Free Kronecker Decision Diagrams and their Application to Atmel 6000 FPGA Mapping.

Philip Ho, and Marek A. Perkowski
Department of Electrical Engineering, Portland State University
P.O. Box. 751, Portland, OR, 97207 U.S.A.

Abstract

This paper introduces the concepts of Pseudo-Kronecker Decision Diagrams (PKDDs) with Negated Edges, as well as Free Kronecker Decision Diagrams (FKDDs), that generalize both the well-known Binary Decision Diagrams and Functional Decision Diagrams, as well as the recently introduced Ordered Kronecker Decision Diagrams (OKDDs). We give efficient algorithm for the generation of FKDDs for multi-output functions and show their application to FPGA mapping. On MCNC benchmarks we demonstrate the advantage of FKDDs in terms of reduced numbers of nodes (cells) and levels in the circuit over the OKDDs and Permuted RM Trees. The mapping algorithm can be easily adopted to other cellular FPGAs, especially those from Motorola.

1 Introduction.

Binary Decision Diagrams (BDDs) proved to be a very efficient representation of Boolean functions and have been used with success in logic synthesis, verification, simulation, test generation, and machine learning. There is recently a quickly increasing interest in various kinds of decision diagrams that generalize the BDDs. While the BDDs are based on Shannon Expansion, the Functional Decision Diagrams (FDDs) [9] use the positive Davio expansion [5]. The ideas of BDDs and FDDs have been combined in [13,7,16] to create the Ordered Kronecker Decision Diagrams (OKDDs), where three types of expansions are used in nodes: Shannon, positive Davio, and negative Davio. Such KDDs have been shown to be a more efficient canonical representation of Boolean functions than the BDDs [7] and are also useful in multilevel circuit synthesis and mapping, especially for new "cellular" or "fine-grain" FPGAs from Atmel and Motorola.

Similarly to BDDs, the levels of canonical OKDDs correspond to input variables. Moreover, at every level all nodes are expanded with respect to the same type of expansion: Shannon, positive Davio or negative Davio. It was observed in [16] that essential improvements in both the number of levels and the number of nodes (cells) are obtained when the nodes in a level of a diagram (corresponding to the same variable), have various expansions. We call such diagram a Pseudo-Kronecker Decision Diagram (PKDD), per analogy to the Pseudo-Kronecker Reed-Muller canonical forms that are obtained by flattening this diagram [14]. In this paper we further improve the results by introducing the concept of Free Kronecker Decision Diagrams (FKDDs), in which there can be arbitrary orders of variables along various branches of the diagram. This has direct application in mapping to FPGAs, and can also lead to the development of canonical FKDDs to be used as a general-purpose Boolean function representation.

Most of logic synthesis methods developed for FPGAs have been based on algebraic factorization methods [3]. However, it is known that logic synthesis methods based on Boolean decomposition methods can produce better results [3]. Moreover, those core CAD tools have been based on the "unate paradigm". The "unate paradigm" is the assumption that most of the logic functions ocuring in logic design are unate or nearly unate. The meaning of "unate" or "nearly unate" for logic minimization purposes is, that the circuit realization of a nearly unate function with AND and OR gates is smaller in terms of the numbers of gates than that of a circuit using the AND and EXOR gates. On the other hand, the meaning of "linear" or "nearly linear" for logic minimization purposes is that the circuit realization of a nearly linear function with AND and EXOR gates is smaller in terms of the numbers of gates than that of a circuit using the AND and OR gates. Arithmetic function like counters, adders, multipliers, signal processing functions and error correcting logic belong to the class of nearly linear functions. Thus those functions will have a smaller circuit realization if the EXOR gate is incorporated into the design.

The synthesis incorporating the EXOR gate has been neglected because the EXOR gate was perceived to be slower and having a larger circuit area. However, those upcoming FPGAs from Xilinx, Actel, and Atmel allow the implementation of the EXOR gate without any speed or circuit size penalty in comparison to the AND and OR gates. Since the Atmel cells directly realize the set of functions used in KDDs, these expansions can be easily applied to this type of FPGAs. The basic cell of the AT 6000 series can be programmed to one-bit multiplexer and the three-input AND/EXOR cell. Use of EXOR-based diagrams has also particular advantages for machine learning [10].

[0]†The work presented in this paper was partially supported by NSF grant MIP-9110772.

The initial phase of many logic synthesis systems, such as MISII and BOLD, restructures the original network to reduce a cost function that is calculated directly from the network itself. The intention is to improve the final circuit by reducing the complexity of the network. In this phase, the method does not consider the type of element that will be used for the final circuit. After the initial phase which produces the optimized network, the technology mapping stage transforms this network into the final circuit. This is done by selecting pieces of the network that can be implemented by one of the available circuit elements and specifying how these elements are to be interconnected. The circuit is optimized to reduce a cost function that typically incorporates measures of both the area and delay. Although very general, this approach seems to be not well suited to fine-grain FPGAs, that have special gates and connection structures.

Another approach to FPGA synthesis is based on general-purpose Boolean Decomposition [18]. It does not assume anything about the gate realization and in general gives better results, but tends to be slow for large functions. The approach proposed here is just a very special case of the Boolean Decomposition, but it is very fast and it uses extensively EXOR gates. It can have thus two main applications: it can be used directly for mapping to FPGAs that have EXOR gates, or it can be used, similarly to BDDs, as only the first stage of logic design in which creation of a decision diagram is followed by its mapping to a particular technology, not necessarily one including EXOR gates.

The plan of this paper is as follows. The recent research in applications of DDs to FPGA mapping is reviewed in section 2. The families of decision diagrams are introduced in section 3. In section 4 the description of our FKDD synthesis program and mapper, RESPER, is given. Section 5 compares the results with OKDDs and Permuted RM Trees [16,19].

2 Recent Research Versus Our Approach

ASYL program [2] applies Shannon Expansion to build the BDD of each function. It also uses the Reduced Order Binary Decision Diagrams (ROBDDs) approach to minimize the area. Its target is on Actel's multiplexer-based FPGAs. Its heuristics to select the variable are the following: (1) Select a variable that appears in all product terms under the same polarity. (2) If a product term is restricted to a simple literal then select this literal. (3) If all the variables appear only once in a function then select the smallest product term. (4) Select the set of variables of maximum occurrence.

Few programs have been recently written that use Davio expansions. The REMIT program [19] starts from a completely specified Boolean function in the form of an array of ON disjoint cubes, and generates a free (permuted) tree using positive Davio Expansion. In such tree various orders of variables exist in various branches. The variable selection rules select the variable that occurrs most often in disjoint cubes, one at a time. RMS program [9] uses Positive Davio Expansion to create a new representation called Functional Decision Diagrams (FDDs). It starts with a two level SOP to calculate an order of variables in the FDD according to the most often used variables. The isomorphic subtrees are next reduced. Paper [7] introduces an efficient package to generate OKDDs for multi-output functions. The approach is, however, limited to completely specified functions. TECHMAP program [16] generalizes the concepts of BDDs and FDDs applied to Actel and Atmel FPGAs by generating the Shared Reduced Ordered Kronecker Decision Diagrams (SROKDDs). It adopted a breadth-first top-down algorithm for the SROKDD generation for *incompletely specified multi-output functions.* During the decomposition, it combines all those isomorphic trees in order to generate a SROKDD. Its heuristics to select the variable are based on the following three conditions. All these conditions can determine that the next level node is redundant.

Condition 1: $f_i = 0, \quad f_i = 1, \quad f_i = x_j, \quad f_i = \overline{x_j}$. It states that the data input function f_i is either a constant value, a single variable, or a negation of a variable.

Condition 2: $f_i = f_j$. It states that data input function f_i is identical to input function f_j in the same level of the tree.

Condition 3: $f_i = \overline{f_j}$. It states that data input function f_i is the complement of data input function f_j in the same level of the tree.

TECHMAP's heuristics to select the expansion are divided into three following modes.

C1. The expansion of a node is selected based on the two functions out of f_{x_i}, $\overline{f_{x_i}}$, and $f_{x_i} \oplus \overline{f_{x_i}}$ having the highest fan-out. In case of a tie, the heuristic C3 is applied.

C2. If the variable occurrs mostly in a positive form in the output function, Davio expansion 2 is selected. If the variable occurrs mostly in a negative form, Davio expansion 3 is selected. If there is a tie, the Shannon expansion is chosen.

C3. The expansion of a node is selected based on the two functions out of f_{x_i}, $\overline{f_{x_i}}$, and $f_{x_i} \oplus \overline{f_{x_i}}$ having the least number of product terms.

The main objective of the above FPGA technology mapping approaches was to minimize the area (number of nodes).

We developed the concept of the Free Kronecker Decision Diagram (FKDD), and we applied most of the heuristics from the above papers. Our FPGA mapping techniques try to construct the network in such a way that:

- the decomposed network is technology-feasible for the Atmel devices.
- the number of nodes in the network is as small as possible.
- the path from the input to output is as short as possible.
- the selected variable and expansion can vary in every level of the tree.

The presented method has the following assets:

- The decomposition methods are specifically adapted to the FPGAs whose general architectures

are based on logic cells which can take up to three input variables.

- It applies a set of rules to select a good variable and an appropriate expansion for each node.
- It uses the shared reduced order approach.

3 Families of Decision Trees, Decision Diagrams and Flat Forms

The *literal* of a variable x_i can be in either positive (x_i) or negative ($\overline{x}_i$) form. The *polarity* of a variable is "1" for a positive literal and "0" for a negative literal. Let f_{x_i} be a cofactor [3] of function f with respect to x_i: $f_{x_i} = f \cap x_i \mid_{x_i=1} = f \mid_{x_i=1}$.

It is known [5] that there can be only three expansions over Galois Field of 2:

(1) $f = x_i\, f_{x_i} \oplus \overline{x}_i\, f_{\overline{x}_i}$ (Shannon expansion).

(2) $f = f_{\overline{x}_i} \oplus x_i\, [\, f_{x_i} \oplus f_{\overline{x}_i}\,] = f_{\overline{x}_i} \oplus x_i\, g_i$ (Positive Davio Expansion)

(3) $f = f_{x_i} \oplus \overline{x}_i\, [\, f_{x_i} \oplus f_{\overline{x}_i}\,] = f_{x_i} \oplus \overline{x}_i\, g_i$ (Negative Davio Expansion)

One important property of the three expansions is that the functions f_{x_i} and $f_{\overline{x}_i}$ obtained by applying any of the three expansions for x_i being an input variable of the function f, are independent from the variable x_i. The circuit realization of Equation (1) is given by a multiplexer gate while Equations (2) and (3) describe an AND-EXOR gate of Atmel 6000.

The application of the Shannon expansion, Equation (1), for all variables of a function leads to the construction of a Binary Decision Diagram. The application of the two Davio expansions for each variable generates an adaptive logic tree [9].. The FDD is obtained by applying the reduction procedures used for BDDs [4]. to the adaptive logic tree created using only expansion (2). If all three expansions are applied to all variables, the Kronecker Reed-Muller tree [5,11,12] is obtained.

A *terminal vertex* has an attribute a value $value(v) \in 0, 1$. A *non-terminal vertex* has an attribute an argument index $index(v) \in 1, ..., n$ and two children $low(v)$, $high(v) \in V$. An *ordered function graph* is a function graph such that for any non terminal vertex v, if $low(v)$ is also non terminal, then $index(v) < index(low(v))$. Similarly, if $high(v)$ is non-terminal, then $index(v) < index(high(v))$.

A *Reed-Muller Tree* is a function graph:

- having root vertex v denoting a function f_v defined recursively as:

1. If v is a terminal vertex:
   a. If value(v) = 1, then $f_v = 1$.
   b. If value(v) = 0, then $f_v = 0$.
2. If v is a non-terminal vertex with index(v) = i, the f_v is the function: $f_v(x_1, ..., x_n) = f_{low(v)}(x_1, ..., x_n) \oplus x \cdot [f_{high(v)}(x_1, ..., x_n) \oplus f_{low(v)}(x_1, ..., x_n)]$.

- any path from the root to the terminal vertices will traverse the same order of variables.

A *Permuted Reed-Muller Tree* is a function graph:

- having root vertex v denoting a function f_v defined recursively as:

1. If v is a terminal vertex (as in Reed-Muller Tree).
2. If v is a non-terminal vertex with index(v) = i, the f_v is the function: $f_v(x_1, ..., x_n) = f_{low(v)}(x_1, ..., x_n) \oplus x \cdot [f_{high(v)}(x_1, ..., x_n) \oplus f_{low(v)}(x_1, ..., x_n)]$.

- any path from the root to the terminal vertices can traverse a different order of variables.

A *Fixed-Polarity Reed-Muller Tree* is a function graph:

- having root vertex v denoting a function f_v denoted recursively as:

1. If v is a terminal vertex (as in Reed-Muller Tree).
2. If v is a non-terminal vertex with index(v) = i, the f_v is a one and only of the functions:
   a. $f_v(x_1, ..., x_n) = f_{low(v)}(x_1, ..., x_n) \oplus x \cdot [f_{high(v)}(x_1, ..., x_n) \oplus f_{low(v)}(x_1, ..., x_n)]$.
   b. $f_v(x_1, ..., x_n) = f_{high(v)}(x_1, ..., x_n) \oplus \overline{x} \cdot [f_{high(v)}(x_1, ..., x_n) \oplus f_{low(v)}(x_1, ..., x_n)]$.

- any path from the root to the terminal vertices will traverse the same order of variables.
- for every variable, just one type of expansion is selected.

A Kronecker Reed-Muller Tree is a function graph:

- having root vertex v denoting a function f_v defined recursively as:

1. If v is a terminal vertex (as in Reed-Muller Tree).
2. If v is a non-terminal vertex with index(v) = i, the f_v is a one and only of the functions:
   a. $f_v(x_1, ..., x_n) = f_{low(v)}(x_1, ..., x_n) \oplus x \cdot [f_{high(v)}(x_1, ..., x_n) \oplus f_{low(v)}(x_1, ..., x_n)]$.
   b. $f_v(x_1, ..., x_n) = f_{high(v)}(, ..., x_n) \oplus \overline{x} \cdot [f_{high(v)}(x_1, ..., x_n) \oplus f_{low(v)}(x_1, ..., x_n)]$.
   c. $f_v(x_1, ..., x_n) = \overline{x} \cdot [f_{low(v)}(x_1, ..., x_n) \oplus x \cdot f_{high(v)}(x_1, ..., x_n)]$.

- any path from the root to the terminal vertices will traverse the same order of variables.
- for every variable, just one type of expansion is selected.

A Pseudo-Kronecker Reed-Muller Tree is a function graph:

- having root vertex v denoting a function f_v denoted recursively as:

1. If v is a terminal vertex (as in Reed-Muller Tree).
2. As in Kronecker Reed-Muller Tree.

- any path from the root to the terminal vertices will traverse the same order of variables.
- for every variable, any expansions: Shannon, Positive Davio, and Negative Davio can be applied in various subtrees.

A Permuted-Kronecker Reed-Muller Tree is a function graph having root vertex v denoting a function f_v defined recursively as in the Pseudo-Kronecker Reed-Muller Tree, but any path from the root to the terminal vertices can traverse a different order of variables. In this tree there is then no any of the previous constraints on variables or expansions. The only limitation that remains is that along a branch, every variable is met only once.

By applying the well-known decision diagrams reduction procedures, the above families of trees are transformed to Directed Acyclic Graphs (DAGs) and form the decision diagram families corresponding to

them. One has then the following ordered diagrams: *Functional Decision Diagrams* corresponding to *Reed-Muller Trees*, *Fixed-Polarity Functional Decision Diagrams* corresponding to *Fixed-Polarity Reed-Muller Trees*, *Ordered Kronecker Decision Diagrams* corresponding to *Kronecker Reed-Muller Trees*, Next, one has the following pseudo- type of diagrams: *Pseudo Reed-Muller Decision Diagrams* corresponding to *Pseudo Reed-Muller Trees*, *Pseudo Kronecker Decision Diagrams* corresponding to *Pseudo Kronecker Reed-Muller Trees*, Pseudo Reed-Muller Trees are trees created as Fixed-Polarity Reed-Muller Trees but for every variable both Davio Expansions are allowed. Finally, one has the following free diagrams: *Free Functional Decision Diagrams* corresponding to *Permuted Reed-Muller Trees*, *Free Fixed-Polarity Functional Decision Diagrams* corresponding to *Permuted Fixed-Polarity Reed-Muller Trees*, and *Free Kronecker Decision Diagrams* corresponding to *Permuted Kronecker Reed-Muller Trees*, respectively. Free Fixed-Polarity Functional Decision Diagrams are similar to Fixed-Polarity Functional Decision Diagrams but the order of variables is not fixed in branches.

The obtained trees and diagrams can be flattened to a two level form which can be realized by an AND-EXOR circuit. Flattening is an inverse operation of "substitution". If G is a fan-in function of F, flattening G into F re-expresses F without G. By flattening above forms we obtain both canonical and non-canonical AND/EXOR circuits. For instance, by flattening Functional Decision Diagrams we get *canonical Reed-Muller Forms.* By flattening Fixed-Polarity Functional Decision Diagrams we get *canonical Fixed-Polarity Reed-Muller Forms.* By flattening Ordered Kronecker Decision Diagrams we get *canonical Kronecker Reed-Muller Forms*, and so on. Flattening of the minimal forms of the most general diagrams, FKDDs, leads to highly minimized, non-canonical, most general AND/EXOR expressions, called Exclusive-Or Sum-of-Product Expressions (ESOPs) [17].

4 Description of RESPER

RESPER is a synthesis algorithm for the calculation of FKDDs. It consists of three parts: realization of a *trivial function, expansion selection* and *decomposition.* The *trivial* function is used for the realization of Boolean functions as a cascade circuit, where only one next level module is allowed or no module at all. Expansion selection option determines an appropriate expansion to be chosen for that module in that level. Finally, the RESPER is a mapper especially suited for the AT 6000 series of Atmel.

4.1. Trivial Function Realization.

The basic principle of the level by level minimization algorithm from [15] is to find the minimal number of next level modules for a given level. This approach has been adopted here. A similar principle is used for the realization of Boolean functions as cascade circuits where only one next level module is allowed or no module at all.

There exist six basic conditions for which a next module is redundant.

Condition 1: $f_{x_i} = 0$. If this condition is applied to equation 3, one gets $f = \overline{x_i} \cdot f_{\overline{x_i}}$. We can use an AND gate with one negated input to implement this function, instead of using AND/EXOR gate which has the longest delay in the AT 6000 series. The AT 6000 series does not provide "0" as one of its inputs.

Condition 2: $f_{\overline{x_i}} = 0$. If this condition is applied to equation 2, one gets $f = x_i \cdot f_{x_i}$. We can use an AND gate with one negated input to implement this function, instead of using AND/EXOR gate which has the longest delay in AT 6000 series. And the AT 6000 series does not provide "0" as one of its inputs.

Condition 3: $f_{x_i} = 1$. If this condition is applied to equation 1, one gets $f = x_i \cdot 1 \oplus \overline{x_i} \cdot \overline{f_{x_i}}$. We are able to use one wire less for the inputs to the multiplexer, since the AT 6000 series allows us to select "1" for one of the inputs.

Condition 4: $f_{\overline{x_i}} = 1$. If this condition is applied to equation 1, one gets $f = x_i \cdot f_{x_i} \oplus \overline{x_i} \cdot 1$. We are able to use one wire less for the inputs to multiplexer, since the AT 6000 series allows us to select "1" for one of the input.

Condition 5: a data-input function is identical to another data-input function to a multiplexer in the same level of the tree circuit $f_{x_i} = f_{\overline{x_j}}$. If this condition is applied to the equation 2, one gives $f_{\overline{x_i}}$ and f_{x_i}. If this condition is applied to the equation 3, one gives $f_{\overline{x_i}}$ and f_{x_i}.

Condition 6: a data-input function is the complement of another data-input function to a multiplexer in the same level of the tree circuit $f_{x_i} = \overline{f_{\overline{x_j}}}$. If this condition is applied to equation 2, the resultant function will be $f = f_{\overline{x_i}} \oplus x_i$. If this condition is applied to equation 3, the result will be $f = f_{x_i} \oplus \overline{x_i}$. As we see, it will give less wire connections and less modules for the next level.

In most algorithms only the first five conditions are taken into consideration to decrease the number of next level modules. The case of a data input function being the complement of another data input function has not been taken into consideration in any synthesis algorithm. The advantage of the presented method is, that it also verifies Condition 6. The complement function can be easily realized by an inverter logic cell.

4.2. Variable and Expansion Selection.

The size of the BDD of a function is sensitive to the ordering of the input variables. There has been a tremendous effort for determining a good variable ordering [8]. Some heuristics are based either on the analysis of an exist multilevel netlists [2] or the number of occurrences of the variables [19]. RESPER adopted the synthesis algorithm from [1]. To reduce the solution space for a large function to a space that is computationally feasible, the heuristic searching algorithm allows all three decomposition choices. The heuristic for the variable selection is to select the variable that will obtain less modules in the next level. In order to obtain the result which is as close as possible to the exact solution, the program starts to check all possible variables at each node in each level. Selecting the variable is determined by the set of conditions defined in the previous section.

The expansion selection is limited by the modules availability and their delay times. In AT 6000 series two modules are provided which fit two of the Davio Expansions. One of the modules is a two input multiplexer which is good for equation 1 and the other is the AND/XOR which is good for equation 2. If equation 3 is used we need to add an inverter to the AND/XOR. The time delay is also an important factor for the choice. If there is a tie for the Equation 2 and Equation 3, we need to choose equation 2 since an inverter needs to be added. An added inverter will increase the number of levels of the tree.

The selection of an appropriate variable also creates the backbone for selecting an appropriate expansion. The expansion selections use the same set of conditions as the variable selections. If condition 3 or condition 4 is met, then expansion 1 will be selected. If condition 2 or condition 6 is met, then expansion 2 will be chosen. If condition 1 is met, then equation 3 is chosen. If condition 5 is met, either equation 2 or 3 will create the same result. For condition 6, the program will select either equation 2 or equation 3. Since the objective for this program is to minimize the delay and area, equation 2 will be chosen if condition 6 is met. If condition 5 is met during the expansion and variable selection, the program will stop searching and will select equation 2 for that module. The reason is that this function is independent of the chosen variable, and it does not need a module to represent this function at this level. However, if none of those conditions had been detected, the cost of each expression for each selected variable is calculated. Whichever combination of expansion type and variables provides the least number of minterms will be chosen. This evaluation is performed for each input variable for every of the output functions.

4.3. Shared Functional Decision Diagram.

FKDD is a canonical representation of the functional domain. Each node of the FKDD decides whether the product term belongs to the function or not. Creating an FKDD the following operations are used.

1. Deleting a node whose two edges direct to the same node.
2. Sharing isomorphic sub-graphs.

Multiple FKDDs can be joined into a single SFKDD which consists of the FKDDs sharing their subgraphs. In other words, two isomorphic subgraphs do not coexist in the SFDD. In SFKDD, there is an input inverter added. Its purpose is to swap a positive edge and negative edge at the next node. By using this input inverter, SFKDD will not only reduce the isomorphic subgraphs but also those subgraphs which are inverses of the others. This constraint brings about the following advantages to manipulate a completely specified Boolean functions.

1. The equivalence between two functions can be checked by $F_n \oplus F_m = 0 \qquad n \neq m$
2. The inversion between two functions can be checked by $F_n \oplus F_m = 1 \qquad n \neq m$
3. By sharing sub-graphs we can compactly represent many functions.

4.4. Implementation of RESPER.

The algorithms described above form the core of the decomposition. The task of these subroutines is to assist the whole program to choose an optimal variable order combined with a suitable expansion. The program reads in the disjoint ON cubes written in PLA format. If the input has n output functions, the program divides n output functions into n single output functions. Each output function is stored in n different modules. Starting from module[0], the program computes all input literals and searches for the best selected variable based on those six conditions. After the selected variable is chosen, it generates a module[n+1] and checks for an isomorphic module. If the module[n+1] is isomorphic with other modules, the module[n+1] will be eliminated. The program will go on to module[1], module[2] until there is no module left.

E	I	O	RESPER		TECHMAP			
			m	l	C1	C2	C3	l
adr2	4	3	4	3	6	6	6	3
b12	15	9	50	8	130	147	125	12
cc	21	20	42	6	117	117	117	15
con1	7	2	10	4	16	24	15	5
cu	14	11	27	9	73	73	78	10
inc	7	9	61	6	81	89	76	6
squar5	5	8	18	4	29	35	29	4
f51m	8	8	35	7	52	53	45	7
xor5	5	1	3	3	3	3	3	3
5xp1	7	10	55	6	51	54	44	6
rd53	5	3	12	4	13	14	12	4
bw	5	28	100	4	102	104	102	4
misex1	8	7	32	5	60	39	56	7
misex2	25	18	107	11	282	271	275	23
TOTAL			560		1015	1029	983	

Table 1: RESPER versus TECHMAP

5 Evaluation of Results

RESPER is run on a networked SUN 4/670MP workstation. The results are listed in Table I. All results were verified by the "verify" command of the *MIS-II* system. The results listed under *TECHMAP* are from [18]. They were also run on a networked SUN 4/670MP workstation. The comparison of TECHMAP with ASYL and RMS is in [16]. In Table I, E is the name of the example. I is the number of input variables. O is the number of output functions. m is the number of modules (cells) in the final mapped circuit. l is the longest path that a signal must go from the primary input to the primary output in the circuit. *C1, C2* and *C3* in Table I are the heuristics used by TECHMAP to select the expansion, mentioned in section 2. Based on Table I, the heuristic used should be the best heuristic among those three in TECHMAP. We observe that RESPER generates better results for multi-output functions. Not only is the number of modules smaller, but also the number of levels. For instance, for function *cc* the number of modules is reduced from 117 to 42, and the number of levels from 15 to 6. This improves not only size

but also speed and routability, which is of extreme importance in cellular FPGAs. In a separate study, we compared RESPER with REMIT, which also generates free diagrams, but the diagrams are trees for which and only positive Davio expansion is used. On 27 functions, the total number of modules was 257 for RESPER and 827 for REMIT. Interestingly, for all tested functions, the numbers of levels for RESPER and REMIT were the same.

6 Conclusions and Current Research

The obtained results are very promising and motivate to further investigate the FKDDs and also other free diagrams. Necessary research on FKDDs is to find better heuristics for the expansion type and variable order selection. Similarly to TECHMAP, the RESPER program is now being generalized to incompletely specified functions. Other possible extensions to RESPER include mapping to other new cellular FPGAs, especially those from Motorola. Variants of the method can be also created that will include geometrical information in the FKDD growing process, in order to improve the operation of the interlinked placement/routing programs. Our current research combines logic synthesis for cellular FPGAs with placement/routing into a single comprehensive process that we call "geometrical logic synthesis". We work also on using FKDDs as canonical general-purpose representations of Boolean functions. Assuming *a given FKDD as a pattern* of variable orders and expansion types, such FKDD becomes *canonical,* since FKDDs for other functions are created to follow this pattern. Then, fast comparison of two FKDDs is possible, as well as execution of Boolean and other operations on them that are implemented in DD packages [7].

References.

[1] A.E.A. Almaini, and M.E. Woodward, "An Approach to the Control Variable Selection Problem for Universal Logic Modules," *Digital Processes,* vol. 3, pp. 189-206, 1977.

[2] T. Besson, H. Bousouzou, M. Crates, and G. Saucier, "Synthesis on Multiplexer-based Programmable Devices Using (Ordered) Binary Decision Diagrams," *Proc. EURO-ASIC,* pp. 8-13, June 1992, Paris, France.

[3] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," *Proc of the IEEE,* Vol. 78, No.2, pp. 264-300, February 1990.

[4] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Comput.,* Vol. 35, No. 8, pp. 667-691, August 1986.

[5] M. Davio, J. P. Deschamps, and A. Thayse, "Discrete and Switching Functions," McGraw-Hill, 1978.

[6] D. H. Green, "Families of Reed-Muller Canonical Forms," *Int. J. of Electronics,* Vol. 63, No. 2, pp. 259-280, January 1991.

[7] R. Drechsler , A. Sarabi , M. Theobald, B. Becker , and M. Perkowski, "Efficient Representation and Manipulation of Switching Functions Based on Ordered Kronecker Functional Decision Diagrams," Proc. of DAC'94, San Diego, CA, June 1994.

[8] S.-W. Jeong, B. Plessier, G. D. Hachtel, and F. Somenzi, "Variable Ordering for Binary Decision Diagrams," *Proc. IEEE Euro-DAC* pp. 447-451, March 1992, Brussels, Belgium.

[9] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel Logic Synthesis Based on Functional Decision Diagrams," *Proc. IEEE Euro-DAC,* pp. 43-47, 1992.

[10] Kohavi, R., "Bottom-Up Induction of Oblivious Read-Once Decision Graphs: Strengths and Limitations," AAAI-94.

[11] Perkowski, M. A. and P. D. Johnson, "Canonical Multi-Valued Input Reed-Muller Trees and Forms," *Proc. 3rd NASA Symp. VLSI Des.* Moscow, ID, Oct. 1991, pp.11.3.1-11.3.13.

[12] M. A. Perkowski, "The Generalized Orthonormal Expansions of Functions with Multiple-Valued Inputs and Some of its Applications," *Proc. 22nd ISMVL,* pp. 442-450, May, 1992, Japan.

[13] A. Sarabi, P.F. Ho, K. Iravani, W.R. Daasch, and M. Perkowski, "Minimal Multi-level Realization of Switching Functions based on Kronecker Functional Decision Diagrams", *IWLS '93*

[14] T. Sasao, "Logic Synthesis and Optimization," Kluwer Academic Publishers, 1993

[15] I. Schaefer, and M.A. Perkowski, "Synthesis of Multi-Level Multiplexer Circuits for Incompletely Specified Multioutput Boolean Functions with Mapping to Multiplexer Based FPGAs," IEEE Trans. on CAD, Nov. 1992, pp. 1655-1664.

[16] I. Schaefer, M.A. Perkowski, and H.M. Wu, "Orthogonal Expansions for Multilevel Logic Synthesis and the Technology Mapping to FPGAs," Proc. Int. Workshop on Application of Reed-Muller Expansion to Circuit Design, Hamburg, Germany, Sept, 1993, pp. 42-51.

[17] N. Song, and M.A. Perkowski, "EXORCISM-MV-2: Minimization of Exclusive Sum Of Products Expressions for Multiple-Valued Input Incompletely Specified Functions," Proc. 23rd ISMVL'93, May 1993, pp. 132-137.

[18] Wan, W., and Perkowski, M. A. "A New Approach to the Decomposition of Incompletely Specified Multi-Output Functions Based on Graph-Coloring and Local Transformations and its Application to FPGA mapping," *Proc. IEEE Euro-DAC,* Hamburg, Germany, 1992.

[19] L.-F. Wu, and M. A. Perkowski "Minimization of Permuted Reed-Muller Trees for Cellular Logic Programmable Gate Arrays," *2nd Int. Workshop on FPGAs,* September 1992, Vienna, Austria.