Synthesis of Multilevel Multiplexer Circuits for Incompletely Specified Multioutput Boolean Functions with Mapping to Multiplexer Based FPGA's

Ingo Schäfer and Marek A. Perkowski

***Abstract*—The introduction of the multiplexer-based Actel FPGA series ACT™ resulted in an increased interest in multiplexer circuits. This paper introduces a level-by-level top-down minimization algorithm for them. The concept of a local transform is applied for the generalization of the ratio parameter method for $M(1)$ multiplexer synthesis having one data select input and a spectral method for $M(2)$ multiplexer synthesis to determine redundant multiplexer inputs for $M(k)$ multiplexer circuits. The algorithm developed for multilevel synthesis of $M(k)$ multiplexer circuits for incompletely specified multioutput Boolean functions takes advantage of the combination of spectral and Boolean methods. The obtained multiplexer circuit can be directly realized with FPGA's like the Actel ACT series or the CLi 6000 series from Concurrent Logic. A simple heuristic is applied to map an $M(1)$ multiplexer circuit to the Actel ACT1 family.**

I. Introduction

Several multilevel synthesis tools for the minimization of logic functions to obtain a low literal count Boolean network have been developed [1]–[6]. A drawback of these tools is that they do not take the efficient synthesis for nearly linear functions into consideration. To overcome this drawback, the Portland logic optimizer (POLO) is currently under development. POLO is a system for efficient multilevel minimization, which takes into account near linearity of logic functions, especially for FPGA synthesis applications. For logic optimization purposes, a nearly linear function can be characterized by having a smaller number of product terms in its minimal exclusive sum of product (ESOP) form than it its minimal sum of product (SOP) form.

One approach to the synthesis of logic functions is to take advantage of the powerful multiplexer gate. It has been shown [7]–[9] that multiplexers are universal logic modules, where a multiplexer of k data-select inputs can realize any function of $k + 1$ input variables under the assumption that the complements of the input variables are also available. Thus, they can be used for the synthesis of multilevel logic networks. One can observe that an n-variable linear function (a linear function is an EXOR of literals) can be realized by a cascade multiplexer circuit of multiplexers with k data-select inputs, which are denoted by $M(k)$, where $\binom{n-1}{k}$ multiplexers are necessary.

Many synthesis algorithms for different kinds of multiplexer circuits have been developed [10]–[22]. One synthesis method with multiplexers is to find a single multiplexer, if possible of the minimum size, which realizes the given Boolean function [14]–[16], [20]. Algorithms to find a cascade multiplexer circuit of a Boolean function, if realizable, have been presented in [10], [18], and [21]. The algorithms [15]–[18] are based on graphical methods that allow only the synthesis for functions with up to six variables. A third realization, which will be further developed in this paper, is known as multiplexer tree circuit [11]–[13], [16], [17], [22]–[25]. All the previous algorithms operate on minterms, whereas the method presented here is based on disjoint cubes [26]. It was shown [26] that the disjoint cube representation of a Boolean function respective to the number of product terms is close to the minimal SOP form obtained by the logic optimizer Espresso [1]. Thus, the disjoint cube representation of a Boolean function is much more memory efficient than its minterm representation.

One of the motivations to investigate the synthesis for multiplexer tree circuits give new FPGA's like the ACT™ FPGA family from Actel [27], where the basic building block consists of multiplexers (see Fig. 1).

Each basic building block of the ACT™ family allows the implementation of an $M(2)$ multiplexer (Fig. 2(a)) and, in the case of the ACT1 family, implementation of three hierarchical $M(1)$ multiplexers (Fig. 2(b)), which is denoted by *actO* [28]. The ACT2 family allows only a restricted realization of three hierarchical $M(1)$ multiplexers, as can be observed from Fig. 1(b).

Another motivation for the interest in multiplexer circuits is that a function realized by multiplexer gates should have less gates than one constructed with conventional logic gates (NAND, NOR) [21]. We will compare the results obtained by technology mappers that support the explicit

Manuscript received July 30, 1993. This work was supported by NSF grant MIP-9110722. This paper was recommended by Associate Editor L. Trevillyan.

I. Schäfer is with the Lattice Semiconductor, Santa Clara, CA 95054.
M. A. Perkowski is with the Department of Electrical and Computer Engineering, Portland State University, Portland, OR 97207.

IEEE Log Number 9212353

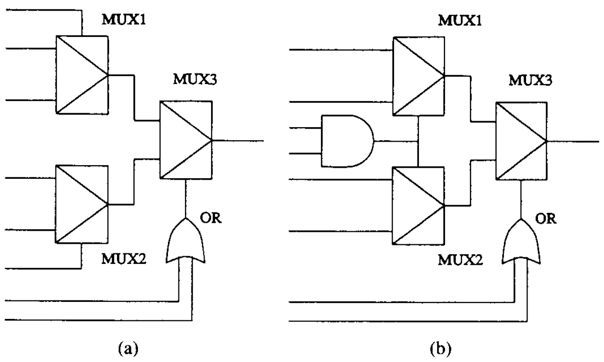


Fig. 1. Basic building block of the ACT™ family: (a) ACT1 family; (b) ACT2 family.

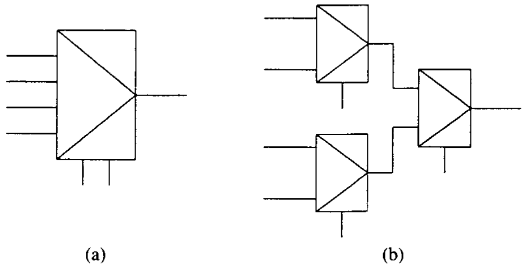


Fig. 2. Realizable multiplexers with ACT™ family: (a) $M(2)$ multiplexer; (b) three $M(1)$ multiplexers, act0.

library of realizable functions of the Actel FPGA's [29], [30] and the results of special mapping algorithms that take advantage of the structure of the Actel macrocells [29], [31], [32] with the results of the mapped multiplexer circuit obtained by the synthesis algorithm presented.

In this paper, we further develop the methods to find redundant multiplexer modules in a tree circuit [10], [12], [20]–[24] for the level-by-level top-down (starting from the output) minimization of multiplexer tree circuits [12], [22], [23]. The two methods for single-output completely specified Boolean functions, where one is based on ratio parameters for $M(1)$ multiplexer synthesis [10], [20], [21] and the second is based on the Rademacher–Walsh spectrum [23], [25] for $M(2)$ multiplexer synthesis are here uniformly generalized to the $M(k)$ multiplexers synthesis for two-level multioutput incompletely specified Boolean functions. Such functions occur, for example, in the synthesis of sequential circuits and from high-level language descriptions [1] or as internal nodes in multilevel network with satisfiability and observability don't cares [1].

Because computing the complete Rademacher–Walsh spectrum [23] to determine the redundant next level $M(2)$ multiplexer gates is complex, we introduce the concept of a *local* transform [25], [33] for $M(k)$ multiplexers. This notion is similar to the ratio parameter developed for the $M(1)$ synthesis in [10]. Therefore, contrary to existing technology mapping tools, the synthesis method developed here is based on Boolean decomposition methods.

II. General Multiplexer Synthesis

First, we review the concept of polarity to be able to give the general formula of a multiplexer $M(k)$.

Definition 1: The *polarity* of a product term $\dot{x}_1\dot{x}_2 \cdots \dot{x}_i \cdots \dot{x}_n$, where $\dot{x}_i \in \{x_i, \bar{x}_i\}$ is defined by the binary string of values being 0 for $\bar{x}_i$ and 1 for x_i, respectively.

The general case of a multiplexer $M(k)$, where k is the number of data-select inputs, can be defined as follows.

Definition 2: The output function $f(x_0, x_1, \cdots x_{n-1}) = f(X)$, where X is the vector of variables $\langle x_0, x_1, \cdots, x_{n-1} \rangle$, of a multiplexer $M(k)$ with k data-select inputs $d_j(X)$ is given by

$$f(X) = \bigcup_{i=0}^{2^k-1} (\dot{d}_1(X) \cap \cdots \cap \dot{d}_1(X) \cap \cdots \cap \dot{d}_1(X) \cap f(X)) \tag{1}$$

where $\dot{d}_j(X) \in \{d_j(X), \overline{d_j(X)}\}$, being any Boolean function, with the polarity determined by the binary representation of i.

For multiplexer synthesis algorithms, the data-select functions $d_j\ (X)$ are commonly restricted to input variables [12], [23]. Thus, the function $f(X)$ can be described by the sum of cofactors [1] with respect to the data-select variables.

$$f(X) = \bigcup_{i=0}^{2^k-1} (\dot{x}_1 \cdots \dot{x}_k (f(X))_{\dot{x}_1 \cdots \dot{x}_k}) \tag{2}$$

where the polarity of the data-select variables $\dot{x}_1 \cdots \dot{x}_k$ is determined by the binary representation of i. This is the well-known Shannon expansion. It follows that the data-input function f_i is given by

$$f_i = (f(X))_{\dot{x}_1 \cdots \dot{x}_k}. \tag{3}$$

Example 1: An n variable Boolean function $f(X)$ with the data-select inputs x_1, x_2 is decomposed to

$$f(X) = \bar{x}_1\bar{x}_2 f_0 + \bar{x}_1 x_2 f_1 + x_1 \bar{x}_2 f_2 + x_1 x_2 f_3 \tag{4}$$

which is illustrated in Fig. 3.

The functions f_i are the data input functions, and the variables x_1, x_2 are the data select variables of the multiplexer. Let us observe that the functions f_i do not depend on x_1 and x_2.

The general problem in multiplexer synthesis is to find a circuit realization with the minimum number of multiplexers for a given multioutput incompletely specified Boolean function. According to (1), such a circuit can have multiplexers with various numbers of data-select inputs where each data-select input can be any function $d_j\ (X)$ [7], [24]. Because finding the optimal data-select functions that may be one of many possible Boolean functions is very complex, the multiplexer synthesis algorithms find (quasi)optimal realizations according to (2), where the data-select variables are input variables [11]–[14], [16]–[21].

Such a minimal multiplexer circuit can have different permutations of data-select variables in any branch of the

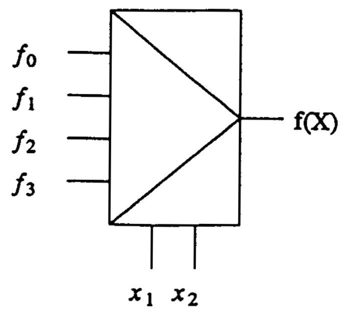


Fig. 3. Standard multiplexer $M(2)$ according to (4).

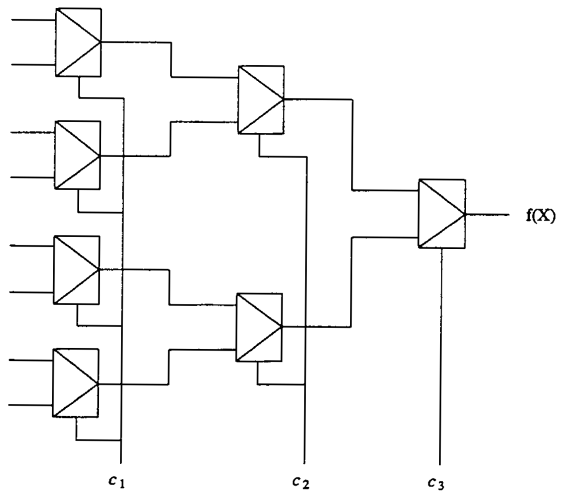


Fig. 4. Restricted multiplexer tree circuit.

circuit [13], [24]. To decrease the complexity of the minimization problem further, most multiplexer synthesis algorithms assume the same data-select variables in any level of the tree circuit [11]–[13], [23], [24], as shown in Fig. 4. This restriction is also advantageous for a possible mapping to the ACT2 family, see (Fig. 1(b)), where the data-select input variables of *MUX*1 and *MUX*2 are given by the AND of two variables.

As was stated in [24], the minimal upper bound for the levels in such a restricted multiplexer tree circuit is given by

$$L = \frac{n-1}{k} \tag{5}$$

and the minimal upper bound for the number of multiplexer modules $M(k)$ by

$$M = \sum_{i=0}^{L-1} 2^{ik}. \tag{6}$$

Still, an exhaustive search has to be performed to find the optimal permutation of the data-select variables [11], [17], [24]. Level-by-level minimization algorithms [12], [22], [23] decrease the necessary computation and storage requirements for the implicitly exhaustive algorithms that find the optimum tree circuit. It was conjectured [12] that level-by-level minimization is still optimum or near optimum. It should be observed that the special case of $M(1)$ multiplexer synthesis is equivalent of finding the minimal shared ordered binary decision diagram (SOBDD) representation of the Boolean function with attributed edges [1], [34], [35]. Thus, a multiplexer synthesis algorithm for $M(1)$ multiplexers can be applied to find a good variable ordering of a SOBDD.

The next section investigates the conditions for which the next-level multiplexer modules are redundant.

II. Redundancy of Multiplexer Modules

Definition 4: A spectrum of an n-variable Boolean function is called *local* spectrum L if the value of any spectral coefficient l_i of the spectrum L does not depend on all 2^n minterms, that is, the transform matrix describing the transform has at least one entry that is 0.

The Rademacher–Walsh spectrum is not a local transform because any entry of its transform matrix W [25] is either 1 or -1.

Notation: The number of true minterms covered by the data-input function f_i will be denoted by a_i and the number of don't care minterms by c_i.

The values a_i and c_i of the function f_i can be easily calculated from its disjoint representation [26]. It should be noted that the values a_i and c_i are independent of the function representation.

Definition 5: A spectral coefficient l_i of the *local* spectrum L for an incompletely specified Boolean function is given by the pair of values $l_i \in \{(a_i, c_i)\}$, whereas for a completely specified Boolean function $l_i \in \{(a_i)\}$. The *local* spectrum L for a $M(k)$ multiplexer consists of a spectral coefficient l_i for each of its data-input functions f_i.

Thus, analogously to the Lloyd method and the ratio parameter method, each coefficient of the local spectrum gives the number of true minterms covered by a data-input function f_i. For $M(2)$ multiplexer synthesis [23] the Rademacher–Walsh spectrum was adopted to find the correlation of the Boolean function to the support functions p_i. A further summation [23], [25] of a subset of spectral coefficients of the Rademacher–Walsh spectrum is necessary to obtain the final spectrum describing the correlation to the support functions p_i. This summation is realized by a 4×4 Rademacher–Walsh transform on subsets of the initial Rademacher–Walsh spectrum. In the general case of an $M(k)$ multiplexer, the summation can be realized by a $2^k \times 2^k$ Rademacher–Walsh transform on the respective subset of the initial Rademacher–Walsh spectrum for each possible set of k data-select variables.

A *local* transform for multiplexers can perform, in one step, both the initial calculation of the complete Rademacher–Walsh spectrum of the function and the following summation of a subset of coefficients. Thus, one avoids the complexity of computing the whole Rademacher–Walsh spectrum for a given function. An $M(k)$ multiplexer for an n-variable Boolean function can have $\binom{n}{k}$ dif-

TABLE I
COMPARISON OF THE NUMBER OF COEFFICIENTS

variables	Rademacher-Walsh coefficients	spectral coefficients for $\binom{n}{2}$ pairs of data-select variables
n	2^n	$\binom{n}{2}\times 4$
5	32	10×4
10	1024	45×4
20	1048576	190×4
30	10^9	435×4

ferent possible combinations of k data select variables. Table I compares the number of coefficients that have to be calculated for an $M(2)$ multiplexer synthesis by using the Rademacher–Walsh transform and the *local* transform.

The first column of Table I gives the number of input variables of a Boolean function. The second column lists the number of spectral coefficients of the initial Rademacher–Walsh spectrum. Finally, the last column gives the number of spectral coefficients for all possible pairs of data-select variables, where a spectrum for a data-select pair consists of four coefficients. The formulas for the calculation of the number of coefficients are given in the second row.

As one can observe from Table I, the computation of the Rademacher–Walsh spectrum for functions with more than 20 variables is not feasible with general fast transforms [36]. It can be calculated directly from a disjoint representation [26], [37], [38] that is faster and much more memory efficient, but still, the complexity of calculating all 2^n coefficients can be prohibitive. However, for the decomposition to multiplexer modules given by (2), we are only interested in the correlation between the support functions p_i and the data-input functions f_i to the multiplexer. Thus, the direct calculation of only the necessary spectral coefficients for the combinations of k data-select variables by a *local* transform is much more efficient.

The basic principle of the level-by-level minimization algorithm from [12], [23] is to find the minimal number of next level modules for a given level. This approach will be also adopted here. A similar principle is used for the realization of Boolean functions as cascade multiplexer circuits or single multiplexer circuits, where only one next-level module is allowed, or no module is allowed at all [10], [14]–[16], [18], [20], [21].

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

Condition 1: A data-input function is trivial: $f_1 = 0$, $f_i = 1, f_i = x_j, f_i = \bar{x}_j$.

Condition 2: A data-input function is identical to another data-input function to a multiplexer in the same level of the tree circuit: $f_i = f_j$ for $i \neq j$.

Condition 3: A data-input function is the complement of another data-input function to a multiplexer in the same level of the tree circuit: $f_i = \bar{f}_j$ for $i \neq j$.

Most multiplexer synthesis algorithms only take the first condition into consideration to decrease the number of next level modules [11]–[13], [16], [20], [24]. The advantage of the presented method is that it verifies all three conditions. Thus, a linear component determined by an Exclusive OR gate can be detected.

Example 2: An $M(1)$ multiplexer $f(X) = xf_x + \bar{x}f_{\bar{x}}$ (see Fig. 5(a)) realizes an Exclusive OR gate if $f_x = \bar{f}_{\bar{x}}$

$$f(X) = xf_x + \bar{x}\bar{f}_x = x \oplus f_{\bar{x}}.$$

A linear component of a function can be detected by finding multiplexers having as data-input functions f_x and its complement $\bar{f}_x$. Even if no inverters are available (like in the Actel FPGA's), only one multiplexer is necessary to realize the complemented function instead of a complete subtree. The complemented function can be realized by a control function circuit [24] as shown in Fig. 5(b). Then, the multiplexer circuit given in Fig. 5(c) is obtained for $f(X) = x \oplus f_{\bar{x}}$.

In the case that in a particular level of the multiplexer circuit different combinations of k data-select variables require the same number of next level multiplexer modules, a selection should take into account the possible further minimization in the next lower levels of the multiplexer circuit. Spectral methods are ideally suited for this case because they give insights into the global structure of the underlying Boolean function [10], [23], [25].

Definition 3: The functions p_i represented by the k data-select variables for an $M(k)$ multiplexer

$$p_i = \dot{x}_1 \cdots \dot{x}_j \cdots \dot{x}_k \tag{7}$$

are called *support functions*, where $i = \{0, \cdots, 2^k - 1\}$ is the binary representation of the polarity of the k data-select variables. It follows from (3)

$$f_i = (f(X))_{p_i}. \tag{8}$$

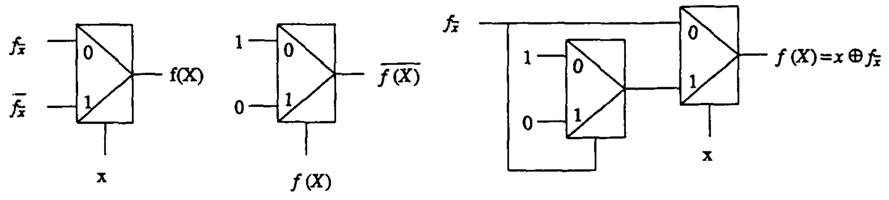
The number of all true and false minterms covered by a support function p_i is

$$pt = 2^{n-k}. \tag{9}$$

Thus, the multiplexer synthesis can be based on the information obtained from the correlation between the Boolean function and the support functions p_i. The concept of ratio parameters [10], [20], [21] has been developed to determine if data-input functions to an $M(1)$ multiplexer are trivial. The ratio parameter method is essentially a spectral method like the method for $M(2)$ multiplexer synthesis [23]–[25]. Both method are based on determining the number of true minterms covered by the data-input function f_i.

The spectral method by Lloyd and the ratio parameter method can be uniformly generalized for the synthesis with $M(k)$ multiplexers by the introduction of a *local* transform [33] for multiplexer gates.

A criterion for the selection of a particular combination of k data-select variables in the case that more than one combination leads to the same minimal number of next level modules has been introduced in [23]. The criterion is based on the values of the spectral coefficients s_i for the Rademacher–Walsh spectrum $S_{x_1 \cdots x_k}$. It was stated in [23]



(a) (b) (c)

Fig. 5. Linear function realization: (a) Complemented input; (b) inverter; (c) complete circuit.

that a high value of the sum of absolute values of the spectral coefficients s_i

$$\text{sum} = \sum_{i=0}^{2^n-1} |s_i| = |s_{x_1 \cdots x_k}| \tag{10}$$

gives a higher possibility for further minimization of the number of multiplexer gates in the following levels. This is based on the assumption that a higher density of false or true minterms in the data-input functions allows further minimization in the next levels.

The following property gives the relation of the spectral coefficients l_i obtained by the *local* transform for completely specified Boolean functions given by Definition 5 to the spectral coefficients s_i of the Rademacher–Walsh spectrum used in [23].

Property: The spectral coefficients s_i of the spectrum $S_{x_1 \cdots x_k}$ for k data-select variables $x_1 \cdots x_k$ are computed as follows:

$$s_i = (pt - 2 \times l_i) \times 4 = (pt - 2 \times a_i) \times 4 \tag{11}$$

where pt is the number of all true and false minterms covered by the support function p_i.

For incompletely specified Boolean functions, the value of $l_i \in \{a_i, a_i + c_i\}$ is chosen whichever leads to the higher absolute value of s_i.

Thus, we can formulate a fourth condition for multilevel realizations.

Condition 4: For different combinations of k data-select variables with the same minimal number of next-level multiplexer modules, one chooses the k data-select variables with the highest value of *sum* calculated according to (10) and (11). If several combinations have the same highest value of *sum*, one of them is selected randomly.

With the above Definitions and Properties, we are able to formulate the conditions for the redundancy of next-level modules by spectral and Boolean means.

Verification of Condition 1: A trivial incompletely specified data-input function f_i has the following properties in the spectral domain, for $f_i = 0$:

$$a_i = 0 \tag{12}$$

and for $f_i = 1$:

$$a_i + c_i = pt \tag{13}$$

In the Boolean domain for the computation with a cube representation, (12) can be verified by

$$f_i = f \cap p_i = f_{dc} \tag{14}$$

where f_{dc} consists only of not specified terms. Equation (13) can be similarly verified by

$$f_i = f \to 1 \cap p_i = p_i \tag{15}$$

where the notation $f \to 1$ denotes the specification of the not-specified terms of the incompletely specified Boolean function f to true literals. In the cube representation, just the respective output literals have to be changed from not specified to true terms to obtain this transformation.

A necessary condition for a completely specified function that is dependent on only one variable x_l is that the coefficient

$$s_l = 2^{n-k} \tag{16}$$

which is equivalent to

$$a_l = 2^{n-k-l}. \tag{17}$$

For an incompletely specified Boolean function, it is necessary that the not-specified terms can be specified in such a way that (16) is fulfilled. This can be verified in Boolean domain by

$$f \to 1 \cap x_l = x_l \tag{18}$$

and

$$f \cap \bar{x}_l = f_{dc} \tag{19}$$

where f_{dc} consists only of not-specified terms.

Verification of Condition 2: The assignment of the not-specified minterms to true or false ones to obtain identical data-input functions ($f_i = f_j$), if possible, can be easily tested in spectral domain for completely specified Boolean functions as:

$$a_i = a_j. \tag{20}$$

However, if (20) is true, the identity of the two functions still has to be verified. For incompletely specified Boolean functions, it has to be computed if 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

$$f_{\text{in}} = f_i \cap f_j \tag{21}$$

having no common minterms with the completely specified part of function f_i

$$f_i \cap \bar{f}_{\text{in}} = f_i^{\#} f_{\text{in}} = f_{dc1} \tag{22}$$

and no common minterms with the completely specified part of function f_j

$$f_j \cap \overline{f}_{\text{in}} = f_j^{\#} f_{\text{in}} = f_{dc2} \tag{23}$$

where # denotes the sharp operation [39], and f_{dc1} and f_{dc2} consist only of not specified terms. If (22) and (23) are true, the not-specified part of f_i and f_j can be specified in such a way that $f_i = f_j$.

Verification of Condition 3: A necessary condition for a data-input function f_i to be the complement of a completely specified data-input function f_j is

$$a_i + a_j = pt. \tag{24}$$

To find a possible specification of the not-specified part of the incompletely specified Boolean function such that two data-input functions are complemented, (25) and (26) have to be true

$$f_i \cap f_j = f_{dc} \tag{25}$$

to verify that the specified parts of both functions have no common minterms. The equivalent of (24) in the Boolean domain is given by

$$f_{i \to 1} + f_{j \to 1} = p_i. \tag{26}$$

If both formulas can be fulfilled, the not-specified part of the data-input functions f_i and f_j can be chosen in such a way that $f_i = \overline{f}_j$.

The above conditions have to be computed for all possible combinations of data-select variables. In the case that there is more than one set of data-select variables that have the minimum number of next-level multiplexer modules, the one according to Condition 4 is chosen.

The verification of the four conditions allows the determination of the data-select variables for a single level of multiplexer modules. The algorithm is based on the bottom-up level-by-level minimization of a multioutput incompletely specified Boolean function. The primary output level of the function (which can consist of more than one multiplexer) is treated like any other level of the multiplexer tree circuit.

The pseudocode for the complete $M(k)$ multiplexer synthesis algorithm *mux_synthesis (k)* is shown for the illustration of the different steps involved.

Algorithm: mux_synthesis(k),

```
// k                : number of data-select variables
// output_functions : number of output functions for given Boolean function
// input_variables  : number of input variables for given Boolean function

mux_synthesis(k)
{
    min_level_modules = output_functions * $2^k$;
    level = 0;    // determines current level in circuit
    while ( min_level_modules != 0 ){
        // computation of data-select variables that lead to the
        // minimal number of next level modules for the current level
        for ( i = 0 ; i < $\binom{input\_variables - level * k}{k}$ ; i++ ) {
            // all permutations of data-select variables
            data_select_variables[] = generate_data_select(k,i);
            // computation of the number of next level modules for given data_select_variables[]
            // through verification of Conditions 1-3
            level_modules = compute_modules(data_select_variables[]);
            sum = $\sum_{n=0}^{level\_modules}$ $sum_n$; // sum of all multiplexers
            if ( min_level_modules > level_modules ){
                // current data_select_variables lead to less next level modules
                max_sum = sum;
                min_level_modules = level_modules;
                best_select[] = data_select_variables[];
            }else if ( min_level_modules == level_modules ){
                // verification of Condition 4
                if ( sum > max_sum ){
                    max_sum = sum;
                    best_select[] = data_select_variables[]; } }
        // computation of the next level for found data-select variables
        decompose(best_select[]);
        level++; } }
```

The function *compute_modules*(*data_select_variables*[ ]) determines, through the verification of Conditions 1-3, the number of necessary next-level modules *best select*[ ] for the given set of data-select variables. Condition 4 is applied for tie cases by comparing their *sum* obtained by the summation over each sum_n—computed according to (10). After the set of data-select variables that leads to the minimum number of next-level modules has been found, the function *decompose(best_select*[ ]*)* computes the output functions for the next level for which the process is repeated. The algorithm stops when no further next-level modules are necessary, that is, a multiplexer circuit realization of the circuit has been found.

The developed methods are illustrated in the next section with a complete example.

IV. Synthesis Example

The synthesis steps from the previous section will be illustrated on the example from [23]:

$$f(X) = \overline{x}_1 x_4 \overline{x}_5 + x_1 x_2 x_4 + \overline{x}_1 x_2 x_3 \overline{x}_4 + \overline{x}_3 \overline{x}_4 x_5 + x_1 \overline{x}_4 x_5.$$

First, the disjoint cube representation of $f(X)$ has to be computed to be able to compute the *local* spectra

$$f(X) = \overline{x}_1 x_4 \overline{x}_5 + x_1 x_2 x_4 + \overline{x}_1 x_2 x_3 \overline{x}_4 + \overline{x}_3 \overline{x}_4 x_5 + x_1 x_3 \overline{x}_4 x_5.$$

The disjoint cubes obtained are illustrated by the circled areas shown in Fig. 6.

For the initial verification of the redundancy of next level modules, the *local* spectrum for each combination of data-select variables and the *sum* parameter are calculated. The calculation of the initial spectra $L_{x_i x_j}$ will be illustrated for the spectrum of the data-select pair x_1, x_2. The value of the spectral coefficients of the spectrum $S_{x_i x_j}$ [23] can be obtained by (11). The support functions p_i for the *local* transform for the chosen data-select pair are $p_0 = \overline{x}_1 \overline{x}_2$, $p_1 = \overline{x}_1 x_2$, $p_2 = x_1 \overline{x}_2$, $p_3 = x_1 x_2$, where each trivial function covers $pt = 2^{5-2} = 8$ minterms: see (9).

To obtain the *local* spectrum for the chosen data-select pair, the cofactor of the function $f(X)$ has to be calculated with respect to each support function, e.g.

$$f_0 = (f(X))_{p_0} = (f(X))_{\overline{x}_1 \overline{x}_2}.$$

Because the function $f(X)$ is completely specified, the spectral coefficients of the *local* spectrum are given by $l_i \in \{a_i\}$. They can be directly computed from the disjoint

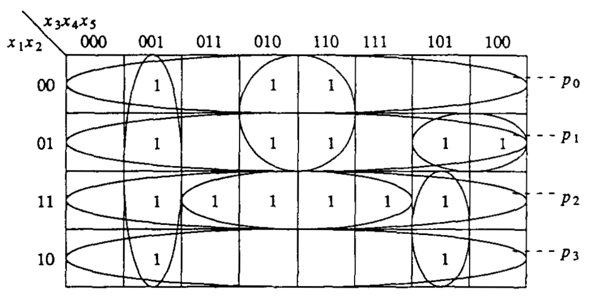


Fig. 6. Disjoint cube representation of synthesis example.

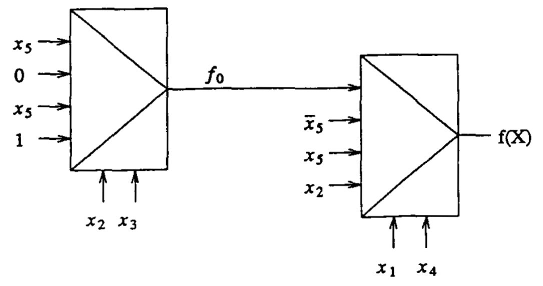


Fig. 7. Final multiplexer realization.

cube representation of the data-input functions f: $a_0 = 3$, $a_1 = 5$, $a_2 = 2$, and $a_3 = 6$. No coefficient $l_i \in \{a_i\}$ fulfills (12), (13), or (17), which verify the possibility of trivial data-input functions. Similarly, the criterion for the identity of two data-input functions (see (20)) is not satisfied. However, the necessary condition for complemented data-input functions is true for $a_2 + a_3 = 8$. Therefore, this case has to be verified further. Applying (25), we obtain $f_2 \cap f_3 = \bar{x}_4 x_5 \neq \phi$. Because the intersection is not empty, f_2 cannot be the complement of f_3. Thus, no data input of the multiplexer is redundant. Therefore, the number of next level modules *Mod* for the data-select variables x_1 and x_2 is *Mod* = 4. To be able to later check the Condition 4, the *sum* = 48 is computed by applying (10) and (11).

The spectra for the other data-select pairs are calculated the same way. The results are listed in Table II. Because all possible cases of redundant next-level modules can be verified by conditions (1)–(3), the exact number of next level modules *Mod* can be directly calculated, whereas the method by Lloyd only determines the upper and lower bound *maxMod* and *minMod*.

In Table II, a^* indicates, that the respective data-input function is either identical to another one or a trivial one. Because $L_{x_1x_4}$ has the lowest number of next-level modules (*mod* = *1*), variables x_1 and x_4 are selected for the output multiplexer, and all other data-select pairs can be discarded. Thus, the best pair of data-select variables has been found in the first step, whereas, for the pure spectral method [23], three further verification steps would be necessary.

The nontrivial data-input functions to the multiplexer module have to be decomposed further for the calculation of the next level of the hierarchical multiplexer circuit. The four data-input functions have already been obtained by the substitution of the support functions p_i for the data-select pair x_1, x_4 with the initial function for the calculation of the spectral coefficients: $f_0 = \bar{x}_3 x_5 + x_2 x_3$, $f_1 = \bar{x}_5$, $f_2 = x_5$, and $f_3 = x_2$. The only nontrivial function is f_0. Therefore, f_0 has to be decomposed further. Because the function f_0 depends on only three variables, it can be realized by one $M(2)$ multiplexer, and the choice of the data-select variables does not matter [21].

To illustrate the general synthesis procedure, all three spectra, $L_{x_ix_j}$ of the remaining variables x_2, x_3 and x_5 are calculated (Table III).

TABLE II
First Level Spectra of f

row	spectrum	l_0	l_1	l_2	l_3	sum	Mod
1	$L_{x_1x_2}$	3	5	2	6	48	4
2	$L_{x_1x_3}$	4	4	4	4	0	4
3	$L_{x_1x_4}$	4	4*	4*	4*	0	1
4	$L_{x_1x_5}$	5	3	2	6	48	4
5	$L_{x_2x_3}$	3	2	5	6	48	4
6	$L_{x_2x_4}$	3	2	5	6	48	4
7	$L_{x_2x_5}$	2	3	5	6	48	4
8	$L_{x_3x_4}$	4*	4	4	4	0	3
9	$L_{x_3x_5}$	3	5	4	4	16	4
10	$L_{x_4x_5}$	1	7	6	2	80	4

TABLE III
Second-Level Spectra for f_0

row	spectrum	l_1	l_2	l_3	l_4	sum	Mod
1	$L_{x_2x_3}$	4*	5*	4*	3*	16	0
2	$L_{x_2x_5}$	5*	4*	4*	3*	16	0
3	$L_{x_3x_5}$	5*	3*	4*	4*	16	0

The expected result that no further multiplexer is necessary for any data-select pair is verified by the result given in Table III. The data-select pair x_2, x_3 is chosen for the realization of the second-level multiplexer. Thus, the final hierarchical multiplexer circuit has the form shown in Fig. 7.

V. Mapping of a Multiplexer Tree Circuit to the ACT1 Macrocell

To allow an evaluation of the presented method, we map the result obtained by the multiplexer synthesis algorithm to multiplexer-based FPGA's for which many synthesis tools exist [28], [31], [40], [41]. Although the hierarchical multiplexer circuits obtained for $M(1)$ multiplexer can be mapped directly to FPGA's like the Actel ACT series [27], the CLi6000 of Concurrent Logic [42], or the CAL 1024 of Algotronix [43], a special mapping algorithm is necessary for the ACT™ family of Actel. The macrocells provided by the ACT™ FPGA family from Actel (see Fig. 1) have a restricted connectivity of $M(1)$ multiplexers. An $M(2)$ multiplexer can be realized with one macrocell from both the ACT1 and ACT2 families.

TABLE IV
BENCHMARK COMPARISONS

benchmark function	k = 2 modules	depth	k = 1 modules	depth	mux-map cells (ti.)	misII cells (ti.)	mis-pga cells (ti.)	Amap cells (ti.)	Prosperine act0	act1	ASYL cells
alu2	96	5	187	8	115 (25.6)	193 (24)	175 (150)	188 (8)			100
b12	53	7	65	6	34 (3.8)						
bw	61	2	106	4	68 (5.1)	81 (7.1)	54 (48)	83 (3.6)	87	63	59
cc	29	4	45	5	29 (3.3)						
clip	48	4	91	8	46 (21.5)	57 (4.9)	43 (88)	60 (2.5)	86	68	44
con1	8	3	24	6	8 (0.2)						
cu	32	5	57	9	34 (2.1)						
duke2	102	6	395	14	221 (59.0)	176 (18)	158 (131)	175 (7)			173
ex5	150	4	253	7	139 (87.0)						
f51m	27	4	55	7	27 (4.2)	52 (5.2)	39 (62)	56 (2.2)	83	59	35
frg1	10	8	110	16	57 (89.2)						
inc	35	3	94	6	45 (2.3)						
i1	23	4	43	9	25 (8.0)						
misex1	17	4	33	5	18 (0.5)	22 (1.9)	16 (20)	25 (1.1)	30	24	21
misex2	90	12	171	22	59 (0.6)	46 (4.3)	38 (10)	47 (1.5)	52	42	43
misex3c	300	7	457	12	253 (345.0)						
pdc	580	8	773	11	431 (219.9)						
rd53	7	2	15	4	11 (0.2)						10
rd73	14	3	29	6	19 (5.4)	32 (12.1)	25 (32)	32 (1.6)			20
rd84	21	4	40	7	25 (24.2)	62 (6.5)	36 (95)	62 (2.6)	87	65	30
sao2	41	5	83	9	43 (3.1)	52 (8.0)	51 (24.4)	56 (2.0)			45
spla	361	8	585	15	318 (64.7)						
squar5	19	2	34	4	19 (0.3)						
table3	793	7	1336	13	772 (117.5)						
vg2	48	7	95	19	50 (121.9)	47 (7)	30 (24)	44 (1)			40
xor5	2	2	4	4	4 (0.0)						
5xp1	28	3	54	6	24 (2.7)	51 (4.4)	35 (48)	42 (1.7)	65	50	37
9sym	11	4	23	8	11 (3.6)	99 (14.1)	17 (4109)	106 (4.1)			17

Therefore, a multiplexer circuit based on $M(2)$ multiplexers can be directly mapped to the Actel FPGA's. However, because of the lack of inverters, for each complemented data-input function, an additional macrocell is necessary.

For our implementation, we chose the ACT1 macrocell. The mapping of an $M(1)$ multiplexer circuit to the ACT1 macrocell is restricted by the internal connectivity of the macrocells. The output functions of the two multiplexers at the input of the macrocell *MUX*1 and *MUX*2 are not available as outputs of the macrocell. Therefore, multiplexers with fanout > 1 have to be realized with the output level multiplexer *MUX*3 of a macrocell, or the respective multiplexer has to be duplicated. As a heuristic, we first map a multiplexer with fanout > 1 to the *MUX*3 multiplexer of a macrocell. Additionally, input multiplexers to this multiplexer with fanout $= 1$ are mapped to the same macrocell.

As a second step, the multiplexers that have a function g and its complement $\bar{g}$ as input are taken into consideration. Because of the restricted internal connectivity of a macrocell, the complementation of a data-input function realized according to Fig. 1 has only one minimal mapping shown in Fig. 5(c). However, if the function $\bar{g}$ is an input to more than two other multiplexers, it is realized in a separate macrocell. Finally, the remaining multiplexers are mapped level by level to the macrocells.

The next section gives the results obtained by the implementation of the multiplexer synthesis algorithm. The results of this algorithm followed by mapping to the ACT1 macrocell are also presented.

VI. Benchmark Results

As stated in the previous section, the mapping of $M(2)$ multiplexer circuits to the ACT™ FPGA's from Actel can be done directly. However, an $M(1)$ multiplexer circuit has to be matched to the macrocells.

Table IV lists the results for the MCNC benchmarks for the multiplexer synthesis algorithm. The results are given in column *mux-map*. For comparison, we give the results for *misII* [29], [40] with the Actel library of realizable functions as a representative for general technology mappers and the result of Actel specific mappers *mis-pga* [40], *Prosperine* [28], [32], *Amap* [31], and *ASYL* [41].

The results of the multiplexer synthesis given in the columns $k = 1$ and $k = 2$ assume that inverters are available to realize complemented data input functions. The columns *modules* give the total number of necessary $M(1)$ or $M(2)$ modules, and the columns *depth* gives the longest path in the circuit. The columns *cells* give the number of necessary macrocells. The time ((*ti.*)) is given in seconds. The computation times given for *mux-map* include the multiplexer synthesis and the mapping step. The results were obtained on a Sparc 4/370 (12.5 mips). For *mis-pga* (new) and *Amap*, the time was obtained on a DEC5500 (28 mips).

It should be stressed, that *mux-map* performs the synthesis and mapping for the *act*0 type combination of $M(1)$ multiplexers (Fig. 2(b)). Additionally, the data-select inputs of the multiplexers are restricted to be input variables except for the realization of an inverter. Nevertheless, results close or even better than the ones obtained by the algorithms that take advantage of all the *act*1 and *act*2 macrocell possibilities are obtained. Moreover, the execution times for *mux-map* are much shorter than for *mis-pga*, which gives the closest results compared to *mux-map*.

VII. Conclusion

The concept of a *local* transform has been applied to the synthesis of multiplexer circuits for incompletely specified multioutput Boolean functions. The program based on this concept has been implemented. The results show that the special case of the $M(1)$ multiplexer synthesis with following mapping to the ACT1 family usually leads to the same number of macrocells and is generally faster than the conventional multilevel minimization followed by mapping [29]–[32]. However, contrary to those algorithms, the presented algorithm has been designed for general multiplexer synthesis and not specifically for the Actel FPGA's. By using a more sophisticated mapping algorithm, the obtained results can be even further improved. Moreover, the obtained multiplexer circuit can be easily converted to a mixed Exclusive OR multiplexer circuit, where multiplexers as in Fig. 5(c) are replaced by an Exclusive OR. The results of *mux-map* for benchmark functions that are known to have a high Exclusive OR component like f51m, rd73, 5xp1, and sao2 are better than the results obtained by any of the other programs. Our future extensions are to expand the mapping algorithm to the ACT2 family. We would also like to implement the algorithm based on OBDD's, where we hope to obtain a significantly improved execution time and smaller memory requirement.

Acknowledgment

The authors would like to thank the reviewers for their careful review and their helpful comments.

References

[1] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proc. IEEE*, vol. 78, no. 2, pp. 264–300, Feb. 1990.
[2] D. Bostick *et al.*, "The Boulder optimal logic design system," in *IEEE Proc. Int. Conf. CAD*, Nov. 1987, pp. 62-65.
[3] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: Multilevel interactive logic optimization system," *IEEE Trans. Comput. Aided Design*, vol. 6, no. 6, pp. 1062-1082, Nov. 1989.
[4] K.-C. Chen and S. Muroga, "SYLON-DREAM: A Multilevel network synthesizer," in *Proc. IEEE Int. Conf. CAD*, 1989, pp. 552-555.
[5] A. L. Lavagno, S. Malik, R. K. Brayton, and A. Sangiovanni-Vincentelli, "MIS-MV: Optimization of multilevel logic with multiple-valued inputs," in *Proc. IEEE Int. Conf. CAD*, (Santa Clara), Nov. 1990, pp. 560-563.
[6] B.-G. Kim and D. L. Dietmeyer, "Multilevel logic synthesis with extended arrays," *IEEE Trans. Comput.*, vol. 41, no. 2, pp. 142-157, Feb. 1992.
[7] S. S. Yau, and C. K. Tang, "Universal logic modules and their applications," *IEEE Trans. Comput.*, vol. G19, no. 2, pp. 142-149, Feb. 1970.
[8] S. L. Hurst, *The Logical Processing of Digital Signals*. Crane Russak, 1978.
[9] X. Chen, and S. L. Hurst, "A consideration of the minimum number of input terminals on universal logic gates and their realization," *Int. J. Electron.*, vol. 50, pp. 1-13, Jan. 1981.
[10] S. Bandyopadhyay, A. Pal, and A. K. Choudhury, "Characterization of unate cascade realizability using parameters," *IEEE Trans. Comput.*, vol. G24, no. 2, pp. 218-219, Feb. 1975.
[11] D. G. Whitehead, "Algorithm for logic-circuit synthesis by using multiplexers," *Electron. Lett.*, vol. 1977, no. 12, pp. 355-356, June 1977.
[12] 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.
[13] R. P. Voith, "ULM implicants for minimization of universal logic module circuits," *IEEE Trans. Comput.*, vol. G26, no. 5, pp. 417-424, May 1977.
[14] B. Dormido and D. Canto, "Systematic synthesis of combinational circuits using multiplexers," *Electron. Lett.*, vol. 14, no. 18, pp. 588-590, Aug. 1978.
[15] G. G. Langdon, "A decomposition chart technique to aid in realizations with multiplexers," *IEEE Trans. Comput.*, vol. G27, no. 2, pp. 157-159, 1978.
[16] L. A. M. Bennett, "The application of map-entered variables to the use of multiplexers in the synthesis of logic functions," *Int. J. Electron.*, vol. 45, no. 4, pp. 373-379, 1978.
[17] D. Mange and E. Sanchez, "Synthese des fonctions logiques avec des multiplexeurs," *Digital Processes*, vol. 4, pp. 29-44, 1978.
[18] A. J. Tosser and D. Aoulad-Syad, "Cascade networks of logic functions built in multiplexer units," *Proc. Inst. Elec. Eng.*, pt.E, vol. 127, no. 2, pp. 64-68, 1980.
[19] D. H. Green and M. A. Chughtai, "Use of multiplexers in direct synthesis of ASM-based designs," *Proc. Inst. Elec. Eng*, pt.E, vol. 133., no. 4, pp. 194-200, July 1986.
[20] A. Pal, "An algorithm for optimal logic design using multiplexers," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 755-757, Aug. 1986
[21] R. K. Gorai and A. Pal, "Automated synthesis of combinational circuits by cascade networks of multiplexers," *Proc. Inst. Elec. Eng.*, pt.E, vol. 137, no. 2, pp. 164-170, Mar. 1990.
[22] R. K. Gorai, and A. Pal, "Automated synthesis of combinational circuits by tree networks of multiplexers," in *Proc. 3rd Int. Conf. VLSI Design*, (Bangalore, India), Jan. 1990, pp. 300-305.
[23] A. M. Lloyd, "Design of multiplexer universal-logic-module networks using spectral techniques," *Proc. Inst. Elec. Eng.*, pt.E., vol. 127, no. 1, pp. 31-36, Jan. 1980.
[24] T. F. Tabloski and F. J. Mowle, "A numerical expansion technique and its application to minimal multiplexer logic circuits," *IEEE Trans. Comput.*, vol. 25, no. 7, pp. 684-702, July 1976.
[25] S. L. Hurst, D. M. Miller, and J. C. Muzio, *Spectral Techniques in Digital Logic*. New York: Academic, 1985.
[26] B. J. Falkowski, I. Schäfer, and M. A. Perkowski, "Effective computer methods for the calculation of the Rademacher–Walsh spectrum for Boolean Functions,"*IEEE Trans. Comput-Aided Design*, vol. 11, no. 10, pp. 1207-1226, Oct. 1992.
[27] Actel, *ACT Family Field Programmable Gate Array Data Book*, Mar. 1991.
[28] 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," in *Proc. 1st ACM Workshop FPGA's*, (Berkeley, CA), Feb. 1992, pp. 103-108.
[29] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic synthesis for programmable gate ar-

rays," in *Proc. 27th ACM/IEEE Design Automat. Conf.*, 1990, pp. 620-625.

[30] F. Mailhot and G. DeMicheli, "Technology mapping using Boolean matching and don't care sets," in *Proc. IEEE Eur. Design Automat. Conf.*, (Glasgow), Mar. 1990, pp. 212-216.

[31] K. Karplus, "Amap: A technology mapper for selector-based field-programmable gate arrays," in *Proc. 28th ACM/IEEE Design Automat. Conf.*, (San Francisco), June 1991, pp. 244-247.

[32] S. Ercolani and G. DeMicheli, "Technology mapping for electrically programmable gate arrays," in *Proc. 28th ACM/IEEE Design Automat. Conf.*, (San Francisco), June 1991, pp. 234-239.

[33] B. J. Falkowski and M. A. Perkowski, "A family of all essential radix-2 addition/subtraction multipolarity transforms: Algorithms and interpretations in Boolean domain," in *Proc. Int. Symp. Circ. Systems (ISCAS)*, (New Orleans), May 1990, pp. 2913-2916.

[34] M. Fujita, Y. Matsunaga, and T. Kakuda, "On variable ordering of binary decision diagrams for the application of multi-level logic synthesis," in *Proc. IEEE Europ. Design Automat. Conf.*, (Amsterdam), Feb. 1991, pp. 50-54.

[35] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchanges of variables," in *Proc. IEEE Int. Conf. CAD*, (Santa Clara, CA), Nov. 1991, pp. 472-479.

[36] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1990.

[37] S. Purwar and A. K. Susskind, "Computation of Walsh spectrum from binary decision diagram and binary decision diagram from Walsh spectrum," *Computers Elect. Eng.*, vol. 15, no. 2, pp. 59-65, 1989.

[38] D. Varma and E. A. Trachtenberg, "Design automation tools for efficient implementation of logic functions by decomposition," *IEEE Trans. Comput.-Aided Design*, vol. 8, pp. 901-916, Aug. 1989.

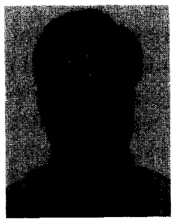
[39] D. L. Dietmayer, *Logic Design of Digital Systems*. New York: Allyn and Bacon, 1978.

[40] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "An improved synthesis algorithm for multiplexor-based PGA's," in *Proc. 29th ACM/IEEE Design Automat. Conf.*, (Anaheim, CA), June 1992, pp. 380-386.

[41] T. Besson, H. Bousouzou, M. Crates, and G. Saucier, "Synthesis on multiplexer-based programmable devices using (ordered) binary decision diagrams," in *Proc. EUROASIC*, (Paris, France), June 1992, pp. 8-13.

[42] Concurrent Logic, *CLi6000 Series Field Programmable Gate Array*, preliminary information, Dec. 1991.

[43] Algotronix Ltd., *Configurable Array Logic User Manual*, Edinburgh, UK, 1991.



Ingo Schäfer received the B.S.E.E. degree from the University of Stuttgart, Germany, in 1987 and the Ph.D. degree in electrical and computer engineering from Portland State University in 1992.

From 1989 to 1992, he was a member of the CAD group at Portland State University. In 1992, he joined the logic synthesis group at Lattice Semiconductor. His research interests include various issues in computer-aided design as well as in digital signal and image processing.

Marek A. Perkowski (M'84) was born in Warsaw, Poland, on October 6, 1946. He received the M.S. degree in electronics (automatic control) in 1970 and the Ph.D. degree in automatics (digital systems) in 1980 from Technical University of Warsaw.

He was an Assistant Professor at Technical University of Warsaw from 1980 to 1981, a Visiting Assistant Professor at the University of Minnesota from 1981 to 1983, and, since 1983, an Associate Professor of Electrical and Computer Engineering at Poland State University. He had summer professor positions with GTE Labs, Intel Scientific Computers, and Sharp Microelectronics Technology and consulted for various companies, most recently Cypress Semiconductor Northwest. He is the co-author of three books, seven book chapters, and over 120 technical articles in design automation, computer architecture, artificial intelligence, image processing, and robotics. His recent research interests include field programmable gate arrays, FPGA-based computer architectures, and building a voice-controlled robot-wheelchair.

Dr. Perkowski is a founding member of the Polish Informatics Society.