Multilevel Logic Synthesis for Cellular FPGAs Based on Orthogonal Expansions.

Ingo Schaefer, Marek A. Perkowski, and Haomin Wu
Department of Electrical Engineering, Portland State University
P.O. Box. 751, Portland, OR, 97207 U.S.A.

Abstract

The cellular fine grain architectures of new Field Programmable Gate Arrays (FPGAs) require special logic synthesis tools. Therefore, this paper addresses multilevel logic synthesis methods based on orthogonal expansions for such kind of architectures. First, the concepts of the Binary Decision Diagrams (BDDs), their derivatives, and the Functional Decision Diagrams (FDDs), which are applied to the technology mapping to Field Programmable Gate Arrays (FPGAs), are generalized here to the Kronecker Functional Decision Diagram (KFDD) with negated edges. Three other new types of functional decision diagrams are also introduced: If-Then-Else KFDDs, Mixed KFDDs, and Permuted KFDDs. The introduced KFDD is based on three orthogonal (Davio) expansions and is created here also for multi-output incompletely specified Boolean functions. Next, the KFDD creating algorithm being a basis of heuristic mapping program to ATMEL 6000 cellular FPGAs is also presented. Besides this application, our algorithm was adopted for technology mapping to Multiplexer-Based (MB) and Table-Look-Up (TLU) based architectures.

1 Introduction.

In the sixties and seventies there was a large interest in the logic synthesis for cellular cascades [4, 46, 18] and arrays [26]. However, the mentioned work was only theoretical and no synthesis tools or hardware architectures based on the presented theories had been developed.

Recently, the CLI 6000 series from Concurrent Logic (now AT 6000 of Atmel) [13] has been brought to market. The AT 6000 FPGA series are fine grain FPGAs (FGFPGAs) with a cellular architecture and very limited routing resources. There are also other similar chips, fabricated or in development, by Algotronix [2], Toshiba, Plessey, Pilkington, Motorola, National Semiconductor, other companies and Universities. Thus, there is a quickly growing need for logic synthesis tools that would take advantage of such kind architectures.

We are familiar with six groups of methods to synthesize Boolean functions in cellular logic. (1) The first group are special methods developed in the past [27]. They include one-dimensional arrays (single- and multi-rail Maitra cascades and their extensions), and two-dimensional arrays (Reed-Muller arrays, minterm arrays, etc). These methods have been created for too much restricted models of cellular array, give non-efficient results, or are not applicable to all Boolean functions. (2) The second group of methods is to create a sum, a Maitra cascade, or in general an arbitrary one-dimensional cascade of one-dimensional cascades. These methods are very well suited to AT 6000 devices and our group is working on them. (3) The third model are the factorized AND/OR circuits [8], the factorized AND/EXOR circuits, [38] or other multilevel AND/OR/EXOR circuits, which are next mapped to respective libraries by MIS II or similar mappers. It seems, however, from our preliminary results that this approach gives results that produce both an excessive number of gates and are very hard to place and route. (Routing in the AT 6000 is done only by programming the cells to become "wires"). The cell is connected only to 4 neighbor cells and one of four buses). (4) The approach from [44, 45] creates new kind of AND/OR/EXOR factorization algorithm that significantly restricts search by taking the cellularity into account during factorization. (5) The approach from [47] uses the generalized Boolean decomposition followed by mapping. (6) The approach from [34, 33, 32, 48, 36, 49, 37] uses orthogonal expansions to create trees and DAGs. This approach will be the subject of this paper.

It has been observed [28, 7], that logic synthesis and technology mapping methods developed for Application Specific ICs are not suitable even for non-cellular FPGAs such as those from Actel [1], and Xilinx [51]. Therefore, to accommodate the need for the specific synthesis tools for different FPGA architectures, several synthesis tools have been developed [42, 23, 16, 30, 7, 11, 22, 5, 48, 47, 40]. Most of the logic synthesis methods for FPGAs are based on the computation of a (quasi) minimal Reduced Ordered Binary Decision Diagram (ROBDD) [16, 28, 7, 11], or the If-Then-Else Directed Acyclic Graph (ITE DAG) [30, 22]. Like the logic synthesis tools for ASICs, these FPGA synthesis methods are based on the "unate paradigm" [9]. The "unate paradigm" is the assumption that most of the practical logic functions occurring in logic design are *unate* or *nearly unate*. The

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

meaning of *unate* and *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 number of gates than 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 number of gates than a circuit using the AND and OR gates. Therefore, arithmetic functions like counters, adders, multipliers, signal processing functions and error correcting logic that belong to the class of *(nearly) linear* functions [9, 20, 50], can not be efficiently minimized for circuit's speed and area. To address these deficiencies, the concepts of: Functional Decision Diagrams (FDDs) [7], Reed-Muller trees [48], and KFDDs [41, 36] have been developed and applied to FPGA mapping. Each node in a BDD represents a two-input multiplexer. Thus, with a multiplexer being an universal logic module, any function of two inputs can be realized. On the other hand, a node in the FDD represents the AND-EXOR gate combination. As it will be shown in the next section, both the two-input multiplexer and the AND-EXOR gate can be readily realized with a cell of the AT 6000 series.

To generalize and extend the methods based on ROBDDs and FDDs we developed the concept of the Kronecker Functional Decision Diagram (KFDD), and a logic synthesis algorithm for the computation of such diagrams. While a ROBDD is obtained by applying the Shannon expansion, and a FDD by applying the first Davio expansion, the KFDD is generated by applying any of the three possible orthogonal expansions of a single variable [35]. Each node of the KFDD can be readily realized with a cell of the AT 6000 series. As for the ROBDDs and the FDDs, the main problem to obtain a close to minimal circuit realization by using synthesis methods based on KFDDs is to obtain a good input variable ordering [17, 21, 7, 48]. For the selection of the input variables we investigate a top-down, breadth-first generation of the KFDD where the selection method was inspired by the multiplexer synthesis method introduced in [3, 24]. Three heuristics are investigated to determine the expansion of a node in the KFDD that leads to a smaller KFDD. The tree expansion is performed for a multi-output function treated as a vector of single-output functions, which means that all these single-output functions are jointly expanded, level by level, and the vector of functions is just treated as the first level of the expansion.

It will be assumed that a multi-output, incompletely specified Boolean function is represented in the .tt format (PLA format) with each cube denoted as 0, 1 or - in the output part. Thus, all cubes that have "0" in output column "i" (function f_i) will be denoted by OFF(f_i). All cubes that have "1" in output column "i" will be denoted by ON(f_i), and all cubes that have "-" in output column "i" by DC(f_i),

The following section reviews the Shannon and Davio expansions and illustrates their circuit realizations.

2 New Types of Kronecker Functional Decision Diagrams.

It is known [14, 19, 34] that there are three expansions over Galois Field (2) (we call them Davio expansions since they were first introduced by him).

(1) $f \;=\; s_i\, f_{s_i} \,\oplus\, \overline{s}_i\, f_{\overline{s}_i}$

(2) $f \;=\; f_{\overline{s}_i} \,\oplus\, s_i\,[\, f_{s_i} \,\oplus\, f_{\overline{s}_i}\,] = f_{\overline{s}_i} \,\oplus\, s_i\, g_i$

(3) $f \;=\; f_{s_i} \,\oplus\, \overline{s}_i\,[\, f_{s_i} \,\oplus\, f_{\overline{s}_i}\,] = f_{s_i} \,\oplus\, \overline{s}_i\, g_i$

where f_{s_i} is the cofactor [9] of function f with respect to s_i

$f_{s_i} \;=\; f \,\cap\, s_i \;\mid_{\,s_i=1} \;=\; f \;|_{s_i=1}$

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

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 [23]. The FDD is obtained by applying the reduction procedures used for BDDs [10] to the adaptive logic tree created using only expansion (2). If all three expansions are applied to all variables, the Kronecker Reed-Muller tree [14, 33, 34] is obtained. The extension to multiple-valued logic, the Multiple-Place Decision Diagram (MDD), has been introduced in [37].

Definition 2.1: The Ordered Kronecker Decision Diagram (OKFDD) is the decision tree obtained by recursively applying any of the three expansions to an incompletely specified Boolean function until the leaf nodes are the trivial functions one, zero or don't care.

Similarly to the Reduced OBDD (ROBDD), the Reduced OKFDD (ROKFDD) is obtained from the OKFDD by removing isomorphic subtrees. The Shared ROKFDD (KFDD) for multi-output functions is obtained analogously to the Shared ROBDD (SROBDD) [25]. Additionally, the KFDD as the SROBDD [25, 7] can have negated edges.

The concept of the OKFDD can be also extended to the IF-THEN-ELSE KFDDs, which are analogous to the If-Then-Else DAGs (ITE-DAGs) [22].

Definition 2.2: The IF-THEN-ELSE Kronecker Functional Decision Diagram (ITE-KFDD) is the decision tree obtained by recursively applying any of the expansions (4), (5), (6) to an incompletely specified Boolean function until the leaf nodes are the trivial functions one, zero or don't care. The expansions for the generation of the ITE-KFDD are given by

(4) $f \;=\; s_i\, c_1 \,\oplus\, \overline{s}_i\, c_2 \,\oplus\, c_3$

(5) $f \;=\; c_2 \oplus c_3 \oplus s_i \left[\, c_1 \oplus c_2 \,\right] = c_2 \oplus c_3 \oplus s_i\, g_i$

(6) $f \;=\; c_1 \oplus c_3 \oplus \overline{s}_i \left[\, c_1 \oplus c_2 \,\right] = c_1 \oplus c_3 \oplus \overline{s}_i\, g_i$

where c_1 is the part of the function that depends on the positive literal s_i of the splitting variable, c_2 is part that depends on the negated literal $\overline{s}_i$ of the splitting variable, and c_3 is independent of the splitting variable s_i. The advantage of the ITE-KFDD over the OKFDD is that the logic part of the function not depending on the splitting variable s_i does not have to be duplicated.

However, in the ROKFDDs the isomorphic subtrees can be merged to overcome this disadvantage.

Recently we developed programs for two more kinds of KFDD generalizations: MKFDDs, and PKFDDs.

Definition 2.3: The Mixed Kronecker Functional Decision Diagram (MKFDD) is the decision tree obtained by recursively applying expansions (1), (2) and (3) and having the variables ordered. However, for any variable (level of a tree) any combination of expansions can be applied to nodes of the level. The function can be incompletely specified, and the expansions are applied until all leaf nodes become trivial functions: one, zero or don't care.

Definition 2.4: The Permuted Kronecker Functional Decision Diagram (PKFDD) is the decision tree obtained by recursively applying expansions (1), (2) and (3) and having no order of variables. For any node of the tree any expansion can be applied. The function can be incompletely specified, and the expansions are applied until all leaf nodes become trivial functions: one, zero or don't care.

As we see, this is the most general kind of above-introduced KFDDs. It also includes Permuted RM trees from [48] as a special case.

All these ITE-KFDDs, MKFDDs, and PKFDDs can be of course also reduced, shared and with negated edges. From now on we will understand that KFDDs are reduced, shared and with negated edges.

It can be observed from Figure 1 that the circuit realizations of the three expansions correspond to realizable functions of a macrocell of the AT 6000 series, while the first expansion corresponds to the three multiplexers available in the macrocells of the $ACT^{T}M$ FPGAs from Actel [1]. Thus, the KFDDs are an ideal starting point for mapping to these FPGAs. Moreover, since the KFDD has fewer nodes than the FDD applied to the TLU architectures [42] it should be also better for the technology mapping to them. The final mapping algorithms are technology specific and are not discussed here.

3 Basic KFDD Synthesis Algorithm.

The crucial part of the KFDD synthesis is the determination of the splitting variable ordering and the expansion type selection. In our algorithm for the KFDD generation the splitting variable selection and the expansion selection are performed concurrently. However, in the selection process the same splitting variable is applied to all three expansions. Therefore, we first discuss the splitting variable selection while the expansion type selection methods are given in section 5.

There has been already a tremendous effort to determine a good variable ordering [17, 7, 21] for BDDs. In contrast to these methods we investigated synthesis methods developed for multiplexer circuits [43, 3, 6, 24, 46]. Inspired by the multiplexer synthesis algorithms developed by Almaini and Lloyd [3, 24] we adopted a breadth-first, top-down algorithm for the KFDD generation. To obtain a KFDD, we applied the restrictions from [3, 24, 39]:

1. the splitting function s_i can only be an input variable,
2. nodes in the same level of the KFDD have the same splitting variable s_i.

The basic algorithm to obtain a good variable ordering is based on finding the splitting variable and the selection of the expansions of the current level nodes such that the number of necessary next level nodes is minimal. The basic procedure, *compute_next_level(f,type)* from Fig. 2, illustrates the calculation of the splitting variable, where f is the multi-output function describing the function of the current level of the KFDD.

For the computation of a SROBDD, only the Shannon expansion has to be applied. This is realized in the routine *shannon_decompose(s, f)*. The three functions f_{s_i}, $f_{\overline{s}_i}$, and g are computed in the subroutine *davio_decompose(s, f)* for each of the output functions $f[k]$ of the current level. Next, the subroutine *expansion_select(ms)* determines an expansion type for each node of the current level, such that the number of the next-level nodes for the chosen splitting variable s_i is minimal. Three heuristics for the expansion type selection are given in section 5.

To determine the minimal number of the next-level nodes, it is necessary to find the redundant nodes in the next level. A node is redundant if it is trivial (0, 1, X_i) or if it can be merged with the existing node on this level (this node can belong to any output function f_r). Therefore, the next section investigates the conditions for the next level node to be redundant.

4 Redundancy Conditions for the Splitting Variable Selection.

In some cases it is not necessary to realize the input functions (f_s, $f_{\overline{s}}$, or g) of a node with additional nodes, depending on the selected expansion type and the selected splitting variable. In such a case no next level node is necessary. For brevity, f_i and f_j stand for any of the three functions f_s, $f_{\overline{s}}$, and g. We will also define: $f'_s = f \cap s, f'_{\overline{s}} = f \cap \overline{s}$, $g' = (f \ \cap \ s) \oplus (f \cap \overline{s})$. Analogously, f'_i and f'_j stand for any of the three functions f'_s, $f'_{\overline{s}}$, and g'. This section investigates how to specify the not specified part of the incompletely specified Boolean function in order to select the splitting variable to obtain the minimal KFDD.

There exist three basic conditions for which a next-level node is redundant.

"Condition 1:" an input function f_i is a trivial function

(7) $f_i \ = \ 0$

(8) $f_i \ = \ 1$

(9) $f_i \ = \ x_j$

(10) $f_i \ = \ \overline{x_j}$

"Condition 2:" an input function f_i is identical to another data-input function f_j to a node in the same level of the KFDD

(11) $f_i \ = \ f_j \qquad\qquad i \neq j$

"Condition 3:" an input function f_i is the complement of another data-input function f_j to a node in the same level of the KFDD

(12) $f_i \;=\; \overline{f_j} \qquad\qquad i \neq j$

The three conditions to determine if an input function is redundant, can be verified by the following cube calculus operations.

Verification of Condition 1: The case that an incompletely specified input function f_i can be specified such that $f_i \;=\; 0$ can be checked by

(13) $f'_i \;=\; f_{dc}$

where f_{dc} is a function that consists of only don't care terms. This is used when the function is stored in [ON, DC] form.

Equivalently, if the function is stored in [ON, OFF] form, one can use:

(13a) $ON(f'_i) \;=\; \emptyset$

(This condition means that f_i has only DC-cubes and OFF-cubes).

The case that an input function can be specified such that $f_i \;=\; 1$ can be verified by

(14) $ON(f_j) \;\cup\; DC(f_j) \;=\; 1$

To improve the program's efficiency, all verifications of conditions are based on the formula $f'_{s_i} = (f \cap s_i) \mid_{s_i=1} \;=\; f \mid_{s_i=1}$, so that the fast operation of intersection is calculated for all attempts (most of them fails). The slower operation of substitution is calculated much more rarely, only when the expansion type and its variable have been finally selected.

Thus for verification of (14) we use the formula

(14a) $f'_j \;=\; s_i$

where s_i is the corresponding splitting variable.

Equivalently, one can use the formula

(14b) $OFF(f'_i) \;=\; \emptyset$

An incompletely specified function f_i can be specified to be dependent on only one variable x_j if Equations (15) and (16) are satisfied:

(15) $f'_{x_l} \;=\; x_l$

and

(16) $f'_{\overline{x}_l} \;=\; f_{dc}$

where f_{dc} consists only of not-specified terms. Similarly for the complement $\overline{x}_l$.

Above, (15) is equivalent to

(15a) $OFF(f'_i) \;\cap\; x_l \;=\; \emptyset$

and (16) is equivalent to

(16a) $ON(f_i) \;\cap\; x_l \;=\; \emptyset$

Verification of Condition 2: For incompletely specified Boolean functions it has to be computed whether the not specified parts of the functions f'_i and f'_j can be specified in such a way that $f'_i \;=\; f'_j$. This can be verified by the complement of the intersection of the two incompletely specified functions

(17) $f'_{in} \;=\; f'_i \;\cap\; f'_j$

having no common minterms with the completely specified part of function f'_i

(18) $f'_i \cap \overline{f'_{in}} \;=\; f'_i \;\#\; f'_{in} \;=\; f_{dc_1}$

and no common minterms with the completely specified part of function f'_j

(19) $f'_j \;\cap\; \overline{f'_{in}} \;=\; f'_j \;\#\; f'_{in} \;=\; f_{dc_2}$

where # denotes the sharp operation [15]. and f_{dc_1} and f_{dc_2} are some functions of only unspecified terms.

If the Equations (18) and (19) are true, the not specified parts of f'_i and f'_j can be specified in such a way that $f'_i \;=\; f'_j$.

The above conditions are equivalent to

(18a) $ON(f'_i) \;\cap\; OFF(f'_j) \;=\; \emptyset$

and

(19a) $OFF(f'_i) \;\cap\; ON(f'_j) \;=\; \emptyset$

Verification of Condition 3: For the specification of the not specified part of the incompletely specified Boolean function such that two data-input functions are complemented, Equations (20) and (21) have to be true.

(20) $f'_i \;\cap\; f'_j \;=\; f_{dc}$

to verify that the specified parts of both functions have no common minterms, and

(21) $f'_{i\rightarrow 1} \;+\; f'_{j\rightarrow 1} \;=\; s_i$

where $\rightarrow 1$ means the specification of not specified minterms of the function to true minterms, to verify that the sum of both functions is only dependent on the chosen splitting variable. If both formulas are fulfilled, the not specified part of the data-input functions f'_i and f'_j can be chosen in such a way that $f'_i \;=\; \overline{f'_j}$.

These conditions are equivalent to:

(20a) $ON(f'_i) \;\cap\; ON(f'_j) \;=\; \emptyset$

and

(21a) $OFF(f'_i) \;\cap\; OFF(f'_j) \;=\; \emptyset$

respectively.

If functions were stored as sets ON and OFF (which is beneficial for strongly unspecified functions) applying the rules (14b), (15a), (16a), (18a), (19a), (20a) and (21a) would be more efficient. Efficient checking of those conditions is important since it takes most of the computer time while executing the algorithm.

5 Expansion Selection and Benchmark Results.

The results obtained by our implementation of the algorithm to obtain a reduced KFDD are given in Table 1. The column ASYL gives the number of nodes in the ROBDD with negated edges obtained by [7]. The columns $d1$, $d2$, and $d3$ give the number of nodes in the KFDD if only the Shannon, or one of the two Davio expansions is applied, respectively. Thus, column $d1$ gives the number of nodes in a SROBDD with negated edges and the columns $d2$ and $d3$ give two special cases of the Reed-Muller trees [14, 34]. The column "*heuristics*" gives the number of nodes in the KFDD when the following heuristics are applied to obtain a good selection out of the three expansion types. The applied heuristics to select the expansion type are the following:

HEURISTIC h1. If the splitting variable occurs mostly in a positive form in the output function, expansion $d2$ is selected. If the splitting variable occurs mostly in the negative form, expansion $d3$ is selected. For a tie $d1$ is chosen.

HEURISTIC h2. The expansion type of a node is selected based on the two functions out of three: f_{s_i},

$f_{\bar{s}_i}$, and g, having the least total number of product terms.

HEURISTIC h3. The expansion type of a node is selected based on the two functions out of three: f_{s_i}, $f_{\bar{s}_i}$, and g, having the highest fan-out. In the case of a tie the heuristic **h2** is applied.

Table 1 gives the results obtained by our synthesis algorithm for several MCNC benchmark functions. The sub-columns n for each of four columns d1, d2, d3 and "heuristics" give respective numbers of nodes in the KFDDs with negated edges, and the columns p give the longest paths in the obtained KFDDs. As a comparison, given are also: the numbers of nodes in the SROBDDs obtained by the ASYL system [7], and the the numbers of nodes in the FDDs obtained in [42].

It should be observed, that the number of nodes of the KFDD with negated edges is very close to the number of macrocells necessary for its implementation with AT 6000. This number can be smaller than the actual cell count, since mapping of a KFDD to AT 6000 requires adding inverter cells for negated edges (this is done in first mnapping). However, there are two reasons why we do not count them in the cost. Firstly, inverted edges are rare. Secondly, adding inverters is easily done during the placement/routing stage since in any case cells are being programmed as "wires" which connect a cell in a given level with its successor(s) in the next level of the tree. Each such "wire" cell can be easily programmed to an "inverter" cell which does not complicate placement/routing. Moreover, it is possible to program a two-output cell with these outputs: $output1 = input$, $output2 = \overline{input}$. The area cost is decreased with respect to the KFDD node cost since the method creates not only "fully utilized" AND/EXOR and multiplexer gates, but also their simplified subcircuits such as EXOR, AND, inverter and NAND. During mapping, several such circuits can be combined to larger single-output and two-output cells realizable by the AT 6000 cell, which further decreases the cells' count. On the other hand, the "wire" macrocells for routing purposes are necessary which in turn increases the cost. However, the level-by-level generation of the KFDD assures that there is only a limited connectivity besides the necessary level-to-level connections, so the number of such cells is limited comparing to realizations other than those based on KFDDs. In conclusion, the number of KFDD nodes seems to be a good (although rather optimistic) approximation of the layout area (counted as the total number of used AT 6000 cells: the cells programmed as logic, the cells programmed as connections, and the non-used cells surrounded by cells programmed to logic or connections.) This hypothesis was verified by hand on few 6-7 input functions, an larger examples done on AT 6000-tuned industrial tools. It must must be, however, further confirmed on many larger functions when our placement program will be ready.

In Table I it can be found, that in many cases the number of nodes in the SROBDD given in column $d1$ is smaller than by applying the other expansions or the heuristics to obtain a KFDD. However, for most cases for which heuristic **h3** led to the best result, no application of just one type of expansion led to an equal result. Moreover, it can be observed, that either **h3** or $d1$ resulted in the least number of nodes in the KFDD. Thus, the KFDD obtained by heuristic **h3** or the KFDD obtained by applying only expansion $d1$ that has the least number of nodes, can be selected to obtain a quasi-minimal KFDD.

It can be seen that out of the functions that were also minimized in [7] our program gave 8 times better solution, ASYL gave 6 times better solution and for two functions the results have the same costs. Since ASYL uses SROBDDs and our program uses KFDDs that are more general, the only reason that our program gives sometimes worse result is the non-optimal selection of variables and expansions for them. Improvement of the respective heuristic is one of our main goals in this research. For completely-specified single-output functions such improved results were already reported in [36]. At least for some functions KFDDs have less nodes than both BDDs and FDDs.

6 Logic Synthesis versus Placement and Routing in Cellular Devices.

Our placement strategy is to preserve, as much as possible, the layered structure of the KFDD. Since, in the worst case, the KFDD can grow exponentially, the scepticism expressed by some reviewers was to have $n \times 2^k - 1$ gates in the k-th level of the KFDD, which would mean substantial waste in placement. We found it then to be very assuring that the practical functions from the MCNC benchmark demonstrate a limited *width of the KFDD* (by the width we understand the maximum number of gates in the levels), and also similar numbers of gates on several adjacent levels. Table II presents the numbers of logic cells in each level, starting from the output. The rows d give the results obtained for applying the heuristic **h3** from section 5. The results for using only the Shannon expansion are given in row s. The reader may compare the numbers of inputs and outputs of each of those functions with Table I. Characteristical here is the function misex2, which has the same number of gates as in the output level, which is 18, in 9 consecutive levels and never exceeds this number. This property is shared by many other functions. Even when the "width" of the KFDD exceeds the number of outputs, it never becomes close to the upper bound. Functions table3 and table5 are perhaps the especially generated "tough cases". They have the ratio of maximum width to the number of outputs 13.35 and 7.21, respectively. (We calculate always this ratio for the better solution variant). Let us, however, observe that these ratios are still much better than the respective upper bounds. For instance, the largest width of the KFDD for function table3 is 187 in the 8-th level, and the maximum bound in this level is $14 * 2^8 - 1 = 1792$.

With exception to functions table3 and table5, the largest ratio was for function frg1 and was equal 5.33. For the other large functions the ratio was 3.55 (pdc), 1.8 (spla), and 1.58 (duke2). In many cases (such as 5xp1, cc) the ratio was below 1. For all functions from

Tables I and II, except for table3 and table5, good placements can be found directly from their respective KFDDs.

There is of course the question: how good are the presented methods compared to the standard industry approach: logic synthesis based on AND/OR factorization, followed by mapping to given cell library, followed by placement and routing. We cannot answer this question until our fitting/placement/routing program will be ready. However, for some functions the preliminary results are very encouraging. Since we do not have a Synopsys system, we asked our friends in industry to run four MCNC benchmarks: rd73, rd84, alu2 and sao2 with Synopsys tools and also with our program, and next compare areas produced by industrial placement/routing program optimized for AT 6000 devices. The improvements of our method over Synopsys were the following: rd73 - 35%, rd84 - 24%, alu2 - 52%, sao2 - 39%. Of course, this large improvements are possible only on functions that have many EXORs. On unate functions there was no improvement or the results were even worse. Concluding this evaluation, in a comprehensive CAD system the mapping method presented above should be *just one of several* mapping algorithms available.

7 Conclusion and Future Work.

This paper introduced the concepts of KFDDs (particularly ITE-KFDDs, MKFDDs, and PKFDD) and an efficient algorithm for finding a KFDD for an incompletely specified multi-output Boolean function. This algorithm is next used as a first stage of mapping to AT 6000 devices. Being an extension of the logic synthesis tools based on BDDs applied for the technology mapping to FPGAs [16, 29, 11, 7, 30], the algorithm can be used for other mapping applications, and (similarly to BDDs) to derive KFDDs of (incompletely specified) functions as a *general-purpose Boolean function representation* for further processing. In particular the advantageous application of KFDDs to nearly linear functions like the arithmetic circuits have been observed. For instance, if an incompletely specified function can be completed to a strictly linear function, the program will find this linear function as an EXOR of literals. The method gives also very good results on symmetric functions. The DAG circuits found from KFDDs map very well to AT 6000 series, so that fitting, placement and routing become essentially simplified.

The obtained results are promising and motivate to further investigate the KFDDs. Necessary research on KFDDs is to find better heuristics for the expansion type and variable order selection. Additionally, we would like to apply them to obtain realizations with the shortest path from input to output. Variants of the method can be also created that will include geometrical information in the KFDD growing process, in order to improve the future placement/routing processes.

Also, other introduced here new types of KFDDs will be further investigated.

We observed that evaluating the quality of the logic synthesis results for ATMEL devices can in general be misleading when done only on the base of cell/literal costs. (Although often a good heuristics for KFDDs). This is because very poorly routable designs can have smaller cell/literal costs than some other designs, which on other hand result in much smaller total layout areas. Therefore, our future research will combine logic synthesis with placement/routing to a single comprehensive process that we call "geometrical logic synthesis". Geometrical logic synthesis will become more and more useful with the proliferation of fine grain FPGAs and other cellular logic.

References

[1] "ACT Family Field Programmable Gate Array Data Book," March 1991. *ACTEL.*

[2] Algotronix, "The CHS 2*4, The world's first custom computer", *Algotronix Ltd,* Kings Buildings - TTC, Mayfield Road, Edinburgh EH9 3JL, Scotland, 1992.

[3] 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.

[4] S. Bandyopadhyay, A. Pal, and A. K. Choudhury, Characterization of Unate Cascade Realizability Using Parameters, *IEEE Trans. on Comput.,* Vol. 24, No. 2, pp. 218-219, February 1975.

[5] A. Bedarida, S. Ercolani, and G. DeMicheli, "A New Technology Mapping Algorithm for the Design and Evaluation of Fuse/Antifuse-based Field-Programmable Gate Arrays," *Proc. 1st ACM Workshop on FPGAs,* pp. 103-108, February 1992, Berkeley, CA.

[6] L. A. M. Bennett, "The Application of Map-Entered Variables to the Use of Multiplexers in the Synthesis of Logic Functions," *Int. J. Electronics,* Vol. 45, No. 4, 1978, pp. 373-379.

[7] 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.

[8] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, " MIS : Multi-Level Interactive Logic Optimization System , " *IEEE Trans. on CAD,* Vol. 6, No. 6, 1989, pp. 1062-1082.

[9] 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.

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

[11] K.-C. Chen, "Logic Minimization of Lookup-Table Based FPGAs," *Proc. 1st ACM Workshop on FPGAs,* pp. 71-76, February 1992, Berkeley, CA.

[12] E.M. Clarke, X. Zhao, M. Fujita, and Y. Matsunaga, "Fast Walsh Transform Computation with Binary Decision Diagrams", *Proceedings of this Workshop.*

[13] Concurrent Logic Inc, "CLi6000 Series Field Programmable Gate Array," *Preliminary Information,* December 1991.

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

[15] D. L. Dietmayer, "Logic Design of Digital Systems," *Allyn and Bacon*, 1978.

[16] R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Proc. 28th ACM/IEEE DAC*, pp. 227-233, June 1991, San Francisco, CA.

[17] S. J. Friedman, and K. J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams," *Proc. 24th ACM/IEEE DAC*, pp. 348-356, 1987.

[18] A. R. K. Gorai and A. Pal, "Automated Synthesis of Combinational Circuits by Cascade Networks of Multiplexers," *IEE Proc. Pt.E*, Vol. 137, No. 2, pp. 164-170, March 1990.

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

[20] C. Jay, "XOR PLDs Simplify Design of Counters and Other Devices," *EDN*, May, 1987.

[21] 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.

[22] K. Karplus, "ITEM: An If-Then-Else Minimizer for Logic Synthesis," *Proc. EURO-ASIC*, pp. 2-7, June 1992, Paris, France.

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

[24] A. M. Lloyd, "Design of Multiplexer Universal-Logic-Module Networks Using Spectral Techniques," *IEE Proc. Pt. E.,* Vol. 127, pp. 31-36, January 1980.

[25] S. Minato, N. Ishiura, and S. Yajima, "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation," *Proc. 27th ACM/IEEE DAC,* pp. 52-57, 1990.

[26] A. A. Mukhopadhyay, "Unate Cellular Logic," *IEEE Trans. on Comput.*, Vol. 18, No. 2, pp. 114-121, February 1969.

[27] Mukhopadhyay, A., "Recent Developments in Switching Theory," *Academic Press,* New York, London, 1971.

[28] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic Synthesis for Programmable Gate Arrays," *Proc. 27th ACM/IEEE DAC*, pp. 620-625, 1990.

[29] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *Proc. IEEE Int. Conf. on CAD*, pp. 564-567, November 1991, Santa Clara, CA.

[30] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "An Improved Synthesis Algorithm for Multiplexor-based PGA's, *Proc. 29th ACM/IEEE DAC*, pp. 380-386, June, 1992, Anaheim, CA.

[31] A. Pal, "An Algorithm for Optimal Logic Design Using Multiplexers," *IEEE Trans. on Comp.,* Vol. 35, No. 8, 1986, pp. 755-757.

[32] Perkowski, M. A., Dysko, P., B. J. and Falkowski, "Two Learning Methods for a Tree-Search Combinatorial Optimizer," *Proc. of IEEE Int. Conf. on Comput. and Comm.,* Scottsdale, Arizona, 1990, pp. 606-613.

[33] Perkowski, M. A. and P. D. Johnson, "Canonical Multi-Valued Input Reed-Muller Trees and Forms," *Proc. 3rd NASA Symposium on VLSI Design,* Moscow, Idaho, October 1991, pp. 11.3.1 - 11.3.13.

[34] 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, Sendai, Japan.

[35] M. A. Perkowski, "A Fundamental Theorem for Exor Circuits," *Proc. of this Workshop.*

[36] 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", *Proc. IWLS '93, International Workshop on Logic Synthesis.*

[37] T. Sasao, "Optimization of Multiple-Valued AND-EXOR Expressions using Multiple-Place Decision Diagrams," *Proc. IEEE 22nd ISMVL,* May 1992, Sendai, Japan.

[38] "An Algorithm for the Multi-Level Minimization of Reed-Muller Representation," *Proc. IEEE International Conference on Computer Design,* 1991, pp. 634-637.

[39] I. Schaefer, and M. Perkowski, "Synthesis of Multi-Level Multiplexer Circuits for Incompletely Specified Multi-Output Boolean Functions with Mapping Multiplexer Based FPGAs", accepted to *IEEE Trans. on CAD,* in October 1992.

[40] I. Schaefer, and M. A. Perkowski, "Synthesis of Multi-Level Multiplexer Circuits for Incompletely Specified Boolean Functions with Mapping to Multiplexer Based FPGAs", accepted to *IEEE Trans. on CAD, 1992.*

[41] I. Schaefer, *Ph.D. Thesis*, PSU, 1992.

[42] E. Schubert, U. Kebschull, and W. Rosenstiel, "FDD Based Technology Mapping for FPGA," *Proc. EURO-ASIC*, pp. 14-18, June 1992, Paris, France.

[43] Tabloski, T.F., and F. J. Mowle, "A Numerical Expansion Technique and Its Application to Minimal Multiplexer Logic Circuits," *IEEE Trans. on Comput.,* Vol. 25., No. 7, 1976, pp. 684-702.

[44] N. Song and M. Perkowski, "A Method of Logic Mapping for Fine Grain FPGAs", *Proc. IWLS '93.*

[45] N. Song and M. Perkowski, "A new approach to mapping for Fine Grain FPGAs," *submitted to Journal on VLSI Design.*

[46] A. J. Tosser, and D. Aoulad-Syad, "Cascade Networks of Logic Functions Built in Multiplexer Units," *IEE Proc. Pt. E,* Vol. 127, No. 2, pp. 64-68, March 1980.

[47] 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.

[48] 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.

[49] H. Wu, M. Perkowski, and N. Zhuang, "Synthesis of Multiplexer Directed-Acyclic-Graph network with application to FPGAs and BDDs," *Proc. IWLS '93.*

[50] D. Varma, and E. A. Trachtenberg, "Design Automation Tools for Efficient Implementation of Logic Functions by Decomposition," *IEEE Trans. on CAD,* Vol. 8, pp. 901-916, August 1989.

[51] Xilinx, "The Programmable Gate Array Data Book," 1989.