Applications of Decision Diagrams in Digital Circuit Design

Per Lindgren
Division of Computer Engineering
Luleå University of Technology
97187 Luleå, Sweden
pln@sm.luth.se

December 1999

Abstract

Design methodology of digital circuits is a rapidly changing field. In the last 20 years, the number of transistors on a single chip has increased from thousands to tens of millions. This sets new demands on the design tools involved, their ability to capture specifications on a high level, and finally synthesize them into hardware implementations. The introduction of ***Decision Diagrams*** *(DDs) has brought new means towards solving many of the problems raised by the increasing complexity of todays designs. In this thesis, we study their use in VLSI CAD and develop a number of novel applications.*

Incomplete specifications are inherent to the functionality of almost all digital circuits. We present a design methodology providing a common basis between design validation and logic synthesis, namely the semantics of ***Kleenean Strong Ternary Logic****. This is called upon as commonly used design methodologies, based e.g. on VHDL, are shown to put design correctness in jeopardy. By an extension of DDs, we can efficiently represent and manipulate incompletely specified functions. The method presented, not only guarantees correctness of the final circuit, but also offers potential towards expressing and utilizing incompleteness in ways other methodologies are incapable of.*

The increasing density and speed of todays target technologies also changes the conditions for logic synthesis; e.g., traditional quality measures based on gate delays are becoming less accurate as delays caused by interconnections are raising their heads. To address this problem we propose methodologies allowing quality measures of the final circuit to be foreseen and considered throughout the whole synthesis process. In general this is a very hard task. We approach the problem by limiting our synthesis methodologies to those rendering regular layouts (such as computational arrays and lattices). The regularity allows us to predict properties of the final circuit and at the same time, ensure design criteria to be met, e.g., path delays and routability of the final circuit. In this thesis, we develop new design methodologies and their algorithms. By our experimental results, they are shown to offer significant improvements to both state of the art two-level and multi-level based tools in the area of layout driven synthesis.

Our minimization methods are based on ***Pseudo Kronecker Decision Diagrams*** *(PKDDs), which are the most general type of ordered bit-level diagrams for switching functions. In the thesis we elaborate on the properties of PKDDs and* ***Ternary*** *PKDDs (TPKDDs) and develop an efficient minimization method based on local variable exchange for TPKDDs. Furthermore, the problem of PKDD minimization is discussed and a number of different strategies are introduced and evaluated; the potential compactness of PKDDs is confirmed.*

The thesis spans from validation and verification of high-level specifications all the way down to layout driven synthesis, combining logic minimization, mapping and routing to the target architecture at hand. We conclude our work to offer new means towards solving many of the crucial problems occurring along the design process of modern digital circuits.

Contents

List of Figures

Acknowledgments

First of all I'd like to express my gratitude to Dr. Rolf Drechsler and Prof. Dr. Bernd Becker, for taking on the challenge to supervise me during my stay in Freiburg, August 1997 - December 1998, on a grant from the Swedish Institute. Also, I'd like to thank the entire staff of Systemteknik at Luleå University of Technology, for being supportive in all practical matters. To name a few; Jerker Delsing for taking me on as a Ph.D. student, Mats Blomqvist for his unselfishness, always helping out when it counts, and to our dean Bengt Lennartsson, for keeping the wheels turning in the right direction. And finally, to my parents who have always supported and encouraged me to work hard with my studies, thanks.

Luleå, December 1999
Per Lindgren

to my grandparents
Torborg and Martin Berggren

Chapter 1

Introduction

Design methodology of digital circuits is a rapidly changing field. In the last 20 years, the number of transistors on a single chip has increased from thousands to tens of millions. This sets new demands on the design tools involved, their ability to capture specifications on a high level, and finally synthesize them into hardware implementations. The introduction of *Decision Diagrams* (DDs) has brought new means towards solving many of the problems raised by the increasing complexity of todays designs. In this thesis, we study their use in VLSI CAD and develop a number of novel applications.

Incomplete specifications are inherent to the functionality of almost all digital circuits. We present a design methodology providing a common basis between design validation and logic synthesis, namely the semantics of *Kleenean Strong Ternary Logic*. This is called upon as commonly used design methodologies are shown to put design correctness in jeopardy. In particular, VHDL is shown to produce both overly optimistic and pessimistic results during simulation in the presence of incompletely specified functions. That is, in the worst case a design validated by simulation on the HDL level may render an erroneous implementation, even though synthesis was carried out correctly. The underlying problems are identified and addressed. By the means of *Kleenean Ternary Decision Diagrams* (Kleene-TDDs) we can efficiently represented and operate on incompletely specified functions under the Kleenean semantics. The methodology presented, not only guarantees correctness of the final circuit, but also offers potential towards expressing and utilizing incompleteness in ways other methodologies are incapable of. The Kleenean semantics allow the unification of unknown values (during simulation) with don't cares (of the specification), thus solving the problem of overly optimistic simulation results. Furthermore, pessimisms during simulation is known to force the designer to give an over-specified description, which in turn reduces the degrees of freedom for synthesis. Also this problem can be solved under the Kleenean algebra, using functional composition in reversed topological order on the RTL network. Furthermore, we show the applicability of Kleene-TDDs to logic synthesis problems; an efficient method for support set minimization is presented. This part of the thesis was first presented June 1997 to meet the requirements of a "licentiate" degree in computer engineering at Luleå University of Technology, and is available separately as a printed document "Dealing with Incompleteness in Circuit Design under Kleenean Strong Ternary Logic", (1997:18. ISSN:1402-1757. ISRN:LTU-LIC–1997/18–SE).

Logic synthesis is commonly carried out in individual steps such as; logic minimization, floorplanning, mapping and routing. Although each intermediate representation is optimized (under

local criteria) the overall result may be sub-optimal, e.g., a highly optimized netlist which fits badly onto the target architecture may cause violation of sought design criteria. Furthermore, the increasing density and speed of todays target technologies sets new demands on the synthesis tools. Traditional quality measures based on gate delays are becoming less accurate as delays caused by interconnections are raising their heads [30].

In order to address these problems we propose methodologies allowing quality measures of the final circuit to be foreseen and considered throughout the whole synthesis process. In general this is a very hard task. We approach the problem by limiting our synthesis methodologies to those rendering regular layouts (such as computational arrays and lattices). The regularity allows us to predict properties of the final circuit, and at the same time ensure design criteria to be met, e.g., path delays and routability of the implementation. In this thesis, we develop new design methodologies and their algorithms.

AND/OR-based arrays (PLAs) [31] constitutes a classic example of layout driven synthesis. The minimization problem involved is well understood and efficient tools have been developed over the years [7]. However, it is well known that certain functions are better represented as AND/EXOR Expressions (ESOPs) [44, 37]. Unfortunately the general ESOP minimization problem has turned out to be hard [42, 13]. A sub-class of ESOPs is obtained by flattening a PKDD into a two-level *Pseudo Kronecker Expression* (PSDKRO) [39]. In order to address the complexity problem of two-level AND/EXOR minimization we develop state of the art PKDD-based, PSDKRO minimization algorithms. Our results show that PKDD-based minimization offer an interesting alternative to general ESOP methods, as high quality results can be obtained within reasonable time bounds. The number of products in the two-level expression can be used directly as a cost measure for the final implementation. To emphasize the applicability, we show how the resulting AND/EXOR expressions can be efficiently mapped to both AND/EXOR PLAs and to fine grain FPGAs or ASICs.

Although an optimal or near optimal solution for the minimization problem can be obtained, not all functions have a compact two-level representation (and are therefore not suitable for PLA implementation). The general solution is to seek a multi-level (decomposed) representation for the function at hand. Our approach is to utilize the decomposed form inherent to DDs. In the special case of totally symmetric functions, DDs are known to form regular lattice structures, [4, 51]. The regularity of the structure is a key issue, allowing e.g., routing and delay properties of the implementation to be estimated throughout the DD minimization process. Unfortunately, totally symmetric functions are rare, therefore methods to make *Pseudo Symmetric Decision Diagrams* have been developed [35, 6, 50, 5, 46]. The complexity of the representation is highly dependent on the variable ordering and the choice of decompositions applied. We propose a number of improvements to state of the art minimization methods and show the advantages on a set of standard MCNC benchmarks. Furthermore, we introduce *Lattice Circuits*, an extension to lattice diagrams allowing additional routing resources of the target architecture to be utilized. The strict neighboring criterion of lattice diagrams is relaxed allowing the sharing of sub-functions between non-neighboring nodes, thus further reducing the circuit complexity.

Our minimization methods are based on *Pseudo Kronecker Decision Diagrams* (PKDDs) [45, 34], which are the most general type of ordered bit-level diagrams for switching functions [2] (under the condition that the negation of cofactors is utilized). In the thesis we elaborate on the properties of PKDDs and *Ternary* PKDDs (TPKDDs). We give the conditions under which TPKDDs having

complemented edges offer a canonical representation and show that local variable exchange on the diagram can be efficiently conducted while preserving canonicity. These properties enable us to devise mechanisms for their minimization, similar to those known for other DDs, [15, 38, 11]. Furthermore, we show how to utilize the redundancy of TPKDDs to derive PKDDs. By our experiments, the potential compactness of PKDDs is confirmed.

The thesis spans from validation and verification of high-level specifications all the way down to layout driven synthesis, combining logic minimization, mapping and routing to the target architecture at hand. We conclude our work to offer new means towards solving many of the crucial problems occurring along the design process of modern digital circuits.

The thesis is outlined as follows; Chapter 2 first gives a background on representation of switching functions using decision diagrams. Methods for minimization of both Ternary-PKDDs and PKDDs are developed and analyzed. We evaluate their performance on a set of MCNC benchmarks. Furthermore, a number of PKDD-based applications are presented. State of the art methodologies for layout driven synthesis are discussed in Chapter 3. Chapter 4 gives a short introduction and some comments to each paper included in the thesis. The papers are reprinted in Chapter 6, together with a reprint of the licentiate thesis in Chapter 7. Finally, the thesis is concluded in Chapter 5, where we also outline some interesting topics for future research.

Chapter 2

Pseudo Kronecker Decision Diagrams

Decision diagrams have brought new means towards solving many interesting problems (not only in VLSI CAD). Their efficiency of representation and manipulation of discrete functions is the key to their success. Many operations on e.g., *Binary Decision Diagrams* (BDDs) can be carried out in time roughly proportional to the size of the diagram(s). Also their ability to implicitly represent sets of functions have been successfully applied to many combinatorial problems in computer science. We can classify decision diagrams into to major domains; bit-level diagrams and word-level diagrams, the latter capable of representing integer valued functions, while the former essentially applies to switching functions. In this thesis we focus on bit-level diagrams and their use in the design process of digital circuits.

In this section we review the essentials of decision diagrams and highlight some important features, for an excellent overview see [27]. We investigate the properties of Ternary-PKDDs and develop an efficient minimization method based on local variable exchange. Furthermore the problem of PKDD minimization is discussed and a number of different strategies are introduced an evaluated. The section is concluded by an elaboration of the use of PKDDs in logic synthesis applications.

2.1 Decision Diagrams

Let f_0 (f_1) denote the *cofactor* of f with respect to $\overline{x}$ (x) and f_2 is defined as $f_2 = f_0 \oplus f_1$, $\oplus$ being the Exclusive OR (EXOR) operation. A Boolean function $f : B^n \rightarrow B$ can then be represented by one of the following formulae:

$$f = \overline{x} f_0 \oplus x f_1 \qquad \textit{Shannon (Sh)} \tag{2.1}$$

$$f = f_0 \oplus x f_2 \qquad \textit{positive Davio (pD)} \tag{2.2}$$

$$f = f_1 \oplus \overline{x} f_2 \qquad \textit{negative Davio (nD)} \tag{2.3}$$

Consider a rooted, *Directed Acyclic Graph* (DAG) having terminals 0 and 1 and non-terminals (internal nodes) G. Each node in G is marked with a decomposition variable x, a decomposition type $\{Sh, pD, nD\}$ and edges to cofactors $\{(f_0, f_1), (f_0, f_2), (f_1, f_2)\}$ according to decomposition type. In the following we assume each variable to occur only once on each path and in the same order for all paths (i.e., the DAG is "ordered"). Nodes having the same decomposition variable are

considered to be at the same *level* in the diagram. Diagram levels are enumerated from root (top level) towards terminals (bottom level). Furthermore, we apply the following rules of "reduction":

- There exist no two nodes expressing the same function (i.e., no two nodes are graph-isomorphic).

- There exist no Shannon (Sh) node having $f_0 = f_1$, and no Davio nodes (pD and nD) having $f_2 = 0$.

This defines a *Ordered Pseudo Kronecker Decision Diagram* (OPKDD) [45]. By applying restrictions other DDs can be defined, e.g., *Ordered Binary Decision Diagrams* (OBDDs) [4] (having Shannon (Sh) nodes exclusively), and *Ordered Kronecker Functional Decision Diagrams* (OKFDDs) [12, 9] (where the decomposition type $\{Sh, pD, nD\}$ is fixed for each variable). A single node is used to represent both f and $\overline{f}$, the latter identified by a complement attribute on the incoming edge (or more specific an odd number of complement attributes along the incoming path) [28, 3]. For a given ordering a reduced BDD offers a canonical representation of the function, under some restrictions for the use of complemented edges. Only one terminal is to be used (e.g., the 0-terminal), and complementation is only allowed to occur on one of the outgoing edges (e.g., the 1-edge). Canonicity also holds for reduced OKFDDs given an ordering and a *Decomposition Type List* (DTL) together with some restrictions on the use of complemented edges. However, PKDDs in their generalized form does not provide such a canonical representation. The DDs described above can be used also to represent multiple-output functions, e.g., by allowing each output function to be rooted arbitrarily in the *shared*-DD. This can be implemented by inferring bit-selection nodes at the top of the diagram. If the bit-selection nodes are placed at the bottom of the diagram, we obtain a representation sometimes referred to as *Multiple-Terminal Decision Diagrams*, (MTDDs). (As being able to represent integer valued functions, MTDDs can be classified as word level diagrams.) In the following we assume all DDs to be ordered and reduced if not else stated.

The following applies to Figure 2.1. Each node is marked with its respective node function, its decomposition type $\{Sh, pD, nD\}$ and its decomposition variable x_i. The line type of an edge {dashed, whole, dotted} indicates a connection to a cofactor $\{f_0,\ f_1,\ f_2\}$, ($f_2 = f_0 \oplus f_1$) respectively. For the sake of clarity, edges are sometimes also marked with its corresponding literal. Furthermore, the diagrams are left unreduced, and complemented edges are not applied. Figure 2.1 together with the examples below highlights some important features.

Example 1 *Figure 2.1 (a) shows a BDD for the parity function* $f(x_1, x_2, x_3) = \overline{x}_1\overline{x}_2x_3 \oplus \overline{x}_1x_2\overline{x}_3 \oplus x_1\overline{x}_2\overline{x}_3 \oplus x_1x_2x_3$.

Example 2 *The DD in Figure 2.1 (b) (representing the same function as (a)), has a (pD) decomposition for node* f*, (Sh) decompositions for nodes* f_0 *and* f_{00} *and a (nD) decomposition for* f_{01}*. As the decomposition types are mixed* $\{Sh, pD, nD\}$ *and not fixed for each variable (e.g.,* x_2*) it is a PKDD.*

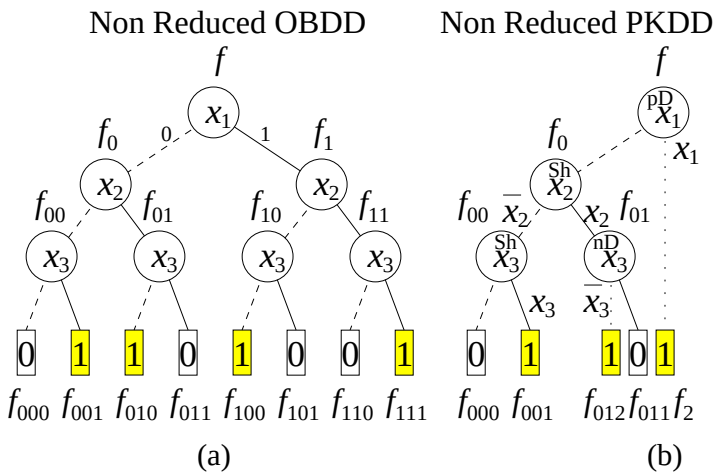

Figure 2.1. Decision Diagrams.

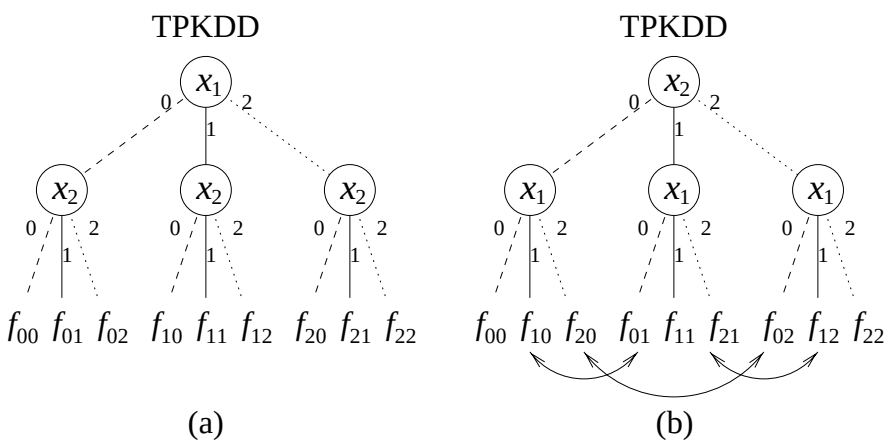

Figure 2.2. Local variable exchange on a TPKDD.

2.2 Ternary Pseudo Kronecker Decision Diagrams

A *Ternary Pseudo Kronecker Decision Diagram* (TPKDD) (also known as canonical *Exor Ternary Decision Diagram* (ETDD) [43]) is a DD according to Section 2.1, with the following exceptions; each node in G has three outgoing edges to cofactors f_0, f_1 and $f_2 = f_0 \oplus f_1$. The same reduction rules as for BDDs apply. In the following we assume diagrams to be ordered and reduced. Although a TPKDD offers a canonical representation (under a given variable order) it is a redundant representation. Given two out of the three outgoing edges, we can always derive the third one by a simple EXOR operation, see Section 2.1. However, in Section 2.7 we will show how this redundancy can be utilized towards finding a good PKDD representation.

As for other DDs, the complexity of the representation depends heavily on the variable order. Therefore, it is crucial to find a good ordering for the function at hand. Heuristic methods have proven successful solving this problem for both BDDs and OKFDDs. The most efficient methods for BDD minimization are based on local variable exchange, "sifting" two neighboring levels in the diagram [15, 38, 11]. The key to their success is the property that such a level exchange is indeed a local operation. I.e., nodes belonging to levels above and below the sifted levels are unaffected by the operation. Furthermore, the number of nodes on a fixed level is unaffected by the ordering of the nodes above and below this level. This property has also shown to improve the efficiency of exact BDD minimization methods [14, 17, 10].

Let us consider the TPKDDs shown in Figure 2.2. (a) shows the initial representation while (b) shows the representation after exchanging the neighboring levels, i.e., (a) has variable order (x_1, x_2) and (b) has (x_2, x_1). We observe that no new cofactors needs to be computed, and that a level exchange can be performed solely by the redirection of edges. The arrows in the figures show the interchange between cofactors applied. The correctness of this may not be obvious, but by simple calculations we prove that this is the case: Consider the three leftmost cofactors of figure (b). The cofactor f_{00} is obviously left unchanged ($f|_{x_1=0,x_2=0} = f|_{x_2=0,x_1=0}$). The next cofactor f_{10} is to be found in (a) by tracing the 1-edge of the top node and the 0-edge of the successor. Now, to the interesting part, cofactor f_{20} (as found in (a) by tracing the 2-edge and 0-edge respectively). We now need to show that $f_{20} = f_{00} \oplus f_{10}$. This becomes obvious as $f_2 = f_0 \oplus f_1$ (by the definition in Section 2.1). Applying further co-factorization w.r.t., $x_2 = 0$ preserves the equality (as the functions are the same). Verifying the remaining cofactors can be done accordingly. The above TPKDD properties lead us to believe that efficient algorithms for dynamic reordering of TPKDDs can be devised.

2.3 Complemented Edges

Complemented edges applied to DDs, have shown not only to reduce diagram sizes but also to improve computational efficiency, [28, 3]. However, as canonicity of the representation is crucial to many DD applications, we must first make certain that the required conditions are met.

In Figure 2.3 an edge marked by a filled dot indicates the complementation of the corresponding cofactor, (negative *polarity*). Each row represents the same function, interpreted as TPKDDs, (shown in the leftmost column), and PKDDs (shown in the three rightmost columns). To obtain a canonical TPKDD we have to consistently choose one of the two possible representations for each class of functions (i.e., row). In our case we have chosen the representations marked *Canonical*. In the following we will prove that evaluation of the diagram is consistent under the proposed

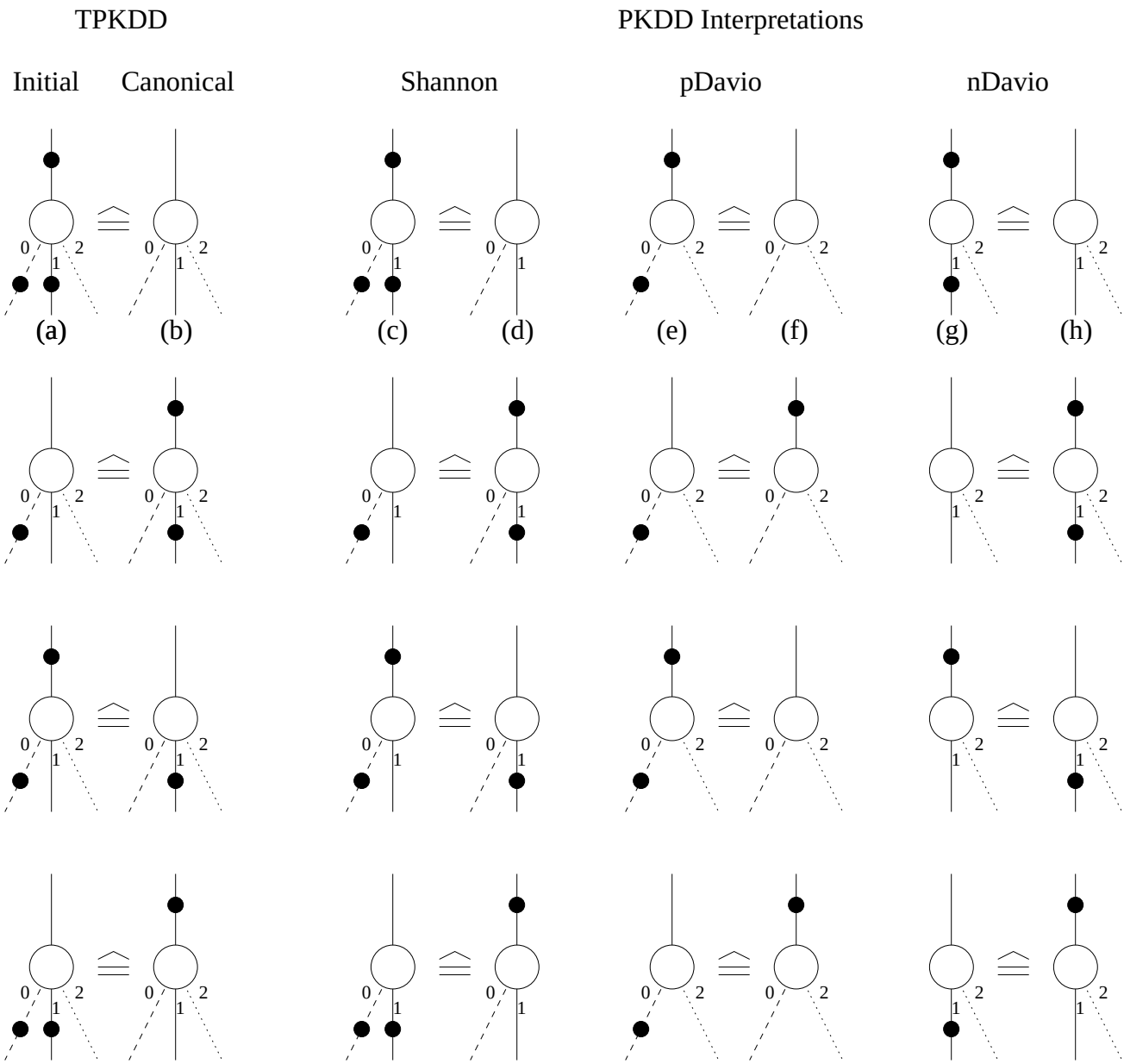

Figure 2.3. Complemented edges for TPKDDs.

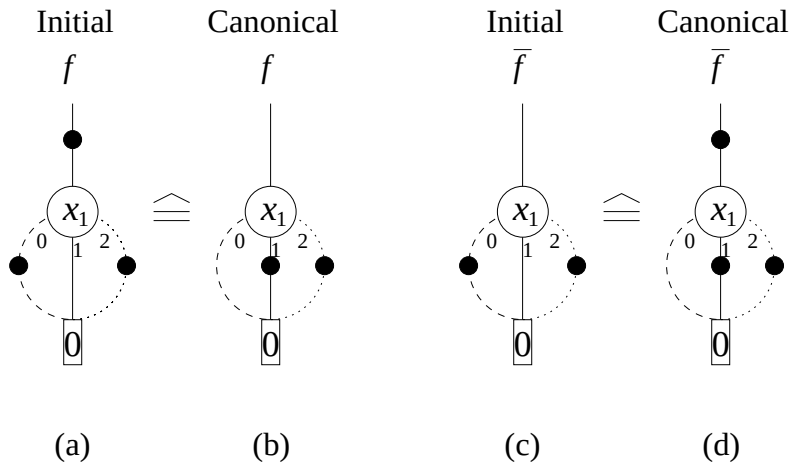

Figure 2.4. Canonical TPKDD representation.

transformations:

Consider the top row in Figure 2.3. The function interpreted as a (non-canonical) Shannon decomposition (c), is $f = \overline{\overline{x}\overline{f_0} \oplus x\overline{f_1}} = \overline{\overline{x}\overline{f_0}} \cdot x\overline{f_1} + \overline{\overline{x}\overline{f_0}} \cdot \overline{x\overline{f_1}} = (x + f_0)(\overline{x} + f_1) = \overline{x}f_0 + xf_1 + f_0f_1 = \overline{x}f_0 + xf_1$, which is obviously correct w.r.t. the canonical representation shown (d). Now consider the (non-canonical) positive Davio representation (e) $f = \overline{\overline{f_0} \oplus xf_2} = f_0 \oplus xf_2$, which is the function defined by the canonical representation (f). From the definition in Section 2.1 this is equivalent to the Shannon decomposition shown in (d). For clarity we carry out the computation, $f = f_0 \oplus xf_2 = f_0 \oplus x(f_0 \oplus f_1) = (f_0 \oplus xf_0) \oplus xf_1 = (\overline{f_0}xf_0 + f_0(\overline{x} + \overline{f_0})) \oplus xf_1 = \overline{x}f_0 \oplus xf_1$. To complete the proof for the top row, let us consider the (non-canonical) negative Davio decomposition (g), $f = \overline{\overline{f_1} \oplus \overline{x}f_2} = f_1 \oplus \overline{x}f_2$, corresponding to the canonical representation (h). From the definition in Section 2.1 this corresponds to the Shannon decomposition shown in (d). Verifying the remaining cases (rows 2,3 and 4) can be done accordingly.

By this we have given the necessary conditions under which TPKDDs offer a canonical representation. The rules applied are essentially the same as those used for BDDs, as the third edge (f_2) neither is used in the canonicity criteria nor is affected by the transformations. The rules can be summarized into a single statement; allow only a single terminal (e.g., the 0-terminal), and never allow complementation of the 0-edge. Note that the third edge (f_2) may or may *not* be complemented, as determined from its corresponding canonical representation.

Example 3 *Consider the TPKDD representation of function $f = x_1$, shown in Figure 2.4 (a). By the 3rd-row transformation from Figure 2.3 we obtain the canonical representation (b). Now consider the function $\overline{f} = \overline{x_1}$, shown in Figure 2.4 (c). The 2nd-row transformation applies and we obtain the canonical representation (d). We observe that the graphs are isomorphic, only the complementation on the incoming edge differs between f and $\overline{f}$.*

Note, for the example above, without applying the transformations, all combinations (a,c), (a,d), (b,c), and (b,d) could occur. Only two out of four express isomorphism. Hence, complemented edges together with the canonicity criteria reduce the diagram size.

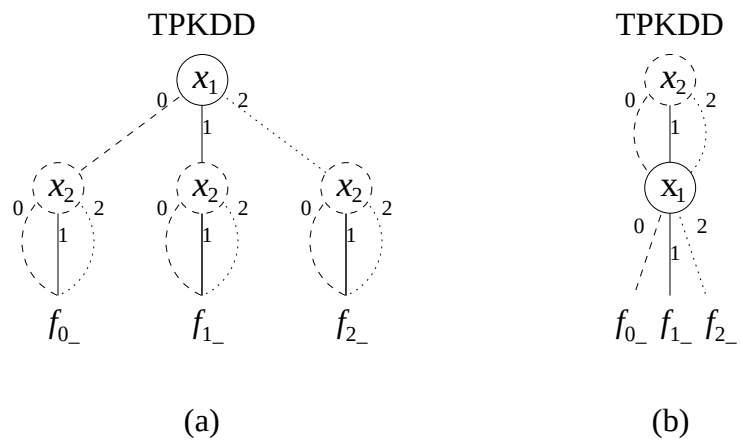

Figure 2.5. TPKDD sifting, reducing the number of cross-point nodes.

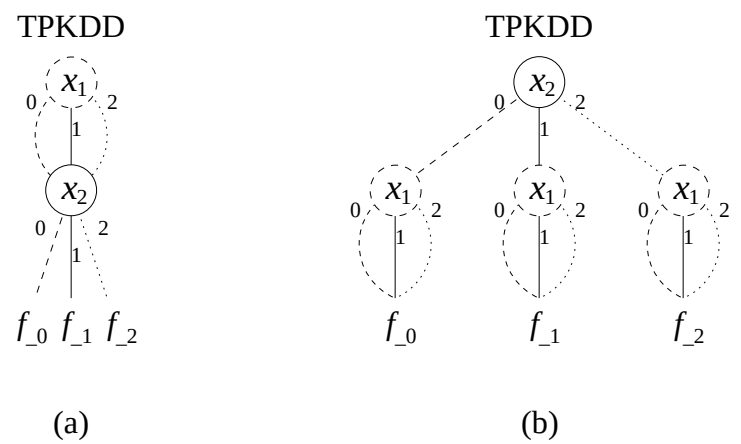

Figure 2.6. TPKDD sifting, increasing the number of cross-point nodes.

2.4 Sifting Operations on TPKDDs

As shown above, sifting levels in a TPKDD can be done solely by local operations on the diagram. This however is true *only* in the case where edges occur exclusively between neighboring levels. This is not really a problem (the number of nodes affected by the sifting is still bounded by the number of incoming edges). However, (for the sake of simplicity), we ensure to have edges exclusively between neighboring levels. That is, the diagram is *quasi reduced*, having *cross-point* nodes f, iff $f_0 = f_1$. Furthermore we assume all functions are rooted at the top level. In order to device a sifting-based dynamic reordering algorithm for PKDDs, let us first consider a number of special cases, exemplified below.

Example 4 *Consider the TPKDDs shown in Figure 2.5, (under the assumption that the diagram quasi reduced). The lower nodes in (a) have a dashed outline, and represent cross-point nodes, i.e., mere connections to the next level in the diagram. After sifting levels, we obtain the PKDD (b), having a single cross-point node. Although it might look like we have done a reduction of the*

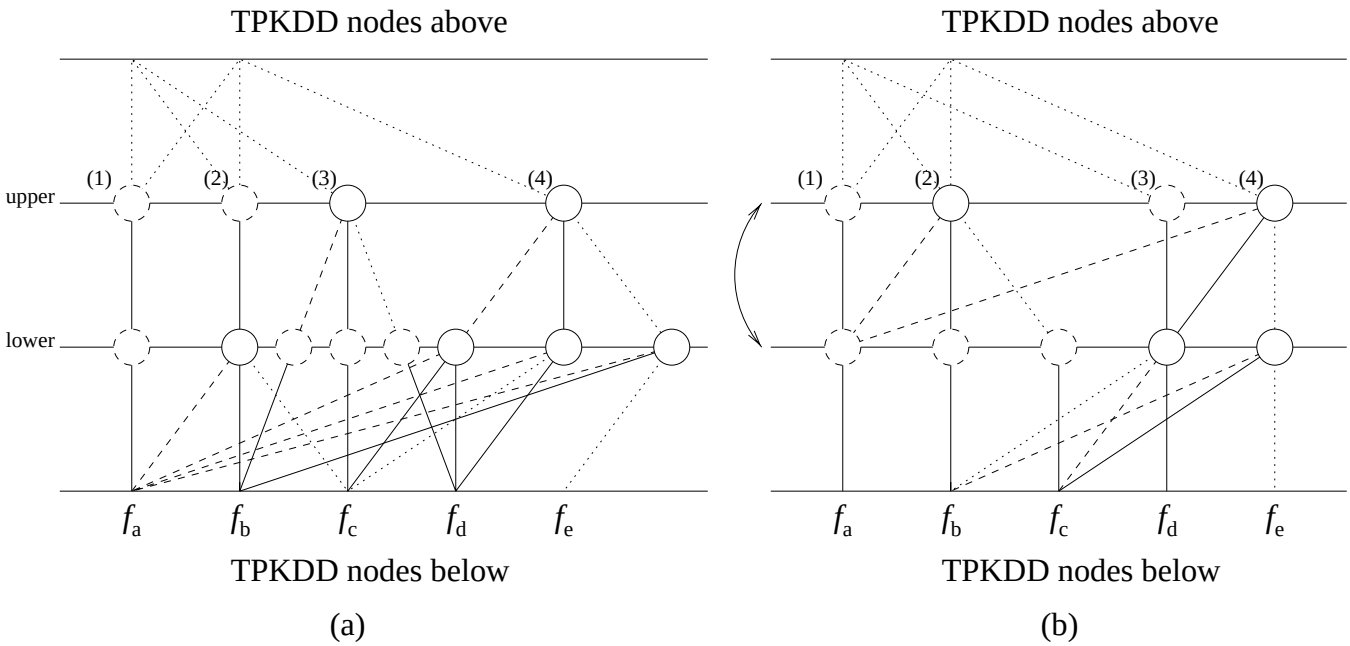

Figure 2.7. TPKDD sifting by edge redirections.

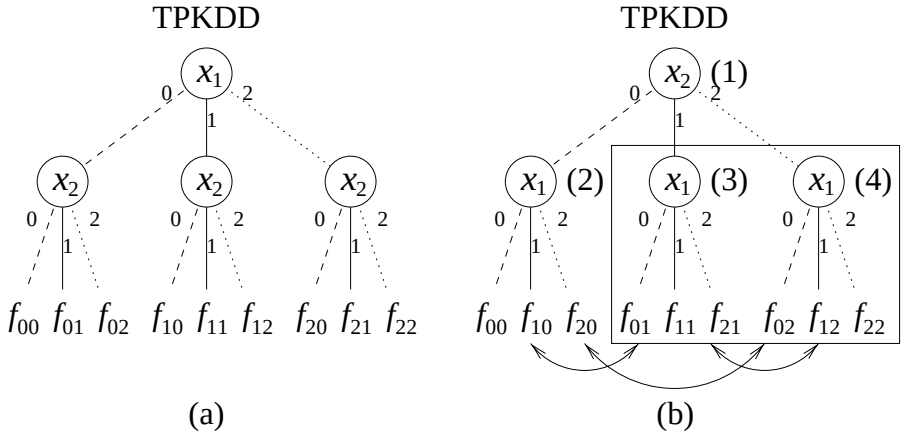

Figure 2.8. Sifting of PKDDs having complemented edges.

TPKDD this is not really the case. A fully reduced TPKDD will have exactly the same number of nodes. The complementary situation is shown in Figure 2.6, where the number of nodes seems to be increased.

From the above example it is obvious that we will have to keep track of the number of "real" nodes (i.e., non cross-point nodes) in the TPKDD. Before going into details with the dynamic reordering algorithm, let us first take a look at another slightly larger example.

Example 5 *Figure 2.7 shows two levels in a TPKDD. (In order to avoid cluttering of the figure, only one edge for cross-point nodes is show.) Consider the initial representation (a) in figure. (1) shows a connection from upper nodes to lower nodes and is unaffected by the sifting. Node (2) corresponds to Figure 2.6 increasing the number of connecting nodes, while node (3) shows the opposite, as described by Figure 2.5. Node (4) represents the general case, having a full TPKDD as shown in 2.2. After sifting the number of real TPKDD nodes is reduced for the* lower *level (from 4 to 2). Note, that the number of real nodes is the* upper *level may also be changed. This occurs in the case that the number of (2) and (3)-like nodes are unequal (in the* upper *level). The key property of local operation holds, nodes above and below the sifted levels are left unchanged.*

2.5 Sifting of TPKDDs having complemented edges

From Section 2.3, we have the necessary conditions under which TPKDDs using complemented edges offers a canonical representation. To be able to utilize complemented edges we need to ensure that canonicity is preserved by the local variable exchange operation. Let us start by making the following observations:

- Nodes below the levels sifted are unaffected by the level exchange operation, i.e., the cofactors below remains intact, see Figure 2.8. That is, edges connecting to those cofactors will preserve their polarity.
- The nodes above the *upper* level from Figure 2.7, are unaffected (under the condition that the diagram is quasi reduced).

Therefore, we need only to consider the canonicity criteria for the two levels about to be sifted. Assume that the TPKDD shown in Figure 2.8 (a) is canonical. After the level exchange, node (2) is still canonical, as canonicity is defined from the polarity of the 0-edge. The edge to f_{00} is un-complemented (positive), since the initial TPKDD (a) is canonical, thus node (2) is also canonical. Now consider nodes (3) and (4). In these cases we have to apply the canonicity criteria, as their 0-edge polarities might change (edges pointing to f_{01} and f_{02} respectively. For node (1) this implies possible polarity changes only on the 1-edge and the 2-edge, thus also node (1) will remain canonical.

By this we have shown that sifting levels in a TPKDD having completed edges is a truly local operation. It can be performed solely by the redirection of edges together with some possible edge polarity changes. Using common BDD techniques, the TPKDD nodes can be kept in a *unique table* (UT). Before creating a new node, the reduction and canonicity criteria are applied and a look-up in the UT is performed. By this, we ensure the diagram to be reduced (or quasi reduced as in our case) and canonical at all times. The UT look-up can be performed almost in constant time, using e.g., hashing techniques known from efficient BDD packages.

```
dynamic_reordering {
1  compute initial cost
2  for each variable; find initial position
3  sift towards top; keep track of best position
4  sift all way to bottom; keep track of best position
5  apply best position; repeat 2 until no further improvement
}
```

Figure 2.9. Dynamic reordering of TPKDDs.

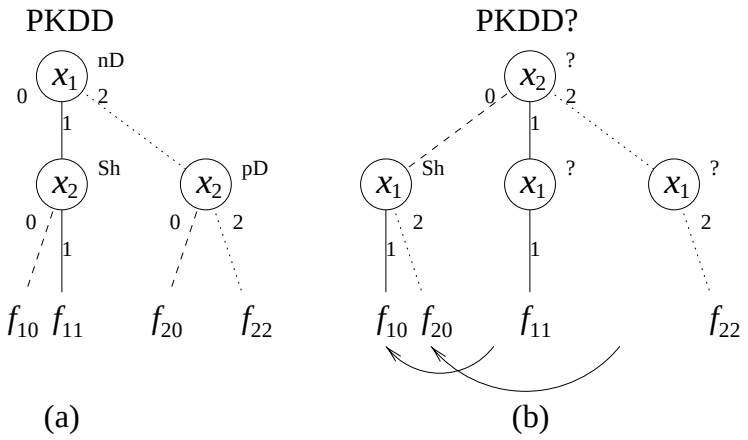

Figure 2.10. Local variable exchange on a PKDD.

2.6 Dynamic Reordering of TPKDDs

The best known algorithms for finding the optimal order for non-symmetric DDs all have exponential worst-case behavior in the number of input variables, see e.g. [14, 10, 16]. Therefore, we have to settle for heuristic approaches. Utilizing the properties shown in Sections 2.2, 2.4 and 2.5, we have developed a dynamic reordering algorithm similar to [15, 38]. The algorithm is simple, iteratively sifting each variable to the position where the number of real TPKDD nodes is minimized. The iteration is aborted when no further improvement is obtained. Our algorithm is outlined in Figure 2.9.

We have also tried a slightly different strategy. By greedily choosing the variable (to sift) corresponding to the level having largest number of real nodes. However, our experimental results have shown the above simple algorithm to produce the best results. Based on the property of local variable exchange more sophisticated reordering techniques known for BDDs can be directly transferred, [33, 29, 32, 47]. We have evaluated our prototype implementation on a number of MCNC benchmarks. The experimental results can be found in Section 2.9.

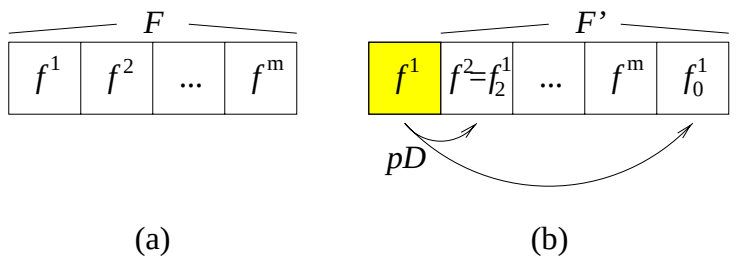

Figure 2.11. A simple algorithm for PKDD minimization.

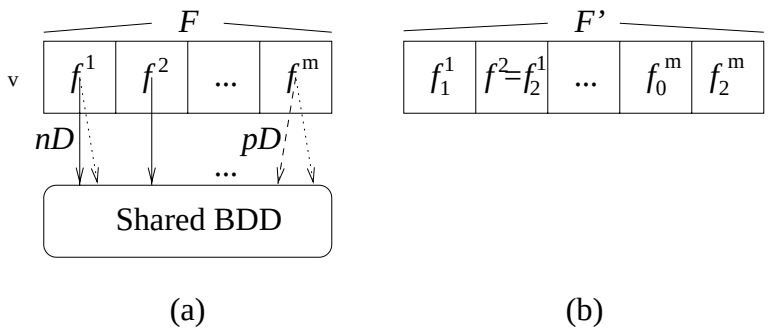

Figure 2.12. Backtracking algorithm for PKDD minimization.

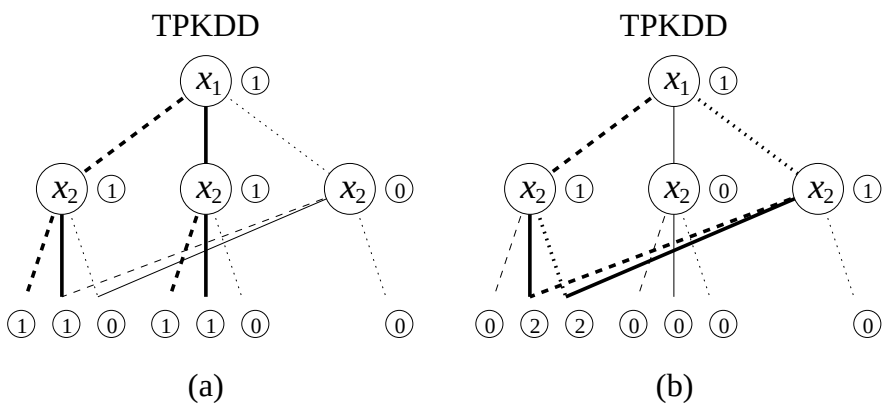

Figure 2.13. TPKDD-based minimization.

2.7 Heuristic Minimization of PKDDs

Diagram size (compactness) is a sought after property, crucial to many CAD applications. As shown in [2] three decomposition types $\{Sh, pD, nD\}$ together with complemented edges, cover all bit-level decompositions. Therefore, PKDDs are the most general bit-level DD representation of switching functions, as we can for each node chose decomposition type arbitrarily. In that, we give up the property of canonicity in order to reduce the diagram size.

The complexity of the PKDD representation is dependent on both the ordering variables and the decomposition type applied to each node. The number of possible PKDDs for a function of n variables is $n!3^{2^n-1}$ [43]. Having this huge search space leave us in despair finding an exact solution. But, if we settle for a good solution we may approach the problem by heuristic methods.

Let us take a step back and consider the following:

- Firstly, there exists efficient algorithms for BDD minimization.
- Secondly, BDDs (as being a very special case of PKDDs) can be used as an upper bound estimate for PKDD complexity.

Based on those facts, let us first device a naive algorithm for PKDD generation, to see if we are on the right track.

A Store the functions (as BDDs) to implement in a list F. Take the first function f^1, and compute the cofactors f_0^1, f_1^1 and f_2^1 w.r.t, f_v^1 (f_v^1, being the decomposition variable for the top node in f^1). From the definition in Section 2.1, only two out of the three cofactors are needed to represent the function. We greedily chose the combination of two (i.e., decomposition type) from the sharing cost with the remaining functions in F'. Note, this is not a measure merely based on the cofactors at hand, but rather an estimate of the complexity to implement all remaining functions (including the selected cofactors). (As being based on BDD size, it is clearly an overestimate of the PKDD cost.) We can now remove f^1 from the list. According to the decomposition type selected we search the list for functions matching (equal to or complement of) the chosen cofactors. Finally, the unmatched cofactors are appended to the list F'. Now, we attend the next element (f^2) from F' in the same manner. E.g., Figure 2.11, (a) shows the initial list F, and (b) the resulting list F' after assigning a pD decomposition for f^1. Repeat the procedure until the list is empty. Convergence of the algorithm is guaranteed, as the cofactors selected depend on one less variable until eventually being constants $\{0, 1\}$. This method is somewhat similar to the heuristic minimization method for Free-BDDs presented in [16].

Our experimental results (presented in Section 2.9) show significant reductions of the number of nodes required compared to the initial BDDs. However, the approach is greedy and provides no backtracking. This tremendously simplifies the minimization procedure at the cost of quality. The method is indeed very fast having worst case complexity $O(|F|^3)$, $|F|$ being the number of nodes in the initial BDD representation F. That is, for each node in F we have in the worst case to try out three different decompositions. If a Davio node is chosen, that is due to a reduction of the number of nodes remaining in F', and therefore also a reduction of the complexity. In each step an EXOR computation is performed,

which is worst case order $O(|F|^2)$, giving us the overall complexity $O(|F|^3)$, (assuming that accounting the BDD cost is done in constant time). The quality gain obtained by the simple and fast algorithm, confirms our expectations and encourage us to develop a method incorporating backtracking facilities.

B Based on the minimization approach presented above, we store the top nodes in a list F. According to the BDD order we choose a decomposition variable v. For each of the functions in the list we seek the best decomposition type w.r.t. v. The method differs in that we apply the cost measure *after* decomposing all functions in F. This allows us to either exhaustively or greedily explore the decomposition search space for the variable under consideration. For exhaustive exploration of decompositions for a variable v, the complexity is $O(3^{|F_v|})$, $|F_v|$ being the number of functions in F having v in their support set ($f|_{v=0} \neq f|_{v=1}$). Obviously this becomes intractable for larger $|F_v|$. Therefore, we propose a greedy heuristic consisting of the following steps:

Initiate Decompose all nodes w.r.t. each single (fixed) decomposition type $\{Sh, pD, nD\}$. Choose the best decomposition (reducing the shared BDD cost) as a starting point for *Iterate*. (This corresponds to the fixed decomposition type for each variable in an OKFDD.)

Iterate Now we visit each function (node) having v in its support and greedily choose the best decomposition type for the function under consideration. We now repeat *Iterate* until no further improvement is obtained.

This iteration obviously terminates. The decomposed functions are now removed from F and replaced by their cofactors corresponding to decomposition types chosen. Now proceed with the next decomposition variable according to the BDD order. Figure 2.12, shows an example of the algorithm. The least BDD cost is given by the decompositions $\{nD, -, \ldots, pD\}$, (w.r.t. to the chosen decomposition variable v), (a) in figure. The resulting F' is shown in Figure 2.12 (b).

Experimental results, (presented in Section 2.9), show the benefits from this second approach. The exhaustive search for decomposition types is applied when $|F_v| < |F_v|_{limit}$, $|F_v|_{limit}$ set to 12, keeps run-times practical, i.e., within CPU seconds. Our experiments show that the greedy approach performs rather well compared to the exhaustive method, while being significantly faster. We were also tempted to apply a simple reordering strategy. Instead of choosing decomposition variable according to BDD order, we tried all (remaining) support variables as the decomposition variable v. For each v we sought the best decomposition types using the above described method while keeping track of the best result obtained. Finally, the best variable and decomposition types were selected and we could proceed with the remaining variables. This method however, did not produce consistent results. We assume the problem to stem from the fact that the BDD forms an upper bound on the PKDD only under the condition that they share the same variable order. I.e., if we derive a PKDD having a different variable order (greedily chosen top-down) we may or may *not* obtain a smaller diagram. Therefore we disregard the simple reordering method. Both our first method and the second method described above lacks two important features; they

sport no stable way of improving on the initial variable order, and they both use the upper bound BDD size as a cost measure for selecting decomposition type.

C Let us start out from a TPKDD representation. This data structure implicitly holds *all* possible PKDD representations under a given variable order (including both the worst and best representation). We will now try to exploit the redundancy of the TPKDD representation observed in Section 2.2. Finding the best representation (under this order) is $O(3^{|G|})$, $|G|$ being the number of nodes in a reduced TPKDD, once again an intractable problem. Choosing decomposition type for a node *activates* two out of three outgoing edges (cross-point nodes used for connecting levels can be considered to have only one outgoing edge). Only nodes on active paths are part of the PKDD encoded in the TPKDD. Figure 2.13 show examples of TPKDDs encoding PKDDs. Active edges are marked with fat lines. Keeping track of active nodes are done by the use of reference counters (shown as circled numbers in the figure), accounting the number of active incoming edges for each node. By this, we can change the decomposition type of any node in the TPKDD while updating the number of PKDD nodes required. If the reference counter for a node g is decreased to 0, visit the active successors (according to decomposition type) of g and decrease their values by 1. The complementary situation occurs if a counter value is increased from 0 to 1, in which case we have to visit the corresponding active successors increasing their value by 1. (Both increasing and decreasing reference counters might cause a recursive traversal of the TPKDD.) Note, only the decomposition type of active nodes will directly affect the number of PKDD nodes.

We have developed a minimization method featuring a parameterized look-ahead technique. A window can be set defining the scope (upper, and lower diagram level limits) in which we explore different decomposition types. Furthermore, we can set the recursion depth of the look-ahead. E.g., a depth of d will consider the decomposition types of any combination of d nodes (within the scope) simultaneously. The method is greedy in that once a reduction is obtained we apply the corresponding decomposition. We repeat the procedure until no further improvements are obtained. In order to reduce the complexity our method considers only the decomposition types for currently active nodes in the scope.

Figure 2.13 (a) shows an initial solution (encoding a BDD), having 7 active nodes (i.e., nodes having, non-0 reference counters). After applying the minimization we obtain a PKDD having only 5 nodes. Note, by a simple approach without look-ahead, the reduction obtained by assigning a pD decomposition to the top node could not be foreseen. Our method is somewhat similar to the one presented in [45] , but to the best of our knowledge, both the the scoping and look-ahead technique is novel. However, the method in [45] uses some sort of simulated-annealing method to derive a suitable variable order under which the PKDD is minimized.

As a starting point we can choose any initial DD representation, e.g., BDD, OKFDD or PKDD. In Section 2.9, experimental results starting from minimized BDDs and OKFDDs confirm our expectations. Furthermore, we give a comparison of the proposed algorithms to other state of the art DD minimization methods.

2.8 Applications of PKDDs

In the previous sections we have shown methods to manage and minimize both PKDDs and TPKDDs but we have mentioned little or nothing why we have an interest in doing so. From Section 2.7 we conclude that PKDDs *may* offer more compact representation of switching functions than any other ordered bit-level decision diagram. This, at the expense of sacrificing diagram canonicity. On the other hand, TPKDDs have been shown to offer a redundant, canonical representation. Their ability to implicitly hold all possible PKDD representations can be used e.g., to efficiently derive PKDDs and Pseudo Kronecker expressions. In the following we will briefly review a number of applications based on TPKDDs and PKDDs.

The compactness of PKDDs (compared to other DDs) has proven useful. In [45] , a method for FPGA synthesis has been proposed. Each DD node is shown to map directly onto a *Look-Up Table* (LUT) cell, and the edges in the DD maps to interconnections between cells. Another feature of DDs, is their ability to provide two-level expressions by flattening the diagram. This is done by tracing all paths leading to a true evaluation of the diagram. This method has been successfully applied e.g., in [40, 41] (using an ETDD representation) and in [8] (proposing a BDD-based approach). Three of the included papers, are devoted to further improvements to these approaches, [22, 23, 20]. Furthermore, from a PKDD, a factored form can be derived, as each node actually represents a factor in the expression. In [19] , this property is used to derive so called Maitra terms. The resulting expressions are shown to efficiently map onto fine-grain FPGAs. We generalize this approach and apply multi-valued PSDKRO minimization to derive the factored expression. In that our method applies to LUT FPGAs. In Chapter 3, we will further elaborate on applications of PKDD-based minimization methods in layout driven synthesis.

2.9 Experimental results

Our algorithms are programmed in a prototype package, utilizing the CUDD 2.3.0 [48] package for BDD operations. Custom data types are used to represent TPKDD and PKDD nodes. Complemented edges are used for both BDDs and TPKDDs. The max number of nodes at one level is set to 1000. The size of DDs is measured by the number of non-terminal nodes.

The first experiment, Table 2.1, shows the effect of dynamic reordering on TPKDDs. Left columns show number of nodes starting from a naive variable ordering (as given by the initial PLA benchmarks), while right columns show the number of nodes starting out from a minimized BDD. For the BDD minimization we have applied the "group sifting with convergence", as implemented in the CUDD package. Numbers in "( )", show the size of minimized PKDDs starting from the corresponding TPKDD, as further discussed in the final experiment below.

Our experimental results show great improvement to the naive ordering, e.g., almost 15 times node reduction for "add6". We observe that a minimized BDD (having a suitable BDD variable order), often provides a plausible starting point for dynamic TPKDD reordering.

The second experiment (shown in Table 2.2), gives a comparison of the heuristic minimization methods for PKDDs proposed in Section 2.7. Column *BDD* gives the number of minimized BDD nodes using CUDD "group sifting with convergence". Columns *PKDD (A)*, *PKDD (B)* and *PKDD (C)* show the results from our minimization methods. Numbers shown to the left in column *PKDD (B)*, are derived by setting $|F_v|_{limit}$ to 1, i.e., the greedy approach is always applied. Setting $|F_v|_{limit}$ to 12, (enabling exhaustive search if $|F_v| < 12$), gives the results shown to the

		Naive Ordering			Minimized BDD		
name	in/out	*BDD*	*TPKDD org*	*TPKDD dyn*	*BDD*	*TPKDD org*	*TPKDD dyn*
5xp1	7/10	74	146	48 (31)	42	49	48 (31)
add6	12/ 7	309	1440	103 (62)	29	28	28 (23)
exp	8/18	210	571	422 (177)	170	454	370 (134)
in2	19/10	2361	-	-	235	1015	760 (176)
in7	26/10	235	701	177 (84)	80	265	144 (71)
inc	7/ 9	77	158	136 (63)	71	167	144 (62)
sao2	10/ 4	155	428	185 (69)	81	186	185 (69)

Table 2.1. TPKDD dynamic reordering.

right. In column *PKDD (C)*, we show the results obtained by the TPKDD-based minimization method. For the experiment, we have applied two different parameter settings. To the left, both the scope size and recursion depth are set to 1. I.e., for each iteration, nodes belonging to a single level is considered individually. To the right, scope size and recursion depth are set to 2. I.e., for each iteration two neighboring levels are considered, (scope set to 2). Nodes belonging two those levels are pairwise decomposed, (recursion depth set to 2). Columns *OKFDD [36]* and *PKDD [45]*, give comparison to state of the art results. (The OKFDD results were obtained on the PLA benchmarks suite, and differ in cases from the results presented in [36] based on the BLIF benchmarks; the discrepancies are under investigation.) Numbers in "( )" show the initial BDD size. As the approach from [45] does not utilize complemented edges, the number of nodes should be at least as large as in column *BDD*. This is not always the case. However, if don't care information of the functions is utilized to minimize the initial BDDs, the numbers presented may prove to be correct. In any case, we cannot claim to make a fair comparison to the results from [45].

This experiment confirms our expectations of PKDD compactness, compared to the initial BDDs. The TPKDD-based approach *PKDD (C)* obtains the best results of our proposed heuristics. (As mentioned, it is unclear if comparison to PKDD results from [45] is fair.) The experiment show that the greedy approach in *PKDD (B)* perform (almost) as well as the exhaustive method. The OKFDDs, [36], are derived by a genetic algorithm, simultaneously seeking suitable ordering and decomposition types. The quality of the results are excellent, see e.g. "apex7" and "duke2", at the expense of CPU resources. However, even though our approach *PKDD (C)* does not consider variable reordering, the flexibility of PKDD representation allows better solutions in some cases, e.g., "bw", "clip" and "tial".

In the third set of experiments (shown in Table 2.3), we show the effect of starting point for PKDD minimization. Once again, column *BDD* gives the number of minimized BDD nodes using CUDD "group sifting with convergence". Column *OKFDD [9]* gives the number of nodes after sifting-based reordering, [9]. For the PKDD minimization we have applied method *C* from Section 2.7 with scope and recursion depth set to 2. The derived PKDDs show significant improvements to their initial representations. In Table 2.1 we derive PKDDs starting from minimized TPKDDs. Also in this case we can improve on the initial DD sizes. We conclude that finding a good starting point for PKDD minimization is crucial to the quality of the final result.

			Proposed Heuristics			Other results	
name	in/out	*BDD*	*PKDD (A)*	*PKDD (B)*	*PKDD (C)*	*OKFDD [36]*	*PKDD [45]*
5xp1	7/10	41	34	29/28	32/28	28	33 (68)
9sym	9/1	24	25	24/24	24/24	24	26 (33)
add6	12/ 7	28	28	23/23	23/23	23	23 (47)
alu2	10/ 6	86	84	80/79	81/75	68	59 (61)
apla	10/12	105	100	97/97	96/94	105	-
apex7	48/37	289	280	268/268	271/268	186	-
bc0	26/11	522	495	469/469	474/469	424	-
bw	5/28	97	95	85/85	86/83	91	90 (108)
chkn	29/7	267	258	256/256	255/255	242	-
clip	9/5	74	71	66/66	66/64	72	69 (97)
co14	14/1	26	27	26/26	26/26	26	26 (27)
con1	7/2	17	15	14/14	14/14	13	14 (15)
dc2	8/7	61	57	56/55	56/55	61	54 (64)
dist	8/5	120	117	113/113	112/109	120	127 (152)
dk17	10/11	92	91	89/89	89/85	84	62 (63)
dk27	8/9	61	57	56/56	56/56	54	25 (22)
duke2	22/29	355	347	341/341	341/338	286	269 (373)
exp	8/18	169	154	148/148	149/144	158	-
f51m	8/8	38	31	26/26	30/26	25	28 (67)
in2	19/10	234	208	190/190	191/187	171	-
in7	26/10	79	80	77/77	77/77	67	-
inc	7/9	70	69	66/66	66/65	67	58 (70)
intb	15/7	537	424	398/398	411/386	385	-
misex1	8/7	35	35	33/33	33/32	32	31 (36)
misex3	14/14	520	501	483/483	481/467	484	-
misj	35/14	39	40	39/39	39/39	40	43 (43)
mlp4	8/8	134	108	107/107	107/103	107	99 (141)
rd53	5/3	16	16	13/13	13/13	13	13 (23)
rd73	7/3	30	28	26/26	26/25	21	21 (43)
rd84	8/4	41	37	36/36	36/46	29	29 (59)
risc	8/31	65	62	60/60	60/57	56	57 (67)
pdc	16/40	602	561	544/544	542/538	564	-
sao2	10/ 4	80	74	69/69	69/66	79	69 (85)
sex	9/14	41	40	38/38	38/38	37	38 (47)
spla	16/46	590	537	526/526	526/525	562	-
t481	16/1	20	21	18/18	18/18	16	17 (32)
tial	14/8	579	431	360/360	362/357	474	381 (690)
ts10	22/16	165	166	165/165	165/165	123	132 (142)
vg2	25/8	80	81	80/80	80/77	83	-
x6dn	39/5	240	221	214/214	215/213	210	-
xor5	7/10	5	6	5/5	5/5	5	5 (9)

Table 2.2. PKDD minimization.

		BDD Order		OKFDD Order	
name	in/out	*BDD*	*PKDD (C)*	*OKFDD [9]*	*PKDD (C)*
5xp1	7/10	41	31	28	28
add6	12/ 7	28	23	23	23
bc0	26/11	523	469	431	412
duke2	22/29	355	338	300	274
exp	8/18	170	147	158	134
in2	19/10	234	187	164	147
in7	26/10	79	77	64	64
inc	7/ 9	70	65	66	62
intb	15/ 7	538	386	480	366
misex3	14/14	520	426	583	491
sao2	10/ 4	80	68	80	73
tial	14/ 8	580	359	550	359
vg2	25/ 8	80	80	175	164
x6dn	39/ 5	241	213	215	202

Table 2.3. PKDD minimization from different orderings.

Chapter 3

Layout Driven Synthesis based on PKDDs

In VLSI CAD, predictability of properties (e.g., path delays) for the final implementation is a key issue. Breaking down the design process into individual steps like, logic minimization, mapping and routing makes this a tremendously hard task. One solution is to over-estimate the cost measure at each step, trying to keep on the "safe side". Clearly, this is sub-optimal, e.g., the tools might mislead us to choose faster and more expensive target technologies than actually needed. To improve on the accuracy of the cost measures, the synthesis process has to be iterated using back annotated information from downstream tools. As an alternative, different methodologies for layout driven synthesis has been proposed in to overcome the problems stated.

The idea behind these approaches is to enforce restrictions on the synthesis tools such that some form of regularity of the implementation is ensured. In the following we will outline a number of layout driven synthesis methodologies where PKDDs have been successfully applied.

3.1 Two-Level Synthesis

Synthesis for AND/OR-based arrays (PLAs) [31] constitutes a classic example in the area of layout driven synthesis. The minimization problem involved is well understood and efficient tools have been developed over the years [7]. However, it is well known that certain functions are better represented as AND/EXOR Expressions (ESOPs) [44, 37]. Unfortunately the general ESOP minimization problem has turned out to be hard [42, 13]. A sub-class of ESOPs is obtained by flattening a PKDD into a two-level *Pseudo Kronecker Expression* (PSDKRO) [39]. In [22, 23, 20] we have developed state of the art PKDD-based, PSDKRO minimization algorithms. By pruning techniques, a good estimate of the PSDKRO cost can be efficiently derived. This allows us to investigate different variable orderings for the PSDKRO expression, [20]. After a good ordering is obtained, we can further minimize the PSDKRO expression using an exact method similar to the one presented in [40]. This method is exact for *Single Output* (SO) functions. In the case of *Multiple-Output* (MO) functions the method derives the exact minimal *Multi-Valued* PSDKRO (MV-PSDKRO). However, if we are interested in implementing the function at the bit-level, product sharing allows further reduction of the number of required products. In [20], we devise a BDD-based representation of AND/EXOR cube covers. (The representation is similar to the AND/OR cube cover from [26], but the manipulation is different.) The representation allows us

to correctly account for product sharing. Utilizing the AND/EXOR cover representation, greedy algorithm for the grouping of output functions is developed. Starting from a MV-PSDKRO interpretation, we can in many cases obtain significant improvements. Furthermore, if the optimization goal is to reduce the number of products in a general AND/EXOR expression, we can relax the strict ordering criteria of PSDKROs. In [20] a minimization method based on Free-BDDs is developed. The experimental results obtained, can in cases challenge those of best known ESOPs, while the minimization is magnitudes faster.

Our results show that PKDD-based minimization offer an interesting alternative to general ESOP methods, as high quality results can be obtained within reasonable time bounds. The number of products in the two-level expression can be used directly as a cost measure for the final two-level implementation.

To emphasize the applicability, we have shown how the resulting AND/EXOR expressions can be efficiently mapped to both AND/EXOR PLAs and to fine grain FPGAs or ASICs [23].

3.2 Lattice Diagrams

Although an optimal or near optimal solution for the minimization problem can be obtained, not all functions have a compact two-level representation (and therefore not suitable for PLA implementation). The general solution is to seek a multi-level (decomposed) representation for the function at hand. Even though a compact multi-level representation can be obtained, it is difficult (during minimization) to foresee the quality of the final implementation. This leads to quality loss for example in cases where highly optimized netlists fit badly onto target architectures and/or cases where interconnecting delay violates the given timing constraints.

The introduction of BDDs has brought new means to multi-level synthesis. In its simplest form a BDD maps to a MUX based multilevel circuit. Also, more general DDs such as *Functional DDs* (FDDs) [18] and PKDDs can be mapped accordingly. The size of the circuit is related to the number of diagram nodes, thus heuristic methods for diagram minimization can be applied (as discussed earlier in this thesis). However, the number of nodes at one level is worst case exponential (to the number of levels). This makes the routing of such circuits often cumbersome, resulting in large area and delay overhead.

As one alternative *Lattice Diagrams* have been proposed, utilizing the inherent multi-level representation of DDs. In a lattice diagram the number of nodes at each level is worst case linear (to the level) which makes the diagram fit onto a two dimensional structure, such overcoming the routing difficulties. In the case of totally symmetric functions it is known that e.g., BDDs form lattice structures [4, 51]. Unfortunately, totally symmetric functions are rare, and we need a way to make functions "pseudo-symmetric" in order to derive lattice structures. This can be done by re-introducing support variables in similar fashion to the variable repetition of *Universal Akers Arrays* [1]. Akers' method, always considering worst case functions, was shown to render exponential lattice depth and was therefore considered infeasible.

Nevertheless, lattice structures have gained renewed interest as they offer properties (e.g., easily routable layout and predictable path length) sought for a number of current technologies, e.g., [30]. Synthesis methods have been revisited and a number of improvements presented. In [6], *Pseudo-Symmetric DDs* (PSDDs) were first introduced. The use of multiple-symmetries for PS-BDD synthesis was exploited in [50]. By the use of external encoders diagram depth could be reduced. However, the strict lattice structure was lost by their approach. The concept of lat-

tice diagrams was generalized in [35] to *Pseudo-Symmetric Pseudo Kronecker DDs* (PSPKDDs) and basic mechanisms for their minimization was outlined. In [50] the special case of *Pseudo-Symmetric FDDs* (PSFDDs) was further studied. However, the approaches from [35, 6, 50, 5], sport no facility to find suitable orderings for the lattice diagram. Therefore, practical results could only be obtained for relatively small benchmark circuits.

Our approach based on PSPKDDs, features not only dynamic reordering methods (to find suitable orderings), but also powerful synthesis heuristics for the choice of decomposition types. The lattice diagram maps e.g., directly onto ASICs or fine grain FPGAs having 3 inputs per cell as shown in [24, 25]. All interconnections are local except for input variables which are routed to all cells in a level as folding is not considered by our synthesis method. The optimization goal is to minimize the number of levels in the lattice representation of the function. In general this is a complex task only manageable to tackle using heuristic methods as e.g., in [35] considering only certain combination of decomposition types. Our method is different in that we consider *all* types of decompositions but *only* in such a way that the choice of decomposition type has only a local effect on the lattice. For each node we consider choice of decomposition type, input variable polarity and negation of the interconnections. For the selection of decomposition variables we use two strategies: pre-processed and dynamic orderings. The latter often is able to produce better results at the cost of run time, while the pre-processing (although very fast) often leads to better results than a naive ordering. In many cases, our synthesis method is able to obtain optimal (or near optimal) depth implementations [24, 25] for single output functions.

Our latest contribution is a generalization of lattice diagrams into *Lattice Circuits* by relaxing the strict neighboring criterion [21]. The additional (but still limited) routing allows sharing of sub-functions between non neighboring nodes. This is done level by level in the synthesis procedure reducing the number of sub-functions to be synthesized and therefore the circuit size. The routing is performed such to ensure design criteria (e.g., path delay and limitation of routing resources) to be met. The power of sharing is best observed when synthesizing multiple-output functions, as optimal or near optimal (w.r.t., depth) results have already been reported for single output functions in [24, 25]. Results on multiple-output functions from [24, 25] show that lattice depth is strongly dependent on the initial layout of the output function nodes. By introducing lattice circuits the sensitivity to initial layout tends to be reduced. The advantages are manifested by experiments on a set of MCNC benchmarks, showing up to 50% win in lattice depth while the number of required complex gates is reduced by nearly 80%, [21].

Chapter 4

Introduction to the Included Publications

In this section we give a brief resume on each paper, highlighting significant research contributions. A major part of the underlying work on papers [22, 23, 20, 24, 25] was conducted at Albert Ludwigs University, Freiburg Germany, under the supervision of Prof. Dr. Bernd Becker and Dr. Rolf Drechsler.

4.1 Paper 1: Improved Minimization Methods of Pseudo Kronecker Expressions for Multiple Output Functions

Published in *IEEE International Symposium on Circuits and Systems* 1998, (ISCAS'98), [22]. Reprinted in Section 6.

This paper is focused on the minimization of two-valued PSDKRO expressions using pruning techniques, allowing to trade-off quality for CPU resources. For the first time a method to find good orderings for PSDKRO expressions is presented. Furthermore, we observe and thoroughly analyze the effect of sharing of products between output functions.

4.2 Paper 2: Look-up Table FPGA Synthesis from Minimized Multi-Valued Pseudo Kronecker Expressions

Published in *IEEE International Symposium on Multiple-Valued Logic* 1998, (ISMVL'98), [23]. Reprinted in Section 6.

We generalize the minimization method for 4-valued PSDKRO expressions presented in [40] to m-valued logic. Furthermore, our proposed minimization method considers both variable ordering and the sharing of products between output functions. By this, we obtain significant quality improvements on the number of required products. We also show that further improvements on the number of products can be obtained if we relax the ordering restriction. A minimization method based on Free-PKDDs is devised. In order to show its applicability, we generalize the method for FPGA synthesis [19] (based on two-valued PSDKRO expressions) to multiple-valued logic.

4.3 Paper 3: Decision Diagram Based Minimization of Pseudo Kronecker Expressions

A journal version of 4.1, submitted to *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, [20]. Reprinted in Section 6.

We elaborate on the minimization methods from 4.1, and give in depth descriptions on the proposed heuristics. A new representation for AND/EXOR cube covers is devised, allowing the sharing of products between output functions to be correctly accounted. Based on the cube cover representation a grouping algorithm is proposed. Our experiments show significant improvements to previously presented results.

4.4 Paper 4: Synthesis of Pseudo Kronecker Lattice Diagrams

Published at the *International Workshop on Reed-Muller Applications of the Reed-Muller Expansion in Circuit Design* 1999, (Reed-Muller'99), [25]. Reprinted in Section 6.

Layout driven synthesis based on lattice diagrams has shown to be an interesting alternative to commonly used methods. They offer not only predictable delay of the final implementation, but also to be easily mapped onto two dimensional structures, such as FPGAs or ASICs. However, previous approaches have shown to produce practical results only for small functions. In this paper we develop a heuristic method for Pseudo Symmetric PKDD minimization, taking both variable ordering and decomposition types in consideration. Furthermore, our method elegantly handles incompletely specified multiple-output functions. Experimental results show our method to outperform previous approaches.

4.5 Paper 5: Synthesis of Pseudo Kronecker Lattice Diagrams

Published in *IEEE International Conference on Computer Design* 1999, (ICCD'99), [24]. Reprinted in Section 6.

Essentially a short version of the paper published at the Reed-Muller workshop. (Included for completeness of the author's publications.)

4.6 Paper 6: Layout Driven Synthesis of Lattice Circuits

Sent for review to the *ACM Design Automation Conference*, fall 1999, [21]. Reprinted in Section 6.

We generalize lattice diagrams into *Lattice Circuits*. The strict neighboring criterion for node sharing in lattice diagrams is relaxed. By this, additional routing resources of the target architecture can be utilized in order to minimize the circuit complexity. The additional delay caused by the routing can be easily predicted, thus violation of design criteria can be avoided already during diagram minimization. Experimental results show substantial reduction in circuit depth (by up to 50 %) and the number of complex gates (by up to 80 %) compared to previously best known results.

Chapter 5

Conclusions

In this thesis we have investigated a number of important properties of Ternary-PKDDs and PKDDs. The conditions under which TPKDDs having complemented edges offer a canonical representation are given. Furthermore, we have shown that sifting operations on TPKDDs (with or without complemented edges) are truly local and can be pursued solely by the redirection of edges. These properties are utilized into a number of new heuristic minimization methods, both for TPKDD and PKDDs. Applied to the problem of two-level AND/EXOR minimization, PKDD methods have been shown to offer an interesting alternative to general ESOP methods. Our methods are particularly plausible when the initial problem holds an extensive number of products, making cube-based representation intractable. Furthermore, (for the first time) a heuristic minimization method for general m-valued PSDKRO expressions is presented. By developing a method for FPGA mapping, their applicability to layout driven synthesis have been manifested. Moreover, we have presented superior minimization methods for PKDD-based lattice diagrams, showing optimal or near optimal solutions for a number of practical problems. Finally, we have proposed a synthesis method for lattice circuits, allowing additional routing resources to be utilized, while meeting sought after design criteria.

5.1 Hot Topics

Based on our experiences gained, we outline a number of promising research topics and open problems in the following:

- In this thesis we have given the basic mechanisms allowing efficient minimization for TPKDDs. The optimization goal is to minimize the number of nodes in the diagram. Furthermore, we have shown how a minimized PKDD can be derived from a TPKDD. For the problem of PKDD minimization, we expect combining TPKDD sifting together with PKDD cost measures to further reduce diagram size. Note that sifting on the TPKDD is still a local operation, but the decomposition types of the nodes belonging to the interchanged levels needs to be re-optimized during each sifting operation. This is exemplified in Figure 2.10, showing that sifting of a PKDD might cause illegal decomposition types. Furthermore, to keep reference counters consistent, we must first de-reference the nodes of the sifted levels, (before sifting is applied), and then (after sifting) re-reference them again according to the decomposition types selected.

- The ability to implicitly represent and manipulate large cube-sets offers new means towards developing efficient minimization methods, [20]. E.g., applied to the problem of general ESOP minimization, if allows sets of cubes to be manipulated simultaneously using DD techniques (as opposed to cube-based methods considering single cubes only). One interesting approach in this direction was presented in [52]. Starting from an OBDD, OFDD or OKFDD, transformations were applied such to reduce the number of 1-paths in a XOR-based DD. As PKDDs may offer a more compact representation (i.e., having fewer 1-paths), the starting point can be improved.

- We expect the potential compactness of PKDDs to be applicable to many minimization problems emerging in VLSI CAD. The implicit multi-level representation of the DD may be utilized directly [45], or as a starting point for finding decomposition trees. In [49] a novel method to derive a multi-level network is devised. From an initial minimized FDD representation decompositions and transformations are iteratively applied. As PKDDs *may* offer a more compact representation we expect them to be a better starting point for such operations (if the method proves to be applicable). Furthermore, other classical decomposition methods applicable to DDs can be investigated. In particular, one promising method based on (k-feasible) bi-decomposition can be directly applied. A *cut* between two diagram levels, divides the diagram into functions having disjoint support. By finding the cut-level minimizing the number of edges crossing the cut-line, the complexity of the multi-level representation is reduced. This is recursively applied until the each network node depends on at most k variables. As the number of nodes (and edges) in a PKDD is potentially less than any other ordered bit-level DD, a PKDD-based approach might prove to be prosperous. Therefore, a future research direction is to study the role of DDs based on EXOR decompositions. It has to be investigated whether the size reduction also corresponds to an area reduction in multi-level synthesis.

- Our method synthesizing lattice diagrams from PKDDs has been shown superior to other methods, [24, 25]. The latest extension into lattice circuits [21] can be taken another step further, combining general decompositions {AND, OR, EXOR} with the strict DD decompositions $\{Sh, pD, nD\}$ currently applied. I.e., under the condition that the lattice structure is preserved *any* bit-level decomposition can be applied. In this, we expect to combine the benefits known from multi-level synthesis (i.e., circuit compactness) with the sought after properties of lattice structures.

5.2 Bottom Line

This thesis spans from validation and verification of high-level specifications, as discussed in Chapter 7, all the way down to layout driven synthesis, combining logic minimization, mapping and routing to the target architecture at hand. We conclude our work to offer new means towards solving many of the crucial problems occurring along the design process of modern digital circuits.

References

[1] S.B. Akers. A rectangular logic array. *IEEE Trans. on Comp.*, C-21:848–857, 1972.

[2] B. Becker and R. Drechsler. How many decomposition types do we need? In *European Design & Test Conf.*, pages 438–443, 1995.

[3] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[4] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[5] M. Chrzanowska-Jeske, X.Y. Ma, and W. Wang. Psedudo-symmetric functional decision diagrams. In *Int'l Symp. Circ. and Systems*, pages VI:175–VI:178, 1998.

[6] M. Chrzanowska-Jeske, Z. Wang, and Y. Xu. A regular representation for mapping to fine-grain, locally-connected FPGAs. In *Proc. Midwest Symp. Circ. Syst*, pages 2749–2752, 1997.

[7] O. Coudert. Two-level logic minimization: an overview. *INTEGRATION, the VLSI Jour.*, 17(2):97–140, 1994.

[8] R. Drechsler. Pseudo Kronecker expressions for symmetric functions. In *VLSI Design Conf.*, pages 511–513, 1997.

[9] R. Drechsler and B. Becker. – Ordered Kronecker functional decision diagrams – a data structure for representation and manipulation of boolean functions. *IEEE Trans. on CAD*, 17(10):965–973, 1998.

[10] R. Drechsler, N. Drechsler, and W. Günther. Fast exact minimization of BDDs. In *Design Automation Conf.*, pages 200–205, 1998.

[11] R. Drechsler and W. Günther. Using lower bounds during dynamic BDD minimization. In *Design Automation Conf.*, pages 29–32, 1999.

[12] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In *Design Automation Conf.*, pages 415–419, 1994.

[13] M. Escobar and F. Somenzi. Synthesis of AND-EXOR expressions via satisfyability. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 80–87, 1995.

[14] S.J. Friedman and K.J. Supowit. Finding the optimal variable ordering for binary decision diagrams. In *Design Automation Conf.*, pages 348–356, 1987.

[15] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level synthesis. In *European Conf. on Design Automation*, pages 50–54, 1991.

[16] W. Günther and R. Drechsler. Minimization of free BDDs. In *ASP Design Automation Conf.*, pages 323–326, 1999.

[17] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchange of variables. In *Int'l Conf. on CAD*, pages 472–475, 1991.

[18] U. Kebschull and W. Rosenstiel. Efficient graph-based computation and manipulation of functional decision diagrams. In *European Conf. on Design Automation*, pages 278–282, 1993.

[19] G. Lee and R. Drechsler. ETDD-based generation of complex terms for incompletely specified boolean functions. In *ASP Design Automation Conf.*, pages 75–80, 1998.

[20] P. Lindgren, R. Drechsler, and B. Becker. Decision Diagram Based Minimization of Pseudo Kronecker Expressions. In *submitted to IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems.*

[21] P. Lindgren, R. Drechsler, and B. Becker. Layout Driven Synthesis of Lattice Circuits. In *submitted to ACM Design Automation Conference, DAC-2000.*

[22] P. Lindgren, R. Drechsler, and B. Becker. Improved minimization methods of pseudo kronecker expressions for multiple output functions. In *Int'l Symp. Circ. and Systems*, pages VI:187–VI:190, 1998.

[23] P. Lindgren, R. Drechsler, and B. Becker. Look-up table FPGA synthesis from minimized multi-valued pseudo Kronecker expressions. In *Int'l Symp. on Multi-Valued Logic*, pages 95–100, 1998.

[24] P. Lindgren, R. Drechsler, and B. Becker. Synthesis of pseudo kronecker lattice diagrams. In *Int'l Conf. on Comp. Design*, pages 307–310, 1999.

[25] P. Lindgren, R. Drechsler, and B. Becker. Synthesis of Pseudo Kronecker Lattice Diagrams. In *International Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, 1999.

[26] S. Minato. Calculation of unate cube set algebra using zero-suppressed bdds. In *Design Automation Conf.*, pages 420–424, 1994.

[27] S. Minato. Graph-Based Representations of Discrete Functions. In T. Sasao and M. Fujita, editors, *Representation of Discrete Functions*, pages 1–26. Kluwer Academic Publisher, 1996.

[28] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagrams with attributed edges for efficient Boolean function manipulation. In *Design Automation Conf.*, pages 52–57, 1990.

[29] D. Möller, P. Molitor, and R. Drechsler. Symmetry based variable ordering for ROBDDs. *IFIP Workshop on Logic and Architecture Synthesis, Grenoble*, pages 47–53, 1994.

[30] A. Mukherjee, R. Sudhakar, M. Marek-Sadowska, and S. Long. Wave steering in YADDs: A novel non-iterative synthesis and layout technique. In *Design Automation Conf.*, pages 466–471, 1999.

[31] R. Murgai, Y. Nishizaki, N. Shenoy, R.K. Brayton, and A. Sangiovanni-Vincentelli. Logic synthesis for programmable gate arrays. In *Design Automation Conf.*, pages 620–625, 1990.

[32] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Int'l Conf. on CAD*, pages 74–77, 1995.

[33] S. Panda, F. Somenzi, and B.F. Plessier. Symmetry detection and dynamic variable ordering of decision diagrams. In *Int'l Conf. on CAD*, pages 628–631, 1994.

[34] M.A. Perkowski. The generalized orthonormal expansion of functions with multiple-valued inputs and some of its application. In *Int'l Symp. on Multi-Valued Logic*, pages 442–450, 1992.

[35] M.A. Perkowski, M. Chrzanowska-Jeske, and Y. Xu. Lattice diagrams using Reed-Muller logic. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 85–102, 1997.

[36] B. Becker R. Drechsler and N. Drechsler. OKFDD Minimization by Genetic Algorithms with Application to Circuit Design. *INTEGRATION, the VLSI Journal*, 28(2):121–139, 1999.

[37] U. Rollwage. The complexity of mod-2 sum PLA's for symmetric functions. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 6–12, 1993.

[38] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[39] T. Sasao. A transformation of multiple-valued input two-valued output functions and its application to simplification of exclusive or sum-of-products expressions. In *Int'l Symp. on Multi-Valued Logic*, pages 270–279, 1991.

[40] T. Sasao. Optimization of multi-valued AND-EXOR expressions using multiple-place decision diagrams. In *Int'l Symp. on Multi-Valued Logic*, pages 451–458, 1992.

[41] T. Sasao. AND-EXOR expressions and their optimization. In T. Sasao, editor, *Logic Synthesis and Optimization*, pages 287–312. Kluwer Academic Publisher, 1993.

[42] T. Sasao. An exact minimization of AND-EXOR expressions using BDDs. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 91–98, 1993.

[43] T. Sasao. Representations of logic functions using EXOR operations. In T. Sasao and M. Fujita, editors, *Representation of Discrete Functions*, pages 29–54. Kluwer Academic Publisher, 1996.

[44] T. Sasao and Ph. Besslich. On the complexity of mod-2 sum PLAs. *IEEE Trans. on Comp.*, 39:262–266, 1990.

[45] T. Sasao and J. Butler. A design method for look-up table type FPGA by pseudo-Kronecker expansion. In *Int'l Symp. on Multi-Valued Logic*, pages 97–106, 1994.

[46] T. Sasao and J.T. Butler. Planar multiple-valued decision diagrams. In *Int'l Symp. on Multi-Valued Logic*, pages 28–35, 1995.

[47] C. Scholl, D. Möller, P. Molitor, and R. Drechsler. BDD minimization using symmetries. *IEEE Trans. on CAD*, 18(2):81–100, 1999.

[48] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0.* University of Colorado at Boulder, 1998.

[49] C.C. Tsai and M. Marek-Sadowska. Multilevel logic synthesis for arithmetic functions. In *Design Automation Conf.*, pages 242–247, 1996.

[50] W. Wang and M. Chrzanowska-Jeske. Optimizing pseudo-symmetric binary decision diagrams using multiple symmetries. In *Int'l Workshop on Logic Synth.*, pages 334–340, 1998.

[51] I. Wegener. Optimal decision trees and one-time-only branching programs for symmetric Boolean functions. *Information and Control*, 62:129–143, 1984.

[52] Y. Ye and K. Roy. Graph-based synthesis algorithms for AND/XOR networks. In *Design Automation Conf.*, pages 107–112, 1997.

Chapter 6

Reprint of Publications

This section features reprints of the author's post licentiate work. No changes to the original manuscripts were made.

Paper 1: Improved Minimization Methods of Pseudo Kronecker Expressions for Multiple Output Functions

Published in *IEEE International Symposium on Circuits and Systems* 1998, (ISCAS'98), [22].

Paper 2: Look-up Table FPGA Synthesis from Minimized Multi-Valued Pseudo Kronecker Expressions

Published in *IEEE International Symposium on Multiple-Valued Logic* 1998, (ISMVL'98), [23].

Paper 3: Decision Diagram Based Minimization of Pseudo Kronecker Expressions

A journal version of 4.1, submitted to *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, [20].

Paper 4: Synthesis of Pseudo Kronecker Lattice Diagrams

Published at the *International Workshop on Reed-Muller Applications of the Reed-Muller Expansion in Circuit Design* 1999, (Reed-Muller'99), [25].

Paper 5: Synthesis of Pseudo Kronecker Lattice Diagrams

Published in *IEEE International Conference on Computer Design* 1999, (ICCD'99), [24].

Paper 6: Layout Driven Synthesis of Lattice Circuits

Sent for review to the *ACM Design Automation Conference*, fall 1999, [21].

Chapter 7

Reprint of Licentiate Thesis: Dealing with Incompleteness in Circuit Design under Kleenean Strong Ternary Logic

Reprint of document, 1997:18. ISSN:1402-1757. ISRN:LTU-LIC–1997/18–SE.

Errata

- Page 5, row 4 from bottom; for $a \neq 1$, should read, for $a = 1$.
- Page 7, Figure 1.2, Table $N3$, column d, row 1; 0 should read 1. Row 2; 1 should read 0.
- Page 19, Figure 2.3, Table NOT, row 1; 0 should read 1. Row 2; 1 should read 0.