On the Use of Autocorrelation Coefficients in the Identification of Three-Level Decompositions

J. Rice
Department of Math & Computer Science
University of Lethbridge
Lethbridge, Alberta, Canada
rice@cs.uleth.ca

J. C. Muzio
Department of Computer Science
University of Victoria
Victoria, BC, Canada
jmuzio@cs.uvic.ca

Abstract

Recent advancements have illustrated that three-level representations of Boolean functions have smaller upper-bounds on the number of products than either sum-of-products or AND-XOR representations. In this work we apply the information available in the function's autocorrelation coefficients to the determination of such a representation. The autocorrelation coefficients highlight various aspects of the structure within a function, the presence of XOR logic being one of these. Based on two theorems relating the autocorrelation coefficients to XOR-based decompositions, we have developed a tool for identifying three-level decompositions. Results comparing this tool to existing three-level minimizers are very promising. This work represents the first step in an on-going effort to develop a complete and competitive three-level minimization tool; future work continues in the extension of this tool

1 Introduction

Boolean functions are functions for which the inputs and output(s) are restricted to the Boolean domain. If a function such as the autocorrelation function is applied to the output vector of the function, the result is a representation of the function in a non-Boolean domain. This representation is generally referred to as the *autocorrelation coefficients* of the function.

The autocorrelation coefficients provide a measure of the function's similarity to itself, shifted by a given amount. This is also called the cross-correlation, or convolution function. The autocorrelation coefficients have been used in various areas including optimisation and synthesis of combinational logic [9], variable ordering for ROBDDs [7], and to compute the estimate $C(f)$ of a function's complexity [9, 4]. However, their use has been limited, likely due to the fact that until recently, methods for computing the autocorrelation coefficients were exponential in the number of inputs to the function(s). New methods for their computation have recently been introduced by Rice *et. al.* [6, 5]

Another area that is gaining more attention is that of three-level representations for switching functions. Traditionally, logic synthesis techniques have attempted to generate a minimal sum-of-products or similar representation for a function. However, Dubrova *et. al.* have demonstrated in [3] that there exists an AND-OR-XOR representation for any Boolean function with upper bound on the number of products smaller than that for either a sum-of-products or an AND-XOR expansion. Additionally, with the advent of field-programmable gate arrays (FPGAs), the cost of an XOR gate is no more than that of a traditional AND or OR gate. The problem is to identify where the XOR is best introduced in order to minimize the function.

In this paper we introduce a new technique for identification of XOR logic within a function. This technique is based on information provided by the function's autocorrelation coefficients. We further make use of this in the early development of a tool for identifying three-level decompositions. This tool has been tested on a variety of small benchmarks, with very promising results as compared to existing three-level minimization tools.

2 Background

We first present some notation and background on areas discussed in this paper.

2.1 Autocorrelation Coefficients

Switching functions can be translated to other domains, such as the *spectral domain*. In this paper we consider the calculation of the function's *autocorrelation coefficients*, which are one possible representation in the spectral domain.

The general cross-correlation (convolution) function between two functions $f(X)$ and $g(X)$ at a distance τ is defined as

$$B^{f\,g}(\tau) = \sum_{v=0}^{2^n-1} f(v) \cdot g(v \oplus \tau) \tag{1}$$

When $f(X) = g(X)$, the resulting equation gives the cross-correlation of a function with itself, translated by τ. The resulting coefficients are referred to as the autocorrelation coefficients of the function. Based on this, the autocorrelation function is defined as follows [4]:

$$B^{f\,f}(\tau) = \sum_{v=0}^{2^n-1} f(v) \cdot f(v \oplus \tau) \tag{2}$$

If the superscripts $f\ f$ are omitted (*e.g.* as in $B(\tau)$) then it is assumed that an autocorrelation function is being computed, rather than a cross-correlation function.

For multiple-output functions a second step must be performed to combine the autocorrelation function for each of the individual functions into the *total autocorrelation function*; however, multiple-output functions are not yet considered in this work.

The above definition for $B(\tau)$ assumes that the outputs of the switching function f are encoded as $\{0,1\}$. If the function is alternately encoded as $\{+1,-1\}$ then the definition of the autocorrelation coefficients is the same, with the resulting coefficients being referred to as $C(\tau)$.

2.2 Three-Level Decomposition

Sum-of-products and product-of-sums representations are widely used in logic synthesis, and have been well-researched. However, it has been shown that the addition of one or more XOR gates may result in a smaller representation. In this paper we are interested in AND-OR-XOR representations, which are expressions of the type

$$f(X) = (P_1 \vee P_2 \vee ... \vee P_p) \oplus (P_{p+1} \vee P_{p+2} \vee ... \vee P_m)$$

where $p \in 1 \leq p \leq m$ and P_i are product terms. Existing tools for determining AND-OR-XOR representations includes AOXMIN-MV [2] and work by Sasao *et. al.* [1].

2.3 General Notation

Some additional notation is also required:

- The variable ordering $x_n, ..., x_1$ is used throughout. Thus a coefficient $B(001)$ or $C(001)$ is the first order coefficient corresponding to x_1.
- τ_i refers to a value whose binary expansion contains a 1 in the i^{th} bit, while the remaining $n-1$ bits are 0.
- $\tau_{i\alpha}$ refers to a set of values for which the binary expansion contains a 1 in the i^{th} bit while the remaining $n-1$ bits have the value $\alpha \in \{0, ..., 2^{n-1}-1\}$. $\tau_{\bar{i}\alpha}$ refers to a set of values for which the binary expansion contains a 0 in the i^{th} bit while the remaining $n-1$ bits have the value α.
- $|\tau|$ is the weight, or the number of ones in the binary expansion of τ. If $|\tau| = j$ then $B(\tau)$ and $C(\tau)$ are said to be j^{th} order coefficients.

3 Identifying XOR logic

The technique used in this paper is based on the autocorrelation coefficients for the switching function. The following two theorems are introduced and proven in [5]:

Theorem 3.1 *$C(\tau_i) = -2^n$ if and only if the function $f(X)$ has a decomposition*

$$f(X) = g(X) \oplus x_i$$

where $g(X)$ is independent of x_i and $i \in \{1, ..., n\}$.

Theorem 3.2 *$C(\tau_i) = C(\tau_j) = C(\tau_{ij}) = 0,\ i \neq j$ if and only if the function $f(X)$ can be decomposed into $g(X) \oplus h(X)$ where $h(X) = x_i * x_j$, $* \in \{\wedge, \vee\}$ and $g(X)$ is independent of both x_i and x_j, $i, j \in \{1, ..., n\}$.*

These two theorems are applied to determine a three-level decomposition for the switching function in question.

4 Implementation

In testing this technique a "proof-of-concept" tool was implemented. This tool attempts only to identify a three-level decomposition, and does not perform any further minimization or optimization. Furthermore, the tool uses only the above two theorems and so is limited to identifying decompositions having at most two variables in the second decomposition. The algorithm is as follows:

```
readcoeffs(ac_infile, coeffvector, uvector);
// identify xor logic
id_xor(coeffvector, uvector,
       single_vars, double_vars);
// generate g1 and g2
generate_g1g2(dd, single_vars,
              double_vars, out1, out2);
```

The majority of the work is done in functions id_xor and generate_g1g2. id_xor does the following:

```
for each variable i:
  find coeff(2^i)
  if the coefficient == -2^n
    set a flag for variable i

for each variable i:
  if the variable i is flagged then skip it
  for each variable j beginning at i+1:
    if the variable j is flagged then skip it
    int uval1 = twoexp(numvars-1-j);
    int uval2 = twoexp(numvars-1-i);
    int uval3 = uval1 bitwise-or uval2;
    find the coefficients for these values
    if all are 0
      set a flag for variables i and j
```

generate_g1g2 creates two decision diagrams, one for the first decomposition (g1) and one for the second decomposition (g2). The CUDD tool [8] was used for the creation of the decision diagrams. If no XOR logic was identified by id_xor then this function informs the user.

```
// first check if there is ANY usable xor logic:
bool xorflag = false;
check singlevars array
check doublevars array
if (no xor logic)
  output "No xor logic to use\n\n";
  g1 = original dd;  g2 = NULL;
  outputfunctions(g1, g2);
else
if singlevars contains xor logic
   create g1 = xi
   create g2 = dd xor g1
else, using doublevars array,
   create g1 = xi or xj
   create g2 = dd xor g1
   check g1 xor g2 = dd
   if not
     create g1 = xi and xj
     create g2 = dd xor g1
outputfunctions(g1, g2);
```

5 Results

The pseudocode above was implemented and tested against the three-level minimization tool AOXMIN-MV. Both tools were used to determine three-level decompositions for a series of benchmarks. The first set of tests used only benchmarks of 10 or fewer inputs due to initial limitations of the AC-based tool. Further work has extended the tool to work with any number of inputs. The benchmarks in both cases were limited to single-output functions.

In the first set of tests the benchmarks were generated by using existing, multiple-output benchmarks of 10 or fewer inputs that were split into individual files, one for each output. In these tests both agree on the detection of XOR logic 74% of the time. However, the AOXMIN-MV tool only completed successfully for 244 of the single-output benchmark files, while our tool completed for all 278 of the benchmarks. Additionally, our tool required an average of approximately 5 seconds to compute the decomposition, while AOXMIN-MV, for its successful benchmarks, required an average of over one minute. It should be noted, however, that AOXMIN-MV requires more time for computing a decomposition because the tool is also attempting to find a balanced decomposition, which our tool does not take into account. The computation of all 2^n autocorrelation coefficients is included in this timing figure, although future improvements could reduce this as only the first and second order coefficients are required. These results are summarized in Table 1[1]

	successes	avg. time	num. benchmarks in which XOR logic detected
AOXMIN-MV	244 / 278	71.1 sec	54
3LEVEL	278 / 278	5.4 sec	59

Table 1. Results of comparing the autocorrelation-based three-level level decomposition tool (3LEVEL) to AOXMIN-MV.

For the second set of tests, a small number of varying size single-output benchmarks were tested, with the results as shown in Tables 2 and 3. The input files were not minimized beforehand.

The purpose of this second set of tests is to give some detail on how the AC-based tool is performing and test its performance on larger files. More comprehensive tests are also underway.

Overall, the results are in agreement with the existing AOXMIN tool. Clearly the AOXMIN tool is finding more balanced decompositions; however, the AC-based tool is performing, on average, faster, and with fewer errors.

[1]Complete results could not be included as over 250 benchmarks were tested.

filename	**g1 prod**	**g2 prod**	**xor?**	**time**
9symml			no xor	0.7
cm150a			no xor	346.76
cm152a	2	20		0.5
co14			no xor	0.3
ex1	1	8		0.2
ex2			no xor	0.3
ex3			no xor	0.2
life			no xor	0.3
majority			no xor	0.1
max46			no xor	0.2
mux			no xor	364.8
o64				error
parity				error
ryy6			no xor	0.6
sym10			no xor	1.7
t481			no xor	2.1
xor5	1	8		0.2

Table 2. Results of testing the AC-based 3-level decomposition tool on a small number of single-output functions.

filename	**g1 prod**	**g2 prod**	**xor?**	**time**
9symml				error
cm150a			no xor	0.1
cm152a			no xor	0.2
co14			no xor	0.3
ex1	2	4		43.3
ex2	1	6		0.2
ex3			no xor	0.2
life	20	40		370.4
majority			no xor	0.1
max46			no xor	0.4
mux			no xor	0.1
o64				error
parity				error
ryy6				error
sym10				error
t481	20	9		362.0
xor5	2	4		42.0

Table 3. Results of testing AOXMIN on a small number of single-output functions.

6 Discussion

In the first set of tests a little more detail can be provided. It is interesting to note that while our autocorrelation-based tool detected XOR logic in 59 of the benchmark files, in 32 of those the AOXMIN-MV tool did not detect XOR logic. Similarly, of the 54 benchmarks for which AOXMIN-MV detected XOR logic, our tool did *not* detect XOR logic in 27 of those. Thus for only 27 of the benchmarks did BOTH tools detect XOR logic. The immediate question is why. For the first situation, when our tool detects XOR logic while AOXMIN-MV does not, the answer is that AOXMIN-MV is attempting to find a solution for which the products are fewer than in the best two-level minimization solution. If this is not found then AOXMIN-MV does not provide a decomposition. Our tool does not take this into consideration, it simply provides the decomposition based on the theorems in Section 3. For the second situation, when our tool does not detect XOR logic while AOXMIN-MV does, the answer is that AOXMIN-MV takes into account more possibilities for three-level decompositions than does our tool; our tool is currently limited to only the two situations described by Theorems 3.1 and 3.2. Future work must be done to extend the tool to identify other types of decompositions.

7 Future Work

There are many areas in which this work may be extended, some of which are detailed below.

The first item to address is that of identification of balanced decompositions, so that a more even comparison with the work done by AOXMIN may be performed. This entails extension of the theorems relating the autocorrelation coefficients to the presence of exclusive-or logic to more general cases, which is currently underway.

Another modification possibly resulting in a more balanced decomposition would be to re-apply the technique to the $g(X)$ part of the decomposition, comparing the resulting number of products with the original to determine if there are any savings. However, with such a process the resulting decomposition would not strictly match the format of an AND-OR-XOR representation, as there is the possibility that more than one XOR gate may be required.

Finally, this work applies only to single-output functions; it is important to extend it to multiple-output functions for it to be a truly useful tool.

For all of the above situations, additional heuristics to examine the identified decompositions and identify

Table 4. Further details on the benchmarks used in the second set of tests.

filename	inputs	products
9symml	9	86
cm150a	21	17
cm152a	11	8
co14	14	47
ex1	5	16
ex2	5	7
ex3	5	4
life	9	512
majority	5	5
max46	9	46
mux	21	36
o64	130	65
parity	16	32768
ryy6	16	112
sym10	10	837
t481	16	481
xor5	5	16

the “best” possibility, either for a single output or for multiple outputs, would be of use, and can certainly be developed even with the current limitations of the two known theorems.

8 Conclusion

This work discusses a new application of the autocorrelation coefficients. Based on two theorems relating patterns in the autocorrelation coefficients to the presence of XOR logic in a switching function, a three-level decomposition of the type AND-OR-XOR is determined. The initial implementation is limited to the identification of only two “flavours” of AND-OR-XOR decomposition. However, even with this limitation, it compares quite favourably with existing tools. Work is currently underway to extend the theorems to a more general case and in the development of heuristics to choose the best decomposition based on the information provided by the autocorrelation coefficients. With such extensions we feel that this tool has great potential in this and many other areas.

References

[1] D. Debnath and T. Sasao. A Heuristic Algorithm to Design AND-OR-EXOR Three-Level Networks. In *Proceedings of ASP-DAC*, pages 69 –74, 1998.

[2] E. Dubrova. AOXMIN-MV: A Heuristic Algorithm for AND-OR-XOR Minimization. In *Proceedings of the 4th International Workshop on Applications of Reed-Muller Expansions in Circuit Design (RM99)*, pages 37–53, 1999.

[3] E. V. Dubrova, D. M. Miller, and J. C. Muzio. Upper Bound on Number of Products in AND-OR-XOR Expansion of Logic Functions. *IEE Electron. Lett.*, 31:541–542, 1995.

[4] M. Karpovsky. *Finite Orthogonal Series in the Design of Digital Devices.* John Wiley & Sons, 1976.

[5] J. Rice. *Autocorrelation Coefficients in the Representation and Classification of Switching Functions.* PhD thesis, University of Victoria, 2003.

[6] J. E. Rice and J. C. Muzio. Methods for Calculating Autocorrelation Coefficients. In *Proceedings of the 4th International Workshop on Boolean Problems, (IWSBP2000)*, pages 69–76, 2000.

[7] J. E. Rice, J. C. Muzio, and M. Serra. The Use of Autocorrelation Coefficients for Variable Ordering for ROBDDs. In *Proceedings of the 4th International Workshop on Applications of the Reed-Müller Expansion in Circuit Design*, 1999.

[8] F. Somenzi. CUDD: Colorado University Decision Diagram Package, version 2.3.0. Department of Electrical and Computer Engineering, University of Colorado at Boulder, *Fabio@Colorado.EDU.*

[9] R. Tomczuk. *Autocorrelation and Decomposition Methods in Combinational Logic Design.* PhD thesis, University of Victoria, 1996.