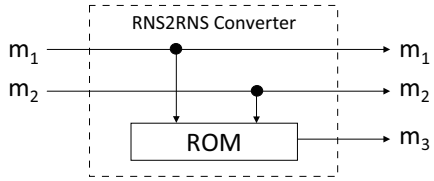

Fig. 11. RNS2RNS converter.

decompose the RNS2RNS converter by applying the functional decomposition.

Fig. 10 shows the function table for the RNS2RNS converter which converts $(m_1, m_2) = (2, 3)$ to $(m_1, m_2, m_3) = (2, 3, 5)$. Although we can use an arbitrary moduli set in the RNS2RNS converter, to reduce the amount of hardware, we use $g(m_1, m_2, \ldots, m_L) = (m_1, m_2, \ldots, m_L, m_{L+1})$ as the RNS2RNS converter. In this case, as shown in Fig. 11, the RNS2RNS converter requires only the circuit for $g'(m_1, m_2, \ldots, m_L) = m_{L+1}$. Let $M = \prod_{i=1}^{L} m_i$ be the dynamic range. A single-memory realization of the RNS2RNS converter requires a memory with $M\lceil log_2 m_{L+1} \rceil$ bits. In our implementation, as shown in Fig. 12, we decompose the RNS2RNS converter into the **RNS2Bin converter** and the **Bin2RNS converter**. Let m_{L+1} be the modulo in the Bin2RNS converter, then the column multiplicity of the decomposition is at most m_{L+1}. Thus, we can reduce the necessary number of LUTs by applying the functional decomposition [13]. In a similar manner, the RNS2Bin converter representing the **mod-EVMDD: Modulo Edge-Valued MDD**, which is a combination of the multi-valued decision diagram (MDD) [5] and the edge-valued decision diagram (EVDD) [7], can be reduced by the functional decomposition [10].

B. Nested RNS (NRNS)

Since moduli set of RNS consists of pairwise prime numbers, we need LUTs with different sizes. Assume that an integer Z is represented by $Z = (z_1, z_2, \ldots, z_L)$ in the RNS, and z_i is represented by the RNS recursively. Then, we have **the nested RNS (NRNS)** [11] which is defined by a set of j moduli $\langle m_{i1}, m_{i2}, \ldots, m_{ij} \rangle$, where no pair of moduli have a common factor with any other. In the NRNS, z_i can be uniquely represented as a tuple of j integers as follows:

$$Z \quad = \quad \big(z_1, z_2, \ldots, (z_{i1}, z_{i2}, \ldots, z_{ij})_i, \ldots, z_L\big).$$

In the NRNS, we describe the original moduli m_i in the subscript of the right parenthesis. Since the NRNS can be applied to each element in the RNS recursively, moduli can be

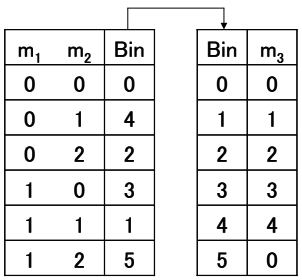

Fig. 12. Decomposition of the RNS2RNS converter.

decomposed into smaller sizes. In this case, we must consider the dynamic range. Let X and Y be integers represented by $X = (x_1, x_2, \ldots, x_L)$ and $Y = (y_1, y_2, \ldots, y_L)$ in the RNS, respectively. By applying the NRNS to x_i and y_i, we have $X = \big(x_1, x_2, \ldots, (x_{i1}, x_{i2}, \ldots, x_{ij})_{m_i}, \ldots, x_L\big)$ and $Y = \big(y_1, y_2, \ldots, (y_{i1}, y_{i2}, \ldots, y_{ij})_{m_i}, \ldots, y_L\big)$. Since the dynamic range for the NRNS is $M_{m_i} = \prod_{l=1}^{j} m_{il}$, the relation $M_{m_i} \geq x_i \circ y_i$ must be satisfied. By rewriting the output of the Bin2RNS converter z_i shown in Fig. 12 to $(z_{i1}, z_{i2}, \ldots, z_{ij})_i$, we have the Bin2NRNS converter.

Example 5.2: Consider that the RNS $\langle 3, 4, 5 \rangle$ is represented by the moduli set $\big\langle 3, 4, \langle 3, 4 \rangle_5 \big\rangle$ by the NRNS. Let $X = 17$ and $Y = 3$. In the NRNS, X and Y are represented by $X \equiv (2, 1, (2, 2)_5)$ and $Y \equiv (0, 3, (0, 3)_5)$, respectively. Thus, $X + Y$ in the NRNS is computed by

$$\begin{aligned} X + Y \quad &\equiv \quad (2 \ mod \ 3, 4 \ mod \ 4, (2 \ mod \ 3, 5 \ mod \ 4)_5) \\ &\equiv \quad (2, 0, (2, 1)_5). \end{aligned}$$

By converting $(2, 1)_5$ in the RNS $\langle 3, 4 \rangle_5$ to the binary number, we have 5. Note that, $5 \ (mod \ 5) = 0$. Then, we have

$$(2, 0, (2, 1)_5) \quad = \quad (2, 0, 5) \equiv (2, 0, 0).$$

By converting $(2, 0, 0)$ in the RNS $\langle 3, 4, 5 \rangle$ into the binary number, we have 20, which is equal to the result of $17 + 3$. ■

VI. Comparison of NRNS-Based Arithmetic Circuits with RNS-Based ones

Let $\langle m_1, m_2, \ldots, m_L \rangle$ be the moduli set for the RNS. Suppose that the moduli set $\langle m_1, m_2, \ldots, m_L, m_{L+1} \rangle$ where the RNS2RNS converter adds a new modulo m_{L+1}. By applying the NRNS to m_{L+1}, we have the moduli set for the NRNS as $\big\langle m_1, m_2, \ldots, m_L, \langle m_{L+1,1}, m_{L+1,2}, \ldots, m_{L+1,j}, \rangle \big\rangle$. In this section, we analyze the number of LUTs with respect to m_{L+1}. First, we estimate the number of LUTs. Let L_{RNS} be the necessary number of LUTs to realize an arithmetic circuit [1] based on the RNS, and L_{NRNS} be that based on the NRNS. Note that, in this paper, since we assume that the RNS2RNS converter is realized by BRAMs instead of LUTs, we do not count the number of LUTs for each converter. Also, we assume a moduli set consisting of prime numbers. The number of LUTs for the RNS is estimated by

$$L_{RNS} \quad = \quad \lceil k 2^{2k} / L_{FPGA} \rceil, \qquad (2)$$

[1]The arithmetic circuit has $2k$ inputs and k outputs, where $k = \lceil log_2 m_{L+1} \rceil$.

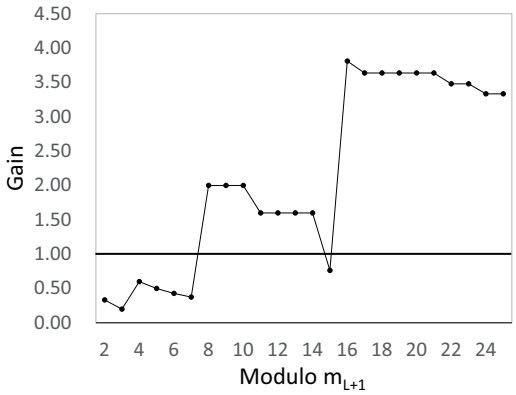

Fig. 13. Gain ($L_{FPGA} = 64$).

TABLE I. SYNTHESIS OPTIONS FOR THE XILINX INC. BINARY FFT.

Option	Parameter
Implementation	Pipelined, Streaming I/O
Data Format	Fixed Point
Input Data Width	8bit
Phase Factor Width	18bit
Scaling Options	Unscaled
Output Ordering	Bit/Digit Reversed
Complex Multipliers	Use CLB Logic
Butterfly Arithmetic	Use CLB Logic

where $k = \lceil log_2 m_{L+1} \rceil$, and L_{FPGA} is the number of bits for the LUT in an FPGA [2].

We estimate the number of LUTs for the NRNS whose moduli set is $\langle m_1, m_2, \ldots, m_L, \langle m_{L+1,1}, m_{L+1,2}, \ldots, m_{L+1,j}, \rangle \rangle$. The dynamic range for the NRNS must satisfy the relation $(m_{L+1})^2 \leq \prod_{i=1}^{j} r_{L+1,i}$. Let $s_i \in \{0, 1\}$ be the **select variable** that satisfies the following:

$$r_{L+1,i} = \begin{cases} m_{L+1,i} & (s_i = 1) \\ 1 & (s_i = 0) \end{cases}$$

Then, the number of LUTs for the NRNS is estimated by

$$L_{NRNS} = \sum_{i=1}^{j} \lceil s_i k_i 2^{2k_i} / L_{FPGA} \rceil,$$

where $k_i = \lceil log_2 m_{L+1,i} \rceil$.

Let G be the **gain** for the number of LUTs. Then, we have

$$G = \frac{L_{RNS}}{L_{NRNS}}. \quad (3)$$

Expr. (3) shows that when $G > 1.00$, the NRNS-based realization requires fewer LUTs than the RNS-based one.

Fig. 13 shows the gain G for $L_{FPGA} = 64$, for different value of the moduli m_{L+1} in the RNS. Note that, in our implementation, we used s_j that minimizes L_{NRNS}. Fig. 13 shows that for $m_{L+1} \geq 8$ except for $m_{L+1} = 15$, $G > 1.00$. Thus, in that case, by applying the NRNS, we can reduce the number of LUTs. Note that for $m_{L+1} = 15$, $G < 1.00$, and the NRNS-based design requires more LUTs.

[2]In the experiment, we used an FPGA that has six-input LUTs. Thus, $L_{FPGA} = 64$

TABLE II. MODULI SET USING IN THE IMPLEMENTATION.

FFT # of points N	Moduli set
1024	(5,7,**9**,**11**,**13**,**16**)
2048	(7,**8**,**9**,**11**,**13**,**17**)
4096	(7,**8**,**11**,**13**,15,**31**)
8192	(7,**11**,**13**,15,**17**,**19**)
16384	(**11**,**13**,**14**,15,**17**,**19**)

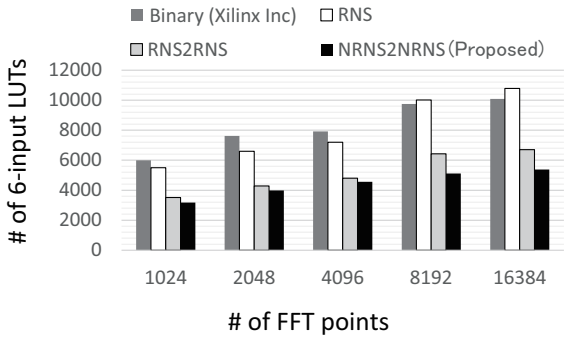

Fig. 14. Comparison of the numbers of 6-input LUTs.

For $m_{L+1} = 15$, since $k = \lceil log_2 m_{L+1} \rceil = 4$, $L_{NRNS} = \lceil k 2^{k+k} / L_{FPGA} \rceil = 16$. We used an optimal moduli set $\langle 3, 7, 11 \rangle$ that minimizes the number of LUTs for the NRNS. In this case, $L_{NRNS} = \lceil 2 \cdot 2^{2+2} / L_{FPGA} \rceil + \lceil 3 \cdot 2^{3+3} / L_{FPGA} \rceil + \lceil 4 \cdot 2^{4+4} / L_{FPGA} \rceil = 21$. We use the NRNS instead of the RNS for $m_{L+1} \geq 8$, except for $m_{L+1} = 15$.

VII. EXPERIMENTAL RESULTS

We implemented the NRNS FFT (NRNS2NRNS) on the Xilinx Inc. Virtex 7 FPGA. Then, we compared it with the Xilinx Inc. binary FFT library (Binary) [18], the conventional RNS FFT (RNS) [16], and the RNS FFT with the RNS2RNS converter (RNS2RNS) [10]. Table I shows the synthesis options for the Xilinx binary FFT library. As for the RNS FFT, we chose moduli set in Table II. Note that, the input signal is represented by 8 bits, and the twiddle factor is represented by 18 bits. Using the experimental result shown in Fig. 13, we applied the NRNS to moduli set that is greater than or equal to 8 except for 15. In Table II, bold numbers show moduli set applied to NRNS.

Fig. 14 compares the numbers of 6-input LUTs, while Fig. 15 compares the numbers of 18Kb BRAMs for different number of FFT points. Since the NRNS decomposed the FFT circuit, it reduced the number of LUTs by 42.4-47.8%. Compared with the RNS FFT, the NRNS reduced the LUT count by 9.4-20.5%. Since the dynamic range is often greater than the bit range of the binary FFT, the amount of swap memory for the RNS FFT are larger than that for the binary FFTs. Fig. 15 shows that compared with the binary FFT, the NRNS-based design required 20.0-156.5% more BRAMs. Compared with the RNS FFT, the NRNS-based design required 34.1% more BRAMs. However, for large N, the number of LUTs is the bottleneck [8], [9], while that of the BRAMs is not. Fig. 16 compares the maximum clock frequencies. Since the proposed NRNS FFT has a smaller realization, it has a shorter critical path. Thus, the proposed one operates 9.3-41.7% faster than conventional ones.

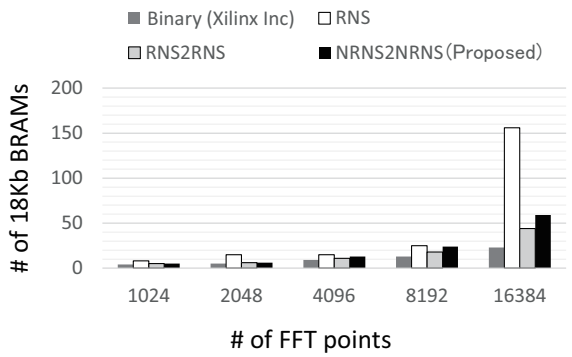

Fig. 15. Comparison of the numbers of 18Kb BRAMs.

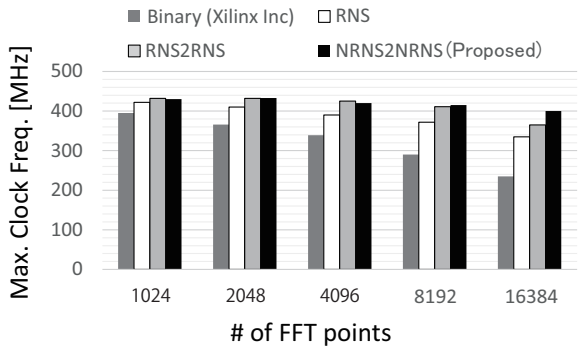

Fig. 16. Comparison of maximum clock frequencies [MHz].

VIII. Conclusion

In this paper, we proposed the FFT circuit based on the NRNS to reduce the number of LUTs. For large modulo m, the NRNS reduces the number of LUTs for arithmetic circuits, while for small m, it does not. This paper compared the NRNS with the RNS for arithmetic circuits. The experimental analysis showed that for $m \geq 8$, the NRNS-based design requires fewer LUTs except for $m = 15$. We realized the NRNS FFT on the Xilinx Inc. Virtex 7 FPGA. Since the NRNS decomposes the FFT circuit into smaller one so that the circuit is implemented efficiently by LUTs, compared with the Xilinx binary FFT, it required 42.4-47.8% fewer LUTs. Compared with the conventional RNS FFT, it required 9.4-20.5% fewer LUTs. Although the NRNS FFT requires more BRAMs, for large inputs N, it is not the bottleneck. As for the maximum clock frequency, since the proposed RNS FFT has a smaller realization, it operated in 9.3-41.7% higher frequency.

IX. Acknowledgments

This research is supported in part by the Grants in Aid for Scientistic Research of JSPS.

References

[1] A. O. Benz, P. C. Grigis, V. Hungerbuhler, H. Meyer, C. Monstein, B. Stuber, and D. Zardet, "A broadband FFT spectrometer for radio and millimeter astronomy," *Astronomy and Astrophysics*, No. 3568, 2008,

[2] CASPER: Collaboration for Astronomy Signal Processing and Electronics Research, https://casper.berkeley.edu/

[3] J. W. Cooley and J. W. Tukey, "An algorithm for the machine computation of complex fourier series," *Mathematics of Computation*, Vol. 19, pp. 297-301, 1965.

[4] S. He and M. Torkelson, "A new approach to pipeline FFT processor," *Proc. of the 10th Int'l Parallel Processing Symposium (IPPS1996)*, pp. 766-770, 1996.

[5] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and applications," *Multiple-Valued Logic: An International Journal*, Vol. 4, No. 1-2, 1998, pp. 9-62.

[6] B. Klein, S. Hochgurtel, I. Kramer, A. Bell, K. Meyer1, and R. Gusten, "High-resolution wide-band Fast Fourier Transform spectrometers," *Astronomy and Astrophysics*, 2012.

[7] Y-T. Lai and S. Sastry, "Edge-valued binary decision diagrams for multi-level hierarchical verification," *DAC1992*, 1992, pp. 608-613.

[8] H. Nakahara, H. Nakanishi, and T. Sasao, "On a wideband fast Fourier transform using piecewise linear approximations: Application to a radio telescope spectrometer," *12th IEEE Int'l Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP2012)*, *Lecture Notes in Computer Science (LNCS 7439)*, 2012, pp.202-217.

[9] H. Nakahara, H. Nakanishi, and T. Sasao, "On a wideband fast Fourier transform for a radio telescope," *3rd Int'l Workshop on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART 2012)*, May 31-June 1, 2012, pp.109-114.

[10] H. Nakahara, T. Sasao, H. Nakanishi, and K. Iwai, "An RNS FFT circuit using LUT cascades based on a modulo EVMDD," *ISMVL2015*, 2015, pp.97-102.

[11] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," *Int'l Conf. on Field Programmable Logic and its applications (FPL2015)*, 2015, pp.1-6.

[12] A. Parsons et.al, "PetaOp/Second FPGA signal processing for SETI and radio astronomy," *Proc. of the Asilomar Conference on Signals, Systems, and Computers*, 2006.

[13] T. Sasao, *Memory-Based Logic Synthesis*, *Springer*, 2011.

[14] SKA, "SKA phase 1 system (level 1) requirements specification," http://www.astronomers.skatelescope.org.

[15] F. J. Taylor, "Residue arithmetic: A tutorial with examples," *IEEE Trans. on Compt.*, Vol. 17, No. 5, pp.50-63, May, 1984.

[16] B. D. Tseng, G. A. Jullien, and W. C. Miller, "Implementation of FFT structures using the residue number system," *IEEE Trans. on Compt.*, Vol.100, No. 11, pp.831-845, 1979.

[17] Xilinx Inc., "7 Series FPGAs Overview," 2015.

[18] Xilinx Inc., "LogiCORE IP fast fourier transform v7.1", 2011.

[19] H. M. Yassine, "Fast arithmetic based on residue number system architectures," *IEEE ISCAS'91*, 1991, pp.2947-2950.