Short Papers

Analysis and Synthesis of Weighted-Sum Functions

Tsutomu Sasao

***Abstract*—A weighted-sum (WS) function computes the sum of selected integers. This paper considers a design method for WS functions by look-up table (LUT) cascades. In particular, it derives upper bounds on the column multiplicities of decomposition charts for WS functions. From these, the size of LUT cascades that realize WS functions can be estimated. The arithmetic decomposition of a WS function is also shown. With this method, a WS function can be implemented with cascades and adders.**

***Index Terms*—Binary decision diagram, column multiplicity, complexity of logic functions, digital filter, distributed arithmetic, field programmable gate array (FPGA), functional decomposition, LUT cascades, radix converter, symmetric function, threshold function.**

I. Introduction

A weighted-sum (WS) function computes the sum of selected integers $\mathrm{WS}(x_0, x_1, \ldots, x_{n-1}) = \sum_{i=0}^{n-1} w_i x_i$, where w_i are integer weights and x_i are binary variables. The WS function is a mathematical model of various computations: bit counting circuits, radix converters [10], distributed arithmetic (DA) for convolution operation, etc.

The look-up table (LUT) cascade has a regular programmable structure that implements many practical functions efficiently [7], [8].

In this paper, we derive upper bounds on the column multiplicities of decomposition charts for WS functions. With these bounds, we can estimate the size of a circuit for the consecutive outputs of the WS function and efficiently realize WS functions with LUT cascades.

This paper is organized as follows. Section II defines WS functions and shows the properties of WS functions. Section III shows the method to implement WS functions with LUT cascades. Section IV shows applications of WS functions. Section V shows the arithmetic decomposition of the WS function. Section VI concludes this paper.

II. WS Functions

A **WS function** is a mathematical model of bit counting circuits, code converters, DA, etc.

Definition 2.1: An n-input WS function $\vec{F}(\vec{X})$ computes

$$\mathrm{WS}(\vec{X}) = \sum_{i=0}^{n-1} w_i x_i. \tag{2.1}$$

Here, $\vec{X} = (x_0, x_1, \ldots, x_{n-1})$ is a binary **input vector** and $\vec{W} = (w_0, w_1, \ldots, w_{n-1})$ is the **weight vector**, where w_i $(i = 0, 1, \ldots, n-1)$ is an integer. Let $\vec{F} = (f_{q-1}, f_{q-2}, \ldots, f_0)$ be the binary representation of the WS function. Then

$$\mathrm{WS}(\vec{X}) = \sum_{i=0}^{q-1} f_i(\vec{X}) 2^i. \tag{2.2}$$

TABLE I
EXAMPLE OF WS FUNCTION

4	2	2	1					
x_3	x_2	x_1	x_0	$WS(X)$	f_3	f_2	f_1	f_0
0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	1
0	0	1	0	2	0	0	1	0
0	0	1	1	3	0	0	1	1
0	1	0	0	2	0	0	1	0
0	1	0	1	3	0	0	1	1
0	1	1	0	4	0	1	0	0
0	1	1	1	5	0	1	0	1
1	0	0	0	4	0	1	0	0
1	0	0	1	5	0	1	0	1
1	0	1	0	6	0	1	0	0
1	0	1	1	7	0	1	0	1
1	1	0	0	6	0	1	0	0
1	1	0	1	7	0	1	0	1
1	1	1	0	8	1	0	0	0
1	1	1	1	9	1	0	0	1

Example 2.1: Consider the case where $n = 4$ and $\vec{W} = (w_0, w_1, w_2, w_3) = (1, 2, 3, 4)$. Table I shows $\vec{X}$, $\mathrm{WS}(\vec{X})$, and $\vec{F} = (f_3, f_2, f_1, f_0)$.

Definition 2.2: Consider a function $\vec{F}(\vec{X}) : B^n \to B^q$, where $B = \{0, 1\}$. Let $(\vec{X}_L, \vec{X}_H)$ be a partition of $\vec{X}$, where $\vec{X}_L = (x_0, x_1, \ldots, x_{n_L-1})$ and $\vec{X}_H = (x_{n_L}, x_{n_L+1}, \ldots, x_{n-1})$. The **decomposition chart** for f is a two-dimensional matrix where the column labels have all possible assignments of values to variables in $\vec{X}_L$, the row labels have all possible assignments of values to variables in $\vec{X}_H$, and the corresponding matrix value is equal to $\vec{F}(\vec{X}_L, \vec{X}_H)$. Among the decomposition charts for $\vec{F}$, 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**. $\vec{X}_L$ denotes **bound variables**, while $\vec{X}_H$ denotes **free variables** [9].

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 $\vec{X}_L = (x_0, x_1, \ldots, x_{n_L-1})$ and the labels of the rows are in increasing order of $\vec{X}_H = (x_{n_L}, x_{n_L+1}, \ldots, x_{n-1})$.

Example 2.2: Table II shows an example of a decomposition chart for $n = 5$, where $\vec{X}_L = (x_0, x_1, x_2)$ and $\vec{X}_H = (x_3, x_4)$. Suppose that $q = 2$, that is, only two least significant bits are considered. Note that each element is a binary vector of 2 bits. In this case, only four different vectors can exist. So, in the first row of the decomposition chart, that is, the row for $(x_3, x_4) = (0, 0)$, at least two elements are equal. Suppose that the values for the columns $(x_0, x_1, x_2) = (0, 1, 1)$ and $(x_0, x_1, x_2) = (1, 0, 0)$ are equal: $w_1 + w_2 = w_0$. This

Manuscript received June 26, 2005; revised September 28, 2005. This work was supported in part by the Grants in Aid for Scientific Research of Japan Society for the Promotion of Science (JSPS) and Ministry of Education, Culture, Sports, Science and Technology (MEXT) and by a Kitakyushu Innovative Cluster Project grant. This paper was presented in part at the International Workshop on Logic and Synthesis, 2005. This paper was recommended by Guest Editor R. I. Bahar.

The author is with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka 820-8502, Japan (e-mail: sasao@cse.kyutech.ac.jp).

Digital Object Identifier 10.1109/TCAD.2006.870407

TABLE II
DECOMPOSITION CHART FOR WS FUNCTION

$\vec{X}_H = (x_3, x_4)$ \ $\vec{X}_L = (x_0, x_1, x_2)$	000	001	010	011	100	101	110	111
00	0	w_2	w_1	$w_1 + w_2$	w_0	$w_0 + w_2$	$w_0 + w_1$	$w_0 + w_1 + w_2$
01	w_4	w_2+ w_4	w_1+ w_4	$w_1 + w_2+$ w_4	w_0+ w_4	$w_0 + w_2+$ w_4	$w_0 + w_1+$ w_4	$w_0 + w_1 + w_2+$ w_4
10	w_3	w_2+ w_3	w_1+ w_3	$w_1 + w_2+$ w_3	w_0+ w_3	$w_0 + w_2+$ w_3	$w_0 + w_1+$ w_3	$w_0 + w_1 + w_2+$ w_3
11	$w_3 + w_4$	w_2+ $w_3 + w_4$	w_1+ $w_3 + w_4$	$w_1 + w_2+$ $w_3 + w_4$	w_0+ $w_3 + w_4$	$w_0 + w_2+$ $w_3 + w_4$	$w_0 + w_1+$ $w_3 + w_4$	$w_0 + w_1 + w_2+$ $w_3 + w_4$

implies that in the second row of the decomposition chart, that is, in the row for $(x_3, x_4) = (0, 1)$, the corresponding two elements are equal: $w_1 + w_2 + w_4 = w_0 + w_4$. This is obvious since the same numbers are added to both sides of the equation. In similar ways, we can show that in the remaining rows, the entries for the columns $(x_0, x_1, x_2) = (0, 1, 1)$ and $(x_0, x_1, x_2) = (1, 0, 0)$ are equal. That is, if the two elements in the first row are equal, then the patterns of the two columns are the same. Hence, we can see that the column patterns for $(x_0, x_1, x_2) = (0, 1, 1)$ and $(x_0, x_1, x_2) = (1, 0, 0)$ are the same.

From the above example we have the following.

Lemma 2.1: The column multiplicity of a decomposition chart of a WS function is equal to the number of different elements in the first row.

Furthermore, we have the following.

Lemma 2.2: The column multiplicity of the decomposition chart for a q-output WS function is at most 2^q.

Proof: Let $\vec{F}(x_0, x_1, \ldots, x_{n-1})$ be an n-input q-output WS function. Let $(\vec{X}_L, \vec{X}_H)$ be a partition of $\vec{X} = (x_0, x_1, \ldots, x_{n-1})$, where $\vec{X}_L = (x_0, x_1, \ldots, x_{n_L-1})$ and $\vec{X}_H = (x_{n_L}, x_{n_l+1}, \ldots, x_{n-1})$. Consider the decomposition chart of $\vec{F}$, where $\vec{X}_L$ denotes the bound variables and $\vec{X}_H$ denotes the free variables.

When $n_L \leq q$, there can be no more than 2^q columns, and the lemma follows. Consider $n_L > q$. In the first row of the decomposition chart, i.e., the row for $\vec{X}_H = (0, 0, \ldots, 0)$, the number of different elements is at most 2^q since each element of the decomposition chart is a vector of q bits. Thus, there exist two different vectors $\vec{a}$ and $\vec{b} \in \{0, 1\}^{n_L}$ such that $\vec{F}(\vec{a}, \vec{0}) = \vec{F}(\vec{b}, \vec{0})$.

Next, consider the jth row $(j > 0)$. Let $\vec{c}$ be the value of $\vec{X}_H = (x_{n_L}, x_{n_L+1}, \ldots, x_{n-1})$. Then, by Definition 2.1, $\vec{F}$ satisfies

$$\vec{F}(\vec{a}, \vec{c}) = \vec{F}(\vec{a}, \vec{0}) + \vec{F}(\vec{0}, \vec{c})$$
$$\vec{F}(\vec{b}, \vec{c}) = \vec{F}(\vec{b}, \vec{0}) + \vec{F}(\vec{0}, \vec{c})$$

where the symbol $+$ denotes the addition of binary vectors that allow to carry propagations. Therefore, we have the relation $\vec{F}(\vec{a}, \vec{c}) = \vec{F}(\vec{b}, \vec{c})$. Since this relation holds for all $j > 0$, two column patterns that correspond to vectors $\vec{a}$ and $\vec{b}$ are the same.

From above, we can conclude that the column multiplicity of the decomposition chart is at most 2^q. ■

Theorem 2.1: Let $\vec{F}(\vec{X})$ be a WS function. Let $(\vec{X}_L, \vec{X}_H)$ be a partition of $\vec{X} = (x_0, x_1, \ldots, x_{n-1})$, where $\vec{X}_L = (x_0, x_1, \ldots, x_{n_L-1})$ and $\vec{X}_H = (x_{n_L}, x_{n_L+1}, \ldots, x_{n-1})$. Consider the decomposition chart of $\vec{F}$, where $\vec{X}_L$ denotes bound variables and $\vec{X}_H$ denotes free variables. Let $\vec{W} = (w_0, w_1, \ldots, w_{n-1})$ be the weight vector. Then, the column multiplicity of the decomposition chart is at most $\text{UB1} = 1 + \sum_{j=0}^{n_L-1} |w_j|$, where n_L denotes the number of variables in $\vec{X}_L$.

TABLE III
DECOMPOSITION CHART OF WS FUNCTION
(INTEGER REPRESENTATION)

$X_H = (x_3, x_4)$ \ $X_L = (x_0, x_1, x_2)$	000	001	010	011	100	101	110	111
00	0	3	2	5	1	4	3	6
01	5	8	7	10	6	9	8	11
10	4	7	6	9	5	8	7	10
11	9	12	11	14	10	13	12	15

Proof: Consider the decomposition chart for $\text{WS}(\vec{X}_L, \vec{X}_H)$. In the first row of the decomposition chart, $\vec{X}_H = (0, 0, \ldots, 0)$. Note that the column multiplicity is equal to the number of different values in the first row.

Consider the case where all the weights are positive. In this case, the number of different values is at most UB1, since WS takes values from 0 to $\sum_{j=0}^{n_L-1} w_j$.

Consider the case where some of the weights are negative. Assume that $w_0, w_1, \ldots, w_{t-1}$ are negative, and $w_t, w_{t+1}, \ldots, w_{n_L-1}$ are positive. Then, the WS takes values from $\sum_{j=0}^{t-1} w_j$ to $\sum_{j=t}^{n_L-1} w_j$. In this case, the number of different values is at most $1 + \sum_{j=0}^{t-1} |w_j| + \sum_{j=t}^{n_L-1} w_j = 1 + \sum_{j=0}^{n_L-1} |w_j|$. From these, we can conclude that the column multiplicity of the decomposition chart is at most UB1. ■

Example 2.3: Consider the case where $n = 5$ and $\vec{W} = (w_0, w_1, w_2, w_3, w_4) = (1, 2, 3, 4, 5)$. Let $\vec{X}_L = (x_0, x_1, x_2)$ and $\vec{X}_H = (x_3, x_4)$. In this case, $\text{UB1} = 1 + w_0 + w_1 + w_2 = 1 + 1 + 2 + 3 = 7$. Table III shows the decomposition chart of the function. Note that the column multiplicity of the decomposition chart is 7. So, the bound UB1 is tight.

A WS function usually has many outputs. When it is implemented as a monolithic circuit, it can be very large. However, if we partition the outputs into groups and implement each group separately, then the whole circuit can be smaller. The next two theorems give upper bounds on the column multiplicity for the block for the least significant i bits (LSBLOCK) and the block for the most significant $(q - i)$ bits (MSBLOCK). These bounds estimate the sizes of component circuits.

Theorem 2.2: Let $\vec{F}_{\text{LSB}}(\vec{X})$ be the logic function that represents the least significant i bits of a WS function. Then, the column multiplicity of the standard decomposition chart for $\vec{F}_{\text{LSB}}(\vec{X})$ is at most $\text{UB2} = 2^i$.

Proof: Let $F_{\text{LSB}}(\vec{X})$ be the integer represented by the least significant i bits of the function. Then, we have

$$F_{\text{LSB}}(\vec{X}) = \text{WS}(\vec{X}) \pmod{2^i}.$$

Since the column is computed in modulo 2^i, we can omit the most significant $(q - i)$ bits and leave only the least significant i bits. From Lemma 2.2, the number of different column patterns is at most 2^i. Hence, the column multiplicity of the standard decomposition chart is at most 2^i. ■

TABLE IV
DECOMPOSITION CHART OF WS FUNCTION (BINARY REPRESENTATION). (a) ALL FOUR BITS. (b) LEAST SIGNIFICANT TWO BITS. (c) MOST SIGNIFICANT TWO BITS

(a)

$\vec{X}_L = (x_0, x_1, x_2)$

$\vec{X}_H = (x_3, x_4)$	000	001	010	011	100	101	110	111
00	0000	0011	0010	0101	0001	0100	0011	0110
01	0101	1000	0111	1010	0110	1001	1000	1011
10	0100	0111	0110	1001	0101	1000	0111	1010
11	1001	1100	1011	1110	1010	1101	1100	1111

(b)

$\vec{X}_L = (x_0, x_1, x_2)$

$\vec{X}_H = (x_3, x_4)$	000	001	010	011	100	101	110	111
00	00	11	10	01	01	00	11	10
01	01	00	11	10	10	01	00	11
10	00	11	10	01	01	00	11	10
11	01	00	11	10	10	01	00	11

(c)

$\vec{X}_L = (x_0, x_1, x_2)$

$\vec{X}_H = (x_3, x_4)$	000	001	010	011	100	101	110	111
00	00	00	00	01	00	01	00	01
01	01	10	01	10	01	10	10	10
10	01	01	01	10	01	10	01	10
11	10	11	10	11	10	11	11	11

Definition 2.3: Let α be a real number. The largest integer that is not greater than α is denoted by $\lfloor \alpha \rfloor$, and the smallest integer that is equal to or greater than α is denoted by $\lceil \alpha \rceil$.

Theorem 2.3: Let $\vec{F}_{\mathrm{MSB}}(\vec{X})$ be the function that represents the ith to the most significant bits of a WS function. Then, the column multiplicity of the standard decomposition chart for $\vec{F}_{\mathrm{MSB}}(\vec{X})$ is at most

$$\mathrm{UB3} = \max_{n_L=1}^{n-1} \left[\min \left\{ 2^{n_L}, \left(\left\lfloor \frac{\sum_{j=0}^{n_L-1} |w_j|}{2^i} \right\rfloor + 1 \right) 2^{n-n_L} \right\} \right] \tag{2.3}$$

where $\vec{W} = (w_0, w_1, \ldots, w_{n-1})$ is the weight vector. Here, the least significant bit is the 0th bit.

Proof: Let $\vec{X}_L = (x_0, x_1, \ldots, x_{n_L-1})$ be the bound variables, and let $\vec{X}_H = (x_{n_L}, x_{n_L+1}, \ldots, x_{n-1})$ be the free variables of the standard decomposition chart. Let n_L be the number of bound variables and n_H be the number of free variables. It is clear that column multiplicity is at most 2^{n_L}, the total number of the columns. The maximal number represented from the ith bit to the most significant bit is $p = \lfloor \sum_{j=0}^{n_L-1} |w_j|/2^i \rfloor$. So, we can regard it as a $(p+1)$ valued function $g : B^n \to \{0, 1, \ldots, p\}$. Reorder the bound variables so that moving from left to right in the decomposition chart will not decrease the value of the function g. In this case, the number of changes of the columns in a row is at most $p+1$. Since there are 2^{n_H} rows, the column multiplicity is at most $2^{n_H}(p+1)$, where $n_H = n - n_L$. Hence, we have the theorem. ■

Example 2.4: Consider the case where $n = 5$ and $\vec{W} = (w_0, w_1, w_2, w_3, w_4) = (1, 2, 3, 4, 5)$. Let $\vec{X}_L = (x_0, x_1, x_2)$ and $\vec{X}_H = (x_3, x_4)$. Table IV(a) shows the decomposition chart, where the function values are represented by binary numbers. Table IV(b) is the decomposition chart of the least significant 2 bits. The column multiplicity is 4. Theorem 2.2 shows that the upper bound on the number of the column multiplicity is $\mathrm{UB2} = 2^2 = 4$. So, the bound UB2 is tight.

Table IV(c) is the decomposition chart of the most significant 2 bits. The column multiplicity is 3. Theorem 2.3 shows that the upper bound on the number of the column multiplicity is

$$\begin{aligned}
\mathrm{UB3} &= \max_{n_L=1}^{4} \left[\min \left(2^{n_L}, \left(\left\lfloor \frac{\sum_{j=0}^{n_L-1} w_j}{2^2} \right\rfloor + 1 \right) 2^{5-n_L} \right) \right] \\
&= \max \Big[\min \Big(2^1, \left(\left\lfloor \frac{w_0}{2^2} \right\rfloor + 1 \right) 2^4 \Big), \\
&\qquad \min \Big(2^2, \left(\left\lfloor \frac{w_0 + w_1}{2^2} \right\rfloor + 1 \right) 2^3 \Big), \\
&\qquad \min \Big(2^3, \left(\left\lfloor \frac{w_0 + w_1 + w_2}{2^2} \right\rfloor + 1 \right) 2^2 \Big), \\
&\qquad \min \Big(2^4, \left(\left\lfloor \frac{w_0 + w_1 + w_2 + w_3}{2^2} \right\rfloor + 1 \right) 2^1 \Big) \Big] \\
&= \max(2, 4, 8, 6) = 8.
\end{aligned}$$

Note that as shown in Table V, when $X_L = (x_0, x_1, x_2, x_3)$, $X_H = (x_4)$, the decomposition chart has a maximal column multiplicity of 5. In this case, the upper bound UB3 is not tight.

III. LUT CASCADE

An arbitrary logic function can be implemented by a single memory. However, as the number of input variables increases, the size of the memory increases exponentially.

In general, practical functions often have decomposition charts with small column multiplicities.

Theorem 3.1: For a given function f, let $\vec{X}_L$ be the variables for the columns, let $\vec{X}_H$ be the variables for the rows, and let μ be the column multiplicity of the decomposition chart. Then, the function f is realizable with the network shown in Fig. 1. In this case, the number of (two-valued) signal lines that connect two blocks H and G is $\lceil \log_2 \mu \rceil$ [2].

When the number of signal lines that connect two blocks is smaller than the number of variables in $\vec{X}_L$, we can often reduce the size of memory to implement the function. This technique is a **functional decomposition**.

By applying functional decomposition repeatedly to the given function, we have the **LUT cascade** shown in Fig. 2.

The cascade consists of **cells**, and the wires connecting adjacent cells are **rails**. Functions with small column multiplicities have compact LUT cascade realizations. To derive column multiplicities, we need not use decomposition charts. We can efficiently obtain column multiplicity by a binary decision diagram (BDD_for_CF) that represents the characteristic function for the multiple-output function [8], [11].

Theorem 3.2: Let μ be the maximum width of the BDD for the function f. Then, f can be implemented by the LUT cascade consisting of cells with at most $\lceil \log_2 \mu \rceil + 1$ inputs and at most $\lceil \log_2 \mu \rceil$ outputs [7].

Corollary 3.1: Let $\vec{F}_{\mathrm{LSB}}(\vec{X})$ be the logic function that represents the least significant q bits of a WS function. Then, $\vec{F}_{\mathrm{LSB}}(\vec{X})$ can be realized with the LUT cascade consisting of cells with at most $q+1$ inputs and at most q outputs.

Corollary 3.2: Let the number of outputs of a WS function be q. Then, the WS function can be realized with the LUT cascade consisting of cells with at most $q+1$ inputs and at most q outputs.

Theorem 3.3: Consider an LUT cascade for a function f. Let n be the number of primary inputs, s be the number of cells, r be the maximum number of rails (i.e., the number of lines between cells), k be the maximum number of inputs of a cell, μ be the maximum

TABLE V
DECOMPOSITION CHART OF MOST SIGNIFICANT TWO BITS OF WS FUNCTION

$\vec{X}_H = (x_4)$	$\vec{X}_L = (x_0, x_1, x_2, x_3)$															
	0000	0010	0100	0110	1000	1010	1100	1110	0001	0011	0101	1011	1001	1011	1101	1111
0	00	00	00	01	00	01	00	01	01	01	01	10	01	10	01	10
1	01	10	01	10	01	10	10	10	10	11	10	11	10	11	11	11

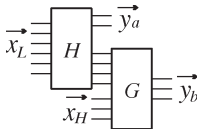

Fig. 1. Realization of logic functions by decomposition.

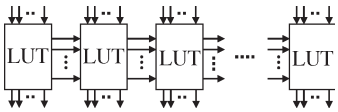

Fig. 2. LUT cascade with intermediate outputs.

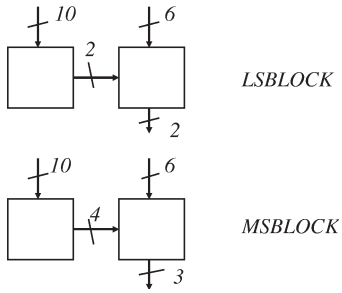

Fig. 3. Bit counting function WGT16.

width of the BDD for f, and $k \geq \lceil \log_2 \mu \rceil + 1$. Then, there is an LUT cascade for f that satisfies the relation

$$s \leq \left\lceil \frac{n-r}{k-r} \right\rceil. \tag{3.1}$$

Proof: From the design method of the LUT cascade, we have

$$k + (k-r)(s-1) \leq n.$$

Here, k in the left-hand side of the inequality denotes the number of inputs of the left-most LUT, and $(k-r)(s-1)$ denotes the sum of inputs for the remaining $(s-1)$ LUTs. When the actual number of rails is smaller than r, we append dummy rails to make the number of rails r. From this, we have

$$s - 1 \leq \frac{n-k}{k-r} \quad \text{and} \quad s \leq \frac{n-r}{k-r}.$$

Since s is an integer, we have (3.1). When this inequality holds, we can realize an LUT cascade for f having cells with at most k inputs. ■

IV. APPLICATIONS OF WS FUNCTIONS

In this part, we consider designs of bit counting circuits, ternary-to-binary converters, decimal-to-binary converters, and finite impulse response (FIR) filter. We also show the application to threshold functions.

A. Bit Counting Circuit

The bit counting function WGT n [6] is the simplest example of an n-input WS function. It counts the number of ones in the inputs and represents it by a binary number.

Example 4.1: Assume that $n = 16$. Then, we have $\vec{W} = (w_0, w_1, \ldots, w_{15}) = (1, 1, \ldots, 1)$. Let $\vec{F} = (f_4, f_3, f_2, f_1, f_0)$ be the outputs of the WS function, then we can show that [6]

$$f_4 = x_0 \cdot x_1 \cdots x_{15}$$

$$f_3 = \sum_{i1<i2<\cdots<i8} \oplus\, x_{i1} \cdot x_{i2} \cdot x_{i3} \cdots x_{i8}$$

$$f_2 = \sum_{i1<i2<i3<i4} \oplus\, x_{i1} \cdot x_{i2} \cdot x_{i3} \cdot x_{i4}$$

$$f_1 = \sum_{i1<i2} \oplus\, x_{i1} \cdot x_{i2}$$

$$f_0 = x_0 \oplus x_1 \oplus \cdots \oplus x_{15}$$

where $i_1, i_2, \ldots, i_8 \in \{0, 1, 2, \ldots, 15\}$. Since $n_L \leq 15$, by Theorem 2.1, we can see that the column multiplicity of the decomposition chart is at most UB1 $= 1 + \sum_{j=0}^{14} 1 = 1 + 15 = 16$. By Theorem 3.2, this function can be realized by a single cascade with five-input four-output cells. If the outputs are partitioned into (f_1, f_0) and (f_4, f_3, f_2), and realize them by the LSBLOCK and the MSBLOCK, respectively, then the column multiplicities for them are 4 and 14, respectively (see Table VII).

Fig. 3 shows the cascades for WGT16, where each cell has at most ten inputs. The upper cascade corresponds to LSBLOCK and realizes the least significant 2 bits (f_1, f_0). By Theorem 3.1, the number of outputs for the first cell is two since $\lceil \log_2 4 \rceil = 2$. The lower cascade corresponds to MSBLOCK and realizes the most significant 3 bits (f_4, f_3, f_2). By Theorem 3.1, the number of outputs for the first cell is four since $\lceil \log_2 14 \rceil = 4$. For this function, we can obtain the cascade structure from the number of inputs of cells.

B. Ternary-to-Binary Converter

Let $\vec{F} = (f_{q-1}, f_{q-2}, \ldots, f_0)$ be the output of a ternary-to-binary converter. Then, in general, f_i depends on all the inputs x_j $(j = 0, 1, \ldots, n-1)$. For ternary-to-binary converters, we use the binary-coded ternary code to represent a ternary digit. That is, zero is represented by (00); one is represented by (01); and two is represented by (10). (11) is an unused code. In the decomposition chart, the input variables are grouped into pairs. The truth table of the two-digit ternary to a 4-bit binary converter is shown in Table VI. In this case, (11) is an undefined input, and the corresponding outputs are don't cares. In Table VI, the binary-coded ternary representation is denoted by $\vec{X} = (x_0, x_1, x_2, x_3)$, the ternary representation is

TABLE VI
TRUTH TABLE FOR A TERNARY-TO-BINARY CONVERTER

Binary – Coded Ternary				Ternary		Binary				Decimal
x_3	x_2	x_1	x_0	t_1	t_0	f_3	f_2	f_1	f_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

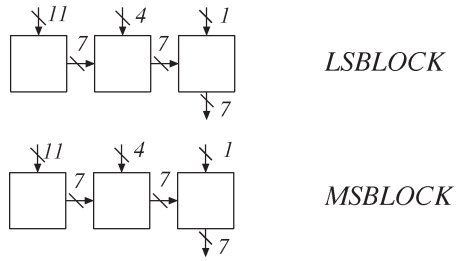

Fig. 4. Eight-digit ternary-to-binary converter.

denoted by $\vec{T} = (t_1, t_0)$, and the binary representation is denoted by $\vec{F} = (f_3, f_2, f_1, f_0)$. When we implement this converter by a WS function, the weight vector is $\vec{W} = (w_0, w_1, w_2, w_3) = (1, 2, 3, 6)$. In this case, the function is completely specified. For example, for the input $(x_0, x_1, x_2, x_3) = (1, 1, 1, 1)$, the output is (1, 1, 0, 0) since WS $= 1 + 2 + 3 + 6 = 12$.

Example 4.2: Consider an eight-digit ternary-to-binary converter. Since a ternary digit requires 2 bits, the total number of inputs is $2 \times 8 = 16$. Further, the number of output bits is 13. To implement the converter by a WS function, the weight vector should be $\vec{W} =$ (1, 2, 3, 6, 9, 18, 27, 54, 81, 162, 243, 486, 729,1458, 2187, 4374). The column multiplicity of this function is bounded above by $1 + \sum_{i=0}^{14} w_i = 5468$. This suggests that the function is unsuitable for a single cascade realization. So we will implement it by a pair of cascades. Assuming that we use cells with 11 inputs, we have the cascade realization shown in Fig. 4. The upper cascade LSBLOCK realizes the least significant seven bits, while the lower cascade MSBLOCK realizes the most significant seven bits. From Theorem 2.2, the column multiplicity of the decomposition chart for the LSBLOCK is at most $2^7 = 128$. Thus, the number of rails for the LSBLOCK is $\lceil \log_2 128 \rceil = 7$. From Theorem 2.3, the column multiplicity of the decomposition chart for the MSBLOCK is at most 128. Thus, the number of rails is $\lceil \log_2 128 \rceil = 7$.

From Fig. 4, we can see that the amount of memory needed for the cascades is $7(2^{11} + 2^{11} + 2^8 + 2^{11} + 2^{11} + 2^8) = 32\,256$ (bits), which is much smaller than the single memory realization. Note that the most significant bit, i.e., 14th bit, is not used for valid inputs and can be omitted. The single memory realization requires $2^{16} \times 13 = 851\,968$ (bits).

C. Decimal-to-Binary Converter

In this part, we consider the design of various decimal-to-binary converters.

Example 4.3: Consider a five-digit decimal to binary converter. When the decimal numbers are represented by the 8421 BCD code, the number of binary inputs is $4 \times 5 = 20$.

Suppose that we realize it by the WS function with the weight vector $\vec{W} = (1, 2, 4, 8, 10, 20, 40, 80, 100, 200, 400, 800, 1000, 2000, 4000,$ $8000, 10\,000, 20\,000, 40\,000, 80\,000)$. We use two LUT cascades to implement the function: the LSBLOCK realizes the least significant nine bits and the MSBLOCK realizes the most significant nine bits. From Theorem 2.2, we can see that the column multiplicity for the LSBLOCK is at most $2^9 = 512$. From Theorem 2.3, we can see that the column multiplicity for the MSBLOCK is at most 1024. So, we can implement these blocks by using a cascade with cells of at most 11 inputs. With 12-input cells, we can implement the WS function consisting of a pair of cascades as shown in Fig. 5. As shown in Table VII, the actual column multiplicity for the MSBLOCK is 521. So, the bound UB3 is not tight. However, it is still useful since $\lceil \log_2 1024 \rceil = \lceil \log_2 521 \rceil = 10$.

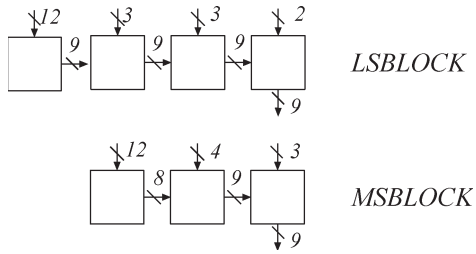

Fig. 5. Five-digit decimal-to-binary converter (natural ordering).

TABLE VII
UPPER BOUNDS AND ACTUAL NUMBERS OF COLUMN MULTIPLICITIES

Name	Inputs	Outputs	Bits	Bound	Actual	
WGT16	16	5	2	4	4	LSBLOCK
			3	16	14	MSBLOCK
8_ter2bin	16	14	7	128	128	LSBLOCK
			7	128	126	MSBLOCK
8421_5digit	20	18	9	512	512	LSBLOCK
			9	1024	521	MSBLOCK
84-2-1_5digit	20	17	9	512	512	LSBLOCK
			8	1024	522	MSBLOCK
2421_5digit	20	17	9	512	512	LSBLOCK
			8	1024	313	MSBLOCK
5211_5digit	20	17	9	512	512	LSBLOCK
			8	1024	313	MSBLOCK
FIR filter	17	15	8	256	256	LSBLOCK
			7	1792	938	MSBLOCK

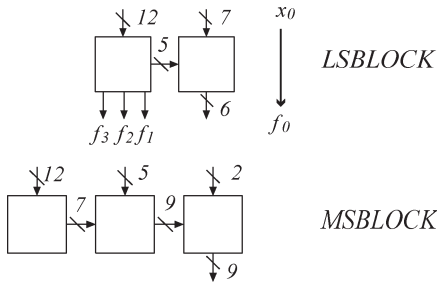

Fig. 6. Five-digit decimal-to-binary converter (optimal ordering).

In the case of the decimal-to-binary converter, some outputs depend on only a part of the inputs. Especially, $f_0 = x_0$. That is, the least significant bit depends on only x_0. Also, the MSBLOCK does not depend on x_0. When we change the ordering of the inputs and outputs, we have smaller cascades shown in Fig. 6. Note that in the LSBLOCK, three outputs $\{f_3, f_2, f_1\}$ depend on only 12 inputs.

Example 4.4: Table VIII shows the 5211, 2421, and 84-2-1 codes, where the 9's complements are easily obtained. Similar to the 8421 code, we can design converters for 5211, 2421, and 84-2-1 codes.

TABLE VIII
VARIOUS CODES FOR DECIMAL-TO-BINARY CONVERTERS

Decimal Number	8421 Code	5211 Code	2421 Code	84-2-1 Code
0	0000	0000	0000	0000
1	0001	0001	0001	0111
2	0010	0011	0010	0110
3	0011	0101	0011	0101
4	0100	0111	0100	0100
5	0101	1000	1011	1011
6	0110	1010	1100	1010
7	0111	1100	1101	1001
8	1000	1110	1110	1000
9	1001	1111	1111	1111

To design these cascades, we used weights as follows: $\vec{W} = (1, 1, 2, 5, 10, 10, 20, 50, 100, 100, 200, 500, 1000, 1000, 2000, 5000, 10\,000, 10\,000, 20\,000, 50\,000)$. $\vec{W} = (1, 2, 2, 4, 10, 20, 20, 40, 100, 200, 200, 400, 1000, 2000, 2000, 4000, 10\,000, 20\,000, 20\,000, 40\,000)$. $\vec{W} = (-1, -2, 4, 8, -10, -20, 40, 80, -100, -200, 400, 800, -1000, -2000, 4000, 8000, -10\,000, -20\,000, 40\,000, 80\,000)$.

Again, we use two modules to implement code converters, namely, 1) the LSBLOCK realizes the least significant nine bits and 2) the MSBLOCK realizes the most significant nine bits. Table VII shows the upper bounds obtained from Theorems 2.2 and 2.3 and the actual numbers for the column multiplicities. Note that the ordering of the variables are fixed to $(x_0, x_1, \ldots, x_{n-1})$. For the LSBLOCKs, if we reorder the variables, the column multiplicities are greatly reduced.

D. FIR Filter

Digital filters are important elements in signal processing [4] and can be classified into two types, namely, FIR filters and infinite impulse response (IIR) filters. FIR filters implement nonrecursive structures and so always have stable operations. Also, FIR filters can have linear phase characteristics, so they are useful for waveform transmission.

To realize FIR filters, we can use DA to convert the multiply accumulation operations into table-lookup operations [3], [14]. In this part, we consider an implementation of the DA of the FIR filter by an LUT cascade. The LUT cascade realization requires much smaller memory than the single memory realization. The structure of the FIR filter mainly depends on the number of taps N, the number of bits in the outputs q, and the number of inputs k of the cells in the LUT cascade.

Definition 4.1: The **FIR filter** computes

$$\mathcal{Y}(n) = \sum_{i=0}^{N-1} w_i \mathcal{X}(n-i) \tag{4.1}$$

where $\mathcal{X}(i)$ is the value of the input $\mathcal{X}$ at the time i and $\mathcal{Y}(i)$ is the value of the output $\mathcal{Y}$ at the time i.[1] w_i is a **filter coefficient** represented by a q-bit binary number, and N is the **number of taps** in the filter.[2]

Fig. 7 implements (4.1) directly. It consists of an N-stage q-bit shift register, N copies of q-bit multipliers, and an adder for N q-bit numbers. To reduce the amount of hardware in Fig. 7, we use the bit serial method shown in Fig. 8, where PSC denotes the parallel to series converter and ACC denotes the shifting accumulator, which accumulates the numbers while doing shifting operations.

[1] $\mathcal{X}$ and $\mathcal{Y}$ denote the values of signal in the filters. x_i denotes a logic variable and $\vec{X}_1$ and $\vec{X}_2$ denote the vectors of logic variables.

[2] In general, the number of bits for h_i and $\mathcal{Y}$ can be different. However, for simplicity, we assume that they are represented by q bits.

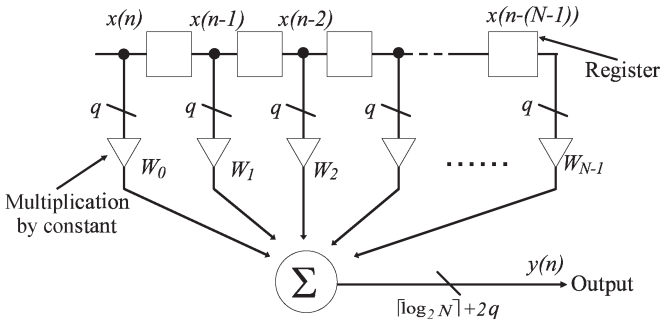

Fig. 7. Parallel realization of FIR filter.

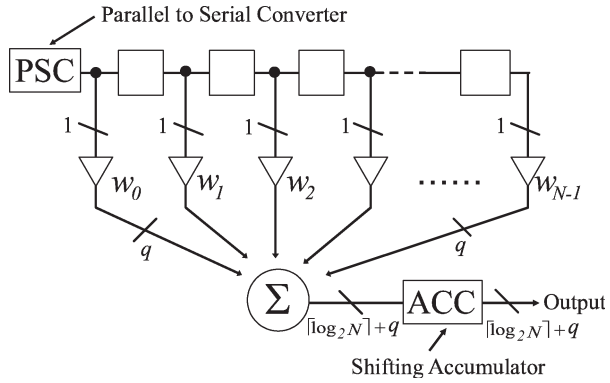

Fig. 8. Serial realization of FIR filter.

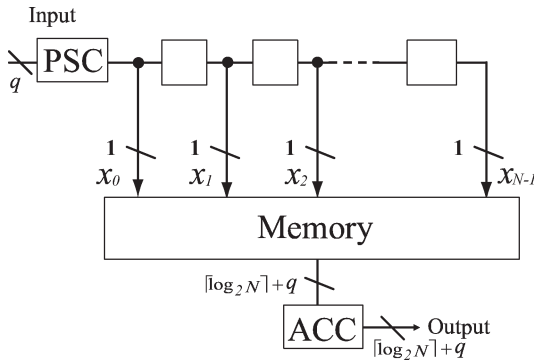

Fig. 9. Single memory realization of FIR filter.

In this case, the inputs to $w_0, w_1, \ldots, w_{N-1}$ are either 0 or 1, and the multipliers are replaced by AND gates. The combinational part in Fig. 8 has N-inputs and $(\lceil \log_2 N \rceil + q)$-outputs. In Fig. 9, the combinational part is implemented by the memory that realizes the WS function

$$\mathrm{WS}(x_0, x_1, \ldots, x_{N-1}) = \sum_{j=0}^{N-1} w_j x_j.$$

This method of computation is known as DA and is often used to implement convolution operations, since many multipliers and an adder with many inputs can be replaced by one memory [3], [14]. It is applicable only when the coefficients w_i are constants. In FIR filters, the coefficients w_i are constants, so we can apply this method. It reduces the amount of hardware by $1/q$ but increases the computation time by a factor of q.

Example 4.5: Consider a low-pass FIR filter with 33 taps. Suppose that it is symmetric, so we need only to realize the WS function with 17 inputs [4]. Let the number of output bits be 15 and let

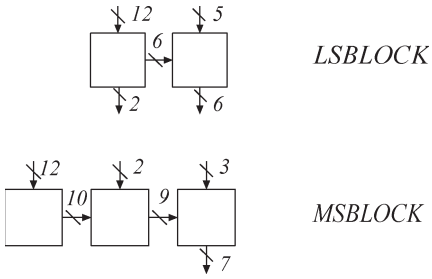

Fig. 10. FIR filter (optimal ordering).

the filter coefficients be $\vec{W} = (378, 188, -521, -1120, -713, 353, 614, -420, -1168, -100, 1538, 920, -1925, -2720, 2167, 10\,164, 14\,125)$. A single memory realization requires $2^{17} \cdot 15 = 1\,966\,080$ bits. Fig. 10 shows the LUT cascades for the filter, where the LSBLOCK realizes the least significant eight bits and the MSBLOCK realizes the most significant seven bits. The bounds obtained from Theorems 2.2 and 2.3 are shown in Table VII. In this case, the ordering of the input and output variables is optimized. Especially, the LSBLOCK is reduced drastically since two outputs depend on only 12 variables. The total amount of memory is $2^{12} \times 8 + 2^{11} \times 6 + 2^{12} \times 10 + 2^{12} \times 9 + 2^{12} \times 7 = 110\,592$ bits.

E. Threshold Function

Definition 4.2: A **threshold function** $f(x_0, x_1, \ldots, x_{n-1})$ satisfies the relation $f = 1$ if $\sum_{i=1}^{n} w_i x_i \geq T$, and $f = 0$ otherwise, where $(w_0, w_1, \ldots, w_{n-1})$ are **weights** and T is the **threshold**.

Although a threshold function is not a WS function, we can estimate the column multiplicity of a threshold function from the theory of WS functions.

Theorem 4.1: The column multiplicity of a decomposition chart of the threshold function with weights $(w_0, w_1, \ldots, w_{n-1})$ is at most

$$\text{UB4} = 1 + \sum_{i=0}^{n-1} |w_i|. \tag{4.2}$$

Proof: The column multiplicity of a decomposition chart for f is not greater than that of the WS function having the same weights. By Theorem 2.1, the column multiplicity of the WS function is at most UB4. Hence, we have the theorem. ■

Threshold functions are useful for neural nets. So, we can see that the LUT cascade is promising for neural nets when the sums of weights are small.

V. Arithmetic Decomposition of WS Functions

In general, a q-output WS function requires $(q+1)$-input q-output cells in a cascade realization. Thus, when q is large, large cells are required. To implement a WS function with many outputs using smaller cells, we can decompose the WS function into smaller ones. Note that this is different from the **partition of outputs**, where each group of outputs is implemented by a cascade independently.

A $2q$-output WS function can be decomposed into a pair of WS functions as follows. Let w_i be a weight of $2q$ output WS function. Then, w_i can be written as

$$w_i = 2^q w_{Ai} + w_{Bi}$$

where w_{Ai} denotes the most significant q bits and w_{Bi} denotes the least significant q bits. In this case, we can implement the $2q$ output WS function by using a pair of WS functions and an adder, as shown in Fig. 11. Note that the adder has $2q$ inputs and q outputs.

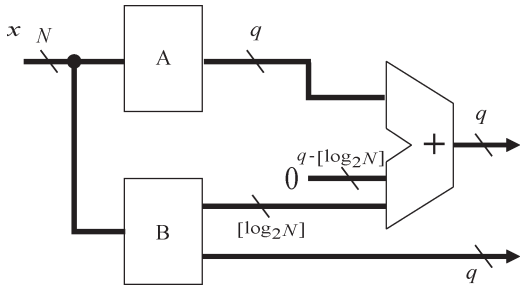

Fig. 11. Arithmetic decomposition of $2q$ output WS function.

Theorem 5.1: A $2q$-output WS function $F(\vec{X})$ that represents

$$\sum_{i=0}^{N-1} w_i x_i$$

can be decomposed into a pair of WS functions $\vec{F}_A(\vec{X})$ and $\vec{F}_B(\vec{X})$, where $\vec{F}_A(\vec{X})$ is a q-output WS function representing

$$\sum_{i=0}^{N-1} w_{Ai} x_i$$

and $\vec{F}_B(\vec{X})$ is a $q + \lceil \log_2 N \rceil$-output WS function representing

$$\sum_{i=0}^{N-1} w_{Bi} x_i$$

and

$$w_i = 2^q w_{Ai} + w_{Bi}.$$

This is an **arithmetic decomposition of a WS function**.

In a similar way, a $4q$-output WS function can be decomposed into four WS functions as follows. Let w_i be a weight of the $4q$ output WS function. Then, w_i can be written as

$$w_i = 2^{3q} w_{Ai} + 2^{2q} w_{Bi} + 2^q w_{Ci} + w_{Di}$$

where w_{Ai}, w_{Bi}, w_{Ci}, and w_{Di} denote q-bit numbers. As shown in Fig. 12, we realize the $4q$-output WS function by using four q-output WS functions and adders. Note that block A realizes a q-output WS function, while blocks B, C, and D realize $(q + \lceil \log_2 N \rceil)$-output WS functions.

Note that the output adder has $4q$ inputs and $2q$ outputs.

By applying the arithmetic decomposition iteratively, we can implement any WS function with small cascades. We applied this method to FIR filters and implemented on field-programmable gate arrays (FPGAs). Note that recent FPGAs have embedded RAMs [1], [15] and we can use these RAMs as cells of the LUT cascades [13].

VI. Conclusion and Comment

In this paper, we first defined WS functions as a mathematical model of bit counting circuits, radix converters, and DA. Then, we

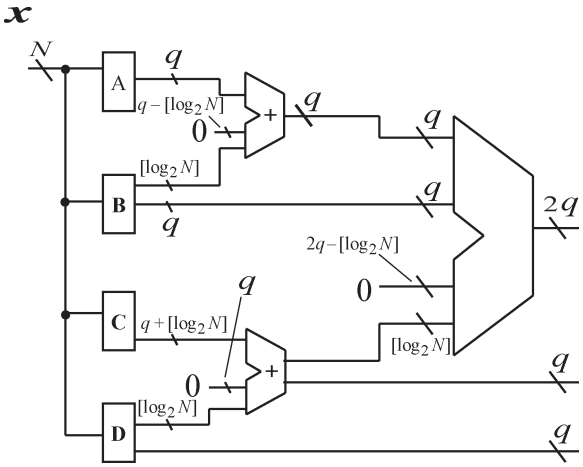

Fig. 12. Arithmetic decomposition of $4q$-output WS function.

derived upper bounds on the column multiplicity for the standard decomposition chart for a WS function.

WS functions with small weights have decomposition charts with small column multiplicity. Thus, they can be efficiently implemented by an LUT cascade. Since column multiplicity is equal to the width of the BDD [5], a WS function with small weights has a small BDD. The results of this paper imply that a neural net with small weights can be efficiently implemented by LUT cascades.

ACKNOWLEDGMENT

The author would like to thank Prof. Iguchi for very helpful discussions and Prof. J. T. Butler for discussions that improved the English presentation.

REFERENCES

[1] Altera Cyclone FPGA. [Online]. Available: http://www.altera.com/
[2] H. A. Curtis, *A New Approach to the Design of Switching Circuits*. Princeton, NJ: Van Nostrand, 1962.
[3] L. Mintzer, "FIR filters with field-programmable gate arrays," *J. VLSI Signal Process.*, vol. 6, no. 2, pp. 120–127, Aug. 1993.
[4] K. K. Parhi, *VLSI Digital Signal Processing Systems Design and Implementation*. New York: Wiley, 1999.
[5] T. Sasao, "FPGA design by generalized functional decomposition," in *Logic Synthesis and Optimization*, T. Sasao, Ed. Norwell, MA: Kluwer, 1993, pp. 233–258.
[6] ——, *Switching Theory for Logic Synthesis*. Norwell, MA: Kluwer, 1999.
[7] T. Sasao, M. Matsuura and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," in *Proc. IWLS*, Lake Tahoe, CA, Jun. 12–15, 2001, pp. 225–230.
[8] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," in *Proc. Design Automation Conf.*, San Diego, CA, Jun. 2–6, 2004, pp. 428–433.
[9] T. Sasao, J. T. Butler, and M. Riedel, "Application of LUT cascades to numerical function generators," in *Proc. 12th SASIMI Workshop*, Kanazawa, Japan, Oct. 18–19, 2004, pp. 422–429.
[10] T. Sasao, "Radix converters: Complexity and implementation by LUT cascades," in *Proc. Int. Symp. Multiple-Valued Logic*, Calgary, AB, Canada, May 18–21, 2005, pp. 256–263.
[11] T. Sasao and M. Matsuura, "BDD representation for incompletely specified multiple-output logic functions and its applications to functional decomposition," in *Proc. Design Automation Conf.*, San Diego, CA, Jun. 2005, pp. 373–378.
[12] T. Sasao, "Analysis and synthesis of weighted-sum functions," in *Proc. Int. Workshop Logic Synthesis*, Lake Arrowhead, CA, Jun. 8–10, 2005, pp. 455–462.
[13] T. Sasao, Y. Iguchi, and T. Suzuki, "On LUT cascade realizations of FIR filters," in *Proc. 8th Euromicro Conf. DSD*, Porto, Portugal, Aug. 30–Sep. 3 2005, pp. 467–474.
[14] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Mag.*, vol. 6, no. 3, pp. 4–19, Jul. 1989.
[15] XILINX Spartan FPGA. [Online]. Available: http://www.xilinx.com/