Intelligent Information Management, **2010, 2, 95-103**
doi:10.4236/iim.2010.22012 Published Online February 2010 (http://www.scirp.org/journal/iim)

Probabilistic Verification over GF(2^m) Using Mod2-OBDDs

José Luis Imaña
Department of Computer Architecture, Faculty of Physics, Complutense University, Madrid, Spain
Email: *jluimana@dacya.ucm.es*

Abstract

Formal verification is fundamental in many phases of digital systems design. The most successful verification procedures employ Ordered Binary Decision Diagrams (OBDDs) as canonical representation for both Boolean circuit specifications and logic designs, but these methods require a large amount of memory and time. Due to these limitations, several models of Decision Diagrams have been studied and other verification techniques have been proposed. In this paper, we have used probabilistic verification with Galois (or finite) field $GF(2^m)$ modifying the CUDD package for the computation of signatures in classical OBDDs, and for the construction of Mod2-OBDDs (also known as ⊕-OBDDs). Mod2-OBDDs have been constructed with a two-level layer of ⊕-nodes using a positive Davio expansion (pDE) for a given variable. The sizes of the Mod2-OBDDs obtained with our method are lower than the Mod2-OBDDs sizes obtained with other similar methods.

Keywords: Verification, Probabilistic, OBDD, Mod2-OBDD, Galois Field $GF(2^m)$

1. Introduction

One of the most important aspects during circuit design is the verification, *i.e.*, checking for functional equivalence. The translation of a circuit design from a high-level specification to a physical implementation depends on the correct transformation of its description at higher levels of abstraction to equivalent descriptions at more detailed levels. At logic level, verification consists of checking the equivalence of a Boolean function specification and its logic implementation. Most of the current successful equivalence checkers use Binary Decision Diagrams (BDDs) [1,2] or their derivatives as a core of the equivalence deduction engine. OBDDs [3,4] are used as canonical representations for both Boolean circuit specifications and logic designs. While OBDD-based methods have been quite successful in verifying combinational and sequential circuits, they have significant limitations. For many circuits, verification systems that represent functions as OBDDs require a large amount of memory and time, and for some circuits the resource requirements are unacceptably large. Furthermore, OBDD sizes are quite sensitive to the ordering of their Boolean variables [5,6]. Various techniques have been proposed to reduce the memory complexity of BDDs by exploiting the structural and functional similarities of the circuits [7–9]. Some alternative to BDD-based equivalence checkers use Boolean Satisfiability (SAT) [10,11] or SAT-like methods (ATPG [12], recursive learning [13]) as a principal engine.

In spite of the considerable advances in the area, the growing complexity of the verification instances motivates exploring the alternative approaches. Verification performed using OBDDs can be considered as a Deterministic Verification, in which if the OBDDs of the functions are the same (isomorphic), then these functions are said to be equivalent [14]. Another method that can be considered in order to circumvent the above limitations is the Probabilistic Verification, based on the theory of Blum, Chandra and Wegman [15], in which numeric codes or signatures represent the functions to be verified. This method is based on algebraic transforms of Boolean functions, so that a function can be substituted by its algebraic representation. A signature representing the function is then obtained assigning numeric codes (randomly selected from a finite field) to the variables in the algebraic representation, and then evaluating the result. The comparison is then performed over signatures, not over

This work was partially supported by the Spanish Government Research Grant TIN2008-00508.

data structures representing the functions [16]. Signatures can be computed more efficiently than graph-based representations, consume less time and space, and distinguish any pair of Boolean functions with a very high probability of success. By performing several such runs with different random input variable assignments, the resultant algebraic simulation has a probability of error in verification that decreases exponentially from an initially small value [16]. The probabilistic approach presents significant advantages over deterministic methods using OBDDs. In deterministic verification, OBDDs are canonical representations of functions to be verified, and provide a basis for efficient computation. In probabilistic verification, functions are represented by signatures, not by a large data structure. Graph-based data structures are used only in intermediate evaluation steps. Therefore, more general OBDD-based models have been studied which can be used as such intermediate data structures [17–20].

In this paper, we consider Mod2-OBDDs [21] (also known as ⊕-OBDDs) which are extensions of OBDDs. Mod2-OBDDs are non canonical representations of Boolean functions. For canonical representations as OBDDs, testing the equivalence of two OBDDs simply reduces to the comparison of their pointers. For non canonical representations as Mod2-OBDDs, a deterministic equivalence test requires time cubic in the number of nodes [22], and thus, it seems not to be suitable for practical purposes. In [21], a fast probabilistic equivalence test for Mod2-OBDDs that requires only a linear number of arithmetic operations is used.

In this contribution, a very efficient OBDD package (CUDD package from Colorado University) has been modified in order to construct Mod2-OBDD representations of Boolean functions given in multilevel BLIF, and probabilistic verification based on Galois field $GF(2^m)$ arithmetic for the equivalence test (signatures comparison) has been used. The addition of two elements from a binary extension field $GF(2^m)$ is simply a bitwise XOR of their corresponding binary representations, the subtraction is exactly the same as addition, and the complexity of the multiplication depends on the irreducible generating polynomial or the basis selected to represent the field elements [23–25]. These properties justify the use of $GF(2^m)$ as finite field.

Firstly, OBDDs with signatures have been constructed. The signature-inclusion is carried out by including a 32 bit signature field on each OBDD-node and computing the signature in the synthesis process. This signature-OBDD obtained has the same number of nodes as the original OBDD, but with the signature computed for the Boolean function that it represents. Secondly, Mod2-OBDDs have been constructed by the inclusion of a two-level layer of ⊕-nodes. This layer is created–using the positive Davio expansion (pDE) for a selected variable–in the synthesis of the BLIF file for the function. Signature is so computed for the Mod2-OBDD and is compared with signature obtained with the signature-OBDD for the equivalence test. Times and sizes are compared for the three used structures (OBDDs, signature-OBDDs and Mod2-OBDDs).

The paper is structured as follows: In Section 2, some basic concepts concerning Mod2-OBDDs are introduced. In Section 3, probabilistic equivalence test using Galois field GF(2^m) is presented. In Section 4, modifications on CUDD package for signature computation are outlined. Section 5 deals with the introduction of ⊕-nodes for the Mod2-OBDD construction. In Section 6, experimental results are presented. Finally, some conclusions are included in Section 7.

2. Mod2-OBDDs

Mod2-OBDDs have been defined in [21]. A Mod2-OBDD (also known as ⊕-OBDDs) over a set $X_n=\{x_1,x_2,\ldots,x_n\}$ of Boolean variables is a directed acyclic connected graph P where each node has out-degree 2 or 0. There is a distinguished non-terminal node, the *root*, which has in-degree 0. The two terminal nodes with out-degree 0, the 0-sink and the 1-sink, are labeled with the Boolean constants 0 and 1, respectively. The remaining non-terminal nodes v are either labeled with Boolean variables $x_i \in X_n$ (denoted as branching or decision nodes), or with the binary Boolean operation XOR (⊕-nodes). Let $l(v)$ denote the label of the node v. The two edges starting in a non-sink node are labeled with 0 and 1. The 0-successor and 1-successor nodes of v are denoted by v_0 and v_1, respectively. If v_2 is a successor of v_1 in P and $l(v_1)$, $l(v_2) \in X_n$ then $l(v_1)<l(v_2)$ according to a given ordering on the set of input variables.

The function f_P associated with a Mod2-OBDD P is determined as follows. Given an input assignment $a=(a_1,a_2,\ldots,a_n)\in\{0,1\}^n$, the Boolean values assigned to the leaves extend to Boolean values associated with all nodes of P as follows:

- If the successor nodes v_0, v_1 of a node v of P carry the Boolean values b_0, b_1, respectively, and if $l(v)=x_i$, then v is associated with the value b_0 or b_1 according to $x_i=0$ or $x_i=1$.
- If $l(v)=\oplus$, then the value $\oplus(b_0,b_1)=(b_0+b_1) \bmod 2$ is associated with v.

The value $f_P(a)$ of the Boolean function f_P represented by a Mod2-OBDD P is the value 1 or 0 associated with the root of P under the assignment a. A more compact representation can be obtained by using complemented edges [26].

3. Probabilistic Verification Using Galois Fields

Probabilistic verification consists of the generation of a code or signature for a Boolean function. The probabilistic comparison of two functions can be carried out by evaluating their representations on a randomly chosen vector of values selected from a finite field and comparing the results (signatures) obtained from the evaluation. If the signatures are different, then the functions are inequivalent with certainty. If they are equal, then the functions are equivalent with a small probability of error.

The signature for a Boolean function $f(x_{n-1},\ldots,x_0)$ is generated by selecting random numerical values for each x_i. These values can be selected from any field, but we will use the finite field $GF(p^m)$, with p prime and $m \in N$, representing a Galois field with p^m elements. The evaluation of the function (with these randomly selected values assigned to its inputs) is carried out by the replacement of $f(x_{n-1},\ldots,x_0)$ by an equivalent arithmetic function defined over the finite field. This replacement is determined by an algebraic transformation of the Boolean function in terms of polynomials over the finite field. If we assign the polynomial $p_x=x$ to a Boolean variable x, we can transform [16] the Boolean functions $\neg f$ and $f_1 \wedge f_2$ into the arithmetic expressions $1-p_f$ and $p_{f1} \cdot p_{f2}$, respectively, where p_f represents the polynomial assigned to the Boolean function f. By using the law of DeMorgan and idempotence, we can also transform $f_1 \vee f_2$ and $f_1 \oplus f_2$ into $p_{f1}+p_{f2}-p_{f1} \cdot p_{f2}$ and $p_{f1}+p_{f2}-2 \cdot p_{f1} \cdot p_{f2}$, respectively. It must be noted that the above arithmetic operations are carried out on the selected finite field.

In this paper, we consider the Galois field $GF(2^m)$ which is a characteristic 2 finite field with 2^m elements, each of them represented as an m-bit vector. $GF(2^m)$ is an extension field of the ground field $GF(2)=\{0,1\}$. The nonzero elements of $GF(2^m)$ are generated by a *primitive* element α, where α is a root of a primitive irreducible polynomial $g(x)=x^m+g_{m-1}x^{m-1}+\ldots+g_0$ over $GF(2)$. The nonzero elements of $GF(2^m)$ can be represented as the powers of α, *i.e.*, $GF(2^m)=\{0, \alpha^1, \alpha^2, \ldots, \alpha^{2^m-2}, \alpha^{2^m-1}=1\}$. Since α is a root of $g(x)$, $g(\alpha)=0$, and $\alpha^m=g_{m-1}\alpha^m+\ldots+g_1\alpha+g_0$. Therefore, an element of $GF(2^m)$ can also be expressed as a polynomial of α with degree less than m, that is, $GF(2^m)=\{a_{m-1}\alpha^{m-1}+\ldots+a_1\alpha+a_0 \mid a_i \in GF(2)$ for $0 \le i \le m-1\}$. Arithmetic in a field of characteristic 2 is essentially modulo arithmetic. Therefore, the addition of two polynomials becomes the bitwise XOR of the corresponding binary representations, and subtraction is the same as addition. Multiplication of two polynomials is the most important and one of the most complex and time-consuming operations. Complexity depends on many factors, such as the selection of the irreducible polynomial or the basis selected to represent the field elements: polynomial, dual or normal bases [25]. Because of its characteristic 2, the product $2 \cdot p_{f1} \cdot p_{f2}$ in $GF(2^m)$ is zero, and the above polynomial for the XOR is simplified to the $GF(2^m)$ addition. Complement of a field element is simply the complementation of its least significant bit (in polynomial basis), and multiplication can be carried out in any of the representation bases. It must be noted that the multiplication operation in $GF(2^m)$ requires $O(m)$ elementary operations [27].

When the signatures $[H_1]$ and $[H_2]$ of two functions f_1 and f_2 to be checked for equivalence are computed (using the above transformations, and evaluating them over the Galois field), then the signatures are compared: if $[H_1] \neq [H_2]$, then the functions are inequivalent with certainty; but if $[H_1] = [H_2]$, then the functions are equivalent, with small probability of error.

The probability of error [16] is bounded by $1-e^{-n/p}$ (where n is the number of variables and p is the cardinality of the finite field) and if $p>>n$, then $1-e^{-n/p} \approx n/p$. Therefore the error probability associated with the probabilistic equivalence check can be reduced by either increasing the size of the field, or by making multiple runs (using on each run an independent set of random assignments and computing the signatures for the functions). If the signatures are different, we are sure that the two functions are not the same. If they are equal, we choose a new set of input assignments and reevaluate. The probability of erroneously deciding that the functions is equal decreases exponentially with the number of such runs.

Applying these concepts, the probabilistic equivalence test of two functions represented by their Mod2-OBDDs is determined by the algebraic transformation of the Mod2-OBDDs in terms of polynomials over $GF(2^m)$. A polynomial p_v: $(GF(2^m))^n \to GF(2^m)$ can be associated with each node v of the Mod2-OBDD as follows [28]:

$$p_v(x_{n-1},\ldots,x_0) = \\ = \begin{cases} 0(1), & v \ is \ 0(1)-sink \\ x \cdot p_{v1}(x_{n-1},\ldots,x_0)+(1-x)\cdot p_{v0}(x_{n-1},\ldots,x_0), & l(v)=x \in X_n \\ p_{v0}(x_{n-1},\ldots,x_0)+p_{v1}(x_{n-1},\ldots,x_0), & l(v)=\oplus \end{cases}$$

The polynomial associated with a Mod2-OBDD P is the polynomial associated with the root of P, so the equivalence test of two functions is the signature comparison of their Mod2-OBDD roots polynomials evaluated at the randomly chosen set of input variables selected from $GF(2^m)$.

Let P_f and P_g be two Mod2-OBDDs representing the Boolean functions f and g, and assume that $a_0,\ldots,a_{n-1} \in GF(2^m)$ are generated independently and uniformly at random. For the Boolean signatures p_f and p_g computed for the Mod2-OBDDs P_f and P_g it holds [28] that $p_f(a_0,\ldots,a_{n-1})=p_g(a_0,\ldots,a_{n-1})$, if $f=g$, and Prob $(p_f(a_0,\ldots,a_{n-1}) = p_g(a_0,\ldots,a_{n-1}))<½$, if $f\neq g$. Therefore, if two given signatures $p_f(a_0,\ldots,a_{n-1})$ and $p_g(a_0,\ldots,a_{n-1})$ are equal, then the functions f and g are equal only with a certain probability. An estimation of the probability that the signatures for two nodes representing different Boolean functions in a

Mod2-OBDD P are equal can be found in [28]. By using s different signatures per node the error probability computes to at most

$$error < \frac{size(P)^2 \cdot n^s}{2 \cdot |GF|^s} \tag{1}$$

where size(P) denotes the number of nodes of the Mod2-OBDD P, n is the number of variables, and $|GF|$ the cardinality of the finite field $GF(2^m)$. For our experiments given in Section 6, we use the field $GF(2^{16})$. Therefore, if we have, for example, a Mod2-OBDD with 10^7 nodes depending on 100 variables, we should use 6 different signatures per node in order to obtain an error probability of less than $6.31 \cdot 10^{-4}$. It must be noted that for the circuit c1908 studied in section 6, for example, we should use 4 different signatures in order to obtain an error probability of less than $1.25 \cdot 10^{-5}$. However, this is an error bound. In fact, in our experiments and for only one signature per node, all the experiments performed in order to check the equivalence of two different circuits, were successfully detected by obtaining two different signatures. Furthermore, the work here presented is a first approach that can be further improved.

4. Including Signatures on CUDD Package

We have used for our implementations the CUDD package, one of the most efficient OBDD packages [29]. CUDD package has been modified in order to include signatures into OBDD nodes, and compute the signature of the Boolean function represented by an OBDD. The signature of the function will be the signature of the root node(s) of the OBDD. We name these decision diagrams signature-OBDDs (or *s*-OBDDs for simplicity). When the *s*-OBDD for a function has been constructed, then it can be verified comparing its signature with the signature computed by the construction of the Mod2-OBDD.

CUDD modifications consist of the inclusion of two new fields for each OBDD node: a 1-bit decision-x or field used to distinguish a decision node from a⊕-node, and a 32-bit signature field used for the signature of the node. The most significant 16-bits of this field store the initial signature randomly assigned to the variable associated with the node, and the least significant 16-bits store the signature of the (sub)function represented by the node. Therefore, $GF(2^{16})$ is used for signature representation.

In the synthesis process, Boolean operations are implemented using the ite operator [30] defined as a ternary Boolean function for three inputs F, G, H by "If F then G else H". This is equivalent to $ite(F,G,H)=F \cdot G+\overline{F} \cdot H$ and can be evaluated by recursive application of the Shannon decomposition $f=x \cdot f|_{x=1}+ \overline{x} \cdot f|_{x=0}$, where positive and negative cofactors $f|_{x=1}$ and $f|_{x=0}$ are the function f evaluated at x=1 and x=0, respectively. Positive and negative cofactors of a function associated with a decision node v, $l(v)=x$, are derived by simply returning the 1-successor and the 0-successor of v, respectively. In our approach, signature computation is performed in the synthesis process using the Shannon decomposition. Let $[x]$ be the signature initially assigned to x variable (the most significant 16-bits of v's signature field), and $[v_0]$ and $[v_1]$ the signatures associated to *0*- and 1-successor of v node (signatures of the functions represented by v_0 and v_1), respectively. Then the signature $[v]$ associated to the function represented by v can be computed by the following arithmetic operations over $GF(2^{16})$:

$$\begin{aligned}[v] &= \overline{[x]} \cdot [v_0] + [x] \cdot [v_1] = (1-[x]) \cdot [v_0] + [x] \cdot [v_1] \\ &= (1+[x]) \cdot [v_0] + [x] \cdot [v_1]\end{aligned} \tag{2}$$

where the properties and transformations of Boolean functions into arithmetic expressions given in Section 3 have been used. If the polynomial basis for the representation of the field elements is used, then the signature 1+[x] is simply the complementation of the least significant bit of [x]. When the signature [v] of the function represented by v has been computed, then it is stored at the least significant 16-bits of the signature field of v. Therefore, the signature of a Boolean function represented by an OBDD can be computed in the synthesis process (ite operation). An OBDD with signatures is named signature-OBDD (*s*-OBDD).

5. Introduction of ⊕-Nodes into the OBDD

Mod2-OBDD for a Boolean function is created by the introduction of an upper two-level layer of ⊕-nodes while the OBDD is being constructed. The main advantage of a data structure with OBDDs and ⊕-nodes is that the signature for a ⊕-node can be directly computed by simply performing the bitwise exclusive-or of the signature associated to its 1- and 0-successors (if Galois field is used).

We have constructed Mod2-OBDDs for combinational circuits given in BLIF format. For the introduction of the two-level layer of ⊕-nodes, we have used the positive Davio expansion. The positive Davio expansion (pDE) of a function f with reference to a variable x_i is given by the following expression:

$$\begin{aligned} f(\ldots,x_i,\ldots) &= f|_{x_i=0} \oplus x_i \cdot (f|_{x_i=0} \oplus f|_{x_i=1}) \\ &= f|_{x_i=0} \oplus (x_i \cdot f|_{x_i=0} \oplus x_i \cdot f|_{x_i=1}) \end{aligned} \tag{3}$$

Using this expansion, an upper layer of two ⊕-nodes is constructed for the function as shown in Figure 1(a). In this representation, the upper ⊕-node of the layer has its else edge pointing to the function $f|_{x_i=0}$, while its then edge points to the lower ⊕-node. The else edge of this lower ⊕-node points to the function $x_i \cdot f|_{x_i=0}$ and the then edge points to the function $x_i \cdot f|_{x_i=1}$. This structure is given in Figure 1(b) for a circuit with many outputs and when a given variable x_i is selected for the pDE. This expansion could be used for the whole circuit construction in Mod2-OBDD form, but we have restricted the ⊕-nodes inclusion to a two-level layer of ⊕-nodes and for a selected input variable for the pDE. This general structure has been selected because the number of ⊕-nodes in the Mod2-OBDD impacts on its size. For many circuits, a few ⊕-nodes lead to small sizes of Mod2-OBDDs, but too many ⊕-nodes lead to large Mod2-OBDDs [28]. The introduction of a limited number of ⊕-nodes in a mainly OBDD structure, tries to take advantage of both approaches (Mod2-OBDDs and OBDDs).

This two-level layer structure is constructed in the synthesis of the BLIF circuit. In BLIF, there are primary inputs, outputs and internal gates. A gate can have only primary inputs as fanins, or can have primary inputs and/or internal gates as inputs. A gate is represented with several lines, each line representing a cube, and the disjunction of these cubes provides the function of that gate. In general, the cubes are not disjoint. Each entry of a line, gives the value (0, 1 or –) that an input to that gate takes.

For the construction of the Mod2-OBDD with two-level layer ⊕-nodes, we select a primary input x_i with reference to which the positive Davio decomposition is performed. We will use the same variable for the pDE in all gates of the circuit. For each gate, we will get two functions $F|_{x_i=0}$ and $F|_{x_i=1}$ that represent the negative and positive cofactors, respectively, with reference to the selected variable x_i. For any gate of the circuit, two cases can be distinguished: a) the gate only has primary inputs as fanins, and b) the gate has primary inputs and/or internal gates as inputs.

In case a), for each line (cube) of the gate we have to perform the conjunction of all the primary inputs different from x_i (we name the result of this conjunction as product). Then the value that the input x_i has for that cube is checked. The possible cases we can take for computing the negative and positive cofactors for each cube ($C|_{x_i=0}$ and $C|_{x_i=1}$) are the following:

- $x_i=0 \Rightarrow$ ($C|_{x_i=0}$ = product) and ($C|_{x_i=1}$ =0).
- $x_i=1 \Rightarrow$ ($C|_{x_i=0}$ =0) and ($C|_{x_i=1}$ =product)
- $x_i = - \Rightarrow$ ($C|_{x_i=0}$ =product) and ($C|_{x_i=1}$ =product)

These operations are carried out for each line of the gate, and finally we compute the conjunction of the $C|_{x_i=0}$'s functions (getting $F|_{x_i=0}$) and the conjunction of the $C|_{x_i=1}$'s functions (getting $F|_{x_i=1}$) obtained for all the lines. After that, we get the two functions $F|_{x_i=0}$ and $F|_{x_i=1}$ that represent the gate.

In case b), we have that some inputs to the gate (the internal gates) have been already decomposed with respect to the x_i variable. If we name the conjunction of the $F|_{x_i=0}$'s functions of these internal gates as $N|_{x_i=0}$, and the conjunction of the $F|_{x_i=1}$'s of the internal gates as $N|_{x_i=1}$, then we will have the following three possibilities in order to get the cofactors $C|_{x_i=0}$ and $C|_{x_i=1}$ for each cube:

- $x_i=0 \Rightarrow$ ($C|_{x_i=0}$ = product$\cdot N|_{x_i=0}$) and ($C|_{x_i=1}$ = 0).
- $x_i=1 \Rightarrow$ ($C|_{x_i=0}$ = 0) and ($C|_{x_i=1}$ = product$\cdot N|_{x_i=1}$).
- $x_i = - \Rightarrow$ ($C|_{x_i=0}$ =product$\cdot N|_{x_i=0}$) and ($C|_{x_i=1}$ =product$\cdot N|_{x_i=1}$)

As in case a), these operations are performed for each cube of the gate. Finally, the conjunction of all the $C|_{x_i=0}$'s functions obtained for all the lines gives $F|_{x_i=0}$, and the conjunction of all the $C|_{x_i=1}$'s functions obtained for all the lines gives $F|_{x_i=1}$, so $F|_{x_i=0}$ and $F|_{x_i=1}$ represent the gate.

The graph so created for internal gates has a structure similar to Figure 1(a), except that then and else edges of the lower ⊕-node are not multiplied by the x_i variable. This is because all decomposition in the circuit is carried out with respect to this variable and internal gates are used only as an intermediate step in order to get the outputs of the circuit, so it is not necessary. We must perform these multiplications if the gate is an output, obtaining the structure given in Figure 1(a). The Mod2-OBDD obtained with this method, is not a canonical representation. As we are interested in the signature computation of the circuit using this data structure, the graph can be simplified not including x_i variable in the Mod2-OBDD. The contribution of the variable is considered multiplying the signature of the lower ⊕-node in Figure 1(a) by the signature initially assigned to the variable x_i. By this way the size of the final Mod2-OBDD is even more reduced. The structure finally obtained is given in Figure 1(b), with an upper two-level layer of ⊕-nodes and classical OBDDs (in fact, signature-OBBDs) at the lower part of the graph. The ⊕-nodes used are introduced in the CUDD package by their specification in the decision-xor field of the nodes, and signatures are stored in their signature field. All arithmetic operations involved above must be performed over $GF(2^m)$ using the signature field of the nodes (decision and ⊕-nodes).

For the experimental results given in the following section, the variable x_i selected to perform the positive Davio decomposition has been the first variable given in the initial ordering for the benchmarks. It must be noted that several heuristics for variable ordering could be used, such as those based on topological or logic information of the circuit [31,33]. Therefore, the variable x_i selected to perform the Davio decomposition could be the first variable obtained in these new orderings.

6. Experimental Results

Experiments have been carried out in a Sun Ultra1 workstation with 128 Mbytes of main-memory for the verification of LGSynth91 Benchmark multilevel BLIF circuits, and the modified CUDD has been used for the construction of OBDDs, *s*-OBDDs and two-level layer Mod2-OBDDs. Arithmetic operations over the finite field GF(2^{16}) generated by the irreducible pentanomial $f(x)=x^{16}+x^5+x^3+x^2+1$ have been performed using a polynomial basis for representation of the field elements, and the multiplication algorithm used has been the one given in [32].

Firstly, OBDDs and *s*-OBDDs for some of the given benchmarks have been constructed, and their construction times have been compared. In the 2nd column of Table 1, the ratios between the times needed to construct *s*-OBDDs and OBDDs of the benchmarks are given. The number of nodes (sizes) of both graphs is the same. The difference is that *s*-OBDD computes the signature of the circuit. From the results, it can be observed that the *s*-OBDD construction is 2.3 times slower in average than OBDD construction (with mux.blif as worst case). Despite of this, the construction of *s*-OBDDs has not an excessive cost in time in comparison with OBDDs, because we can get canonical representation in OBDD form together with the signature(s) of the circuit, so we could perform deterministic and/or probabilistic verifications. It is important to note that the more time needed for the *s*-OBDD construction is mainly due to the GF(2^m) multiplication of signatures. Therefore, the use of more efficient multiplication algorithms [23] will reduce the time needed for the s-OBDD construction.

Secondly, we have constructed two-level layer Mod2-OBDDs for the benchmarks, as given in Section 5. The times needed for their construction have been compared with the times needed for *s*-OBDD construction, and their signatures have been compared in order to check the correctness of the results (probabilistic verification). In the 3rd column of Table 1, the ratios between the times

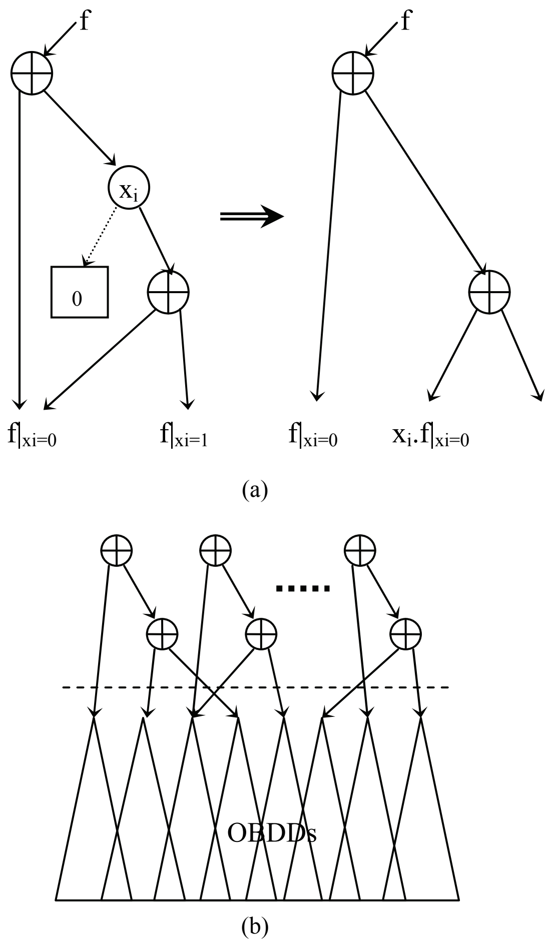

Figure 1. (a) Positive Davio expansion for variable x_i; (b) Structure for multiple outputs.

Table 1. Comparison of time ratios for LGSynth91 multi-level benchmarks.

Circuit	*s*-OBDD/OBDD	Mod2-OBDD/*s*-OBDD
alu2	2.4	0.80
apex6	2.1	0.66
apex7	2.4	0.22
c1355	2.4	0.99
c1908	2.3	0.44
cm151a	2.5	0.05
cordic	1.5	0.77
count	1.5	0.90
des	2.9	0.96
example2	1.7	0.83
frg2	2.3	0.93
i2	2.3	0.70
k2	2.9	0.16
mux	3.1	0.94
pcler8	2.0	0.70
term1	2.5	0.73
too_large	3.0	0.61
ttt2	2.0	0.87
vda	2.9	0.22
x3	2.3	0.51
x4	1.8	0.84

Table 2. Sizes for OBDDs, two-level layer Mod2-OBDDs, and ratios between sizes of Mod2-OBDD and OBDDS.

Circuit	OBDD	Mod2-OBDD	Mod2-OBDD/OBDD
alu2	231	231	1.00
apex6	2760	1712	0.62
apex7	1660	566	0.34
c1355	45922	45953	1.00
c1908	36007	19697	0.55
cm151a	511	49	0.10
cordic	45	50	1.11
count	234	232	0.99
des	73919	74028	1.00
example2	469	553	1.18
frg2	6471	6216	0.96
i2	335	268	0.80
k2	28336	7351	0.26
mux	131071	131071	1.00
pcler8	139	146	1.05
term1	580	459	0.79
too_large	7096	5129	0.72
ttt2	223	228	1.02
vda	4345	1919	0.44
x3	2760	1712	0.62
x4	891	917	1.03
Total size	344005	298487	0.87

needed to construct two-level layer Mod2- OBDDs and *s*-OBDDs of the benchmarks are given. It can be observed that in all cases, Mod2-OBDD construction is faster than *s*-OBDD construction (0.66 times faster in average), even with hard benchmarks as c1908.blif (0.44 times faster).

The sizes (number of nodes) of OBDDs and two-level layer Mod2-OBDDs have been also compared. In the 2nd column of Table 2 the OBDD sizes for the benchmarks are given, in the 3rd column the two-level layer Mod2-OBDD sizes are also given, and in the 4th column the ratios between the Mod2-OBDD and OBDD sizes are represented. From the experimental results, it can be observed that Mod2-OBDD sizes are in average 0.87 times smaller than OBDD sizes. Significant examples are apex7, c1908, k2, vda and x3 blif files, which Mod2-OBDD sizes are 0.34, 0.55, 0.26, 0.44 and 0.62 times smaller than OBDD sizes, respectively.

We can also compare our method with other similar approaches given in the literature. The comparison of the sizes obtained by our method with the best results obtained in the work of Meinel and Sack [34] for some Benchmarks is given in Table 3. In [34], Mod2-OBDDs are constructed depending on a threshold factor, and dynamic variable reordering techniques are neither used. From this comparison, it can be observed that the Mod2-OBDD sizes obtained by Meinel and Sack (with threshold factor equal to 1.0) are 0.99 times smaller than OBDD sizes, while the two-level layer Mod2-OBDD sizes obtained using our approach is 0.83 times smaller than OBDD sizes. In this table, ratio represents the ratio between the Mod2-OBDD and OBDD sizes.

Finally, from the experimental results, it can be observed that the probabilistic verification procedure using signature-OBDDs and two-level layer Mod2-OBDDs seems to be a promising approach for verification. The times needed for the construction of signature-OBDDs are not very large compared with the times needed for the construction of classical OBDDs, and the times needed for two-level layer Mod2-OBDD construction are always smaller than the times needed for *s*-OBDDs. Furthermore, the two-level layer Mod2-OBDDs sizes are in average smaller (in some cases, much smaller) than classical OBDDs sizes. The comparison of our experimental results with similar work done in the literature shows the suitability of our approach in order to obtain Mod2-OBDDs with reduced sizes. The achieved results could be further improved by trying heuristics for initial variable ordering, using dynamic variable reordering [34], [35], and using efficient multiplication algorithms over $GF(2^m)$, which is part of our ongoing work.

7. Conclusions

Probabilistic approach seems to be a promising alternative for circuit verification. For probabilistic methods

Table 3. Results of the comparison of our approach with the work done by Meinel and Sack for some benchmarks.

Circuit	OBDD	Meinel & Sack		Our approach	
		Mod2-OBDD	ratio	Mod2-OBDD	ratio
alu2	231	237	1.02	231	1.00
apex7	1660	1570	0.94	566	0.34
c1355	45922	45921	1.00	45953	1.00
c1908	36007	35869	0.99	19697	0.55
count	234	226	0.96	232	0.99
example2	469	456	0.97	553	1.18
frg2	6471	6348	0.98	6216	0.96
i2	335	334	1.00	268	0.80
k2	28336	26361	0.93	7351	0.26
mux	131071	131072	1.00	131071	1.00
term1	580	584	1.01	459	0.79
too_large	7096	7091	1.00	5129	0.72
vda	4345	4214	0.97	1919	0.44
x3	2760	2429	0.88	1712	0.62
Total size	265517	262712	0.99	221357	0.83

where Galois fields GF(2^m) are used, Mod2-OBDDs are suitable data structures for signature computation due tothe properties of finite fields with characteristic 2. In this work, a highly optimised package (CUDD) for OBDD construction has been modified to compute signatures in the synthesis process (signature-OBDD) and in order to construct two-level layer Mod2-OBDDs using positive Davio expansion with reference to a selected variable. Signatures obtained from signature-OBDDs and those obtained from Mod2-OBDDs are then compared for checking correctness (probabilistic verification). Experimental results have proven that the signature computation has not an excessive time cost and that Mod2-OBDDs with controlled number of ⊕-nodes provide reduced sizes compared with classical OBDDs, so probabilistic verification can be a suitable alternative to classical verification methods. Comparisons with experimental results obtained by other similar approaches found in the literature have been also given, proving that our method is very suitable for the construction of reduced Mod2-OBDDs.

8. References

[1] C. Y. Lee, "Representation of switching circuits by binary-decision programs," Bell Systems Technology Journal, Vol. 38, pp. 985–999, 1959.

[2] S. B. Akers, "Binary decision diagrams," IEEE Transactions on Computers, Vol. C-27, pp, 509–516, 1978.

[3] L. Fortune, J. Hopcroft, and E. M. Schmidt, "The complexity of equivalence and containment for free single variable program schemes," in: Goos, Hartmanis, Ausiello, Baum (Eds.), Lecture Notes in Computer Science, Springer-Verlag, New York, Vol. 62, pp. 227–240, 1978.

[4] R. E. Bryant, "Graph based algorithms for Boolean function representation," IEEE Transactions on Computers, Vol. C-35, pp. 677–690, 1986.

[5] R. Drechsler, B. Becker, and N. Göckel, "A genetic algorithm for variable ordering of OBDDs," International Workshop on Logic Synthesis, pp. P5c:5.55–5.64, 1995.

[6] P. W. C. Prasad, A. Assi, A. Harb, and V. C. Prasad, "Binary decision diagrams: An improved variable ordering using graph representation of boolean functions," International Journal of Computer Science, Vol. 1, No. 1, pp. 1–7, 2006.

[7] J. R. Burch and V. Singhal, "Tight integration of combinational verification methods," IEEE/ACM International Conference on CAD, pp. 570–576, 1998.

[8] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," Proceedings of Design Automation Conference, pp. 263–268, 1997.

[9] V. Paruthi and A. Kuehlmann, "Equivalence checking using cuts a structural SAT-solver, BDDs and simulation," International Conference Computer Design, 2000.

[10] E. Goldberg, M. R. Parasad, and R. K. Brayton, "Using SAT for combinational equivalence checking," IEEE/ACM Design, Automation and Test in Europe, Conference and Exhibition'01, pp. 114–121, 2001.

[11] J. Marques-Silva and T. Glass, "Combinational equivalence checking using satisfiability and recursive learning," IEEE/ACM Design, Automation and Test in Europe, pp. 145–149, 1999.

[12] D. Brand, "Verification of large synthesized designs," IEEE/ACM International Conference on Computer-Aided Design, pp. 534–537, 1993.

[13] W. Kunz, "HANNIBAL: An efficient tool for logic verification based on recursive learning," IEEE/ACM International Conference on CAD, pp. 538–543, November 1993.

[14] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary decision diagrams," ACM Computing Surveys, Vol. 24, No. 3, pp. 293–318, 1992.

[15] M. Blum, A. K. Chandra, and M. N. Wegman, "Equivalence of free Boolean graphs can be decided probabilistically in polynomial time," Information Processing Letters, Vol. 10, No. 2, pp. 80–82, 1980.

[16] J. Jain, J. Bitner, D. Fussell, and J. Abraham, "Probabilistic verification of Boolean functions," Formal Methods in System Design, Kluwer, Vol. 1, pp. 61–115, 1992.

[17] R. E. Bryant and Y. Cheng, "Verification of arithmetic functions with binary moment diagrams," Carnegie Mellon University Technical Report: CMU-CS-94-160, May 1994.

[18] R. Drechsler, B. Becker, and S. Ruppertz, "K*BMDs: A new data structure for verification," IEEE European Design and Test Conference, pp. 2–8, 1996.

[19] U. Kebschull, E. Schubert, and W. Rosentiel, "Multilevel logic based on functional decision diagrams," European Design Automation Conference, pp. 43–47, 1992.

[20] Y. T. Lai and S. Sastry, "Edge-valued binary decision diagrams for multi-level hierarchical verification," 29th Design Automation Conference, pp. 608–613, 1992.

[21] J. Gergov and C. Meinel, "Mod2-OBDDs: A data structure that generalizes exor-sum-of-products and ordered binary decision diagrams," Formal Methods in System Design, Kluwer, Vol. 8, pp. 273–282, 1996.

[22] S. Waack, "On the descriptive and algorithmic power of parity ordered binary decision diagrams," Proceedings of 14th Symposium on Theoretical Aspects of Computer Science, LNCS 1200, Springer, 1997.

[23] J. L. Imaña, J. M. Sánchez, and F. Tirado, "Bit-parallel finite field multipliers for irreducible trinomials," IEEE Transactions on Computers, Vol. 55, No. 5, pp. 520–533, May 2006.

[24] Ç. K. Koç and B. Sunar, "Low-complexity bit-parallel canonical and normal basis multipliers for a class of finite fields," IEEE Transactions on Computers, Vol. 47, No. 3, pp. 353–356, March 1998.

[25] R. Lidl and H. Niederreiter, “Finite fields,” Addison-Wesley, Reading, MA, 1983.

[26] J. C. Madre and J. P. Billón, “Proving Circuit correctness using formal comparison between expected and extracted behaviour,” Proceedings of 25th ACM/IEEE Design Automation Conference, pp. 308–313, 1988.

[27] P. A. Scott, S. E. Tavares, and L. E. Peppard, “A fast VLSI multiplier for GF(2^m),” IEEE Journal on Selected Areas in Communications, Vol. 4, pp. 62–66, 1986.

[28] C. Meinel and H. Sack, “⊕-OBDDs–A BDD structure for probabilistic verification,” Pre-LICS Workshop on Probabilistic Methods in Verification (PROBMIV’98), Indianapolis, IN, USA, pp. 141–151, 1998.

[29] F. Somenzi, “CUDD: CU decision diagram package,” University of Colorado at Boulder. http://vlsi.colorado. edu/ ~ fabio/CUDD/.

[30] K. S. Brace, R. L. Rudell, and R. E. Bryant, “Efficient implementation of a BDD package,” 27th ACM/IEEE Design Automation Conference, pp. 40–45, 1990.

[31] K. M. Butler, D. E. Ross, R. Kapur, and M. R. Mercer, “Heuristics to compute variable orderings for efficient manipulation of ordered binary decision diagrams,” Proceedings of 28th ACM/IEEE DAC, pp. 417–420, June 2001.

[32] C. S. Yeh, I. S. Reed, and T. K. Truong, “Systolic multipliers for finite fields GF(2^m),” IEEE Transactions on Computers, Vol. 33, No. 4, pp. 357–360, April 1984.

[33] M. Fujita, H. Fujisawa, and Y. Matsunaga, “Variable ordering algorithms for ordered binary decision diagrams and their evaluation,” IEEE Transactions on CAD, Vol. 12, No. 1, pp. 6–12, January 1993.

[34] C. Meinel and H. Sack, “Improving XOR-Node placement for ⊕-OBDDs,” 5th International Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Starkville, Mississippi, USA, pp. 51–56, 2001.

[35] C. Meinel and H. Sack, “Variable Reordering for ⊕-OBDDs,” International Symposium on Representations and Methodology of Future Computing Technologies (RM2003), Trier, Germany, pp. 135–144, 2003.