Chapter 2
Biconditional Logic

In this chapter, 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 chapter, a solid ground for BBDDs by studying their underlying theory and manipulation properties. Thanks to an efficient BBDD software package implementation, we validate (i) runtime reduction in traditional decision diagrams applications with respect to other DDs, and (ii) improved synthesis of circuits in standard and emerging technologies.

2.1 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

© Springer International Publishing Switzerland 2017

L.G. Amaru, *New Data Structures and Algorithms for Logic Synthesis and Verification*, DOI 10.1007/978-3-319-43174-1_2

decision diagrams [24]. 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 [16].

In this chapter, we present *Biconditional Binary Decision Diagrams* (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.

We validate the benefits deriving from the use of BBDDs in EDA tasks through an efficient software manipulation package, available online [19]. 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 combinational verification of the optimized design is also sped 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 *Double-Gate* (DG) *Silicon NanoWires Field Effect Transistors* (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 chapter is organized as follows. Section 2.2 first provides a background on BDDs and then discusses the motivations for the study of BBDDs. In Sect. 2.3, the formal theory for BBDDs is introduced, together with efficient manipulation algorithms. Section 2.4 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 2.5 presents the application of BBDDs to circuit synthesis and verification in traditional technology. Section 2.6 presents the application of BBDDs to circuit synthesis in emerging technologies. This chapter is concluded in Sect. 2.7.

2.2 Background and Motivation

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

2.2.1 Binary Decision Diagrams

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 [1, 21]) and we review hereafter the basic terminology used in the rest of the paper.

2.2.1.1 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. 2.1a) in a BDD is labeled by a Boolean variable v and has two out-edges labeled 0 and 1.

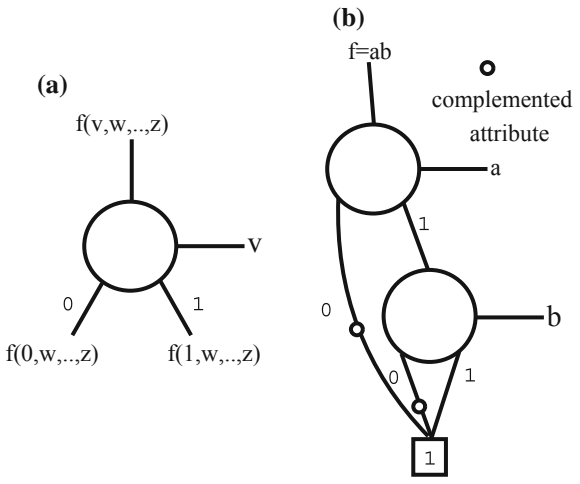


Fig. 2.1 BDD *non-terminal* node (**a**) canonical BDD for $a \cdot b$ function (**b**)

Each internal node also represents the Shannon's expansion with respect to its variable v:

$$f(v, w, \ldots, z) = v \cdot f(1, w, \ldots, z) + v' \cdot f(0, w, \ldots, z) \tag{2.1}$$

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 (i) each input variable is encountered at most once in each root to sink path and in the same order on all such paths, (ii) each internal node represent a distinct logic function and (iii) only 0-edges can be complemented. Figure 2.1b 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 2-operand Boolean operator.

2.2.1.2 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 [24] 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 [22]. 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 [25]. 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 [30] are BDDs with modified reduction rules (node elimination) targeting efficient manipulation of sparse sets. *Transformation BDDs* (TBDDs) [33, 35] 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 [35]. 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 re-ordering) are not efficient with general TBDDs due to the presence of the input transformation

[22]. Nevertheless, helpful and practical TBDDs have been proposed in literature, such as linear sifting of BDDs [31, 32] and *Hybrid Decision Diagrams* (HDDs) [34]. Linear sifting consists of linear transformations between input variables carried out on-line 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 Sect. 2.3.1. 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 3 orders of magnitude) with respect to BDDs [34] 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:

$$f(v, w, \ldots, z) = f(0, w, \ldots, z) \oplus v \cdot (f(0, w, \ldots, z) \oplus f(1, w, \ldots, z)) \quad (2.2)$$

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 [40] 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 [22] 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 chapter, 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.

2.2.2 *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.

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. 2.2.

In this chapter, we mainly 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. 2.3 for the sake of clarity.

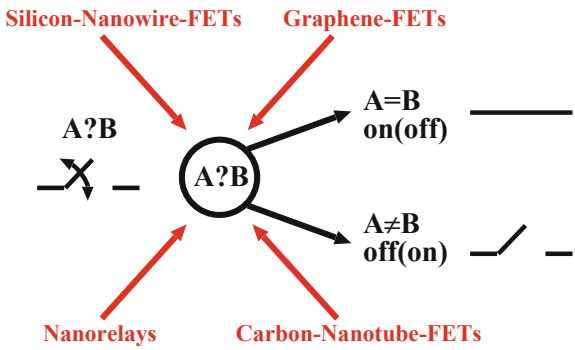


Fig. 2.2 Common logic abstraction for emerging devices: controllable polarity double-gate FETs in silicon nanowires [12], carbon nanotubes [13], graphene [14] but also six terminal nanorelays [15]

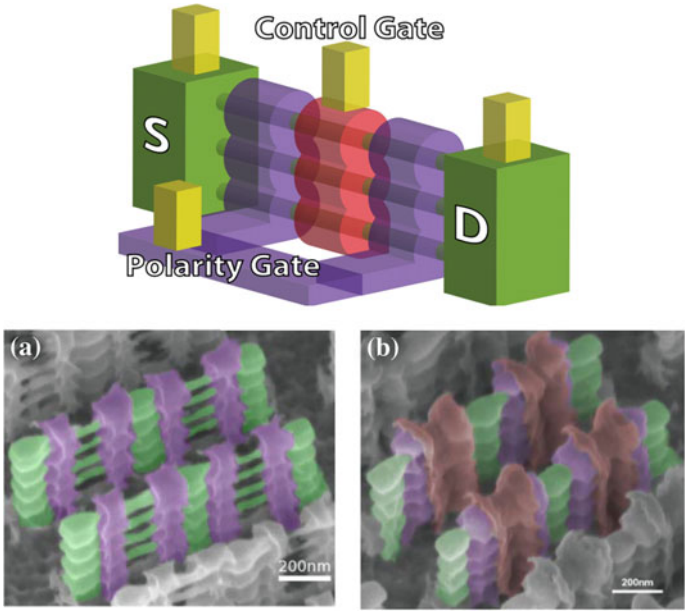


Fig. 2.3 Sketch structure and fabrication images of controllable polarity double-gate SiNWFETs from [12]

Then, we also present results for two other nanotechnologies featuring a two-input comparator functionality: nanorelays [15] and reversible logic [55].

Without a dedicated logic abstraction and synthesis methodology, the full potential of these technologies may remain unveiled. We propose in this paper a novel logic representation form, based on the biconditional connective, that naturally harnesses the operation of a two-input comparator. Section 2.6 will show the impact of our representation form in the synthesis of emerging nanotechnologies.

2.3 Biconditional Binary Decision Diagrams

This section introduces *Biconditional Binary Decision Diagrams* (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.

2.3.1 Biconditional Expansion

Logic expansions, also called decompositions, are the driving core of various types of decision diagrams. In [42], 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 2.1 The *biconditional expansion* is a two-variable expansion defined $\forall f \in \mathbb{B}^n$, with $n > 1$, as:

$$f(v, w, \ldots, z) = (v \oplus w) \cdot f(w', w, \ldots, z) + (v \odot w) \cdot f(w, w, \ldots, z) \tag{2.3}$$

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

As per the *biconditional expansion* in (2.3), 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 [31, 32] 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 [31, 32] operate as post-processing 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.

2.3.2 BBDD Structure and Ordering

Biconditional Binary Decision Diagrams (BBDD) are driven by the *biconditional expansion*. Each non-terminal node in a BBDD has the branching condition *biconditional* on two variables. We call these two variables the *Primary Variable* (PV) and

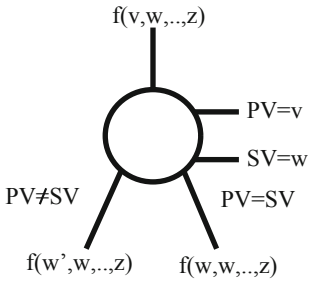


Fig. 2.4 BBDD *non-terminal* node

the *Secondary Variable* (SV). An example of a BBDD *non-terminal* node is provided by Fig. 2.4. 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:

$$\begin{cases} PV_i = \pi_i \\ SV_i = \pi_{i+1} \end{cases} \text{with } i = 0, 1, \ldots, n-2; \begin{cases} PV_{n-1} = \pi_{n-1} \\ SV_{n-1} = 1 \end{cases} \tag{2.4}$$

Example 2.1 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 (2.4) are applied as $PV_2 = \pi_2$ and $SV_2 = 1$. The consecutive ordering by couples (PV_i, SV_i) is thus $((\pi_0, \pi_1), (\pi_1, \pi_2), (\pi_2, 1))$.

The *Chain Variable Order* (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 2.2 A *complete* OBBDD of n variables has 2^n-1 distinct internal nodes, no sharing, and 2^n terminal 0–1 nodes.

Lemma 2.1 *For a Boolean function f and a variable order π, there exists only one* complete *OBBDD ordered with CVO(π).*

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 CVO(π). 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 CVO(π) are unique. ■

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

2.3.3 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.

2.3.3.1 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.

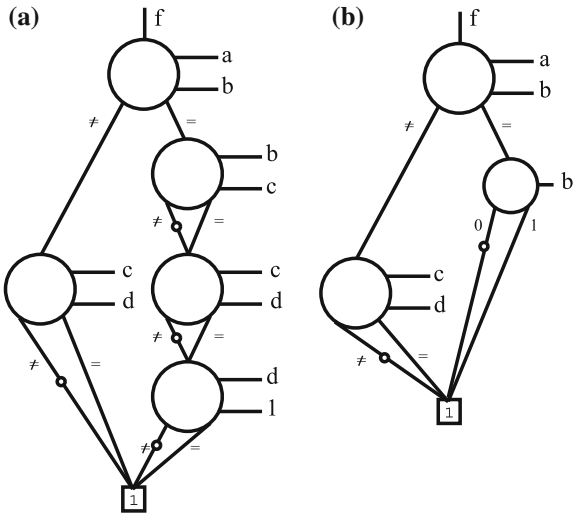


Fig. 2.5 Function to be represented: $f = a \cdot b + (a \oplus b) \cdot (c \odot d)$, *weak* ROBBDD for f (**a**) and *strong* ROBBDD for f (**b**)

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

Figure 2.5 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 4 levels (one for each variable) to fully represent the target function resulting in 5 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 3 levels of depth and 3 internal nodes.

2.3.3.2 Canonicity

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

Lemma 2.2 *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 [44], 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 2.1, and the iterative reduction based on rules **R1** and **R2** leads to a unique outcome, owing to [44], also *weak* ROBBDD are unique for a CVO(π), i.e., canonical. ■

Theorem 2.1 *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. 2.5). 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. ■

2.3.4 BBDD Complemented Edges

Being advantageously applied in modern ROBDD2s 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 [22]. Similarly, we extend ROBBDDs to use complemented edges only at $\neq$-edges, with also only logic 1 terminal node available, to maintain canonicity.

Theorem 2.2 *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 [22]. 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 2.2 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 2.1 maintains its validity with complemented edges. Consequently, *strong* ROBBDDs with complemented edges at $\neq$-edges are canonical. ■

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

2.3.5 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 [19].

2.3.5.1 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*.

2.3.5.2 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 (i) *unique* table to store BBDD nodes in a *strong canonical form*,[1] (ii) recursive formulation of Boolean operations in terms of *biconditional expansions* with relative *computed* table, (iii) memory management to speed up computation and (iv) *chain* variable re-ordering to minimize the BBDD size. We discuss in details each point hereafter.

2.3.5.3 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.

[1] A strong canonical form is a form of data pre-conditioning to reduce the complexity of equivalence test [46].

2.3.5.4 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.1α). 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. 1β). Otherwise, $f \otimes g$ has to be explicitly computed (Alg. 1γ). The top level in the CVO for $f \otimes g$ is determined as $i = max_{level}\{f, g\}$ 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 $min_{level}\{f, g\}$ 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 $f \otimes g$:

$$f \otimes g = (v \oplus w)(f_{v \neq w} \otimes g_{v \neq w}) + (v \overline{\oplus} w)(f_{v=w} \otimes g_{v=w}) \tag{2.5}$$

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.

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** } α
  $\{R, Attr\}$ = identical_terminal($\{f, g, \otimes\}$);
  return $\{R, Attr\}$;
else if *computed* table has entry $\{f, g, \otimes\}$ **then** } β
  $\{R, Attr\}$ = lookup *computed* table($\{f, g, \otimes\}$);
  return $\{R, Attr\}$;
else } γ
  $i = max_{level}\{f, g\}$;
  $\{v, w\} = \{PV, SV\}@(level = i)$;
  if $(|supp(f)| == 1)||(|supp(g)| == 1)$ **then**
    chain-transform(f, g);
  end if
  $\{E, E \rightarrow Attr\} = f_{v=w} \otimes g_{v=w}$;
  $\otimes_D = update_{op}(\otimes, f_{v \neq w} \rightarrow Attr, g_{v \neq w} \rightarrow Attr)$;
  $\{D, D \rightarrow Attr\} = f_{v \neq w} \otimes_D g_{v \neq w}$;
  if reduction rule **R4** applies **then**
    R =BDD-node $@(level = i)$;
  else if $\{E, E \rightarrow Attr\} == \{D, D \rightarrow Attr\}$ **then**
    $R = E$;
  else
    $D \rightarrow Attr = update_{attr}(E \rightarrow Attr, D \rightarrow Attr)$;
    $R = lookup_insert(i, D, D \rightarrow Attr, E)$;
  end if
  insert *computed* table $(\{f, g, \otimes\}, R, E \rightarrow Attr)$;
  return $\{R, E \rightarrow Attr\}$;
end if

Observe that the maximum number of recursions in Eq. 2.5 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.

2.3.5.5 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 {size×access-time} quality metric.

The core hashing-function for all BBDD tables is the Cantor pairing function between two integer numbers [45]:

$$C(i, j) = 0.5 \cdot (i + j) \cdot (i + j + 1) + i \tag{2.6}$$

which is a bijection from $\mathbb{N}_0 \times \mathbb{N}_0$ to $\mathbb{N}_0$ and hence a *perfect hashing function* [45]. 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 re-arrange the elements in the table. Standard modifications of the hash-function consist of nested Cantor pairings re-ordering and re-sizing of the prime number m.

2.3.5.6 Chain Variable Re-Ordering

The chain variable order for a BBDD influences the representation size and therefore its manipulation complexity. Automated chain variable re-ordering assists the BBDD package to boost the performance and reduce the memory requirements. Efficient reordering techniques for BDDs are based on local variable swap [47] 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 Eq. 2.4, 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

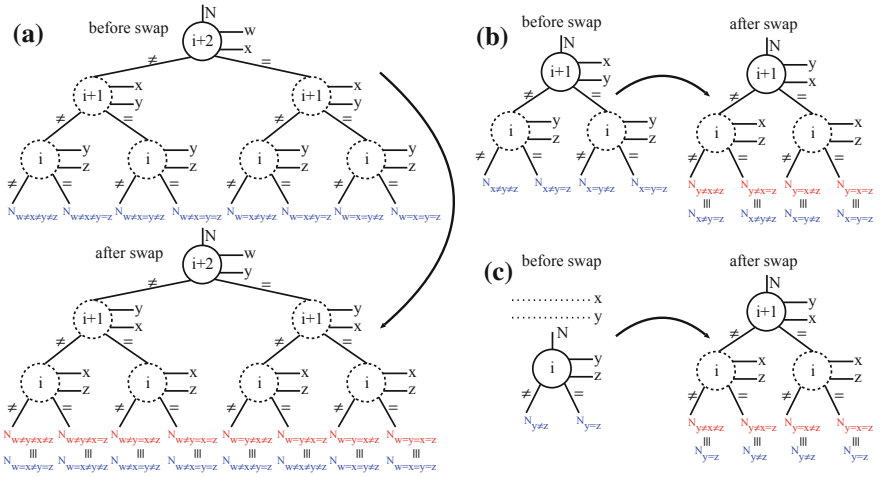


Fig. 2.6 Variable swap $i \rightleftharpoons i+1$ involving the CVO levels $(PV_{i+2} = w, SV_{i+2} = x)$, $(PV_{i+1} = x, SV_{i+1} = y)$ and $(PV_i = y, SV_i = z)$. Effect on nodes at level $i+2$ (**a**) $i+1$ (**b**) and i (**c**)

$i+2$ 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. Figure 2.6 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:

$$\begin{aligned} &\star \in \{=, \neq\}, \qquad \overline{\star} : \{=, \neq\} \rightarrow \{\neq, =\} \\ &(w \star_{i+2} x = y \star_i z) \rightarrow (w \star_{i+2} y = x \star_i z) \\ &(w \star_{i+2} x \neq y \star_i z) \rightarrow (w \overline{\star}_{i+2} y \neq x \overline{\star}_i z) \end{aligned} \tag{2.7}$$

Following remapping rules in Eq. 2.7, the children for N can be repositioned coherently with the variable swap. In Fig. 2.6, 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 re-ordering can be carried out in different fashions. A popular approach for BDDs is the sifting algorithm presented in [47]. 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 [49], simulated annealing techniques [50], genetic algorithms [51] 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.

2.4 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.

2.4.1 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.

2.4.1.1 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. 2.7, 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:
(**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.

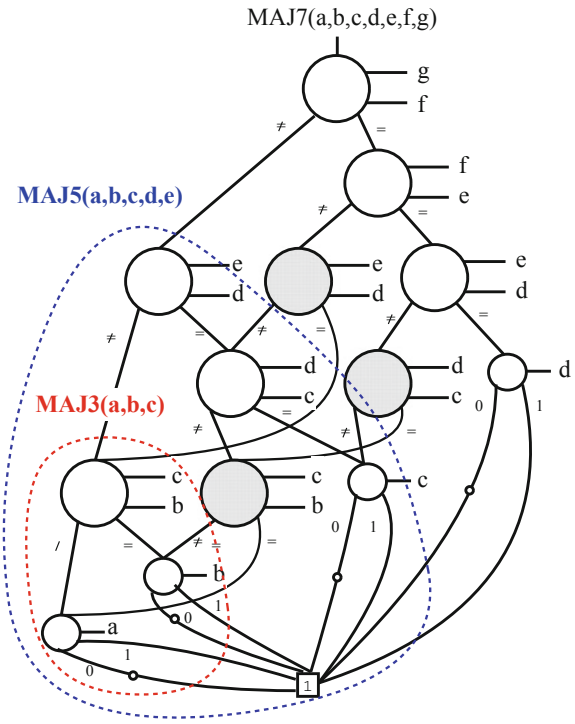


Fig. 2.7 BBDD for the 7-input majority function. The inclusion of MAJ_5 and MAJ_3 functions is illustrated. *Grey* nodes are nodes with inverted children due to n to $n-2$ majority reduction

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. 2.7).

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 [22].

Theorem 2.3 *A BBDD for the majority function of n (odd) variables has* $\frac{1}{4}(n^2+7)$ *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 non-homogeneous recurrence relation. Linear algebra methods [27] can solve such recurrence equation. The closed-form solution is $|MAJ_n| = \frac{1}{4}(n^2+7)$. ■

Note that with standard BDDs, the number of nodes is $|MAJ_n| = \lceil \frac{n}{2} \rceil (n - \lceil \frac{n}{2} \rceil + 1) + 1$ [22]. It follows that BBDDs are always more compact than BDDs for majority,

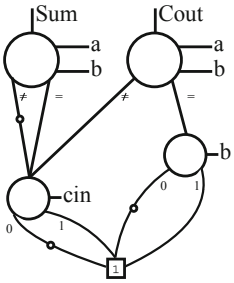


Fig. 2.8 Full adder function with BBDDs, variable order $\pi = (a, b, cin)$

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* 2.3, have been verified experimentally.

2.4.1.2 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. 2.8.

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

In Fig. 2.9, 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.

Theorem 2.4 *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. 2.9. ■

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 [22]. For n-bit adders, BBDDs save about 40 % of the nodes compared to BDDs. These results, and the law of *Theorem* 2.4, have been verified experimentally.

2.4.2 Experimental Results

The manipulation and construction techniques described in Sect. 2.3.5 are implemented in a BBDD software package [19] using C programming language. Such

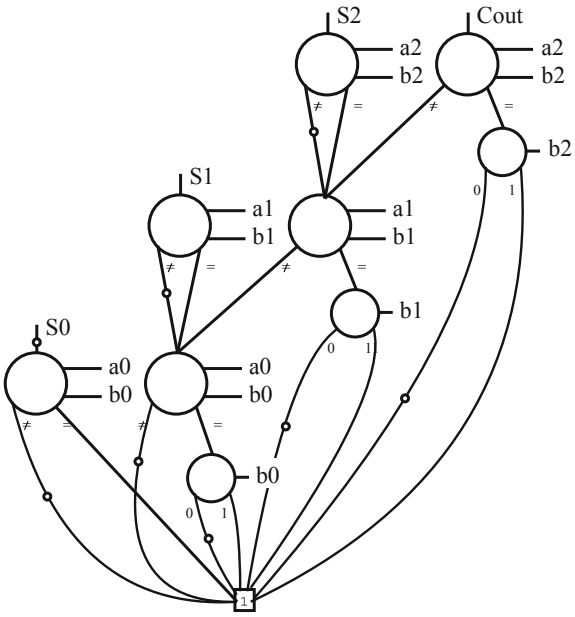


Fig. 2.9 BBDD for the 3-bit binary adder function, variable order $\pi = (a_2, b_2, a_1, b_1, a_0, b_0)$

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: (i) MCNC suite, (ii) portion of Open Cores designs and (iii) 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 [17]. For all packages, dynamic reordering during construction is enabled and based on the original sifting algorithm [47]. 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 2.1 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

Table 2.1 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

(continued)

Table 2.1 (continued)

Benchmarks	Inputs	Outputs	Wires	BBDD		CUDD (BDD)		*puma* (KFDD)	
				Node Count	Runtime (s)	Node Count	Runtime (s)	Node Count	Runtime (s)
Combinational portions of open cores benchmarks									
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

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 highly optimized during years.

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 $5\times$ smaller than BDDs and KFDDs for such benchmarks. Moreover, the runtime of the BBDD package is $4.4\times$ 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.

2.5 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 silicon nanowire technology. We consider the application of BBDDs in logic synthesis and formal equivalence checking tasks for a real-life telecommunication circuit.

2.5.1 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. Moreover, when intertwined with random logic, arithmetic portions are difficult to identify. Differently, BBDD nodes inherently act as two-variable comparators, a basis function for arithmetic operations. This feature opens the opportunity to restructure and identify arithmetic logic via BBDD representation.

We employ the BBDD package as front-end 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.

2.5.2 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, because equivalence test between two functions corresponds to a simple pointer comparison. 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.

2.5.3 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 earlier, one compelling reason to study BBDDs is to provide a natural design abstraction for emerging 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: (i) a conventional CMOS 22-nm technology and (ii) 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 8 main modules, among them 2 are sequential, one is the top entity, and 6 are potentially arithmetic intensive. We process the 6 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*. Figure 2.10a depicts the logic network before processing and Fig. 2.10b illustrates the equivalent circuit after

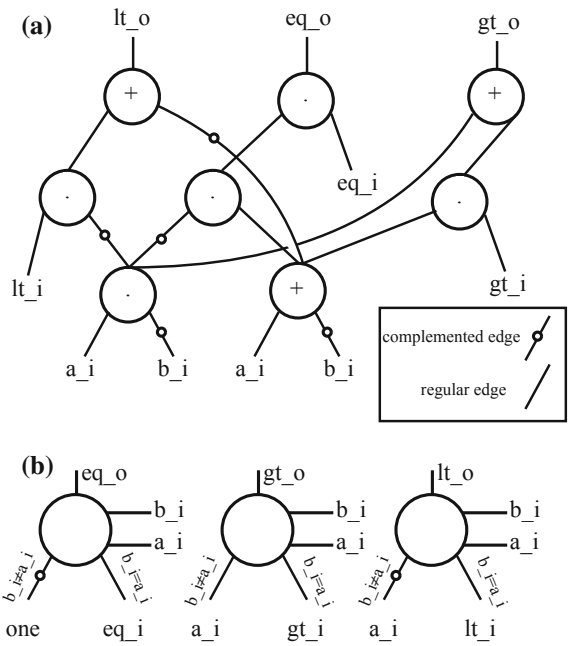


Fig. 2.10 Representations for the bit_comparator circuit in [48] (inverters are bubbles in edges). **a** Original circuit **b** BBDD re-writing, reduced BDD nodes are omitted for the sake of illustration

BBDD-restructuring. BDD nodes due to rule **R4** are omitted for simplicity. An advantage in both size and depth is reported. Table 2.2 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.

2.5.3.1 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 %.

2.5.3.2 Emerging DG-SiNWFET Technology

The controllable-polarity DG-SiNWFET technology features much more compact arithmetic (XOR, MAJ, etc.) gates than in CMOS, enabling faster and smaller implementation opportunities. For this reason, we set a tighter clock constraint

Table 2.2 Experimental results for BBDD-based design synthesis and verification

Case Study for Design and 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	x
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	✓

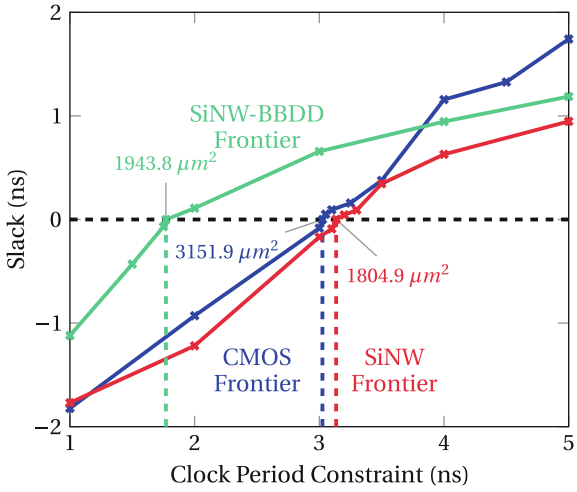


Fig. 2.11 Target *versus* obtained frequency curves and frequency frontiers for CMOS, SiNW-standard and SiNW-BBDD designs

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 %.

2.5.3.3 Post Place and Route Results

Using physical models for both CMOS and DG-SiNWFET technology, we also generated physical design results for the iterative product code decoder. In this set of experiments, the maximum clock period is determined by sweeping the clock constraint between 1 ns (1 GHz) and 5 ns (200 MHz) and repeating the implementation process. Figure 2.11 shows the post-*Place and Route* slack *versus* target clock constraint curves. Vertical lines highlight the clock constraint barriers for standard-SiNW (red), CMOS (blue) and BBDD-SiNW (green) designs. In the following, we report the shortest clock period achieved.

After place and route, the CMOS design reaches 331 MHz of clock frequency with area occupancy of 4271 μm^2 and EDP of 13.4 nJ.ns. The SiNWFET version, synthesized with plain design tools, has a slower clock frequency of 319 MHz and a larger EDP of 14.2 nJ.ns, but a lower area occupancy of 2473 μm^2. The final SiNWFET design, synthesized with BBDD-enhanced synthesis techniques, attains

the fastest clock frequency of 565 MHz and the lowest EDP of 8.7 nJ.ns with a small 2643 μm^2 of area occupancy.

If just using a standard synthesis tool suite, SiNWFET technology shows similar performances to CMOS, at the same technology node. This result alone would discard the SiNWFET technology from consideration because it brings no advantage as compared to CMOS. However, the use of BBDD abstraction and synthesis techniques enable a fair evaluation on the SiNWFETs technology, that is indeed capable of producing a faster and more energy efficient realization than CMOS for the *Iterative Product Code Decoder*.

2.5.3.4 Combinational Verification

The verification of the combinational 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.

2.6 BBDDs as Native Design Abstraction for Nanotechnologies

BBDDs are the natural and native design abstraction for several emerging technologies where the circuit primitive is a comparator, rather than a switch. In this section, we test the efficacy of BBDDs in the synthesis of two emerging nanotechnologies other than the previously considered silicon nanowires: reversible logic and nanorelays. We start by introducing general notions on these two nanotechnologies in order to explain their primitive logic operation. Then, we show how the BBDD logic model fits and actually helps in exploiting at best the expressive power of the considered nanotechnologies.

Note that many other nanodevices may benefit from the presented biconditional synthesis methodologies [53, 54] however a precise evaluation of their performance is out of the scope of the current study.

2.6.1 Reversible Logic

The study of reversible logic has received significant research attention over the last few decades. This interest is motivated by the asymptotic zero power dissipation ideally achievable by reversible computation [57, 58]. Reversible logic finds application in a wide range of emerging technologies such as quantum computing [58], optical computing [59], superconducting devices and many others [60].

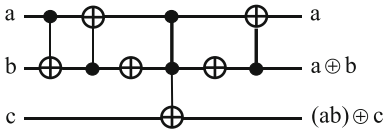


Fig. 2.12 Reversible circuit made of Toffoli, CNOT and NOT reversible gates

Reversible logic circuits are made of reversible logic gates [61]. Prominent reversible logic gates are, NOT gate: $Not(x) = x'$; CNOT gate: $CNOT(x, y) = (x, x \oplus y)$, which can be generalized with Tof_n gate with first $n-1$ variables acting as control lines: $Tof_n(x_1, x_2, \ldots, x_n, y) = (x_1, x_2, \ldots, x_n, (x_1 \cdot x_2 \cdot \ldots \cdot x_n) \oplus y)$. From a conceptual point of view, a CNOT gate is nothing but a Tof_n gate with $n = 1$. Analogously, a NOT gate is nothing but a Tof_n gate with $n = 0$. The Tof_n set of reversible logic gates form an universal gate library for realizing any reversible Boolean function. For the sake of clarity, we report in Fig. 2.12 an example of reversible circuit made of Toffoli reversible gates. We follow the established drawing convention of using the symbol $\oplus$ to denote the target line and solid black circles to indicate control connections for the gate. An $\oplus$ symbol with no control lines denotes a NOT gate.

Whether finally realized in one emerging technology or the other, reversible circuits must exploit at best the logic expressive power of reversible gates. Being the Toffoli gate the most known reversible gate, harnessing the biconditional connective embedded in its functionality is of paramount importance.

The efficiency of reversible circuits strongly depends on the capabilities of reversible synthesis techniques. Due to the inherent complexity of the reversible synthesis problem, several heuristics are proposed in the literature. Among those, the ones based on decision diagrams offer an attractive solution due to scalability and ability to trade-off diverse performance objectives.

Reversible circuit synthesis based on decision diagrams essentially consists of two phases. First, the generation of decision diagrams is geared towards efficient reversible circuit generation. This typically involves nodes minimization or other DD complexity metric reduction. Second, node-wise mapping is performed over a set of reversible gates.

The current standard for DD-based reversible synthesis uses binary decision diagrams generation via existing packages [10] and a custom node-wise mapping. However, standard BDDs do not match the intrinsic functionality of popular reversible gates that are comparator(XOR)-intensive. Instead, BBDDs are based on the biconditional expansion which natively models reversible XOR operations. In this way, BBDDs enable a more compact mapping into common reversible gates, such as Toffoli gates [55]. Figure 2.13 depicts the efficient mapping of a single BBDD node into reversible gates. The additional reversible gates w.r.t. a traditional BDD mapping are marked in gray. As one can notice, two extra gates are required. However, when comparing the functionality of BBDD nodes w.r.t. BDD nodes, it is apparent that more information is encoded into a single BBDD element. This is because the BBDD core expansion examines two variables per time rather than only one. Consequently,

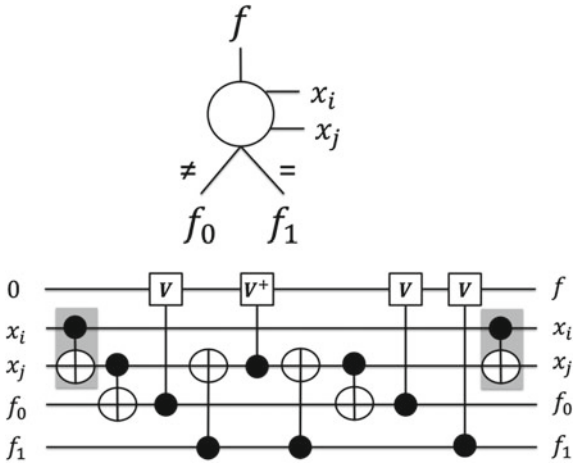


Fig. 2.13 Reversible circuit for a BBDD node [55]

the node count reductions deriving from the use of BBDDs overcompensate the slight increase in the direct mapping cost w.r.t. BDDs.

Our novel reversible synthesis flow uses BBDD logic representation and minimization using the package [19] and a final one-to-one mapping of BBDD nodes as depicted by Fig. 2.13. As reference flow, we consider the traditional BDD-based reversible synthesis approach. To validate the BBDD effectiveness, we run synthesis experiments over reversible benchmarks taken from the RevLib online library [23]. In this context, we estimate the implementation cost using the *Quantum-Cost* (QC) [55]. Table 2.3 shows the reversible synthesis results. Out of 26 benchmarks functions studied, 20 reported improved QC and 13 reported improvement in QC as well as line count. A closer study reveals that some benchmark functions, e.g., *plus63mod4096*, contain major contribution from non-linear sub-circuits, which are represented in more compact form by BDD. This translates to better performance in BDD-based synthesis. Nevertheless, future improvement in BBDD construction heuristics may bridge also this gap.

These results provide a fair perspective on the efficacy of BBDDs in reversible synthesis for emerging nanotechnologies.

2.6.2 NEMS

Nano-Electro-Mechanical Relay (NEMS), or simply nanorelays, are electrostatically actuated mechanical switches [64]. The good properties of nanorelays are (i) very low on-state intrinsic resistance (0.5Ω) and (ii) virtually infinitely large off-state resistance [63]. On the other hand, the key hurdles of nanorelays are (i) long switching time (hundreds of nanoseconds), (ii) relatively short lifetime (10^8 switching cycles)

Table 2.3 Results for reversible circuit synthesis using BBDDs versus traditional BDDs

Benchmark		BDD			BBDD			Improvement (%)	
Name	I/O	Line	QC	Runtime (s)	Line	QC	Runtime (s)	Line	QC
4mod5_8	4/1	7	24	<0.01	6	10	0.01	14.28	58.33
decod24_10	2/4	6	27	<0.01	6	23	0.02	0	14.81
mini-alu_84	4/2	10	60	<0.01	8	42	0.03	20.00	30.00
alu_9	5/1	7	29	0.01	7	25	0.02	0	13.79
rd53_68	5/3	13	98	<0.01	13	81	0.03	0	17.34
mod5adder_66	6/6	32	292	<0.01	32	269	0.05	0	7.76
rd73_69	7/3	13	217	<0.01	15	117	0.04	–15.38	46.08
rd84_70	8/4	34	304	<0.01	31	256	0.04	8.82	15.79
sym6_63	6/1	14	93	<0.01	11	49	0.02	21.43	47.31
sym9_71	9/1	27	206	<0.01	22	124	0.06	18.52	39.81
cycle10_2_61	12/12	39	202	0.09	25	183	0.03	35.89	9.41
cordic	23/2	52	325	0.06	50	222	0.02	3.84	31.69
bw	5/28	87	943	0.11	78	645	0.03	10.35	31.6
apex2	39/3	498	5922	0.24	744	5242	9.3	–49.39	11.48
seq	41/35	1617	19632	1.14	2440	18366	27.78	–50.89	6.45
spla	16/46	489	5925	0.10	788	5315	1.16	–61.15	10.3
ex5p	8/63	206	1843	0.24	251	1682	1.1	–21.85	8.74
e64	65/65	195	907	0.04	192	826	1.14	1.54	8.93
ham7_29	7/7	21	141	<0.01	18	153	0.03	14.29	–8.51
haml5_30	15/15	45	309	0.25	43	573	0.06	4.44	–85.44
hwb5_13	5/5	28	276	0.01	30	238	0.02	–7.14	13.77

(continued)

Table 2.3 (continued)

Benchmark		BDD			BBDD			Improvement (%)	
Name	I/O	Line	QC	Runtime (s)	Line	QC	Runtime (s)	Line	QC
hwb6_14	6/6	46	507	<0.01	49	488	0.06	–6.52	3.75
hwb7_15	7/7	73	909	<0.01	102	978	0.12	–39.73	–7.59
hwb8_64	8/8	112	1461	0.01	189	1831	0.35	–68.75	–25.33
plus63mod4096_79	12/12	23	89	0.08	28	186	0.16	–21.74	–108.99
plusl27mod8192_78	13/13	25	98	0.21	31	210	0.02	–24.00	–114.28

and (iii) limited scalability of minimum feature size [62, 63]. Nanorelays can be fabricated by top-down approaches using conventional lithography techniques or bottom-up approaches using carbon nanotubes or nanowire beams [63].

Nanorelays are a promising alternative to CMOS for ultralow-power systems [62–66] where their ideally zero leakage current (consequence of the large off-resistance) is a key feature to be harnessed.

Different nanorelay structures for logic have been proposed in the literature. Most of them are based on electrostatic actuation and they implement different switching (logic) functions depending on their number of terminals and device geometry. Mechanical contacts (connections) are enforced via electric fields between the various terminals. Two-terminals (2T) and three-terminals (3T) nanorelays are simple devices useful to solve preliminary process challenges. Trading off simplicity for functionality, four-terminals (4T) and six-terminals (6T) nanorelays are more expressive and desirable for compact logic implementations.

In [68], a 4T NEM relay is proposed consisting of a movable poly-SiGe gate structure suspended above the tungsten body, drain, and source electrodes. Figure 2.14 shows the 4 T relay conceptual structure and a fabrication microphotograph. When a voltage is applied between the gate structure and the body electrode a corresponding electric field arises and the relay is turned on by the channel coming into contact with the source and drain electrodes.

In [67], a 6T NEM relay is realized by adding an extra body (*Body2*) and an extra source (*Source2*) contacts to the previous 4T NEM relay. Figure 2.15 shows the 6 T relay conceptual structure and a fabrication microphotograph. The two body contacts are designed to be biased by opposite voltages. Either *Source1* or *Source2* to *Drain* connection is controlled by the gate to body positive or negative voltage and its corresponding electric field polarity.

Because of the electrostatic forces among the different terminals, both 4 T and 6T NEM relay naturally acts as a logic multiplexer driven by a bit comparator.

In this study, we focus on 6T NEM relays. To assess the potential of nanorelays in VLSI, a BDD-based synthesis flow has been presented in [67]. It first partitions a design in sub-blocks and then creates BDDs for those sub-blocks. For each local BDD, a one-to-one mapping strategy generates a netlist of nanorelays implementing the target logic function. Indeed, the functionality of each BDD node can be realized by a single nanorelay device. We consider this as the reference design flow for nanorelays.

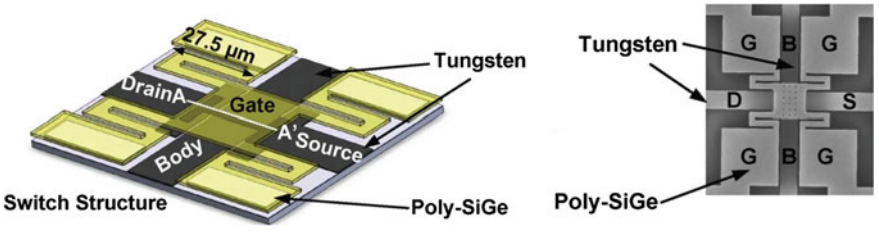


Fig. 2.14 Four-terminals nanorelay structure and fabrication image from [68]

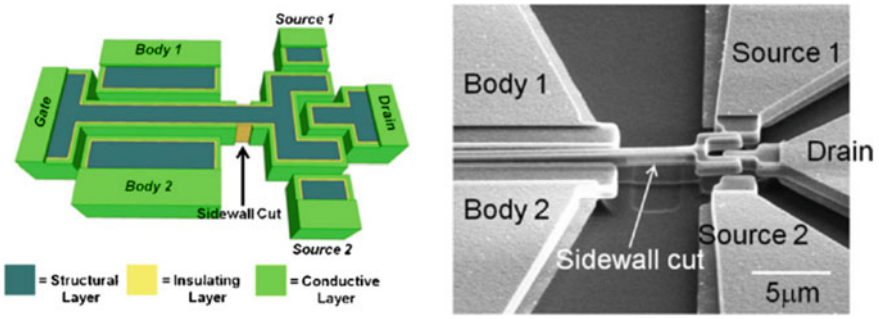


Fig. 2.15 Six-terminals nanorelay structure and fabrication image from [67]

Table 2.4 Total number of relays, the number of relays on the critical path, and ratios compared to [67] (MCNC benchmark circuits)

Circuit name	Relays	Levels	R. Ratio [67]	L. Ratio [67]
alu4	599	14	0.77	1.00
apex4	992	8	0.90	0.89
des	3130	18	0.78	1.00
ex1010	1047	10	0.94	0.91
ex5p	283	8	0.92	1.00
misex3	846	14	1.29	1.00
pdc	865	14	0.35	0.88
spla	691	16	0.82	1.00
8-b adder	28	9	0.19	0.53
16-b adder	56	17	0.31	0.52
8×8 multiplier	14094	16	1.05	1.00
Average	2057.36	13.09	0.76	0.88

From the analysis we performed above, we know that nanorelays can implement much more complex Boolean functions than just 2:1 multiplexers. Indeed, the functionality of these nanorelays is naturally modeled by a BBDD node. For this reason, we propose a novel design flow based on BBDDs to take full advantage of the nanorelays expressive power. Analogously to the BDD design flow, the design is first pre-partitioned if necessary. Then, local BBDDs are built and each BBDD node is mapped into a single nanorelay device.

We first test the BBDD-design flow against the MCNC benchmark suite. Table 2.4 shows the number of relays and the number of relays on the critical path. It compares these numbers with the corresponding numbers in [67] and shows the BBDD to BDD ratio for the different benchmark circuits. We also provide the ratios for the number of relays on the critical path. The BBDD design flow results in an average reduction in NEM relays of 24 %. This is due to the compactness of the BBDD representation relative to the BDD representation. Since BBDDs require less nodes

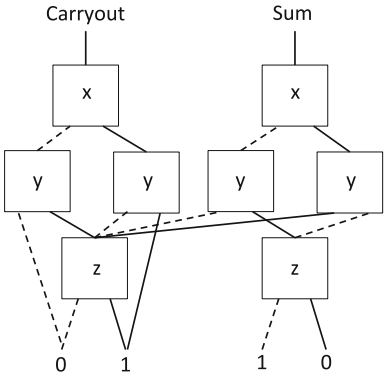


Fig. 2.16 Nanorelay implementation of a full-adder using a BDD-based design approach [67]

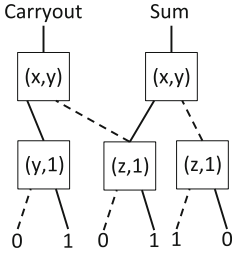


Fig. 2.17 Nanorelay implementation of a full-adder using a BBDD-based design approach. *Dotted lines* represent $\neq$-edges and *solid lines* are $=$-edges

than BDDs, BBDD circuits require less NEM relays. Furthermore, the BBDD design flow enables us to obtain circuits with shorter critical paths. On average the critical path length is reduced by 12 %. This decrease in the critical paths is due to the BBDD reduction rules, which can be leveraged to decrease the height of BBDDs more than the reduction rules for their BDD counterparts.

We also compare the BBDD-approach in the case of synthesizing an 8×8 array multiplier. In [67] the BDD-based approach is tested on such a multiplier implemented using a carry-save adder tree followed by a ripple carry adder. We represent the same multiplier, but using BBDDs instead of BDDs. The main source of advantage here is that BBDDs represent more compactly full-adders and half-adders as compared to BDDs. Figures 2.16 and 2.17 depicts the nanorelay implementation of a full-adder using BDD and BBDD approaches, respectively. Each squared box in the figures represents a six-terminal nanorelay device. We show that this compact representation allows us to implement the multiplier using a smaller number of NEM relays. Table 2.5 shows the corresponding results. It is possible to see that the BBDD design flow requires a smaller number of relays. On average, reduce the number of relays is reduced by 36 % w.r.t. the BDD design flow. Furthermore, as the number of mechanical delays decreases, so does the ratio of the number of relays required by the BBDD representation versus the BDD representation.

These results show the impact of a dedicated logic abstraction to design a comparator-intrinsic nanotechnology, such as nanorelays.

Table 2.5 Comparison of BDD-based versus BBDD-based synthesis of an 8×8 array multiplier

Mech. Delays	BBDD-Relays	BDD-Relays	Ratio BBDD/BDD
2	2491	4129	0.60
3	1367	2186	0.63
4	647	875	0.74
5	434	590	0.74
6	407	533	0.76
Average	1069.2	1662.6	0.64

2.7 Summary

Following the trend to handle ever-larger designs, and in light of the rise of emerging technologies that natively implement new Boolean primitives, the study of innovative logic representation forms is extremely important. In this chapter, 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 (i) push further the efficiency of traditional decision diagrams and (ii) 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 % (14 % post *place and route*). With an emerging 22-nm controllable-polarity DG-SiNWFET technology, BBDD-restructuring shrinks more the critical path by 32 % (22 % post *place and route*), 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. Results on other two nanotechnologies, i.e., reversible logic and nanorelays, demonstrate that BBDDs are essential to permit a fair technology evaluation where the logic primitive is a binary comparator.

References

1. C.Y. Lee2, Representation of switching circuits by binary-decision programs. Bell Syst. Tech. J. **38**(4), 985–999 (1959)
2. S.B. Akers2, Binary decision diagrams. IEEE Trans. Comp. **100**(6), 509–516 (1978)

3. R.E. Bryant, Graph-based algorithms for Boolean function manipulation. IEEE Trans. Comput. **100**(8), 677–691 (1986)
4. C. Yang, M. Ciesielski, BDS: A BDD-based logic optimization system. IEEE Trans. CAD IC Syst. **21**(7): 866–876 (2002)
5. S. Malik et al., Logic verification using binary decision diagrams in a logic synthesis environment in *IEEE International Conference on CAD* (1988), pp. 6–9
6. M.S. Abadir et al., Functional test generation for digital circuits using binary decision diagrams. IEEE Trans. Comput. **100**(4), 375–379 (1986)
7. C. Scholl, R. Drechsler, B. Becker, Functional simulation using binary decision diagrams in *IEEE International Conference on CAD* (1997), pp. 8–12
8. U. Kebschull, W. Rosenstiel, E. Schubert, Multilevel logic synthesis based on functional decision diagrams, in *IEEE European Conference Design Automation* (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. CAD IC Syst. **17**(10), 965–973 (1998)
10. *CUDD: CU Decision Diagram Package Release 2.5.0*. http://vlsi.colorado.edu/fabio/CUDD/cuddIntro.html
11. *Decision Diagram-Package PUMA*. 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 Meeting* (2012), pp. 8–14
13. Y. Lin et al., High-performance carbon nanotube field-effect transistor with tunable polarities. IEEE Trans. Nanotech. **4**(5), 481–489 (2005)
14. N. Harada et al., A polarity-controllable graphene inverter. Appl. Phys. Lett. **96**(1), 012102 (2010)
15. D. Lee2 et al., Combinational logic design using six-terminal NEM relays. IEEE Trans. CAD IC Syst. **32**(5), 653–666 (2013)
16. L. Amaru, P.-E. Gaillardon, S. Mitra, G. DeMicheli, New logic synthesis as nanotechnology enabler, in *Proceedings of the IEEE* (2015)
17. L. Amaru, P.-E. Gaillardon, G. De Micheli, Biconditional BDD: a novel canonical representation form targeting the synthesis of XOR-rich circuits, in Design Automation and Test in Europe (2013), pp. 1014–1017
18. L. Amaru, P.-E. Gaillardon, G. De Micheli, An efficient manipulation package for biconditional binary decision diagrams, in *Design Automation and Test in Europe* (2014), pp. 296–301
19. *BBDD package*. http://lsi.epfl.ch/BBDD
20. L. Kathleen, *Logic and Boolean Algebra*, Barrons Educational Series (1979)
21. G. De Micheli, *Synthesis and Optimization of Digital Circuits* (McGraw-Hill, New York, 1994)
22. I. Wegener, *Branching Programs and Binary Decision Diagrams: Theory and Applications*, vol. 4 (SIAM, Philadelphia, 2000)
23. M. Kreuzer, L. Robbiano, *Computational Commutative Algebra*, vol. 1 (Springer, Berlin, 2005)
24. R.E. Bryant, On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. IEEE Trans. Comput. **40**(2), 205–213 (1991)
25. J. Gergov, C. Meinel, Mod-2-OBDDs A data structure that generalizes EXOR-sum-of-products and ordered binary decision diagrams. Form. Methods Syst. Des. **8**(3), 273–282 (1996)
26. B. Bollig, Improving the variable ordering of OBDDs is NP-complete. IEEE Trans. Comput. **45**(9), 993–1002 (1996)
27. T. Koshy, *Discrete Mathematics with Applications* (Academic Press, Cambridge, 2004)
28. T.S. Czajkowski, S.D. Brown, Functionally linear decomposition and synthesis of logic circuits for FPGAs. IIEEE Trans. CAD IC Syst. **27**(12), 2236–2249 (2008)
29. J.F. Groote, J. Van de Pol, *Equational Binary Decision Diagrams*, Logic for programming and automated reasoning (Springer, Heidelberg, 2000)
30. S. Minato, Zero-suppressed BDDs for set manipulation in combinatorial problems, in *IEEE Conference on Design Automation (DAC)* (1993), pp. 272–277

31. C. Meinel, F. Somenzi, T. Theobald, Linear sifting of decision diagrams, in *IEEE Conference on Design Automation (DAC)* (1997), pp. 202–207
32. W. Gunther, R. Drechsler, BDD minimization by linear transformations. Adv. Comput. Syst. 525–532 (1998)
33. M. Fujita, Y. Kukimoto, R. Brayton, BDD minimization by truth table permutation, in *IEEE International Symposium on CAS* (1996), pp. 596–599
34. E.M. Clarke, M. Fujita, X. Zhao, Hybrid decision diagrams, in *IEEE International Conference on CAD* (1995), pp. 159–163
35. E.I. Goldberg, Y. Kukimoto, R.K. Brayton, Canonical TBDD's and their application to combinational verification, in *ACM/IEEE International Workshop on Logic Synthesis* (1997)
36. U. Kebschull, W. Rosenstiel, Efficient graph-based computation and manipulation of functional decision diagrams, in *IEEE Euro Conference on Design Automation* (1993), pp. 278–282
37. J.E. Rice, Making a choice between BDDs and FDDs, in *ACM/IEEE International Workshop on Logic Synthesis* (2005)
38. R. Drechsler, Ordered Kronecker Functional Decision Diagrams und ihre Anwndung, Ph.D. Thesis, 1996
39. S. Grygiel, M.A. Perkowski, New compact representation of multiple-valued functions, relations, and non-deterministic state machines, in *IEEE International Conference on Computer Design* (1998), pp. 168–174
40. A. Srinivasan, T. Kam, S. Malik, R. Brayton, Algorithms for Discrete Function Manipulation, in *IEEE International Conference on CAD* (1990), pp. 92–95
41. S. Minato et al., Shared BDD with attributed edges for efficient boolean function manipulation, in *IEEE Conference on Design Automation (DAC)* (1990), pp. 52–57
42. B. Becker, R. Drechsler, How many decomposition types do we need?, in *IEEE Euro Conference on Design Automation* (1995), pp. 438–442
43. B. Becker, R. Drechsler, M. Theobald, On the expressive power of OKFDDs. Form. Methods Syst. Des. **11**(1), 5–21 (1997)
44. R. Drechsler, B. Becker, *Binary Decision Diagrams: Theory and Implementation* (Kluwer Academic Publisher, The Netherlands, 1998)
45. P. Tarau, *Pairing Functions, Boolean Evaluation and Binary Decision Diagrams*, arxiv preprint arXiv:0808.0555 (2008)
46. K.S. Brace, R.L. Rudell, R.E. Bryant, Efficient implementation of a BDD package, in *IEEE Conference on Design Automation (DAC)* (1990), pp. 40–45
47. R. Rudell, Dynamic variable ordering for ordered binary decision diagrams, in *IEEE International Conference on CAD* (1993), pp. 42–47
48. *An iterative decoder for Product Code—from Open Cores*: http://opencores.org/project, product_code_iterative_decoder
49. S. Panda, F. Somenzi, Who are the variables in your neighborhood, in *IEEE International Conference on CAD* (1995), pp. 74–77
50. B. Bollig et al., Simulated annealing to improve variable orderings for OBDDs, in *ACM/IEEE International Workshop on Logic Synthesis* (1995)
51. R. Drechsler et al., A genetic algorithm for variable ordering of OBDDs, in *ACM/IEEE International Workshop on Logic Synthesis* (1995)
52. ABC synthesis tool - available online
53. J. Hagenauer, E. Offer, L. Papke, Iterative decoding of binary block and convolutional codes. IEEE Trans. Inf. Theory **42**(2), 429–445 (1996)
54. A. Picart, R. Pyndiah, Adapted iterative decoding of product codes, in *Global Telecommunications Conference, 1999. GLOBECOM'99*, vol. 5 (IEEE, New York, 1999)
55. A. Chattopadyay, et al., Reversible logic synthesis via biconditional binary decision diagrams, in *Proceedings of the ISMVL 15*
56. RevLib is an online resource for benchmarks within the domain of reversible and quantum circuit design. http://www.revlib.org
57. C.H. Bennett, Logical reversibility of computation. IBM J. Res. Dev. **17**(6), 525532 (1973)

58. M. Nielsen, I. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2000)
59. R. Cuykendall, D.R. Andersen, Reversible optical computing circuits. Opt. Lett. **12**(7), 542544 (1987)
60. R.C. Merkle, Reversible electronic logic using switches, in *Nanotechnology*, vol. 4 (1993), p. 2140
61. A. Barenco et al., Elementary gates for quantum computation, in *Physical Review* (1995)
62. O. Loh, H. Espinosa, Nanoelectromechanical contact switches. Nat. Nanotechnol. **7**(5), 283–295 (2012)
63. *Nano-Electro-Mechanical Switches*, ITRS, white paper (2008)
64. V. Pott et al., Mechanical computing redux: relays for integrated circuit applications. Proc. IEEE **98**(12), 2076–2094 (2010)
65. Sharma, P., Perruisseau-Carrier, J., Moldovan, C., Ionescu, A. Electromagnetic performance of RF NEMS graphene capacitive switches. IEEE Trans. Nanotech. (2014)
66. Dana Weinstein, Sunil A. Bhave, The resonant body transistor. Nano Lett. **10**(4), 1234–1237 (2010)
67. D. Lee et al., Combinational logic design using six-terminal NEM relays. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(5), 653–666 (2013)
68. M. Spencer et al., Demonstration of integrated micro-electro-mechanical relay circuits for VLSI applications. IEEE J. Solid State Circuits **46**(1), 308–320 (2011)

http://www.springer.com/978-3-319-43173-4

New Data Structures and Algorithms for Logic Synthesis and Verification

Amaru, L.G.

2017, XVI, 156 p. 44 illus., 20 illus. in color., Hardcover
ISBN: 978-3-319-43173-4