MINIMIZATION OF MULTIOUTPUT TANT NETWORKS FOR UNLIMITED FAN-IN NETWORK MODEL

Marek A. Perkowski, Malgorzata Chrzanowska-Jeske, and Tushar Shah

Department of Electrical Engineering, Portland State University,
P.O.Box 751, Portland, Oregon 97207,
tel: (503) 725-3806x23

ABSTRACT

The paper describes a program for the minimization of multi-output three-level Boolean networks from NAND gates of unlimited fan-in. This model suits very well several recently introduced PLD devices, for instance the "NAND Foldback Architecture" PLHS501 and PLHS502 chips from Signetics Corp. In our model the function is multi-output, and it includes don't cares. The presented algorithm is fast and creates good-quality approximate solutions, its efficiency increases with the percentage of don't cares.

1. THE NAND FOLDBACK PLD ARCHITECTURES

The PLD devices have been recently introduced that allow to design systems with lower component counts and higher operating speeds [32]. For most of Boolean functions, these new *PML (Programmable Macro Logic)* architectures can improve gate utilization and timing, as compared to the previous generations of PLD devices. The PLHS501 from Signetics Corporation provides NAND-folded array design - a sea of NAND gates with a multi-point connectivity matrix. The single array allows for the full connectivity of inputs, outputs and embedded macro functions, making possible full gate utilization and embedded multi-level block designs. It has 72 NAND gates with 72 inputs each, which means that an unlimited fan-in can be assumed by the synthesis algorithm.

The PLHS501 device has 24 dedicated inputs, 8 I/O buffers, 8 EXOR buffers, 4 Active-Low buffers, and 4 Active-High buffers. It can be observed that in this device a single-level logic function has a very short path through the device, 18nS max [31]. Additional levels incur only one NAND foldback delay per level, which is 8 nS. The two-level circuit has then 18 + 8 = 26 nS delay only, which is much better than in the previous generation of devices which stipulated that the logic signal must pass through I/O buffers after one or two levels of logic are performed [31].

2. THE TLN NETWORK MODEL

When one realizes the circuits composed of several connected blocks (as, for instance, the iterative circuits) in PLHS501, some inputs to the blocks come directly from the chip's inputs (the so called *direct inputs).* The PLHS501 has both negative and positive inputs, which can be used as the direct inputs. The feedback signals coming from the internal NAND gates of PLHS501 have no negations. Therefore, while designing a single logic block to be realized among other blocks in a PLHS501 device, these signals (like for instance the carry signals of an iterative circuit) have to be treated in logic minimization procedure as input variables available in only affirmative form [26]. In [19,26] we presented a circuit model for incompletely specified Boolean functions where any subset of input variables is available in both forms (in particular cases: all variables or none of them). It results from the above discussion that the variable availability model from [19,26] is the most general one for PLHS501.

The advantage of *Three level networks (TLN)* is that the TLN design for function f can never have more gates than the corresponding PLA. Usually, the TLN design has two advantages: it has much less gates and is faster. These are the main reasons why there has been recently an increased interest in such networks. Three level networks are important from the speed optimization point of view, since three levels is the minimum number of logic levels necessary to realize an arbitrary logic function from inclusive gates (even PLA has three levels, including inverters). The algorithms for hazardless synthesis of TLN networks which have been also proposed [25] makes them useful for the design of asynchronous state machines. Moreover, one can easily expand this model to take into account the "fast" (arriving quickly) and "slow" (arriving late) combinational input signals of a state machine, by assigning the slow inputs to the second level of the TLN circuit [25]. In the TLN model, the one-level (such as OR of negated variables), and two-level functions (such as a positive unate function) need only one or two levels of logic, respectively, which makes their realizations faster than in PLAs.

TLN networks [26] are the generalizations of the *Three level And-Not Networks with True inputs (TANT)* of McCluskey and Gimpel [15,9]. Several algorithms to minimize TANT networks have been published [4,5,15,14,11,33,34], and some algorithms have been implemented as computer programs [9,12,8,18,26]. A closely related topic of negative gate network minimization is presented in [19,20].

Other advantage of TLN networks is that they can be designed in VLSI: using one, two or three NAND-arrays (some of them possibly folded), or using a NAND array and a standard NAND/NAND PLA-like pair of arrays. Each NAND array is of better density than the NAND/NAND array of the two-level designs. The routing among the arrays is simplified. In the case of TANT networks (as well as for the "single-polarity" TLN networks where each variable is available as either positive or complemented but not in both [24]), the number of input columns of an array is reduced by half with respect to a PLA (only one column exists for each input variable). Other advantage of such layout style is the ease of adding EXOR logic to the NAND array for improved testability.

3. BASIC NOTIONS OF APPROXIMATE MULTIOUTPUT TANT MINIMIZATION

The Algorithm 5.1, presented in section 5 of this paper deals with a multioutput, incompletely specified function in which all input variables are available in affirmative (positive) form. The cost to be minimized is the gate cost. This cost definition speeds-up this algorithm with respect to the algorithms which use gate/input costs [26,24] and it is sufficient for the fan-in model of PLHS501-like circuits. The improved speed of this algorithm with respect to the previous algorithms was achieved due to the assumption of unlimited fan-in of all NAND gates, and due to the speed of modern two-level logic synthesizers.

The fast approximate algorithm for the general, all-variable availability model is quite complex. Its basic notions and partial algorithms have been presented in [26,19,20], and will appear in the forthcoming papers [23,24,25]. Therefore, because of the limited space, only the case of the multi-output incompletely specified functions with only positive variables available will be discussed in this paper. One of the methods to expand the model presented here is to introduce a new variable symbol for a negation of each variable available in the complemented form [34]. In the most difficult case when all variables are available in both forms, this method doubles the number of input variables, but does not require otherwise any other modifications to the algorithms presented in the sequel. Two other methods for the general case are presented in [26,23,24]: the complex but more efficient, and the exact method.

The source data, multioutput Boolean function $f = (f^1, \ldots, f^i, \ldots, f^m)$ is given as a *multioutput ON array* (specifying ON-cubes of all single *component (output) functions* f^i, i = 1, ..., m), and a *multioutput OFF array* (specifying OFF-cubes of all these functions). Each cube has $n+m$ bits, the first n bits ($b_r \in \{0, 1, X, \varepsilon\}$) is the *input part*, the last m bits ($b_i \in \{0, 1\}$) is the *output part* [8]. By ON(f^i) we will denote the set of all such cubes j for which ON[j,n+i] = 1. Similar notation will be used for other sets of multioutput cubes and subfunctions. The cubes need not to be the minterms. The program assumes that all not specified values are don't cares. Such representation has advantages for strongly unspecified functions.

4. CATEGORIES OF IMPLICANTS FOR TANT NETWORK MINIMIZATION

The following example will illustrate how TANT optimization leads to network reduction.

Example 4.1. The function f = $\sum$(0,1,2,4,5,6,12,14) can be minimized as the following Boolean expression: $f = \bar{a}\,\bar{c} + \bar{a}\,\bar{d} + b\,\bar{d}$. Its NAND-based PLA realization (so called **PLA-expression**) has gate cost 7. The corresponding optimal expression for TANT network is $f = \bar{a}\,\overline{cd} + b\,\bar{d}$, which has one gate less than the previous realization.

One can observe from the above example, that in TANT networks the first level realizes a logical sum, the second realizes a product and the third one the negation of the variables' product. TANT network minimization problem consists of finding the Boolean expression that minimizes the total cost. It means that the synthesis method should *minimize simultaneously* the second and third levels.

Definition 4.1. The **positive core (core,** for short) of a product of literals is the product of those literals which are positive.

Example 4.2. The positive core of $ab\bar{c}\,\bar{d}$ is ab.

The core is a *cube,* so that definitions of *cube calculus operations* [1,2,6,10,27,28], such as *sharp* (#), *absorption,* and *intersection* are applicable to it.

Definition 4.2. A **permissible expression** is a Boolean expression of form $P = H\,\overline{T_1}\,\overline{T_2} \cdots \overline{T_n}$ where both H and T_i are positive cores. H is called the **head of permissible expression** and each T_i is called a **tail core** while each $\overline{T_i}$ is called a **tail factor.**

Definition 4.3. A **permissible implicant (PP-implicant) of function** f is a permissible expression which implies f (i.e. covers a subset of ON cubes of f).

Example 4.3. A Boolean expression $b\,\bar{d}$ is the *prime implicant (prime,* for short) of the function from Example 4.1, and $b\,\bar{d}$, $b\,\overline{bd}$, $b\,\overline{ad}\,\bar{c}$, $b\,\overline{ad}\,\overline{cd}$ are some of PP-implicants of this function.

Definition 4.4. A head of a PP-implicant of function f is called the **second level group.** A tail core of a PP-implicant is called the **third level group.**

Example 4.4. For the TANT network of the function f from Example 4.1, products b, and 1 are the second level groups, while products a, b, and cd are the third level groups. 1 is the head of the PP-implicant $\bar{a}\,\bar{c}$.

Definition 4.5. A **permissible realization** for function f is the inclusive sum of the set of PP-implicants which cover all ON cubes of the function. An **optimal permissible realiza-**

tion for function f, denoted by $OPR(f)$, is such a permissible realization that its corresponding TANT network has the minimum total cost.

Definition 4.6. A **prime permissible implicant,** pp_f-implicant for short, is a permissible implicant that:

- is not included in a prime implicant,
- if a tail factor is removed from it then the resulting expression is no longer a PP-implicant.

The set of all pp_f-implicants is denoted by PP_f.

Example 4.5. For function f from Example 4.1:
$PP_f = \{\bar{a}\,\bar{c},\ \bar{a}\,\bar{d},\ \bar{a}\,\overline{cd},\ b\,\bar{d},\ b\,\bar{a}\,\overline{cd},\ b\,\overline{ad}\,\overline{cd},$
$b\,\overline{bd},\ b\,\overline{ab}\,\overline{cd},\ b\overline{a}\,\overline{bcd},\ b\,\overline{abd}\,\overline{cd},\ b\,\overline{ad}\,\overline{bcd},\ b\,\overline{abd}\,\overline{bcd}\}$.

Definition 4.7. A **principal PP-implicant,** pc_f-implicant for short, is such a pp_f-implicant that its tail cores do not contain variables from its head. The set of all pc_f-implicants is denoted by PC_f.

Example 4.6. For function f from Example 4.1:
$PC_f = \{\bar{a}\,\bar{c},\ \bar{a}\,\bar{d},\ a\,\overline{cd},\ b\,\bar{d},\ b\,\bar{a}\,\overline{cd},\ b\,\overline{ad}\,\overline{cd}\}$.

Definition 4.8. A **maximal** pc_f**-implicant,** mp_f-implicant for short, is such a pc_f-implicant that is not included in other pc_f-implicants. The set of all mp_f-implicants is denoted by M_f.

The **maximal implicant of a multioutput function** is the maximal implicant of its component function f^i, or the maximal implicant of a product of component functions.

Theorem 4.1. The tail cores of a mp_f-implicant are included in all the tail cores of pc_f-implicants included in this mp_f-implicant.

Example 4.7. For function f from Example 4.1: $M_f = \{\bar{a}\,\overline{cd},\ b\,\overline{ad}\,\overline{cd}\}$.
The tail cores ad and cd of a mp_f-implicant $b\,\overline{ad}\,\overline{cd}$ are included in the tail core d of the pc_f-implicant $b\,\bar{d}$. They are also included in the tail cores a and cd of $b\,\bar{a}\,\overline{cd}$.

Definition 4.9. An **augmented** pp_f**-implicant,** ap_f**-implicant** 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 4.8. For function f from Example 4.1:
$AP_f = \{b\,\overline{bd},\ b\,\overline{ab}\,\overline{cd},\ b\,\overline{a}\,\overline{bcd},\ b\,\overline{ab}\,\overline{bcd},\ b\,\overline{abd}\,\overline{cd},\ b\,\overline{ad}\,\overline{bcd},\ b\,\overline{abd}\,\overline{bcd}\}$.

Definition 4.10. A **necessary** ap_f**-implicant,** na_f**-implicant** for short, is such a 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. An **unnecessary** ap_f-implicant is the ap_f-implicant which is not an na_f-implicant. It is called an una_f**-implicant.**

Theorem 4.2. If the una_f-implicant is not selected to an exact $OPR(f)$ then at least one optimal solution is not lost.

Example 4.9. For function f from Example 4.1: $NA_f = \phi$.

Definition 4.11. A **necessary** pp_f**-implicant,** np_f**-implicant** for short, is such a pp_f-implicant that is not a una_f-implicant. The set of all np_f-implicants is denoted by N_f.
From Definitions 4.6, 4.7, 4.9, 4.10 and 4.11 one can conclude that $N_f = PC_f \cup NA_f$.

Example 4.10. For function f from Example 4.1: $N_f = PC_f$.
The next theorem results directly from the Boolean Rule: $A\,\bar{B} = A\,\overline{AB}$ and the definitions of pc_f-implicants and na_f-implicants.

Theorem 4.3. Every ap_f-implicant can be generated from a PP-implicant by addition of certain variables contained in its head to a subset of its tail cores.

Theorem 4.4. The positive cores of all prime implicants of function f are sufficient as the heads of the pp_f-implicants.

Example 4.11. The prime implicants of function f from Example 4.1 are: $\{\bar{a}\,\bar{c},\ \bar{a}\,\bar{d},\ b\,\bar{d}\}$. The positive cores of these implicants are $\{1, b\}$ which are the heads of the pp_f-implicants from Example 4.5.

Theorem 4.5. The positive cores of all prime implicants of function $\bar{f}$ (the *complement of function f*) are sufficient as tail cores of the mp_f-implicants of f.

Example 4.12. Prime implicants of the complement of function f from Example 4.1 are $\{a\,\bar{b},\ ad,\ cd\}$. The positive cores of these implicants are $\{a,\ ad,\ cd\}$ which are tail cores of the mp_f-implicants from Example 4.7.

Definition 4.12. An **essential prime implicant of single-output function** f is the prime that is not entirely covered by other primes of f. An **essential maximal implicant of a component function** f^i is the maximal implicant that is not entirely covered by other maximal implicants of this function. An **essential implicant of a multioutput function** f is the essential implicant of at least one of its component functions f^i.

Definition 4.13. A **positive implicant of** f is the product of positive variables that is an implicant of f. A **positive prime implicant** is the prime implicant that is composed of only positive variables. An **essential positive implicant** is the essential prime implicant that is a positive implicant as well.

Definition 4.14. An **essential tail core** is the single-variable tail core from an essential maximal implicant of positive core 1.

Definition 4.15. An **essential necessary implicant** is either the essential maximal implicant whose all tail cores are essential, or the positive essential implicant.

Let us observe that in order to minimize the second level of TANT network the essential maximal implicants should be selected. First an essential prime is found and in the next phase several implicants are generated with the positive core of this implicant as their head, and one or more of them are included to a solution. Then, in a sense, only its head is essential. We will call it an **essential head.** On the contrary, the essential necessary implicant of a single-output function or component function f^i should be directly selected to the solution because no other better necessary implicant can be created from it. Let us also observe that, in function f, an essential positive implicant of a component function f^i can be useful as a tail core or an implicant in other component function f^j of the same function f, $j \neq i$. Therefore, an already selected group can be used free for other applications (this can lead to solutions of nonminimal input cost but this is not a concern of our circuit model).
When one wants to find the minimum solution for the multioutput function, maximal implicants must be generated that are maximal not only for each individual f^i but also for products of several f^i (so-called **multioutput maximal implicants).** This is done by finding products of maximal implicants of the same head generated for each f^i. If , for instance, there are maximal implicants m_1, m_2, and m_3 in functions f^1, f^2, and f^3, respectively, then one creates products: m_1 & m_2, m_1 & m_3, m_2 & m_3, m_1 & m_2 & m_3. These multioutput maximal implicants are used for realization in the second phase of the algorithm.

Definition 4.16. A **free tail core of** f is one that has been already selected as a positive implicant or as a tail core in one of the component functions f^i.
Let us observe that free tail cores don't increase the solution cost when taken as tail cores of new implicants selected to the cover.
By "sharping" function g from function f we understand the result of operation $f \# g$.

Definition 4.17. A **secondary essential prime** is the prime that becomes essential after sharping from set ON of an essential or secondary essential prime. A **secondary essential maximal implicant** is the maximal implicant that becomes essential after sharping from set ON of an essential or secondary essential maximal implicant.

Definition 4.18. A **minimal minterm** of cube HEAD is the cube obtained from HEAD by replacing all its bits X by bits 0.

Definition 4.19. The **simple extension** HEAD1 of core HEAD in f^i is the core that originates from HEAD by the removal of a single variable, under condition that the minimal minterm of HEAD1 is not included in $OFF(f^i)$.
(A core of k literals has from 0 to k simple extensions).

Definition 4.20. The **extension** of core HEAD is the simple extension or the extension of a simple extension. A **maximum extension** is the extension that has no further simple extension.
All implicants and tail cores are called **subfunctions.**

5. THE ALGORITHM FOR FINDING QUASIMINIMAL TANT SOLUTION

In order to find the optimum $OPR(f)$ from the set of the np_f-implicants, the *covering problem with changing costs* is used in [26]. Most other approaches use variants of the *covering-closure model* [33,34,9]. Since both those approaches are memory and time consuming for TLN networks, in this paper we will present a heuristic search algorithm that does not generate all the np_f-implicants, and solves the covering/closure problem only approximately and without creating the covering/closure table. It can, however, prove optimality for a class of Boolean functions that essentially includes the class of non-cyclic functions [28]. The basic idea of this algorithm is to find the essential and "good" subfunctions of some f^i and next use them, or their tail factors, in all component functions f^i in which they are useful.

The following theorems can be proven.

Theorem 5.1. *If a single-output function is composed of only essential implicants then* all its maximal implicants are essential as well.
The opposite is not true, as exemplified by the function $f = \sum(1,3,6,7,8,9,12,14)$. It has no essential primes since it is a cyclic function. It includes, however, three essential maximal implicants: $d\,\overline{bd}\,\overline{acd}$, $bc\,\overline{acd}$, and $a\,\overline{bd}\,\overline{ac}$, which together constitute the minimum second level cover of f. The above function is **cyclic,** but not **TANT-cyclic.** The function $f = \sum(0,1,5,6,7,10,14,15)$ is both cyclic and TANT-cyclic.

Theorem 5.2. *Algorithm 5.1 finds the* **minimum second level solution** for each single-output function that is composed from essential and secondary essential maximal implicants only. If additionally the tail cores are all essential then the solution is globally minimum. Even if the function has a TANT-cyclic component, the minimal second-level realization of the not TANT-cyclic component is found.
The minimum second level solution is one that has the minimum possible number of gates in the second level. It was found practically to be very close to the global minimum for many single-output functions of less than 10 variables [5,24].
It results from these theorems that without solving the covering problem the minimal second level solutions can be found for a wider class of functions in the TANT model than in the PLA model.

Algorithm 5.1.
Minimization of a multioutput TANT network.

1. Using sets of cubes ON and OFF, generate the set ESSENTIAL of all essential prime cubes of f (the efficient algorithm from [22] is used that generates the essential primes without generating the set of primes of f).
2. Using sets of cubes ON, OFF, and ESSENTIAL, generate the set ESSENTIAL2 of all secondary essential prime cubes of f (the efficient algorithm from [22] is used that generates the secondary essential primes without generating all primes).
3. ESSENTIAL := ESSENTIAL $\cup$ ESSENTIAL2. ON1 := ON.
4. Create set ESSENTIAL3 by partitioning set ESSENTIAL into groups of essential primes, by assigning to each group the essential primes with the same positive core.
5. Make set ESSENTIAL4 of essential necessary implicants by selecting from set ESSENTIAL3 the positive implicants and the implicant of positive core 1 being the single element of a group.
6. *Realization of essential tail cores and FREE_TAIL_CORES.*
   A. ESSENTIAL_TAIL_CORES := { the tail cores being negations of the tail factors from ESSENTIAL4 }.
   FREE_TAIL_CORES := { set of those implicants from ESSENTIAL3 that are positive }.
   B. For each tail core G from ESSENTIAL_TAIL_CORES $\cup$ FREE_TAIL_CORES, and all f^i, i = 1,...,m, do:
      a) if G is an implicant of f^i then realize it as an implicant of f^i (by the *realization* of implicant G in function f^i one means:
      SOLUTION(f^i) := SOLUTION(f^i) $\cup$ G, ON1(f^i) := ON(f^i) # G).

(SOLUTION(f^i) is the set of implicants selected for component function f^i. SOLUTION = (SOLUTION(f^1), ..., SOLUTION(f^m)).

b) if for a component function f^i there is: $ON1(f^i) \subseteq \overline{G}$ and $OFF(f^i) \cap \overline{G} = \phi$ then realize the entire f^i as $\overline{G}$. (steps a), b) can be performed since these tail cores G do not add gate cost when used).

7. *Realization of essential necessary implicants.* Sharp the essential necessary implicants found in step 5 from the function and put them to the solution sets of f^i in which they exist:
ON1 := ON1 # ESSENTIAL4, SOLUTION := ESSENTIAL4.

8. Create the set ESSENTIAL_HEADS from the positive cores of all the groups from the set ESSENTIAL3 — ESSENTIAL4.

9. ON2 := OFF, OFF2 := ON1. Find a single SOP of function $\overline{f}$.

10. Find set TAIL_CORES of positive cores of the primes from this SOP.

11. *Realization of essential maximal implicants.*
For each HEAD from ESSENTIAL_HEADS do:

A. for each component function f^i for which there is an implicant of head HEAD do:

a) create a maximal implicant MAX(f^i) of head HEAD, using the algorithm from [32]. This algorithm selects tail cores from the sets TAIL_CORES, ESSENTIAL_TAIL_CORES, and FREE_TAIL_CORES, in this order of priority.

b) Mark in SOLUTION3 the component functions f^i in which MAX (f^i) is an implicant: MAX (f^i) $\cap$ ON1(f^i) $\neq \phi$ *and* MAX (f^i) $\cap$ OFF(f^i) $= \phi$.

c) Sharp MAX(f^i) from all ON1(f^j) sets of all component functions f^j in which it is an implicant.

d) FREE_TAIL_CORES1 := {tail cores from MAX(f^i)}.

e) realize the cores from FREE_TAIL_CORES1 analogously as the essential tail cores were realized in step 6B.

f) FREE_TAIL_CORES := FREE_TAIL_CORES $\cup$ FREE_TAIL_CORES1.

12. *Checking the solution.* If ON1 = ϕ then do:

A. SOLUTION3 := SOLUTION $\cup$ ESSENTIAL_MAX is the minimal second-level solution,

B. go to 16.

13. *Realization of secondary essential maximal implicants.*

A. Using the algorithm from [24] and sets ON1, OFF create the set ESSENTIAL5 of secondary essential maximal implicants. New tail factors are realized as in 11A c), d), e).

B. ON1 := ON1 # ESSENTIAL5,
SOLUTION2 := SOLUTION $\cup$ ESSENTIAL5.

C. *Checking the solution.* If ON1 = ϕ then do:

a) SOLUTION3 := SOLUTION2 is the minimal second-level solution,

b) go to 16.
else ADDITIONAL_TAIL_CORES := ϕ.

14. ON3 := ON1. SOLUTION3 := ϕ ; cyclic part of solution

15. *Heuristic Search of maximal implicants.*

A. Find in ON3 the cube *cube*, that has firstly the minimum number of bits "1" in the input part of the cube and secondly the maximum number of bits "1" in the output part of the cube. If there are many such cubes, pick randomly any of them.

B. Find its positive core HEAD. HEADS := {HEAD}.

C. For each component function f^i in which the minimal minterm MINIMAL(HEAD) for the HEAD is not included in OFF(f^i), do:

a) Find the set HEADS(f^i) of all heads being the maximum extensions of HEAD. For each of those heads, mark the functions f^i in which it exist.

b) HEADS := HEADS $\cup$ HEADS(f^i).

D. For all heads HEAD1 from set HEADS do:

a) Generate the maximal implicant MAX of head HEAD1 and negations of some tail cores as the tail factors [26]: the tail cores from sets ESSENTIAL_TAIL_CORES, and FREE_TAIL_CORES have the priority in the tail core selection process, the cores from TAIL_CORES, and ADDITIONAL_TAIL_CORES are selected in the next order.

b) SOLUTION3 := SOLUTION3 $\cup$ MAX.

c) Execute for MAX, ON3, and SOLUTION3 the algorithm described in steps 11A b) - e) for MAX(f^i), ON1, and SOLUTION.

E. If ON3 $\neq \phi$ then go to A.

16. SOLUTION4 := SOLUTION2 $\cup$ SOLUTION3.
If the function is single-output and all implicants in set SOLUTION4 have only the essential heads in the second level and the essential tail cores and positive essential implicants as the third level groups then print SOLUTION4 as the globally minimal one, and terminate.

17. *Improvement of the third level for the second level cover realized in steps 6 - 15.*

A. For each maximal implicant from SOLUTION4 create a set of possible tail cores from TAIL_CORES with which this implicant can be realized. These are the tail cores of the implicants from SOLUTION3, and their augmented cores.

B. Create a covering table with maximal implicants as columns and tail cores as rows.

C. Solve the covering problem to minimize the number of tail cores. Give priority to the cores in this order: ESSENTIAL_TAIL_CORES, FREE_TAIL_CORES, ADDITIONAL_TAIL_CORES. Additionally, the evaluation of the minimum bound on the third level is found at this stage [24,19].

18. *Iterative improvement of the TANT-cyclic part of the solution without reshaping the ON and OFF sets.*

A. ADDITIONAL_TAIL_CORES := { set of tail cores created from the rated cores of the "best" implicants from all solutions found until now, and the cores being their intersections }. (For instance, cores *abcd* and *acde* produce a new core *acd*).

B. Repeat three times the steps 14 - 18A. In step 15 Ca), the multioutput maximal implicants are created, as explained in section 4. Also, instead of the maximum extensions of heads, the extensions that minimize the "predicted cost" [26] of the created necessary implicants are taken (the search involves now not only the maximal implicants but also the necessary implicants).

19. If no improvement of solution cost was achieved in step 18
then print the SOLUTION4, the global minimum bound, and terminate.

20. *Iterative improvement of the solution with reshaping the ON and OFF sets* and recalculating implicants.

A. reshape cubes in sets ON and OFF,

B. ADDITIONAL_TAIL_CORES := ADDITIONAL_TAIL_CORES $\cup$ { set of positive cores of prime implicants of a SOP of $\overline{f}$ }.

C. ON3 := ON, SOLUTION2 := ϕ, SOLUTION3 := ϕ.

D. Execute steps 14-19, using in step 14 the set ON instead of the set ON1.

E. If no improvement of solution cost was achieved in step D
then print the SOLUTION4, the global minimum bound, and terminate.
Else go to 20.

Example 5.1. For function $f = \sum(0,2,4,5,6,7,12,13,14)$ the essential primes of core b are $b\,\overline{c}$, $b\,\overline{a}$ and $b\,\overline{d}$. The essential primes of core 1 is $\overline{a}\,\overline{d}$. After sharping the essential necessary implicant $\overline{a}\,\overline{d}$ (it is necessary as being a single essential prime of head 1), the only head remaining is b. Head b overlaps the positive core acd of prime implicant acd of $\overline{f}$. The essential maximal implicant $b\,acd$ is created (the implicants like $b\,\overline{c}$, $b\,\overline{a}$, and $b\,\overline{d}$ of head b are not maximal). After sharping it from set ON, the set ON becomes empty. Therefore, the solution obtained, $f = \overline{a}\,\overline{d} + b\,\overline{acd}$, has the minimum number of gates in the second level. The tail cores a, d are essential, since they come from an essential prime implicant of head 1. The tail core acd is also essential, since at least one tail core, other than c and d, is needed to realize the essential prime $b\overline{c}$, which is included in the maximal implicant of head b. The third level cover is then minimal as well. The solution is minimal on both levels, and since the function is single-output it is globally minimal.

Example 5.2. For function f = $\sum(0,1,3,4,5,6,9,10,11,12,14,15)$, the essential primes are: $\overline{a}\,\overline{c}$, $a\,c$, $b\,\overline{d}$, $d\,\overline{b}$. The essential necessary implicants are: $\overline{a}\,\overline{c}$ and $a\,c$. Tail cores a and c are essential as coming from an essential prime of head 1 being a single prime of this head. Tail core ac is essential because it is an essential positive prime. Set ESSENTIAL_TAIL_CORES = {a,c,ac}. The essential necessary implicants are sharped from ON and put to the solution. After this sharping, the prime implicants of function $\overline{f}$ are: $\overline{b}\,\overline{d}$ and $b\,d$. The positive core of the first one is 1 and is useless, so that TAIL_CORES = {bd}. After realization of $\overline{a}\,\overline{c}$ and $a\,c$ the set HEADS = {b, d}. Essential maximal implicants $b\,\overline{bd}$ and $d\,\overline{bd}$ are created and sharped from ON. ON = ϕ in step 12, so that the solution $f = \overline{a}\,\overline{c} + a\,c + b\,\overline{bd} + d\,\overline{bd}$ is minimal on the second level. It is also minimal on the third level, since the only way to realize primes $b\,\overline{d}$ and $d\,\overline{b}$ with a single group in the third level is to use the augmented group bd. Therefore, it is also globally minimal.

Example 5.3. Given is function f(a,b,c,d) = (f^1, f^2, f^3, f^4, f^5), where:
$f^1 = \sum(0,7,8,9\ (10,11,12,13,14,15))$, $f^2 = \sum(0,4,5,6\ (10,11,12,13,14,15))$,
$f^3 = \sum(2,3,6,9\ (10,11,12,13,14,15))$, $f^4 = \sum(1,3,5,8\ (10,11,12,13,14,15))$,
$f^5 = \sum(1,2,4,7\ (10,11,12,13,14,15))$.
(This is a converter from code "8421" to code "2 out of 5"). The essential primes of f^1 are: $\overline{b}\,\overline{c}\,\overline{d}$, bcd, a. The essential primes of f^2 are: $\overline{a}\,\overline{c}\,\overline{d}$, $b\overline{c}$, $\overline{bd}$. The essential primes of f^3 are: ad, $c\overline{d}$, $c\overline{b}$. The essential primes of f^4 are: $a\,\overline{d}$, $\overline{a}\,\overline{c}d$, $\overline{a}\,\overline{b}d$. The essential primes of f^5 are: $b\,\overline{c}\,\overline{d}$, $\overline{a}\,\overline{b}\,\overline{c}\,d$, bcd, $\overline{b}\,c\,\overline{d}$. Hence the essential necessary implicants are: for f^1: $\overline{b}\,\overline{c}\,\overline{d}$, bcd, a; for f^2: $\overline{a}\,\overline{c}\,\overline{d}$; for f^3: ad; for f^4: none; for f^5: bcd. They are put to respective ESSENTIAL4 sets of each function. The negations of their tail factors are: b, c, d, a. The essential positive prime implicants are: {a, ad, bcd}. The sum of these two sets: {a, ad, bcd, b, c, d, a}, becomes then the set of essential tail cores. Necessary implicants from ESSENTIAL4 are sharped from sets ON (step 7). At this point, ON(f^1) = ϕ, function f^1 has the minimal realization, as being a sum of the essential necessary implicants: $f^1 = \overline{b}\,\overline{c}\,\overline{d} + bcd + a$. In function f^2, the only head remaining after sharping of essential prime $\overline{a}\,\overline{c}\,\overline{d}$ from ON(f^2) is b. A single essential maximal implicant is $b\,\overline{cd}$. (Since bcd is an essential tail core, implicant $b\,\overline{bcd}$ is created for f^2 instead of $\overline{bcd}$). There are no more cubes in ON(f^2), so the minimal second level solution for this function has been found: $f^2 = \overline{a}\,\overline{c}\,\overline{d} + b\,\overline{bcd}$. In function f^3, after sharping the essential necessary implicant ad, the only head remaining is c. The essential maximal implicant $c\,\overline{bcd}$ is generated in step 11 since bcd

is the essential tail core: $f^3 = ad + c\,\overline{bcd}$. Similarly, the function f^4 is entirely created from essential maximal implicants: $f^4 = a\,\overline{ad} + d\,\overline{bcd}$. Function $f^5 = b\,\overline{c}\,\overline{d} + \overline{a}\,\overline{b}\,\overline{c}d + bcd + \overline{b}\,c\,\overline{d}$ is composed of only essential maximal implicants. This way, since all selected essential maximal implicants are also composed only of the essential heads and the essential tail cores (step 16), the optimal solution for each separate component function was found in step 12, without creating the covering table to minimize the third level. Since the function is multioutput, the is no warranty that this solution is globally minimum, and the program iterates. In one of iterations the multioutput maximal implicant $\overline{a}\,\overline{b}\,\overline{c}\,\overline{d}$ is selected for realization in functions f^1 and f^2. Similarly, the positive implicant ad is found in f^3 and realized in f^1 and f^3. Implicant $a\,\overline{ad}$ is found in f^4 and realized in f^1 and f^4. Now: $f^1 = ad + a\overline{ad} + \overline{a}\,\overline{b}\,\overline{c}\,\overline{d}$, $f^2 = b\,\overline{bcd} + \overline{a}\,\overline{b}\,\overline{c}\,\overline{d}$, which improves the gate cost by two gates (other functions are as in the previous solution). The method, similarly to the classical covering/closure algorithms for multioutput functions, generates some non-maximal implicants for each component function. (For instance, the necessary implicant $a\,\overline{ad}$ used in f^1 is a maximal implicant of another component function, f^4. It is also the product of maximal implicants a and $a\,\overline{ad}$. The necessary implicant $\overline{a}\,\overline{b}\,\overline{c}\,\overline{d}$ of f^1 is the product of $\overline{b}\,\overline{c}\,\overline{d}$ from f^1 and $\overline{a}\,\overline{c}\,\overline{d}$ from f^2.) However, the alsgorithm is much more efficient than those from [8,9,33,34], since only the "best", most prospective implicants are generated, so that their number is essentially limited. No cost improvement was brought by the next iterations for this example.

6. CONCLUSIONS

Program TANT-PLD is written in FORTRAN 77 and uses several routines from TLN-MINI [26]. It has been tried on about 40 Boolean functions of not more than 14 inputs, and yielded always correct results. For most completely specified functions of less than 10 inputs the solutions were minimum in second level and for functions of less than 6 inputs about 15% of the solutions were globally optimum. The realized circuits required up to 68% (on the average 35%) less gates than the corresponding PLAs. The program can consider trade-offs among the solution cost and the processing speed by using various types of the source data. For instance: each ON-cube can be a minterm, a disjoint ON-cube [7], a minimal ON-cube [17,3], a prime cube [22], a subminimal implicant [22,17], or any other ON-cube. Similarly the OFF-cubes. By selecting respective types of cubes, the size of the function that can be handled is sacrificed for the prize of the improved cost of the solution.

7. REFERENCES.

[1] Brayton, R.K., Hachtel, G.D., McMullen, C.T., and A.L. Sangiovanni-Vincentelli, *"Logic Minimization Algorithms for VLSI Synthesis"*, Boston, MA, Kluwer, 1984.

[2] Brayton, R.K., Camposano, R., De Micheli, G., Otten, R.H.J.M., and J. Van Eijndhoven, *"The Yorktown Silicon Compiler System"*, Chapter 7 in Gajski, D., (ed), Silicon Compilation, 1987.

[3] Ciesielski, M.J., Yang, S., and M.A. Perkowski: "Multiple-Valued Minimization Based on Graph Coloring", *Proc. International Conference on Computer Design: VLSI in Computers, ICCD'89*, October 1989.

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

[5] 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.

[6] Dietmayer, D.L. "Logic Design of Digital Systems", *Allyn and Bacon*, Boston, Mass, 1971.

[7] Falkowski, B., Perkowski, M.A., "An Algorithm for the Generation of Disjoint Cubes for Completely and Incompletely Specified Boolean Functions", *Accepted for publication in Intern. Journal of Electronics*, to be published in 1991.

[8] Frackowiak, J., "The minimization of hazardless TANT networks", *IEEE Trans. on Comp. Vol.*, C-21., No. 10, pp. 1099-1108, Oct. 1972.

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

[10] Hong, S.J., R.G. Cain, and D.L. Ostapko, "MINI: A heuristic approach for logic minimization", *IBM J. Res. Develop.*, Vol. 18, pp. 443 - 458, Sept. 1974.

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

[12] 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.

[13] Lawler, E.L., "An Approach to Multilevel Boolean Minimization", *Journal of ACM*, Vol. 11., No. 3., pp. 283-295, July 1964.

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

[15] McCluskey, E.J. Jr., "Introduction to the Theory of Switching Circuit", *McGraw-Hill*, 1965.

[16] Muller, D.E., "Complexity in Electronic Switching Circuits", *IRE Trans. Electr. Comp.*, EC-5, No. 1, pp. 15-19, 1956,

[17] Nguyen, L., Perkowski, M., and N.B. Goldstein, "PALMINI - Fast Boolean Minimizer for Personal Computers", *Proc. 24th Design Automation Conference*, June 28-July 1, 1987, Miami, Florida, Paper 33.3.

[18] Perkowski, M.A., "Synthesis of multioutput three level NAND networks", *Proc. of the Seminar on Computer Aided Design*, Budapest, 3-5 November 1976, pp.238-265.

[19] Perkowski, M.A., "Minimization of Two-Level Networks from Negative Gates", *Proc. Midwest 86 Conference on Circuits and Systems*, Lincoln, Nebraska, 1- 12 August 1986.

[20] Perkowski, M.A., "A Parallel Programming Approach to the Design of Two-Level Networks with Negative Gates", *International Workshop on Logic Synthesis, Research* Triangle Park, North Carolina, May 12 - 15, 1987.

[21] Perkowski, M.A., Ming, L.J., and A. Wieclawski, "An Expert System for Optimization of Multi-Level Logic", *IASTED Conference, Applied Simulation and Modeling, ASM '87*, Santa Barbara, CA, May 26 - 29, 1987.

[22] Perkowski, M.A., Wu, P., and K. Pirkl, "KUAI-EXACT: A New Approach for Multi-Valued Logic Minimization in VLSI Synthesis", *Proc. 1989 ISCAS - International* Symposium on Circuits and Systems, May 9-11, 1989.

[23] Perkowski, M.A., "Fast Algorithm for Synthesis of Three-Level NOR Networks with Constraints", *Report, Dept. EE.*, PSU, 1990.

[24] Perkowski, M.A., "Minimization of Limited-Level NOR/EXOR/NOT Networks with Constraints", *Report, Dept. EE.*, PSU, 1990.

[25] Perkowski, M.A., Chrzanowska-Jeske, M., Shah, T., "Synthesis of Fast Asynchronous State Machines in Programmable Logic", *Report, PSU, Department of Electrical Engineering*, 1990.

[26] Perkowski, M.A., Liu, J., "A Program for Exact Synthesis of Three Level NAND Networks", *Proc. ISCAS '90*, pp. 1118-1121, New Orleans, May 1-3, 1990.

[27] Rudell, R.L., and A.L. Sangiovanni-Vincentelli, "ESPRESSO-MV: algorithms for multiple-valued logic minimization, *Proc. IEEE Custom Integrated Circuits Conf.*, 1985.

[28] Rudell, R.L., and A.L. Sangiovanni-Vincentelli, "Exact minimization of multiple-valued functions for PLA optimization", *ICCAD-86*, Nov. 1986.

[29] Sasao, T., "MACDAS: Multi-level AND-OR circuit synthesis using two-variable function generators", *23-rd Design Automation Conference*, Las Vegas, pp. 86-93, June 1986.

[30] Sasao, T., "On the Complexity of Three-Level Logic Circuits", *Proc. International Workshop of Logic Synthesis, MCNC, ACM SIGDA*, May 23-26 1989, paper 10.2.

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

[32] Taghvaei, N., "Macros Aid High-Density Design Work", *Programmable Logic*, pp. 28-36, 1990.

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

[34] 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., November 1978.