Biconditional Binary Decision Diagrams: A Novel Canonical Logic Representation Form

Luca Amarú, *Student Member, IEEE*, Pierre-Emmanuel Gaillardon, *Member, IEEE*, and Giovanni De Micheli, *Fellow, IEEE*

***Abstract*—In this paper, we present biconditional binary decision diagrams (BBDDs), a novel canonical representation form for Boolean functions. BBDDs are binary decision diagrams where the branching condition, and its associated logic expansion, is biconditional on two variables. Empowered by reduction and ordering rules, BBDDs are remarkably compact and unique for a Boolean function. The interest of such representation form in modern electronic design automation (EDA) is twofold. On the one hand, BBDDs improve the efficiency of traditional EDA tasks based on decision diagrams, especially for arithmetic intensive designs. On the other hand, BBDDs represent the natural and native design abstraction for emerging technologies where the circuit primitive is a comparator, rather than a simple switch. We provide, in this paper, a solid ground for BBDDs by studying their underlying theory and manipulation properties. Thanks to an efficient BBDD software package implementation, we validate 1) speed-up in traditional decision diagrams applications with up to 4.4× gain with respect to other DDs, and 2) improved synthesis of circuits in emerging technologies, with about 32% shorter critical path than state-of-art synthesis techniques.**

***Index Terms*—Biconditional connective, canonicity, decision diagrams, design methods and tools, nanocircuits.**

I. Introduction

THE CHOICE of data structure is crucial in computing applications, especially for the automated design of digital circuits. When logic functions are concerned, binary decision diagrams (BDDs) [1]–[3] are a well-established cogent and unique, i.e., canonical, logic representation form. BDDs are widely used in electronic design automation (EDA) to accomplish important tasks, e.g., synthesis [4], verification [5], testing [6], simulation [7], and others. Valuable extensions [8] and generalizations [9] of BDDs have been proposed in literature to improve the performance of EDA applications based on decision diagrams. The corresponding software packages [10], [11] are indeed mature and supported by a solid theory. However, there are still combinational designs, such as multipliers and arithmetic circuits, that do not fit modern computational capabilities when represented by existing canonical decision diagrams [23]. The quest for new data structures handling such hard circuits, and possibly pushing further the performance for ordinary circuits, is of paramount importance for next-generation digital designs. Furthermore, the rise of emerging technologies carrying new logic primitives demands for novel logic representation forms that fully exploit a diverse logic expressive power. For instance, controllable polarity double-gate (DG) transistors, fabricated in silicon nanowires [12], carbon nanotubes [13] or graphene [14] technologies, but also nanorelays [15], intrinsically behave as comparators rather than switches. Hence, conventional data structures are not appropriate to model natively their functionality.

In this paper, we present BBDDs, a novel canonical BDD extension. While original BDDs are based on the single-variable Shannon's expansion, BBDDs employ a two-variable biconditional expansion, making the branching condition at each decision node dependent on two variables per time. Such feature improves the logic expressive power of the binary decision diagram. Moreover, BBDDs represent also the natural and native design abstraction for emerging technologies [12]–[15] where the circuit primitive is a comparator, rather than a switch.

Note that a preliminary version of this work has been published in [16], [17]. This work differentiates by providing 1) formal theory about BBDDs, 2) enhanced BBDD-manipulation algorithms, 3) extended experimental results and comparisons with packages for other decision diagrams, and 4) application of BBDDs to the synthesis and verification of circuits in emerging technologies.

We validate the benefits deriving from the use of BBDDs in EDA tasks through an efficient software manipulation package, available online [18]. Considering the MCNC benchmark suite, BBDDs are built 1.4× and 1.5× faster than original BDDs and Kronecker functional decision diagrams (KFDDs) [9], while having also 1.5× and 1.1× fewer nodes, respectively. Moreover, we show hard arithmetic circuits that fit computing capabilities with BBDDs but are not practical with state-of-art BDDs or KFDDs. Employed in the synthesis of an iterative decoder design, targeting standard CMOS technology, BBDDs advantageously pre-structure arithmetic circuits as front-end to a commercial synthesis tool, enabling to meet tight timing constraints otherwise beyond the capabilities of traditional synthesis. The combinatorial verification of the optimized design is also speeded-up by 11.3% using BBDDs in place of standard BDDs. Regarding the automated design for emerging technologies, we similarly employed BBDDs as front-end to a commercial synthesis tool but then targeting a controllable-polarity DG silicon nanowires field effect transistors

Manuscript received March 06, 2014; revised July 10, 2014; accepted August 04, 2014. Date of publication October 15, 2014; date of current version December 09, 2014. This work was supported by the European Research Council under Senior Grant ERC-2009-AdG-246810 "NANOSYS." This paper was recommended by Guest Editor S. Bhunia.

The authors are with the Integrated Systems Laboratory, Swiss Federal Institute of Technology, Lausanne, EPFL, 1015 Lausanne, Switzerland (e-mail: luca.amaru@epfl.ch; pierre-emmanuel.gaillardon@epfl.ch; giovanni.demicheli@epfl.ch).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2014.2361058

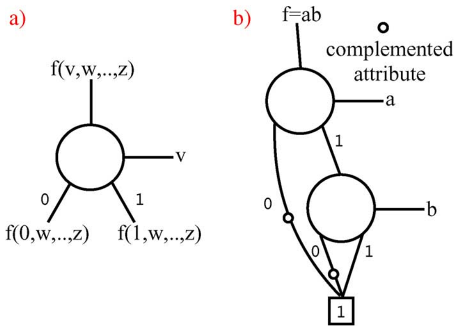


Fig. 1. BDD nonterminal node (a) canonical BDD for $a \cdot b$ function (b).

(SiNWFETs) technology [12]. Controllable-polarity DG-SiNWFETs behave as binary comparators [12]. Such primitive is naturally modelled by BBDDs. Experimental results show that the effectiveness of BBDD pre-structuring for circuits based on such devices is even higher than for standard CMOS, thus enabling a superior exploitation of the emerging technology features.

The remainder of this paper is organized as follows. Section II first provides a background on BDDs and then discusses the motivations for the study of BBDDs. In Section III, the formal theory for BBDDs is introduced, together with efficient manipulation algorithms. Section IV first shows theoretical size bounds for notable functions represented with BBDDs and then compares the performance of our BBDD software package with other state-of-art packages for BDDs and KFDDs. Section V presents the application of BBDDs to circuit synthesis and verification in both traditional and emerging technologies. The paper is concluded in Section VI.

II. Background and Motivation

This section first provides the background and the basic terminology associated with BDDs and their extensions. Then, it discusses the motivations to study BBDDs, from both a traditional EDA and an emerging technology perspectives.

A. Binary Decision Diagrams

BDDs are logic representation structures first introduced by Lee [1] and Akers [2]. Ordering and reduction techniques for BDDs were introduced by Bryant in [3] where it was shown that, with these restrictions, BDDs are a canonical representation form. Canonical BDDs are often compact and easy to manipulate. For this reason, they are extensively used in EDA and computer science. In the following, we assume that the reader is familiar with basic concepts of Boolean algebra (for a review, see [19] and [20]) and we review hereafter the basic terminology used in the rest of the paper.

1) Terminology and Fundamentals: A BDD is a direct acyclic graph (DAG) representing a Boolean function. A BDD is uniquely identified by its *root*, the set of *internal nodes*, the set of *edges* and the *1/0-sink terminal nodes*. Each internal node [Fig. 1(a)] in a BDD is labeled by a Boolean variable v and has two out-edges labeled 0 and 1. Each internal node also represents the Shannon's expansion with respect to its variable v

$$f(v, w, \dots, z) = v \cdot f(1, w, \dots, z) + v' \cdot f(0, w, \dots, z). \quad (1)$$

The 1- and 0-edges connect to positive and negative Shannon's cofactors, respectively.

Edges are characterized by a regular/complemented attribute. Complemented edges indicate to invert the function pointed by that edge.

We refer hereafter to BDDs as to canonical reduced and ordered BDDs [3], that are BDDs where 1) each input variable is encountered at most once in each root to sink path and in the same order on all such paths, 2) each internal node represent a distinct logic function, and 3) only 0-edges can be complemented. Fig. 1(b) shows the BDD for function $f = a \cdot b$.

In the rest of this paper, symbols $\oplus$ and $\odot$ represent XOR and XNOR operators, respectively. Symbol $\otimes$ represents any two-operand Boolean operator.

2) Previous BDD Extensions: Despite BDDs are typically very compact, there are functions for which their representation is too large to be stored and manipulated. For example, it was shown in [23] that the BDD for the multiplier of two n-bit numbers has at least $2^{n/8}$ nodes. For this reason, several extensions of BDDs have been suggested.

One first extension are free BDDs, where the variable order condition is relaxed allowing polynomial size representation for the multiplier [21]. However, such relaxation of the order sacrifices the canonicity of BDDs, making manipulation of such structures less efficient. Indeed, canonicity is a desirable property that permits operations on BDDs to have an efficient runtime complexity [3]. Another notable approach trading canonicity for compactness is parity-BDDs ($\oplus$-BDDs) presented in [24]. In $\oplus$-BDDs, a node can implement either the standard Shannon's expansion or the $\oplus$ (XOR) operator. Thanks to this increased flexibility, $\oplus$-BDDs allow certain functions having exponential size with original BDDs to be instead represented in polynomial size. Again, the manipulation of $\oplus$-BDDs is not as efficient as with original BDDs due to the heterogeneity introduced in the diagrams by additional $\oplus$-nodes.

Considering now BDD extensions preserving canonicity, zero-suppressed BDDs [29] are BDDs with modified reduction rules (node elimination) targeting efficient manipulation of sparse sets. transformation BDDs (TBDDs) [32], [34] are BDDs where the input variables of the decision diagram are determined by a logic transformation of the original inputs. When the input transformation is an injective mapping, TBDDs are canonical representation form [34]. In theory, TBDDs can represent every logic function with polynomial size given the perfect input transformation. However, the search for the perfect input transformation is an intractable problem. Moreover, traditional decision diagram manipulation algorithms (e.g., variable reordering) are not efficient with general TBDDs due to the presence of the input transformation [21]. Nevertheless, helpful and practical TBDDs have been proposed in literature, such as linear sifting of BDDs [30], [31] and hybrid decision diagrams (HDDs) [33]. Linear sifting consists of linear transformations between input variables carried out online during construction. The linear transformations are kept if they reduce the size of the BDD or undone in the other case. On the one hand, this makes the linear transform dependent itself on the considered BDD and therefore very effective to reduce its size. On the other hand, different BDDs may have different transforms and logic operations between them become more complicated. More discussion for linear sifting and comparisons to our proposed BDD extension are given in Section III-A. HDDs are TBDDs having as transformation matrix the Kronecker product of different 2×2 matrices. The entries of such matrices are determined via heuristic algorithms. HDDs are reported to achieve a remarkable size compression factor (up to three orders of magnitude) with respect to BDDs [33] but they suffer similar limitations as linear sifting deriving from the dependency on the particular, case-dependent, input transformation employed.

Other canonical extensions of BDDs are based on different core logic expansions driving the decision diagram. Functional decision diagrams (FDDs) [8] fall in this category employing the (positive) Davio's expansion in place of the Shannon's one

$$f(v, w, \ldots, z) = f(0, w, \ldots, z) \oplus v \cdot (f(0, w, \ldots, z) \oplus f(1, w, \ldots, z)). \quad (2)$$

Since the Davio expansion is based on the $\oplus$ operator, FDDs provide competitive representations for XOR-intensive functions. Kronecker FDDs (KFDDs) [9] are a canonical evolution of FDDs that can employ both Davio's expansions (positive and negative) and Shannon's expansion in the same decision diagram provided that all the nodes belonging to the same level use the same decomposition type. As a consequence, KFDDs are a superset of both FDDs and BDDs. However, the heterogeneity of logic expansion types employable in KFDDs makes their manipulation slightly more complicated than with standard BDDs. For problems that are more naturally stated in the discrete domain rather than in terms of binary values, multi-valued decision diagrams (MDDs) have been proposed [39] as direct extension of BDDs. MDDs have multiple edges, as many as the cardinality of the function domain, and multiple sink nodes, as many as the cardinality of the function codomain. We refer the reader to [21] for more details about MDDs.

Note that the list of BDD extensions considered above is not complete. Due to the large number of extensions proposed in literature, we have discussed only those relevant for the comprehension of this work.

In this paper, we present a novel canonical BDD extension where the branching decisions are biconditional on two variables per time rather than on only one. The motivation for this study is twofold. First, from a theoretical perspective, considering two variables per time enhances the expressive power of a decision diagram. Second, from an application perspective, there exist emerging devices better modeled by a two-variable (biconditional) comparator rather than a single variable switch. In this context, the proposed BDD extension serves as natural logic abstraction. A discussion about the technology motivation for this work is provided hereafter.

B. Emerging Technologies

Many logic representation forms are inspired by the underlying functionality of contemporary digital circuits. Silicon-based metal–oxide–semiconductor field-effect transistors (MOSFETs) form the elementary blocks for present electronics. In the digital domain, a silicon transistor behaves as a two-terminal binary switch driven by a single input signal. The Shannon's expansion captures such operation in the form of a Boolean expression. Based on it, logic representation and manipulation of digital circuits is efficient and automated.

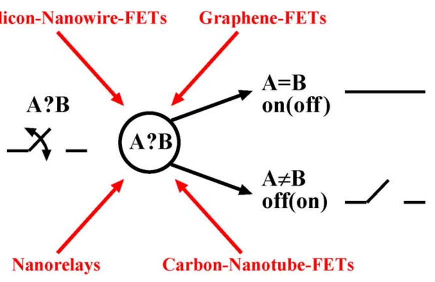


Fig. 2. Common logic abstraction for emerging devices: controllable polarity double-gate FETs in silicon nanowires [12], carbon nanotubes [13], graphene [14] but also six terminal nanorelays [15].

With the aim to support the exponential growth of digital electronics in the future, novel elementary blocks are currently under investigation to overcome the physical limitations of standard transistors. Deriving from materials, geometries and physical phenomena different than MOSFETs, many emerging devices are not naturally modeled by traditional logic representation forms. Therefore, novel CAD methodologies are needed, which appropriately handle such emerging devices.

We concentrate here on a promising class of emerging devices that inherently implement a two-input comparator rather than a simple switch. These innovative devices come in different technologies, such as silicon nanowires [12], carbon nanotubes [13], graphene [14], and nanorelays [15]. In the first three approaches, the basic element is a double-gate controllable-polarity transistor. It enables online configuration of the device polarity (n or p) by adjusting the voltage at the second gate. Consequently, in such a double-gate transistor, the *on/off* state is biconditional on both gates values. The basic element in the last approach [15] is instead a six-terminals nanorelays. It can implement complex switching functions by controlling the voltages at the different terminals. Following to its geometry and physics, the final electric way connection in the nanorelay is biconditional on the terminal values[15]. Even though they are based on different technologies, all the devices in [12]–[15] have the same common logic abstraction, depicted by Fig. 2.

In this work, we focus on double-gate controllable polarity SiNWFETs [12] to showcase the impact of novel logic representation forms in emerging technology synthesis. A device sketch and fabrication views from [12] are reported in Fig. 3 for the sake of clarity. While having an enhanced functionality, this device also presents an implementation overhead deriving from the second gate. Without a dedicated logic abstraction and synthesis methodology, the full potential of this technology may remain obscured by the extra fabrication cost. We propose in this paper a novel logic representation form, based on the biconditional connective, that naturally harnesses the operation of controllable polarity switches. By using this logic abstraction in synthesis, the full logic expressive power of double-gate controllable polarity SiNWFETs, but also of devices in [13]–[15], is unlocked. Section V will show the impact of our representation form in the synthesis of high-performance SiNWFET circuits.

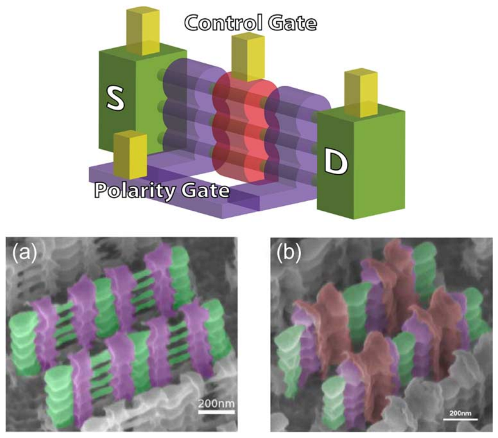


Fig. 3. Sketch structure and fabrication images of controllable polarity double-gate SiNWFETs from [12].

III. Biconditional Binary Decision Diagrams

This section introduces BBDDs. First, it presents the core logic expansion that drives BBDDs. Then, it gives ordering and reduction rules that makes reduced and ordered BBDDs (ROBBDDs) compact and canonical. Finally, it discusses efficient algorithms for BBDD manipulation and their practical implementation in a software package.

A. Biconditional Expansion

Logic expansions, also called decompositions, are the driving core of various types of decision diagrams. In [41], a theoretical study concluded that, among all the possible one-variable expansions, only Shannon's, positive Davio's and negative Davio's types help to reduce the size of decision diagrams. While this result prevents from introducing more one-variable decomposition types, new multi-variable decompositions are still of interest. In this work, we consider a novel logic expansion, called biconditional expansion, examining two variables per time rather than one, in order to produce novel compact decision diagrams. The biconditional expansion is one of the many possible two-variable decompositions. Note that other advantageous two-variable decompositions may exist but their study is out of the scope of this work.

Definition 1: The biconditional expansion is a two-variable expansion defined $\forall f \in \mathbb{B}^n$, with $n > 1$, as

$$f(v,w,\ldots,z)=(v\oplus w)\cdot f(w',w,\ldots,z)+(v\odot w)\cdot f(w,w,\ldots,z) \tag{3}$$

with v and w distinct elements in the support for function f.

As per the biconditional expansion in (3), only functions that have two or more variables can be decomposed. Indeed, in single variable functions, the terms $(v \oplus w)$ and $(v \odot w)$ cannot be computed. In such a condition, the biconditional expansion of a single variable function can reduce to a Shannon's expansion by fixing the second variable w to logic 1. With this boundary condition, any Boolean function can be fully decomposed by biconditional expansions.

Note that a similar concept to biconditional expansion appears in [30], [31] where linear transformations are applied to BDDs. The proposed transformation replaces one variable x_i with $x_i \odot x_j$. In the BDD domain, $x_i \mapsto x_i \odot x_j$ transforms a Shannon's expansion around variable x_i into a biconditional expansion around variables x_i and x_j. We differentiate our work from linear transformations by the abstraction level at which we embed the biconditional connective. Linear transformations in [30], [31] operate as postprocessing of a regular BDD, while we propose to entirely substitute the Shannon's expansion with the biconditional expansion. By changing the core engine driving the decision diagram new compact representation opportunities arise. However, a solid theoretical foundation is needed to exploit such potential. We address this requirement in the rest of this section.

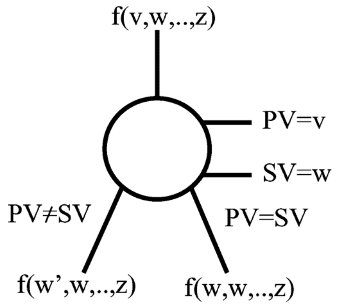


Fig. 4. BBDD nonterminal node.

B. BBDD Structure and Ordering

BBDD are driven by the biconditional expansion. Each nonterminal node in a BBDD has the branching condition biconditional on two variables. We call these two variables the primary variable (PV) and the secondary variable (SV). An example of a BBDD *nonterminal* node is provided by Fig. 4. We refer hereafter to $PV \neq SV$ and $PV = SV$ edges in a BBDD node simply as $\neq$-*edges* and $=$-*edges*, respectively.

To achieve ordered BBDDs (OBBDDs), a variable order must be imposed for PVs and a rule for the other variables assignment must be provided. We propose the chain variable order (CVO) to address this task. Given a Boolean function f and a variable order $\pi = (\pi_0, \pi_1, \ldots, \pi_{n-1})$ for the support variables of f, the CVO assigns PVs and SVs as

$$\begin{cases} PV_i = \pi_i \\ SV_i = \pi_{i+1} \end{cases} \text{with } i = 0,1,\ldots,n-2; \begin{cases} PV_{n-1} = \pi_{n-1} \\ SV_{n-1} = 1 \end{cases}. \tag{4}$$

CVO Example: From $\pi = (\pi_0, \pi_1, \pi_2)$, the corresponding CVO ordering is obtained by the following method. First, $PV_0 = \pi_0$, $PV_1 = \pi_1$, and $SV_0 = \pi_1$, $SV_1 = \pi_2$ are assigned. Then, the final boundary conditions of (4) are applied as $PV_2 = \pi_2$ and $SV_2 = 1$. The consecutive ordering by couples $(\mathrm{PV}_i, \mathrm{SV}_i)$ is thus $((\pi_0, \pi_1), (\pi_1, \pi_2), (\pi_2, 1))$.

The CVO is a key factor enabling unique representation of ordered biconditional decision diagrams. For the sake of clarity, we first consider the effect of the CVO on *complete* OBBDDs and then we move to generic reduced BBDDs in the next subsection.

Definition 2: A *complete* OBBDD of n variables has $2^n - 1$ distinct internal nodes, no sharing, and 2^n terminal 0-1 nodes.

Lemma 1: For a Boolean function f and a variable order π, there exists only one *complete* OBBDD ordered with $\mathrm{CVO}(\pi)$.

Proof: Say n the number of variables in f. All *complete* OBBDD of n variables have an identical internal structure, i.e., a full binary tree having $2^n - 1$ internal nodes. The distinctive feature of a *complete* OBBDD for f is the distribution of terminal 0-1 nodes. We need to show that such distribution is unique in a *complete* OBBDD ordered with $\mathrm{CVO}(\pi)$. Consider the unique truth table for f with 2^n elements filled as per π. Note that in a *complete* OBBDD there are 2^n distinct paths by construction. We link the terminal value reached by each path to an element of the truth table. We do so by recovering the binary assignment of π generating a path. That binary assignment is the linking address to the truth table entry. For example, the terminal value reached by the path $(\pi_0 \neq \pi_1, \pi_1 = \pi_2, \pi_2 \neq 1)$ corresponds to the truth table entry at the address $(\pi_0 = 1, \pi_1 = 0, \pi_2 = 0)$. Note that distinct paths in the CVO(π) corresponds to distinct binary assignments of π, owing to the isomorphism induced by the *biconditional expansion*. By exhausting all the 2^n paths we are guaranteed to link all entries in the truth table. This procedure establishes a one-to-one correspondence between the truth table and the *complete* OBBDD. Since truth tables filled as per π are unique, also *complete* OBBDD ordered with $\mathrm{CVO}(\pi)$ are unique.*Q.E.D.*

We refer hereafter to OBBDDs as to BBDDs ordered by the CVO.

C. BBDD Reduction

In order to improve the representation efficiency, OBBDDs should be reduced according to a set of rules. We present hereafter BBDD reduction rules, and we discuss the uniqueness of the so obtained ordered and reduced BBDDs.

1) Reduction Rules: The straightforward extension of OBDD reduction rules [3] to OBBDDs, leads to weak Reduced OBBDDs (ROBBDDs). This kind of reduction is called *weak* due to the partial exploitation of OBBDD reduction opportunities. A *weak* ROBBDD is an OBBDD respecting the two following rules:

R1: It contains no two nodes, root of isomorphic subgraphs;

R2: It contains no nodes with identical children.

In addition, the OBBDD representation exhibits supplementary interesting features enabling further reduction opportunities. First, levels with no nodes (empty levels) may occur in OBBDDs. An empty level is a level in the decision diagram created by the Chain Variable Order but containing no nodes as a result of the augmented functionality of the *biconditional expansion*. Such levels must be removed to compact the original OBBDD. Second, subgraphs that represent functions of a single variable degenerates into a single DD node driven by the Shannon's expansion followed by the sink terminal node. The degenerated node functionality is the same as in a traditional BDD node. Single variable condition is detectable by checking the cardinality of the support set of the subgraph.

Formally, a *strong* ROBBDD is an OBBDD respecting **R1** and **R2** rules, and in addition:

R3: It contains no empty levels;

R4: Subgraphs representing single variable functions degenerates into a single DD node driven by the Shannon's expansion.

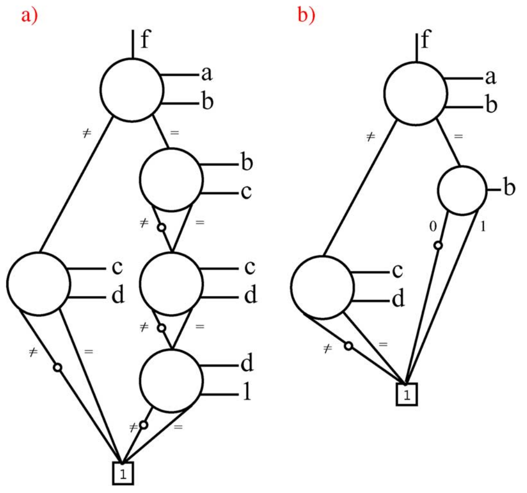


Fig. 5. Function to be represented: $f = a \cdot b + (a \oplus b) \cdot (c \odot d)$, *weak* ROBBDD for f (a) and *strong* ROBBDD for f (b).

For the sake of simplicity, we refer hereafter to a single variable subgraph degenerated into a single DD node as a BDD node.

Fig. 5 depicts *weak* and *strong* ROBBDDs for the function $f = a \cdot b + (a \oplus b) \cdot (c \odot d)$. The *weak* ROBBDD is forced to allocate four levels (one for each variable) to fully represent the target function resulting in five internal nodes. On the other hand, the *strong* ROBBDD exploits reduction rule **R4** collapsing the $=$-branch of the root node $(a = b)$ into a single BDD node. Moreover, rule **R3** suppresses empty level further compressing the diagram in three levels of depth and three internal nodes.

2) Canonicity: Weak and *strong* reduced OBBDDs are canonical, as per:

Lemma 2: For a given Boolean function f and a variable order π, there exists only one *weak* ROBBDD.

Proof: Weak ROBBDDs are obtained by applying reduction rules **R1** and **R2**, in any combination, to an OBBDD until no other **R1** or **R2** rule can be applied. Without loss of generality, suppose to start from a *complete* OBBDD. Any other valid OBBDD can be reached during the reduction procedure. In [43], it is shown that the iterative reduction of general decision diagrams, based on rules **R1** and **R2**, reaches a unique structure. Since the initial *complete* OBBDD is unique, owing to *Lemma 1*, and the iterative reduction based on rules **R1** and **R2** leads to a unique outcome, owing to [43], also *weak* ROBBDD are unique for a $\mathrm{CVO}(\pi)$, i.e., canonical. *Q.E.D.*

Theorem 1: A *strong* ROBBDD is a canonical representation for any Boolean function f.

Proof: Strong ROBBDDs can be directly derived by applying reduction rules **R3** and **R4**, in any combination, to *weak* ROBBDDs until no other **R3** or **R4** rule can be applied.

In order to prove the canonicity of *strong* ROBBDD, we proceed by five succeeding logical steps. The final goal is to show that any sequence of reductions drawn from {**R3**,**R4**}, that continues until no other reduction is possible, reaches a unique *strong* ROBBDD structure, preserving the uniqueness property of the starting *weak* ROBBDD.

1) Reductions **R3** and **R4** preserve distinctness. As it holds for rules **R1** and **R2**, also **R3** and **R4** preserve distinctness.

Rule **R3** compacts the decision diagram structure without any ambiguity in the elimination of levels, i.e., when a level is empty it is uniquely removed. Rule **R4** substitutes single variable functions with a single BDD node (followed by the sink node). This operation has a specific and unique outcome since it is combined with rules **R1** and **R2** (each node represents a distinct logic function).

2) The set of applicable rules **R4** is fixed. In a given *weak* ROBBDD, the set of all possible single variable subgraph collapsing (rule **R4**) is fixed *a priori*, i.e., there exists a specific set of applicable **R4** reductions independent of the reduction sequence employed. Consider a top-down exploration of the starting *weak* ROBBDD. At each branching condition, the support sets of the current node children are checked. If the cardinality of the support set is 1 (single variable) then this subgraph is reducible by **R4**. Regardless of the particular exploration order, the support set of all subgraphs remains the same. Therefore, the applicability of rules **R4** depends only on the given *weak* ROBBDD structure.
3) Rules **R4** are independent of rules **R3**. Rules **R3** (empty levels elimination) cannot preclude the exercise of rules **R4** (single-variable subgraphs collapsing) because they eliminate levels with no nodes, where no rule **R4** could apply.
4) Rules **R3** can be precluded by rules **R4**. Rules **R4** can preclude the exercise of rules **R3** since the collapse of subgraphs into a single node can make some levels in the decision diagram empty (see Fig. 5). Nevertheless, each rule **R3** is reachable in a reduction sequence that guarantees to exhaust all the blocking **R4** before its termination.
5) Iterative reduction strategy is order independent. We refer to an iterative reduction strategy as to a sequence of reductions drawn from {**R3**,**R4**} applied to a *weak* ROBBDD, that continues until no other reduction is possible. At each step of reduction sequence, the existence of a new reduction **R3** or **R4** is checked. Following points 2 and 3, all possible **R4** are identifiable and reachable at any time before the end of the reduction sequence, regardless of the order employed. Consider now rules **R3**. Some of them are not precluded by rules **R4**. Those are also identifiable and reachable at any time before the end of the reduction sequence. The remaining **R3** are precluded by some **R4**. However, all possible **R4**, included those blocking some **R3**, are guaranteed to be accomplished before the end of the reduction. Therefore, there always exists a step, in any reduction sequence, when each rule **R3** is reachable as the blocking **R4** are exhausted. Consequently, any iterative reduction strategy drawn from {**R3**,**R4**} achieves a unique reduced BBDD structure (*strong* ROBBDD).

It follows that any combination of reduction rules **R3** and **R4** compact a canonical *weak* ROBBDD into a unique *strong* ROBBDD, preserving canonicity. *Q.E.D.*

D. BBDD Complemented Edges

Being advantageously applied in modern ROBDDs packages [10], complemented edges indicate to invert the function pointed by an edge. The canonicity is preserved when the complement attribute is allowed only at 0-edges (only logic 1 terminal node available). Reduction rules **R1** and **R2** continue to be valid with complemented edges [21]. Similarly, we extend ROBBDDs to use complemented edges only at $\neq$-edges, with also only logic 1 terminal node available, to maintain canonicity.

Theorem 2: ROBBDDs with complemented edges allowed only at $\neq$-edges are canonical.

Proof: Reduction rules **R1** and **R2** support complemented edges at the else branch of canonical decision diagrams [21]. In BBDDs, the else branch is naturally the $\neq$-edge, as the *biconditional* connective is true (then branch) with the $=$-edge. We can therefore extend the proof of *Lemma 2* to use complemented edges at $\neq$-edges and to remove the logic 0 terminal node. It follows that *weak* ROBBDDs with complented edges at $\neq$-edges are canonical. The incremental reduction to *strong* ROBBDDs does not require any knowledge or action about edges. Indeed, the proof of *Theorem 1* maintains its validity with complemented edges. Consequently, *strong* ROBBDDs with complemented edges at $\neq$-edges are canonical. *Q.E.D.*

For the sake of simplicity, we refer hereafter to BBDDs as to canonical ROBBDDs with complemented edges, unless specified otherwise.

E. BBDD Manipulation

So far, we showed that, under ordering and reduction rules, BBDDs are unique and potentially very compact. In order to exploit such features in real-life tools, a practical theory for the construction and manipulation of BBDDs is needed. We address this requirement by presenting an efficient manipulation theory for BBDDs with a practical software implementation, available online at [18].

Rationale for Construction and Manipulation of BBDDs: DDs are usually built starting from a netlist of Boolean operations. A common strategy employed for the construction task is to build bottom-up the DD for each element in the netlist, as a result of logic operations between DDs computed in the previous steps. In this context, the core of the construction task is an efficient Boolean operation algorithm between DDs. In order to make such approach effective in practice, other tasks are also critical, such as memory organization and re-ordering of variables. With BBDDs, we follow the same construction and manipulation rationale, but with specialized techniques taking care of the *biconditional expansion*.

Considerations to Design an Efficient BBDD Package: Nowadays, one fundamental reason to keep decision diagrams small is not just to successfully fit them into the memory, that in a modern server could store up to 1 billion nodes, but more to maximize their manipulation performance. Following this trend, we design the BBDD manipulation algorithms and data structures aiming to minimize the runtime while keeping under control the memory footprint. The key concepts unlocking such target are 1) *unique* table to store BBDD nodes in a *strong canonical form*,[1] 2) recursive formulation of Boolean operations in terms of *biconditional expansions* with relative

[1]A strong canonical form is a form of data pre-conditioning to reduce the complexity of equivalence test [45].

Algorithm 1 : $f \otimes g$

```
INPUT: BBDDs for $\{f, g\}$ and Boolean operation $\otimes$.
OUTPUT: BBDD top node $R$ for $f \otimes g$, edge attribute $(Attr)$ for $f \otimes g$.
  if (terminal case)||$(f == g)$ then                                  }
    $\{R, Attr\}$ = identical_terminal($\{f, g, \otimes\}$);             } $\alpha$
    return $\{R, Attr\}$;                                               }
  else if computed table has entry $\{f, g, \otimes\}$ then             }
    $\{R, Attr\}$ = lookup computed table($\{f, g, \otimes\}$);         } $\beta$
    return $\{R, Attr\}$;                                               }
  else
    $i = max_{level}\{f, g\}$;                                          }
    $\{v, w\} = \{PV, SV\}@(level = i)$;                               }
    if $(|supp(f)| == 1)||(|supp(g)| == 1)$ then                       }
      chain-transform($f, g$);                                          }
    end if                                                              }
    $\{E, E \to Attr\} = f_{v=w} \otimes g_{v=w}$;                     }
    $\otimes_D = update_{op}(\otimes, f_{v\neq w} \to Attr, g_{v\neq w} \to Attr)$;   }
    $\{D, D \to Attr\} = f_{v\neq w} \otimes_D g_{v\neq w}$;           }
    if reduction rule R4 applies then                                   }
      $R$ =BDD-node $@(level = i)$;                                     } $\gamma$
    else if $\{E, E \to Attr\} == \{D, D \to Attr\}$ then              }
      $R = E$;                                                          }
    else                                                                }
      $D \to Attr = update_{attr}(E \to Attr, D \to Attr)$;            }
      $R = lookup\_insert(i, D, D \to Attr, E)$;                       }
    end if                                                              }
    insert computed table ($\{f, g, \otimes\}, R, E \to Attr$);         }
    return $\{R, E \to Attr\}$;                                         }
  end if
```

computed table, 3) memory management to speed up computation, and 4) *chain* variable reordering to minimize the BBDD size. We discuss in details each point hereafter.

1) Unique Table: BBDD nodes must be stored in an efficient form, allowing fast lookup and insertion. Thanks to canonicity, BBDD nodes are uniquely labeled by a tuple {CVO-level, $\neq$-child, $\neq$-attribute, $=$-child}. A *unique* table maps each tuple {CVO-level, $\neq$-child, $\neq$-attribute, $=$-child} to its corresponding BBDD node via a hash-function. Hence, each BBDD node has a distinct entry in the *unique* table pointed by its hash-function, enabling a *strong canonical form* representation for BBDDs.

Exploiting this feature, equivalence test between two BBDD nodes corresponds to a simple pointer comparison. Thus, lookup and insertion operations in the *unique* table are efficient. Before a new node is added to the BBDD, a lookup checks if its corresponding tuple {CVO-level, $\neq$-child, $\neq$-attribute, $=$-child} already exists in the *unique* table and, if so, its pointed node is returned. Otherwise, a new entry for the node is created in the *unique* table.

2) Boolean Operations Between BBDDs: The capability to apply Boolean operations between two BBDDs is essential to represent and manipulate large combinatorial designs. Consequently, an efficient algorithm to compute $f \otimes g$, where $\otimes$ is any Boolean function of two operands and $\{f, g\}$ are two existing BBDDs, is the core of our manipulation package. A recursive formulation of $f \otimes g$, in terms of *biconditional* expansions, allows us to take advantage of the information stored in the existing BBDDs and hence reduce the computation complexity of the successive operation. Algorithm 1 shows the outline of the recursive implementation for $f \otimes g$. The input of the algorithm are the BBDDs for $\{f, g\}$, and the two-operand Boolean function $\otimes$ that has to be computed between them. If f and g are identical, or one of them is the sink 1 node, the operation $f \otimes g$ reaches a terminal condition. In this case, the result is retrieved from a pre-defined list of trivial operations and returned immediately (Alg.1 α). When a terminal condition is not encountered, the presence of $\{f, g, \otimes\}$ is first checked in a *computed* table, where previously performed operations are stored in case of later use. In the case of positive outcome, the result is retrieved from the *computed* table and returned immediately (Alg.1 β). Otherwise, $f \otimes g$ has to be explicitly computed (Alg.1 γ). The top level in the CVO for $f \otimes g$ is determined as $i = max_{\text{level}}\{f, g\}$ with its $\{PV_i, SV_i\}$ referred as to $\{v, w\}$, respectively, for the sake of simplicity. The root node for $f \otimes g$ is placed at such level i and its children computed recursively. Before proceeding in this way, we need to ensure that the two-variable *biconditional* expansion is well defined for both f and g, particularly if they are single variable functions. To address this case, single variable functions are prolonged down to $min_{\text{level}}\{f, g\}$ through a chain of consecutive BBDD nodes. This temporarily, and locally, may violate reduction rule **R4** to guarantee consistent $\neq$- and $=$-edges. However, rule **R4** is enforced before the end of the algorithm. Provided such handling strategy, the following recursive formulation, in terms of *biconditional* expansions, is key to efficiently compute the children for $f \otimes g$

$$f \otimes g = (v \oplus w)(f_{v\neq w} \otimes g_{v\neq w}) + (v \overline{\oplus} w)(f_{v=w} \otimes g_{v=w}). \quad (5)$$

The term $(f_{v\neq w} \otimes g_{v\neq w})$ represents the $\neq$-child for the root of $f \otimes g$ while the term $(f_{v=w} \otimes g_{v=w})$ represents the $=$-child. In $(f_{v\neq w} \otimes g_{v\neq w})$, the Boolean operation $\otimes$ needs to be updated according to the regular/complemented attributes appearing in the edges connecting to $f_{v\neq w}$ and $g_{v\neq w}$. After the recursive calls for $(f_{v=w} \otimes g_{v=w})$ and $(f_{v\neq w} \otimes g_{v\neq w})$ return their results, reduction rule **R4** is applied. Finally, the tuple {top-level, $\neq$-child, $\neq$-attribute, $=$-child} is found or added in the *unique* table and its result updated in the *computed* table.

Observe that the maximum number of recursions in (5) is determined by all possible combination of nodes between the BBDDs for f and g. Assuming constant time lookup in the *unique* and *computed* tables, it follows that the time complexity for Algorithm 1 is $O(|f| \cdot |g|)$, where $|f|$ and $|g|$ are the number of nodes of the BBDDs of f and g, respectively.

3) Memory Management: The software implementation of data-structures for *unique* and *computed* tables is essential to control the memory footprint but also to speed-up computation. In traditional logic manipulation packages [10], the *unique* and *computed* tables are implemented by a hash-table and a cache, respectively. We follow this approach in the BBDD package, but we add some specific additional technique. Informally, we minimize the access time to stored nodes and operations by dynamically changing the data-structure size and hashing function, on the basis of a $\{\text{size} \times \text{access} - \text{time}\}$ quality metric.

The core hashing-function for all BBDD tables is the Cantor pairing function between two integer numbers [44]

$$C(i, j) = 0.5 \cdot (i + j) \cdot (i + j + 1) + i \quad (6)$$

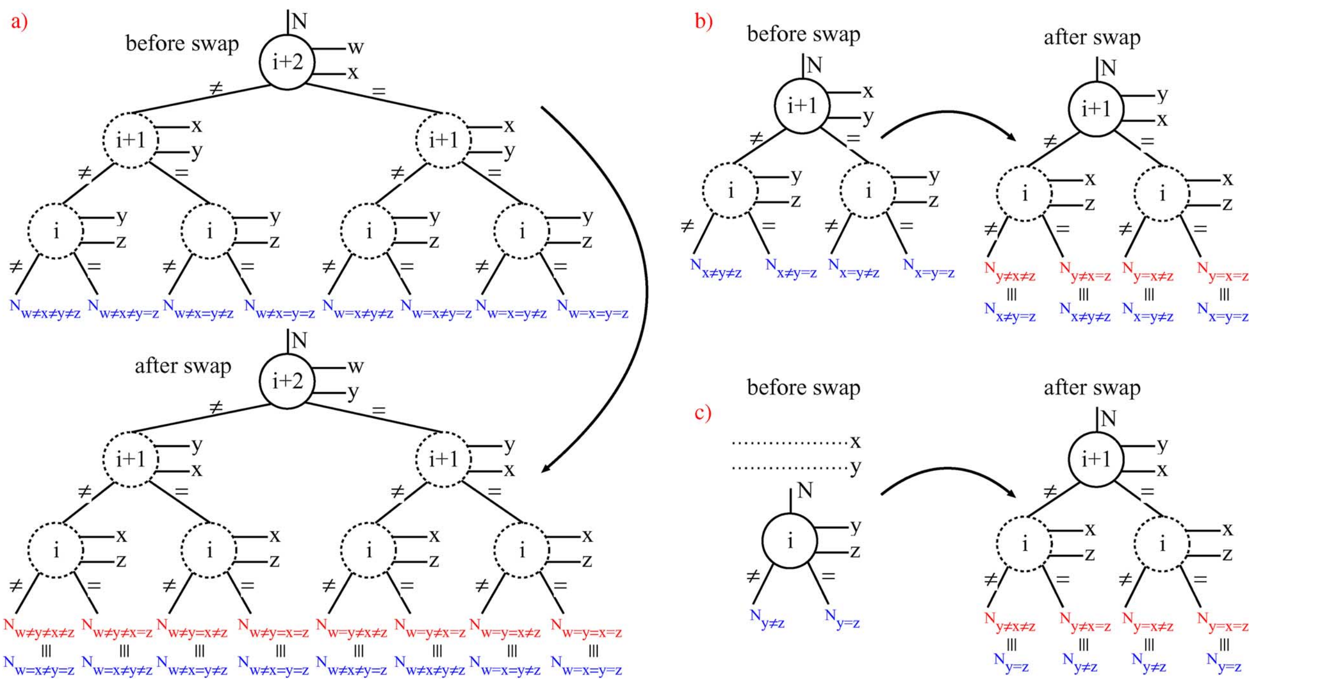


Fig. 6. Variable swap $i \rightleftharpoons i+1$ involving the CVO levels ($PV_{i+2} = w, SV_{i+2} = x$), ($PV_{i+1} = x, SV_{i+1} = y$), and ($PV_i = y, SV_i = z$). Effect on nodes at level $i+2$ (a) $i+1$ (b) and i (c).

which is a bijection from $\mathbb{N}_0 \times \mathbb{N}_0$ to $\mathbb{N}_0$ and hence a *perfect hashing function* [44]. In order to fit the memory capacity of computers, modulo operations are applied after the Cantor pairing function allowing collisions to occur. To limit the frequency of collisions, a first modulo operation is performed with a large prime number m, e.g., $m = 15485863$, for statistical reasons. Then, a final modulo operation resizes the result to the current size of the table.

Hashing functions for *unique* and *computed* tables are obtaining by nested Cantor pairings between the tuple elements with successive modulo operations.

Collisions are handled in the *unique* table by a linked list for each hash-value, while, in the *computed* table, the cache-like approach overwrites an entry when collision occurs.

Keeping low the frequency of collisions in the *unique* and *computed* tables is of paramount importance to the BBDD package performance. Traditional garbage collection and dynamic table resizing [10] are used to address this task. When the benefit deriving by these techniques is limited or not satisfactory, the hash-function is automatically modified to rearrange the elements in the table. Standard modifications of the hash-function consist of nested Cantor pairings reordering and resizing of the prime number m.

4) Chain Variable Reordering: The chain variable order for a BBDD influences the representation size and therefore its manipulation complexity. Automated chain variable reordering assists the BBDD package to boost the performance and reduce the memory requirements. Efficient reordering techniques for BDDs are based on local variable swap [46] iterated over the whole variable order, following some minimization goal. The same approach is also efficient with BBDDs. Before discussing on convenient methods to employ local swaps in a global reordering procedure, we present a new core variable swap operation adapted to the CVO of BBDDs.

BBDD CVO Swap: Variable swap in the CVO exchanges the PVs of two adjacent levels i and $i+1$ and updates the neighbor SVs accordingly. The effect on the original variable order π, from which the CVO is derived as per (4), is a direct swap of variables π_i and π_{i+1}. Note that all the nodes/functions concerned during a CVO swap are overwritten (hence maintaining the same pointer) with the new tuple generated at the end of the operation. In this way, the effect of the CVO swap remains local, as the edges of the above portion of the BBDD still point to the same logical function.

A variable swap $i \rightleftharpoons i+1$ involves three CVO levels ($PV_{i+2} = w$, $SV_{i+2} = x$), ($PV_{i+1} = x$, $SV_{i+1} = y$) and ($PV_i = y$, $SV_i = z$). The level $i+2$ must be considered as it contains in SV the variable x, which is the PV swapped at level $i+1$. If no level $i+2$ exists ($i+1$ is the top level), the related operations are simply skipped. In the most general case, each node at level $i+2$, $i+1$ and i has 8, 4, and 2 possible children on the portion of BBDD below level i. Some of them may be identical, following to reduction rules **R1–4**, or complemented, deriving by the $\neq$-edges attributes in their path. Fig. 6 depicts the different cases for a general node N located at level $i+2$, $i+1$ or i, with all their possible children. After the swap $i \rightleftharpoons i+1$, the order of comparisons $w \star x \star y \star z$ is changed to $w \star y \star x \star z$ and the children of N must be rearranged consequently ($\star \in \{=, \neq\}$). Using the *transitive property of equality and congruence* in the binary domain, it is possible to remap $w \star x \star y \star z$ into $w \star y \star x \star z$ as

$$\begin{aligned} &\star \in \{=, \neq\}, \quad \bar{\star} : \{=, \neq\} \rightarrow \{\neq, =\} \\ &(w \star_{i+2} x = y \star_i z) \rightarrow (w \star_{i+2} y = x \star_i z) \\ &(w \star_{i+2} x \neq y \star_i z) \rightarrow (w \bar{\star}_{i+2} y \neq x \bar{\star}_i z). \end{aligned} \tag{7}$$

Following remapping rules in (7), the children for N can be repositioned coherently with the variable swap. In Fig. 6, the actual children rearrangement after variable swap is shown. In a bottom-up approach, it is possible to assemble back the swapped levels, while intrinsically respecting reduction rules **R1–4**, thanks to the *unique* table *strong canonical form*.

BBDD Reordering Based on CVO Swap: Using the previously introduced CVO swap theory, global BBDD reordering can be carried out in different fashions. A popular approach for BDDs is the sifting algorithm presented in [46]. As its formulation is quite general, it happens to be advantageous also for BBDDs. Its BBDD implementation works as follows: Let n be the number of variables in the initial order π. Each variable π_i is considered in succession and the influence of the other variables is locally neglected. Swap operations are performed to move π_i in all n potential positions in the CVO. The best BBDD size encountered is remembered and its π_i position in the CVO is restored at the end of the variable processing. This procedure is repeated for all variables. It follows that BBDD sifting requires $O(n^2)$ swap operations.

Evolutions of the original sifting algorithm range between grouping of variables [48], simulated annealing techniques [49], genetic algorithms [50] and others. All of them are in principle applicable to BBDD reordering. In any of its flavors, BBDD reordering can be applied to a previously built BBDD or dynamically during construction. Usually, the latter strategy produces better results as it permits a tighter control of the BBDD size.

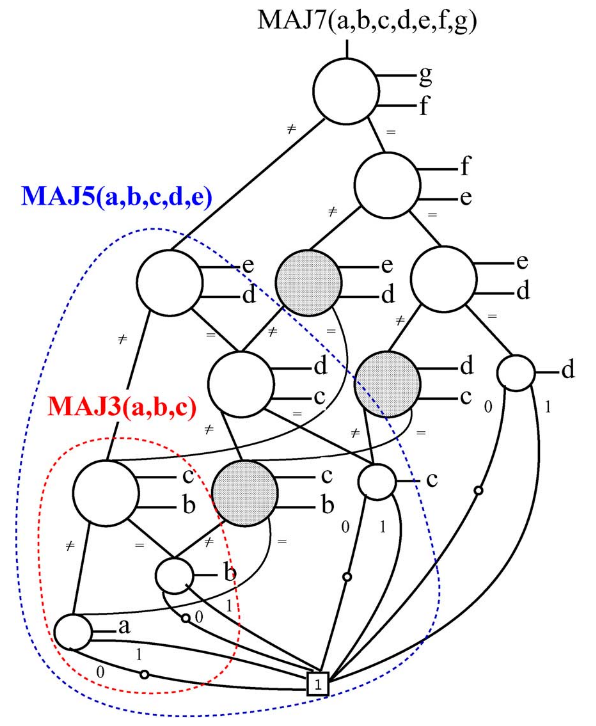


Fig. 7. BBDD for the 7-input majority function. The inclusion of MAJ_5 and MAJ_3 functions is illustrated. Grey nodes are nodes with inverted children due to n to $n-2$ majority reduction.

IV. BBDD Representation: Theoretical and Experimental Results

In this section, we first show some theoretical properties for BBDDs, regarding the representation of majority and adder functions. Then, we present experimental results for BBDD representation of MCNC and HDL benchmarks, accomplished using the introduced BBDD software package.

A. Theoretical Results

Majority and adder functions are essential in many digital designs. Consequently, their efficient representation has been widely studied with state-of-art decision diagrams. We study hereafter the size for majority and adders with BBDDs and we compare these results with their known BDD size.

1) Majority Function: In Boolean logic, the majority function has an odd number n of inputs and an unique output. The output assumes the most frequent Boolean value among the inputs. With BBDDs, the MAJ_n function has a hierarchical structure. In Fig. 7, the BBDD for MAJ_7 is depicted, highlighting the hierarchical inclusion of MAJ_5 and MAJ_3. The key concepts enabling this hierarchical structure are as follows:

M1) $\neq$-edges reduce MAJ_n to MAJ_{n-2}: when two inputs assume opposite Boolean values they do not affect the majority voting decision;

M2) $\lceil n/2 \rceil$ consecutive $=$-edges fully-determine MAJ_n voting decision: if $\lceil n/2 \rceil$ over n (odd) inputs have the same Boolean value, then this is the majority voting decision value.

The M1 condition traduces in connecting $\neq$-edges to the BBDD structure for MAJ_{n-2}, or to local duplicated nodes with inverted children (see grey nodes in Fig. 7).

The M2 condition implies $\lceil n/2 \rceil$ consecutive BBDD nodes cascaded through $=$-edges.

Note that the variable order is not affecting the BBDD structure for a MAJ function as its behavior is invariant under input permutations [21].

Theorem 3: A BBDD for the majority function of n (odd) variables has $(1/4)(n^2+7)$ nodes.

Proof: The M2 condition for MAJ_n requires $n-1$ nodes while the M1 condition needs the BBDD structure for MAJ_{n-2}. Consequently, the number of BBDD nodes is $|MAJ_n| = |MAJ_{n-2}| + n - 1$ with $|MAJ_3| = 4$ (including the sink node) as boundary condition. This is a nonhomogeneous recurrence relation. Linear algebra methods [26] can solve such recurrence equation. The closed-form solution is $|MAJ_n| = (1/4)(n^2+7)$. *Q.E.D.*

Note that with standard BDDs, the number of nodes is $|MAJ_n| = \lceil n/2 \rceil (n - \lceil n/2 \rceil + 1) + 1$ [21]. It follows that BBDDs are always more compact than BDDs for majority, e.g., the BBDD for the 89-inputs majority function has 1982 nodes while its BDD counterpart has 2026 nodes. These values, and the law of *Theorem 3*, have been verified experimentally.

2) Adder Function: In Boolean logic, a n-bit adder is a function computing the addition of two n-bit binary numbers. In many logic circuits, a n-bit adder is represented as n cascaded 1-bit adders. A 1-bit binary adder, commonly called full adder, is a 3-input 2-output Boolean function described as $Sum = a \oplus b \oplus cin$ and $Cout = MAJ(a, b, cin)$. The BBDD for the full adder is depicted by Fig. 8.

With BBDDs, the 1-bit adder cascading concept can be naturally extended and leads to a compact representation for a general n-bit adder.

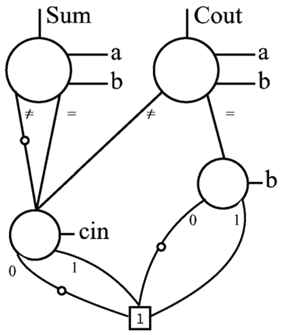


Fig. 8. Full adder function with BBDDs, variable order $\pi = (a, b, cin)$.

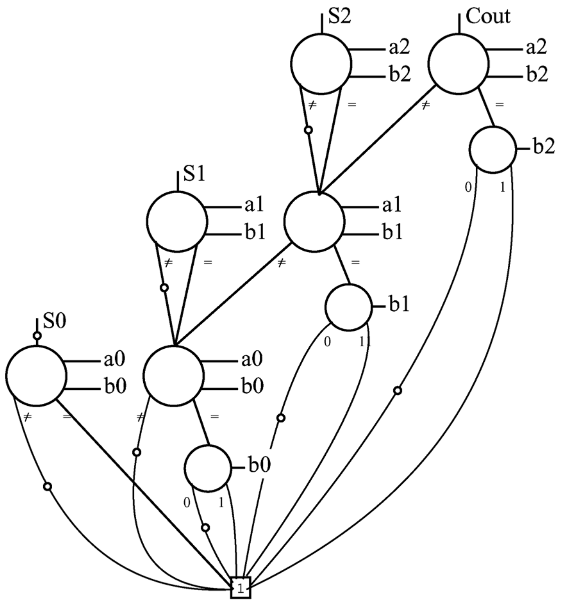


Fig. 9. BBDD for the 3-bit binary adder function, variable order $\pi = (a_2, b_2, a_1, b_1, a_0, b_0)$.

In Fig. 9, the BBDD of a 3-bit binary adder $(a + b)$, with $a = (a_2, a_1, a_0)$ and $b = (b_2, b_1, b_0)$, employing variable order $\pi = (a_2, b_2, a_1, b_1, a_0, b_0)$, is shown.

3) Theorem 4: A BBDD for the n-bit binary adder function has $3n + 1$ nodes when the variable order $\pi = (a_{n-1}, b_{n-1}, a_{n-2}, b_{n-2}, \ldots, a_0, b_0)$ is imposed.

Proof: The proof follows by induction over the number of bit n and expanding the structure in Fig. 9. *Q.E.D.*

Note that the BDD counterpart for n-bit adders (best) ordered with $\pi = (a_{n-1}, b_{n-1}, a_{n-2}, b_{n-2}, \ldots, a_0, b_0)$ has $5n+2$ nodes [21]. For n-bit adders, BBDDs save about 40% of the nodes compared to BDDs. These results, and the law of *Theorem 4*, have been verified experimentally.

B. Experimental Results

The manipulation and contruction techniques described in Section III-E are implemented in a BBDD software package [18] using C programming language. Such package currently counts about 8k lines of code. For the sake of comparison, we consider CUDD [10] (manipulation package for BDDs) and *puma* [11] (manipulation package for KFDDs). We examine three categories of benchmarks: 1) MCNC suite, 2) portion of Open Cores designs, and 3) arithmetic HDL benchmarks. CUDD and *puma* packages read BLIF format files while the BBDD package reads a Verilog flattened onto primitive Boolean operations. The appropriate format conversion is accomplished using ABC synthesis tool [51]. For all packages, dynamic reordering during construction is enabled and based on the original sifting algorithm [46]. For *puma*, also the choice of the most convenient decomposition type is enabled. The machine running the experiments is a Xeon X5650 24-GB RAM machine. We verified with *Synopsys Formality* commercial tool the correctness of the BBDD output, which is also in Verilog format.

Table I shows the experimental results for the three packages. Note that the sizes and runtime reported derives from heuristic techniques, so better results may exist. Therefore, the following values provide an indication about the practical efficiency of each decision diagram but do not give the means to determine if any of them is globally superior to the others.

MCNC Benchmarks: For large MCNC benchmarks, we report that BBDDs have an average size 33.5% and 12.2% smaller than BDDs and KFDDs, respectively. Regarding the runtime, the BBDD is 1.4× and 1.5× faster than CUDD and *puma*, respectively. By handling two variables per time, BBDDs unlock new compact representation opportunities, not apparent with BDDs or KFDDs. Such size reduction is responsible for the average runtime reduction. However, the general runtime for a decision diagram package is also dependent on the implementation maturity of the techniques supporting the construction. For this reason, there are benchmarks like C5315 where even if the final BBDD size is smaller than BDDs and KFDDs, its runtime is longer as compared to CUDD and *puma*, which have been perfected and highly optimized during years. Further runtime improvements are expected with next releases of the BBDD package.

Open Cores Benchmarks: Combinational portions of Open Cores circuits are considered as representative for contemporary real-life designs. In this context, BBDDs have, on average, 30.9% and 20.9% fewer nodes than BDDs and KFDDs, respectively. The average runtime is roughly the same for all packages. It appears that such benchmarks are easier than MCNC, having fairly small sizes and negligible runtime. To test the behavior of the packages at their limit we consider a separate class of hard circuits.

Arithmetic HDL Benchmarks: Traditional decision diagrams are known to face efficiency issues in the representation of arithmetic circuits, e.g., multipliers. We evaluate the behavior of the BBDD package in contrast to CUDD and *puma* for some of these hard benchmarks, i.e., a 10 × 10-bit multiplier, a 32-bit width square root unit, a 20-bit hyperbola and a 16-bit divisor. On average, BBDDs are about 5× smaller than BDDs and KFDDs for such benchmarks. Moreover, the runtime of the BBDD package is 4.4× faster than CUDD and *puma*. These results highlight that BBDDs have an enhanced capability to deal with arithmetic intensive circuits, thanks to the expressive power of the *biconditional expansion*. A theoretical study to determine the asymptotic bounds of BBDDs for these functions is ongoing.

TABLE I
EXPERIMENTAL RESULTS FOR DD CONSTRUCTION USING BBDDs, BDDs, AND KFDDs

Benchmarks	Inputs	Outputs	Wires	BBDD		CUDD (BDD)		*puma* (KFDD)	
				Node Count	Runtime (s)	Node Count	Runtime (s)	Node Count	Runtime (s)
MCNC Benchmarks									
C1355	41	32	212	27701	1.22	68427	2.70	49785	8.32
C2670	233	64	825	29833	0.99	30329	0.88	36154	0.10
C499	41	32	656	32305	5.07	122019	5.60	49785	18.41
C1908	33	25	279	22410	0.53	18274	0.73	12716	0.08
C5315	178	123	1689	22263	1.03	42151	0.31	26658	0.57
C880	60	26	363	29362	0.40	22077	0.72	7567	0.03
C3540	50	22	1259	99471	8.93	93762	15.53	111324	0.73
C17	5	2	8	12	0.01	14	0.01	9	0.01
misex3	14	14	3268	766	0.08	870	0.02	1853	0.10
too_large	38	3	5398	1234	0.17	1318	0.26	6076	0.45
my_adder	33	17	98	166	0.09	620	0.11	456	0.21
Average	66.0	33.7	1277.7	**24138.4**	**1.7**	36351.0	2.4	27489.3	2.6
Combinational Portions of Open Cores Benchmarks									
custom-alu	37	17	193	2327	0.06	2442	0.01	2149	0.02
sin	35	64	2745	873	0.13	3771	0.12	1013	0.15
cosin	35	64	2652	851	0.10	3271	0.13	862	0.16
logsig	32	30	1317	1055	0.04	1571	0.09	1109	0.20
min-max	42	23	194	2658	0.40	2834	0.67	26736	0.76
h264-LUT	10	11	690	499	0.02	702	0.02	436	0.01
31-bit voter	31	1	367	242	0.01	257	0.01	256	0.01
ternary-adder	96	32	1064	366	0.32	8389	0.20	8389	0.20
max-weight	32	8	234	7505	0.15	7659	0.35	7610	0.55
cmul8	16	16	693	14374	0.55	12038	0.41	10979	0.21
fpu-norm	16	16	671	4209	0.12	7716	0.37	8608	0.32
Average	34.2	25.6	983.6	**3178.1**	**0.2**	4604.5	0.2	4022.2	0.2
Hard Arithmetic Benchmarks									
sqrt32	32	16	1248	223340	1145.53	11098772	3656.18	9256912	2548.92
hyperbola20	20	25	12802	126412	281.45	4522101	1805.20	4381924	2522.01
mult10x10	20	20	1123	123768	24.77	91192	15.74	91941	0.95
div16	32	32	3466	3675419	1428.87	7051263	7534.78	7842802	1583.22
Average	26.0	23.2	4659.7	**1.0e06**	**720.1**	5.6e06	3253.0	5.4e06	1671.3

V. BBDD-Based Synthesis and Verification

This section showcases the interest of BBDDs in the automated design of digital circuits, for both standard CMOS and emerging technologies. We consider the application of BBDDs in logic synthesis and formal equivalence checking tasks for a real-life telecommunication circuit.

A. Logic Synthesis

The efficiency of logic synthesis is key to realize profitable commercial circuits. In most designs, critical components are arithmetic circuits for which traditional synthesis techniques do not produce highly optimized results. Indeed, arithmetic functions are not natively supported by conventional logic representation forms. Differently, BBDD nodes inherently act as two-variable comparators, a basis function for arithmetic operations. This feature opens the opportunity to restructure arithmetic logic via BBDD representation.

We employ the BBDD package as frontend to a commercial synthesis tool. The BBDD restructuring is kept if it reduces the original representation complexity, i.e., the number of nodes and the number of logic levels. Starting from a simpler description, the synthesizer can reach higher levels of quality in the final circuit.

B. Formal Equivalence Checking

Formal equivalence checking task determines if two versions of a design are functionally equivalent. For combinational portions of a design, such task can be accomplished using canonical representation forms, e.g., decision diagrams. BBDDs can speed up the verification of arithmetic intensive designs, as compared to traditional methods, thanks to their enhanced compactness.

We employ BBDDs to check the correctness of logic optimization methods by comparing an initial design with its optimized version.

C. Case Study: Design of an Iterative Product Code Decoder

To assess the effectiveness of BBDDs for the aforementioned applications, we design a real-life telecommunication circuit. We consider the Iterative Decoder for Product Code from Open Cores. The synthesis task is carried out using BBDD restructuring of arithmetic operations for each module, kept only if advantageous. The formal equivalence checking task is also carried out with BBDDs with the aim to speed-up the verification process. For the sake of comparison, we synthesized the same design without BBDD restructuring and we also verified it with BDDs in place of BBDDs.

As mentioned in Section I, one compelling reason to study BBDDs is to provide a natural design abstraction for emerging

TABLE II
EXPERIMENTAL RESULTS FOR BBDD-BASED DESIGN SYNTHESIS AND VERIFICATION

Case Study for Design & Verification: *Iterative Product Decoder*								
Optimization via BBDD-rewriting								
Logic Circuits	Type	I/O		BBDD-rewriting		Original		Gain
		Inputs	Outputs	Nodes	Levels	Nodes	Levels	
adder08_bit.vhd	Comb.	16	9	16	8	78	19	✓
bit_comparator.vhd	Comb.	5	3	3	1	8	3	✓
comparator_7bits.vhd	Comb.	14	3	21	7	58	14	✓
fulladder.vhd	Comb.	3	2	2	1	9	4	✓
ext_val.vhd	Comb.	16	8	674	16	173	29	✗
twos_c_8bit.vhd	Comb.	8	8	20	8	29	8	✓
ser2par8bit.vhd	Seq.	11	64	-	-	-	-	-
product_code.vhd	Top	10	4	-	-	-	-	-
Synthesis in 22-nm CMOS Technology – Clock Period Constraint: 0.6 ns (1.66 GHz)								
				BBDD + Synthesis Tool		Synthesis Tool		
		Inputs	Outputs	Area (μm^2)	Slack (ns)	Area (μm^2)	Slack (ns)	Constraint met
product_code.vhd	Top	10	4	1291.03	0.00	1177.26	-0.12	✓
Synthesis in 22-nm DG-SiNWFET Technology – Clock Period Constraint: 0.5 ns (2 GHz)								
				BBDD + Synthesis Tool		Synthesis Tool		
		Inputs	Outputs	Area (μm^2)	Slack (ns)	Area (μm^2)	Slack (ns)	Constraint met
product_code.vhd	Top	10	4	1731.31	0.00	1673.78	-0.16	✓
Formal Equivalence Checking								
				BBDD		CUDD (BDD)		
		Inputs	Outputs	Nodes	Runtime	Nodes	Runtime	Verification
product_code.vhd	Comb.	130	68	241530	185.11	227416	208.80	✓

technologies where the circuit primitive is a comparator, whose functionality is natively modeled by the *biconditional expansion*. For this reason, we target two different technologies: 1) a conventional CMOS 22-nm technology and 2) an emerging controllable-polarity DG-SiNWFET 22-nm technology. A specific discussion for each technology is provided in the following subsections while general observations on the arithmetic restructuring are given hereafter.

The Iterative Decoder for Product Code consists of eight main modules, among them two are sequential, one is the top entity, and six are potentially arithmetic intensive. We process the six arithmetic intensive modules and we keep the restructured circuits if their size and depth are decreased. For the sake of clarity, we show an example of restructuring for the circuit *bit_comparator*. Fig. 10(a) depicts the logic network before processing and Fig. 10(b) illustrates the equivalent circuit after BBDD-restructuring. BDD nodes due to rule **R4** are omitted for simplicity. An advantage in both size and depth is reported. Table II shows the remaining results. BBDD-restructuring is favorable for all modules except *ext_val* that instead is more compact in its original version. The best obtained descriptions are finally given in input to the synthesis tool.

1) CMOS Technology: For CMOS technology, the design requirement is a clock period of 0.6 ns, hence a clock frequency of 1.66 GHz. The standard synthesis approach generates a negative slack of 0.12 ns, failing to meet the timing constraint. With BBDD-restructuring, instead, the timing constraint is met (slack of 0.00 ns), which corresponds to a critical path speedup of 1.2×. However, BBDD-restructuring induces a moderate area penalty of 9.6%.

2) Emerging DG-SiNWFET Technology: The controllable-polarity DG-SiNWFET technology features much more compact arithmetic (XOR, MAJ, etc.) gates than in CMOS,

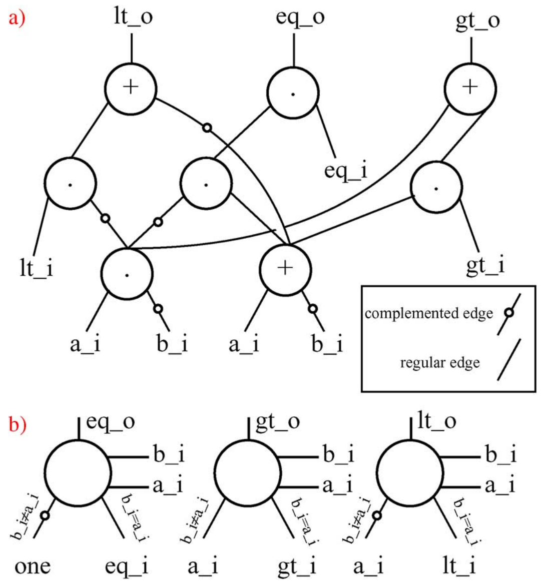


Fig. 10. Representations for the bit_comparator circuit in [47] (inverters are bubbles in edges). a) original circuit b) BBDD rewriting, BDD nodes are omitted for the sake of illustration.

enabling faster and smaller implementation opportunities. For this reason, we set a tighter clock constraint than in CMOS, i.e., 0.5 ns corresponding to a clock frequency of 2 GHz. Direct synthesis of the design fails to reach such clock period with 0.16 ns of negative slack. With BBDD-restructuring, the desired clock period is instead reachable. For DG-SiNWFET technology, the benefit deriving from the use of BBDDs is even higher than in CMOS technology. Indeed, here BBDD-restructuring is capable to bridge a negative timing gap equivalent to 32% of the overall desired clock period. For CMOS instead the same gap is just 20%. This result confirms that BBDDs can further harness the expressive power of emerging technologies as compared to traditional synthesis techniques alone. Furthermore, the area penalty relative to BBDD-restructuring for DG-SiNWFET technology is decreased to only 3.3%.

3) Combinatorial Verification: The verification of the combinatorial portions of the Iterative Decoder for Product Code design took 185.11 s with BBDDs and 208.80 s and with traditional BDDs. The size of the two representations is roughly the same, thus the 12% speed-up with BBDDs is accountable to the different growth profile of the decision diagrams during construction.

VI. Conclusion

Following the trend to handle ever-larger designs, and in light of the rise of emerging technologies based on new Boolean primitives, the study of innovative logic representation forms is of paramount importance. In this paper, we proposed Biconditional BDDs, a new canonical representation form driven by the *biconditional expansion*. BBDDs implement an equality/inequality switching paradigm that enhances the expressive power of decision diagrams. Moreover, BBDDs natively models the functionality of emerging technologies where the circuit primitive is a comparator, rather than a simple switch. Employed in electronic design automation, BBDDs 1) push further the efficiency of traditional decision diagrams and 2) unlock the potential of promising post-CMOS devices. Experimental results over different benchmark suites, demonstrated that BBDDs are frequently more compact than other decision diagrams, from 1.1× to 5×, and are also built faster, from 1.4× to 4.4×. Considering the synthesis of a telecommunication circuit, BBDDs advantageously restructure critical arithmetic operations. With a 22-nm CMOS technology, BBDD-restructuring shorten the critical path by 20%. With an emerging 22-nm controllable-polarity DG-SiNWFET technology, BBDD-restructuring shrinks more the critical path by 32%, thanks to the natural correspondence between device operation and logic representation. The formal verification of the optimized design is also accomplished using BBDDs in about 3 min, which is about 12% faster than with standard BDDs.

References

[1] C. Y. Lee, "Representation of switching circuits by binary-decision programs," *Bell Syst. Tech. J.*, vol. 38, no. 4, pp. 985–999, 1959.
[2] S. B. Akers, "Binary decision diagrams," *IEEE Trans. Comput.*, vol. 100, no. 6, pp. 509–516, 1978.
[3] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. 100, no. 8, pp. 677–691, Aug. 1986.
[4] C. Yang and M. Ciesielski, "BDS: A BDD-based logic optimization system," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 21, no. 7, pp. 866–876, Jul. 2002.
[5] S. Malik *et al.*, "Logic verification using binary decision diagrams in a logic synthesis environment," in *Proc. IEEE Int. Conf. Comput. Aided Design*, 1988, pp. 6–9.
[6] M. S. Abadir *et al.*, "Functional test generation for digital circuits using binary decision diagrams," *IEEE Trans. Comput.*, vol. 100, no. 4, pp. 375–379, Apr. 1986.
[7] C. Scholl, R. Drechsler, and B. Becker, "Functional simulation using binary decision diagrams," in *Proc. IEEE Int. Conf. Comput. Aided Design*, 1997, pp. 8–12.
[8] U. Kebschull, W. Rosenstiel, and E. Schubert, "Multilevel logic synthesis based on functional decision diagrams," in *Proc. IEEE Eur. Conf. Design Automat.*, 1992, pp. 43–47.
[9] R. Drechsler *et al.*, "Ordered Kronecker functional decision diagrams-a data structure for representation and manipulation of Boolean functions," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 17, no. 10, pp. 965–973, Oct. 1998.
[10] CUDD: CU Decision Diagram Package Release 2.5.0 [Online]. Available: http://vlsi.colorado.edu/fabio/CUDD/cuddIntro.html
[11] Decision Diagram-Package PUMA [Online]. Available: http://ira.informatik.uni-freiburg.de/software/puma/pumamain.html
[12] M. De Marchi *et al.*, "Polarity control in double-gate, gate-all-around vertically stacked silicon Nanowire FETs," in *IEEE Electron Devices Meet.*, 2012, p. 8.
[13] Y. Lin *et al.*, "High-performance carbon nanotube field-effect transistor with tunable polarities," *IEEE Trans. Nanotechnol.*, vol. 4, no. 5, pp. 481–489, Sep. 2005.
[14] N. Harada *et al.*, "A polarity-controllable graphene inverter," *Appl. Phys. Lett.*, vol. 96, no. 1, p. 012102, 2010.
[15] D. Lee *et al.*, "Combinational logic design using six-terminal NEM relays," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 32, no. 5, pp. 653–666, May 2013.
[16] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Biconditional BDD: A novel canonical representation form targeting the synthesis of XOR-rich circuits," in *Design, Automat. Test Eur.*, 2013, pp. 1014–1017.
[17] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "An efficient manipulation package for biconditional binary decision diagrams," in *Design, Automat. Test Eur.*, 2014, pp. 296–301.
[18] BBDD Package [Online]. Available: http://lsi.epfl.ch/BBDD
[19] K. Levitz, *Logic and Boolean Algebra*. New York: Barrons, Feb. 1979.
[20] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.
[21] I. Wegener, *Branching Programs and Binary Decision Diagrams: Theory and Applications*. Philadelphia, PA: SIAM, 2000, vol. 4.
[22] M. Kreuzer and L. Robbiano, *Computational Commutative Algebra*. Berlin, Germany: Springer, 2005, vol. 1.
[23] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication," *IEEE Trans. Comput.*, vol. 40, no. 2, pp. 205–213, Feb. 1991.
[24] J. Gergov and C. Meinel, "Mod-2-OBDDs A data structure that generalizes EXOR-sum-of-products and ordered binary decision diagrams," *Formal Methods Syst. Design*, vol. 8, no. 3, pp. 273–282, 1996.
[25] B. Bollig, "Improving the variable ordering of OBDDs is NP-complete," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 993–1002, Sep. 1996.
[26] T. Koshy, *Discrete Mathematics With Applications*. New York: Academic, 2004.
[27] T. S. Czajkowski and S. D. Brown, "Functionally linear decomposition and synthesis of logic circuits for FPGAs," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 27, no. 12, pp. 2236–2249, Dec. 2008.
[28] J. F. Groote and J. Van de Pol, "Logic for programming and automated reasoning," in *Equational Binary Decision Diagrams*. Berlin, Germany: Springer, 2000.
[29] S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," in *Proc. IEEE Conf. Design Automat.*, 1993, pp. 272–277.
[30] C. Meinel, F. Somenzi, and T. Theobald, "Linear sifting of decision diagrams," in *Proc. IEEE Conf. Design Automat.*, 1997, pp. 202–207.
[31] W. Gunther and R. Drechsler, "BDD minimization by linear transformations," in *Adv. Comput. Syst.*, 1998, pp. 525–532.
[32] M. Fujita, Y. Kukimoto, and R. Brayton, "BDD minimization by truth table permutation," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1996, pp. 596–599.
[33] E. M. Clarke, M. Fujita, and X. Zhao, "Hybrid decision diagrams," in *Proc. IEEE Int. Conf. Comput. Aid. Design*, 1995, pp. 159–163.
[34] E. I. Goldberg, Y. Kukimoto, and R. K. Brayton, "Canonical TBDD's and their application to combinational verification," in *ACM/IEEE Int. Workshop Logic Synthesis*, Tahoe City, CA, 1997.
[35] U. Kebschull and W. Rosenstiel, "Efficient graph-based computation and manipulation of functional decision diagrams," in *Proc. IEEE Eur. Conf. Design Automat.*, 1993, pp. 278–282.
[36] J. E. Rice, "Making a choice between BDDs and FDDs," in *ACM/IEEE Intl. Workshop Logic Synthesis*, Lake Arrowhead, CA, 2005.

[37] R. Drechsler, “Ordered Kronecker functional decision diagrams und ihre Anwndung,” in *Reihe Informatik*. Germany: Modell-Verlag, 1996.

[38] S. Grygiel and M. A. Perkowski, “New compact representation of multiple-valued functions, relations, non-deterministic state machines,” in *Proc. IEEE Conf. Comput. Design*, 1998, pp. 168–174.

[39] A. Srinivasan, T. Kam, S. Malik, and R. Brayton, “Algorithms for discrete function manipulation,” in *IEEE Int. Conf. Comput. Aided Design*, Nov. 1990, pp. 92–95.

[40] S. Minato *et al.*, “Shared BDD with attributed edges for efficient boolean function manipulation,” in *Proc. IEEE Conf. Design Automat.*, 1990, pp. 52–57.

[41] B. Becker and R. Drechsler, “How many decomposition types do we need?,” in *Proc. IEEE Eur. Conf. Design Automat.*, 1995, pp. 438–442.

[42] B. Becker, R. Drechsler, and M. Theobald, “On the expressive power of OKFDDs,” *Formal Methods Syst. Design*, vol. 11, no. 1, pp. 5–21, 1997.

[43] R. Drechsler and B. Becker, *Binary Decision Diagrams: Theory and Implementation*. Norwell, MA: Kluwer, 1998.

[44] P. Tarau, Pairing functions, Boolean evaluation and binary decision diagrams arXiv preprint arXiv:0808.0555, 2008.

[45] K. S. Brace, R. L. Rudell, and R. E. Bryant, “Efficient implementation of a BDD package,” in *Proc. IEEE Conf. Design Automat.*, 1990, pp. 40–45.

[46] R. Rudell, “Dynamic variable ordering for ordered binary decision diagrams,” in *Proc. IEEE Int. Conf. Comput. Aided Design*, 1993, pp. 42–47.

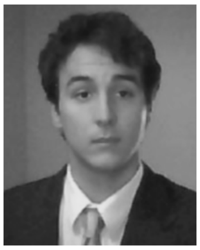
[47] An iterative decoder for product code—From open cores [Online]. Available: http://opencores.org/project,product_code_iterative_decoder

[48] S. Panda and F. Somenzi, “Who are the variables in your neighborhood,” in *Proc. IEEE Int. Conf. Comput. Aided Design*, 1995, pp. 74–77.

[49] B. Bollig *et al.*, “Simulated annealing to improve variable orderings for OBDDs,” in *ACM/IEEE Int. Workshop Logic Synth.*, Tahoe City, CA, 1995.

[50] R. Drechsler *et al.*, “A genetic algorithm for variable ordering of OBDDs,” in *ACM/IEEE Int. Workshop Logic Synthesis*, Tahoe City, CA, 1995.

[51] ABC Synthesis Tool [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/



Luca Amarú (S’13) received the B.S. degree in electronic engineering from Politecnico di Torino, Torino, Italy, in 2009. In 2011, he received the joint M.S. degree in electronic engineering from Politecnico di Torino and Politecnico di Milano, Milan, Italy. He is currently working toward the Ph.D. degree in computer and communication sciences at École Polytechnique Fédérale de Lausanne (EPFL), Integrated Systems Laboratory, Lausanne, Switzerland.

His research interests include electronic design automation, beyond CMOS technologies, information and communication theory, and computer science in general.

Mr. Amarú received the Best Presentation Award at FETCH 2013 conference and a Best Paper Award Nomination at ASP-DAC 2013 conference.

Pierre-Emmanuel Gaillardon (S’10–M’11) received the electrical engineer degree from CPE Lyon, Villeurbanne, France, in 2008, the M.Sc. degree from INSA Lyon, Villeurbanne cedex, France, in 2008, and the Ph.D. degree in electrical engineering from the University of Lyon, Lyon, France, in 2011.

He works as a Research Associate at the Laboratory of Integrated Systems (LSI), École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. Previously, he was a research assistant at CEA-LETI, Grenoble, France. Involved in the Nanosys project, his research activities and interests are currently focused on emerging nanoscale devices and their use in digital circuits and architectures.

Dr. Gaillardon is recipient of the C-Innov 2011 best thesis award and the Nanoarch 2012 best paper award. He has been serving as TPC member for Nanoarch 2012, 2013 conferences and is reviewer for several journals (AIP APL, IEEE TNANO, IEEE TVLSI, ACM JETC), conferences (ICECS 2012, ISCAS 2013) and funding agencies (ANR, Chairs of Excellence program of Nanosciences Foundation).

Giovanni De Micheli (F’94) received the nuclear Eng. degree from Politecnico di Milano, Milan, Italy, in 1979, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, CA, USA, in 1980 and 1983, respectively.

He is a Professor and the Director of the Institute of Electrical Engineering and of the Integrated Systems Centre at EPF Lausanne, Lausanne, Switzerland. He is a Program Leader of the Nano-Tera.ch program. His research interests include several aspects of design technologies for integrated circuits and systems, such as synthesis for emerging technologies, networks on chips and 3-D integration. He is also interested in heterogeneous platform design including electrical components and biosensors, as well as in data processing of biomedical information. He is author of “Synthesis and Optimization of Digital Circuits” (McGraw-Hill, 1994), co-author and/or co-editor of eight other books and of more than 500 technical articles. His citation h-index is 84 according to Google Scholar.

Dr. De Micheli is a Fellow of ACM and a member of the Academia Europaea. He is member of the Scientific Advisory Board of IMEC and STMicroelectronics. He is the recipient of the 2012 IEEE/CAS Mac Van Valkenburg award for contributions to theory, practice, and experimentation in design methods and tools and of the 2003 IEEE Emanuel Piore Award for contributions to computer-aided synthesis of digital systems. He received also the Golden Jubilee Medal for outstanding contributions to the IEEE Circuits and Systems Society in 2000, the D. Pederson Award for the best paper on the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN (CAD)/INTEGRATED CIRCUITS AND SYSTEMS (ICAS), in 1987, and several Best Paper Awards, including Design Automation Conference (DAC) in 1983 and 1993, Design, Automation, and Test in Europe (DATE) in 2005, and Nanoscale Architectures in 2010 and 2012. He has been serving IEEE in several capacities, namely: Division 1 Director during 2008–2009, the Co-Founder and President Elect of the IEEE Council on Electronic Design Automation during 2005–2007, the President of the IEEE CAS Society in 2003, an Editor-in-Chief of the IEEE TRANSACTIONS ON CAD/ICAS during 1987–2001. He has been the Chair of several conferences, including DATE in 2010, Public Health Conferences in 2006, IEEE International Conference on Very Large Scale Integration in 2006, DAC in 2000, and IEEE International Conference on Computer Design in 1989.