

## PAPER

# Design Methods of Radix Converters Using Arithmetic Decompositions

Yukihiro IGUCHI<sup>†a)</sup>, Tsutomu SASAO<sup>††b)</sup>, and Munehiro MATSUURA<sup>††c)</sup>, *Members*

**SUMMARY** In arithmetic circuits for digital signal processing, radices 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. First, it considers Look-Up Table (LUT) cascade realizations. Then, it introduces a new design technique called arithmetic decomposition by using LUTs and adders. Finally, it compares the amount of hardware and performance of radix converters implemented by FPGAs. 12-digit ternary to binary converters on Cyclone II FPGAs designed by the proposed method are faster than ones by conventional methods.

**key words:** radix converter, LUT cascades, FPGA, functional decomposition

## 1. Introduction

Arithmetic operations of digital systems usually use radix two [9]. However, in digital signal processing, for high speed operations,  $p$ -nary ( $p > 2$ ) numbers are often used [2], [6]. In such cases, conversions between binary numbers and  $p$ -nary numbers are necessary. Such an operation is called **radix conversion** [3], [8]. Various methods exist to convert  $p$ -nary numbers into binary numbers. Many of them require large amount of computation. Especially when the radix conversion is implemented by a random logic circuit, the network tends to be quite complex [7]. 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 memory.

In [11], LUT cascade realizations [10] of binary to ternary converters, ternary to binary converters, binary to decimal converters, and decimal to binary converters are presented. In [13], 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 [12]. We also consider the implementations on field programmable gate arrays (FPGAs). For readability, we use examples for  $p = 3$ , however the meth-

ods can be easily extended to any prime number  $p$ .

A preliminary version of this paper was presented at ISMVL-2006 [4].

## 2. Radix Converter

### 2.1 Radix Conversion

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

$$\sum_{i=0}^{n-1} x_i p^i = \sum_{j=0}^{m-1} y_j q^j, \quad (1)$$

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

Let  $\vec{y} = (y_{m-1}, y_{m-2}, \dots, y_0)_2$ ,  $y_i \in \{0, 1\}$  be the output functions of  $p$ -nary to binary converter. Then, when  $p$  is a prime number,  $y_i$  depends on all the inputs  $x_i$  ( $i = 0, 1, \dots, n-1$ ). 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 0s to the undefined outputs.

**Example 1:** In the case of a 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 1 is the truth table of a two-digit ternary to binary converter. In the binary-coded-ternary representation, (11) is an undefined input, and the corresponding output is *don't care*. In Table 1, the inputs in the binary-coded-ternary representation are denoted by  $\vec{z} = (z_3, z_2, z_1, z_0)$ .

**Table 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

Manuscript received August 28, 2006.

Manuscript revised February 1, 2007.

<sup>†</sup>The author is with the Department of Computer Science, Meiji University, Kawasaki-shi, 214-8571 Japan.

<sup>††</sup>The authors are with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820-8502 Japan.

a) E-mail: iguchi@cs.meiji.ac.jp

b) E-mail: sasao@cse.kyutech.ac.jp

c) E-mail: matsuura@cse.kyutech.ac.jp

DOI: 10.1093/ietisy/e90-d.6.905

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

### 2.2.1 Realization by Random Logic

A straightforward way to implement a radix converter is to apply a logic synthesis tool directly to Eq. (1).

**Example 2:** Consider the 8-digit ternary to binary converter (8ter2bin). In this case, we realize

$$3^7 x_7 + 3^6 x_6 + 3^5 x_5 + 3^4 x_4 + 3^3 x_3 + 3^2 x_2 + 3^1 x_1 + 3^0 x_0. \quad (2)$$

Figure 1 shows the circuit for 8ter2bin produced by a logic synthesis tool. Note that  $3^2 x_2$  is implemented as  $8x_2 + x_2$  and  $3^1 x_1$  is implemented as  $2x_1 + x_1$ , but  $3^i x_i$  ( $i = 3, 4, \dots, 7$ ) 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 rather complex. (End of Example)

### 2.2.2 Realization by a Single Memory

The simplest realization uses a single memory that stores the truth table of the radix converter.

A  $p$ -nary number of  $n$ -digit takes values from 0 to  $p^n - 1$ . A binary number requires  $m = \lceil \log_2(p^n - 1) \rceil$  bits to represent the number. When the input is represented by a binary-coded  $p$ -nary number, let  $d$  be the number of bits for an input digit. Then, we have the relation:  $d = \lceil \log_2 p \rceil$ . In this paper, we will consider networks that convert  $p$ -nary numbers

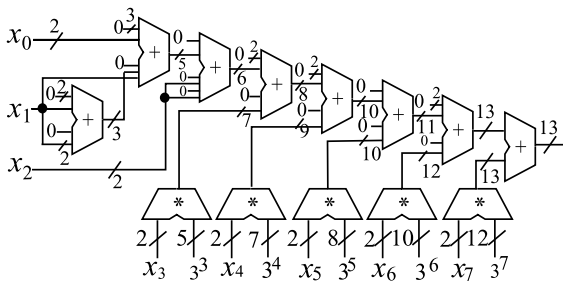


Fig. 1 8-digit ternary to binary converter: Direct method.

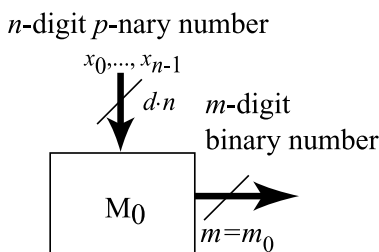


Fig. 2  $n$ -digit  $p$ -nary to binary converter: A single memory realization.

into binary numbers, where each input is represented by a binary-coded  $p$ -nary number. Let  $SIZE(n, p)$  be the number of bits to realize the network by a single memory. We have the relation:  $SIZE(n, p) = 2^{\lceil \log_2 p \rceil \cdot n} \cdot \lceil \log_2(p^n - 1) \rceil$ .

Figure 2 shows the network that converts  $n$ -digit binary-coded  $p$ -nary numbers into binary numbers.

**Example 3:** An 8-digit ternary to binary converter takes the range  $[0, 3^8 - 1] = [0, 6560]$ . The number of bits in the output is  $m = \lceil \log_2 6560 \rceil = 13$ . When it is realized by a single memory, the necessary number of bits is  $SIZE(8, 3) = 2^{2 \cdot 8} \cdot 13 = 851,968$ . (End of Example)

When the number of digits for the radix converter is large, the memory will be huge.

## 3. LUT Cascade Realization and WS Function

In a direct realization 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, LUT cascades realizations are used in [11], where outputs are partitioned into groups, and each group is synthesized by using functional decomposition.

From here, we will consider LUT cascades realization without partitioning outputs.

### 3.1 LUT Cascade Realization of a Radix Converter

A single memory realization of a radix converter is simple, but requires a large memory. We can reduce the total amount of memory by decomposing the memory. First, we consider the method to decompose the memory into  $M_0$  and  $M_1$ , shown in Fig. 3. In Fig. 3, the lower  $k$ -digits of a binary-coded  $p$ -nary number are connected to the inputs of  $M_0$ . Outputs from  $M_0$  and the upper  $n - k$  digits of the input are connected to the inputs of  $M_1$ . Outputs of  $M_1$  represent the converted binary number.

By using the functional decomposition theory [9], we can decide such realization is possible or not.

**Definition 2:** Consider the function  $f(\vec{X}) : P^n \rightarrow Q$ , where  $P = \{0, 1, \dots, p - 1\}$  and  $Q = \{0, 1, \dots, q\}$ . Let  $(\vec{X}_H, \vec{X}_L)$  be a partition of  $\vec{X}$ , where  $\vec{X}_H = (X_0, X_1, \dots, X_{k-1})$  and  $\vec{X}_L = (X_k, X_{k+1}, \dots, X_{n-1})$ . The decomposition chart for  $f$  is a two-dimensional matrix, where the column labels have all possible assignments of elements of  $X$  to  $\vec{X}_H$ , the row labels have all possible assignments of elements of  $X$  to  $\vec{X}_L$ ,

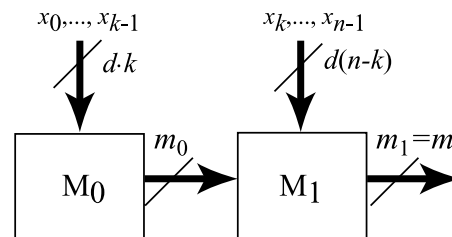


Fig. 3  $m$ -digit  $p$ -nary to binary converter: LUT cascade method.

and the corresponding matrix value is equal to  $f(\vec{X}_H, \vec{X}_L)$ . The one whose column label values and row label values increase when the label moves from left to right, and from top to bottom, is the standard decomposition chart. The number of different column patterns in the decomposition chart is the column multiplicity.

Note that in an ordinary decomposition chart, the partitions of variables and the order of labels in the columns and rows are arbitrary. However, in the standard decomposition chart, the labels of the columns are in increasing order of  $\vec{X}_H = (X_0, X_1, \dots, X_{k-1})$ , and the labels of the rows are in increasing order of  $\vec{X}_L = (X_k, X_{k+1}, \dots, X_{n-1})$  [11].

**Lemma 1:** Consider the function  $f(\vec{X})$  that represents the conversion from  $p$ -nary numbers into binary numbers. The column multiplicity of the standard decomposition chart for  $f(\vec{X})$  is  $p^k$ , where  $(\vec{X}_H, \vec{X}_L)$  is the partition of  $\vec{X}$ , and the number of variables in  $\vec{X}_H$  is  $k$ .

(Proof) In the standard decomposition chart for  $f$ , when we move from the left to the right, all values increase by one, and all values are different each other. The number of columns of this decomposition chart is  $p^k$ , and all columns have different patterns. Therefore, we have the lemma.

(Q.E.D.)

**Example 4:** Consider the converter from 4-digit ternary numbers into 7-digit binary numbers. Table 2 shows a standard decomposition chart of it. (End of Example)

### 3.2 WS Functions

The weighted sum function (WS function) is a mathematical model of radix converters, bit-counting circuits, and convolution operations [12], [13].

**Definition 3:** An  $n$ -input WS function [13] is defined as

$$WS(\vec{x}) = \sum_{i=0}^{n-1} w_i \cdot x_i, \tag{3}$$

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

**Table 2** Standard decomposition chart for a function of 4-digit ternary to binary conversion.

		0	0	0	1	1	1	2	2	2	$X_1$	$\vec{X}_H$
		0	1	2	0	1	2	0	1	2	$X_0$	
$\vec{X}_L$	$X_3$ $X_2$											
0	0	0	1	2	3	4	5	6	7	8		
0	1	9	10	11	12	13	14	15	16	17		
0	2	18	19	20	21	22	23	24	25	26		
1	0	27	28	29	30	31	32	33	34	35		
1	1	36	37	38	39	40	41	42	43	44		
1	2	45	46	47	48	49	50	51	52	53		
2	0	54	55	56	57	58	59	60	61	62		
2	1	63	64	65	66	67	68	69	70	71		
2	2	72	73	74	75	76	77	78	79	80		

In this paper, we represent a radix converter with a WS function, where inputs are represented by binary-coded  $p$ -nary numbers. From here, unless otherwise noted,  $w_i$  and  $x_i$  denote non-negative integers.

**Definition 4:** Let  $MIN_n$  ( $MAX_n$ ) be the minimum (maximum) number represented by an  $n$ -variable WS function  $WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i$ , where  $w_i \geq 0$ ,  $x_i \in \{0, 1, \dots, p-1\}$ , and  $p \geq 2$ .

When all the input  $x_i$  are 0's, a WS function takes the minimum value,  $MIN_n = WS(0, 0, \dots, 0) = 0$ . When all the input  $x_i$  are  $p-1$ , the WS function takes the maximum value,  $MAX_n = WS(p-1, p-1, \dots, p-1) = \sum_{i=0}^{n-1} \{w_i \cdot (p-1)\} = (p-1) \sum_{i=0}^{n-1} w_i$ .

**Definition 5:** Given  $i < j$ ,  $[i, j]$  denotes the set of integers,  $\{i, i+1, \dots, j\}$ .

Next, we will consider the range of WS functions.

**Definition 6:**  $Range(f(x))$  denotes the range of a function  $f(x)$ .

**Example 5:** For  $WS_1(\vec{x}) = x_0 + 3x_1$ , and  $x_i \in \{0, 1, 2\}$ , we have  $Range(x_0) = \{0, 1, 2\}$ , and  $Range(3x_1) = \{0, 3, 6\}$ . Therefore,  $Range(WS_1(\vec{x})) = \{0, 1, 2, 3, 4, 5, 6, 7, 8\} = [0, 8]$ . On the other hand, for  $WS_2(\vec{x}) = x_0 + 4x_1$ , and  $x_i \in \{0, 1, 2\}$ , we have  $Range(x_0) = \{0, 1, 2\}$ , and  $Range(4x_1) = \{0, 4, 8\}$ . Therefore,  $Range(WS_2(\vec{x})) = \{0, 1, 2, 4, 5, 6, 8, 9, 10\} \neq [0, 10]$ . For  $WS_3(\vec{x}) = x_0 + 2x_1$ , and  $x_i \in \{0, 1, 2\}$ , we have  $Range(x_0) = \{0, 1, 2\}$ , and  $Range(2x_1) = \{0, 2, 4\}$ . Therefore,  $Range(WS_3(\vec{x})) = \{0, 1, 2, 3, 4, 5, 6\} = [0, 6]$ . (End of Example)

Example 5 shows that  $Range(WS(\vec{x})) = [0, MAX_n]$  holds in some cases and does not in other cases. It depends on the combinations of values of  $w_i$ . In the case of  $WS_2(\vec{x}) = x_0 + 4x_1$ ,  $MIN = 0$ , and  $MAX = 2(1+4) = 10$ . Because the number of values for  $x_i$  is 3,  $WS_2(\vec{x})$  takes at most  $3^2 = 9$  different values. Therefore,  $Range(WS_2(\vec{x})) \neq [0, 10]$ .

**Lemma 2:** The number of values produced by a WS function,  $WS(\vec{x})$ , is at most  $p^n$ , where  $WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i$ ,  $w_i \geq 0$ ,  $x_i \in \{0, 1, \dots, p-1\}$ , and  $p \geq 2$ .

(Proof) Since the cardinality of the domain is  $p^n$ , the cardinality of the range is at most  $p^n$ . (Q.E.D.)

The next theorem shows the necessary and sufficient condition for  $Range(WS(\vec{x})) = [0, MAX_n]$ .

**Theorem 1:** Consider a WS function  $WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i$ ,  $w_i \geq 0$ ,  $x_i \in \{0, 1, \dots, p-1\}$ , and  $p \geq 2$ . The necessary and sufficient condition for  $Range(WS(\vec{x})) = [0, MAX_n]$  is  $w_0 = 1$  and  $w_i \leq MAX_i + 1$ .

(Proof) Assume that the elements of the weight vector are arranged in the ascending order.

First, we will show the sufficiency. Consider the one-variable function,  $WS(\vec{x}) = w_0 x_0$ . When  $w_0 = 1$ ,  $MAX_1 = p-1$ . Since  $WS(\vec{x})$  takes all the values in  $[0, p-1]$ , therefore, the theorem holds.

Next, assume that the theorem holds for  $k$ -variable WS

function,  $WS_k(\vec{x}) = \sum_{i=0}^{k-1} w_i x_i$ . That is if  $w_0 = 1$  and  $w_i < w_j$  ( $i < j$ ) then  $Range(WS_k(\vec{x})) = [0, MAX_k]$ , where  $i < j \leq k$ , and  $MAX_k = (p-1) \sum_{i=0}^{k-1} w_i$ .

Then, consider the  $(k+1)$ -variable WS function,  $WS_{k+1}(\vec{x}) = \sum_{i=0}^k w_i x_i$ . Note that  $WS_{k+1}(\vec{x}) = \sum_{i=0}^k w_i x_i = (\sum_{i=0}^{k-1} w_i x_i) + (w_k x_k)$ . Since,  $Range(w_k x_k) = \{0, w_k, 2w_k, \dots, (p-1)w_k\}$ , the range for  $WS_{k+1}(\vec{x}) = \sum_{i=0}^k w_i x_i$  is  $[0, MAX_k] \cup [w_k, w_k + MAX_k] \cup [2w_k, 2w_k + MAX_k] \cup \dots \cup [(p-1)w_k, (p-1)w_k + MAX_k]$ . From the hypothesis of induction, we have  $w_k \leq MAX_k + 1$ . Hence, we have  $Range(WS_{k+1}(\vec{x})) = [0, (p-1) \cdot w_k + MAX_k] = [0, MAX_{k+1}]$ . Thus, the theorem holds.

Next, we will show the necessity. If  $w_0 \geq 2$ ,  $MAX_1 = (p-1)w_0$ . On the other hand, since  $x_0$  can take only  $p$  values,  $Range(WS(\vec{x})) \neq [0, MAX_n]$ . If  $w_i > MAX_i + 1$ ,  $WS(\vec{x})$  cannot represent  $MAX_i + 1$ . Therefore,  $Range(WS(\vec{x})) \neq [0, MAX_n]$ . Hence, we have the theorem. (Q.E.D.)

**Corollary 1:**  $WS(\vec{x}) = \sum_{i=0}^{n-1} p^i x_i$  takes all the value in  $[0, p^n - 1]$ .

(Proof)  $w_0 = p^0 = 1$ ,  $MAX_k = \sum_{i=0}^{k-1} (p^i (p-1)) = (p-1) \sum_{i=0}^{k-1} p^i = (p-1) \frac{p^k - 1}{p-1} = p^k - 1$ . Since  $w_k = p^k = MAX_k + 1$ ,  $Range(WS(\vec{x})) = [0, p^n - 1]$  by Theorem 1. (Q.E.D.)

From here, we will consider LUT cascade realizations for radix converter, where total amount of memory is the minimum. Figure 4 shows an LUT cascade realization of an  $n$ -digit  $p$ -nary to binary converter, where each LUT has only one external input  $x_i$ . In this case,  $i$ -th LUT ( $0 \leq i \leq n-1$ ) produces values in  $[0, p^{i+1} - 1]$ ,  $r_i = \lceil (i+1) \log_2 p \rceil$ -output lines is necessary and sufficient. To find the minimum cascades, we have to consider the cases where adjacent LUTs are merged and not. The number of different combinations is  $2^{n-1}$ . In this case, input variable  $x_i$  incidents to each  $LUT_i$ . Each LUT has  $d = \lceil \log_2 p \rceil$  two-valued inputs. With these constraints, we have  $2^{n-1}$  combinations for all LUT cascades realizations.

**Theorem 2:** Under the fixed variable ordering, we can obtain the minimum LUT cascade realization of  $n$ -digit radix converter by checking at most  $2^{n-1}$  combinations. (Proof : Omitted)

**Example 6:** Figure 5 shows all possible realizations of a 4-digit ternary to binary converter by LUT cascades. We have  $2^{4-1} = 8$  different realizations shown in Fig. 5 (0) – (7). The number in each LUT denotes the size of memory.

(0) is the realization using 4 LUTs, where the total amount of memory is  $8 + 64 + 320 + 896 = 1288$  (bits). (1) is

obtained by merging the leftmost two LUTs in (0). (7) is obtained by merging all the LUTs in (0). Among these 8 combinations, (3) is the minimum realization. (End of Example)

The next lemma shows a method to detect mergible LUTs in an LUT cascade.

**Lemma 3:** Consider the LUT cascade in Fig. 6, where LUT H has  $k$ -input and  $k$ -output. In this case, without loss of minimality, LUTs H and G can be merged into  $G'$ . (Proof : Omitted)

By using Lemma 3, we can find the mergible LUTs, and reduce the number of combinations to find the minimum LUT cascades. In the case of Example 6, we can see that the leftmost two LUTs in (0) can be merged, to obtain the LUT cascade in (1). Next, by merging the leftmost two LUTs in (1), we have the LUT cascade in (3). In (3), the number of LUTs is two. So, we need only to consider the cascades where these two LUTs are merged and not.

The next algorithm finds the LUT cascade for the radix converters with the minimum amount of memory.

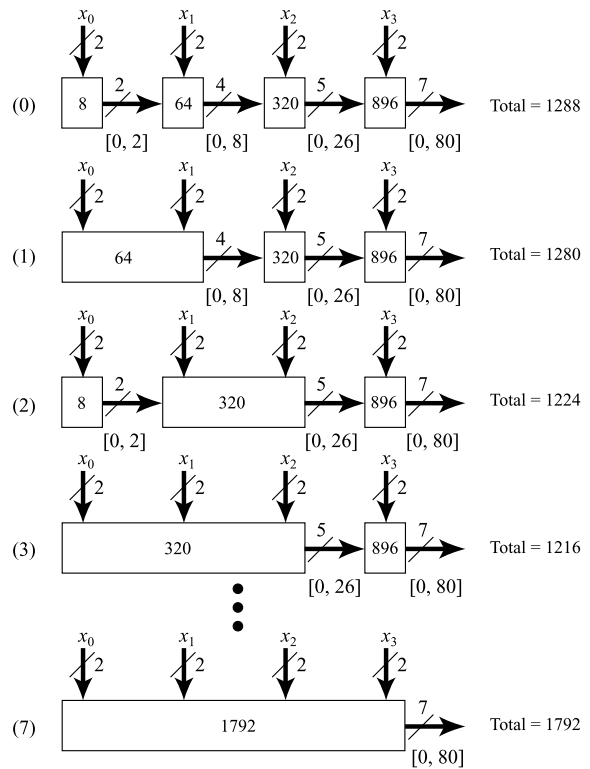


Fig. 5 All possible LUT cascade realizations for 4ter2bin.

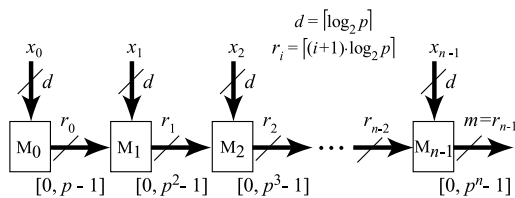


Fig. 4 LUT cascade realization of a radix converter.

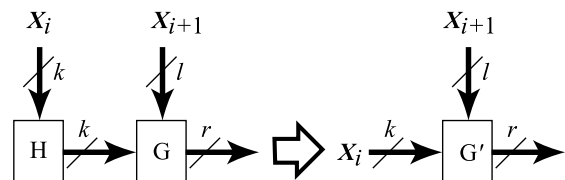


Fig. 6 Mergible LUT.

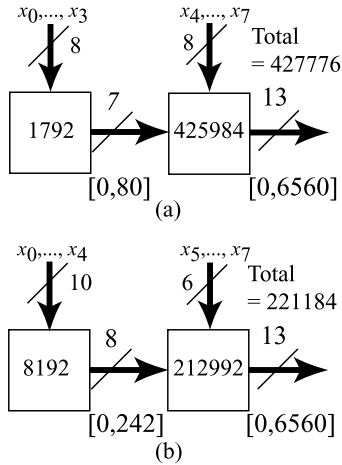


Fig. 7 8-digit ternary to binary converter: LUT cascade realization.

**Algorithm 1:**

1. Obtain the LUT cascade for the radix converter where  $LUT_i$  has only one external input  $x_i$ , ( $i = 0, 1, \dots, n-1$ ).
2. Apply Lemma 3, and merge the LUT whose number of inputs is equal to the number of outputs with the succeeding LUT to make a new LUT, and produce the data for a new LUT. Iterate this step while we can.
3. Let  $s$  be the number of LUTs in the LUT cascade obtained by the previous step. The number of combinations to merge adjacent LUTs or not is  $2^{s-1}$ . Calculate the sizes of these cascades, and select the LUT cascade that has the minimum memory among  $2^{s-1}$  realizations.

**Example 7:** Figure 7(a) and (b) show two LUT cascade realizations for 8ter2bin shown in Example 3. While a single memory realization needs 851,968-bit memory, Fig. 7(a) requires 427,776 bits, and Fig. 7(b) requires only 221,184 bits.

By partitioning outputs into plural groups, we can realize radix converters with plural LUT cascades [11]. This method uses binary decision diagrams to find functional decompositions. Example 8 shows LUT cascades realizations obtained by partitioning outputs.

**Example 8:** Realize an 8-digit ternary to binary converter (8ter2bin) by partitioning outputs [11]. Assume that we use LUTs with 10 inputs. Figure 8 shows the cascade realization. The 13 outputs are divided into two groups: The upper cascade realizes the least significant 6 bits, while the lower cascade realizes the most significant 7 bits. The total amount of memory is  $2^{10}(7 + 7 + 7 + 6 + 6) + 2^8(6) = 35,328$  bits. (End of Example)

To find an optimal solution for the radix converter with many digits, the method [11] requires a large amount of computation. On the other hand, Algorithm 1 requires only small amount of computation. In the next section, we will present the more compact realizations of the radix converter by using arithmetic decomposition [12].

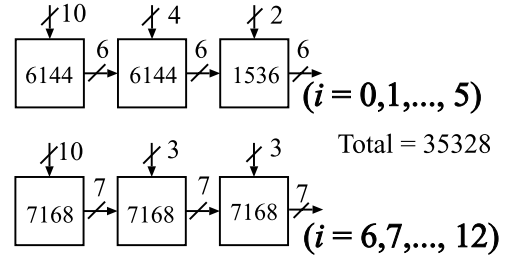


Fig. 8 8-digit ternary to binary converter: Outputs are partitioned into two groups.

**4. Realization Based on Arithmetic Decomposition**

4.1 Arithmetic Decompositions of WS Functions

In the previous section, we presented a design method of radix converters by using LUT cascades. In this section, we will propose design methods of radix converters by using arithmetic decompositions [12].

**Theorem 3:** A WS function can be represented as a sum of two WS functions as follows:

$$WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i = \alpha WS_A(\vec{x}) + WS_B(\vec{x}), \tag{4}$$

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  $\alpha$  is an integer. This is the arithmetic decomposition, and  $\alpha$  is the decomposition coefficient.

$\alpha$  can be an arbitrary integer, where  $2 \leq \alpha < MAX_n$ . Let  $MAX^B$  be the maximum value of  $WS_B(\vec{x})$ .  $\alpha$  need not satisfy  $MAX^B < \alpha$ , however, in this paper, we consider only cases where  $MAX^B < \alpha$ .

Next, we will consider properties of WS functions obtained by arithmetic decompositions of the WS function representing a radix converter.

**Theorem 4:** Consider the arithmetic decomposition for a  $WS(\vec{x})$  which represents a radix converter, where  $WS(\vec{x}) = \sum_{i=0}^{n-1} w_i x_i = \alpha WS_A(\vec{x}) + WS_B(\vec{x})$ ,  $\alpha$  is an integer,  $2 \leq \alpha < p^n - 1$ ,  $WS_A(\vec{x}) = \sum_{i=0}^{n-1} w_i^A x_i$ , and  $WS_B(\vec{x}) = \sum_{i=0}^{n-1} w_i^B x_i$ . In this case,  $Range(WS_B(\vec{x})) = [0, (p-1) \sum_{i=0}^{n-1} w_i^B]$ .

(Proof)  $WS_B(\vec{x})$  is a remainder, which is obtained by dividing  $WS(\vec{x})$  by  $\alpha$ . Assume that  $Range(WS_B(\vec{x})) \neq [0, (p-1) \sum_{i=0}^{n-1} w_i^B]$ , then  $Range(WS(\vec{x})) \neq [0, p^{n-1}]$  holds. This contradicts the hypothesis that  $WS(\vec{x})$  represents the radix conversion. Hence, the theorem. (Q.E.D.)

4.2 Arithmetic Decompositions for Different Decomposition Coefficients

A radix converter can be represented as a WS function. By using arithmetic decompositions with different decomposition coefficients  $\alpha$ , we can realize different 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.

(1) 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 realized by the shifting, it is realized compactly. However,  $WS_B$  depends on all the input variables, and the total size of the circuit is not so small.

(2) 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, when  $k = n/2$ , the numbers of inputs for  $WS_A$  and  $WS_B$  can be a half of the original function, so the total network can be much smaller.

**Example 9:** Let us design the 8-digit ternary to binary converter (8ter2bin). Consider two cases where the decomposition coefficients are  $\alpha = 2^6 = 64$  and  $\alpha = 3^4 = 81$ . The ternary number is represented by the binary-coded ternary code. Table 3 shows the coefficients of arithmetic decompositions of  $3^i$  ( $i = 0, 1, 2, \dots, 7$ ). 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, we have two different realizations for 8ter2bin.

1. 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  $\{x_4, x_5, x_6, x_7\}$ . 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.

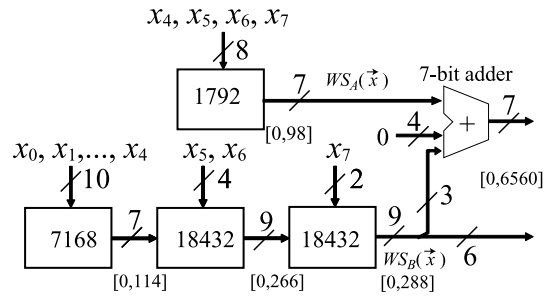
**Table 3** Coefficients of arithmetic decompositions of the powers of 3.

		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	$64 \times 34 + 11$	$81 \times 27 + 0$

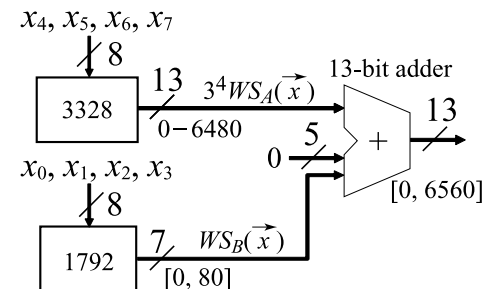
- $WS_B(\vec{x})$  depends on all the inputs  $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ , so the number of inputs is 16. Since each weight vector  $w_i$  satisfies the condition of Theorem 1, the output range is  $[0, 2(1 + 3 + 9 + 27 + 17 + 51 + 25 + 11)] = [0, 288]$ , 9 bits are necessary to represent it.  $WS_B(\vec{x})$  has 16 inputs and 9 outputs. It is implemented by a 3-LUTs cascade with 11-input cells.
- The multiplication by  $2^6$  can be 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. Figure 9 shows the network, which uses memory with 45824 bits and a 7-bit adder.

2. When the decomposition coefficient is  $3^4$ .

- $WS(\vec{x}) = 3^4 WS_A(\vec{x}) + WS_B(\vec{x})$ .
- $WS_A(\vec{x}) = 3^3 x_7 + 3^2 x_6 + 3^1 x_5 + 3^0 x_4$ .
- $WS_B(\vec{x}) = 3^3 x_3 + 3^2 x_2 + 3^1 x_1 + 3^0 x_0$ .
- $WS_A(\vec{x})$  depends on inputs  $\{x_4, x_5, x_6, x_7\}$ , so the number of the inputs is 8. By Theorem 1, the maximum value of the output is  $2(1 + 3 + 9 + 27) = 80$ . It can be represented by 7 bits. Thus,  $WS_A(\vec{x})$  can be implemented by an 8-input 7-output LUT.
- $WS_B(\vec{x})$  depends on inputs  $\{x_0, x_1, x_2, x_3\}$ , so the number of inputs is 8. By Theorem 1, the output range is  $[0, 2(1 + 3 + 9 + 27)] = [0, 80]$ .
- The maximum value of  $3^4 WS_A$  is 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. Figure 10 shows the network, which uses memory



**Fig. 9** 8-digit ternary to binary converter: Arithmetic decomposition with coefficient  $2^6$ .



**Fig. 10** 8-digit ternary to binary converter: Arithmetic decomposition with coefficient  $3^4$ .

with 5120 bits and a 13-bit adder.

(End of Example)

### 4.3 Arithmetic Decomposition Using the Binary Representation of Inputs

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

**Definition 7:** 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 10:**  $BIT(2, 1) = 1$ ,  $BIT(2, 0) = 0$ ,  $BIT(1, 1) = 0$ , and  $BIT(1, 0) = 1$ .

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

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

From this, we have the following:

**Theorem 5:** A  $p$ -nary to binary converter can be represented by

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

(Proof)

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

(Q.E.D.)

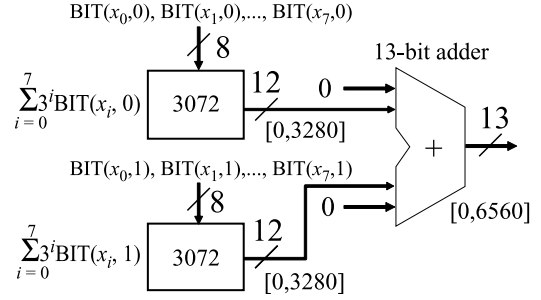
**Example 11:** Consider the 8-digit ternary to binary converter (8ter2bin). By Theorem 5,  $WS(\vec{x})$  can be represented as:

$$WS(\vec{x}) = 2 \sum_{i=0}^7 3^i BIT(x_i, 1) + \sum_{i=0}^7 3^i BIT(x_i, 0). \quad (5)$$

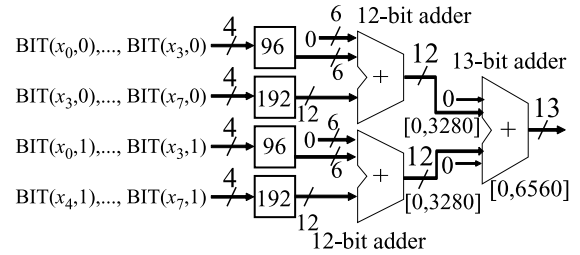
Figure 11 shows the circuit corresponding to the above decomposition. Each cell has 8 inputs. Since  $\sum_{i=0}^7 3^i = 3280$ , each cell has 12 outputs. The multiplication by two is implemented by shifting one bit position. The circuit uses 6144 bits of memories and a 13-bit adder.

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

$$\begin{aligned} WS(\vec{x}) &= 2 \left[ 3^4 \sum_{i=4}^7 3^{i-4} BIT(x_i, 1) + \sum_{i=0}^3 3^i BIT(x_i, 1) \right] \end{aligned}$$



**Fig. 11** 8-digit ternary to binary converter: Decomposed using the binary representation of inputs.



**Fig. 12** 8-digit ternary to binary converter: Decomposed using the binary representation of inputs and further decomposed with coefficient  $3^4$ .

$$+ \left[ 3^4 \sum_{i=4}^7 3^{i-4} BIT(x_i, 0) + \sum_{i=0}^3 3^i BIT(x_i, 0) \right]. \quad (6)$$

Figure 12 is the network corresponding Eq. (6), where each cell has only 4 inputs. The total amount of memory is 576 bits. It uses two 12-bit adders and a 13-bit adder.

(End of Example)

## 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. 13 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 4 compares 7 different designs of 8-digit ternary to binary converters (8ter2bin).

1. Direct Method (DM): The system generated a Verilog-

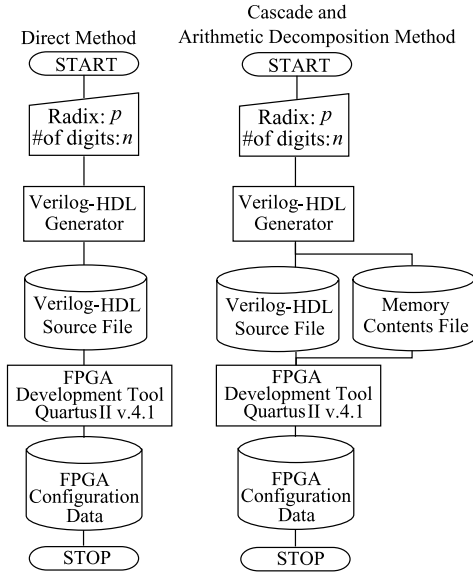


Fig. 13 Development system for radix converters.

Table 4 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. 1	66	0	7	26.7
DM2	W/O EM	Fig. 1	195	0	0	23.7
AD1	$2^6$	Fig. 9	8	13	0	30.0
AD2	$3^4$	Fig. 10	13	2	0	14.8
AD3	BIT	Fig. 11	12	2	0	14.3
AD4	BIT+ $3^4$	Fig. 12	36	4	0	16.8
PAR	M4K only	Fig. 8	0	11	0	22.2

HDL code from the specification: radix  $p$  and the number of digits  $n$ . We could not implement a radix converter by a single memory because size of memory was too large for this FPGA.

- DM1 directly implements  $\sum_{i=0}^7 3^i x_i$ . Figure 1 is the circuit generated by the Quartus II. After mapping, the Quartus II replaced the multipliers with 7 EMs, and adders with 66 LEs. It uses no M4Ks.
  - DM2 also corresponds to Fig. 1. In this case, however, the Quartus II replaced multipliers with LEs instead of EMs. So, the circuit consists of LEs only. It has 195 LEs, which means 129 LEs were replaced by 7 EMs. It is faster than DM1, since LEs perform constant multiplications faster than EMs.
2. Arithmetic Decomposition Method (AD): The system generated Verilog-HDL codes and data for M4Ks.
- AD1 corresponds to Fig. 9, which was obtained with the decomposition coefficient  $2^6$ . The Quartus II replaced four LUTs with 13 M4Ks, and the adder with 8 LEs.
  - AD2 corresponds to Fig. 10, which was obtained

Table 5 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. 1	139	0	15	35.8
DM2	W/O EM	Fig. 1	457	0	0	32.0
AD2	$3^6$	Fig. 10	20	30†	0	17.3
AD3	BIT	Fig. 11	19	38‡	0	16.6
AD4	BIT+ $3^6$	Fig. 12	57	4	0	17.6

†:Used EP2C8T144C7

‡:Used EP2C20F256C7

with the decomposition coefficient  $3^4$ . The Quartus II replaced two LUTs with two M4Ks, and the adder with 13 LEs.

- AD3 corresponds to Fig. 11, which was obtained with the arithmetic decomposition using binary representation of inputs. The Quartus II replaced two LUTs with two M4Ks, and the adder with 12 LEs.
- AD4 corresponds to Fig. 12, which was obtained by the arithmetic decomposition with the coefficient  $3^4$  and using binary representation of inputs. The Quartus II replaced four LUTs with four M4Ks, and adders with 36 LEs. It is slower than AD3 since the adder is more complex.

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

- PAR corresponds to Fig. 8, which consists of 6 LUTs. The Quartus II 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 the smallest amount of hardware.

### 5.3 12-Digit Ternary to Binary Converters

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

1. Direct Method (DM):

- DM1 is similar to Fig. 1, but uses 15 EMs.
- DM2 is similar to Fig. 1. Also in this case, it is faster than DM1.

2. Arithmetic Decomposition Method (AD):

- AD2 is similar to Fig. 10, but the decomposition coefficient is  $3^6$ . In this case, it uses a 12-input 20-output LUT, a 12-input 10-output LUT, and a 20-bit adder. The Quartus II replaced these LUTs with 30 M4Ks, and the adder with 20 LEs. So, it requires a larger FPGA, EP2C8T144C7 which contains 36 M4Ks.
- AD3 is similar to Fig. 11, but uses a pair of 12-input 19-output LUTs and a 20-bit adder. The

Quartus II replaced these LUTs with 38 M4Ks. So, we had to use a larger FPGA, EP2C20F256C7 which contains 52 M4Ks.

- AD4 is similar to Fig. 12, but uses the decomposition coefficient  $3^6$ . It uses four LUTs with 6 inputs. The Quartus II replaced these LUTs with four M4Ks, and the adders with 57 LEs.

Since AD1 uses too many M4Ks and PAR requires too much computation time, they are not used in the design. In the case of 8ter2bin, AD2 and AD3 are faster. On the other hand, in the case of 12ter2bin, AD2 and AD3 require larger FPGAs, so AD4 is the best choice.

Almost all embedded memories in recent FPGAs are synchronous. M4Ks in Cyclone II FPGAs are also synchronous. So, the circuits require clock signals to operate. Therefore, when we realize radix converters by using LUT cascades, we require at least  $s$  clocks to convert a binary number from a  $p$ -nary number. Delay [nsec] in Table 4 and 5 shows latency.

AD4 is the best choice for implementing radix converters with 12 digits.

#### 5.4 Observations

1. In Fig. 9, variables  $x_4$ ,  $x_5$ ,  $x_6$ , and  $x_7$  appear in both the upper and the lower cascades. This decomposition is a non-disjoint [1]. On the other hand, in Figs. 10, 11, and 12, each variable appears only once. These decompositions are disjoint [1]. The disjoint decomposition in Fig. 10 is easy to find from Eq. (2) or Fig. 1, while the disjoint decomposition in Fig. 11 is not so easy to find. Also, the decomposition in Fig. 10 produces similar but different sub-circuits, while the decomposition in Fig. 11 produces two identical sub-circuits.
2. These techniques can be combined to design radix converters with more digits, and other arithmetic circuits [5].
3. Since the propagation delay of an adder and an LUT are almost the same, we can achieve higher throughput for AD2, AD3, and AD4 by pipelining them.

## 6. Conclusion

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. An interesting future work is the extension to radix converters for signed-digit numbers.

### Acknowledgements

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

### References

- [1] H.A. Curtis, *A New Approach to the Design of Switching Circuits*, D. Van Nostrand Company, 1962.
- [2] T. Hanyu and M. Kameyama, "A 200 MHz pipelined multiplier using 1.5 V-supply multileveled MOS current-mode circuits with dual-rail source-coupled logic," *IEEE J. Solid-State Circuits*, vol.30, no.11, pp.1239–1245, 1995.
- [3] C.H. Huang, "A fully parallel mixed-radix conversion algorithm for residue number applications," *IEEE Trans. Comput.*, vol.32, no.4, pp.398–402, 1983.
- [4] Y. Iguchi, T. Sasao, and M. Matsuura, "On design of radix converters using arithmetic decompositions," 36th International Symposium on Multiple-Valued Logic, p.3, Singapore, May 2006.
- [5] 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, pp.334–339, Canada, May 2004.
- [6] I. Koren, *Computer Arithmetic Algorithms*, 2nd ed., A.K. Peters, Natick, MA, 2002.
- [7] S. Muroga, *VLSI System Design*, pp.293–306, John Wiley & Sons, 1982.
- [8] D. Olson and K.W. Current, "Hardware implementation of supplementary symmetrical logic circuit structure concepts," 30th IEEE International Symposium on Multiple-Valued Logic, pp.371–376, Portland, Oregon, May 2000.
- [9] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.
- [10] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," International Workshop on Logic and Synthesis (IWLS01), pp.225–230, Lake Tahoe, CA, June 2001.
- [11] T. Sasao, "Radix converters: Complexity and implementation by LUT cascades," 35th International Symposium on Multiple-Valued Logic, pp.256–263, Calgary, Canada, May 2005.
- [12] 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, pp.467–474, Porto, Portugal, Aug.-Sept. 2005.
- [13] T. Sasao, "Analysis and synthesis of weighted-sum functions," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol.25, no.5, pp.789–796, May 2006.



**Yukihiko Iguchi** was received the B.E., M.E., and Ph.D. degrees in electronic engineering from Meiji University, Kanagawa Japan, in 1982, 1984, and 1987, respectively. He is now an associate professor of Meiji University. His research interest includes logic design, switching theory, and reconfigurable systems. In 1996 and 2006, he spent each year at Kyushu Institute of Technology. He received Takeda Techno-Entrepreneurship Award in 2001.



**Tsutomu Sasao** was received the B.E., M.E., and Ph.D. degrees in Electronics Engineering from Osaka University, Osaka Japan, in 1972, 1974, and 1977, respectively. He has held faculty/research positions at Osaka University, Japan, IBM T.J. Watson Research Center, Yorktown Height, NY and the Naval Postgraduate School, Monterey, CA. He has served as the Director of the Center for Microelectronic Systems at the Kyushu Institute of Technology, Iizuka, Japan. Now, he is a Professor of Department of Computer Science and Electronics, His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than 8 books on logic design including, Logic Synthesis and Optimization, Representation of Discrete Functions, Switching Theory for Logic Synthesis, and Logic Synthesis and Verification, Kluwer Academic Publishers 1993, 1996, 1999, respectively. He has served Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan in 1998. He received the NIWA Memorial Award in 1979, Takeda Techno-Entrepreneurship Award in 2001, and Distinctive Contribution Awards from IEEE Computer Society MVL-TC for papers presented at ISMVLs in 1987, 1996, 2003 and 2004. He has served an associate editor of the IEEE Transactions on Computers. He is a Fellow of the IEEE.

department of Computer Science and Electronics, His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than 8 books on logic design including, Logic Synthesis and Optimization, Representation of Discrete Functions, Switching Theory for Logic Synthesis, and Logic Synthesis and Verification, Kluwer Academic Publishers 1993, 1996, 1999, respectively. He has served Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan in 1998. He received the NIWA Memorial Award in 1979, Takeda Techno-Entrepreneurship Award in 2001, and Distinctive Contribution Awards from IEEE Computer Society MVL-TC for papers presented at ISMVLs in 1987, 1996, 2003 and 2004. He has served an associate editor of the IEEE Transactions on Computers. He is a Fellow of the IEEE.



**Munehiro Matsuura** was born in Kitakyushu City, Japan. He studied at the Kyushu Institute of Technology from 1983 to 1989, and received the B.E. degree from the University of the Air, in Japan, 2003. He has been working as a Technical Assistant at the Kyushu Institute of Technology since 1991. He has implemented several logic design algorithms under the direction of Professor Tsutomu Sasao. His interests include decision diagrams and exclusive-OR based circuit design.