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

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

Rolf Drechsler Bernd Becker
Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
{drechsle/becker}@informatik.uni-freiburg.de

Abstract

In this paper we outline a method for Look-up Table-FPGA (LUT-FPGA) synthesis from minimized Multi-Valued Pseudo Kronecker Expressions (MV PSDKROs). By restricting logic minimization to consider only easily mappable expressions, a regular Cellular Architecture (CA) layout without routing overhead is obtained. In this way our method combines logic minimization, mapping and routing.

The transformation into the MV domain reduces the area as the number of products in the PSDKRO expression can be further minimized. Deriving the exact minimum MV PSDKRO is known to be hard or even intractable. We address this by applying pruning techniques based on cost estimation and dynamic methods to find suitable variable orderings. Results on a set of MCNC benchmarks show the advantages of the proposed minimization methods.

1 Introduction

The increasing complexity of modern VLSI circuitry is only manageable through advanced CAD systems which as one important component include logic synthesis tools. Problems encountered in synthesis can often be formulated in terms of Boolean functions. One interesting design style is based on FPGAs [1, 8]. Many powerful tools have been developed in the past to minimize Boolean functions and map them to commercially available FPGA types. One of the major drawbacks of most methods presented so far is that they consider logic minimization, mapping and routing independently, i.e., during synthesis only “synthesis measures” like number of gate equivalents are considered. These approaches suffer from large area overhead in cases where the highly optimized netlist do not fit well on the target FPGA [14].

Recently, first approaches have been presented combining minimization, mapping and routing (see e.g. [10]). By restricting the synthesis process to consider only easily

mappable expressions, e.g. Maitra terms, a suitable layout can be derived directly during synthesis. In [5, 6] a method to derive Maitra terms from *EXOR Ternary Decision Diagrams* (ETDDs) (as used for 2-valued PSDKRO minimization in [13]) was presented. Although elegant, this synthesis method is directly applicable only to CAs where each cell computes a 2-input function. However, many of today's popular programmable architectures have cells capable of computing functions of 3 or more variables.

In this paper we outline a synthesis methodology applicable to general k_{in} - p_{out} -LUT FPGA architectures. By the use of DD techniques a minimized 2^{k-p} -valued PSDKRO expression is derived. From this 2-level form a regular layout structure without routing overhead is directly obtained, (giving an upper bound area estimate as further optimization may be applied). In this way minimization, mapping and routing are combined. By solving the minimization problem in the MV domain the number of PSDKRO products can be further minimized [13] and the computational power of the k_{in} -LUTs is better exploited.

As deriving the exact minimum MV PSDKRO is known to be hard or even intractable (for many practical examples), we apply pruning techniques based on cost estimation to obtain heuristic results. Furthermore, the quality of the result is known to be dependent on the grouping of input variables and the variable ordering applied during minimization. For this we discuss the use of dynamic methods. Furthermore, we present a “free” DD minimization method (relaxing the fixed ordering constraint of PSDKROs). The resulting 2-level forms can in some cases even challenge the best known ESOPs. This especially becomes interesting as a starting point for further 2- or multi-level minimization and synthesis.

2 Synthesis for Cellular Architectures

In this section we briefly review previous work on synthesis for *Cellular Architectures* (CAs), and outline our extension applicable for general LUT FPGAs.

*The co-operative work was conducted at Albert-Ludwigs-University on a grant from the Swedish Institute.

2.1 Previous work

In [10] a comprehensive synthesis method for CAs was presented. The resulting rectangular structure implements a *Complex Maitra Logic Array* (CMLA) having a “complex” input plane and a “collecting” output plane. The structure is similar to that of an AND-EXOR PLA, but in the CA each “row” can implement a broader class of functions, hence possibly reduce the number of required “rows”. A forward (reverse) Maitra term is a factored form recursively defined as; $M\alpha a$ ($a\alpha M$) where M is a forward (reverse) Maitra term, a is a literal occurring only once in the expression and α is a Boolean function of two arguments. $M_{forward}\alpha M_{reverse}$ where each literal occurs only once in the expression forms a bidirectional Maitra term. The “complex” terms used in [10] are forward, reverse or bidirectional Maitra terms, derived by the use of a “cube” based ESOP minimizer. This factored form fits directly onto a “row” in CAs where each cell computes a 2-input function, thus a regular structure is obtained. Another approach to derive Maitra terms was presented in [5], where Lee adopts an elegant DD based method similar to that used for PSDKRO minimization.

2.2 Synthesis from multi-valued PSDKROs

The approaches described in the previous section are efficiently utilizing only CAs where each cell implements a 2-input function. In the following we outline a synthesis method based on multi-valued PSDKRO minimization applicable to available $k_{in}-p_{out}$ -LUT FPGA architectures (see Figure 1).

Each product in a k -valued PSDKRO expression, can be computed by a CA of k_{in} -LUTs (analogously to the 2-valued case [5]) as shown in Figure 2 “Input Plane”. The resulting output function(s) can be computed as the EXOR of such products (see Figure 2 “Output Plane”). Many available FPGAs (e.g., Xilinx C3000, C4000, AT&T Orca etc.) can compute p ($p > 1$) output functions, under the constraint that some inputs are shared. As shown in Figure 2, the $k - p$ input variables (encoding a MV variable) are common to each LUT, thus meeting the sharing constraint. Besides for the input variables all interconnections are ensured to be local. E.g., this layout allows us to efficiently utilize “direct interconnects” between cells and “long lines” for the input variables in the Xilinx architectures. In general, synthesis for a $k_{in}-p_{out}$ -LUT architecture is performed by minimization of 2^{k-p} -valued PSDKROs. An upper bound of required cells for a function f is given by $(n_{cells} + m_{cells}) * R_{cells}$:

$$\begin{aligned} n_{cells} &= \lceil f_{nr_in} / (k - p) \rceil && \text{(Input Columns)} \\ m_{cells} &= \lceil f_{nr_out} / p \rceil && \text{(Output Columns)} \\ R_{cells} &= \lceil |2^{k-p}\text{-valued PSDKRO}(f)| / p \rceil && \text{(Rows)} \end{aligned}$$

This outlines a first “naive” approach to CA synthesis from MV PSDKROs. By extending Maitra term factorization [5] to the MV case we expect to reduce the number

Figure 1. $k_{in}-p_{out}$ -LUT FPGA cell.

Figure 2. Resulting regular layout.

of cells. Furthermore we can apply PLA-like folding techniques [10] in the “Input Plane”, under the conditions that routing resources are sufficient and that the “Output Plane” manages to collect the result. Figure 2 (bottom) shows possible folding for $4_{in}-1_{out}$ -LUTs. The extension to don't care assignment [6] for MV PSDKRO minimization is also an interesting topic for future research.

3 Preliminaries on Pseudo Kronecker Expressions

In the first subsection we briefly review the essential definitions of (2-valued) *Pseudo Kronecker Expressions* (PSDKROs). (For more details see [13, 3].) In the next subsection we discuss the properties of MV PSDKRO expressions, details can be found in [13].

3.1 2-Valued PSDKRO Expressions

Let f_0 (f_1) denote the *cofactor* of f with respect to x^0 (x^1) and f_2 is defined as $f_2 = f_0 \oplus f_1$, $\oplus$ being the Exclusive OR operation. A Boolean function $f : B^n \rightarrow B$ can then be represented by one of the following formulae:

$$f = x^0 f_0 \oplus x^1 f_1 \quad \text{Shannon (S)} \quad (1)$$

$$f = f_0 \oplus x^1 f_2 \quad \text{positive Davio (pD)} \quad (2)$$

$$f = f_1 \oplus x^0 f_2 \quad \text{negative Davio (nD)} \quad (3)$$

If we apply to a function f either S , pD or nD we get two sub-functions. To each sub-function again S , pD or nD can be applied. This is done until constant functions are reached. If we multiply out the resulting expression we get a 2-level AND/EXOR form, called a PSDKRO.

The decompositions are applied with respect to a fixed variable ordering. Note that the choice of the variable ordering in which the decompositions are applied and the choice of the decomposition per sub-function largely influences the size of the resulting representation, and may vary from linear to exponential.

3.2 Multi-Valued PSDKRO Expressions

In the general v -valued case a variable $x = \{0, \dots, v-1\}$ is encoded by k Boolean variables where $k = \lceil \log_2(v) \rceil$. In the following we assume $v = 2^k$. Let $f_0, \dots, f_{v-1}$ denote the *cofactors* of f with respect to $x^0, \dots, x^{v-1}$. A Boolean function $f : (B^k)^n \rightarrow B$ is represented by $f = x^0 f_0 \oplus \dots \oplus x^{v-1} f_{v-1}$ (corresponding to the MV Shannon expansion). The number of possible EXOR combinations of the cofactors is $2^v - 1$. E.g., in the 4-valued case the 15 successors $(F_1, \dots, F_{15}) = (f_{\{0\}}, f_{\{1\}}, f_{\{0,1\}}, f_{\{2\}}, f_{\{0,2\}}, f_{\{1,2\}}, f_{\{0,1,2\}}, f_{\{3\}}, f_{\{0,3\}}, f_{\{1,3\}}, f_{\{0,1,3\}}, f_{\{2,3\}}, f_{\{0,2,3\}}, f_{\{1,2,3\}}, f_{\{0,1,2,3\}})$, in the 16-valued case the 255 successors $(F_1, \dots, F_{255}) = (f_{\{0\}}, \dots, f_{\{0,1,2,3,4,5,6,7\}})$, and in the general case the $2^v - 1$ successors $(f_{\{0\}}, \dots, f_{\{0, \dots, v-1\}})$.

To derive the number of PSDKRO expansions (i.e., non-linear combinations of successors) in the general v -valued case one would need to consider $(2^v - 1)$ possible combinations. In [13] a computer enumeration for the 4-valued case shows that 840 expansions exist, but the problem becomes computationally intractable by enumeration for $v > 4$.

If we apply a v -valued decomposition to f , we get v sub-functions, for which we can recursively apply v -valued decompositions until constant functions are reached.

As for the 2-valued case, the decompositions are applied with respect to a fixed variable order. Both the choice of decompositions applied and the order of which the decompositions are carried out will affect the size of the resulting representation. If we start out from a 2-valued function $f : B^{(k^n)} \rightarrow B$ the choice of grouping the 2-valued variables into MV variables will also largely influence the result.

3.3 PSDKRO Minimization

In this section we review previously presented (DD based) algorithms for minimizing the number of products for PSDKRO expressions.

In [13] a method for exact 2-valued PSDKRO minimization based on ETDDs was presented. The three successors f_0 , f_1 and $f_2 = f_0 \oplus f_1$ are stored for each node. It has been proven that the memory complexity is $O(3^n/n)$. A similar minimization method based on *Ordered Binary Decision Diagrams* (OBDDs) [2] has been presented in [3], having lower cost for each node as f_2 is not stored. In [7] this method was extended with heuristic strategies for

Figure 3. Pruning of search space

search space pruning, tunable to trade off quality for computational resources.

In the DD representation of f a *1-path* represents a product term. The basic minimization algorithm recursively traverses f from the top towards the terminals. At each node an EXOR operation is carried out (or prior to minimization in the ETDD approach) which by the use of OBDDs can be performed in polynomial time [2]. From the fact that for each of the decomposition formulae from Section 3.1 only two out of the three possible successors f_0 , f_1 and f_2 are needed to represent the function, we can choose the two least costly in order to minimize the resulting PSDKRO. This can easily be formulated by a simple recursive algorithm on OBDDs [3].

For 4-valued PSDKRO minimization an extension of the ETDD approach was presented in [13]. All 15 successors are recursively (and exhaustively) computed and stored in an *Ordered Penta-decimal Decision Diagram* (OPDD). Out of the 840 possible 4-valued expansions, the one having the least cost is chosen for each node in the diagram. In [12] a heuristic method based on extended truth tables was presented, greedily choosing the decomposition that minimizes the resulting expression.

The quality of the resulting PSDKROs depends on the variable ordering applied as well as the grouping of input variables in the MV case. In [13] a heuristic from [11] was applied (in a preprocessing step) to group 2-valued variables into 4-valued variables. In [7] dynamic reordering strategies are shown to drastically reduce the (2-valued) PSDKROs in many cases.

4 New PSDKRO Minimization methods

In this section we first introduce pruning techniques based on cost estimation for 2-valued PSDKRO minimization, and show how both exact and heuristic results can be obtained. In Section 4.2 we present the extension to 4-valued PSDKRO minimization for heuristic results, and finally we give a method for heuristic v -valued PSDKRO minimization in Section 4.3.

4.1 2-valued PSDKRO Minimization

To the original algorithm [13, 3] we have added a number of new features; the use of cost estimation, a pruning

```

prod psdkro(node  $f$ , int  $prune$ ) {
1  if ( $f == 0(1)$ ) return  $0(1)$ ;
2  if ( $f.prod$  defined) return  $f.prod$ ;
3  derive cofactors  $f_0, f_1$  and compute  $f_2 = \text{EXOR}(f_0, f_1)$ ;
4  compute cost estimates  $p_0^{est}, p_1^{est}$  and  $p_2^{est}$ 
5  sort successors in increasing cost order  $p_a^{est} \leq p_b^{est} \leq p_c^{est}$ 
6   $p_a = \text{psdkro}(f_a, prune)$ ;
7  (greedy)  $\rightarrow$  if ( $p_a \geq prune$ ) return MAXINT;
8   $p_b = \text{psdkro}(f_b, prune)$ ;
9  if ( $\min(p_a, p_b) \geq prune$ ) return MAXINT;
10  $p_c = \text{psdkro}(f_c, \text{pf}(prune, p_a, p_b))$ ;
11  $f.prod = p_a + p_b + p_c - \max(p_a, p_b, p_c)$ ;
12 return  $f.prod$ ;
}

```

Figure 4. Algorithm using cost estimation.

parameter and the possibility to obtain heuristic results by a greedy approach (the latter two also appears in [7]).

We make the following observations: The OBDD representation holds an initial (non-optimal) PSDKRO solution having only Shannon decompositions ($x^0 f_0 \oplus x^1 f_1$) with the cost $p_0 + p_1$. Assume we have computed the exact costs for p_a and p_b (for f_a and f_b respectively), then f_c is a part of the solution iff $p_c < \max(p_a, p_b)$, i.e., f_c must be less costly than at least one other successor f_a or f_b (see Figure 3 (a)). Given a cost limit “*prune*”, f_c is a part of the solution iff $p_c < prune - \min(p_a, p_b)$, since the cost of such a decomposition is $\min(p_a, p_b) + p_c$, which in turn must be less than “*prune*” (see Figure 3 (b)). Our hands on experiences show that a high cost for f_a or f_b in many cases also leads to a high cost for f_c .

These observations are exploited by the algorithm shown in Figure 4. The additional parameter “*prune*” gives a cost limit for the (sub)function under consideration. The cost (i.e., *l-paths*) of the initial OBDD representations for f_0, f_1 and f_2 give upper bound cost estimates p_0^{est}, p_1^{est} and p_2^{est} (line 4). The successors are ordered (line 5) such that $p_a^{est} \leq p_b^{est} \leq p_c^{est}$, i.e., f_a has the lowest upper bound, hence considered as the most promising successor (line 6). As each decomposition (S, pD or nD) requires at least one of f_a or f_b further computation can be aborted when $\min(p_a, p_b) \geq prune$ (line 9). By returning *MAXINT*, the function is ensured not to become part of the final result. The function “pf” (line 10) combines the two observations (a) and (b) as the new pruning value, i.e., $\min(\max(p_a, p_b), prune - \min(p_a, p_b))$. This algorithm favors promising parts of the search space and rejects pointless computations at an early stage without compromising the quality of the result. However, from our last observation above, inspecting the most promising successor f_a gives a rough estimate of the cost. The *greedy* approach in line 7 utilizes this estimation to heuristically abort unpromising calculations. For each node f the minimal number of product terms needed for the representation as a PSDKRO is stored in the variable $f.prod$. Thus, each node has to be evaluated only once. Other pruning aspects, simultaneous minimization of literal

```

prod psdkro_4v(node  $f$ ) {
  if ( $f == 0(1)$ ) return  $0(1)$ ;
  if ( $f.prod$  defined) return  $f.prod$ ;
  derive cofactors ( $f_0, f_1, f_2, f_3$ )
  compute successors ( $F_1, \dots, F_{15}$ )
  compute cost estimates ( $p_1^{est}, \dots, p_{15}^{est}$ )
  derive optimal decomposition according to estimates
  select ( $F_a, F_b, F_c, F_d$ ) correspondingly
   $f.prod = \text{psdkro\_4v}(F_a) + \text{psdkro\_4v}(F_b) +$ 
     $\text{psdkro\_4v}(F_c) + \text{psdkro\_4v}(F_d)$ ;
  return  $f.prod$ ;
}

```

Figure 5. 4-valued PSDKRO minimization.

count and the effect of complemented edges for the implementation of the algorithm are further discussed in [7].

4.2 4-Valued PSDKRO Minimization

In [13], only results for small benchmarks are reported, which confirms the intractability of the exact minimization approach using OPDDs.

Instead of recursively computing the exact 4-valued PSDKRO for all the 15 possible successors of each node in order to find the optimal decomposition, we choose the decomposition having the least estimated cost, i.e., derived from the estimated cost of 4 linearly independent successors according to decomposition. As a rough cost estimate the upper bound can be used, derived as the number of products, i.e., *l-paths* in a 4-valued DD representation. In our implementation all the 840 possible 4-valued PSDKRO decompositions are pre-computed in order to speed up the selection. A sketch of the heuristic 4-valued minimization algorithm is given in Figure 5.

4.3 Multi-valued PSDKRO Minimization

For general v -valued PSDKRO minimization, we have to tackle several problems. Firstly the number of successors is $2^v - 1$, thus for large v becomes intractable to compute. Secondly, the number of v -valued expansions has not been enumerated for $v > 4$. The latter problem is the most critical, since we first seek a method for 8- and 16-valued PSDKROs for the FPGA synthesis application.

Our approach to the problem is to guide the search for valid expansions (i.e., non-linear combinations of the successors) from the estimated costs of the successors. As for the 4-valued case the upper bound gives a rough estimate, and is in the general case derived from the number of *l-paths* in the v -valued DD representation. By iteratively adding v successors having the least estimated cost (preserving non-linearity), a valid expansion is obtained.

Practical experiences for 16-valued PSDKRO minimization show that the main computational cost lies within the search for expansions, not the computation of successors or the cost estimates. A sketch of the algorithm is given in Figure 6.

```

prod psdkro_mv(node f, v) {
  if (f == 0(1)) return 0(1);
  if (f.prod defined) return f.prod;
  derive cofactors ( $f_0, \dots, f_{v-1}$ )
  compute successors, ( $F_1, \dots, F_{2^v-1}$ );
  compute cost estimates ( $p_1^{est}, \dots, p_{2^v-1}^{est}$ )
  f.prod = 0; for ( $i = 1 \dots v$ ) {
     $F^i$  = successor having least  $p^{est}$ ;
    if (non-linear combination( $F^1, \dots, F^i$ ))
      f.prod = f.prod + psdkro_mv( $F^i, v$ );
  } return f.prod;
}

```

Figure 6. MV PSDKRO minimization.

4.4 Influence of Variable Ordering

In [13] the effect of DD variable ordering was concluded by enumerating all possible orderings for a given function and minimizing the corresponding PSDKROs. However, no method to approach the ordering problem was reported. In general dynamic variable ordering [9] has proven useful to many DD problems. In [7] dynamic reordering strategies (“move to top” and “sifting”) are applied to 2-valued PSDKRO minimization. The first method seeks the best top variable iteratively until no further improvement is obtained, while the latter sifts each variable to the position that results in a local minima. In general the sifting approach is able to derive better results at the cost of computation time.

These strategies can be applied to find a heuristic ordering both for 2- and general v -valued minimization. In the v -valued case, the v -valued variables (encoded by $k = \lceil \log_2(v) \rceil$ 2-valued variables) are moved/sifted together in the diagram, using a “group”-move/sift method. As a cost estimate function we can apply the upper bound, i.e., l -paths in the v -valued DD representation. To find a grouping of the 2-valued variables into v -valued variables, we can initially apply 2-valued reordering to the diagram.

4.4.1 Minimization using “Free” DDs

If we relax the fixed ordering constraint for PSDKROs, we can apply “free” DDs to the minimization. In Figure 7 we outline the “free” reordering method. This approach combines reordering with minimization. Also in this case we have the choice of different cost estimation functions. In Section 5, we show how “free” minimization greatly can improve on PSDKRO results, and even in some cases challenges best known ESOPs.

5 Experimental Results

In this section we present experimental results for a set of MCNC benchmark functions. All run-times are given in CPU seconds on a *Sun Ultra 1* workstation with 256Mb RAM. All experiments have been carried out on the BDD package from [15] and we set a runtime limit of 1000 CPU seconds.

```

prod free_mv(node f, v) {
  if (f == 0(1)) return 0(1);
  if (f.prod defined) return f.prod;
  find  $f'$  (having top variable minimizing  $f'^{est}$ )
  f.prod = 0; for ( $i = 1 \dots v$ ) {
    choose  $F^i$  according to decomposition for  $f'^{est}$ 
    f.prod = f.prod + free_mv( $F^i, v$ );
  } return f.prod
}

```

Figure 7. Free minimization.

In a first series of experiments given in Table 1, we demonstrate the effect of pruning by the use of cost estimation. The results are obtained for the variable ordering given from the PLA benchmarks. To allow comparison to previously reported best known results [13, 3] (marked *ex*), multiple output functions are encoded by Output Selection (OS) nodes placed at the bottom of the diagram, thus ensuring minimization of all outputs simultaneously [7]. Columns marked *est* and *greedy* report the results for the proposed 2-valued minimizer. For comparison column *MV* gives the results for the general MV PSDKRO minimizer applied to the 2-valued case. The *greedy* approach obtains minimal or near minimal results while saving computational resources.

In the next set of experiments given in Table 2, we show the effect of “free” ordering of non-symmetrical benchmarks. In column *MINT* the number of products for best previously known 2-valued ESOPs [4] are shown. During reordering either *est* or *MV* is applied to give a cost estimate. We are in many cases able to vastly improve on previously reported PSDKRO results (Table 1) and even in some cases challenge best known ESOPs, while being magnitudes faster. Inherent in the DD structure is also the possibility for multi-level minimization and synthesis. By also considering single-output minimization [7] we expect to further improve the results.

In the last set of experiments given in Table 3 we show the extension to MV PSDKRO minimization and report the first results for up to 16-valued PSDKROs. “-” indicates that time limit was reached. “*” indicates a first exact 4-valued result. The initial orders given by the PLAs are used for input grouping and MV variable order. Columns *est* and *MV*, show the heuristic 4-V and the general MV minimization results. While the greedy search of expansion used in the general MV minimizer (*MV*) consumes considerably less computational resources, the quality of the result equals that of the exhaustive search method (*est*). Columns marked *free* shows the result of the “free” reordering method. Note that the grouping of input variables into MV variables remains unchanged during reordering. For 8- and 16-valued minimization the drastic pruning of the search space may compromise the quality, and a “grouping” approach from e.g. 2-valued PSDKROs may be applicable.

The presented results are directly applicable to the combined minimization, mapping and routing method given

name	Number of Products			CPU Time in Seconds			
	<i>ex/est</i>	<i>greedy</i>	<i>MV</i>	<i>ex</i>	<i>est</i>	<i>greedy</i>	<i>MV</i>
5xp1	47	47	53	0.01	0.02	0.02	0.01
add6	132	132	132	0.03	0.11	0.10	0.01
bc0	180	180	184	5.35	2.61	2.22	0.06
co14	14	14	14	0.01	0.01	0.01	0.01
duke2	108	108	110	25.05	20.91	19.68	0.04
in2	117	117	137	2.50	2.31	2.06	0.34
in7	42	42	43	0.17	0.13	0.13	0.01
inc	31	31	31	0.01	0.01	0.01	0.01
intb	500	500	638	0.71	0.85	0.80	0.08
misex3	754	754	877	5.08	5.39	4.52	0.19
rd53	20	20	31	0.01	0.01	0.01	0.01
rd73	63	63	127	0.01	0.01	0.01	0.01
rd84	107	107	255	0.01	0.01	0.01	0.01
sao2	41	41	46	0.01	0.02	0.02	0.01
t481	13	13	13	0.01	0.01	0.01	0.01
tial	939	940	1247	1.28	1.82	1.72	0.18
x6dn	104	104	107	5.40	2.06	1.78	0.02

Table 1. Pruning by cost estimation.

name	“free” prod		“free” CPU		ESOP prod	
	<i>est</i>	<i>MV</i>	<i>est</i>	<i>MV</i>	<i>MINT</i>	<i>CPU</i>
5xp1	41	40	0.06	0.06	32	10.10
add6	132	132	0.56	0.51	127	139.80
bc0	171	171	83.50	29.75	168	742.17
duke2	85	84	553.19	90.84	78	174.16
in2	115	117	36.85	6.74	108	438.36
in7	35	35	2.94	2.44	35	19.68
inc	30	31	0.04	0.02	27	10.22
intb	308	379	12.60	4.39	267	774.61
misex3	610	642	76.91	9.85	502	10744.99
sao2	33	34	0.09	0.08	27	8.58
t481	13	13	0.21	0.05	13	94.40
tial	606	662	21.12	7.22	425	6779.55
x6dn	99	99	122.88	7.88	95	520.81

Table 2. Best known number of products.

in Section 2.2, e.g. $4_{in}-2_{out}$ -LUT FPGAs are synthesized from 4-valued PSDKROs. By the use of cost estimation we can prune the search space for MV PSDKROs, thus obtain results for previously intractable problems. Furthermore we propose a “free” DD based minimization method for 2-level expressions, in some cases challenging best known ESOPs while being magnitudes faster. Its application to multi-level synthesis is focus of current research.

References

- [1] S.D. Brown, R.J. Francis, J. Rose, and Z.G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publisher, 1992.
- [2] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.
- [3] R. Drechsler. Pseudo Kronecker expressions for symmetric functions. In *VLSI Design Conf.*, pages 511–513, 1997.
- [4] T. Kozłowski, E. L. Dagless, and J. M. Saul. An enhanced algorithm for the minimization of exclusive-or sum-

name	<i>in/out</i>	4-Valued				8-V		16-V
		<i>ex</i>	<i>est</i>	<i>MV</i>	<i>free</i>		<i>free</i>	
5xp1	7 / 10	40	43	43	35	41	32	22
add6	12 / 7	132	132	132	132	132	132	—
bc0	26 / 11	153*	162	161	150	151	148	—
co14	14 / 1	7	7	7	7	5	5	4
duke2	22 / 29	108*	109	109	86	107	80	—
in2	19 / 10	93*	96	96	95	113	110	—
in7	26 / 10	41*	42	42	37	43	36	—
inc	7 / 9	31	31	31	30	29	29	29
intb	15 / 7	345*	392	392	335	373	352	—
misex3	14 / 14	630	747	747	601	472	444	—
rd53	5 / 3	10	10	10	10	7	7	5
rd73	7 / 3	25	26	26	26	13	13	9
rd84	8 / 4	34	36	36	36	21	21	11
sao2	10 / 4	39	43	43	34	31	30	25
t481	16 / 1	9	9	9	9	9	9	3
tial	14 / 8	544*	776	776	541	871	594	—
x6dn	39 / 5	88*	93	93	89	89	88	—

Table 3. MV PSDKRO Minimization.

- of-products for incompletely specified functions. In *Int'l Conf. on Comp. Design*, pages 244–249, 1995.
- [5] G. Lee. Logic synthesis for cellular architecture FPGAs using BDDs. In *ASP Design Automation Conf.*, pages 253–258, 1997.
 - [6] 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.
 - [7] P. Lindgren, R. Drechsler, and B. Becker. Improved minimization methods of pseudo kronecker expressions. In *Int'l Symp. Circ. and Systems*, 1998.
 - [8] R. Murgai, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. *Logic Synthesis for Field-Programmable Gate Arrays*. Kluwer Academic Publisher, 1995.
 - [9] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.
 - [10] A. Sarabi, N. Song, M. Chrzanowska-Jeske, and M.A. Perkowski. A comprehensive approach to logic synthesis and physical design for two-dimensional logic arrays. In *Design Automation Conf.*, pages 321–326, 1994.
 - [11] T. Sasao. Input variable assignment and output phase optimization of PLA's. *IEEE Trans. on Comp.*, 33:879–894, 1984.
 - [12] 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.
 - [13] 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.
 - [14] 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.
 - [15] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.1.2*. University of Colorado at Boulder, 1997.