Computing Walsh, Arithmetic, and Reed-Muller Spectral Decision Diagrams Using Graph Transformations*

Whitney J. Townsend
Electrical and Computer Engineering
Mississippi State University
Mississippi State, MS
wjt1@ece.msstate.edu

Mitchell A. Thornton
Electrical and Computer Engineering
Mississippi State University
Mississippi State, MS
mitch@ece.msstate.edu

Rolf Drechsler
Institute of Computer Science
University of Bremen
Bremen, Germany
drechsle@informatik.uni-bremen.de

D. Michael Miller
Department of Computer Science
University of Victoria
Victoria, BC, Canada
mmiller@csr.uvic.ca

ABSTRACT

Spectral techniques have found many applications in computer-aided design, including synthesis, verification, and testing. Decision diagram representations permit spectral coefficients to be calculated via graph-based algorithms. In this paper, algorithms are described for transforming multi-output functions to produce Walsh, arithmetic, and Reed-Muller spectral decision diagrams and the experimental results of those implementations are presented.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids – *automatic synthesis, optimization, switching theory, verification.*

J.6. [**Computer-Aided Engineering**]: Computer -aided design.

General Terms

Algorithm, Design, Verification.

Keywords

Data Structures, Decision Diagrams, Discrete Functions, Spectral Methods.

1. INTRODUCTION

Spectral methods and decision diagrams have been applied to many areas of digital systems design. These include synthesis [3, 10, 11, 15], function classification [11], partitioning techniques [24], verification [13, 16, 22], and testing [20]. Spectral techniques can offer a view of a problem that illuminates different properties than are readily evident in the functional domain. Traditionally, the computational costs of spectral coefficients have been too high for many of these techniques to see practical application using linear algebra-based methods of computation or fast transform techniques [11].

Decision diagrams [1, 4] are the state-of-the-art representation for Boolean functions in *Computer-Aided Design* (CAD) applications. It is thus very attractive to consider decision diagrams when considering alternatives such as spectral techniques. This paper concentrates on transforming functional decision diagrams into spectral decision diagrams, a fundamental step in the application of spectral techniques to any area.

The use of decision diagrams as a compact representation of discrete functions provides for a variety of ways that spectra may be computed or represented [6, 7, 8, 14, 18, 19, 21, 23]. The differences between the transformation operations described here and previous techniques is that here multi-output functions represented as shared decision diagrams are transformed simultaneously with respect to each output in contrast to processing each output individually and the incorporation of a caching mechanism during the transformation procedure that allows for significant decreases in computer runtime.

The description of the implementation of graph-based spectral transformation algorithms for decision diagrams representing multi-output circuits is included and experimental results of these implementations for the Walsh, the arithmetic, and the Reed-Muller spectra are provided. Additionally an improvement

*This work was supported in part by the National Science Foundation under grants CCR-0000891 and CCR-0097246.

to the algorithm provided by caching "skipped" nodes during decision diagram traversals is also implemented and those results are presented in comparison with the result of the original algorithms [14, 21].

The organization of this paper is as follows. Section 2 contains the necessary background information. Section 3 explains the multi-output algorithm through pseudo-code and then illustrates it with an example for a small multi-output circuit. Section 4 presents the experimental results including the number of nodes and coefficients for the resulting spectral decision diagrams and also the runtimes needed for the transformations. Section 5 provides concluding remarks on the work presented.

2. BACKGROUND

In the method described here, a multi-output circuit represented as a *Binary Decision Diagram* (BDD) is transformed into a *Spectral Decision Diagram* (SDD). The resulting SDD is represented as a *Multi-Terminal Binary Decision Diagram* (MTBDD) [3] in which each non-terminal node has two outgoing edges, one edge representing the Boolean value 0 and the other edge representing the Boolean value 1. No edge complementation is used within the MTBDD. It is possible to further reduce the size of the SDD using edge negations as described in [14]. The terminal nodes of the MTBDD can take on any integer value allowing for the representation of spectral coefficient values at the terminal nodes.

Traversing the decision diagram and transforming each Shannon node encountered with the 2x2 matrix for the desired transform accomplishes the transformation from the Boolean domain to the spectral domain. When all nodes have been transformed in this way the result is the SDD for the circuit with the spectral coefficients present at the terminal nodes. A recursive Kronecker product definition [9] is used in the algorithms as follows in Equation (1).

$$G^n = \bigotimes_{i=1}^{n} G^1 \qquad (1)$$

G in Equation (1) is replaced by the appropriate matrix from those given in Equation (2) depending on the transformation desired, either the Walsh (W), the arithmetic (A), or the Reed-Muller (M).

$$W^1 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix} A^1 = \begin{bmatrix} +1 & 0 \\ -1 & +1 \end{bmatrix} M^1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad (2)$$

Recall that the Walsh and the arithmetic transforms are applied over the integer field, while the Reed-Muller transform is applied over *GF(*2).

A noteworthy fact is that SDDs describing the arithmetic transform of a function are in fact *Binary Moment Diagrams* (BMD) [2] and that Reed-Muller SDDs are *Functional Decision Diagrams* (FDD) [5, 12] without the presence of complemented edges. The inverse transforms can also be easily implemented allowing for techniques to transform directly from BDDs to FDDs and from BDDs to BMDs and vice versa. It is easy to see that BMDs result from the application of the arithmetic transform since examination of the 2x2 matrix in Equation (2) results in a pseudo-Boolean decomposition of the original function and likewise, the Reed-Muller transformation is actually a matrix representation of a positive-Davio decomposition.

The transformation occurs in a depth-first fashion, and no node can be transformed until all nodes below it have been transformed. Special consideration must be given to those portions of the BDD in which a variable is present in both polarities and thus is not present on the path, a so-called "skipped" node. The nodes below this "skipped" node must be transformed as if the "skipped" node was present. It is often the case that "skipped" nodes occur in BDDs since this results from the application of the reduction rules. The fewer the number of skipped nodes in a BDD, the more closely the BDD approaches an exponentially sized Shannon expansion tree.

3. METHODOLOGY

The following pseudo-code illustrates the implementation of the graph-based transformation algorithm for multi-output functions. This is the framework for the technique used for all three algorithms by including the appropriate terminal manipulations after the check for terminal nodes in the *else* and *then* branches of the pseudo-code. For each output the pointer to the top node for that output is passed to a traversal function that controls the transformation. The pseudo-code for this traversal code is shown below.

```
Traverse (f)
  if (f is a terminal node) return
  Low = Traverse (Low(f))
  High = Traverse (High(f))
  LowTemp = TransformLow(Low, High)
  HighTemp = TransformHigh(Low, High)
  return (NewNonterminal(Index(f), HighTemp, LowTemp))
```

The *TransformLow* and *TransformHigh* transformation functions are identical except for the action taken once the terminal nodes are reached. For illustration the code for *TransformLow* that transforms the *else*-child is shown below followed by the specific terminal manipulations in pseudo-code for the terminal branches of all three transforms. The *TransformLow* function has four possible courses of action: 1) if both nodes are terminal nodes it performs the required manipulation and returns the new terminal node; 2) if both nodes are non-terminal nodes at the same level, a new node is formed with children found by transforming the corresponding children of the two original nodes; 3) if the level of the *else*-child is greater than the level of the *then*-child, a "skipped" node is present in the *else* branch and must be considered; and 4) if the level of the *then*-child is greater than the level of the *else*- child, a "skipped" node is present in the *then* branch and must be considered. Detail of the handling of skipped nodes is given in the following pseudo-code.

```
TransformLow(g, h)
  if (g and h are terminals)
    return (appropriate new terminal manipulation)
  else if (Level(g) = Level(h))
    return (NewNonterminal(Index(g), TransformLow(Low(g),
           Low(h)), TransformLow(High(g), High(h))))
```

```
else if (Level (g) > Level (h))
    return(NewNonterminal(Index(h), TransformLow(Twice(g),
  Low(h)), High(h)))
else (Level(g) < Level (h))
    return (NewNonterminal(Index(g), TransformLow(Low(g),
            Twice(h)), High(g)))
```

For the Walsh transform the appropriate manipulation in the *TransformLow* function of the pseudo-code is as follows.

return (NewTerminal(Value(g) + Value(h)))

For the *TransformHigh* function of the Walsh transform the pseudo-code is as follows.

return (NewTerminal(Value(g) - Value (h)))

The arithmetic transform *TransformLow* function terminal manipulation is described by the following pseudo-code.

return (NewTerminal (Value(g) + 0))

For the *TransformHigh* function the arithmetic transform pseudo-code is as follows.

return (NewTerminal (Value(h) - Value(g)))

The *TransformLow* function for the Reed-Muller transform uses the same terminal manipulation as the *TransformLow* function for the arithmetic transform. The *TransformHigh* function for the Reed-Muller transform uses the *TransformLow* function from the Walsh transform stated above although the Reed-Muller transform is applied over *GF(*2).

"Skipped" nodes within the diagram must be considered, and their children transformed as if they were present. As this allowance must be made when transforming both the *else*-child and the *then*-child, an improvement was made to the original algorithm providing for caching of the result of a "skipped" node when it is first encountered so that the result is available on the subsequent encounter. Caching the result of this computation resulted in significant decreases in the computation time required for transformation as shown in Section 4.

For illustration of the algorithm, a small multi-output circuit with two outputs representing an AND gate and an OR gate is shown as a BDD in Figure 1. As is common in spectral transformation [11], the function is represented with +1denoting logic 0 and -1 denoting logic 1. In Figure 2, the multi-output circuit with $f_1=x_1x_2$ and $f_2=x_1+x_2$ is shown undergoing Walsh transformation. The first node to be transformed is x_{2dd}. Next x_1 is transformed, thus f_1's transformation is complete and transformation is ready to begin for the second output. In transforming x_1 note that the algorithm has had to handle a case of uneven levels and to account for the "skipped" node before continuing the transformation. Next the x_2 node of f_2 is transformed. Finally, the entire circuit is now transformed into an SDD and the Walsh spectral coefficients can be read at the terminal nodes.

4. EXPERIMENTAL RESULTS

In this section experimental results are presented that have been carried out on a SUN Ultra 10. This code is implemented in conjunction with the CUDD package [17]. All runtimes are given in CPU seconds with a runtime limit set at 1 hour of CPU time.

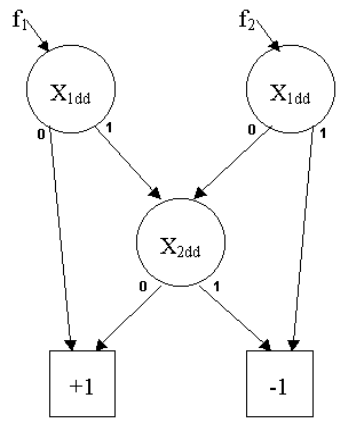

Figure 1. BDD Before Transformation

Each transform was performed on over one hundred benchmark circuits. The circuits represented here were selected based upon the size of the Walsh SDD, restricting the set to diagrams that had over 5000 nodes as large enough to be of interest. Table 1 contains the number of nodes and the number of coefficients for each benchmark circuit for each of the transforms.

In Table 2, runtimes are shown for the original multi-output algorithm and for the improved algorithm utilizing "skipped" node caching as well as the percentage of improvement achieved. Because CPU time was limited to one hour, the transformation of some circuits failed to complete. Approximately 10% of the total benchmark circuits attempted could not be completed within this time limitation.

5. CONCLUSION

An algorithm for computing the spectral coefficients for the Walsh, the arithmetic, and the Reed-Muller transforms of a multi-output circuit represented as a BDD has been presented. The results from the implementation of transformation algorithms with regard to the number of nodes and the number of coefficients in the resulting SDD are also presented. Additionally an improvement to the algorithm that decreases the runtime by caching the results obtained when considering "skipped" nodes has been implemented and the comparison between the runtimes of these two methods has been shown.

The approach presented makes it practical to compute the spectra of large multi-output problems and extends the possibility of applying spectral techniques. The approach is readily extended to the Haar spectral domain and to transformation among the various spectral domains [23].

6. ACKNOWLEDGMENT

The authors would like to thank Dr. Alan Mishchenko for his advice in implementing these algorithms using the CUDD package.

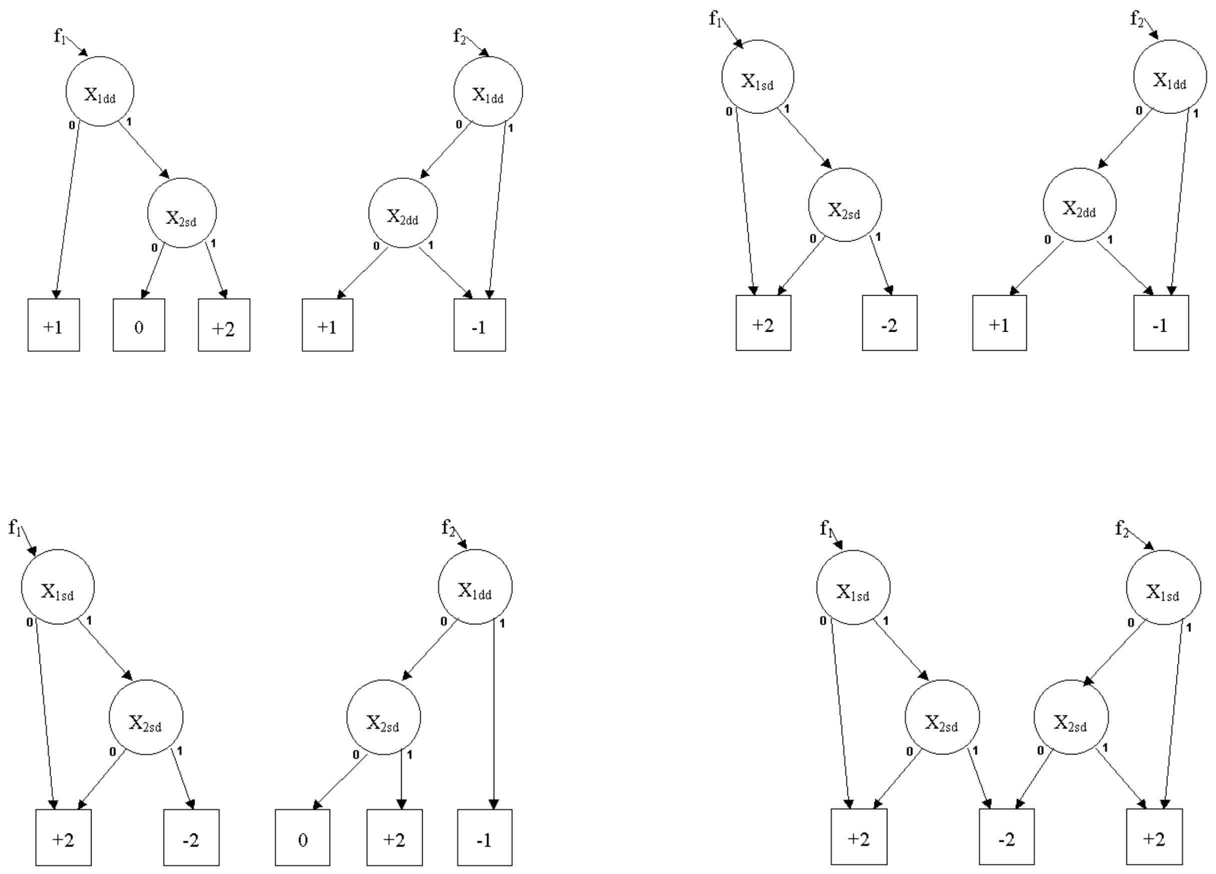

Figure 2. Transformation to a Walsh Spectral Decision Diagram.

Table 1 Complexity Comparisons Between Walsh, Arithmetic, and Reed-Muller Spectral Decision Diagrams

Circuit	**Number inputs**	**Number outputs**	**Walsh**		**Arithmetic**		**Reed-Muller**
			(SDD) nodes	**coefficients**	**(BMD) nodes**	**coefficients**	**(FDD) nodes**
alu4	14	8	6554	171	4652	39	1266
apex5	117	88	7341	226	5757	11	2777
bc0	26	11	21026	2529	5019	11	3000
chkn	29	7	9772	863	2482	14	761
cps	24	109	13388	790	3642	15	1708
duke2	22	29	6601	506	3057	11	1236
ex1010	10	10	6844	116	4604	50	1520
ex4	128	28	15112	384	3217	10	1137
frg2	143	139	6007	398	5209	13	2272
in2	19	10	6087	256	2822	22	958
misex3	14	14	24527	386	3176	29	1140
tial	14	8	7685	171	3739	39	953
vda	17	39	11302	935	4333	15	2117
x1	51	35	10809	1406	2332	5	1040

Table 2. Comparison of Runtimes Between the Original Algorithm and the Improved Algorithm.

Circuit	**Walsh** original (sec)	improved (sec)	%change	**Arithmetic** original (sec)	improved (sec)	%change	**Reed-Muller** original (sec)	improved (sec)	%change
alu4	0.94	0.83	11.7%	0.80	0.68	15.0%	0.71	0.59	16.9%
apex5	-	1900.35	-	-	1690.51	-	-	1561.31	-
bc0	32.52	15.61	52.0%	28.98	13.54	53.3%	27.26	12.42	54.4%
chkn	687.25	404.19	41.2%	620.68	366.41	41.0%	578.86	349.07	39.7%
cps	487.90	264.72	45.7%	441.88	236.28	46.5%	409.60	220.42	46.2%
duke2	15.18	9.42	37.9%	13.35	8.78	34.2%	12.57	8.04	36.0%
ex1010	0.20	0.19	5.0%	0.17	0.17	0.0%	0.15	0.14	6.7%
ex4	3.46	2.72	21.4%	2.93	2.30	21.5%	2.69	2.16	19.7%
frg2	1270.07	552.20	56.5%	1154.11	498.62	56.8%	1111.28	445.46	59.9%
in2	18.47	10.88	41.1%	16.76	10.40	37.9%	15.31	9.47	38.1%
misex3	2.85	2.27	20.4%	2.60	1.94	25.4%	2.36	1.73	26.7%
tial	1.07	0.92	14.0%	0.96	0.82	14.6%	0.86	0.73	15.1%
vda	9.21	4.57	15.4%	8.34	4.30	48.4%	7.92	3.90	50.8%
x1	465.79	274.27	41.1%	434.46	256.90	40.9%	408.72	246.08	39.8%

7. REFERENCES

[1] Bryant, R. E. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp*. 35, 8, (August 1986), 677-691.

[2] Bryant, R. E. and Chen, Y.-A. Verification of arithmetic functions with binary moment diagrams. in *Design Automation Conf.* (San Francisco CA, June 1995), 535-541.

[3] Clarke, E. M., McMillan, K. L., Zhao, X., Fujita, M., and Yang, J. Spectral transforms for large Boolean functions with application to technology mapping. in *Design Automation Conf*. (Dallas TX, June 1993), 54-60.

[4] Drechsler, R. and Becker, B. *Binary Decision Diagrams - Theory and Implementation*. Kluwer Academic Publishers. 1998.

[5] Drechsler, R., Sarabi, A., Theobald, M., Becker, B., and Perkowski, M. A. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. in *Design Automation Conf.* (San Diego CA, June 1994), 415-419.

[6] Falkowski, B. Forward and inverse transforms between Haar wavelet and arithmetic functions. *Electronic Letters*. 34, 10, (May 13, 1998), 1084-1085.

[7] Falkowski, B. Relationship between arithmetic and Haar wavelet transforms in the form of layered Kronecker matrices. *Electronic Letters*. 35, 11, (May 28, 1999), 799-800.

[8] Falkowski, B. and Chang, C.-H. Forward and inverse transformations between Haar spectra and ordered binary decision diagrams of Boolean functions. *IEEE Trans. on Comp*. 46, 11, (November 1997), 1272-1279.

[9] Graham, A. *Kronecker Products and Matrix Calculus: with Applications.* Ellis Horwood Limited and John Wiley & Sons. 1981.

[10] Hansen, J. P. and Sekine, M. Synthesis by spectral translation using Boolean decision diagrams. in *Design Automation Conf.* (Las Vegas NV, June 1996), 248-253.

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

[12] Kebschull, U., Schubert, E., and Rosenstiel, W. Multilevel logic synthesis based on functional decision diagrams. in *European Conf. on Design Automation.* (Brussels Belgium, March 1992), 43-47.

[13] Kunz, W. and Pradhan, D. K. Recursive learning: a new implication technique for efficient solutions of cad problems: test, verification and optimization. *IEEE Trans. on CAD.* 13, 9, (September 1994), 1143-1158.

[14] Miller, D. M. Graph algorithms for the manipulation of Boolean functions and their spectra. in *Congressus Numerantium,* (1987), 177-199.

[15] Perkowski, M. A., Driscoll, M., Liu, J., Smith, D., Brown, J., Yang, L., Shamsapour, A., Helliwell, M., and Falkowski, B. Integration of logic synthesis and high-level synthesis into the diades design automation system. in *Int'l. Symp. Circuits and Systems*. (Portland OR, May 1989), 718-751.

[16] Radecka, K. and Zilic, Z. Relating arithmetic and Walsh spectra for verification by implicit error modeling. in *Int'l. Workshop on Applications of the Reed-Muller Expansion in*

Circuit Design. (Mississippi State MS, August 2001), 205-214.

[17] Somenzi, F. CUDD: CU Decision Diagram Package Release 2.3.0. http://www.vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html.

[18] Stanković, R. S. A note on the relation between Reed-Muller and Walsh transforms. *IEEE Trans. on EMC.* 24, 2, (February 1982), 68-70.

[19] Stanković R. S., Sasao, T., and Moraga, C. Spectral transforms decision diagrams. in Sasao, T. and Fujita, M. (eds.), *Representation of Discrete Functions.* Kluwer Academic Publishers. 1996.

[20] Susskind, A. K. Testing by verifying Walsh coefficients. *IEEE Trans. on Comp.* 32, 2, (February 1983), 198-201.

[21] Thornton, M. A. and Drechsler, R. Spectral decision diagrams using graph transformations. in *Design Automation and Test in Europe.* (Munich Germany, March 2001), 713-717.

[22] Thornton, M. A., Drechsler, R., and Gűnther, W. A method for approximate equivalence checking. in *Int'l Symposium on Multi-Valued Logic.* (Portland OR, May 2000), 447-452.

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

[24] Varma, D. and Trachtenberg, E. A. Design automation tools for efficient implementation of logic functions by decomposition. *IEEE Trans. on CAD.* 8, 8, (August 1989), 901-916.