On Designs of Radix Converters Using Arithmetic Decompositions

Yukihiro Iguchi [1] Tsutomu Sasao [2] Munehiro Matsuura [2]

[1] Dept. of Computer Science, Meiji University, Kawasaki 214-8571, Japan
[2] Dept. of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka 820-8502, Japan

February 28, 2006

Abstract

In arithmetic circuits for digital signal processing, radixes other than two are often used to make circuits faster. In such cases, radix converters are necessary. However, in general, radix converters tend to be complex. This paper considers design methods for p-nary to binary converters. It introduces a new design technique called arithmetic decomposition. It also compares the amount of hardware and performance of radix converters implemented on FPGAs.

1 Introduction

Arithmetic operations of digital systems usually use radix two [7]. However, in digital signal processing, for high speed operations, p-nary ($p > 2$) numbers are often used [1, 4]. In such cases, the conversion between binary numbers and p-nary numbers are necessary. Such operation is **radix conversion** [2, 6]. Various methods exist to convert p-nary numbers into binary numbers. Many of them require large amount of computations. Especially when the radix conversion is implemented by a random logic circuit, the network tends to be quite complex [5]. Radix converters can be implemented by table lookup. That is, to store the conversion table in the memory. This method is fast but requires a large amount of memory.

In [9], LUT cascade realizations [8] of binary to ternary converters, ternary to binary converters, binary to decimal converters, and decimal to binary converters are presented. In [11], the concept of Weighted-Sum functions (WS functions) is used to design radix converters by using LUT cascades.

In this paper, we consider the design of circuits that convert p-nary numbers into binary numbers by using arithmetic decomposition [10]. We also consider the implementations on FPGAs (Field Programmable Gate Arrays). For the readability, we use examples for $p = 3$, however the method can be easily extended to any value of p.

2 Radix Converter

2.1 Radix Conversion

Definition 2.1 *Let a p-nary number of n-digit be $\vec{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)_p$, and let a q-nary number of m-digit be $\vec{y} = (y_{m-1}, y_{m-2}, \ldots, y_0)_q$. Given a vector $\vec{x}$, the* **radix conversion** *is the operation that obtains $\vec{y}$ that satisfies the relation:*

$$\sum_{i=0}^{n-1} x_i p^i = \sum_{j=0}^{m-1} y_j q^j, \tag{2.1}$$

where $x_i \in P$, $y_j \in Q$, $P = \{0, 1, \ldots, p-1\}$, and $Q = \{0, 1, \ldots, q-1\}$.

Let $\vec{y} = (y_{m-1}, y_{m-2}, \ldots, y_0)$, $y_i \in \{0, 1\}$ be the output functions of p-nary to binary converter. Then, when p is a prime number, y_i depends on all the inputs $x_i (i = 0, 1, \ldots, n-1)$. When p is not a power of two, we have an incompletely specified function. When we implement a p-nary to binary converter, unused combinations exist. Usually, we assign 0 to the undefined outputs.

Example 2.1 *In the case of ternary to binary converter, we use the binary-coded-ternary code to represent a ternary number. That is* 0 *is represented by* (00), 1 *is represented by* (01), *and* 2 *is represented by* (10). *Note that* (11) *is the unused code. Table 2.1 is the truth table of two-digit ternary to binary converter. In the binary-coded-ternary representation,* (11) *is an undefined input, and corresponding output is don't care. In Table 2.1, the inputs in the binary-coded-ternary representation are denoted by $\vec{z} = (z_3, z_2, z_1, z_0)$. The inputs in the ternary representations are denoted by $\vec{x} = (x_1, x_0)$. The output in the binary representations are denoted by $\vec{y} = (y_3, y_2, y_1, y_0)$.* (*End of Example*)

2.2 Direct Method

A straightforward way to implement a radix converter is to realize the circuit for equation (2.1).

Example 2.2 *Consider the 8-digit ternary to binary converter (***8ter2bin***). In this case, we implement*

$$3^7 x_7 + 3^6 x_6 + 3^5 x_5 + 3^4 x_4 + 3^3 x_3 + 3^2 x_2 + 3^1 x_1 + 3^0 x_0. \tag{2.2}$$

Table 2.1: Truth table of a ternary to binary converter.

Binary-coded -ternary				Ternary		Binary				Decimal
z_3	z_2	z_1	z_0	x_1	x_0	y_3	y_2	y_1	y_0	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	1	1
0	0	1	0	0	2	0	0	1	0	2
0	1	0	0	1	0	0	0	1	1	3
0	1	0	1	1	1	0	1	0	0	4
0	1	1	0	1	2	0	1	0	1	5
1	0	0	0	2	0	0	1	1	0	6
1	0	0	1	2	1	0	1	1	1	7
1	0	1	0	2	2	1	0	0	0	8

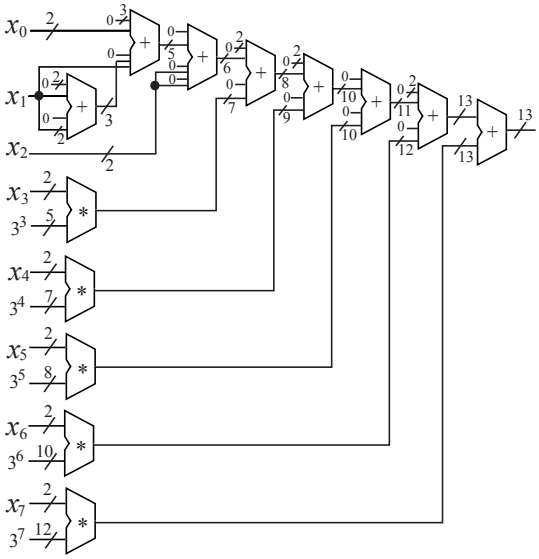

Figure 2.1: 8-digit ternary to binary converter: Direct Method.

Fig. 2.1 shows the circuit for 8ter2bin produced by a logic synthesis program. Note that 3^2x_2 is implemented by $8x_2+x_2$ and 3^1x_1 is implemented by $2x_1+x_1$, but 3^ix_i $(2<i\leq 7)$ are implemented by multipliers. Also, a cascade adder is used to obtain the result. Since the coefficients are constants, the multipliers can be replaced by adders. However, the wiring of resulting circuit is quite complex. $(End\ of\ Example)$

3 Arithmetic Decomposition

In the direct implementation of a p-nary to binary converter, the amount of hardware and the propagation delay increase with the number of input digits. To reduce the amount of hardware, we can use arithmetic decomposition [10]. In this section, we introduce arithmetic decompositions that are suitable for radix converters.

3.1 WS functions and Their Arithmetic Decomposition

The Weighted Sum function (WS function) is a mathematical model of radix converters, bit-counting circuits, and convolution operations [11, 10].

Definition 3.1 *An n input WS function [11] is defined as*

$$WS(\vec{x}) = \sum_{i=0}^{n-1} w_i \cdot x_i, \tag{3.1}$$

where $\vec{x} = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$ is the input vector, $\vec{w} = (w_{n-1}, w_{n-2}, \ldots, w_1, w_0)$ is the **weight vector***, and each element is an integer.*

Theorem 3.1 *A WS function can be represented as a sum of two WS functions as follows:*

$$WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i = \alpha WS_A(\vec{x}) + WS_B(\vec{x}),$$

where $WS_A(\vec{x}) = \sum_{i=0}^{n-1} a_i x_i$, $WS_B(\vec{x}) = \sum_{i=0}^{n-1} b_i x_i$, and α is an integer. This is the **arithmetic decomposition***, and α is a* **decomposition coefficient***.*

3.2 Arithmetic Decompositions using Different Coefficients

A radix converter can be represented as a WS function. Thus, by using arithmetic decompositions with different decomposition coefficients α, we can implement various radix converters. In this part, we consider two cases: One uses 2^k as the decomposition coefficient, and the other uses p^k as the decomposition coefficient.

When the decomposition coefficient is 2^k : In this case, the radix converter is realized as

$$WS(\vec{x}) = 2^k WS_A(\vec{x}) + WS_B(\vec{x}).$$

Since the multiplication of 2^k can be implemented by the shift (wiring), it is implemented compactly. However, WS_B depends on all the input variables, and the total size of the circuit is not so small.

When the decomposition coefficient is p^k : In this case, the radix converter is realized as

$$WS(\vec{x}) = p^k WS_A(\vec{x}) + WS_B(\vec{x}).$$

The multiplication of p^k increases the number of outputs for $p^k WS_A$. However, the numbers of inputs for WS_A and WS_B are half of the original function, so the total network will be much smaller.

Example 3.1 *Let us design the 8-digit ternary to binary converter (***8ter2bin***). Consider two cases where the decomposition coefficients are α=$2^6 = 64$ and α=$3^4 = 81$. The*

Table 3.1: Coefficients of arithmetic decompositions of the powers of 3.

		Decomposition Coefficients	
i	3^i	$\alpha = 2^6 = 64$	$\alpha = 3^4 = 81$
0	*1*	$64\times 0+1$	$81\times 0+1$
1	*3*	$64\times 0+3$	$81\times 0+3$
2	*9*	$64\times 0+9$	$81\times 0+9$
3	*27*	$64\times 0+27$	$81\times 0+27$
4	*81*	$64\times 1+17$	$81\times 1+0$
5	*243*	$64\times 3+51$	$81\times 3+0$
6	*729*	$64\times 11+25$	$81\times 9+0$
7	*2187*	$64\times 34+11$	$81\times 27+0$

ternary number is represented by the binary-coded-ternary code. Table 3.1 shows the coefficients of arithmetic decompositions of 3^i, $(i = 0,1,2,\ldots,7)$. Note that these coefficients are equal to the weights for $WS_A(\vec{x})$ and $WS_B(\vec{x})$. We assume that 11-input cells are available for cascade realization. From Table 3.1, we have two different realizations for 8ter2bin.

When the decomposition coefficient is 2^6.

- $WS(\vec{x}) = 2^6 WS_A(\vec{x}) + WS_B(\vec{x})$.
- $WS_A(\vec{x}) = 34x_7 + 11x_6 + 3x_5 + 1x_4 + 0x_3 + 0x_2 + 0x_1 + 0x_0$.
- $WS_B(\vec{x}) = 11x_7 + 25x_6 + 51x_5 + 17x_4 + 27x_3 + 9x_2 + 3x_1 + 1x_0$.
- *$WS_A(\vec{x})$ depends on the inputs x_4∼x_7. The number of inputs is 8. Since the output takes values from 0 to $2(1+3+11+34) = 98$, 7 bits are necessary to represent the output. $WS_A(\vec{x})$ has 8 inputs and 7 outputs, so it is implemented by a single cell.*
- *$WS_B(\vec{x})$ depends on all the inputs x_0∼x_7, so the number of inputs is 16. Since the output takes the values from 0 to $2(1+3+9+27+17+51+25+11) = 288$, 9 bits are necessary to represent the output. $WS_B(\vec{x})$ has 16 inputs and 9 outputs. It is implemented by a cascade with 11-input cells.*
- *The multiplication by 2^6 can be simply implemented by shifting 6 bits positions. We add the upper 3 bits of WS_B and the outputs of WS_A by a 7-bit adder. Fig. 3.1 shows the network, which uses memory with 39.5K bits and a 7-bit adder.*

When the decomposition coefficient is 3^4.

- $WS(\vec{x}) = 3^4 WS_A(\vec{x}) + WS_B(\vec{x})$.
- $WS_A(\vec{x}) = 3^3x_7 + 3^2x_6 + 3^1x_5 + 3^0x_4$.
- $WS_B(\vec{x}) = 3^3x_3 + 3^2x_2 + 3^1x_1 + 3^0x_0$.
- *$WS_A(\vec{x})$ depends on inputs x_4∼x_7, so the number of the inputs is 8. Since the output takes the values from 0 to $2(1+3+9+27) = 80$, 7 bits are necessary to represent the output. Thus, $WS_A(\vec{x})$ is implemented by a single cell.*

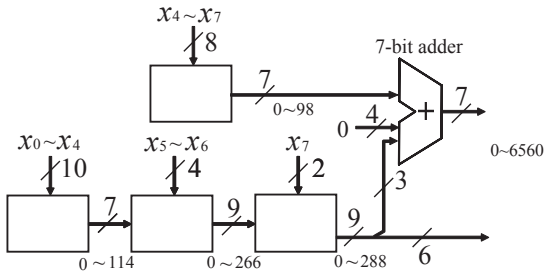

Figure 3.1: 8-digit ternary to binary converter: Arithmetic decomposition with coefficient 2^6.

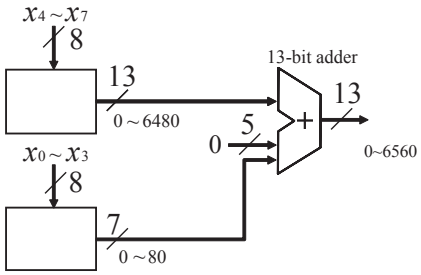

Figure 3.2: 8-digit ternary to binary converter: Arithmetic decomposition with coefficient 3^4.

- *$WS_B(\vec{x})$ depends on inputs x_0∼x_3, so the number of inputs is 8. The output takes values from 0 to $2(1+3+9+27) = 80$.*
- *The output range of $3^4 WS_A$ is $0 \sim 6480$. So 13 bits are necessary to represent the output. We directly implement $3^4 WS_A$ by a cell. We add $3^4 WS_A$ with WS_B by a 13-bit adder. Fig. 3.2 shows the network, which uses 5K bits memories and a 13-bit adder.*

(End of Example)

3.3 Arithmetic Decomposition using the Binary Representation of Inputs

In this part, we will introduce an arithmetic decomposition with respect to the binary representation of inputs.

Definition 3.2 *Let i be an integer. $BIT(i,j)$ denotes the j-th bit of the binary representation of i, where the LSB is the 0-th bit.*

Example 3.2 $BIT(2,1) = 1, BIT(2,0) = 0, BIT(1,1) = 0,$ *and* $BIT(1,0) = 1$.

An integer number i can be represented by $\lceil \log_2 i \rceil$ bits. Thus, we have the relation:

$$i = \sum_{j=0}^{\lceil \log_2 i \rceil - 1} 2^j BIT(i,j).$$

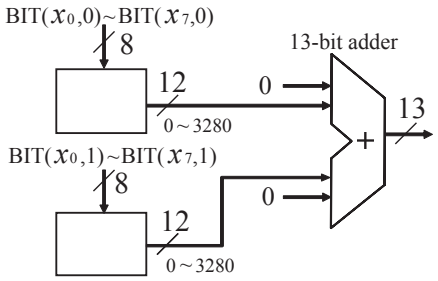

Figure 3.3: 8-digit ternary to binary converter: Decomposed using the binary representation of inputs.

From this, we have the following:

Theorem 3.2 *A p-nary to binary converter can be represented as the form*

$$WS(\vec{x}) = \sum_{j=0}^{\lceil \log_2 p \rceil - 1} 2^j \sum_{i=0}^{n-1} p^i BIT(x_i, j)$$

(Proof)

$$WS(\vec{x}) = \sum_{i=0}^{n-1} p^i x_i = \sum_{i=0}^{n-1} p^i \sum_{j=0}^{\lceil \log_2 p \rceil - 1} 2^j BIT(x_i, j)$$
$$= \sum_{j=0}^{\lceil \log_2 p \rceil - 1} 2^j \sum_{i=0}^{n-1} p^i BIT(x_i, j)$$

$(Q.E.D.)$

Example 3.3 *Consider the 8-digit ternary to binary converter (***8ter2bin***). By Theorem 3.2,* $WS(\vec{x})$ *can be represented as:*

$$WS(\vec{x}) = 2\sum_{i=0}^{7} 3^i BIT(x_i, 1) + \sum_{i=0}^{7} 3^i BIT(x_i, 0).$$

Fig. 3.3 is the circuit corresponding to the above decomposition. Each module has 8 inputs. Since $3^7+3^6+3^5+3^4+3^3+3^2+3^1+3^0 = 3280$*, each module has 12 outputs. The multiplication by two is implemented by shifting one bit position. The circuit uses 6K bits of memories and a 13-bit adder.*

We can further reduce the circuit by using Theorem 3.1, where 3^4 *is the decomposition coefficient:*

$$WS(\vec{x}) = 2 \cdot [3^4 \cdot \sum_{i=4}^{7} 3^{i-4} BIT(x_i, 1) + \sum_{i=0}^{3} 3^i BIT(x_i, 1)]$$
$$+1 \cdot [3^4 \cdot \sum_{i=4}^{7} 3^{i-4} BIT(x_i, 0) + \sum_{i=0}^{3} 3^i BIT(x_i, 0)].$$

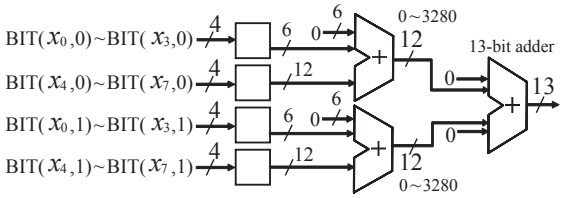

Figure 3.4: 8-digit ternary to binary converter: Decomposed using the binary representation of inputs and further decomposed with coefficient 3^4.

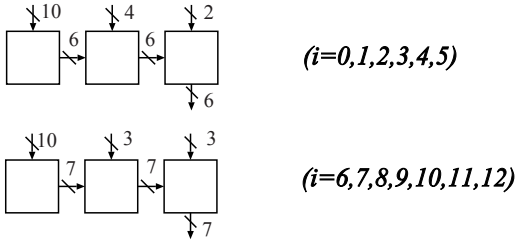

Figure 4.1: 8-digit ternary to binary converter: Outputs are partitioned into two groups.

Fig. 3.4 is the circuit corresponding the above decomposition, where each logic block has only 4 inputs. In this case, the total amount of memory is only 576 bits. $(End\ of\ Example)$

4 Partition of Outputs

The **decomposition** of a logic function is to design the circuit by partitioning the inputs, while the **partition** of a logic function is to design the circuit by partitioning the outputs. The arithmetic decomposition requires an adder in the outputs, while the partition of the outputs requires no adder in the outputs. So, the circuit can be faster.

Example 4.1 *[9] Consider the* 8*-digit ternary to binary converter (***8ter2bin***). Let the output values for the don't care inputs be* 0*. Then, the column multiplicity of the decomposition chart of the radix converter will be* $3^8 = 6551$*. This shows that the single cascade realization of the radix converter requires cells with* $\lceil \log_2 6551 \rceil + 1 = 14$ *inputs, which is rather large. So, we partition the output to reduce the amount of hardware.*

Assume that LUT cascades with 10*-input cells are used. We can implement the radix converter as shown in Fig.4.1. In this implementation, the outputs are partitioned into two groups: the upper* 7 *bits and the lower* 6 *bits are implemented separately. As shown in Fig. 4.1, the amount of memory in the cascades is* $2^{10}(7+7+7+6+6)+2^8(6) = 35,328$ *(bits).* $(End\ of\ Example)$

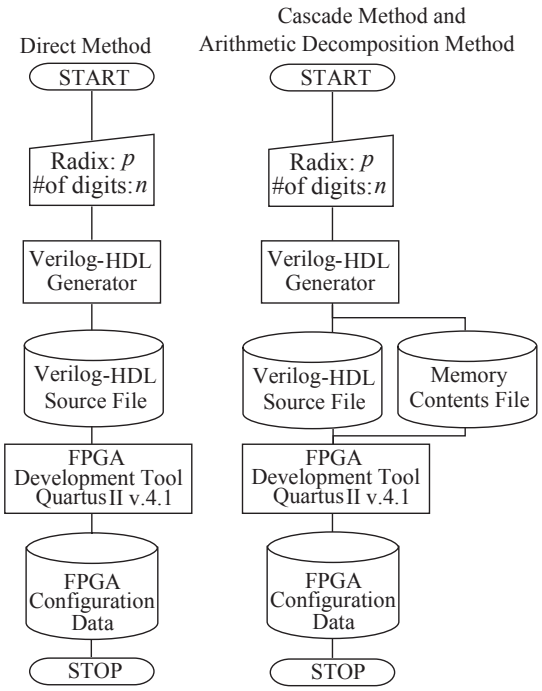

Figure 5.1: Development system for radix converters.

5 Implementation on FPGAs

To see the effectiveness of the approach, we implemented various designs of ternary to binary converters on FPGAs, and compared the amount of hardware and performance.

5.1 FPGAs and Their Development System

We used Altera Cyclone II (EP2C5T144C7) FPGA device, having 13 Embedded Multipliers (EMs) that perform the multiply-and-sum operations, 26 embedded memories (M4Ks), and 4608 logic elements (LEs). Each M4K contains 4096 bits. We used Altera Quartus II V.4.1 as the development tool. We also developed a radix converter synthesis system shown in Fig. 5.1 that generates Verilog-HDL codes describing various designs, and data for M4Ks. In the FPGAs, LUTs (cells) were implemented by M4Ks, while adders were implemented by LEs.

5.2 8-digit Ternary to Binary Converters

Table 5.1 compares 7 different designs of 8-digit ternary to binary converters (**8ter2bin**).

Direct Method (DM): The system generated Verilog-HDL code from the specification: radix p and the number of digits n.

- DM1 directly implements $\sum_{i=0}^{7} 3^i x_i$. Fig. 2.1 is the circuit generated by the Quartus. After mapping, the Quartus replaced the multipliers with 7 EMs, and adders with 66 LEs.
- DM2 also corresponds to Fig. 2.1. In this case, however, the Quartus replaced multipliers with LEs instead of EMs. So, the circuit consists of LEs only. It has 195 LEs, which means 129 LEs replaced 7 EMs. It is faster than DM1, since LEs perform constant multiplications faster than EMs.

Table 5.1: Amount of hardware and performance of 8-digit ternary to binary converters on Cyclone II.

Design Method			LE	M4K	EM	Delay [nsec]
DM1	With EM	Fig.2.1	66	0	7	26.7
DM2	W/O EM	Fig.2.1	195	0	0	23.7
AD1	2^6	Fig.3.1	8	13	0	30.0
AD2	3^4	Fig.3.2	13	2	0	14.8
AD3	BIT	Fig.3.3	12	2	0	14.3
AD4	BIT+3^4	Fig.3.4	36	4	0	16.8
PAR	M4K only	Fig.4.1	0	11	0	22.2

Arithmetic Decomposition Method (AD): The system generated Verilog-HDL code and data for M4Ks.

- AD1 corresponds to Fig. 3.1, which was obtained with the decomposition coefficient 2^6. The Quartus replaced four LUTs with 13 M4Ks, and the adder with 8 LEs.
- AD2 corresponds to Fig. 3.2, which was obtained with the decomposition coefficient 3^4. The Quartus replaced two LUTs with two M4Ks, and the adder with 13 LEs.
- AD3 corresponds to Fig. 3.3, which was obtained with the arithmetic decomposition using binary representation of inputs. The Quartus replaced two LUTs with two M4Ks, and the adder with 12 LEs.
- AD4 corresponds to Fig. 3.4, which was obtained by the arithmetic decomposition with the coefficient 3^4 and using binary representation of inputs. The Quartus replaced four LUTs with four M4Ks, and adders with 36 LEs. It is slower than AD3 since the adder is more complex.

Partition of Outputs Method (PAR): The system generated Verilog-HDL code and data for M4Ks.

- PAR corresponds to Fig. 4.1 consisting of 6 LUTs. The Quartus replaced 6 LUTs with 11 M4Ks.

In the case of **8ter2bin**, we can conclude that AD3 is the best realizations: It is the fastest and requires smaller amount of hardware.

Table 5.2: Amount of hardware and performance of 12-digit ternary to binary converters on Cyclone II.

Design Method			LE	M4K	EM	Delay [nsec]
DM1	With EM	Fig.2.1	139	0	15	35.8
DM2	W/O EM	Fig.2.1	457	0	0	32.0
AD2	3^6	Fig.3.2	20	30†	0	17.3
AD3	BIT	Fig.3.3	19	38‡	0	16.6
AD4	BIT+3^6	Fig.3.4	57	4	0	17.6

†:Used EP2C8T144C7
‡:Used EP2C20F256C7

5.3 12-digit Ternary to Binary Converters

Table 5.2 compares 5 different designs of 12-digit ternary to binary converters (**12ter2bin**).

Direct Method (DM):

- DM1 is similar to Fig. 2.1, but uses 15 EMs.
- DM2 is also similar to Fig. 2.1. Also in this case, it is faster than DM1.

Arithmetic Decomposition Method (AD):

- AD2 is similar to Fig. 3.2, but the decomposition coefficient is 3^6. In this case, we need a 12-input 20-output LUT, a 12-input 10-output LUT, and a 20-bit adder. The Quartus replaced these LUTs with 30 M4Ks, and the adder with 20 LEs. So, we had to use a larger FPGA, EP2C8T144C7 which contains 36 M4Ks.
- AD3 is similar to Fig. 3.3, but uses a pair of 12-input 19-output LUTs and a 20-bit adder. To replace these LUTs, the Quartus required 38 M4Ks. So, we had to use a larger FPGA, EP2C20F256C7 which contains 52 M4Ks.
- AD4 is similar to Fig. 3.4, but uses the decomposition coefficient 3^6. It uses four LUTs with 6 inputs. The Quartus replaced these LUTs with four M4Ks, and the adders with 57 LEs.

In the case of **12ter2bin**, AD2 and AD3 are faster, but require larger FPGAs, so AD4 is the best choice.

6 Conclusion and Comments

In this paper, we presented arithmetic decompositions to design p-nary to binary converter. We used ternary to binary converters to illustrate the idea. We also implemented the converts on FPGAs to confirm the effectiveness of the methods.

Note that Fig. 3.1 is a non-disjoint decomposition, while Fig. 3.2, Fig. 3.3, and Fig. 3.4 are disjoint decompositions. The disjoint decomposition in Fig. 3.2 is easy to find from equation (2.2) or Fig.2.1, while the disjoint decomposition in Fig. 3.3 is not so easy to find.

Also, the decomposition in Fig. 3.2 produces similar but different sub-circuits, while the decomposition in Fig. 3.3 produces two identical sub-circuits. The circuit in Fig. 3.3 is faster than Fig. 3.2.

These techniques can be combined to design radix converts with more digits, and other arithmetic circuits [3].

Acknowledgments

This research is supported in part by the Grant in Aid for Scientific Research of MEXT, and the Kitakyushu Area Innovative Cluster Project of MEXT.

References

[1] T. Hanyu and M. Kameyama, "A 200 MHz pipelined multiplier using 1.5 V-supply multiplevalued MOS current-mode circuits with dual-rail source-coupled logic," *IEEE Journal of Solid-State Circuits 30, 11*, (1995), 1239-1245.

[2] C. H. Huang, "A fully parallel mixed-radix conversion algorithm for residue number applications," *IEEE Trans. Comput.*, vol. 32, pp. 398-402, 1983.

[3] K. Ishida, N. Homma, T. Aoki, and T. Higuchi, "Design and verification of parallel multipliers using arithmetic description language: ARITH," *34th International Symposium on Multiple-Valued Logic*, Toronto, Canada, May 2004, pp.334-339.

[4] I. Koren, *Computer Arithmetic Algorithms, 2nd Edition*, A. K. Peters, Natick, MA, 2002.

[5] S. Muroga, *VLSI System Design*, John Wiley & Sons, 1982, pp. 293-306

[6] D. Olson, and K. W. Current, "Hardware implementation of supplementary symmetrical logic circuit structure concepts," *30th IEEE International Symposium on Multiple-Valued Logic* Portland, Oregon, May 23-25, 2000.

[7] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[8] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *International Workshop on Logic and Synthesis(IWLS01),* Lake Tahoe, CA, June 12-15, 2001, pp. 225-230.

[9] T. Sasao, "Radix converters: Complexity and implementation by LUT cascades," *35th International Symposium on Multiple-Valued Logic*, Calgary, Canada, May 19-21, 2005, pp.256-263.

[10] T. Sasao, Y. Iguchi, and T. Suzuki, "On LUT cascade realizations of FIR filters," *DSD2005, 8th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, Porto, Portugal, Aug. 30 - Sept. 3, 2005, pp.467-474.

[11] T. Sasao, "Analysis and synthesis of weighted-sum functions," *IEEE Trans. on CAD* (to be published).