Hardware to Compute Walsh Coefficients

Yukihiro Iguchi [1] Tsutomu Sasao [2]

[1] Department of Computer Science, Meiji University
[2] Department of Computer Science and Electronics, Kyushu Institute of Technology

Abstract

This paper presents a method to compute a fragment of the Walsh coefficients of logic functions using hardware. First, it introduces the Walsh transformation tree, and shows a method to compute Walsh coefficients using the Walsh transformation tree. Next, it shows the hardware realization for the Walsh tree. The amount of hardware to compute a coefficient and the entire coefficients are $O(2^n)$ and $O(n^2 \cdot 2^n)$, respectively. FPGA implementations show their feasibility up to $n = 14$. The FPGA realization is at least 1253 times faster than a software implementation on a microprocessor for $n = 14$.

1. Introduction

Spectrum analysis of logic functions [13] is useful logic synthesis [20, 8, 10], Boolean matching [5, 7], test [17, 9, 11], and verification [18]. Various methods to compute spectrum are known: Fast Fourier transform (FFT) [3], cubes [20, 6], and decision diagrams [5, 7].

A disadvantage of the spectral method is that the sizes of the representations tend to be large, especially when the entire spectrum is represented at one time. In many applications, only a fragment of the spectrum coefficients is sufficient, and thus a smaller amount of computation time is needed compared to that of the entire spectrum. Most research on spectrum computation focus on software. Especially, [7] and [12] have considered efficient computation methods for a fragment of Walsh spectrum of a given logic functions.

In this paper, however, we use hardware to compute a fragment of the Walsh spectrum. Theoretically, the FFT realization computes the entire Walsh spectrum at one time, However, in practice, the straightforward FFT realization requires an excessive amount of hardware to implement by an FPGA. Thus, [2] proposes a bit-serial method to compute the spectrum. It is hardware typically used for digital signal processing, and assumes the following conditions:

1. The entire spectrum is computed at one time.
2. Each input is a signal of 8 bits.

In this paper, we consider hardware to compute a part of Walsh spectrum, and assume the following conditions:

1. A part of coefficients of the spectrum is computed at one time.
2. Each input is a signal of a single bit. (We compute the spectrum of single-output logic function. Extension to multiple-output function is shown in Section 4.)

Such a hardware is applicable for the fault diagnosis of semiconductor memories [11], and Boolean matching [5].

2. Definitions and Basic Properties

In this part, we define Walsh spectrum, Walsh transformation trees and Walsh transformation diagrams. Also we show a method to compute Walsh coefficients from the Walsh transformation tree and the Walsh transformation diagram [15].

2.1. Walsh Transformation

Definition 2.1 *Let*

$$\mathbf{W}(n) = \begin{bmatrix} \mathbf{W}(n-1) & \mathbf{W}(n-1) \\ \mathbf{W}(n-1) & -\mathbf{W}(n-1) \end{bmatrix},$$

and

$$\mathbf{W}(1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

$\mathbf{W}(n)$ *is the* **Walsh transformation matrix of** n **variables**, *and* $\mathbf{W}(1)$ *is the* **basic Walsh transform matrix**. *The* **inverse** *of a Walsh transformation matrix* $\mathbf{W}(n)$ *is* $2^{-n}\mathbf{W}(n)$.

Definition 2.2 *Let* $\vec{F} = (f_0, f_1, \ldots, f_{2^n-1})$ *be the* **truth vector** *of an* n*-variable logic function* f, *and let* $\vec{S} = (s_0, s_1, \ldots, s_{2^n-1})$ *be the* **Walsh spectrum** *of* f. *Then,*

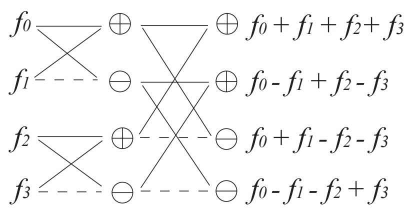

Figure 2.1. Fast Fourier transformation (Butterfly operation).

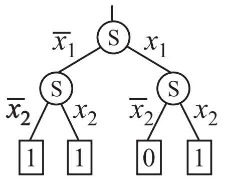

Figure 2.2. Binary decision tree for $f = \bar{x}_1 \vee x_2$.

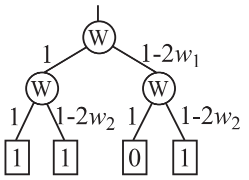

Figure 2.3. Binary decision tree for $f = \bar{x}_1 \vee x_2$.

two relations $\vec{S} = 2^{-n}\mathbf{W}(n)\vec{F}^t$ and $\vec{F} = \mathbf{W}(n)\vec{S}^t$ hold, where t denotes transpose of the vector. In this case, $s_i(i = 0, 1, \ldots, 2^n - 1)$ is a **Walsh coefficient** *of f. In this paper, we call this the* **Walsh transformation** *(it is sometimes called as* **Walsh-Hadamard transformation** *or* **Hadamard transformation***.) Each row of the Walsh transformation matrix represents a* **Walsh function***.*

In the computation of the Walsh spectrum for logic functions, we often omit the multiplication of the constant factor 2^{-n}. The Walsh transformation of an n-variable logic function is computed as the product of $\mathbf{W}(n)$ and $\vec{F}^t$. However, a straightforward computation method requires a large matrix, and thus a large amount of computation. The fast Fourier transformation method (i.e., butterfly operations) shown in Fig. 2.1, computes the Walsh spectrum more efficiently.

Example 2.1 *Consider the function $f = \bar{x}_1 \vee x_2$. Thus, $\vec{F} = (f_0, f_1, f_2, f_3) = (1, 1, 0, 1)$. The Walsh spectrum is*

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ -1 \\ 1 \\ 1 \end{bmatrix}.$$

Therefore, we have $\vec{S} = (s_0, s_1, s_2, s_3) = (3, -1, 1, 1)$. ■

2.2. Walsh Transformation Tree

Definition 2.3 *The* **binary decision tree** *(BDT) for a logic function f is obtained from f by applying the Shannon expansion $f = \bar{x}f_0 \vee xf_1$, repeatedly. In the BDT, a 0-edge has the label $\bar{x}_i$, while a 1-edge has the label x_i. Each node is labeled "S" to denote the Shannon expansion.*

Definition 2.4 *Given a BDT, in a path from the root node to a terminal node, the products of all the labels in the edges in the path and the value of the terminal node is the* **path product***. The sum of all the path products is the* **sum-of-path-products (SOPP)***.*

In the case of BDT for f, the SOPP corresponds to the canonical sum-of-products expression for f.

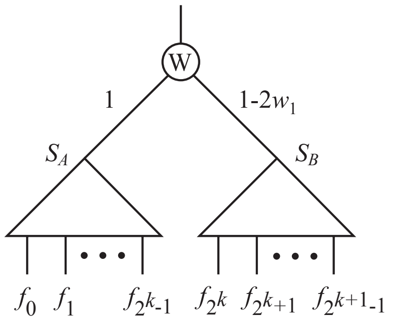

Figure 2.4. Walsh transformation tree for $k+1$ variables.

Example 2.2 *Fig. 2.2 shows the BDT for $n = 2$. The SOPP is $1 \cdot \bar{x}_1\bar{x}_2 + 1 \cdot \bar{x}_1 x_2 + 0 \cdot x_1\bar{x}_2 + 1 \cdot x_1 x_2 = \bar{x}_1\bar{x}_2 + \bar{x}_1 x_2 + x_1 x_2$.* ■

Definition 2.5 *A* **multi-terminal binary decision tree** *(MTBDT) is the BDT, where the leaf nodes have m bits ($m \geq 2$). An MTBDT represents a multiple-output logic function $(f^0, f^1, \ldots, f^{m-1})$, and each bit of the terminal nodes represents a component function $f^i(i = 0, 1, \ldots, m-1)$.*

Definition 2.6 *[15] A* **Walsh transformation tree** *(WTT) is a binary decision tree (BDT), where the 0-edge has the label 1, while the 1-edge has the label $(1 - 2w_i)$. Each node is labeled "W" to denote the Walsh expansion.*

Fig. 2.3 shows an example for $n = 2$.

Definition 2.7 *The* **Walsh expression** *is SOPP of the Walsh transformation tree. That is, let $f_0, f_1, f_2, f_3, \ldots, f_{2^n-1}$ be the values of the terminal nodes. Then, the Walsh expression for f is*

$$\begin{aligned} & s(f : w_1, w_2, \ldots, w_n) \\ &= f_0 + f_1 \cdot (1 - 2w_n) + f_2 \cdot (1 - 2w_{n-1}) \\ &\quad + f_3 \cdot (1 - 2w_{n-1}) \cdot (1 - 2w_n) + \cdots \\ &\quad + f_{2^n-1} \cdot (1 - 2w_1) \cdot (1 - 2w_2) \cdot \cdots \cdot (1 - 2w_n). \end{aligned}$$

In the Walsh expression, by specifying the value of $\vec{w} = (w_1, w_2, \ldots, w_n)$, we can compute an arbitrary Walsh coefficient. That is, the WTT represents a row of $\mathbf{W}(n)$, and the value $(w_1, w_2, \ldots, w_n)$ specifies the row. When $w_1 = w_2 = \cdots = w_n = 0$, the SOPP represents $f_0 + f_1 + \cdots + f_{2^n-1} = s_0$. This corresponds to the inner product of the truth vector and the first row of $\mathbf{W}(n)$. When $w_1 = w_2 = \cdots = 0, w_n = 1$, the SOPP represents $(f_0 - f_1) + (f_2 - f_3) + \cdots + (f_{2^n-2} - f_{2^n-1}) = s_1$. This corresponds to the inner product of the truth vector and the 2nd row of $\mathbf{W}(n)$. And, similarly, by specifying the value of $\vec{w} = (w_1, w_2, \ldots, w_n)$, we can compute the inner product the arbitrary row of $\mathbf{W}(n)$ and $\vec{F}$.

Example 2.3 *The Walsh expression obtained from the WTT in Fig 2.3 is*

$$\begin{aligned} s(w_1, w_2) &= f_0 \cdot 1 \cdot 1 + f_1 \cdot 1 \cdot (1 - 2w_2) + f_2 \\ &\quad \cdot (1 - 2w_1) \cdot 1 + f_3 \cdot (1 - 2w_1) \cdot (1 - 2w_2) \\ &= 1 \cdot 1 \cdot 1 + 1 \cdot 1 \cdot (1 - 2w_2) + 0 \\ &\quad \cdot (1 - 2w_1) \cdot 1 + 1 \cdot (1 - 2w_1) \cdot (1 - 2w_2) \\ &= 3 - 2w_1 - 4w_2 + 4w_1 w_2. \end{aligned}$$

From this, we can compute the Walsh coefficients as follows:

$$\begin{aligned} s(0,0) &= f_0 + f_1 + f_2 + f_3 = 3 \\ s(0,1) &= f_0 - f_1 + f_2 - f_3 = -1 \\ s(1,0) &= f_0 + f_1 - f_2 - f_3 = 1 \\ s(1,1) &= f_0 - f_1 - f_2 + f_3 = 1 \end{aligned} \qquad \blacksquare$$

Theorem 2.1 *In a Walsh transformation tree (WTT), when the leaf nodes represent the truth values of a logic function f, the SOPP represents the Walsh coefficient specified by $\vec{w} = (w_1, w_2, \ldots, w_n)$.*

(Proof) We will prove the theorem by the mathematical induction with respect to n, the number of variables in f.

1. When $n = 1$, the theorem holds. That is, when $w_1 = 0$ the SOPP of the WTT represents $f_0 + f_1$, while when $w_1 = 1$ the SOPP represents $f_0 - f_1$.
2. Assume that the theorem is true for the WTT with k variables.
3. Consider the case for $n = k+1$. In this case, the WTT can be represented as Fig. 2.4. When $w_1 = 0$, the SOPP of the WTT represents $s_A(w_2, w_3, \ldots, w_{k+1}) + s_B(w_2, w_3, \ldots, w_{k+1})$. Also, when $w_1 = 1$, the SOPP of the WTT represent $s_A(w_2, w_3, \ldots, w_{k+1}) - s_B(w_2, w_3, \ldots, w_{k+1})$. This means that the WTT represent the matrix

$$\mathbf{W}(k+1) = \begin{bmatrix} \mathbf{W}(k) & \mathbf{W}(k) \\ \mathbf{W}(k) & -\mathbf{W}(k) \end{bmatrix}.$$

   From this, we can see that the theorem also holds for $n = k + 1$.

From 1, 2, and 3, the theorem holds for all n. $\square$

2.3. Walsh Transformation Diagram

Definition 2.8 *[4] The* **binary decision diagram** *(BDD) is obtained from the BDT by applying the following operations repeatedly to reduce the number of nodes:*

1. *Delete a node v that has the same children, and*
2. *Share the node, if two nodes v_1 and v_2 represent the same function.*

Definition 2.9 *[15] The* **Walsh transformation diagram** *(WTD) is obtained from the WTT by applying the following operations repeatedly to reduce the number of nodes:*

1. *Delete a node v that has the same children. In this case, attach an edge with the label $2(1 - w_i)$.*
2. *Share the node, if two nodes v_1 and v_2 represent the same function.*

Note that, in a BDD, when a node v has the same children, the reduced graph has an edge with the label $\bar{w}_i + w_i = 1$. On the other hand, in a WTD, when a node v has the same children, the reduced graph has an edge with the label $1 + (1 - 2w_i) = 2(1 - w_i)$.

Theorem 2.2 *In the Walsh transformation diagram (WTD) for an n variable function, suppose that the terminal nodes represent the truth values of a logic function f. Then, the SOPP of the WTD represents a Walsh expression.*

Example 2.4 *In the BDD in Fig 2.5(a), a node at the x_2 level is deleted. The deleted node is represented by a* **cross point** *[15], denoted by • in the figure. The corresponding WTD is shown in Fig. 2.5(b). The Walsh expression is*

$$\begin{aligned} s(w_1, w_2) &= 1 \cdot 1 \cdot 2(1 - w_2) + 0 \cdot (1 - 2w_1) \\ &\quad \cdot 1 + 1 \cdot (1 - 2w_1) \cdot (1 - 2w_2) \\ &= 3 - 2w_1 - 4w_2 + 4w_1 w_2. \end{aligned}$$

Note that this expression is equivalent to $s(w_1, w_2)$ in Example 2.3. From this, we can compute the Walsh coefficients. $\blacksquare$

3. Amount of Hardware

3.1. Computing a Single Coefficient

A hardware realization of a WTT can be obtained by replacing each node in the WTT by an adder-subtracter. A

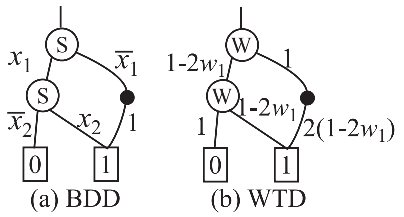

Figure 2.5. BDD and WTD for $f = \bar{x}_1 \vee x_2$.

k-bit **adder-subtracter** realizes the function $y(w, s_a, s_b) = s_a + (1-2w)s_b$, where s_a and s_b are k-bit binary numbers. Note that $y(w, s_a, s_b)$ represents addition $(s_a + s_b)$ when $w = 0$, and subtraction $(s_a - s_b)$ when $w = 1$, where w is the control input. A k-bit adder-subtracter has $(2k+1)$ inputs and $(k+1)$ outputs. We assume that the cost of hardware for a k-bit adder-subtracter is αk, where α is a constant. The hardware for WTT has a structure of binary tree. Note that the adder-subtracters that are near to the root node have higher cost than ones that are near to the leaf nodes. However, we can prove that the total cost of hardware is exactly $O(2^n)$.

Lemma 3.1 $\sum_{i=1}^{n} i2^{i-1} = (n-1)2^n + 1.$

Theorem 3.1 *Hardware cost for the WTT of n variables is $O(2^n)$.*

(Proof) Let w_1 correspond to the root node, and w_n correspond to the nodes that connect directly to the leaves. The cost of the adder-subtracter for w_i is $\alpha(n-i+1)$. We need 2^{i-1} adder-subtracters for w_i. Thus, the total cost of the hardware is

$$\begin{aligned} &\sum_{i=1}^{n} \alpha(n-i+1)2^{i-1} \\ &= \alpha(n+1)\sum_{i=1}^{n} 2^{i-1} - \alpha \sum_{i=1}^{n} i2^{i-1} \\ &= \alpha(n+1)(2^n - 1) - \alpha[(n-1)2^n + 1] \\ &= \alpha(2^{n+1} - n - 2) = O(2^n) \qquad \square \end{aligned}$$

3.2. Computing the Entire Coefficients

Theorem 3.2 *The cost of hardware that computes the entire Walsh coefficients of n variables at one time is $O(n^2 2^n)$.*

(Proof) By replacing each node in the butterfly diagram with an adder or a subtracter, we have hardware to compute the entire Walsh coefficients at one time. For each stage we need 2^{n-1} copies of adders and subtracters. Also, the cost of an adder-subtracter in the i-th stage is βi, where β is a constant. Thus, the total cost of the hardware is

$$\sum_{i=1}^{n} \beta i 2^n = \beta 2^n \sum_{i=1}^{n} i = \beta 2^n \frac{n(n-1)}{2} = O(n^2 2^n) \qquad \square$$

4. Multiple-Output Functions

With the integer function[15]: $Z = 2^{m-1}f^{m-1} + 2^{m-2}f^{m-2} + \cdots + 2^0 f^0$, we can represent a multiple-output function $\vec{f} = (f^{m-1}, f^{m-2}, \ldots, f^0)$ by a WTT or a WTD [1] .

Theorem 4.1 *Let $s(f^i : w_1, w_2, \ldots, w_n)$ be the SOPP for $f^i (i = 0, 1, \ldots, m-1)$. Then, the SOPP for Z is,*

$$\sum_{i=0}^{m-1} s(f^i : w_1, w_2, \ldots, w_n) \cdot 2^i.$$

From Theorem 4.1, we can calculate the Walsh coefficient for the multiple-output function as follows: For $i = 0, 1, \ldots, m-1$, obtain the sum of Walsh coefficients for f^i multiplied by 2^i.

The Walsh coefficients of a multiple-output function can be also obtained from the MTBDT. However, the straightforward implementation of MTBDT requires excessive hardware. In the method of Theorem 4.1, most of the hardware is independent of m, the number of outputs. The only hardware that depends on m is the adder in the final stage. This realization drastically reduces the amount of hardware, but the computation time will be proportional to m.

5. Experimental Results

5.1. Circuits to Compute Single Coefficient

In Sections 2.2 and 2.3, we presented two methods to compute the coefficients: WTT and WTD. In this part, we only consider the hardware realization of WTTs, since the method using WTD is feasible only for fixed functions.

In the computation of spectrum for logic functions, two encodings exist: one is $(0,1)$ encoding, and the other is $(1,-1)$ encoding [10]. In this paper, we use the $(0,1)$ encoding. In this case, the maximum value of the spectrum for an n-variable function is 2^n, and the minimum value is -2^{n-1}.

Fig. 5.1(a) shows the **combinational method** for $n = 3$. Fig. 5.1(b) illustrates the computation for $\vec{w} =$

[1] Recall that f_0 denotes $f(0, 0, \ldots, 0)$, while f^0 denotes 0-th output of the function.

Table 5.1. Encoding for adder-subtracter.

Code			Number in 2's complement	Number in adder-subtracter
0	0	0	0	0
0	0	1	1	1
0	1	0	2	2
0	1	1	3	3
1	0	0	-4	4
1	0	1	-3	-3
1	1	0	-2	-2
1	1	1	-1	-1

$(w_1, w_2, w_3) = (1, 1, 0)$. As shown in Fig. 5.1(c), the adder-subtracter of a WTT has $(2k + 1)$ inputs and $k + 1$ outputs. In this realization, to reduce the amount of hardware, we use a special encoding: The code $(1, 0, 0, \ldots, 0)$ represents 2^k, while other codes represent 2's complement numbers. For example, Table 5.1 represents encoding for $k = 2$. In this case, the code $(1, 0, 0)$ represents $2^k = 4$.

Table 5.2 shows the environment and conditions of the experiments. Table 5.3 shows the number of ALUTs and delay time for the combinational method. For the FPGA device used in this experiment, the combinational method is feasible with up to $n = 10$. When $n = 10$, only 2 percents of total ALUTs are used. The number of ALUTs increases exponentially with n.

To compute the spectrum with $n \geq 11$, we developed the **time-division multiplexing method (TDM method)**, as shown in Fig. 5.2. In this method,

1. we partition the data inputs $f_i, (i = 0, 1, \ldots, 2^n - 1)$ into groups,
2. for each group load the data of the functions into the registers sequentially by setting the $Load_i$ signal to 1, and
3. after loading all the values of functions into registers, compute the value of the Walsh function.

Fig 5.2 illustrates the case where the width is 4 bits. With the FPGA device, the width of the data can be extended to 1024 bits. Table 5.4 shows the amount of hardware and delay time for TDM method. With this device, the method is feasible with up to $n = 14$.

5.2. Circuits to Compute All the Coefficients

We also implemented circuits to compute all the coefficients at one time. The networks simply realize butterfly networks shown in Fig. 2.1. Up to $n = 7$, we could implement combinational circuits to compute all the coefficients at one time. Table 5.5 shows the numbers of ALUTs and delay time. For, $n \geq 8$, the numbers of pins in the FPGA are not sufficient, so we used the TDM method. Table 5.6 shows the amount of hardware and delay time. From these tables, we can see that the numbers of ALUTs increase with $O(n^2 \cdot 2^n)$.

Table 5.2. Environment and conditions for experiments.

FPGA device: Stratix II	
Device type:	EP2S180F1508C4[1]
Number of ALUTs:	143520
I/O pins:	1173 (out of 1508 total pins)
Memory bits:	9383040
DSP blocks (9-bit):	768
Computer	
PC:	IBM ThinkPad T41
MPU:	Pentium M (1.6GHz)
RAM:	1GB
Tool for Logic Synthesis and Fitting	
Altera, Quartus II V4.1 [1]	
Optimization Parameter for the Tool	
Fitter setting: Physical Synthesis Optimization, Physical synthesis for combinational logic Extra	
Timing-driven compilation: Optimize timing, Extra effort,	
Analysis Synthesis Settings: Standard Fit (highest effort), Speed	

Table 5.3. Combinational method to compute single coefficient.

n	Pins	ALUTs	Delay$[ns]$
6	77	230 (<1%)	13.3
7	143	471 (<1%)	15.2
8	273	954 (<1%)	18.7
9	531	1924 (1%)	21.0
10	1045	3965 (2%)	26.2

5.3. Comparison with Microprocessor

Various methods exist to compute Walsh coefficients by software. As for the data structure, we assume the array of the truth vector. For computation of any coefficient, we need to access all the 2^n elements of the truth vector, and to do $(2^n - 1)$ additions and/or subtractions. So, to calculate

Table 5.4. TDM method to compute single coefficient.

n	Pins	ALUTs	Registers	Delay$[ns]$
11	1050	8764 (6%)	2048 (1%)	39.3
12	1054	17547 (12%)	4095 (2%)	43.0
13	1060	35118 (24%)	8192 (5%)	47.0
14	1070	70256 (48%)	16374 (10%)	51.7

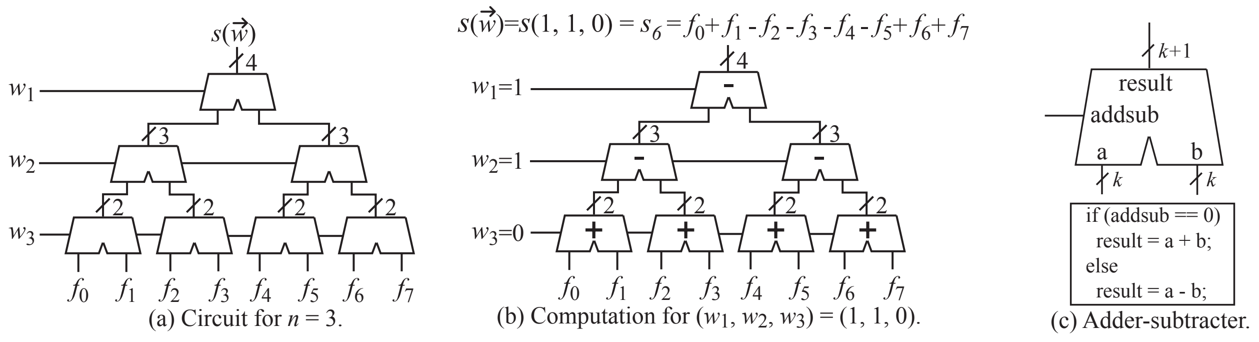

Figure 5.1. Circuit to compute Walsh coefficients ($n = 3$, combinational method).

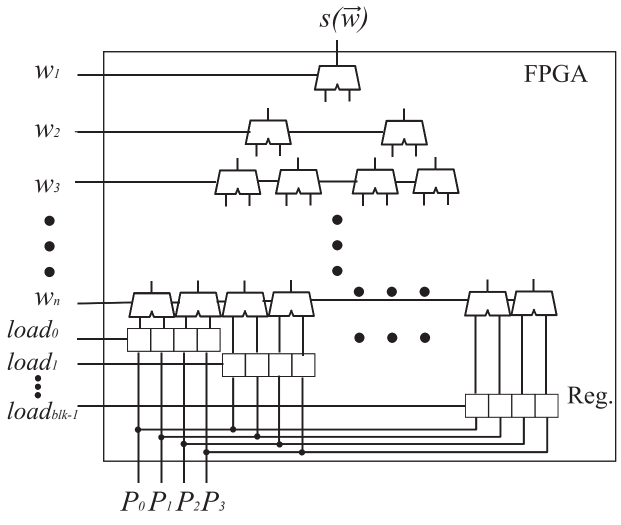

Figure 5.2. Computation circuit for Walsh coefficients (TDM method).

Table 5.5. Combinational method to compute all the coefficients.

n	Pins	ALUTs	Delay$[ns]$
4	96	218 (<1%)	8.4
5	224	676 (<1%)	11.4
6	512	1896 (1%)	15.4
7	1152	5072 (3%)	18.8

a Walsh coefficient, at least we need time to computer $f_0 + f_1 + \cdots + f_{2^n-1}$. Thus, for each n, we generated a code to compute the value $f_0 + f_1 + \cdots + f_{2^n-1}$. We obtained the average computation time by performing it 10^6 times. We used the computer shown in Table 5.2, and used the gcc compiler.

Table 5.6. TDM method to compute all the coefficients.

n	Pins	ALUTs	Delay$[ns]$
8	273	14398 (10%)	24.4
9	532	35496 (25%)	32.9
10	1046	85395 (59%)	44.7

Table 5.7. Comparison of computation time.

n	time ($n$sec)		Speed-up
	FPGA	MPU	
8	18.7	962	51
9	21.0	1903	91
10	26.2	3914	149
11	39.3	8181	208
12	43.0	16100	372
13	47.0	32200	685
14	51.7	64800	1253

Table 5.7 compares computation time. In the case of the microprocessor (MPU), the computation time is proportional to 2^n. From the table, we can see that the FPGA realization is at least 1253 times faster than the MPU when $n = 14$. Note that the software implementations in [5, 12] can compute the coefficients only for the fixed functions, and require precomputation. On the other hand, in our implementation, we can compute the coefficients for any function without any precomputation.

6. Conclusion

In this paper, we have shown hardware to compute a Walsh coefficient of a logic function directly from the Walsh transformation tree. Also, we have designed the circuits using FPGAs. With the current FPGAs, our approach is feasible for $n \leq 14$ inputs. It is at least 1253 times faster than a software realization on a microprocessor when $n = 14$. We have also shown that the amount of hardware to compute a coefficient and the entire coefficients are $O(2^n)$ and $O(n^2 \cdot 2^n)$, respectively.

Acknowledgments

This research is partly supported by the Grant in Aid for Scientific Research of JSPS and MEXT, and Kitakyushu Innovative Cluster Project. Discussion with S. Nagayama and Jon T. Butler improved presentation of the paper.

References

[1] Altera: http://www.altera.com/

[2] A. Amira, A. Bouridane, P. Milligan, and M. A. Roula, "Novel FPGA implementations of Walsh Hadamard transforms for signal processing," *IEE Proceedings on Vision, Image and Signal Processing*, pp. 377-383, Vol. 148, No. 6, December 2001.

[3] K. G. Beauchamp, *Applications of Walsh and Related Functions*, New York: Academic Press, 1984.

[4] R. E. Bryant, "Graph-based algorithms for Boolean functions manipulation," *IEEE Trans. Computers,* vol. 35, no. 8, pp. 667-691, Aug. 1986.

[5] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with application to technology mapping," *Proc. Design Automation Conf.*, pp. 54-60, June, 1993.

[6] B. J. Falkowski, I. Schaefer, and M. A. Perkowski, "Effective computer methods for the calculation of Rademacher-Walsh spectrum for completely and incompletely specified Boolean functions," *IEEE Trans. Computer-Aided Design,* vol. 11, no. 10, pp. 1207-1226, Oct. 1992.

[7] M. Fujita, J. C.-Y. Yang, M. Clarke, X. Zhao, and P. McGeer, "Fast spectrum computation for logic functions using binary decision diagrams," *Proc. Int'l Symp. Circuit and Systems (ISCAS '94)*, pp. 275-278, 1994.

[8] J. P. Hansen and M. Sekine, "Synthesis by spectral translation using Boolean decision diagrams," *Proc. 33rd Design Automation Conf.*, pp. 248-253, June, 1996.

[9] T-C. Hsiao and S. C. Seth, "An analysis of the use of Rademacher-Walsh spectrum in compact testing," *IEEE Trans. Computers*, vol. 33, pp. 934-938, Oct. 1984.

[10] S. L Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic*, Academic Press, 1985.

[11] A. Iseno, Y. Iguchi, and T. Sasao, "Fault diagnosis for RAMs using Walsh spectrum," *IEICE Trans. Information and Systems*, Vol. E87-D, No.3, March 2004, pp. 592-600.

[12] D. Jankovic, R. S. Stankovic, and R. Drechsler, "Decision diagram method for calculation of pruned Walsh transform," *IEEE Transactions on Computers*, Vol. 50, No. 2, Feb. 2001, pp. 147-157.

[13] R. J. Lechner, "Harmonic analysis of switching functions," *In A. Mukhopadhyay, editor, Recent Developments in Switching Theory,* New York, Academic Press, 1971.

[14] Synplify: http://www.synplicity.com/

[15] R. S. Stankovic, T. Sasao, and C. Moraga, "Spectral transform decision diagrams," *Representations of Discrete Functions*, T. Sasao and M. Fujita, eds., pp. 55-92, Kluwer Academic, 1996.

[16] R. Stankovic and J. Astola, *Spectral Interpretation of Decision Diagrams, Springer-Verlag,* May 16, 2003.

[17] A. K. Susskind, "Testing by verifying Walsh coefficients," *IEEE Transactions on Computers*, Vol. 32, No.2, pp. 198-201, Feb. 1983.

[18] M. A. Thornton, R. Drechsler, and D. M. Miller, *Spectral Techniques in VLSI CAD*, Kluwer Academic Publishers, July, 2001.

[19] M. A. Thornton and V. S. S. Nair, "Efficient calculation of spectral coefficients and their applications," *IEEE Trans. Computer-Aided Design Integrated Circuits and Systems*, Vol. 4, no. 11, pp. 1328-1341, Nov. 1995.

[20] D. Varma and E. A. Trachtenberg, " Design automation tools for efficient implementation of logic functions by decomposition," *IEEE Trans. CAD,* Vol. 8, pp. 901-916, Aug. 1989.