2015 IEEE 45th International Symposium on Multiple-Valued Logic

An RNS FFT Circuit Using LUT Cascades Based on a Modulo EVMDD

Hiroki Nakahara
Ehime University,
3 Chome, Matsuyama,
Ehime 790–8577, Japan

Tsutomu Sasao
Meiji University,
Kawasaki, Kanagawa,
214–8571, Japan

Hiroyuki Nakanishi
Kagoshima University,
1-21-40, Korimoto,
Kagoshima 890–0065, Japan

Kazumasa Iwai
Nobeyama Radio Observatory,
Minamimaki, Minamisaku,
Nagano, 384–1305, Japan

***Abstract*—This paper proposes an FFT circuit based on a residue number system (RNS) using LUT cascades. To reduce the number of look-up tables (LUTs) in an FPGA, we used two techniques. The first one is the functional decomposition of multipliers using RNS. The second one is the increase of the dynamic range stage by stage. The circuit requires the RNS2RNS converter which converts a small dynamic range to a large dynamic range. To compactly realize the RNS2RNS converter, we decompose it into an RNS2Binary converter and a Binary2RNS converter. Although the Binary2RNS converter can be realized by an LUT cascade based on a multi-terminal multi-valued decision diagram (MTMDD), the RNS2Binary converter tend to be large for the conventional circuit. Thus, we introduce an LUT cascade based on a modulo edge-valued multi-valued decision diagram (mod-EVMDD). The mod-EVMDD is a new type of a decision diagram that efficiently represents the RNS2Binary converter. We implemented the proposed RNS FFT on the Xilinx Corp. Virtex 6 FPGA. Compared with the conventional binary FFT implementation, although the number of block RAMs (BRAMs) increased by 11.1-25.0%, the number of LUTs decreased by 44.2-52.2% and the maximum clock frequency increased by 9.3-41.7%. With this technique, we successfully implemented a required FFT on an available FPGA, since the excessive number of LUTs was the bottleneck of the binary FFT.**

I. Introduction

A. Fast Fourier Transform (FFT)

A fast Fourier transform (FFT) is an algorithm to compute the discrete Fourier transform (DFT). The basic idea of the FFT was proposed by Cooley and Tukey in 1965 [2]. In this paper, we realize a compact FFT circuit on a field-programmable gate array (FPGA). The FPGA consists of look-up tables (LUTs) and block RAMs (BRAMs). When a wide-band and high-resolution FFT is implemented on an FPGA, the number of LUTs for the complex multipliers becomes a bottleneck [7], [8]. Thus, the reduction of the number of LUTs is essential. Generally, the required number of LUTs is $O(2^n)$ to implement an n-bit parallel multiplier. A residue number system (RNS) represents a large integer using a set of smaller integers [15], [11]. This means that the RNS can decompose the arithmetic circuit into a set of smaller ones. In this paper, we reduce the number of LUTs by decomposing large multipliers using the RNS.

B. Proposed Method

To reduce the number of LUTs on an FPGA, we used two techniques. The first one is the functional decomposition [3] for modulo arithmetic circuits. The second one is increase of the dynamic range stage by stage. The circuit requires an RNS2RNS converter which converts a small dynamic range into a large dynamic range. In this paper, to realize the RNS2RNS converter compactly, we decompose it into an RNS2Binary converter and a Binary2RNS converter. Although the Binary2RNS converter can be realized the LUT cascade [10] based on the multi-terminal multi-valued decision diagram (MTMDD) [5], the RNS2Binary converter tends to be large for the conventional circuit. We introduce an LUT cascade based on a modulo edge-valued multi-valued decision diagram (mod-EVMDD). The mod-EVMDD is a new type of a decision diagram that efficiently represents the RNS2Binary converter, which is a hybrid of an edge-valued multi-valued decision diagram (EVMDD) [6] and a modulo p MDD (Mod-p MDD) [9].

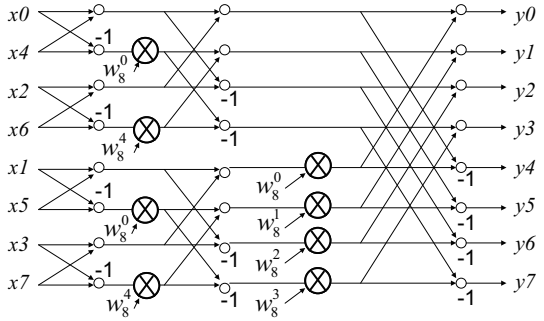

Fig. 1. Signal flow graph.

C. Organization of the Paper

The rest of the paper is organized as follows: Chapter 2 shows the binary FFT circuit; Chapter 3 introduces the residue number system; Chapter 4 shows the FFT circuit based on the RNS (RNS FFT); Chapter 5 proposes the functional decomposition for the butterfly circuits; Chapter 6 proposes the RNS2RNS converter using an LUT cascade based on a mod-EVMDD; 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**

0195-623X/15 $31.00 © 2015 IEEE
DOI 10.1109/ISMVL.2015.41

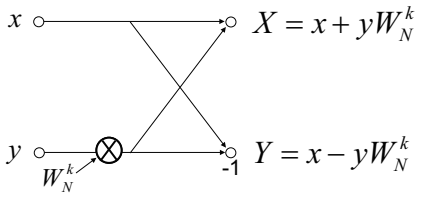

Fig. 2. Radix-2 butterfly operator.

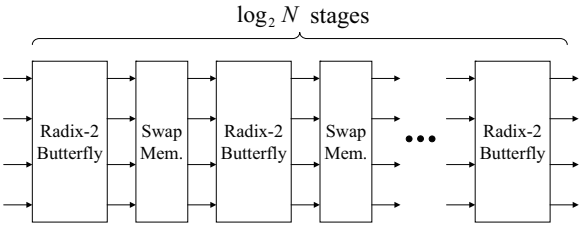

Fig. 3. Pipeline radix-2 binary FFT.

DFT) for $(c_0, c_1, \ldots, c_{N-1})$ is

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

where $w_N^{jk} = exp(-2\pi i \frac{jk}{N})$ is a **twiddle factor**. A time complexity of a direct computation for Expr. (1) is $O(N^2)$. Let r be a radix of the FFT, and s be the number of stages. By applying a decomposition to the N point DFT $s = \lceil \log_r N \rceil$ times recursively, we have a **Cooley-Tukey Fast Fourier Transform (N point FFT)** [2]. Let $s = \lceil log_r N \rceil$ be the number of stages, and r be the radix of the FFT. In the paper, we assume that $r = 2$. Fig. 1 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. 1, 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 memory is wN, and the total amount of memory for the swap memory is $wN\lceil log_r N \rceil$. Fig. 2 shows a **radix-2 butterfly operator** for $r = 2$, which consists of two complex multipliers. Fig. 3 shows a pipeline radix-2 FFT [4], which allows continuous data processing. The problem in the radix-2 binary FFT is that the multipliers tend to be too large, since the dynamic range of the latter stages are very large.

III. Residue Number System

A **residue number system (RNS)** [15], [11] is defined by a set of L integer constants as follows:

$$(m_1, m_2, \ldots, m_L),$$

where no pair of modulus 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:

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

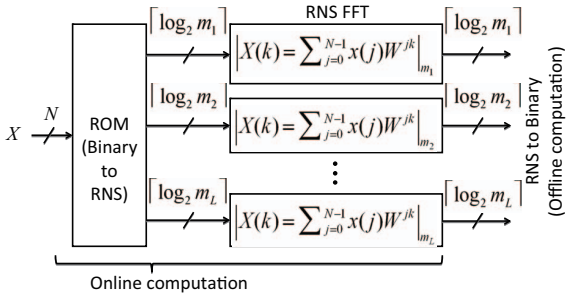

Fig. 4. Radix-2 RNS FFT.

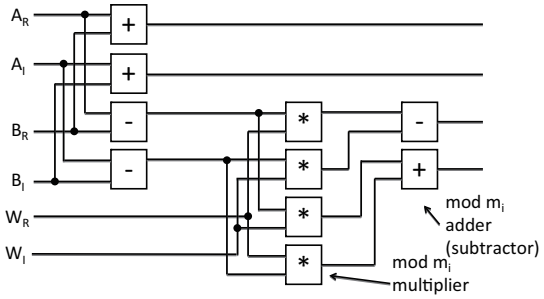

Fig. 5. Modulo m_i RNS butterfly operator.

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 can be performed in digit-wise. Let X and Y be integers, x_i and y_i be integers in the RNS defined by m_i $(1 \leq i \leq L)$, $\circ$ includes $+$ (addition), $-$ (subtraction), and $*$ (multiplication). Then $Z = X \circ Y$ satisfies

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

where $z_i = (X_i \circ Y_i) \ mod \ m_i$. Note that, the division is not included in the operations.

Example 3.1: Let $(m_1, m_2, m_3) = (3, 4, 5)$ be the moduli set. Consider the multiplication $X \times Y$, where $X = 8$ and $Y = 2$. Since $X \times Y = 16$, it is represented by $(1, 0, 1)$ in the RNS. X and Y is represented by $(2, 0, 3)$ and $(2, 2, 2)$ in the RNS, respectively. Thus, $X \times Y$ in the RNS is computed as follows:

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

■

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

IV. Radix-2 RNS FFT Circuit

As shown in Expr. (1), the FFT operation consists of the addition, the subtraction, and the multiplication. Thus, we can apply the RNS to the FFT. Fig. 4 shows the **FFT circuit based on the RNS (RNS FFT)**. First, we convert the binary input signal into the RNS by read only memories (ROMs). Typically,

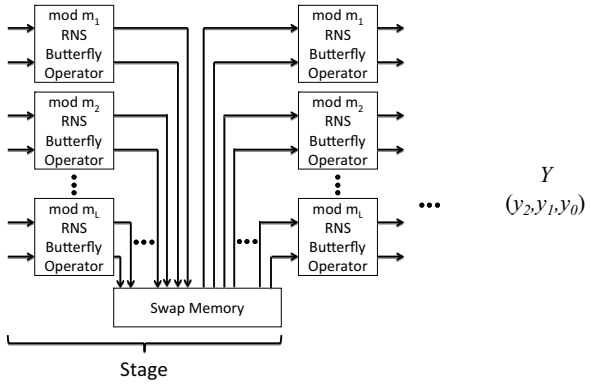

Fig. 6. Swap memory on the radix-2 RNS FFT.

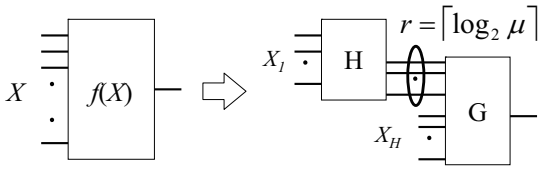

Fig. 7. Functional decomposition.

$X(x_2,x_1,x_0)$

	0 000	1 001	2 010	3 011	4 100
0 000	0 000	1 001	2 010	3 011	4 100
1 001	1 001	2 010	3 011	4 100	5 101
2 010	2 010	3 011	4 100	5 101	6 110
3 011	3 011	4 100	5 101	6 110	7 111
4 100	4 100	5 101	6 110	7 111	8 1000

$X+Y$

Fig. 8. Example of addition.

$X(x_2,x_1,x_0)$

Y (y_2,y_1,y_0)

	0 000	1 001	2 010	3 011	4 100
0 000	0 000	1 001	2 010	3 011	4 100
1 001	1 001	2 010	3 011	4 100	0 000
2 010	2 010	3 011	4 100	0 000	1 001
3 011	3 011	4 100	0 000	1 001	2 010
4 100	4 100	0 000	1 001	2 010	3 011

$|X+Y|_{m=5}$

Fig. 9. Example of modulo addition.

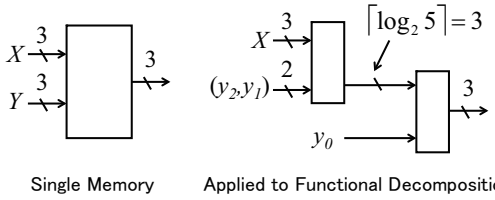

Fig. 10. Example of the functional decomposition of the modulo 5 addition.

the input signals from analog-digital converters (ADCs) are 8-14 bits. The binary to RNS converter can be realized 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. 5 shows the **modulo** m_i **RNS butterfly**, which is derived from the binary butterfly operatior shown in Fig. 2. In Fig. 5, $A = (A_R, A_I)$ and $B = (B_R, B_I)$ denote the complex input, $W = (W_R, W_I)$ denotes the complex twiddle factor, R denotes the real part, and I denotes the imaginary part. The module m_i RNS butterfly can be ralized by the look-up table (LUT) with a small amount of memory [13]. Let m_i be the modulo in the RNS. Then, the amount of memory for the modulo m_i butterfly is $10 \times (m_i)^2 \lceil log_2 m_i \rceil$, since the butterfly operator uses 10 arithmetic circuits. From Fig. 6, as for the N points RNS FFT, the necessary mount of memory is $10 \times (m_i)^2 \lceil log_2 m_i \rceil \times log_2 N$. In other word, the number of LUTs becomes $O((m_i)^2 log m_i log N)$. Therefore, we can decreases the number of LUTs by decreasing m_i .

Fig. 6 shows the swap memory on the radix-2 RNS FFT. As shown in Fig. 4, 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. Reduction of the Number of LUTs by Functional Decompositions

A. Functional Decomposition

In the paper, we realize modulo arithmetic circuits by LUTs. By applying functional decompositions [3], we can reduce the number of LUTs.

Consider a function $F(\vec{X}) : B^n \rightarrow \{0, 1, \ldots m-1\}$, where $B = \{0, 1\}$ and $\vec{X} = (x_1, x_2, \ldots, x_n)$. Let $(\vec{X}_L, \vec{X}_H)$ be a partition of $\vec{X}$ into two parts. A **decomposition chart** of F is the two-dimensional matrix, where each column label has distinct assignment of elements in $\vec{X}_L$, and each row label has distinct assignment of elements in $\vec{X}_H$, and the corresponding matrix value is $F(\vec{X}_L, \vec{X}_H)$. The number of different column patterns in the decomposition chart is the **column multiplicity**. $\vec{X}_L$ denotes the **bound variables**, and $\vec{X}_H$ denotes the **free variables**.

Fig. 7 shows the functional decomposition. When $f(X)$ is realized by a single memory, its amount of memory is 2^n bits. Let $r_1 = \lceil log_2 \mu \rceil$, $|X_L| = n_1$, and $|X_H| = n_2$. By applying the functional decomposition, its amount of memory is reduced to $2^{n_1} \times r_1 + 2^{r_1 + n_2}$ bits.

B. Functional Decomposition for Modulo Addition

Let the modulo be $m = 5$. Fig. 8 shows an example of a conventional addition, and Fig. 9 shows an example of the modulo addition. In the modulo addition, the column multiplicity is at most m. Fig. 10 shows an example of the functional decomposition of the modulo 5 addition. In this case, when it is realized by a single memory, its amount of memory is $2^{3+3} \times 3 = 192$ bits. On the other hand, by applying the functional decomposition, its amount of memory is reduced to $2^5 \times 3 + 2^4 \times 3 = 144$. This means that we can reduce the number of LUTs by the functional decomposition.

$X(x_2,x_1,x_0)$

$Y(y_2,y_1,y_0)$	0 000	1 001	2 010	3 011	4 100
0 000	0 0000	0 0000	0 0000	0 0000	0 0000
1 001	0 0000	1 0001	2 0010	3 0011	4 0100
2 010	0 0000	2 0010	4 0100	6 0110	8 1000
3 011	0 0000	3 0011	6 0110	9 1001	12 1100
4 100	0 0000	4 0100	8 1000	12 1100	16 10000

$X \times Y$

Fig. 11. Example of multiplication.

$X(x_2,x_1,x_0)$

$Y(y_2,y_1,y_0)$	0 000	1 001	2 010	3 011	4 100
0 000	0 0000	0 0000	0 0000	0 0000	0 0000
1 001	0 0000	1 0001	2 0010	3 0011	4 0100
2 010	0 0000	2 0010	4 0100	1 0001	3 0011
3 011	0 0000	3 0011	1 0001	4 0100	2 0010
4 100	0 0000	4 0100	3 0011	2 0010	1 0001

$|X \times Y|_{m=5}$

Fig. 12. Example of modulo 5 multiplication.

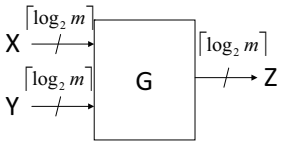

Fig. 13. Explain of Corollary 5.1.

C. Functional Decomposition for the Modulo Multiplication

Let the modulo be $m = 5$. Fig. 11 shows an example of a conventional multiplication, and Fig. 12 shows an example of the modulo 5 multiplication. In a similar manner to the modulo addition, we have the upper bound of the column multiplicity for any modulo operation.

Corollary 5.1: Let X and Y be k bits integers in the RNS. Then, the function $Z = X \circ Y \ (mod\ m)$ takes at most m unique values, where $k = \lceil log_2 m \rceil$.

(Proof) As shown in Fig. 13, from the property of the modulo operation, Z takes at most m unique values (Q.E.D).

Theorem 5.1: Let $X = (x_{k-1}, x_{k-2}, \ldots, x_0)$, and $Y = (y_{k-1}, y_{k-2}, \ldots, y_0)$, where X and Y are the integers in the RNS. Let $Y_1 = (y_{t-1}, y_{t-2}, \ldots, y_0)$ and $Y_2 = (y_{k-1}, y_{k-2}, \ldots, y_t)$ be a partition of Y, where $1 < t < k-1$. Then, the circuit shown in Fig. 14 realizes the function $Z = X \circ Y \ (modm)$, where $k = \lceil log_2 m \rceil$. And the output of G takes at most m unique values.

(Proof) Since Y_1 is derived from the lower t bits part of the Y, Y_1 represents a part of m unique values. Thus, G takes at most m unique values (Q.E.D).

In particular, for the modulo multiplier, we can reduce the number of LUTs by the functional decomposition.

VI. Reduction of the Number of LUTs by the RNS2RNS Converter

A. RNS FFT using the RNS2RNS Converter

When the RNS FFT shown in Fig. 4 is directly realized, since the dynamic range is too large for the first half stages of the RNS butterflies, the number of LUTs tends to be large. In this paper, we increase the dynamic range stage by stage. Fig. 15 shows the RNS FFT inserted the **RNS2RNS converter**

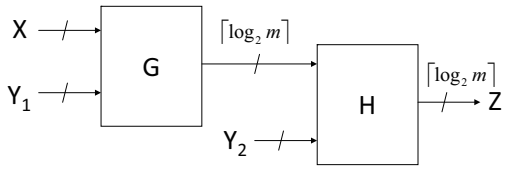

Fig. 14. Explain of Theorem 5.1.

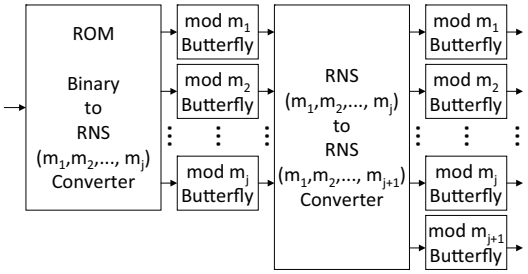

Fig. 15. RNS FFT inserting the RNS2RNS converter.

which converts a small dynamic range to a large dynamic range. As shown in Fig. 6, in the first part of the RNS FFT, since large moduli are removed, the number of LUTs for the first parts is reduced. However, it requires the RNS2RNS converter. In this paper, we use a compact realization of the RNS2RNS converter.

Fig. 16 shows the truth table for the RNS2RNS converter which converts $(m_1, m_2) = (2, 3)$ to $(m_1, m_2, m_3) = (2, 3, 5)$. Generally, we can use an arbitrary moduli set in the RNS2RNS converter. In this paper, 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. 17, we can realize the RNS2RNS converter by realizing only the function $g'(m_1, m_2, \ldots, m_L) = m_{L+1}$. Let $M = \prod_{i=1}^{L} m_i$ be the dynamic range. When the RNS2RNS converter is realized by a single memory, its amount of memory is $M\lceil log_2 m_{L+1} \rceil$ bits. In this paper, as shown in Fig. 18, 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 its column multiplicity is at most m_{L+1}. In the same manner as the modulo addition/multiplication, we can reduce the number of LUTs by the functional decomposition.

Fig. 19 shows an example of the MTMDD (Multi-Terminal Multi-Valued Decision Diagram) [5] for the RNS2Bin converter. As shown in Fig. 19, the column multiplicity is up to M, which is too large to apply the functional decomposition.

B. LUT Cascade for the RNS2RNS Converter

As shown in Fig. 19, all the adjacent terminal values can be calculated by +4 mod 6. In this example, the dynamic range is $2 \times 3 = 6$. Similarly, the upper index values can be calculated by +3 mod 6. This example shows that the MTMDD representing the RNS2Bin converter has a regularity. We propose a new type of decision diagram that compactly represents the RNS2Bin converter. Fig. 20 shows the **modulo edge-valued MDD (mod-EVMDD)** which is a hybrid of the edge-valued multi-valued decision diagram (EVMDD) [6] and the module p MDD (Mod-p MDD) [9]. In the mod-EVMDD, by adding the integer weights mod M, we have the function

(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. 16. Example of the truth table of the RNS2RNS converter.

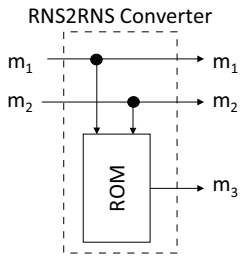

Fig. 17. Example of the RNS2RNS converter.

m_1	m_2	Bin
0	0	0
0	1	4
0	2	2
1	0	3
1	1	1
1	2	5

Bin	m_3
0	0
1	1
2	2
3	3
4	4
5	0

Fig. 18. Decomposition the RNS2RNS converter.

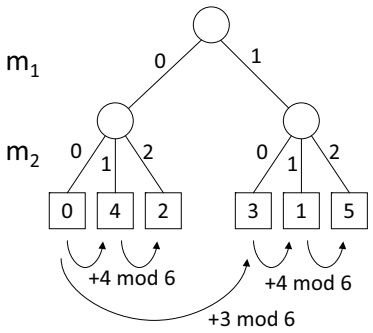

Fig. 19. Example of MTMDD representing the RNS2Bin converter.

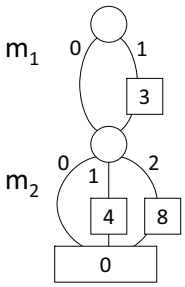

Fig. 20. Example of the mod-EVMDD for the RNS2Bin converter.

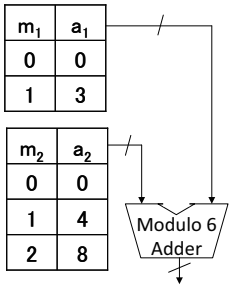

Fig. 21. Example of the LUT cascade based on the mod-EVMDD.

value. When $(m_1, m_2) = (1, 1)$, by traversing the MTMDD shown in Fig. 19, we have 1. By traversing the mod-EVMDD as shown in Fig. 20, we have weights 3 and 4. In this example, since the dynamic range is $M = 6$, we have $3+4 \equiv 1(mod\ 6)$.

The RNS2Bin converter is efficiently realized by an LUT cascade [10] with modulo adders shown in Fig. 21. In this case, since the width of the mod-EVMDD is at most one, no rail is necessary. The output from each LUT represet the weights of edges. We call such outputs $Arailsa_i$. By connecting the Arails a_i through modulo adders, we have the LUT cascade based on the mod-EVMDD.

Fig. 22 shows the RNS2RNS converter using LUT cascades. The RNS2Bin converter is realized by the LUT cascade based on the mod-EVMDD, while the Bin2RNS converter is realized by one based on the MTMDD. Let $M = \prod_{i=1}^{L} m_i$ be the dynamic range. Since the proposed cascade decomposes the memory of $O(2^M)$ bits into $O(2^{m_i})$ bits, it drastically reduces the number of LUTs.

VII. Experimental Results

A. Comparision with Binary FFT

We implemented the proposed RNS FFT on the Xilinx Corp. Virtex 6 FPGA, and we compared it with the binary FFT (Xilinx Corp. FFT library [14]). Table I shows the synthesis options for the Xilinx FFT library. As for the RNS FFT, we chose moduli set as shown in Table II. Note that, for both FFTs, the input signal is represented by 8 bits, and the twiddle factor is represented by 18 bits.

Fig. 23 (a) compares the number of 6-input LUTs in the Virtex 6 FPGA, while Fig. 23 (b) compares the number of 18Kb BRAMs. Since the RNS FFT decompose butterfly operators into smaller ones, it reduced the number of LUTs by 44.2-52.2%. Typically, the dynamic range exceeds the bit range of the binary FFT, the amount of swap memory for the RNS FFT tends to be larger than that for the binary FFT. From Fig. 23 (b), the number of BRAMs is increased by 11.1-25%. However, for large N, the number of LUTs becomes a bottleneck [7], [8], however, that of the BRAMs is not a bottleneck. Fig. 23 (c) compares the maximum clock frequency. Since the proposed RNS FFT has a smaller realization, it has a shorter critical path. Thus, the proposed one has a higher clock frequency by 9.3-41.7%.

VIII. Conclusion

In this paper, we reduced the number of LUTs for the RNS FFT. To reduce the number of LUTs, we used two techniques. First, we applied the functional decomposition to modulo butterfly operators. Second, we increase the dynamic range stage by stage. To compactly realize the RNS2RNS converter, we decomposed the RNS2RNS converter into the RNS2Bin converter and the Binary2RNS converter. We proposed the mod-EVMDD representing the RNS2Bin converter compactly. The RNS2Bin converter was realized by the LUT cascade based on the mod-EVMDD, while Bin2RNS converter was realized by one based on the MTMDD. We implemented the proposed RNS FFT on the Xilinx Corp. Virtex 6 FPGA. Compared with the binary FFT, although the proposed RNS FFT requires 11.1-25.0% more BRAMs, it requires 44.2-52.2% fewer LUTs and has 9.3-41.7% higher clock frequency.

The future projects are an error analysis [12]; and comparison with existing RNS FFT circuits [1], [13].

IX. Acknowledgments

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

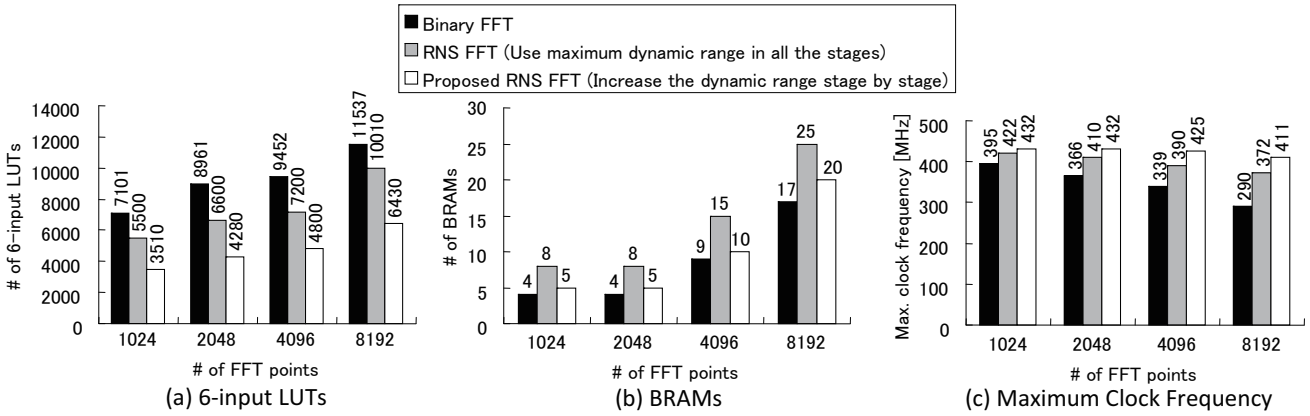

Fig. 23. Comparison with the binary FFT.

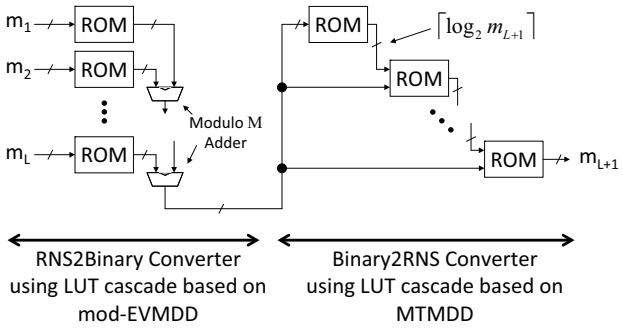

Fig. 22. RNS2RNS converter using the LUT cascades.

TABLE I. SYNTHESIS OPTIONS FOR THE 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

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)

REFERENCES

[1] G. Alia, F. Barsi and E. Martinelli, "A fast near optimum VLSI implementation of FFT using residue number systems," *Integration, the VLSI Journal*, Vol. 2, No. 2, pp. 133-147, 1984.

[2] 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.

[3] H. A. Curtis, "A New Approach to the Design of Switching Circuits," *D. Van Nostrand Co.,* Princeton, NJ, 1962.

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

[7] 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.

[8] 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.

[9] H. Sack, E. Dubrova, and C. Meinel, "Mod-p decision diagrams: A data structure for multiple-valued functions," *ISMVL2000*, 2000, pp.233-238.

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

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

[12] B. Tseng, W. Miller, G. Jullien, J. Soltis and A. Baraniecka, "An error analysis of a FFT implementation using the residue number system," *Proc. ICASSP'78*, pp. 800-803, 1978.

[13] 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.

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

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