Representation of Incompletely Specified Switching Functions Using Pseudo-Kronecker Decision Diagrams

Munehiro MATSUURA[1] and Tsutomu SASAO[1,2]
[1]Department of Computer Science and Electronics, Kyushu Institute of Technology
[2]Center for Microelectronic Systems, Kyushu Institute of Technology

June 11, 2001

Abstract

Pseudo-Kronecker decision diagram (PKDD) is a generalization of binary decision diagram (BDD). A PKDD requires not more nodes than a BDD to represent the same function. In this paper, we consider a method to represent incompletely specified functions by using PKDDs. We developed a heuristic method to obtain PKDDs. Many PKDDs for MCNC benchmark functions with *don't cares* are simplified. Experimental results show that the number of nodes of PKDD can be reduced by 14% by considering don't cares.

1 Introduction

Binary decision diagrams (BDDs) are extensively used to represent logic functions. Pseudo-Kronecker decision diagrams (PKDDs) are generalization of binary decision diagrams (BDDs): A PKDD requires not more nodes than a BDD to represent the same function. PKDDs can be directly converted into multi-level networks [6]. PKDDs can be also used to design LUT type FPGA [5, 9]. In this paper, we consider a method to represent incompletely specified functions by using PKDDs. We want to obtain PKDDs having as few nodes as possible. An optimum PKDD that represents a given incompletely specified function can be found by considering $O(2^d \cdot n!3^n \cdot 3^{2^{n-1}})$ different PKDDs, where n is the number of input variables, and d is the number of *don't cares*. However it is impractical to consider all these possibilities. Thus, we have to resort to a heuristic method to obtain near optimal PKDDs. To the best of our knowledge, no paper have considered an algorithm to obtain PKDDs for incompletely specified functions. It is quite interesting to know how much we can reduce size of PKDDs by considering *don't cares*. In this paper, we show experimental results for MCNC benchmark functions with *don't cares*, and compare the number of nodes of PKDDs with and without considering *don't cares*.

2 Pseudo-Kronecker Decision Diagram

2.1 Binary Decision Diagram (BDD)

An arbitrary logic function $f(x_1, x_2, \ldots, x_n)$ can be expanded as

$$f = \bar{x}_1 f_0 \oplus x_1 f_1 \tag{2.1}$$

$$f = f_0 \oplus x_1 f_2 \tag{2.2}$$

$$f = \bar{x}_1 f_2 \oplus f_1 \tag{2.3}$$

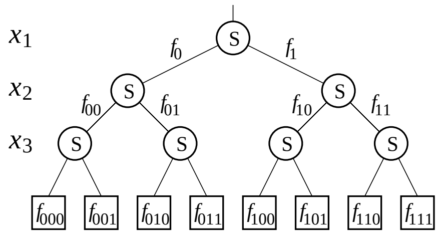

Figure 2.1: Binary decision tree.

Equations (2.1)–(2.3) show the *positive Davio expansion*, the *negative Davio expansion*, and the *Shannon expansion*, respectively. Given a function f and a variable x_1 for these expansions, subfunctions f_0, f_1, and f_2 are unique. By applying the Shannon expansions to subfunctions f_0 and f_1, we have

$$f_0 = \bar{x}_2 f_{00} \oplus x_2 f_{01},$$

$$f_1 = \bar{x}_2 f_{10} \oplus x_2 f_{11}.$$

Similarly, by applying the Shannon expansion to f_{00}, f_{01}, f_{10}, and f_{11}, we have

$$f_{00} = \bar{x}_3 f_{000} \oplus x_3 f_{001},$$

$$f_{01} = \bar{x}_3 f_{010} \oplus x_3 f_{011},$$

$$f_{10} = \bar{x}_3 f_{100} \oplus x_3 f_{101},$$

$$f_{11} = \bar{x}_3 f_{110} \oplus x_3 f_{111}.$$

Fig. 2.1 shows the expansion tree, where the symbol S denotes the Shannon expansion. This tree is a *binary decision tree (BDT)*. The *binary decision diagrams* (BDDs) are obtained from BDTs using the following rules:

1) Eliminate all the redundant nodes whose two edges point to the same nodes.
2) Merge two nodes if they represent the same function.

In a BDD, when we fix the ordering of the variables, we have a unique BDD.

2.2 Kronecker Decision Diagram (KDD)

When we can use any of the three expansions (2.1)–(2.3) for each variable, we have a *Kronecker tree*. Fig. 2.2 shows an example of a Kronecker tree, where the symbol pD denotes the positive Davio expansion, and nD denotes the negative Davio expansion. In this tree, variable x_1 uses the Shannon expansion, variable x_2 uses the positive Davio expansion, and variable x_3 uses the negative Davio expansion. This expansion is called a Kronecker expansion, and a simplified Kronecker trees is called a *Kronecker Decision Diagram* (KDD). For an n-variable function, at most 3^n different KDDs exist.

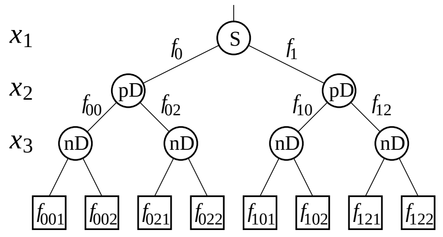

Figure 2.2: Kronecker decision tree.

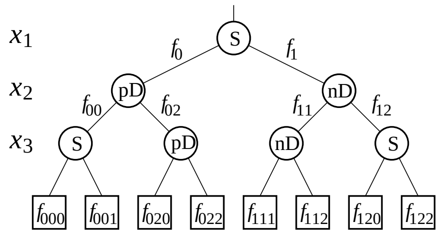

Figure 2.3: Pseudo-Kronecker decision tree.

2.3 Pseudo-Kronecker Decision Diagram (PKDD)

When we can use any of the three expansions (2.1)–(2.3) for each node, then we have a *pseudo-Kronecker tree*. Fig. 2.3 shows an example of a pseudo-Kronecker tree, where the first variable uses the Shannon expansion, the second variable uses both the positive and the negative Davio expansions, and the last variable uses all the three expansions. This expansion is called a pseudo-Kronecker expansion, and the simplified pseudo-Kronecker trees is called a *pseudo-Kronecker Decision Diagram* (PKDD). For n-variable functions, at most 3^{2^n-1} different expansions exist. Fig. 2.4 shows the relation among PKDDs, KDDs, and BDDs.

2.4 Strategy to Optimize PKDDs for Incompletely Specified Functions

As for the number of different ways to construct PKDDs for an n-variable function, we have

a) $n!$ ways to permute the input variables, and

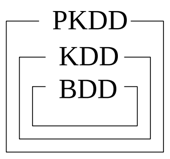

Figure 2.4: Relation of BDDs, KDDs, and PKDDs.

b) 3^{2^n-1} ways to select the expansions for the nodes.

Thus, it is virtually impossible to consider all the possibilities, and heuristic methods to minimize PKDDs have been proposed [6, 3]. Furthermore, for an incompletely specified function with d *don't cares*, we have to consider 2^d different functions. Finding the assignment that leads to the smallest BDD is known to be NP-complete [11]. Therefore, we utilize existing heuristic methods [4, 6], and construct PKDD as follows:

1. First, we will find the *don't care* assignment that reduces the number of nodes in the BDD (Algorithm 3.1).
2. Second, we will find the variable ordering that reduces the size of DDs, and obtain a small KDD (Algorithm 4.1).
3. Finally, we will find good expansion type for each node to obtain a small PKDD (Algorithm 4.2).

3 Simplification of BDDs for Incompletely Specified Functions

In this section, we will review a method to reduce the number of nodes of BDDs using *don't cares* [4].

3.1 Incompletely Specified Function

Incompletely specified functions can be represented as a pair of completely specified functions $[f, g]$, where g denotes the *care* set, $f \cdot g$ denotes the ON set, $\bar{f} \cdot g$ denotes the OFF set, and $\bar{g}$ denotes the *don't care* set.

Definition 3.1 *h is a* **cover** *of $[f, g]$ if $f \cdot g \subseteq h \subseteq f \vee \bar{g}$. Especially, if an arbitrary cover of $[f_1, g_1]$ is also a cover of $[f_2, g_2]$, then $[f_1, g_1]$ is an i-***cover** *of $[f_2, g_2]$. If a common i-cover exists for two incompletely specified functions, then they* **match**.

When two incompletely specified functions match, we can replace the representation with a common i-cover function. In the case of a BDD, if two nodes represent functions having a common i-cover, then these nodes can be replaced by the node of the i-cover function.

3.2 Don't Care Assignment

When two incompletely specified functions match, we use three criteria for *don't care* assignment.

Let $[f_1, g_1]$ and $[f_2, g_2]$ be incompletely specified functions.

1. One-sided DC match (OSDM)

   $[f_1, g_1]$ OSDM $[f_2, g_2]$ iff $g_1 = 0$.

   This shows that all the values of the first function are *don't care*.

2. One-sided match (OSM)

   $[f_1, g_1]$ OSM $[f_2, g_2]$ iff $f_1 \oplus f_2 \subseteq \bar{g}_1$ and $\bar{g}_2 \subseteq \bar{g}_1$.

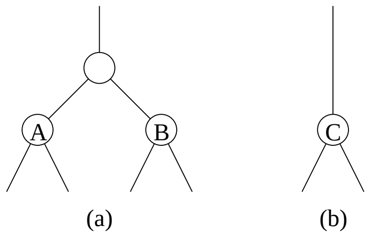

Figure 3.5: Simplification of BDDs using *don't cares*.

This shows that two functions can be made equal by assigning *don't cares* of the first function.

3. Two-sided match (TSM)

   $[f_1, g_1]$ TSM $[f_2, g_2]$ iff $f_1 \oplus f_2 \subseteq \bar{g}_1 \vee \bar{g}_2$.

   This shows that two functions can be made to equal by assigning *don't cares* of both functions.

We want to retain as many *don't cares* unassigned as possible. When two functions $[f_1, g_1]$ and $[f_2, g_2]$ match, we assign *don't cares* as follow:

1. OSDM: $[f_2, g_2]$
2. OSM: $[f_2, g_2]$
3. TSM: $[f_1 g_1 \vee f_2 g_2, g_1 \vee g_2]$

3.3 Simplification of BDDs using Don't Cares

When matching two nodes in a BDD, we pay attention to two children of a node. When two children have a common i-cover, these nodes are replaced by a node of i-cover, and the parent node is deleted.

The next example shows a method to simplify BDDs by using *don't care* matching.

Example 3.1 *In Fig. 3.5(a), if node A and node B have a common i-cover, then they are replaced by a node representing the i-cover function as shown in Fig. 3.5(b).* *(End of Example)*

Algorithm 3.1 *From the root node to the leaf nodes, for each node apply the following steps recursively:*

1. *If $g = 1$, then return.*
2. *Check whether if there exists an common i-cover for two children, in the order of OSDM, OSM, and TSM.*
   (a) *If no common i-cover exist, then apply this procedure to two children nodes.*
   (b) *If a common i-cover exists, then replace the nodes with the i-cover node, and apply this procedure to the i-cover node.*

4 Simplification of PKDD

In a BDD, a non-terminal node has two children that represent subfunctions f_0 and f_1 in (2.1). On the other hand, in an EXOR ternary decision diagram (ETDD), a non-terminal node has three children that represent three functions: f_0, f_1, and $f_2 = f_0 \oplus f_1$. Our simplification algorithms for KDDs and PKDDs use ETDDs [8]. First, we show an algorithm that simplifies KDDs.

Algorithm 4.1 *(Simplification of KDDs)*

1. *Find variable orderings for ETDD and BDD by an algorithm that minimizes number of nodes [1, 10].*
2. *Construct ETDDs for two variable orderings found in Step 1.*
3. *Do the following for each case when all variables are expanded by S, pD, and nD expansions.*
4. *For each variable, change the expansion type in the order of S, pD, and nD, and select the type that minimizes the number of nodes.*
5. *Do Step 3 for all variables from the root to a terminal node.*
6. *Repeat the above steps while the number of nodes decreases.*
7. *Choose the one with the fewer nodes between two variable orderings found in Step 1.*

The resulting KDD is used as an initial solution of the PKDD.

Algorithm 4.2 *(Simplification of PKDDs)*

1. *Find variable orderings for ETDD and BDD by using the algorithm [1, 10].*
2. *Construct ETDD for two variable orderings found in Step 1.*
3. *Use the expansion type of the KDD as the initial solution of the PKDD.*
4. *For each node, change the expansion type in the order of S, pD, and nD, and select one with the minimum number of nodes.*
5. *Do Step 3 for all variables from the root to a terminal node.*
6. *Repeat the above steps while the number of nodes decreases.*
7. *Choose the one with the fewer nodes between two variable orderings found in Step 1.*

Next, we show algorithm to simplify PKDDs for incompletely specified functions.

Algorithm 4.3 *(Simplification of PKDDs for incompletely specified functions)*

1. *Assign 0 to all the don't cares. Find a variable ordering by using the algorithm [1, 10].*

2. *Reduce the number of nodes of the BDD by using Algorithm 3.1.*

3. *Reduce the number of nodes of the KDD by using Algorithm 4.1.*

4. *Reduce the number of nodes of the PKDD by using Algorithm 4.2.*

5. *Assign* 1 *to all the don't cares, and repeat Steps 2 to 4.*

6. *Choose the PKDD with the fewer nodes.*

5 Experimental Results

We used MCNC benchmark functions with *don't cares*. Our BDD package dose not use negation edges.

5.1 Simplification of BDDs Using Don't Cares

Table 5.1 shows the result of simplification of BDDs by using Algorithm 3.1. We considered two cases: The column headed with $DC = 0$ denotes that all the *don't cares* are set to 0, and the column headed with $DC = 1$ denotes that all the *don't cares* are set to 1 in the incompletely specified functions. In Table 5.1, Total denotes the total number of nodes, and Ratio denotes the average of reduction ratio for the functions. This table shows that Algorithm 3.1 reduces the total number of nodes by 15%, and the reduction ratio is about 11%. Ratio for $DC = 1$ is reduced more than the case of $DC = 0$, but Total for $DC = 1$ is larger. Note that ex1010 and pdc can be reduced into a half.

5.2 Simplification of PKDDs Using Don't Cares

Table 5.2 compares the sizes of BDDs and PKDDs. The PKDDs are obtained from the result of Table 5.1. In Table 5.2, Total denotes the total number of nodes, and Ratio denotes the average reduction ratio for $DC = 0$. The total number of PKDD nodes is reduced by 16% from the case of $DC = 0$ in Table 5.1. Ratio is reduced by 14%. The total number of nodes in the last column of Table 5.2 is reduced by 30% from the BDD of the column $DC = 0$ in Table 5.1.

For example, in the case of ex1010, the number of BDD nodes is decreased about 49% by using Algorithm 3.1, and further reduced about 6% by using Algorithm 4.2. The number of PKDD nodes is reduced by 58% from $DC = 0$ in Table 5.1.

5.3 Relative Sizes of Various Decision Diagrams

Fig. 5.6 compares of relative sizes of nodes of BDDs and PKDDs with and without *don't care* optimization. The column footed with *BDD* denotes the size of BDDs with $DC = 0$; *BDD Match* denotes the size of BDDs optimized by using Algorithm 3.1; *PKDD* denotes the size of PKDDs with $DC = 0$; and *PKDD Match* denotes the size of PKDDs optimized by using Algorithm 4.3. Compared with the original BDD, *BDD Match* is decreased by 11%, *PKDD* is decreased by 23%, and *PKDD Match* is reduced by 34%.

Table 5.1: Number of nodes of BDDs.

Name	In	Out	$DC = 0$	Match0	$DC = 1$	Match1
alu2	10	8	70	68	76	70
apla	10	12	115	91	119	93
b10	15	11	290	288	290	288
b11	8	31	99	98	99	98
b3	32	20	409	408	409	408
b4	33	23	219	218	225	224
b7	8	31	99	98	99	98
bca	26	46	894	892	897	895
bcb	26	39	756	751	756	751
bcc	26	45	801	793	780	771
bcd	26	38	606	606	608	608
bcddiv	4	4	19	19	20	19
bench	6	8	85	51	101	52
bench1	9	9	608	365	687	360
bw	5	28	137	122	137	122
dekoder	4	7	27	27	28	25
dk17	10	11	74	69	81	67
dk27	9	9	35	33	42	38
dk48	15	17	81	79	102	72
ex1010	10	10	1421	726	1426	729
exam	10	10	341	161	372	158
exep	30	63	675	675	756	739
exp	8	18	212	187	222	202
exps	8	38	585	583	585	583
fout	6	10	141	114	155	114
inc	7	9	82	75	80	73
l8err	8	8	84	83	84	83
mark1	20	31	119	115	177	105
p1	8	18	208	151	210	147
p3	8	14	134	106	152	106
pdc	16	40	609	337	696	343
sex	9	14	62	61	62	61
spla	16	46	628	608	636	616
t2	17	16	145	125	147	125
t4	12	8	44	44	52	44
wim	4	7	26	26	27	24
Total			10940	9253	11395	9311
Ratio			1.00	0.89	1.00	0.84

$DC = 0$:	Optimized BDD nodes with all *don't cares* are assigned 0.
Match0:	Applied Algorithm 3.1 to BDDs with $DC = 0$.
$DC = 1$:	Optimized BDD nodes with all *don't cares* are assigned 1.
Match1:	Applied Algorithm 3.1 to BDDs with $DC = 1$.

6 Conclusion

In this paper, we proposed a method to represent incompletely specified functions by using PKDDs. PKDDs require fewer nodes than corresponding BDDs, and they can be converted into multi-level logic networks directly. Experimental results show that the number of nodes in PKDDs can be reduced by 14% by considering *don't cares*.

Table 5.2: Number of nodes of PKDDs.

Name	In	Out	BDD	PKDD		
				$DC = 0$	$DC = 1$	Algorithm 4.3
alu2	10	8	70	60	64	48
apla	10	12	115	96	96	68
b10	15	11	290	249	246	244
b11	8	31	99	56	54	54
b3	32	20	409	308	308	308
b4	33	23	219	157	196	157
b7	8	31	99	56	54	54
bca	26	46	894	797	791	784
bcb	26	39	756	751	664	658
bcc	26	45	801	704	684	670
bcd	26	38	606	536	530	531
bcddiv	4	4	19	12	12	10
bench	6	8	85	61	68	39
bench1	9	9	608	513	540	335
bw	5	28	137	90	90	83
dekoder	4	7	27	17	19	15
dk17	10	11	74	65	63	49
dk27	9	9	35	22	30	22
dk48	15	17	81	60	74	52
ex1010	10	10	1421	1268	1274	684
exam	10	10	341	272	289	116
exep	30	63	675	543	671	581
exp	8	18	212	156	179	152
exps	8	38	585	495	493	489
fout	6	10	141	111	117	92
inc	7	9	82	62	58	55
l8err	8	8	84	72	69	69
mark1	20	31	119	60	124	60
p1	8	18	208	174	179	114
p3	8	14	134	111	127	77
pdc	16	40	609	400	557	248
sex	9	14	62	41	41	41
spla	16	46	628	578	525	531
t2	17	16	145	119	120	105
t4	12	8	44	33	41	29
wim	4	7	26	16	18	14
Total			10940	9121	9465	7631
Ratio				1.00		0.86

$DC = 0$: Optimized PKDD nodes with all *don't cares* are assigned 0.

$DC = 1$: Optimized PKDD nodes with all *don't cares* are assigned 1.

Acknowledgments

This research is partly supported by Japan Society for the Promotion of Science (JSPS) under Grant. Mr. Kazuya Sugiki developed a prototype of Algorithm 3.1.

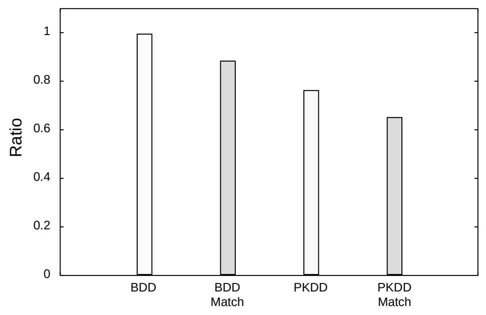

Figure 5.6: Relative sizes of decision diagrams.

References

[1] N. Ishiura, H. Sawada, and S. Yajima, “Minimization of binary decision diagrams based on exchange of variables,” *ICCAD-91*, pp. 472-475, Nov. 1991.

[2] P. Lindgren, *Applications of Decision Diagrams in Digital Circuit Design*, Ph. D. Thesis, Lulea University, Sweden, Dec. 1999.

[3] P. Lindgren, R. Drechsler, and B. Becker, “Minimization of ordered pseudo Kronecker decision diagrams,” *Proceedings 2000 International Conference on Computer Design*, 504-10, 2000

[4] T. R. Shiple, R. Hojati, A. L. Sangiovanni-Vincentelli, and R. K. Brayton, “Heuristic minimization of BDDs using don’t cares,” *Design Automation Conference 1994*, pp. 225-231, 1994.

[5] T. Sasao and J. T. Butler, “A design method for look-up table type FPGA by pseudo-Kronecker expansion,” *Proc. International Symposium on Multiple-Valued Logic*, pp. 97-106, May 1994.

[6] T. Sasao, H. Hamachi, S. Wada, and M. Matsuura, “Multi-level logic synthesis based on pseudo-Kronecker decision diagrams and logical transformation,” *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansions in Circuit Design (Reed-Muller’95)*, Makuhari, Japan, Aug. 27-29, 1995.

[7] T. Sasao and M. Fujita (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers, Boston, 1996.

[8] T. Sasao, “Representation of logic functions using EXOR operators,” Chapter 2 in [7].

[9] T. Sasao, K. Kurimoto, and M. Matsuura, “FPGA design using pseudo-Kronecker decision diagrams,” (in Japanese), *Technical Report of IEICE*, VLD2000-95, Nov. 2000.

[10] R. Rudell, “Dynamic variable ordering for ordered binary decision diagrams,” *Proc. ICCAD*, pp. 42-47, 1993.

[11] M. Sauerhoff and I. Wegener, “On the complexity of minimizing the OBDD size for incompletely specified functions,” *IEEE Trans. Computer-Aided Design*, Vol. 15, pp. 1435-1437, Nov. 1996.

[12] S. Yang, “Logic synthesis and optimization benchmark user guide,” Version 3.0, MCNC, Jan. 1991.