Mathematical Framework for Representing Discrete Functions as Word-level Polynomials

Dhiraj K. Pradhan*, Serkan Askar+, Maciej Ciesielski+

* University of Bristol, Department of Computer Science
Bristol BS8 1UB, UK
pradhan@cs.bris.ac.uk

+University of Massachusetts, Dept. of Electrical & Computer Engineering
Amherst, MA 01003-4410, USA
{saskar, ciesiel}@ecs.umass.edu

Abstract—
This paper presents a mathematical framework for modeling arithmetic operators and other RTL design modules as discrete word-level functions and proposes a polynomial representation of those functions. The proposed representation attempts to bridge the gap between bit-level BDD representations and word-level representations, such as *BMDs and TEDs.

I. Introduction

The increase in size and functional complexity of digital designs necessitates the development of robust, automated verification tools at higher (behavioral or register-transfer) levels of abstraction. Binary Decision Diagrams (BDDs) [1] [2], Binary Moment Diagrams (*BMDs), [3], and their derivatives [4] [5], [6] [7], [8], [9], [10], [11], [12], etc. play an important role as canonical representations of Boolean and arithmetic functions and have significantly contributed to the success of verification tools. However, a major limitation of these representations is that they require that the inputs, outputs, or both, be represented in terms as Boolean functions, in terms of bits. As a result, these representations are often characterized by prohibitively large size or require excessive computation times. Specifically, BDDs represent logic functions that map Boolean inputs into Boolean outputs and employ binary Shannon decomposition with respect to the function variables. Binary Moment Diagrams (*BMD [3] and K*BMDs [5]), represent mapping of Boolean inputs into integer-valued functions. They depart from point-wise, binary decomposition, and perform a decomposition of an arithmetic function based on its moments (constant and first moment).

A recently developed *Taylor Expansion Diagrams* (TED) [13] [14] represent arithmetic functions that map *integer* inputs into *integer* functions, thus raising the level of abstraction from bits to bit-vectors and words. TEDs are based on a more general, non-binary decomposition of an arithmetic function using Taylor Series Expansion w.r.t its support variables. This representation naturally applies to all functions that can be modeled as *polynomials*. Furthermore, it is not limited to algebraic expressions or designs characterized by continuous-function representation, such as data-flows or datapath designs. It can be used whenever a function (continuous or discrete) can be represented or approximated by means of a *finite polynomial*.

Polynomial methods [15], [16] have been used as an efficient technique for representing both arithmetic specifications and bit-level descriptions of implementations. In [15], polynomial representation has been used to perform component matching by expressing a specification (given for example as a transfer function, or rational polynomial, in MATLAB) and an implementation, as word-level polynomials. The problem of generating a word level polynomial reduces to determining the order of the polynomial. Specifically, [15] shows that any combinational circuit can be uniquely represented by a minimum order polynomial and describes the method for finding the order. Since the resulting polynomial representation is canonical for a fixed word width, two designs can be compared by comparing coefficients of their polynomial representations.

Polynomial methods and symbolic algebra have also been used to perform arithmetic-level decomposition. In this case, an arithmetic function represented by a multi-variate polynomial is decomposed into polynomials representing the building blocks in the target library [17]. In effect, it performs *simplification modulo set of polynomials*, using Groebner basis. In summary, it can be argued that polynomials can serve as a convenient means to represent arithmetic functions for the purpose of high level synthesis and verification.

A. Problem Statement

The goal of this paper is to show how to derive *polynomial representation* of arithmetic functions typically encountered in RTL designs. The challenge is to derive a compact polynomial representation for an *arbitrary* function, including continuous arithmetic functions as well as as discrete logic. This work addresses specifically the following problems: 1) how to represent an *arbitrary* function as a polynomial, and 2) how to efficiently generate and store the polynomial representation.

When deriving a word-level representation one should keep in mind its application to realistic RTL designs. While

BMDs and TEDs have already raised a level of abstraction of design representations by operating on word-level, these representations are not practical in cases when *partial extraction* of word-level signals are needed. For example, the fact that a multiplier can be represented by *BMD or TED as an arithmetic operator with a *single* word-level output is not particularly useful in a design where partial expansion of the output vector is required. Such an expansion (informally referred to as *bit nibbling*) is often necessary to derive a few bits (a bit subvector) from the output vector as input to the other part of the design. The same argument applies to the adder/subtract units where the carry out (or sign) bit should be treated separately from the sum output.

To further motivate our work in the context of realistic RTL designs consider the following design shown in Fig. 1. It has been demonstrated in [13] that TED can be used to represent the output of such a design in the case when bit nibbling occurs at the primary inputs only. In this case the inputs are broken into smaller, word-level variables, and the word-level, integer-valued outputs of the subsequent functions can be readily represented as continuous arithmetic function (polynomials) in terms of those inputs. As long as all the intermediate functions are integer-valued, and need not to be broken into subvectors, there is no problem. However, often a group of bits of a local function block (intermediate word-level signal) needs to fan out to another portion of the design and hence must be broken into several subvectors. In this case we are faced with the problem of representing a *multiple output* function. Such a function, in general, is not continuous but *discrete*, and as such cannot be easily represented analytically. This discrete nature of the arithmetic operators is the major problem with all the known word-level representations, as they do not provide *enough granularity* at the function *output*.

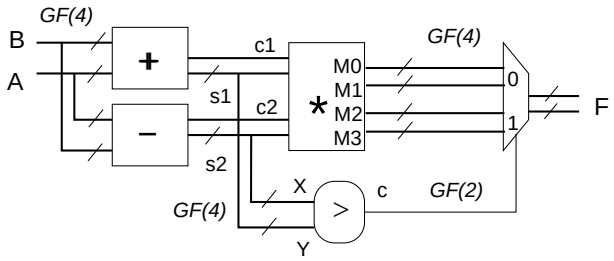
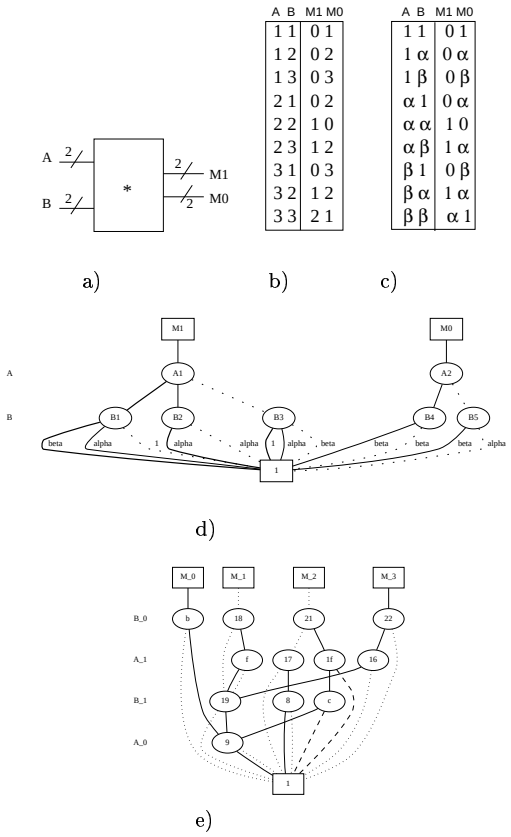


Fig. 1. An example of an RTL design with partial (bit-level) fanout

The representation proposed in this paper attempts to fix this problem, by modeling a discrete multiple-output function as finite, word-level *polynomials*.

II. Characteristic Polynomials

The idea of representing an arbitrary discrete function as a multivariate polynomial is illustrated in Fig. 2 with the help of a simple 2×2 multiplier. The multiplier has two 2-bit inputs A, B and a 4-bit output M. In principle, the output M can be represented as a continuous function of its inputs, that is $M = A \cdot B$, where, $0 \leq A, B \leq 3$. However, if access to a subset of bits of the output is needed, it is difficult to express that portion as a continuous function. In our case the output M is divided into outputs M_1 and M_0, so that $M = 4M_1 + M_0$. Each of these outputs can then be treated as a four-valued discrete function of two four-valued inputs A, B. The figure shows a "truth table" defining these discrete functions representing the 2×2 multiplier. All variables in the table are *modulo 4*.



A	B	M1	M0
1	1	0	1
1	2	0	2
1	3	0	3
2	1	0	2
2	2	1	0
2	3	1	2
3	1	0	3
3	2	1	2
3	3	2	1

A	B	M1	M0
1	1	0	1
1	α	0	α
1	β	0	β
α	1	0	α
α	α	1	0
α	β	1	α
β	1	0	β
β	α	1	α
β	β	α	1

Fig. 2. A 2×2 multiplier as a discrete function: $\{M_1 M_0\} = A \cdot B$; a) block diagram; b) truth table in $radix$-4 number system; c) truth table in $GF(4)$ system; d) characteristic polynomial representation (TED); e) equivalent bit-level BDD representation

We shall now attempt to interpolate each of the discrete outputs as a continuous, *finite polynomial*. First we define a *characteristic polynomial* of a function.

Definition 1: A polynomial correctly interpolates the discrete function, if *the value of the polynomial evaluated at all the discrete points for which the function is defined matches exactly the value of the function.* We refer to such a polynomial as a *characteristic polynomial* of the function.

The challenge is to derive and efficiently store a characteristic polynomial for a given discrete function. Our approach to generate a continuous polynomial representation is based on Galois representation of switching functions, proposed in [18].

Assume for simplicity that the function is given as a truth table, such as the one shown in Fig. 2(b). The entries of the table are expressed in $radix$-n number system, where $N = 2^k$, and k is the number of bits representing the number. Alternatively, the entries can be interpreted

as the elements of *Galois Field* of size n, $GF(N)$. That is, the numbers (0, 1, 2, ..., $n-1$) are represented as $(0, 1, \alpha, \alpha^2, ..., \alpha^{N-2})$, where α is the primitive element in the field $GF(N)$. In case of 2-bit variables, the elements $(0, 1, \alpha, \alpha^2)$ are in $GF(4)$ number system. It is customary in this case to use the notation $\alpha^2 = \beta$.

Let us now derive the polynomial equation for M_1 and M_0 using the truth table shown in Fig. 2(c), obtained by replacing the radix-4 numbers with elements of $GF(4)$. The derivation of the polynomial representation relies on GF number system rather than on radix (modular) system. The desired polynomial expression is obtained by summing up the product terms of multi-valued literals corresponding to the nonzero outputs. The k^{th} literal of variable X, or X^k, is defined as 1 if $X = k$, and 0 otherwise. In Galois field $GF(N)$ a literal X^k is represented as

$$X^k = (1 - (X - k)^{N-1}), \tag{1}$$

where $k \in GF(N)$. A product of all literals in a given row of the table gives a multi-valued product term. Each product term evaluates to 1 for one and exactly one combination of input values. In this sense, this is an extension of classical Boolean logic, whose variables are elements of $GF(2)$. The desired polynomial representation for a given function can then be obtained by summing up all the product terms corresponding to non-zero outputs, multiplied by the respective coefficients of the given output.

For output M_1 in Fig. 2(c) we have:

$$M_1 = A^\alpha B^\alpha + A^\alpha B^\beta + A^\beta B^\alpha + \alpha A^\beta B^\beta \tag{2}$$

or equivalently

$$\begin{aligned} M_1 = \quad & (1-(A-\alpha)^3)(1-(B-\alpha)^3) + \\ & (1-(A-\alpha)^3)(1-(B-\beta)^3) + \\ & (1-(A-\beta)^3)(1-(B-\alpha)^3) + \\ & \alpha(1-(A-\beta)^3)(1-(B-\beta)^3) \end{aligned} \tag{3}$$

A. Representing Characteristic Polynomials

The resulting polynomial, after simplification of the constant terms (using rules of $GF(4)$) can be obtained as:

$$M1 = A^3(\beta B^3 + aB^2 + B) + A^2(aB^3 + aB) + A(B^3 + aB^2 + \beta B) \tag{4}$$

There are several methods that can be used to derive characteristic polynomials of discrete functions. The *evaluation form* representation stores discrete *values* of the polynomial for all combinations of its inputs. This representation is particularly useful when function is given as a truth table. Also, the time complexity of basic operations (addition and multiplication) for two functions represented in this form is linear.

Another canonical representation, called the *interpolation form*, represents the polynomial using the coefficients corresponding to all the combinations of the powers of the input variables [19]. In general, this form is more computationally expensive than the evaluation form. However, we use this form as it is compatible with the diagram representation employed in our system.

In our 2×2 multiplier example, this would give:

$$M1 = [\beta\alpha 10\ \alpha 0\alpha 0\ 1\alpha\beta 0\ 0000]$$

which is equivalent to equation 4.

In general, the number of coefficients that need to be stored to identify a polynomial in $GF(N)$, with m input variables, each taking $N = 2^k$ values, is 2^{mk}. An ordered array of such coefficients uniquely represents the function.

Another alternative is to use Taylor Expansion Diagram (TED) [13] to generate and store the polynomial in form of a graph. TED is in fact a graphical variant of the interpolation form. The weights of the edges of TED represent the non-zero coefficients of the characteristic polynomial of the function. An example of TED for a 4×4 multiplier is shown in Fig. 2(d).

An important advantage of using characteristic polynomial to represent discrete function operators is that outputs of such operators can be made independent of the intermediate signals and variables, i.e., they can be expressed in terms of primary (or arbitrary set of) inputs. We say that those intermediate variables can be *smoothed* out, in a manner similar to that in BDDs.

The following theorem states the canonicity of the characteristic polynomial for an arbitrary discrete function.

Theorem 1: The Characteristic Polynomial representation of a discrete function is *minimal* and *unique* for a fixed ordering of primary input variables.

Proof. The proof of canonicity follows directly from the fact that the coefficient of the polynomial are unique. See also the proof of the uniqueness of TED [13]. To prove minimality, we must consider a suitable word-level representation and show that for that representation the form is minimal. Consider a TED, or equivalently, an interpolation form representation. At a given decomposition level, associated with one variable, all the children of the variable are grouped according to the degree of the edge (degree of derivative). Any simplification that takes place at the edges involves only manipulation of constants (in the given field) and can be done in constant time. □

This result is basically a generalization of the well known theorem that BDD of a Boolean function is unique for a fixed variable ordering. This is not surprising since $GF(2)$ is a special case of $GF(N)$, for $N = 2^k$.

Fig. 3 shows the characteristic polynomial representation for a comparator in $GF(4)$: (a) its TED diagram representation, and (b) the equivalent bit-wise BDD representation.

Characteristic polynomial representation for the *Sum* and C_{out} of a full adder can be obtained in similar fashion. Larger, $2n$-bit adders can be constructed from a set of full adders in a ripple carry (RC) fashion. In fact, this can be done for any function that exhibits this serial, iterative dependence. Unfortunately, this is not the case for a multiplier, which has hierarchical dependence.

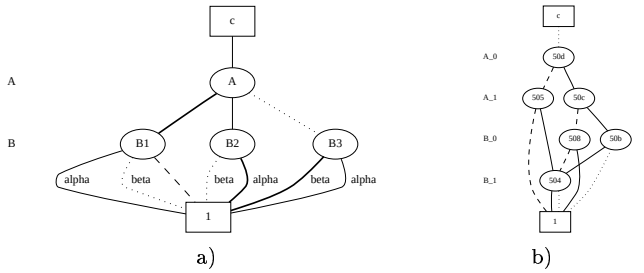


Fig. 3. A 2-bit comparator; a) characteristic polynomial in GF(4); b) bit-level BDD representation

B. Normalization and minimization of CP

Consider the result of a 4×4 multiplier in $GF(4)$, where the 4-bit inputs and the 8-bit output are divided into 2-bit vectors (4-valued variables).

Assume ordering of input variables: $A_1, A_0,\ B_1, B_0$. The coefficient matrix shown in Figure 4 represents output M_1 of the multiplier (here `a` stands for α and `b` for β). Careful examination of the matrix in Figure 4 reveals several *common patterns*, which – after factoring out a constant – correspond to the *same* expression in $GF(4)$. Four of these occurrences, shown in the figure as groups $R1 \ldots R4$, share a common pattern $L = [0000\ \ 01\alpha 0\ \ 0\alpha\beta 0\ \ \alpha 1\beta 0]$ which can be readily identified after factoring out a corresponding constant in $GF(4)$. Namely, we have $R1 = \beta L$, $R2 = R3 = L$, and $R4 = \alpha L$.

```
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000
0000 0000 0000 0000

0000 0000 0000 ba10
0000 0b10 01a0 1ba0   R1
0000 01a0 0ab0 a1b0   R2
0000 0000 0000 0bb0

0000 0000 0000 1ba0
0000 01a0 0ab0 a1b0   R3
0000 0ab0 0b10 ba10   R4
0000 0000 0000 0ba0

0000 b1a0 1ab0 ba10
0000 ab1b b1ab a0a0
0000 1abb ab1a 1ab0
0000 0000 0000 0000
```

Fig. 4. Characteristic polynomial for M1 output of the 4×4 multiplier in $GF(4)$

Specifically, the characteristic polynomial (CP) of groups $R1$, $R2$, $R3$, $R4$ in the matrix corresponds to the following expressions:

$$CP(R1) = A_1^2 A_0^2 \cdot \beta L \quad (5)$$
$$CP(R2) = A_1^2 A_0 \cdot L \quad (6)$$
$$CP(R3) = A_1 A_0^2 \cdot L \quad (7)$$
$$CP(R4) = A_1 A_0 \cdot \alpha L \quad (8)$$

where

$$L = B_1^2(B_0^2 + \alpha B_0) + \alpha B_1(B_0^2 + \alpha B_0) + \alpha(B_0^3 + \beta B_0^2 + \alpha B_0)) \quad (9)$$

which in the matrix form is: $L = [0000\ \ 01\alpha 0\ \ 0\alpha\beta 0\ \ \alpha 1\beta 0]$.

The process of identifying and normalizing such common patterns (subfunctions) can be readily accomplished using canonical graph based representations, such as TEDs [13]. This can be seen in the diagram for M_1, shown in Fig. 5 as a common node $B1\_23$.

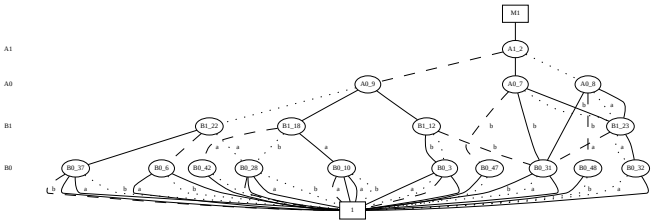


Fig. 5. TED representation of output M1 of the 4×4 multiplier in $GF(4)$

The TED for the entire 4×4 multiplier in $GF(4)$ is shown in Fig. 6. It contains 86 nodes, compared to 135 nodes of the minimized BDD.

C. RTL Design Representation

Equipped with the characteristic polynomials of all the design blocks (adders/subtractors, multipliers, comparators, multiplexers, etc.) we are now ready to complete the design shown in Fig. 1. The generation of the characteristic polynomial $CP(F)$ for output F is performed by generating $CP's$ of all the blocks in topological order, starting at the primary inputs. The generation of $CP's$ for the outputs of the adder and subtractor is simple, since all its input variables are primary inputs. This process has been described in Section II. However, for all the subsequent blocks, the occurrence of all symbolic variables need to be replaced (substituted) with their characteristic polynomials. The substitution of variables by their respective polynomials accomplishes variable *smoothing* (known as *existential quantification* in BDDs), making the primary output depend only on the primary input. This variable smoothing can be readily accomplished using the interpolation form, mentioned in Section II-A.

For example, in the characteristic polynomial of the comparator ($c = X > Y$) in $GF(4)$:

$$CP(c) = X^3(\alpha Y^2 + \beta Y + 1) + X^2(\alpha Y^3 + \beta Y) + X(\beta Y^3 + \alpha Y^2)$$

we must replace all the occurrences of X and Y by their characteristic polynomials, based on how the X, Y signals were derived from other blocks. In our case (refer to Fig. 1),

$$CP(Y = s_1) = A^2(B^2 + \alpha B) + A(\alpha B^2 + \beta B + 1) + B$$

and

$$CP(X = s_2) = A^2(B^2 + \alpha B) + A(\alpha B^2 + \beta B + 1) + \beta B^2$$

Substituting these polynomials into $CP(c)$ of the comparator will give the following characteristic polynomial for

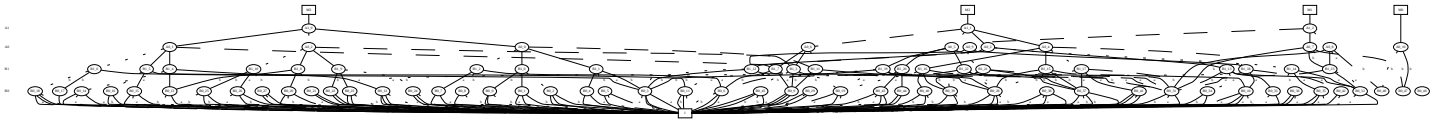


Fig. 6. TED representation of the 4×4 multiplier in $GF(4)$

its output.

$$CP(c) = A^2(B^2 + \alpha B) + A(\beta B^2 + B) + B^3 + B^2 + B \quad (10)$$

Note that the resulting expression relates the output of the comparator to the *primary inputs* A, B, with the intermediate signals (s_1, s_2) smoothed out.

III. Conclusions

The work presented in this paper is by no means complete. We were intrigued by the idea of representing discrete functions with their characteristic polynomials in $GF(N)$, where N is determined by the size of the subvector of the intermediate signals. We did some preliminary comparisons of the proposed representations with BDDs for selected functions. It seems that the proposed GF polynomial representation is smaller in size of the representation (the number of nodes) than BDD by only a (small) constant. We suspect that the reason for such a small gain is that we attempt to represent inherently *integer* functions with the GF base, and such a representation cannot offer much compression. More studies are needed to see if this representation can lead to useful applications.

IV. Acknowledgment

The research reported in this paper has been supported in part by the National Science Foundation (NSF), contract CCR-0204146, and by Engineering and Physical Sciences Research Council (EPSRC), UK, grant No. EPSRC GR/S40855/01.

References

[1] R. E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, vol. C-35, pp. 677–691, August 1986.

[2] K. S. Brace, R. Rudell, and R. E. Bryant, "Efficient Implementation of the BDD Package," in *DAC*, 1990, pp. 40–45.

[3] R. E. Bryant and Y-A. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams," in *DAC*, 95.

[4] A. Narayan and *et al.*, "Partitioned ROBDDs: A Compact Canonical and Efficient Representation for Boolean Functions," in *Proc. ICCAD*, '96.

[5] R. Dreschler, B. Becker, and S. Ruppertz, "The K*BMD: A Verification Data Structure," *IEEE Design & Test*, pp. 51–59, 1997.

[6] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping," in *DAC*, 93, pp. 54–60.

[7] I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic Decision Diagrams and their Applications," in *ICCAD*, Nov. 93, pp. 188–191.

[8] Y-T. Lai, M. Pedram, and S. B. Vrudhula, "FGILP: An ILP Solver based on Function Graphs," in *ICCAD*, 93, pp. 685–689.

[9] U. Kebschull, E. Schubert, and W. Rosentiel, "Multilevel Logic Synthesis based on Functional Decision Diagrams," in *EDAC*, 92, pp. 43–47.

[10] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski, "Efficient Representation and Manipulation of Switching Functions based on Order Kronecker Function Decision Diagrams," in *DAC*, 1994, pp. 415–419.

[11] E. M. Clarke, M. Fujita, and X. Zhao, "Hybrid Decision Diagrams - Overcoming the Limitation of MTBDDs and BMDs," in *ICCAD*, 95.

[12] S. Horeth and Drechsler, "Formal Verification of Word-Level Specifications," in *DATE*, 1999, pp. 52–58.

[13] M. Ciesielski, P. Kalla, Z. Zheng, and B. Rouzyere, "Taylor Expansion Diagrams: A Compact Canonical Representation with Applications to Symbolic Verification," in *Proc. Design Automation and Test in Europe, DATE-02*, Mar 2002, pp. 285–289.

[14] P. Kalla, M. Ciesielski, E. Boutillon, and E. Martin, "H—igh-level Design Verification using Taylor Expansion Diagrams: First Results," in *IEEE Intl. High Level Design Validation and Test Workshop, HLDVT-02*, 2002, pp. 13–17.

[15] J. Smith and G. DeMicheli, "Polynomial Methods for Compontent Matching and Verification," in *International Conference on Computer-Aided Design, ICCAD'98*, 1998.

[16] J. Smith and G. DeMicheli, "Polynomial Methods for Allocating Complex Compontents," in *Design Automation and Test In Europe Conference, DATE'99*, 1999.

[17] A. Peymandoust and G. DeMicheli, "Using Symbolic Algebra in Algorithmic Level DSP Synthesis," in *Design Automation Conference, DAC2001*, 2001.

[18] D.K. Pradhan, "A Theory of Galois Switching Functions," *IEEE Transactions on Computers*, vol. C-27, no. 3, pp. 239–248, March 1978.

[19] A.. Aho, J. Hopcroft, and J. Ullman, *The Design And Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1976.