

# Analysis and Synthesis of Weighted-Sum Functions

Tsutomu Sasao

Department of Computer Science and Electronics,  
Kyushu Institute of Technology,  
Iizuka 820-8502, Japan

April 28, 2005

## Abstract

A weighted-sum (WS) function computes the sum of selected integers. This paper considers a design method for WS functions by LUT cascades. In particular, it derives upper bounds on the column multiplicities of decomposition charts for WS functions. From these, we can estimate the size of LUT cascades that realize WS functions. These bounds are useful to realize WS functions, since they show strategies to partition the outputs into groups.

## 1 Introduction

A weighted-sum function (WS function) computes sum of selected integers:  $WS(x_0, x_1, \dots, x_{n-1}) = \sum_{i=0}^{n-1} w_i x_i$ , 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 and distributed arithmetic for convolution operation, etc.

The LUT cascade has a regular structure and is easy to design and modify. It efficiently implements the function whose column multiplicity of the decomposition chart is small [6, 2].

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 we can efficiently realize WS functions with LUT cascades.

## 2 WS Functions

A WS function is a mathematical model of bit counting circuits, code converters, and distributed arithmetic, etc.

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

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

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

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

**Definition 2.2** [8] Consider a function  $\vec{F}(\vec{X}) : B^n \rightarrow B^q$ , where  $B = \{0, 1\}$ . Let  $(\vec{X}_L, \vec{X}_H)$  be a partition of  $\vec{X}$ , where  $\vec{X}_L = (x_0, x_1, \dots, x_{n_L-1})$  and  $\vec{X}_H = (x_{n_L}, x_{n_L+1}, \dots, x_{n-1})$ . 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  $B$  to  $\vec{X}_H$ , 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 the left to the right, and from the top to the 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**.

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

**Example 2.1** Table 2.1 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 two 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 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 the both elements. 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. (End of Example)

**Lemma 2.1** Let  $\vec{F}(x_0, x_1, \dots, x_{n-1})$  be an  $n$ -input  $q$ -output WS function. Let  $(\vec{X}_L, \vec{X}_H)$  be a partition of  $\vec{X} = (x_0, x_1, \dots, x_{n-1})$ , where  $\vec{X}_L = (x_0, x_1, \dots, x_{n_L-1})$  and  $\vec{X}_H = (x_{n_L}, x_{n_L+1}, \dots, x_{n-1})$ . Consider the decomposition chart of  $\vec{F}$ , where  $\vec{X}_L$  denotes the bound variables and  $\vec{X}_H$  denotes the free variables. In this case, the column multiplicity of the decomposition chart is at most  $2^q$ .

**(Proof)** 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  $\vec{X}_H = (0, 0, \dots, 0)$ , 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  $\vec{a}, \vec{b} \in \{0, 1\}^{n_L}$ , such that  $\vec{F}(\vec{a}, \vec{0}) = \vec{F}(\vec{b}, \vec{0})$ .

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

$$\begin{aligned}\vec{F}(\vec{a}, \vec{c}) &= \vec{F}(\vec{a}, \vec{0}) + \vec{F}(\vec{0}, \vec{c}) \\ \vec{F}(\vec{b}, \vec{c}) &= \vec{F}(\vec{b}, \vec{0}) + \vec{F}(\vec{0}, \vec{c}),\end{aligned}$$

where the symbol  $+$  denotes the vector addition of binary numbers that allows the 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  $2^q$ . (Q.E.D.)

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

**(Proof)** Consider the decomposition chart for  $WS(\vec{X}_L, \vec{X}_H)$ . In the the first row of the decomposition chart,  $\vec{X}_H = (0, 0, \dots, 0)$ . 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  $w_0, w_1, \dots, w_{t-1}$  are negative, and  $w_t, w_{t+1}, \dots, w_{n_L-1}$  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  $UB1$ . (Q.E.D.)

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 may 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  $\vec{F}_{LSB}(\vec{X})$  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  $\vec{F}_{LSB}(\vec{X})$  is at most  $UB2 = 2^i$ .

**(Proof)** The least significant  $i$  bits represent the function

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

Since the column is computed in  $\text{mod } 2^i$ , we can omit the most significant  $(q - i)$  bits, and leave only the least significant  $i$  bits. From Lemma 2.1, the number of different column patterns is at most  $2^i$ . Hence, the column multiplicity of the standard decomposition chart is at most  $2^i$ . (Q.E.D.)

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

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

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

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

Table 2.1: Decomposition Chart for a WS function.

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

**(Proof)** Let  $\vec{X}_L = (x_0, x_1, \dots, x_{n_L-1})$  be the bound variables, and let  $\vec{X}_H = (x_{n_L}, x_{n_L+1}, \dots, x_{n-1})$  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 the column multiplicity is at most  $2^{n_L}$ , the total number of the columns. The maximal number represented from the  $i$ -th bit to the most significant bit is  $p = \lfloor \frac{\sum_{j=0}^{n_L-1} |w_j|}{2^i} \rfloor$ . So, we can regard it as a  $(p+1)$  valued function  $g: B^n \rightarrow \{0, 1, \dots, p\}$ . Reorder the bound variables so that moving from the 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  $2^{n_H} \cdot (p+1)$ , where  $n_H = n - n_L$ . Hence, we have the theorem. (Q.E.D.)

### 3 LUT Cascade

An arbitrary logic function can be implemented by a single memory. However, with the increase of the number of input variables, 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, and let  $\vec{X}_H$  be the variables for the rows, and let  $\mu$  be the column multiplicity of the decomposition chart. Then, the function  $f$  is realizable with the network shown in Fig. 3.1. In this case, the number of (two-valued) signal lines that connect two blocks  $H$  and  $G$  is  $\lceil \log_2 \mu \rceil$ .

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 **functional decomposition**.

By applying functional decomposition repeatedly to the given function, we have the **LUT cascade** shown in Fig. 3.2. The cascade consists **cells**, and the wires connecting adjacent cells are **rails**. Functions with small column multi-

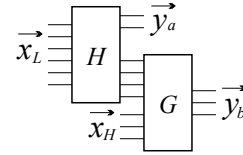


Figure 3.1: Realization of logic functions by decomposition.

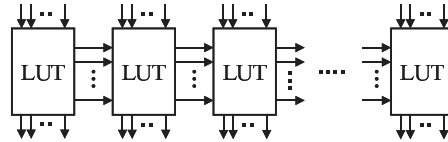


Figure 3.2: LUT cascade with intermediate outputs.

plicities have compact LUT cascade realizations. To derive column multiplicities, we need not use decomposition charts. We can efficiently obtain the column multiplicity by a binary decision diagram (BDD\_for\_CF) that represents the characteristic function for the multiple-output function [7, 10].

**Theorem 3.2** [6] Let  $\mu$  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.

**Corollary 3.1** Let  $\vec{F}_{LSB}(\vec{X})$  be the logic function that represents the least significant  $i$  bits of a WS function. Then,  $\vec{F}_{LSB}(\vec{X})$  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,  $\mu$  be the maximum 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:

$$s \leq \lceil \frac{n-r}{k-r} \rceil \quad (3.1)$$

**(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. (Q.E.D.)

## 4 Applications of WS Functions

In this part, we consider design of bit counting circuits, ternary-to-binary converters, decimal-to-binary converters, and FIR filter. We also show the application to threshold functions.

### 4.1 Bit Counting Circuit

The **bit counting function**  $\text{WGT}_n$  [5]. is the simplest example of an  $n$ -input WS function. It counts the number of 1's in the inputs, and represent it by a binary number.

**Example 4.1** Assume that  $n = 16$ . Then, we have  $\vec{W} = (w_0, w_1, \dots, w_{15}) = (1, 1, \dots, 1)$ . 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 [5]:

$$\begin{aligned} f_4 &= x_0 \cdot x_1 \cdots x_{15} \\ f_3 &= \bigoplus_{i_1 < i_2 < \dots < i_8} x_{i_1} \cdot x_{i_2} \cdot x_{i_3} \cdots x_{i_8} \\ f_2 &= \bigoplus_{i_1 < i_2 < i_3 < i_4} x_{i_1} \cdot x_{i_2} \cdot x_{i_3} \cdot x_{i_4} \\ f_1 &= \bigoplus_{i_1 < i_2} x_{i_1} \cdot x_{i_2} \\ f_0 &= x_0 \oplus x_1 \oplus \dots \oplus x_{15}, \end{aligned}$$

where  $i_1, i_2, \dots, i_8 \in \{0, 1, 2, \dots, 15\}$ . By Theorem 2.1, we can see that the column multiplicity of the decomposition chart is at most 16. By Theorem 3.1, this function can

Table 4.1: 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

be realized by a cascade with 5-input 4-output cells. If the outputs is 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 4.2.) (End of Example)

### 4.2 Ternary-to-Binary Converter

Let  $\vec{F} = (f_{q-1}, f_{q-2}, \dots, f_0)$  be the output of a ternary-to-binary converter. Then, in general,  $f_i$  depends on all the inputs  $x_j$  ( $j = 0, 1, \dots, n-1$ ). For ternary-to-binary converters, we use the binary-coded-ternary code to represent a ternary digit. That is 0 is represented by (00); 1 is represented by (01); and 2 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 2-digit ternary to 4-bit binary converter is shown in Table 4.1. In this case, (11) is an undefined input, and the corresponding outputs are *don't cares*. In Table 4.1, the binary-coded-ternary representation is denoted by  $\vec{X} = (x_0, x_1, x_2, x_3)$ , the ternary representation is 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 8-digit ternary-to-binary converter. Since a ternary digit requires two 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  $1 + \sum_{i=0}^{14} w_i = 5468$ , and it is almost impossible to implement the function by a single cascade. So, we will implement it by a pair of cascades. Assume that we use cells with 11 inputs, we have the cascade realization shown in Fig. 4.1. The upper cascade **LSBLOCK** realizes the least

Table 4.2: Upper Bounds and actual numbers of the 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

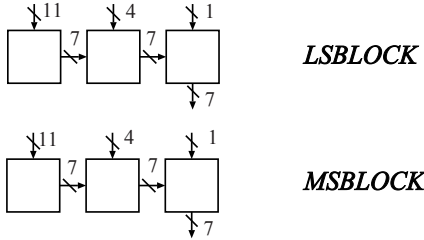


Figure 4.1: 8-digit ternary-to-binary converter.

significant 7 bits, while the lower cascade *MSBLOCK* realizes the most significant 7 bits. From Theorem 2.2, the column multiplicity of the decomposition chart for the *LSBLOCK* is  $2^7 = 128$ . 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 128. Thus, the number of rails is  $\lceil \log_2 128 \rceil = 7$ .

From Fig. 4.1, we can see that the necessary amount of memory of 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 requires  $2^{16} \times 13 = 851,968$  (bits). (End of Example)

### 4.3 Decimal-to-Binary Converter

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

**Example 4.3** Consider a 5-digit decimal to binary converter. When the decimal numbers are represented by the

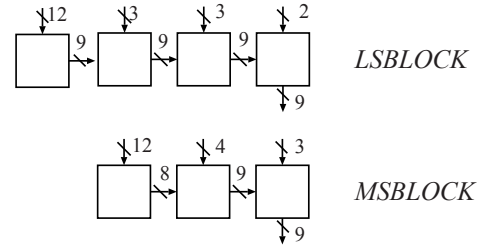


Figure 4.2: 5-digit decimal-to-binary converter (ordering original).

*8421 BCD code*, the number of binary inputs is  $4 \times 5 = 20$ . Note that the number of different combinations represented by the valid inputs is  $10^5$ . So, the number of don't care combinations is  $2^{20} - 10^5$ . Thus, the ratio of the don't care is  $(2^{20} - 10^5)/2^{20} \approx 0.90$ . In other words, about 90% of the input combinations are don't cares. This means that the assignment of don't care values greatly influences the complexity of converter.

Suppose that we realize it by the WS function with the weight vector  $\vec{W} = (1, 2, 4, 8, 10, 20, 40, 80, 100, 200, 400, 800, 1000, 2000, 4000, 8000, 10000, 20000, 40000, 80000)$ . We use two LUT cascades to implement the function: the *LSBLOCK* realizes the least significant 9 bits, and the *MSBLOCK* realizes the most significant 9 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 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. 4.2.

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. 4.3. Note that in the *LSBLOCK*, three outputs  $\{y_3, y_2, y_1\}$  depends on only 12 inputs. (End of Example)

**Example 4.4** Table 4.3 shows the 5211, 2421, and 84-2-1 codes, where the 9's complement are easily obtained. Similarly to the 8421 code, we can design converters for 5211, 2421, and 84-2-1 codes. To design these cascades, we used weights as follows:  $\vec{W} = (1, 1, 2, 5, 10, 10, 20, 50, 100, 100, 200, 500, 1000, 1000, 2000, 5000, 10000, 10000, 20000, 50000)$ .  $\vec{W} = (1, 2, 2, 4, 10, 20, 20, 40, 100, 200, 200, 400, 1000, 2000, 2000, 4000, 10000, 20000, 20000, 40000)$ .  $\vec{W} = (-1, -2, 4, 8, -10, -20, 40, 80, -100, -200, 400, 800, -1000,$

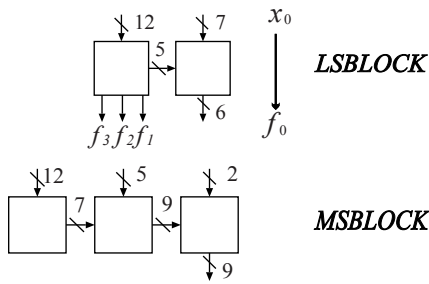


Figure 4.3: 5-digit decimal-to-binary converter (ordering optimized).

Table 4.3: 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

–2000, 4000, 8000, –10000, –20000, 40000, 80000). Again, we use two modules to implement code converters: the *LSBLOCK* realizes the least significant 9 bits, and the *MSBLOCK* realizes the most significant 9 bits. Table 4.2 shows the upper bounds obtained from Theorem 2.2 and Theorem 2.3, and actual numbers of the column multiplicities. Note that the ordering of the variables are fixed to  $(x_0, x_1, \dots, x_{n-1})$ . For the *LSBLOCKs*, if we reorder the variables, the column multiplicities were greatly reduced. (End of Example)

#### 4.4 FIR Filter

Digital filters are important elements in signal processing [3], and can be classified into two types: FIR (Finite Impulse Response) filters and IIR (Infinite Impulse Response) filters. FIR filters implement nonrecursive structure, and so always have stable operations. Also, FIR filters can have linear phase characteristics, so they are useful for wave form transmission.

To realize FIR filters, we can use Distributed Arithmetic (DA) to convert the multiply-accumulation operations into table-lookup operations [1, 12]. In this part, we consider an

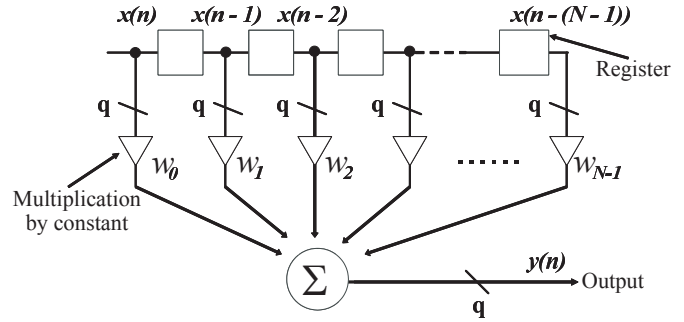


Figure 4.4: Parallel Realization of FIR Filter.

implementation of the distributed arithmetic of the FIR filter by an LUT cascade. The LUT cascade realization require much smaller memory than the single memory realization. The structure of 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

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

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  $i$ <sup>1</sup>.  $w_i$  is a **filter coefficient** represented by a  $q$ -bit binary number, and  $N$  is the **the number of taps in the filter**<sup>2</sup>.

Fig. 4.4 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. 4.4, we use bit-serial method shown in Fig. 4.5, where PSC denotes the parallel to series converter, and ACC denotes the shifting accumulator, which accumulates the numbers while doing shifting operations.

In this case, the inputs to  $w_0, w_1, \dots, w_{N-1}$  are either 0 or 1, and the multipliers are replaced by AND gates. The combinational part in Fig. 4.5 has  $N$ -inputs and  $q$ -outputs. In Fig. 4.6, the combinational part is implemented by the ROM that realizes the WS function:

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

This method of computation is the **Distributed Arithmetic**, and is often used to implement convolution operations, since many multipliers and an adder with many inputs can

<sup>1</sup>  $\mathcal{X}$  and  $\mathcal{Y}$  denotes the values of signal in the filters,  $x_i$  denotes a logic variable,  $\vec{X}_1$  and  $\vec{X}_2$  denote the vectors of logic variables.

<sup>2</sup> 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.

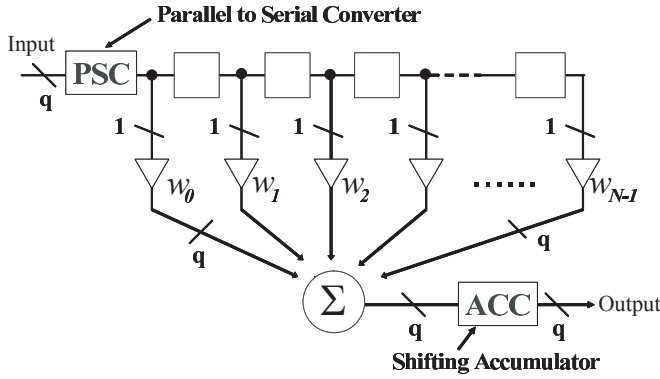


Figure 4.5: Serial Realization of FIR Filter.

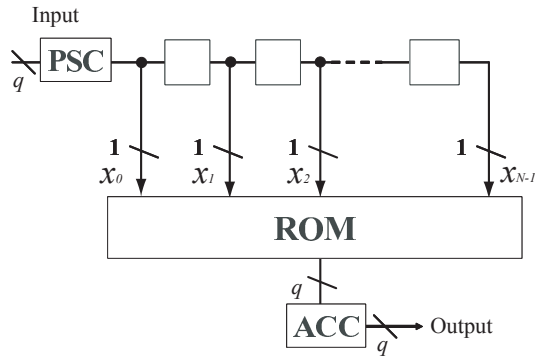


Figure 4.6: Serial ROM Realization of FIR Filter.

be replaced by one memory [1, 12]. 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 [3]. Let the number of output bits be 15, and let the filter coefficients be  $\vec{W} = (378, 188, -521, -1120, -713, 353, 614, -420, -1168, -100, 1538, 920, -1925, -2720, 2167, 10164, 14125)$ . A single ROM realization requires  $2^{17} \cdot 15 = 1,966,080$  bits. Fig. 4.7 shows the LUT cascades for the filter, where the LSBLOCK realizes the least significant 8 bits, and the MSBLOCK realizes the most significant 7 bits. The bounds obtained from Theorems 2.1 and 2.2 are shown in Table 4.2. In this case, the ordering of the input and the output variables are optimized. Especially, the LSBLOCK is reduced drastically since two outputs depend on only 12 variables. The total amount of memory is  $2^{12} \cdot 8 + 2^{11} \cdot 6 + 2^{12} \cdot 10 + 2^{12} \cdot 9 + 2^{12} \cdot 7 = 110,592$  bits. (End of Example)

In FIR filters, the weights  $w_i$  are real numbers, and we

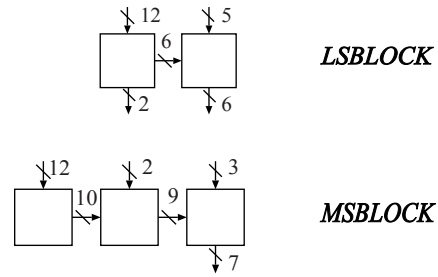


Figure 4.7: FIR Filter (Ordering Optimized).

represent them by fixed point fraction numbers. In this case, we have two different methods to generate the truth table for the FIR filter functions.

**Definition 4.2** Let  $w_0, w_1, \dots, w_{n-1}$  be coefficients of an FIR filter. The **WS1 function** (addition-after-rounding) is generated first by rounding the coefficients  $w_i$  into  $n$ -bit precision, and then adding the coefficients. On the other hand, the **WS2 function** (rounding-after-addition) is generated first by adding the coefficients as real numbers, and then rounding the results into  $q$ -bit precision.

In general, WS1 function and WS2 function are different. The WS1 function satisfies the definition of the WS function, while the WS2 function may not.

We designed 192 different FIR filters by distributed arithmetic. Major results [11] are

1. LUT cascades require much smaller memory than single ROM realizations.
2. The WS1 functions require much smaller LUT cascades than WS2 functions.
3. The WS2 functions produce higher quality filters than WS1 functions.

In the digital signal processing, we often use fixed point numbers to represent real numbers. In this case we cannot avoid **round-off errors**. For example, when  $(0.1011111\dots)_2$  is represented by a binary number of 8-bit precision, we can use either  $(0.1011111)_2$  or  $(0.1100000)_2$ . In many cases, we have an option to select one of the two representations. Although, we can use a logic minimizer for Boolean relation to find the better representation, it is very time consuming. The concept of WS function simply this problem. Distributed arithmetic also can implement Discrete Cosine Transform (DCT), Discrete Fourier Transform (DFT) and other convolution operations.

## 4.5 Threshold Function

**Definition 4.3** A threshold function  $f(x_0, x_1, \dots, x_{n-1})$  satisfies the relation:  $f = 1$  if  $\sum_{i=1}^n w_i x_i \geq T$ , and  $f = 0$

otherwise, where  $(w_0, w_1, \dots, w_{n-1})$  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  $(w_0, w_1, \dots, w_{n-1})$  is at most*

$$UB4 = 1 + \sum_{i=0}^{n-1} w_i. \quad (4.2)$$

**(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. *(Q.E.D.)*

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

## 5 Conclusion and Comments

In this paper, we first defined weighted-sum functions as a mathematical model of bit counting circuits, radix converters and distributed arithmetic. Then, we derived upper bounds on the column multiplicity for the standard decomposition chart for a WS function.

If the weights are bounded above by a polynomial function of  $n$ , then the column multiplicity is also bounded above a polynomial function of  $n$ . In the case of radix converters, the weights exponentially increase with  $n$ . The detailed analysis of radix converters that convert  $p$ -nary numbers into  $q$ -nary numbers is shown in a separate paper [9].

We also presented methods to realize WS functions by LUT cascades. In some cases, the size of LUT cascades increases exponentially with  $n$ . In such a case, we can reduce the size of the cascade by partitioning the outputs into several groups. Another method to reduce the size of cascades is to partition the inputs into several groups. For each group, we can implement a WS function, and then we can obtain the sum by using an adder. This will greatly reduce the necessary amount of memory. Note that LUT cascades can be used for these WS functions and the adder.

## Acknowledgments

This research is supported in part by the Grants in Aid for Scientific Research of JSPS and MEXT, and the grant of Kitakyushu Innovative Cluster Project. Discussion with Prof. Jon T. Butler improved English presentation.

## References

- [1] L. Mintzer, "FIR filters with field-programmable gate arrays," *Journal of VLSI Signal Processing*, Aug. 1993, pp. 120-127.
- [2] K. Nakamura, T. Sasao, M. Matsuura, K. Tanaka, K. Yoshizumi, H. Qin, and Y. Iguchi, "Programmable logic device with an 8-stage cascade of 64K-bit asynchronous SRAMs," *Cool Chips VIII*, IEEE Symposium on Low-Power and High-Speed Chips, April 20-22, 2005, Yokohama, Japan.
- [3] K. K. Parhi, *VLSI Digital Signal Processing Systems Design and Implementation*, John Wiley, New York, 1999.
- [4] T. Sasao, "FPGA design by generalized functional decomposition," In *Logic Synthesis and Optimization*, Sasao ed., Kluwer Academic Publisher, pp. 233-258, 1993.
- [5] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.
- [6] 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.
- [7] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," *Design Automation Conference*, San Diego, CA, USA, June 2-6, 2004, pp. 428-433.
- [8] T. Sasao, J. T. Butler, and M. Riedel, "Application of LUT cascades to numerical function generators," *12th SASIMI Workshop*, Kanazawa, Japan, Oct 18-19, 2004, pp. 422-429.
- [9] T. Sasao, "Radix converters: Complexity and implementation by LUT cascades," *International Symposium on Multiple-Valued Logic*, Calgary, Canada, May 18-21, 2005 (to be published).
- [10] T. Sasao and M. Matsuura, "BDD representation for incompletely specified multiple-output logic functions and its applications to functional decomposition," *Design Automation Conference*, June 2005, (to be published).
- [11] T. Sasao, Y. Iguchi, and T. Suzuki, "On LUT cascade realizations of FIR filters," (Draft).
- [12] S. A. White, "Applications of distributed arithmetic to digital signal processing: a tutorial review," *IEEE ASSP Magazine*, Vol. 6, No. 3, July 1989, pp.4 - 19.