Multiple-Valued-Input TANT Networks

Marek A. Perkowski

Department of Electrical Engineering
Portland State University
Portland, OR 97207

Malgorzata Chrzanowska-Jeske

Department of Electrical Engineering
Portland State University
Portland, OR 97207

Abstract

The paper proposes mvTANTs, three-level networks with multiple-valued inputs and binary outputs. These networks are a generalization of binary TANTs (Three level And Not networks with True Inputs). One of possible interpretations of mvTANT is a four-level binary network with input decoders which realize multiple-valued literals. Similarly to mvPLAs, mvTANTs have regular structures with predictable timing. Comparing to mvPLAs, however, they have at least 25 % less input wires to the third-level (NAND) plane and not more outputs from the second-level (AND) plane than the mvPLA. Thus, in many cases they have less gates and connections, and are useful to minimize Boolean functions in cellular FPGAs and other regular structures.

1 Introduction

Some Electronically Programmable Logic Devices [27] and Cellular Field Programmable Gate Arrays, especially those from Motorola, Plessey, or Pilkington, require new kinds of logic synthesis tools, since the classical approaches, as well as the FPGA-specific methods developed recently [11,21] are not very suitable for them.

In cellular FPGA architectures such as Motorola [10], the "design for speed", "design for regularity" and the "design for minimizing connections" requirements are becoming more and more important, while the minimization based on the number of gates as the sole criterium is becoming of less practical value. It is then interesting to investigate various logic structures with a limited number of levels that would display high degree of the connection regularity and small fan-in to the gates. High connection regularity can make the timing more predictable and allow to better optimize the circuit to improve the speed. The requirement that the design method would reduce the fan-in of gates is the result of the main technology constraint in these new FPGAs - all gates in the circuit have two inputs. Such regular array structures would be easy to map to "cellular" FPGAs, and particularly Motorola's MPA10XX series.

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

One possible approach is to generalize the concept of a PLA to a regular structure with planes having other gates than only ANDs in the input plane, and other than only ORs in the output plane [17,16,15]. Additionally, a limited factoring and folding of rows and columns in the planes of this new structure decreases the area [22]. Furthermore, instead of having variables and their negations at the input to the AND plane, one can use two-input, four-output decoders [24,25,26]. This leads to the usage of multiple-valued input algebra as a convenient tool in the synthesis process.

Most of previous applications of mv logic to binary circuit design have been for circuits with two levels (not counting the level of decoders that realize the literals). In the multi-valued literature, there are just a few papers on applying multiple-valued-input algebra to circuits with more than two levels. Works describing PLA decomposition methods [2,26] and multi-level circuits with literal generators realized with Sasao's MACDAS [23] belong to this category.

In this paper we will investigate a new structure, called mvTANT, which generalizes the structure of TANT networks [6] and uses the multiple-valued logic as a mathematical technique to realize binary multi-level logic. The paper is organized as follows. Section 2 presents the structure of binary TANTs and gives the rationale for mvTANTs. Section 3 introduces the literal constraints that lead to the concept of the Multiple-Valued-Input TANT Networks (mvTANT). Basic notions and definitions are given in Section 4. Section 5 gives theorems used to generate implicants for mvTANTs. In Section 6 a method to minimize mvTANT networks using mvTANT implicants is described. Section 7 presents conclusions and future work.

2 Binary TANT Networks

The **"Three level AND NOT networks with** *True inputs"* (so-called *TANT networks*) were introduced by McCluskey and Gimpel. They have the meaningful advantage over the PLA representation. TANT design for function f can never be worse in terms of the number of gates than the corresponding PLA [6]. The structure of a binary TANT network is shown in Fig. 2.1. For presentation purposes, we assume that the network is composed of NAND gates. TANT has only affirmative variables as its inputs, while the PLA has both affirmative variables and their negations as its inputs. Thus, assuming rectangular layout realization, PLA has one dimension of the input plane two times larger. The number of the TANT implicants is also smaller than that of the prime implicants in PLA. Therefore, TANT is usually better. It allows also for better incorporation of the fan-in constraints than the "standard cell" realization of the PLA-type two-level logic. Several algorithms to minimize TANT networks have been published, and some of them were realized as computer programs [6,5,9,7,8,12,3,4,29,30,13].

Another argument for three level logic is given in [25]. It was proven by Sasao that for "nearly all" Boolean functions three levels is enough in the sense, that while increasing the number of levels from two to three the number of gates is substantially reduced. By increasing the number of levels from three to more than three, the reduction in the number of gates is minimal [25]. However, the analysis of solutions obtained with the Sasao's method leads to the observation, that the number of inputs to a gate in a three-level network is usually large. There is also a trade-off of the number of levels and the total number of inputs to gates. From construction, the four-level binary networks are not worse than the networks with a smaller number of levels, and additionally they have usually a much decreased number of gate inputs. Therefore, we increase here the number of levels to four, hoping that the the decreased fan-in and number of gates will outweight the difficulty of the synthesis algorithm. Another rationale for the four-level binary-realized mvTANTs is the following. Since it was experimentally proven that both the mvPLA and the binary TANT networks improve on the binary PLA, the multiple-valued TANT should improve on both the binary TANT and the multiple-valued PLA. In the worst case, the binary TANT reduces to a binary PLA. The same property holds for mv logic: in the worst case, the mvTANT network reduces to the mvPLA. There is then no risk involed in using mvTANTs. By paying a price of more complex synthesis, one gets always a solution that is not worse than the popularly used mvPLA with input decoders.

In addition, the mv TANT concept can be applied to minimize four-level networks with p-valued inputs and binary outputs for any value of p.

3 The Basics of the mvTANT

We will denote binary variables by small letters and multiple-valued (mv) variables by capital letters. The *multiple-valued input literal* (*mv literal,* for short) is defined in a standard way [18].

Let us observe, that in the case of the binary TANT network, the main design constraint is, that out of two polarities, 0 and 1, of a 2-valued variable, the circuit accesses only one polarity. Below we will generalize this constraint to a p-valued logic.

Definition 3.1. The **allowed set of literals** for variable X is a set of literals such that every single-value literal X^i of this variable can be expressed **in a unique way** as a product of some of these literals and their negations.

The set of all values of variable X is denoted by P. A single value is denoted by i, $i = 0, ..., p-1$. There exist p literals such that each of these literals has a set of values $P - \{i\}$, $i = 0 ,..., p-1$. Let us denote by PP the set of these p literals.

Theorem 3.1. Any set PS of $p-1$ literals selected out of set PP creates for variable X an allowed set of literals.

Example 3.1. Let $P = \{0,1,2,3,4\}$. $p = 5$. $PP = \{X^{\{0,1,2,3\}}, X^{\{0,1,2,4\}}, X^{\{0,1,3,4\}}, X^{\{0,2,3,4\}}, X^{\{1,2,3,4\}}\}$. $PS = \{X^{\{0,1,2,3\}}, X^{\{0,1,2,4\}}, X^{\{0,1,3,4\}}, X^{\{0,2,3,4\}}\}$. Let the literal from PP not selected to PS be $X^{\{1,2,3,4\}}$. Value 0 = $\{0,1,2,3\} \cap \{0,1,2,4\} \cap \{0,1,3,4\} \cap \{0,2,3,4\}$. For values 1, 2, 3, 4, $X^{\{i\}} = \overline{X_j}$, where X_j is in PS. For instance $X^{\{1\}} = \overline{X^{\{0,2,3,4\}}}$ $= X^{\{0,1,2,3,4\}-\{0,2,3,4\}}$. From now, we will denote sets in a simplified way: set $\{0,1,2,3\}$ will be denoted as 0123.

For any multi-valued variable, there are several possible sets that are allowed for it. One of them is selected by the designer, and we are not interested here how this set is selected. After the selection, each of the elements of this set is called the *selected literal.* The set of the selected literals is realized as the output functions of a *"unique decoder"*. The name "unique decoder" comes from the property that a decoder with a larger number of outputs either would not create the allowed sets of literals (the single-values could be created in a not unique way), or would have more than the minimum number of outputs. Also, a decoder with a smaller number of outputs than the unique decoder cannot be used, since it would not be able to allow realization of an **arbitrary** Boolean function.

Below, we will use 2-valued, 3-valued and 4-valued logics as illustrations of our general approach to p-valued-input TANT. For 2-valued variable there are two allowed sets of literals $\{X^0\}$ and $\{X^1\}$, and one is selected as the selected literal. For 3-valued variable there are three possible literals X^{01}, X^{02} and X^{12}, and any two of them are selected as the selected literals. For every 4-valued variable X_i the allowed set of literals is: $\{{X_i}^J, {X_i}^K, {X_i}^L\}$, where J, K, L are various subsets of the set { 012, 013, 023, 123 }.

The set of values of the selected literal is called the *set of selected values.* Every given set of selected values for a variable will be called a *polarity* of this variable. It results from the above definitions and Theorem 2.1 that for every variable and associated polarity there exists a set of selected literals and *a unique decoder of this variable.*

From the point of view of using the mvTANT concept to design binary circuits, the 2^k-valued input logic must be used. In the case of 2^k-valued logic, the mvTANT network has k-input, $2^k - 1$-output decoders in the fourth level (the input level). (It has NAND gates in the remaining three levels, see Fig. 3.1.) From the practical point of view, k = 2,3, since larger values of k would lead to too many outputs from the k-to-$2^k - 1$ decoders.

Since in a binary TANT only signals corresponding to affirmative variables are on the inputs to the third level plane, in comparison with a PLA which has both affirmative and negated variables, one dimension of this plane is reduced in binary TANT by 50 %. In case of mvPLAs k=2 leads to 2-input, 4-output decoders, but for mvTANT only 3-output unique decoders are used. This makes a gain of 25 % in the third level plane, when compared to an mvPLA, or a standard PLA which uses 4 outputs for two variables (two variables, plus their two negations). The decrease of the total area of mvTANT with respect to mvPLA is, however, even more substantially caused by the large decrease in the number of gates of the second level mvTANT plane. In case when variables are grouped to triplets and an 8-valued logic is used to describe the outputs of the 3-input 8-output decoders, the number of decoder outputs which has to be used on the input to the third level plane of the mvTANT is seven. This is one wire more than in the case of three variables with their negations in a binary PLA, but the reduction in the number of gates on the second level usually much outweights this increase.

We distinguish three types of binary variables: *single variables, paired variables,* and *tripled variables.* The *paired variables* are binary variables allocated to groups of two variables. The *tripled variables* are variables allocated to groups of three variables. In the first synthesis stage, not presented here, an allocation program analyzes the partial symmetries of all pairs of binary variables, and on this base allocates every binary variable to only one group: with one, two or three variables. This algorithm selects also the polarity of each multiple-valued variable that corresponds to this group of binary variables.

The problem of pairing binary variables to $2^k - 1$-valued variables, as well as the related problem of selecting the polarities of $2^k - 1$-valued variables, are quite difficult. Even more challenging is the problem of grouping binary variables to any number of groups of $1, 2, 3, 4, \ldots, k$ ($k \leq n$) variables. This can be still generalized to the case that the same variable is used as an input to several neighboring decoders. This means, allocating the same variable to *more than one group.* The Binary Decision Diagrams (BDDs) and Orthogonal Decision Diagrams (ODDs) [15], as well as total and partial symmetries of variables observed in them, are crucial to finding those variables' groupings and polarities.

A general structure of mvTANT with single, paired, and tripled variables in the decoder level is shown in Fig. 3.1. The unique decoder for a single binary variable corresponds to using this variable on the input in a positive polarity (as a wire), or in a negative polarity (with an inverter). Other unique decoders were explained above. Our approach (both the binary and the multi-valued variant) allow also for direct connections between any two levels of a mvTANT, while the approaches from [25] and [6] restrict the connectivity only to the neighboring levels. One of possible layout floorplans for FPGA/PGA/VLSI realization of binary circuit corresponding to a mvTANT is shown in Fig. 3.2. The functions realized in this network are:

$$y_1 = \overline{(a \Leftrightarrow b)(c + \overline{d})e} \ \ \overline{f} \ + \ (\overline{a} + \overline{b}) \ c \ d \ \overline{f},$$

$$y_2 = \overline{(\overline{a} + \overline{b}) \ c \ d \ \overline{f}} \ (c + \overline{d}) \ + \ \overline{e},$$

$$y_3 = \overline{(a \Leftrightarrow b)(c + \overline{d})e} \ \ \overline{(\overline{a} + \overline{b}) \ c \ d \ \overline{f}} \ \ \overline{\overline{b} \ e \ \overline{f}} \ (\overline{a} + b) \ + \ \overline{(a \Leftrightarrow b)(c + \overline{d}) \ e} \ \ \overline{f} \ + \ \overline{(\overline{a} + \overline{b}) \ c \ d \ \overline{f}} \ (c + \overline{d}),$$

$$y_4 = (\overline{a} + \overline{b}) \ c \ d \ \overline{f} \ + \ \overline{b} \ e \ \overline{f}.$$

The outputs of the third level are: $\overline{r_1} = (a \Leftrightarrow b)(c + \overline{d})e$, $\overline{r_2} = (\overline{a} + \overline{b}) \ c \ d \ \overline{f}$, $\overline{r_3} = \overline{b} \ e \ \overline{f}$. The outputs of the second level are: $\overline{s_1} = \overline{(a \Leftrightarrow b)(c + \overline{d})e} \ \ \overline{f}$, $\overline{s_2} = \overline{(\overline{a} + \overline{b}) \ c \ d \ \overline{f}} \ \ (c + \overline{d})$, $\overline{s_3} = r_1 \ r_2 \ r_3 \ (\overline{a} + b)$. If necesary, the outputs of all 4 levels can be used as primary outputs.

Figure 3.3. presents a layout of a mvTANT with truly multiple-valued inputs, in this case, 3-valued. The function realized in this layout is $F = A^{01} \ \overline{C^{01} \ C^{02}} \ \ \overline{C^{02} \ D^{02}} \ \ \overline{D^{01} \ D^{02} \ B^{01}} + C^{01} C^{02} \ B^{02} \ \overline{D^{01} \ D^{02} \ B^{01}}$ and its optimization will be discussed in more detail in Example 4.1.

4 Fundamental Definitions

Example 4.1. The 3-valued-input function presented in the Marquand map from Fig. 4.1 can be minimized as the following mvTANT expression:

$$F = A^{01} \ C^{12} \ \overline{C^{02} \ D^{02}} \ \ \overline{D^{0} B^{01}} \ + \ C^{0} \ B^{02} \ \overline{D^{0} \ B^{01}}$$
$$= \ A^{01} \ \overline{C^{01} \ C^{02}} \ \ \overline{C^{02} \ D^{02}} \ \ \overline{D^{01} \ D^{02} \ B^{01}}$$
$$+ \ C^{01} \ C^{02} \ \ B^{02} \ \overline{D^{01} \ D^{02} \ B^{01}}$$

It is assumed that the selected literals are: A^{01}, A^{02}, B^{01}, B^{02}, C^{01}, C^{02}, D^{01}, and D^{02}. The mvTANT-implicants from above expressions are shown in Fig. 4.2. The corresponding network is shown in Fig. 4.3. It has 6 gates and 17 connections (decoders not counted). We will say that the cost is (6, 17). The **gate cost** is 6 and the **connection cost** is 17. We can minimize the **total cost** that can be **any** weighted sum of these two costs, but some minimization properties hold only for some weight combinations.

By applying the de Morgan's theorem to the first and second levels in Fig. 4.3, one can observe that the first level plane of NANDs in the mvTANT realizes a logical sum (the OR plane), the second level of NANDs realizes a product (the AND plane), and the third level of NANDs realizes the negation of the product of selected literals (the NAND plane). The expressions written in Fig. 4.3 near the second level NANDs correspond then to the output of the AND plane (negations of NAND gates).

TANT network minimization problem consists in finding the Boolean expression that minimizes the total cost. It means that the synthesis method should minimize simultaneously the second and the third levels. (It is assumed that the decoders have been already selected earlier, and are not included to the total network cost.)

Definition 4.1. The **available literal AVL(X),** for variable X, is a selected literal of this variable, or a product of any number of selected literals of this variable.

Definition 4.2. The **available product AVP(PL),** of a product PL of literals, is a product of available literals for the variables from product PL, such that $AVP(PL)$ includes PL.

Example 4.2. Assuming that the selected polarities of 4-valued variables X and Y are {012,013,123}, and the selected polarity of variable Z is {023,013,123}, the selected literals are: $X^{012}, X^{013}, X^{123}, Y^{012}, Y^{013}, Y^{123}, Z^{023}, Z^{013}, Z^{123}$. The available literals for variable X are: AVL(X) $= \{ X^{012}, X^{013}, X^{123}, X^{01}, X^{13}, X^{12}, X^{1}\}$. The available literals for variable Y are: AVL(Y) $= \{Y^{012}, Y^{013}, Y^{123}, Y^{01}, Y^{13}, Y^{12}, Y^{1}\}$. The available literals for variable Z are: AVL(Z) $= \{Z^{023}, Z^{013}, Z^{123}, Z^{23}, Z^{13}, Z^{03}, Z^{3}\}$. The available products of product $PL = X^0\ Y^1\ Z^1$ are those elements of the Cartesian Product: AVL(X) $\times$ AVL(Y) $\times$ AVL(Z) that cover the product PL. For instance, $X^{012}\ Y^{013}\ Z^{013}$ is one of the available products of PL: $AVP(PL) = AVP(X^0\ Y^1\ Z^1) = X^{012}\ Y^{013}\ Z^{013} \supseteq X^0\ Y^1\ Z^1 = PL$. Available products are mv cubes [18,28], and the cube inclusion operation is defined in a standard way.

Let us observe, that the set of these available products is very large, when compared to a number of products of affirmative variables in a binary TANT, or to a number of literals in mvPLA. The solution space searched here includes the previous spaces and is significantly larger, which is the reason why an mvTANT gives usually much better solutions than both a binary TANT and a standard mvPLA with the same literals selected.

Definition 4.3. A **permissible expression** is a Boolean expression of the form P $= H\ \overline{T_1}\ \overline{T_2}\ \ldots\ \overline{T_m}$ where both H and T_i are products of selected literals. H is called the **head of permissible expression** and each T_i is called a **tail product** while $\overline{T_i}$ is called a **tail factor.** A **permissible implicant of a function** f **(mvTANT-implicant)** is a permissible expression which implies function f.

Example 4.3. The Boolean expression $A^{01}\ C^1\ D^{12}$ is a prime implicant of the function from Example 4.1, and $A^{01}\ \overline{C^{02}\ C^{01}}\ \overline{C^{02}\ D^{02}}\ \overline{D^0 B^{01}}$, $A^{01}\ \overline{C^{02}\ C^{01}}\ \overline{C^{02}\ D^{02}}\ \overline{B^{01}}$, $A^{01}\ \overline{C^{02}\ C^{01}}\ \overline{D^{02}}\ \overline{D^0 B^{01}}$, $C^0\ B^{02}\ \overline{D^0\ B^{01}}$, $C^0\ B^{02}\ \overline{D^0}$ are some of the mvTANT-implicants of this function.

Definition 4.4. The heads of mvTANT-implicants of function f are called the **second level groups.** The set of all second level groups is denoted by H_f. The tail products of mvTANT-implicants are called the **third level groups.** The set of all third level groups is denoted by T_f.

Example 4.4. For the mvTANT network of the function from Example 4.1, realized in Fig. 4.3, the products (cubes) A^{01} and $C^{01}\ C^{02}\ B^{02}$ are the second level groups. The products, $C^{02}\ D^{02}$ and $D^{01}\ D^{02}\ B^{01}$ ($= D^0\ B^{01}$), are the third level groups.

Similar to two-level minimization, in which a solution is a covering with prime implicants, the solution of mvTANT is a covering with mvTANT-implicants. In mvTANT case, however, the situation is more complicated, since several mvTANT-implicants cover the same set of minterms, but have different tail factors. The selection of mvTANT-implicants must be then done in such a way that the third level groups are maximally shared among the mvTANT-implicants, and are also of the smallest cost.

Let us observe in Example 4.4. that there are two representations for the last group that we will use interchangeably. The rule $A^s\ \cdot\ A^r\ =\ A^{s\ \cap\ r}$ is used to change from one form to the other. In general, the process called *normalization* is used to create a standard form of implicants, called the principal mvTANT-implicants.

Property 4.1. The following rules of Boolean transformations are true:

(4.1) $A^r\ \overline{B^l\ A^k}\ =$

$$A^r\ \overline{B^l\ A^{\{r \cap k\}\ \cup\ SUBSET\{VAL\ -\ (r\ -\ l)\}}}$$

where VAL is the set of all values of variables A, B. $SUBSET(S)$ is any subset of set S, and r, k, l are non-empty subsets of VAL.

(4.2) $A^r\ \overline{B^l\ A^{r\ \cap\ s}}\ \rightarrow\ A^r\ \overline{B^l\ A^s}\ , when\ \ r\ \neq\ \ s$

(4.3) $A^r\ \overline{B^l\ A^r}\ \rightarrow\ A^r\ \overline{B^l}$

Definition 4.5. Normalization of an mvTANT-implicant is the process of applying rules (4.2) and (4.3) to this implicant until all literals will become the selected literals.

Example 4.5. For the 4-valued-input function from Fig. 4.4 assume the following selected literals: X^{023}, X^{013}, X^{123}, Y^{023}, Y^{013}, Y^{123}. Then some of the available products are: $Y^{23} = Y^{023} \cdot Y^{123}$, $Y^{13} = Y^{013} \cdot Y^{123}$, $Y^{03} = Y^{013} \cdot Y^{023}$, $Y^3 = Y^{023} \cdot Y^{013} \cdot Y^{123}$, $X^3 = X^{023} \cdot X^{013} \cdot X^{123}$. Applying rule (4.1) we get: $Y^{23}\ \overline{X^3\ Y^{013}} = Y^{23}\ \overline{X^3\ Y^{03}} = Y^{23}\ \overline{X^3\ Y^3} = Y^{23}\ \overline{X^3\ Y^{13}}$ Applying normalization to any of the above, we get: $Y^{023}\ Y^{123}\ \overline{X^{023}\ X^{013}\ X^{123}\ Y^{013}}$.

Definition 4.6. The **permissible realization** for the function f is a logic sum of the set of mvTANT-implicants which cover all minterms of the function. The **optimal permissible realization for function** f, denoted by $OPR(f)$, is such a permissible realization that its corresponding TANT network has the minimum total cost.

Definition 4.7. The **prime permissible implicant,** pp_f-implicant for short, is such a permissible implicant that it is not properly included by prime implicants and if any tail factor is removed from it, the resulting expression will not be an mvTANT-implicant any more. The set of all pp_f-implicants is denoted by PP_f.

Example 4.6. For function f from Example 4.1 some of the pp_f-implicants are:

$A^{01}\ C^{12}\ \overline{C^{02}\ D^{02}}\ \overline{D^0 B^{01}}$, $C^0\ B^{02}\ \overline{D^0\ B^{01}}$,

$A^{01}\ C^{12}\ \overline{C^2\ A^{01}\ D^{02}}\ \overline{D^0\ B^{01}\ C^{12}}$,

$A^{01}\ \overline{C^{01}\ C^{02}}\ \overline{C^{02}\ D^{02}}\ \overline{D^{01}\ D^{02}\ B^{01}}$,

$C^0\ B^{02}\ \overline{D^0\ B^{01}}$, $C^{01}\ C^{02}\ B^{02}\ \overline{D^{01}\ D^{02}\ B^{01}}$.

For instance, by removing $\overline{D^0\ B^{01}}$ from $C^0\ B^{02}\ \overline{D^0\ B^{01}}$ one creates $C^0\ B^{02}$ which **is not** an mvTANT-implicant.

Definition 4.8. The **principal mvTANT-implicant,** pc_f-implicant for short, is such a normalized pp_f-implicant that its tail products don't contain the literals from its head. The set of all pc_f-implicants is denoted by PC_f.

Example 4.7. Assuming selected literals: A^{01}, A^{02}, B^{01}, B^{02}, C^{01}, C^{02}, D^{01}, and D^{02}, the mvTANT-implicant: $A^{01}\ C^{12}\ \overline{C^{2}\ A^{01}\ D^{02}}\ \ \overline{D^{0}\ B^{01}\ C^{12}}$

is transformed to:

$A^{01}\ C^{12}\ \ \overline{C^{12}\ C^{02}\ A^{01}\ D^{02}}\ \ \overline{D^{01}\ D^{02}\ B^{01}\ C^{12}}$

next to: $A^{01}\ C^{12}\ \ \overline{C^{02}\ D^{02}}\ \ \overline{D^{01}\ D^{02}\ B^{01}}$

and next to: $A^{01}\ \overline{C^{01}\ C^{02}}\ \ \overline{C^{02}\ D^{02}}\ \ \overline{D^{01}\ D^{02}\ B^{01}}$

which is the normalized form of the pc_f-implicant for these selected literals.

Assuming now another selected literals: A^{01}, A^{12}, B^{01}, B^{12}, C^{01}, C^{02}, D^{01}, and D^{02}, the mvTANT-implicant: $A^{01}\ C^{12}\ \overline{C^{2}\ A^{01}\ D^{02}}\ \ \overline{D^{0}\ B^{01}\ C^{12}}$

is transformed to:

$A^{01}\ C^{12}\ \ \overline{C^{02}\ C^{12}\ D^{02}}\ \ \ \overline{D^{01}\ D^{02}\ B^{01}\ C^{12}}$

and next to: $A^{01}\ C^{12}\ \ \overline{C^{02}\ D^{02}}\ \ \overline{D^{01}\ D^{02}\ B^{01}}$

which is the normalized form of the pc_f-implicant for these selected literals.

Example 4.8. Another normalization, assuming selected literals B^{01}, B^{02}, C^{01}, and C^{02}, is:

$C^{0}\ B^{02}\ \overline{B^{0}}\ \Rightarrow\ C^{01}\ C^{02}\ B^{02}\ \ \overline{B^{02}\ B^{01}}\ \Rightarrow$
$C^{01}\ C^{02}\ B^{02}\ \ \overline{B^{01}}$.

All mvTANT-implicants, that produce the same principal mvTANT-implicant after normalization, have the same "shape" on the Marquand chart - they cover exactly the same true minterms and don't care minterms.

5 Generating Necessary mvTANT Implicants

In this section we will give definitions and theorems which are next used to generate efficiently only those mvTANT implicants, called the *necessary mvTANT-implicants*, that can be included in a minimal solution.

Definition 5.1. The **maximal pc_f-implicant,** mp_f-implicant for short, is such an pc_f-implicant which is not included in other pc_f-implicants. The set of all mp_f-implicants is denoted by M_f.

Theorem 5.1. Every tail product of an mp_f-implicant is included (cube inclusion) in some tail product of all pc_f-implicants which are covered by this mp_f-implicant (set inclusion for minterms).

Example 5.1. For function from Example 4.1:

$A^{01}\ C^{12}\ \overline{C^{02}\ D^{02}}\ \ \overline{D^{01}\ D^{02}\ B^{01}}$

is the maximal implicant, but $C^{01}\ C^{02}\ B^{02}\ \ \overline{B^{01}}$ is not, since it is included in the principal implicant $C^{01}\ C^{02}\ B^{02}\ \ \overline{D^{01}\ D^{02}\ B^{01}}$. The tail products of the mp_f-implicant $A^{01}\ C^{12}\ \overline{C^{02}\ D^{02}}\ \ \overline{D^{01}\ D^{02}\ B^{01}}$ are $C^{02}\ D^{02}$ and $D^{01}\ D^{02}\ B^{01}$. They are included in the tail products of all pc_f-implicants created from $A^{01}\ C^{12}\ \overline{C^{02}\ D^{02}}\ \ \overline{D^{01}\ D^{02}\ B^{01}}$ by removing any combinations of literals from the tail factors.

In two-level minimization, one does not use in the covering stage those products implicants that are included in prime implicants. Similarly, the analysis of various kinds of mvTANT-implicants allows to detect those mvTANT-implicants that will not occur in at least one exact minimum solution, and can be, therefore, excluded from further considerations.

Some other category of useful mvTANT-implicants are called *maximal*. They cover locally the maximum number of minterms and are thus useful in greedy heuristic algorithms. They are also used to create pp_f-implicants included in them. This is done by removing any literals from tail factors.

Definition 5.2. The **augmented** pp_f-implicant, ap_f for short, is such a pp_f-implicant that it is not a pc_f-implicant. The set of all ap_f-implicants is denoted by AP_f.

Example 5.2. For function from Example 4.1 $A^{01}\ C^{12}\ \ \overline{C^{2}\ A^{01}\ D^{02}}\ \ \overline{D^{0}\ B^{01}\ C^{12}}$ is an example of an augmented pp_f-implicant.

Definition 5.3. The **necessary** ap_f-implicant, na_f-implicant for short, is such an ap_f-implicant that all of its tail factors can be shared by other pp_f-implicants of a different head. The set of all na_f-implicants is denoted by NA_f. The **unnecessary** ap_f-implicant is the ap_f-implicant which is not an na_f-implicant and is called an una_f-implicant.

Example 5.3. For a binary case, $f = a\ \overline{b}\ +\ b\ \overline{a}$. TANT-implicant $a\ \overline{ab}$ is necessary since its tail factor $\overline{ab}$ can be used in another TANT-implicant, $b\overline{a}$, creating a TANT-implicant $b\ \overline{ab}$ and thus leading to the minimal TANT solution $f = a\ \overline{ab} + b\ \overline{ab}$. However, in $f2 = a\overline{b}\ +\ b\overline{c}$ the TANT-implicant $a\ \overline{ab}$ is **not** necessary since its tail factor $\overline{ab}$ is useless in all other TANT-implicants of $f2$ (in this case, in $b\ \overline{c}$).

Theorem 5.2. If an una_f-implicant is excluded from being selected to OPR sets, then at least one exact optimum solution is retained as an OPR.

This can be compared to dominated primes in Quine table for PLA minimization. If one removes all primes dominated by other primes, some exact solutions may be not generated, but at least one exact solution will be retained.

Definition 5.4. The **necessary pp_f-implicant,** np_f-implicant for short, is such a pp_f-implicant that is not an una_f-implicant. The set of all np_f-implicants is denoted by N_f.

From Definitions 4.6, 4.7, 4.8, 5.2, 5.3, and 5.4. one can conclude that $N_f = PC_f \cup NA_f$

The next theorem results directly from these properties and the definitions of pc_f-implicants and na_f-implicants.

Theorem 5.3. Every ap_f-implicant can be generated by addition of any number of literals included in its head to any subset of its tail products.

Theorem 5.4. The available products of all the prime implicants of function f (array ON) are sufficient as the heads of pp_f-implicants.

Example 5.4. The prime implicants for the function from Example 4.5 are: $X^{01}\ Y^{23}$, $X^{02}\ Y^{23}$, $X^{12}\ Y^{23}$, Y^{2}. The available products of these implicants are $X^{013}\ Y^{23}$, $X^{023}\ Y^{23}$, $X^{123}\ Y^{23}$, Y^{23}, which are the heads of pp_f-implicants $X^{013}\ Y^{23}\ \ \overline{X^{3}\ Y^{013}}$, $X^{023}\ Y^{23}\overline{X^{3}\ Y^{013}}$, $X^{123}\ Y^{23}\overline{X^{3}\ Y^{013}}$, $Y^{23}\ \ \overline{X^{3}\ Y^{013}}$, respectively.

Theorem 5.5. The available products of all prime implicants of the function f' (complement of f) are sufficient as the tail products of mp_f-implicants of f.

Example 5.5. The prime implicants for the negation of the function in Example 4.5 are Y^{01}, $X^3\ Y^{013}$. The available product of implicant $X^3\ Y^{013}$ is $X^3\ Y^{013}$ which is the tail product of the mp_f-implicant $Y^{23}\ \overline{X^3\ Y^{013}}$ from Example 5.4.

6 Algorithm to minimize mvTANTs

Similar to PLA and TANT minimization, the algorithm to minimize mvTANTs has two stages. In the first stage all the principal mvTANT-implicants are generated from the set MSI of the *minimally split* implicants *(MSI-implicants)* [1]. Next, the tail factors that are useful to create necessary mvTANT-implicants are generated, as well as necessary mvTANT-implicants. This determines the search space for the *" mvTANT covering problem."* The mvTANT covering problem is in essence a *covering/closure* or *binate covering* problem, which is solved in the second stage using a decision function DF, similar to the *Petrick* or *Helliwell* [14] decision functions.

The first stage is executed in the following way.

Given is a Boolean function as sets ON and DC.

1) generated is set MSI of minimally split implicants [1].

2) set of prime implicants of function f', a complement to f, is found. This set is denoted by $PFNOT$.

3) set M_f of mp_f-implicants is generated using sets MSI and $PFNOT$ (Theorem 5.5 is used). The heads of mp_f-implicants are available products of implicants from MSI. For each maximal implicant, it is noted, what MSI-implicant it was generated from. Each tail product of this mp_f-implicant obtains a unique name.

4) set PC_f of pc_f-implicants is created from set M_f by removing all possible subsets of literals. Every generated tail factor obtains a unique name. It is noted, in which pc_f-implicants it can be used, and which tail factor from an mp_f-implicant would be created from it by normalization.

5) set N_f of na_f-implicants is generated. All new tail factors generated obtain unique names. It is noted, in which pc_f-implicants they can be used, and to which tail factors from pc_f-implicants they correspond.

In the second stage for every minimally split implicant MSI_i an elementary decision function $DF(MSI_i)$is created. This function describes all possible conditions of covering this MSI_i with principal and necessary mvTANT-implicants. The global exact minimization problem is stated as a problem of *finding the minimum satisfying set of literals to a Boolean product DF*, the product of all the decision functions for all the minimally split implicants:

$DF = \bigcap DF(MSI_i)$, for all MSI_i from MSI.

This problem is NP-hard.

The same sub-functions that exist in various decision functions $DF(MSI_i)$ are encoded with the same Boolean variables in function DF.

The $DF(MSI_i)$ function for a MSI_i implicant is created as follows:

$$(\bigcup pc_j) \cdot \ \bigcap (pc_j \Rightarrow \bigcap t_{j_r}) \ \cdot \ \bigcap (t_{j_r} \Rightarrow t_{j_{r_n}})$$

where: $\Rightarrow$ is an implication relation.

pc_j are the decision variables for all pc_f-implicants that cover MSI_i. Below, we will refer to an mvTANT-implicant pc_j and not to a *variable pc_j corresponding to an mvTANT-implicant*, for short. We will keep this notation for all decision variables below.

t_{j_r} are the decision variables for all tail factors t_{j_r} from pc_j.

$t_{j_{r_n}}$ are the decision variables for all tail factors of na_f-implicants created for tail factor t_{j_r} from pc_j.

Example of such a decision function: $DF(MSI_i) =$

$$(pc_1 + pc_2 + pc_3) \quad \cdot$$
$$(pc_1 \Rightarrow (t_{1_1} \cdot t_{1_2})) \ (pc_2 \Rightarrow (t_{2_1} \cdot t_{2_2})) \ (pc_3 \Rightarrow t_{3_1}) \quad \cdot$$
$$(t_{1_1} \Rightarrow (t_{1_{1_1}} + t_{1_{1_2}})) \ (t_{1_2} \Rightarrow (t_{2_{1_1}} + t_{2_{1_2}} + t_{2_{1_3}})) \quad \cdot$$
$$(t_{2_1} \Rightarrow t_{2_{1_1}})) \quad (t_{2_2} \Rightarrow (\ t_{2_{2_1}} + t_{2_{2_2}})) \quad \cdot$$
$$(t_{3_1} \Rightarrow (\ t_{3_{1_1}} + t_{3_{1_2}}\ ))$$

The *first component* is a sum of variables for three pc_f-implicants: pc_1, pc_2, and pc_3, created from MSI_i. The *second component* is a product of three implication equations for tail factors from the three principal mvTANT-implicants. The implication $(pc_1 \Rightarrow (t_{1_1} \cdot t_{1_2}))$ means: if you select pc_f-implicant pc_1 then you have to select tail factor t_{1_1}, or tail factor $\cdot t_{1_2}$. The *third component* is an implication equation for the tail factors from the first pc_f-implicant pc_1. For t_1 it includes variables $t_{1_{1_1}}$ and $t_{1_{1_2}}$, which correspond to all tail factors in na_f-implicants generated from tail factor t_{1_1} of pc_f-implicant pc_1. The implication means: if you selected (in pc_1), the tail factor t_{1_1}, then you have to select one of tail factors: $t_{1_{1_1}}$ or $t_{1_{1_2}}$, in na_f-implicants created from pc_1. The *fourth component* is an implication equation for the tail factors from the second pc_f-implicant. The *fifth component* is an implication equation for the tail factors from the third pc_f-implicant.

The decision function DF is created as a Boolean product of all such elementary decision functions. It is next normalized to a standard form of *a product of sums of products of literals*, by using the implication removal transformation: $a \rightarrow b$ **is reduced to** $\overline{a} + b$. There are several methods in the literature (both exact and approximate) to minimize such decision functions DF; *binate covering, Boolean Equations, Binary Decision Diagrams, integer programming, data flow, cube calculus, AND-OR tree search, tree search,* and many other [29,30,14,12,8,7,6].

7 Conclusions and future work

We presented a new kind of circuit: Multiple-Valued-Input TANT Network, mvTANT for short. Similarly to Multiple-Valued Input SOPs (mvSOPs) and Multiple-Valued-Input Exclusive Sums-of-Products (mvESOPs), such circuits may find applications to the minimization of binary regular arrays, cellular FPGAs, and gate arrays. We presented also the fundamentals of the minimization of such circuits.

In a forthcoming paper, this approach will be extended to exact and approximate minimization of multiple-output incompletely specified functions. It has been also generalized for the case of circuits with full decoders and function generators: in full decoders we use 2^k decoder outputs for k decoder input variables; in function generators there are more than 2^k decoder output functions used. Such decoder functions are used to create literals for some, or all, mv variables. The problems of finding variables' groupings and polarities will be also a subject of the forthcoming paper.

Since the solution space of the mvTANT minimization problem is very large, larger than the space of the classical mvSOP minimization problem [18,20] only an approximate variant is reasonable from the practical point of view. But, similarly to Espresso-Exact [18,19,20] and the approach from [14], the creation of **exact algorithms** may be of some use to understand better the problem, and can also help in the creation of an efficient approximate algorithm for the same problem.

8 References

[1] Ciesielski, M. J., Yang, S., and M. A. Perkowski, "Multiple-Valued Minimization Based on Graph Coloring," *Proc. ICCD '89*, Oct. 1989.

[2] Ciesielski, M. J., and S. Yang, "PLADE: A Two-Stage PLA Decomposition," *IEEE Trans. on CAD*, Vol. 11, No. 8., pp. 943-954, Aug. 1992.

[3] Chakrabarti, K.K., Choudhury, A.K., and M.S. Basu, "Complementary Function Approach to the Synthesis of Three-Level NAND Network", *IEEE Trans. Comput.,* Vol. C-19, pp. 509-514, June 1970.

[4] Choudhury, A.K., Chakrabarti, K.K., and D. Sharma, "Some Studies on the Problem of Three-level NAND Network Synthesis", *Int. Journal of Control,* Vol. 6., No. 6., pp. 547-572, 1967.

[5] Frackowiak, J., "The Minimization of Hazardless TANT Networks", *IEEE Trans. Comp.,* Vol. C-21., No. 10, pp. 1099-1108, Oct. 1972.

[6] Gimpel, J.F., "The Minimization of TANT Networks", *IEEE TEC,* Vol. EC-16, pp. 18-38, Febr. 1967.

[7] Koh, K.S., "A Minimization Technique for TANT Networks", *IEEE Trans. Comp.,* January 1971, pp. 105-107.

[8] Kulpa, Z., "Synthesis of Quasi-Minimal Logic Circuits of Many Variables with use of NAND and NOR gates", *M.Sc. Thesis,* Institute of Automatic Control, Warsaw Technical University, 1970.

[9] Lee, H-P.S., "An Algorithm for Minimal TANT Network Generation", *IEEE Trans. Comp.* Vol. C-27, No. 12, Dec. 1978, pp. 1202-1206.

[10] Motorola, "MPA10XX Field Programmable Gate Array", *Product Brief,* 9/27/93.

[11] Murgai, R., Shenoy, N., Brayton, R. K., and A. L. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *Proc. ICCAD-91,* pp. 564 - 567, Nov. 1991.

[12] Perkowski, M.A., "Synthesis of Multioutput Three Level NAND Networks", *Proc. of the Seminar on Computer Aided Design,* Budapest, 3-5 Nov. 1976, pp. 238-265.

[13] Perkowski, M. A., Chrzanowska-Jeske, M., and T. Shah, "Minimization of Multioutput TANT Networks for Unlimited Fan-In Network Model," *Proc. ICCD'90*, pp. 360 - 363, Boston, MA, September 1990.

[14] Perkowski, M.A., and M. Chrzanowska-Jeske, "An Exact Algorithm to Minimize Mixed-Radix Exclusive Sums of Products for Incompletely Specifed Boolean Functions," *Proc. ISCAS'90,* pp. 1652 - 1655, New Orleans, May 1-3, 1990.

[15] Perkowski, M. A., "The Generalized Orthonormal Expansion of Functions with Multiple-Valued Inputs and Some of its Applications," *Proc. ISMVL'92,* pp. 442 - 450, Sendai, Japan, May 27-29, 1992.

[16] Perkowski, M. A., "A Fundamental Theorem for EXOR Circuits," *Proc. of IFIP W.G. 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design,"* Hamburg, Germany, Sept. 16-17, pp. 52 - 60, 1993.

[17] Perkowski, M. A., Sarabi, A., and F. R. Beyl, "Universal XOR Canonical Forms of Switching Functions," *ibid*, pp. 27 - 32, 1993.

[18] Rudell, R. L., "Multiple-Valued Logic Minimization for PLA Synthesis", *M.S. Report, June 5, 1986. University of California, Berkeley*, California 94720.

[19] Rudell, R. L., and A.L. Sangiovanni-Vincentelli, "Exact Minimization of Multiple-Valued Functions for PLA Optimization", *Proc. ICCAD'86,* Nov. 1986.

[20] Rudell, R. L., and A.L. Sangiovanni-Vincentelli, "Multiple-Valued Minimization for PLA Optimization," *Proc. ISMVL '87,* pp. 198-208, May 26-28, Boston, MA, 1987.

[21] Schaefer, I., Perkowski, M. A., and H. Wu, "Multilevel Logic Synthesis for Cellular FPGAs Based on Orthogonal Expansions," *as [16]*, pp. 42 - 51, 1993.

[22] Sarabi, A., Song, N., Chrzanowska-Jeske, M., and M. A. Perkowski, "Comprehensive Logic and Layout Synthesis for Cellular FPGAs," *Proc DAC '94.*

[23] T. Sasao, "MACDAS: Multi-Level AND-OR Circuit Synthesis Using Two-Variable Function Generators", *Proc. DAC '86,* Las Vegas, pp. 86-93, June 1986.

[24] Sasao, T., and M. Higashida, "A Design Method for Three-Level Logic Circuits", (in Japanese). *The Technical Papers of IEICE Japan,* VLD88-84, Dec. 1988.

[25] Sasao, T., "On the Complexity of Three-Level

Logic Circuits", *Proc. Intern. Workshop on Logic Synthesis,* MCNC, ACM SIGDA, May 23-26 1989, paper 10.2.

[**26**] Sasao, T., "Application of Multiple-Valued Logic to a Serial Decomposition of PLAs," *Proc. ISMVL '89,* 1989.

[**27**] Signetics, *PLD Data Manual,* Signetics' Approach to Logic Flexibility for the '80's", 1986.

[**28**] Song, N., and M. A. Perkowski, "EXORCISM-MV-2: Minimization of Exclusive Sum of Products Expressions for Multiple-Valued Input Incompletely Specified Functions," *Proc. ISMVL '93,* pp. 132 - 137, Sacramento, CA, May 24-27, 1993.

[**29**] Vink, H.A., Van Dolder B., and J. Al, "Reduction of CC-tables Using Multiple Implication", *IEEE Trans. Comp.,* Vol. C-27, No. 10, Oct. 1978.

[**30**] Vink, H.A., "Minimal TANT Networks of Functions with Don't Care's and Some Complemented Input Variables", *IEEE Trans. Comp.,* Vol. C-27, No. 11., Nov. 1978.