VLSI DESIGN
1995, Vol. 3, Nos. 3-4, pp. 225-248
Reprints available directly from the publisher
Photocopying permitted by license only

General Decomposition and Its Use in Digital Circuit Synthesis

LECH JÓŹWIAK
Eindhoven University of Technology, Faculty of Electrical Engineering, P.O. Box 513, 5600 MB Eindhoven, the Netherlands

Modern microelectronic technology gives opportunities to build digital circuits of huge complexity and provides a wide diversity of logic building blocks. Although logic designers have been building circuits for many years, they have realized that advances in microelectronic technology are outstripping their abilities to make use of the created opportunities. In this paper, we present the fundamentals of a logic design methodology which meets the requirements of today's complex circuits and modern building blocks. The methodology is based on the theory of general full-decompositions which constitutes the theory of digital circuit structures at the highest abstraction level. The paper explains the theory and shows how it can be used for digital circuit synthesis. The decomposition methodology that is presented ensures "correctness by construction" and enables very effective and efficient post-factum validation. It makes possible extensive examination of the structural features of the required information processing in relation to a given set of objectives and constraints.

Key Words: *Decomposition, Digital Circuits, Logic Synthesis, Formal Methods, Automata Theory*

1. INTRODUCTION

Modern microelectronic technology gives opportunities to build digital circuits of huge complexity and provides a large variety of logic building blocks. A rapidly growing interest in programmable devices has also been observed, as a result of their very attractive characteristic features. However, programmable devices impose limitations on various circuit parameters due to input, output, functionality, memory, communication and speed constraints.

On the other hand, traditional logic design methods are not suitable for very complex circuits or implementations with constrained building blocks for the following main reasons: they are only devoted to some very special cases of possible implementation structures, they often leave unconsidered some important parameters that sufficiently influence the actual design objectives, they often fail to find global optima for large designs, they do not consider hard constraints, and they often do not consider correctness aspects in an appropriate manner.

Logic synthesis is typically performed without any relation to the target implementation structure and therefore, a technology mapping must be applied in order to map the synthesized logic network into a network of building blocks that can be implemented. Sometimes, the technology mapping algorithm is unable to construct any implementable network from a given initial network and it cannot guarantee an optimal solution, if the initial network is constructed without any regard to future implementation.

The bad practice of target-independent logic synthesis follows from the lack of appropriate modelling tools and synthesis methods for digital circuit structures. Traditional logic modelling tools model circuits in terms of functionally complete systems composed of a minimal number of some special structural elements (e.g. AND+OR+NOT, NAND, NOR, MUX or AND+EXOR) instead of modelling them in terms of all structural elements at the designer's disposal or, just generally, in terms of all possible subcircuits. For example, the commonly used Boolean algebra enables us to express all the possible Boolean functions but fails to model their implementation structures. Boolean algebra makes it possible to decompose functions exclusively into networks consisting of AND, OR, and NOT subfunctions, or into the equivalent NAND or NOR networks, while in general they can be decomposed into subfunctions of any kind.

Similarly, binary decision diagrams enable us to express the Boolean functions exclusively in the form of two-input multiplexer networks.

From the above we can conclude that the opportunities created by microelectronic technology cannot effectively be exploited. It has become extremely important to develop a new generation of methods which will effectively and efficiently deal with design complexity and the characteristic features of modern building blocks, enabling modelling and synthesis of all reasonable circuit structures and providing "correctness by construction", easy correctness verification and intelligent search algorithms for the effective and efficient exploration of the huge space of correct circuit structures.

In order to solve the problem a structural decomposition approach may be used. It consists of transforming a system into the structure of two or more cooperating sub-systems in such a way that the original system's behaviour is retained and certain constraints and objectives are satisfied.

The theoretical work in this field was started by Ashenhurst [4] and Curtis [10] for combinational circuits and by Hartmanis [16][17][18] for sequential circuits in the early 1960s. However, they over-simplified the actual problems and left some important parameters, that sufficiently influence the actual design objectives, unconsidered. For example, Hartmanis and others only partly considered decomposing the internal states of sequential machines. It was an incomplete solution, because the most important design parameters of a circuit for implementing a sequential machine (complexity, speed, testability etc.), or the possibility of implementing a machine with limited building blocks, depend on the whole implementation structure, i.e. on the distribution of the machine's inputs, outputs, state memory, functionality, and interconnections between the building blocks. So, from the practical viewpoint, decomposing the whole sequential or combinational process into an appropriate structure is necessary, i.e. **full-decomposition**. The theoretical works devoted to decomposition from the 1960s and 1970s should be considered as first steps on the way towards a complete decomposition theory. The first practical solutions were obtained in the 1980s (e.g. [1][5][21][27][35][45][46]).

The strongest stimulus for developing decomposition methods and tools came recently from the newest generation of multi-block programmable devices. In the case of fine granular multi-block FPGA's the hard constraints are active for virtually all non-trivial circuits. Implementation is impossible without decomposition.

In this paper, we will present the fundamentals of a decompositional design methodology which meets the requirements of today's complex circuits and modern microelectronic technology. The methodology is based on the theory of general full-decompositions which has been developed by us during the last few years and applied in a number of prototype decomposition tools [22]–[30]. General decomposition consists of transforming a sequential machine or a Boolean function into the structure of two or more cooperating partial machines in such a way that the original machine's behaviour is retained, and all the important structural attributes relating to inputs, outputs, state memory elements, functional units, and interconnections between the units, are appropriately considered at the same time.

Our previous publications focused on heuristic search algorithms for decomposition and presented some benchmark results [26]–[30]. The main aim of this paper is to present a general full-decomposition model and to show that this model, together with its theorem, constitute the theory of digital circuit structures at the highest abstraction level and form a sound base for the construction of decomposition algorithms. Since general full-decomposition includes various special decomposition types for sequential and combinational circuits, the general decomposition theorem will also be interpreted for some important special cases. Other aims of this paper are to explain how the model can be used for the digital circuit synthesis when focusing on correctness and optimization aspects.

2. BASIC DEFINITIONS

2.1. Sequential Machines and Realisations

A sequential (Mealy) machine M (Fig. 1) is an algebraic system defined by:

$M = (I,S,O,\delta,\lambda)$

where:

- I - a finite set of inputs,
- S - a finite non-empty set of internal states,
- O - a finite set of outputs,
- δ - the next state function $\delta: S \times I \rightarrow S$,
- λ - the output function $\lambda: S \times I \rightarrow O$.

Sometimes, the design requirements do not completely specify a machine. For example, certain input/state

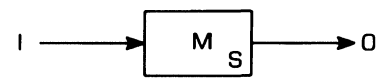

FIGURE 1 An original sequential machine.

combinations may never occur due to external constraints or due to realizing the machine in such a way that some of the input/state combinations of the realization are not used for implementing the original machine. Therefore, from the functional viewpoint, the designer does not care what will be the next-states or outputs for such combinations. Sometimes, outputs are sampled only at specified times: when they are not being sampled, they may be unspecified. If a certain input/state combination is followed by a general reset signal, the output for this combination should be specified, but the next-state need not. In all such situations one talks about so called "don't care" conditions. "Don't cares" are commonly denoted by "_". To account for "don't care" conditions, the sequential machine definition should be extended by slightly changing the definitions of functions δ and λ: δ: $S \times I \rightarrow S \cup \{-\}$ and λ: $S \times I \rightarrow O \cup \{-\}$ (for a single output machine) or $\lambda=[\lambda_j]$, λ_j: $S \times I \rightarrow O_j \cup \{-\} \wedge O=[O_j]$ (for a multiple output machine). A sequential machine without "don't cares" will be referred to as **completely specified** and with "don't cares" as **incompletely specified**.

A hardware implementation of a sequential machine is called a **sequential circuit**. If the output values are independent of the input values, i.e. λ: $S \rightarrow O$ then the sequential machine is called a **Moore machine**. If the output set O and output function λ are not defined or λ is an identity function of a Moore machine, then the sequential machine $M = (I, S, \delta)$ is called a **state machine**. A sequential machine with one state and a trivial next-state function is called a **combinational machine** (combinational function) and its hardware implementation is called a **combinational circuit**. Since the output values of a combinational machine are independent of the state values, i.e. λ: $I \rightarrow O$ and its trivial state behaviour is not important, the combinational machine is completely defined by $M = (I, O, \lambda)$.

Since a Moore machine, a state machine, and a combinational machine can be considered as special cases of a Mealy machine, formal considerations of this paper will be limited to the Mealy machines.

Machine $M' = (I',S',O',\delta',\lambda')$ **realizes (is realization of) machine** $M = (I,S,O,\delta,\lambda)$ if and only if the following relationships exist: Ψ: $I \rightarrow I'$ (a function), Φ: $S \rightarrow 2^{S'}$ (a function into nonvoid subsets of S'), Θ: $O' \rightarrow O$ (a surjective partial function), so that $\forall s \in S$, $s' \in \Phi(s)$ and $x \in I$: $\delta'(\Phi(s), \Psi(x)) \subseteq \Phi(\delta(s, x))$ and $\lambda(s, x) = \Theta(\lambda'(s', \Psi(x)))$.

If M' is a realization of M then, for all possible input sequences, the output sequences produced by machine M and its imitation M' are identical after renaming them. Realization in this sense will be referred to as the **realization of the output behaviour**. Since Φ is a function into nonvoid subsets of S' it is a **multi-state realization**.

Machine $M' = (I',S',O',\delta',\lambda')$ **realizes output behaviour** of machine $M = (I,S,O,\delta,\lambda)$ **when using single states** for realizing the states of M (M' is a single-state output behaviour realization of M) if and only if the following relationships exist:

Ψ: $I \rightarrow I'$ (a function), Φ: $S \rightarrow S'$ (a function), Θ: $O' \rightarrow O$ (a surjective partial function), so that $\forall s \in S$ $x \in I$: $\delta'(\Phi(s), \Psi(x)) = \Phi(\delta(s, x))$ and $\lambda(s, x) = \Theta(\lambda'(\Phi(s), \Psi(x)))$ (see Fig. 2).

In some cases, the internal state of the machine must be known outside. Therefore, the state behaviour realization is also important. Realization of the state and output behaviour is a special case of the output behaviour realization for which function Φ is a one-to-one function, so that Φ': $\Phi' = \Phi^{-1}$ exists.

Machine $M' = (I',S',O',\delta',\lambda')$ will realize the **state and output behaviour** of machine $M = (I,S,O,\delta,\lambda)$ when using single states for realizing the states of M if and only if the following relationships exist: Ψ: $I \rightarrow I'$ (a function), Φ': $S' \rightarrow S$ (a surjective partial function), Θ: $O' \rightarrow O$ (a surjective partial function), such that $\forall s' \in S'$ $x \in I$: $\delta(\Phi'(s'), x) = \Phi'(\delta'(s', \Psi(x)))$ and $\lambda(\Phi'(s'), x) = \Theta(\lambda'(s', \Psi(x))$ (see Fig. 3).

The sequential machine composed as a structure consisting of Ψ, M' and Θ (and $\Phi' = \Phi^{-1}$ for the state behaviour realization) (Fig. 2 and 3) will be referred as

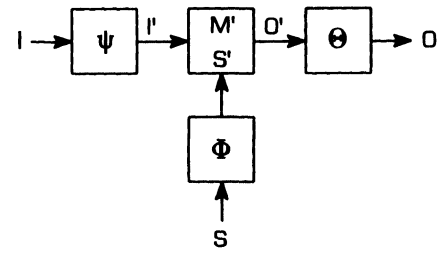

FIGURE 2 Output *H* behaviour realization

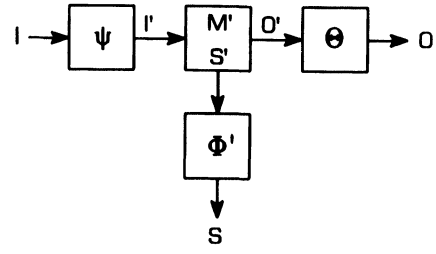

FIGURE 3 State and output behaviour realization.

the **realization structure** for M defined by M′ and will be denoted by **str(M′)**.

2.2 Partitions and Partition Pairs

Partitions and partition pairs are useful for modelling information and information flows inside and between the machines. Let S be any set of elements. **Partition** π on S is defined as follows:

$$\pi = \{B_i \mid B_i \subseteq S \text{ and } B_i \cap B_j = \varnothing \text{ for } i \neq j \text{ and } \bigcup_i B_i = S\}$$

For a given $s \in S$, the **block of partition π containing s** is denoted as **$[s]\pi$;** while **$[s]\pi=[t]\pi$** denotes that **s and t are in the same block of π**. Similarly, the block of partition π containing S′, where $S' \subseteq S$, is denoted by $[S']\pi$. The partition containing only one element of S in each block is called a **zero partition** and it is denoted by $\pi_S(\mathbf{0})$. The partition containing all the elements of S in one block is called an **identity partition** and it is denoted by $\pi_S(\mathbf{1})$.

Let π_1 and π_2 be two partitions on S. The **partition product $\pi_1 \cdot \pi_2$** is a partition on S such that $[s]\pi_1 \cdot \pi_2 = [t]\pi_1 \cdot \pi_2$ if and only if $[s]\pi_1 = [t]\pi_1$ and $[s]\pi_2 = [t]\pi_2$. The **partition sum** $\pi_1 + \pi_2$ is a partition on S such that $[s]\pi_1 + \pi_2 = [t]\pi_1 + \pi_2$ if and only if a sequence: $s=s_0, s_1,\ldots,$ $s_n = t$, $s_i \in S$ for $i = 1..n$, exists for which either $[s_i]\pi_1 = [s_{i+1}]\pi_1$ or $[s_i]\pi_2 = [s_{i+1}]\pi_2$, $0 \leq i \leq n-1$. From the above definitions, it follows that the blocks of $\pi_1 \cdot \pi_2$ can be obtained by intersecting the blocks of π_1 and π_2, while the blocks of $\pi_1 + \pi_2$ are obtained by combining all the blocks of π_1 and π_2 which contain common elements. Also, π_2 **is greater than or equal** to π_1: $\pi_1 \leq \pi_2$ if and only if each block of π_1 is included in a block of π_2. Thus, $\pi_1 \leq \pi_2$ if and only if $\pi_1 \cdot \pi_2 = \pi_1$, or if and only if $\pi_1 + \pi_2 = \pi_2$.

Any partition π on S can be interpreted as an equivalence relation defined on S with the equivalence classes being the blocks of π. Using this interpretation, the *partition π gives information about the elements of S with limited precision* to the equivalence class. With this information it is possible to distinguish the elements from different classes although it is impossible to distinguish elements of the same class. The partition product can be interpreted as a product of the appropriate equivalence relations introduced by these partitions; it represents combined information about the elements of S that is provided by the relations together. The partition sum can be interpreted as the sum of the appropriate equivalence relations introduced by the partitions and it represents information about elements of S after applying the combined abstraction provided by the relations involved. The partial ordering relation $\leq$ denotes the fact that if $\pi_1 \leq \pi_2$, then π_1 (and thus the associated equivalence relation) provides information about elements of S, that is at least as precise as information given by π_2 (and its associated equivalence relation). A zero partition provides complete information about the elements of S and an identity partition gives no information.

A set system π on a set S is defined as a collection of subsets $B_1, B_2,\ldots, B_i$ of S such that $\bigcup_i B_i = S$ and $B_i \not\subseteq B_j$ for $i \neq j$. The only difference between a partition and a set system is that the subsets B_i of the set system are not required to be disjoint. A set system π on S can be interpreted as a compatibility relation defined on S with the compatibility classes being the blocks of π. Such a compatibility relation is reflexive and symmetric but is not required to be transitive. If it is transitive, i.e. the subsets B_i of the set system π are disjoined, then the compatibility relation is an equivalence relation and the set system π is a partition.

Let $\bar{\delta}$ and $\bar{\lambda}$ denote the **functions which map the subsets** of S, I or $S \times I$ in the subsets of S or O, respectively, in accordance with mappings provided by the functions δ and λ for the elements of the appropriate subsets, i.e. $\bar{\delta}(B, x) = \{\delta(s, x) \mid s \in B \subseteq S \wedge x \in I\}$, $\bar{\delta}(s, A) = \{\delta(s, x) \mid s \in S \wedge x \in A \subseteq I\}$, $\bar{\delta}(D) = \{\delta(s, x) \mid (s,x) \in D \subseteq S \times I\}$, $\bar{\lambda}(B, x) = \{\lambda(s, x) \mid s \in B \subseteq S \wedge x \in I\}$, $\bar{\lambda}(s, A) = \{\lambda(s, x) \mid s \in S \wedge x \in A \subseteq I\}$, and $\bar{\lambda}(D) = \{\lambda(s, x) \mid (s,x) \in D \subseteq S \times I\}$.

Let π_S, τ_S, π_I, π_O, $\pi_{S\times I}$ be partitions on $M = (I,S,O,\delta,\lambda)$. In particular π_S, τ_S on S, π_I on I, π_O on O, and $\pi_{S\times I}$ on $S \times I$.

(π_S,τ_S) is an **S-S partition pair** if and only if
$\forall B \in \pi_S\ \forall x \in I$: $\bar{\delta}(B,x) \subseteq B'$, $B' \in \tau_S$.

(π_I,π_S) is an **I-S partition pair** if and only if
$\forall A \in \pi_I\ \forall s \in S$: $\bar{\delta}(s,A) \subseteq B$, $B \in \pi_S$.

(π_S,π_O) is an **S-O partition pair** if and only if
$\forall B \in \pi_S\ \forall x \in I$: $\bar{\lambda}(B, x) \subseteq C$, $C \in \pi_O$.

(π_I,π_O) is an **I-O partition pair** if and only if
$\forall A \in \pi_I\ \forall s \in S$: $\bar{\lambda}(s,A) \subseteq C$, $C \in \pi_O$.

$(\pi_{S\times I},\pi_S)$ is an **S × I-S partition pair** if and only if
$\forall D \in \pi_{S\times I}$: $\bar{\delta}(D) \subseteq B$, $B \in \pi_S$.

$(\pi_{S\times I},\pi_O)$ is an **S × I-O partition pair** if and only if
$\forall D \in \pi_{S\times I}$: $\bar{\lambda}(D) \subseteq C$, $C \in \pi_0$.

The interpretation of the notions introduced above is as follows: (π_S, τ_S) is an S-S partition pair if and only if the blocks of π_S are mapped by M into the blocks of τ_S, i.e. the input and the block of π_S will unambiguously determine the block of τ_S in which the next-state will be contained. In other words, knowing the input and having

information about the present state with precision to the equivalence classes introduced by π_S, it is possible to compute the information about the next-state with precision to the equivalence classes introduced by τ_S. Interpreting the notions of I-S, S-O and I-O partition pairs is similar. Partition π_S has a **substitution property** (it is an SP-partition) if and only if (π_S, π_S) is an S-S pair.

Let π be a partition on S. The **minimal second partition** which forms an S-S partition pair with π as a first partition will be denoted by $\mathbf{m}_{S\text{-}S}(\pi)$. The **maximal first partition** which forms an S-S partition pair with π as a second partition will be denoted by $\mathbf{M}_{S\text{-}S}(\pi)$. It can be proved [18] that:

$$m_{S-S}(\pi) = \prod\{\pi_i \mid (\pi,\pi_i) \text{ is a S-S partition pair}\}$$

$$M_{S-S}(\pi) = \sum\{\pi_i \mid (\pi_i,\pi) \text{ is a S-S partition pair}\}$$

The interpretation of $m_{S\text{-}S}(\pi)$ and $M_{S\text{-}S}(\pi)$ partition is the following: For a given π, $m(\pi)$ describes the largest amount of information which can be computed about the next-state of M knowing the block of π which contains the present state. $M(\pi)$ describes the minimum amount of information which must be known about the present state of M in order to be able to compute the information about the next-state with precision to π. In a strictly similar way, m and M operators can be defined and interpreted for I-S, S-O and I-O partition pairs.

$\pi_{S\times I}$ is a **partition on S × I induced by an output partition** π_O ($\pi_{S\times I} = \text{ind}(\pi_o)$) if and only if:

$$\forall s,t \in S\ \forall x,y \in I\text{: if } [\lambda(s,x)]\pi_O = [\lambda(t,y)]\pi_O \text{ then } [(s,x)]\pi_{S\times I} = [(t,y)]\pi_{S\times I},$$

i.e., if $\pi_{S\times I}$ is a partition on S × I induced by an output partition π_O (notation: $\pi_{S\times I} = \text{ind}(\pi_O)$) and, if it is known that the output y of M is contained in the block C: $C \in \pi_O$ then it is also known that the pair (s,x) consisting of the present state and input of M is contained in the block D: $D \in \pi_{S\times I}$, where block D is unambiguously indicated by block C. It can be said that block D of $\pi_{S\times I}$ is induced on S × I by block C of π_O and denoted by D = ind(C).

It is possible to prove that $\pi_{S\times I}$ is a partition induced on S × I by an output partition π_O if and only if $\pi_{S\times I} \geq M_{S\times I\text{-}O}(\pi_O)$, i.e. the smallest induced partition for a certain π_O is $\pi_{S\times I}$: $\pi_{S\times I} = M_{S\times I\text{-}O}(\pi_O)$.

$\pi_{S\times I}$ is **a partition on S × I induced by a state partition** π_S ($\pi_{S\times I} = \text{ind}(\pi_S)$) if and only if:

$$\forall s,t \in S\ \forall x,z \in I\text{: if } [s]\pi_S = [t]\pi_S \text{ then } [(s,x)]\pi_{S\times I} = [(t,z)]\pi_{S\times I},$$

i.e. if it is known that the present state s of M is contained in a block B: $B \in \pi_S$ then it is also known that each pair (s,x) is contained in the block D: $D \in \pi_{S\times I}$, where block D is unambiguously indicated by block B (D = ind(B)). In a similar way, the notion of a partition induced on S × I by an input partition π_I can be defined and interpreted.

$\pi_{S\times I}$ is **a partition on S × I induced by an input partition** π_I if and only if:

$$\forall s,t \in S\ \forall x,z \in I\text{: if } [x]\pi_I = [z]\pi_I \text{ then } [(s,x)]\pi_{S\times I} = [(t,z)]\pi_{S\times I}$$

It is easy to prove that the **smallest induced partition** for a certain π_S (π_I) is $\pi_{S\times I}$: $[(s,x)]\pi_{S\times I} = [(t,z)]\pi_{S\times I}$ if and only if $[s]\pi_S = [t]\pi_S$ ($[x]\pi_I = [z]\pi_I$).

For the purpose of bit decompositions (in which the input and/or output bits are appropriately distributed instead of the input/output symbols), the concept of bit partitions has been introduced. Let: $B = \{b_1,b_2,..,b_{|B|}\}$ be a set of bits. Let: $T = \{t_1,t_2,..t_{|T|}\}$ be a set of symbols (bit value patterns) on B. Now, each bit b_k:$b_k \in B$, induces a two block partition $\pi_T(b_k)$ on the set T (in the case of incompletely specified machines on the subset of T for which the value of this bit is specified). One block of $\pi_T(b_k)$ contains the symbols for which the bit b_k has the value 0. In the second block of $\pi_T(b_k)$ we find the symbols for which b_k has the value 1. In the following text we will also use $\tau_T(b_k)$ to denote partition on T induced by b_k.

A partition π_B on the set of bits B: $\pi_B = \{\bar{b}_1,\bar{b}_2,\ldots,\bar{b}_k, \overline{(b_{k+1},\ldots,b_{|B|})}\}$, where important (for distinguishing between certain symbols) bits $b_1,\ldots,b_k$ are kept in separate blocks and don't care bits $b_{k+1},..,b_{|B|}$ are kept in a single block called a **don't care block** (denoted by **dcb(π)**), is called a **bit partition**.

The product (·) and sum (+) operations as well as the ordering relations (≤) for bit partitions are defined in the same way as for "normal" partitions, but the block of the bit-partition's product being the product of a block (important or don't care) with an important block is an important block; and the block of the bit-partition's sum being the sum of some blocks (important or don't care) with a don't care block is a don't care block. The zero bit-partition is defined as a bit partition with an empty don't care block.

π_T is a **symbol partition induced by a bit partition** π_B ($\pi_T = \text{ind}(\pi_B)$) if and only if $\pi_T \geq \prod_{b_i \in (B\setminus dcb(\pi_B))} \pi_T(b_k)$.

π_B is a **bit partition induced by a symbol partition** π_T ($\pi_B = \text{ind}(\pi_T)$) if and only if $\forall b_k \in (B\setminus dcb(\pi_B))$: $\pi_T(b_k) \geq \pi_T$.

TABLE I
State/output table of the original machine M

$S\backslash x_2x_1$	00	01	10	$y_3y_2y_1$
1	6	3	2	000
2	5	4	1	001
3	2	5	4	100
4	1	6	3	101
5	4	1	6	110
6	3	2	5	111

If $\pi_T = \text{ind}(\pi_B)$ then having π_B, one can compute the blocks of π_T. If $\pi_B = \text{ind}(\pi_T)$ then, knowing the block of π_T one can compute the values of all the important bits from π_B.

The last two definitions relate the symbol and bit partitions and allow the bit full-decomposition to be considered as a special case of a symbol full-decomposition.

Example: Consider a Moore machine M defined by the Table 1 and the following partitions on M: state partitions: $\pi_1 = \{\overline{1,3,5};\overline{2,4,6}\} = \{A,B\}$ and $\pi_2 = \{\overline{1,2,3,4};\ \overline{5,6}\} = \{C,D,E\}$, an input bit partition: $\pi_{IB1} = \{\overline{x_1};(\overline{x_2})\}$, an input partition $\pi_{I1} = \{\overline{00,\ 10};\ \overline{01}\}$, output bit partitions: $\pi_{OB_1} = \{\overline{y_1};(\overline{y_2,y_3})\}$ and $\pi_{OB2,3}=\{\overline{y_2};\overline{y_3};(\overline{y_1})\}$, and an output partition $\pi_O(y_1) = \{\overline{000,\ 100,\ 110};\ 001,\ 101,\ 111\}$.

For the above partitions, the following statements are true:

(1) π_1 and π_2 are SP-partitions,
(2) $\pi_1 \cdot \pi_2 = \pi_s(0)$ (i.e. the products of the blocks from π_1 and π_2 are empty or contain just one element from S, so that the resulting product partition keeps each element from S in a separate block),
(3) $\pi_{I1} = \pi_I(x_1) = \text{ind}(\pi_{IB1})$ and (π_{I1}, π_1) is an I-S partition pair (i.e. the blocks of π_{I1} are unambiguously defined by values of x_1 and the blocks of π_{I1} are mapped by δ into the blocks of π_1),
(4) $\pi_O(y_1) = \text{ind}(\pi_{OB1})$, $\pi_{OB1} = \text{ind}(\pi_O(y_1))$ and $(\pi_1, (\pi_O(y_1))$ is an S-O partition pair (i.e. the values of y_1 are unambiguously defined by the blocks of $\pi_O(y_1)$ and the blocks of π_1 are mapped by λ into the blocks of $\pi_O(y_1)$),
(5) $\pi_{OB1} \cdot \pi_{OB2,3} = \pi_{OB}(0)$.

3. GENERAL FULL-COMPOSITION

Let us consider realization of M = (I,S,O,δ,λ) (Fig. 3) by M′ being a composition of n partial machines M_i (as shown in Fig. 4 for the case of two machines M_i).

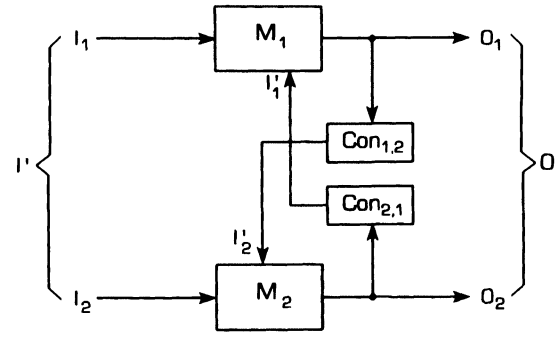

FIGURE 4 General full-composition of two component machines M_1 and M_2 without local connections.

Definition 1. A **general composition** of n sequential machines M_i:e GC = $(\{M_i\}, \{Con_i\})$ consists of the following objcts:

(1) $\{M_i = \{I_i^*, S_i, O_i, \delta_i, \lambda_i\}, I_i^* = I_i \times I'_i, 1 \le i \le n\}$, a set of sequential machines referred to as **component machines**,
(2) $\{Con_i: \times O_j \rightarrow I'_i, 1 \le i,j \le n\}$, a set of surjective functions referred to as **connecting rules** of the component machines.

A general composition is said to be in **canonical form** if and only if the connection rules Con_i compute the vector values and have the following form:

$$Con_i(y_1,\ldots,y_n) = (Con_{1,i}(y_1),\ldots, Con_{n,i}(y_n))$$

i.e. a (partial) output information from a certain component machine j, $1 \le j \le n$, is separately transmitted to the input of a certain machine i, $1 \le i \le n$, i.e. without combining it with a (partial) output information from other partial machines k, $1 \le k \le n$, $k \ne j$.

A general composition is said to be in **maximally preprocessed form** if the connection rules Con_i compute the scalar values, i.e. information transmitted from various partial machines to a certain machine is combined prior to connecting it to the input of this machine. Of course, the compositions in **partially preprocessed form**, lying between the two above extremes, are also possible.

Allowing for **external local connections** between the outputs and inputs of a certain component machine M_i, gives more freedom in describing the circuit structure. The machine M_i can influence it own behavior partly through its internal state and partly by affecting the inputs. The precise form of this influence is defined by a specific choice of the connections Con_i and machine functions δ_i and λ_i.

Definition 2. A general composition GC of n sequential machines defines the **general composition machine** $M_{GC}(GC) = M_{GC}(\{M_i\}, \{Con_i\}) = (I_{GC}, S_{GC}, O_{GC}, \delta_{GC}, \lambda_{GC})$ with $I_{GC} = \times I_i$, $S_{GC} = \times S_i$, $O_{GC} = \times O_i$,

$\delta_{GC}(s_{GC}, x_{GC}) = \delta_{GC}((s_1,\ldots, s_n), (x_1,\ldots, x_n)) = \times \delta_i(s_i, (x_i, Con_i(y_1,\ldots, y_n))$,

$\lambda_{GC}(s_{GC}, x_{GC}) = \lambda_{GC}((s_1,\ldots, s_n), (x_1,\ldots, x_n)) = \times \lambda_i(s_i, (x_i, Con_i(y_1,\ldots, y_n))$.

Formal definitions for compositions TC of various special types T can be introduced in a very similar way, as special cases of the above definition [22]–[25]. Each one of them defines the appropriate type T composition machine M_{TC}. We will say that **composition TC** of the type T of n sequential machines M_i **realizes** machine M if and only if M_{TC} realizes M. We will not distinguish between the type T composition TC and the type T composition machine M_{TC} unless it can lead to misunderstanding.

In a general composition, there is a danger of information loops occurring in the exchanged information. Such loops at the level of elementary (binary) signal lines will result in sequential behavior of the two interconnected combinational circuits which compute λ_i instead of the required combinational behaviour. We say that a general **composition is legal** if and only if the composition λ^* of λ_i is guaranteed to be a function.

This is satisfied if and only if the signal values of each elementary (binary) signal used for information exchange between the partial machines are computed independently of the values of this signal. Of course the cyclic signal flows can occur exclusively due to the interconnection circuity. Checking for acyclic signal flow is equivalent to tracing the primary information sources i.e. to check if the signal values of each elementary signal line, used for information exchange between partial machines or for transmitting information between the outputs and inputs of a certain machine, are originally computed from the primary input and state information of the composition machine only. Therefore, the partial machines must together possess enough primary input and state information to compute all the information transmitted by interconnections. The composition's legality guarantees that the information that has to be transmitted will be computed by the partial machines.

Fortunately, the legality is structurally guaranteed for most of the special cases of the general composition because the information loops are, in most cases, impossible at the level of total information flows between the outputs and inputs of the partial machines. One has to check for non-closed loops at the level of partial information flows (signal lines) only in the most general cases of the general composition, i.e., for machines other than Moore machines in cases where the exchanged information is computed in more than one partial machine when using some information transmitted from the other machines or in the presence of local connections. In particular, the composition legality is structurally guaranteed for Moore machines (where the partial state information of the component machines is transmitted between the partial machines) and for the following compositions of Mealy machines without local connections: parallel compositions (without information exchange), serial compositions (with unidirectional information flow) and compositions where the exchanged information is only computed from the (primary) input and state information of each partial machine itself.

4. GENERAL FULL-DECOMPOSITION

Let us consider realizations of M by M′ being a general full composition of n partial machines M_i. In order to construct a general full-decomposition of a machine M, it is necessary to find partial machines $M_i = (I_i^*, S_i, O_i, \delta_i, \lambda_i)$, their interconnection structure represented by Con_i as well as the mappings: Ψ: $I \rightarrow \times I_i$, Φ: $S \rightarrow \times S_i$ and Θ: $\times O_i \rightarrow O$ such that the machines M_i interconnected by Con_i, together with the mappings Ψ, Φ and Θ, can realize the behavior of a machine M (Fig. 5).

The state S_i, input I_i^* and output O_i of each component machine provide partial information about the state S, input I and output $S \times I$ of the original machine and in this way, they all provide partial information about $S \times I$. Information I'_i about $\times O_i$ can be transmitted between the component machines. Each component machine M_i must be able to compute its next-state and output information from the partial information about $S \times I$ only, provided by S_i, I_i, and I'_i. A general composition of the component machines, together with the input, output, and state decoders Ψ, Φ, and Θ, must be able to realize the behavior of the original machine M.

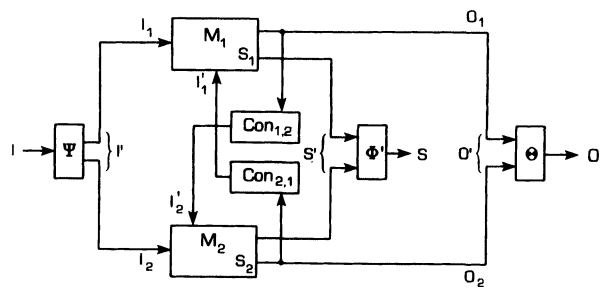

FIGURE 5 General full-decomposition of a machine M into two component machines M_1 and M_2 without local connections.

Implementation of the general decomposition model requires four sorts of components: an input coder (preprocessor) Ψ, simultaneously operating component machines (main processors) M_i, output/state decoders (postprocessors) Θ/Φ', and the communication circuitry Con_i. The input coder, the output/state decoders, and the communication circuitry can be implemented as combinational circuits. In some special cases these circuits can be reduced to the appropriate distribution or joining of the appropriate input, state, or output bit lines. Implementation of the component machines depends on the nature of the original machine M. The component machines can take the appropriate form of a general sequential machine, a Moore machine, a state machine, or a combinational circuit. Each component of the general full-decomposition model can be further decomposed using this model or its special cases.

The general full-decomposition model is powerful and very natural. It models an information-processing system in terms of information flows and cooperating information-processing units. Special cases of the general full-decomposition model cover all other known models for decomposing sequential and combinational circuits.

Various full-decomposition types can be distinguished on the basis of the type of connections between the component machines (general, serial or parallel connections with state and input, state or input information transmitted), and on the type of coding/decoding (symbol or bit coding/decoding) [22]–[27].

Definition 3. The machine $str(M_{TC}(\{M_i\}, \{Con_i\}))$ is a **full-decomposition of a certain type T** of machine M if and only if the type T composition of machines M_i realizes M. In particular, the machine $str(M_{GC}(\{M_i\}, \{Con_i\}))$ is a **general full-decomposition** of the machine M if and only if a general composition GC of M_i realizes M.

Each component machine M_i of a general decomposition computes partial information about the next-state and output of the original machine M and the component machines cooperating together realize the required behaviour specified by the original machine M. To realize this behaviour, the component machines must be able to compute together enough information about the next-state and output of the original machine M within each period between the successive sampling moments of the state and output information.

Below, the theorem concerning the existence of a general decomposition will be presented. For reasons of simplicity in presentation, the single-state realizations of the completely specified sequential machines will only be considered further. However, the presented results can be quite naturally extended. All the concepts presented will remain valid except for replacing the partitions by the set systems in order to cover the multi-state realizations and the incompletely specified machines (also the weak or extended partition pairs can be used in the place of partition pairs) [18].

Let π_I^i, π_S^i, and $\pi_{S\times I}^i$ be partitions on $M = (I,S,O,\delta,\lambda)$ on I, S, and $S \times I$, respectively. Let $\pi_{S\times I}^{ij}$ be a partition on $S \times I$, such that $\pi_{S\times I}^{ij} \geq \pi_{S\times I}^i$ and $\pi'^j_{S\times I} = \prod_{i=1,\ldots,n} \pi^{ij}_{S\times I}$. Let $\pi_{S\times I} = \prod_{i=1,\ldots,n} \pi^i_{S\times I}$. Let $\pi^{I\,i}_{S\times I}$ and $\pi^{S\,i}_{S\times I}$ be partitions induced on $S \times I$ by π_I^i and π_S^i, respectively. Let $\pi^I_{S\times I} = \prod_{i=1,\ldots,n} \pi^{I\,i}_{S\times I}$ and $\pi^S_{S\times I} = \prod_{i=1,\ldots,n} \pi^{S\,i}_{S\times I}$. Below, the term "trinity of partitions" will be used in the sense of three strongly related partitions.

Theorem 1 (general decomposition)
A sequential machine $M = (I,S,O,\delta,\lambda)$ has a **general full-decomposition** with the output behaviour realization with n component machines if and only if n trinities of partitions $(\pi_I^i, \pi_S^i, \pi_{S\times I}^i)$ exist, so that:

(1) $(\pi^I_{S\times I}{}^i \cdot \pi^S_{S\times I}{}^i \cdot \pi'_{S\times I}{}^i,\ \pi_S^i)$ is an $S \times I$ - S partition pair,

(2) $\pi^I_{S\times I}{}^i \cdot \pi^S_{S\times I}{}^i \cdot \pi'_{S\times I}{}^i \leq \pi_{S\times I}{}^i$,

(3) $\pi^I_{S\times I} \cdot \pi^S_{S\times I} \leq \pi'_{S\times I}{}^i$,

(4) $(\pi_{S\times I}, \pi_O(0))$ is an $S \times I$ - O partition pair.

Additionally, if

(5) $\prod_i \pi_S^i = \pi_S(0)$

is satisfied, then the state behaviour of M will be realized too.

The proof of Theorem 1 can be found in Appendix.

Theorem 1 can be interpreted in the terms of the equivalence relations introduced by the appropriate partitions and it is interpreted graphically in Fig. 6 for the case of two partial machines without local connections. When computing the output function λ, the sequential machine M classifies the elements of $S \times I$ into the classes of an equivalence relation. In this equivalence relation, the elements of $S \times I$, mapped by λ in the same values, are in the same classes and those mapped in different values are in different classes. Analogously, when computing the next-state function δ, the sequential machine M classifies the elements of $S \times I$ into the classes of an equivalence relation defined by values of δ. It is possible to obtain each of those classifications as a product of n other classifications. These are defined by the appropriate output and next-state functions of the partial machines if and only if the following conditions are satisfied:

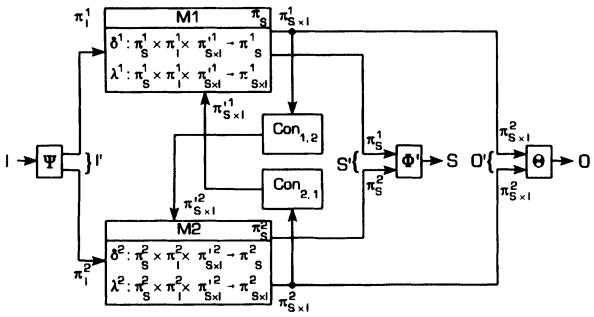

FIGURE 6 General full-decomposition of M with two component machines M_1 and M_2 defined by the trinities $(\pi_I^1, \pi_S^1, \pi_{S\times I}^1)$ and $(\pi_I^2, \pi_S^2, \pi_{S\times I}^2)$.

- the product $\prod_i \pi_S^i$ of the state classification relations of the partial machines forms the state classification relation $\pi_S(0)$ of the original machine M (condition 5),
- the product $\prod_i \pi_{S\times I}^i$ of the output classification relations of the partial machines forms the classification relation which enables unambiguous computation of the output classification relation $\pi_o(0)$ of the original machine M (condition 4),
- each partial machine is able to compute its own classifications π_S^i and $\pi_{S\times I}^i$ based on the present state and primary input classification provided by its own state and primary input, and the classification of the elements from $S \times I$ provided by the extra input from the other machines (conditions (1) and (2)),
- the composition of the partial machines is legal (condition (3)).

For incompletely specified machines or multi-state realizations, partitions have to be replaced with set systems and a theorem similar to Theorem 1 can be proved. Its interpretation will be slightly different. The classifications of the elements from $S \times I$ computed by the original and component machines will no more define the equivalence relations but the compatibility relations denoted by the appropriate set systems. In these relations each element can be a member of many compatibility classes, because the compatibility relations are not required to be transitive.

The next section of the paper is devoted to the discussion of some important special cases of the general full-decomposition model and Theorem 1. Usage of the model for decompositional logic synthesis is discussed in Section 6 and an example to illustrate the usage is presented in Section 7.

5. SPECIAL CASES

The general full-decomposition model covers all other known structural models for sequential and combinational circuits, including the following:

- *parallel full-decompositions* [19]–[27], in which each of the component machines can compute its own next-state and output independently (Fig. 7);
- *serial full-decompositions* [22]–[27], in which only one of the component machines (M_2) uses information from the second machine (M_1) in order to compute its own next-state and output (Fig. 8);
- decompositions with the *separate realization of the next-state and output functions* [24] (Fig. 9);
- *bit full-decompositions* [25]–[27], where the decoders Ψ and Θ are reduced to the appropriate distribution of the input and output bit lines (Fig. 10);
- *input-bit parallel full-decompositions* (referred to in the literature as cascade decompositions [30], serial decompositions [35], Boolean decompositions [14][44], decompositions with generalized decoders [9][44], three-level decompositions [37], or decompositions into submachines [46]) (Fig. 11);
- *bit parallel full-decompositions* (parallel decompositions [21], output decompositions [30]) (Fig. 12).
- *all the special structures modelled by Boolean algebra, BDD's or any other traditional means* (for example, the two-level logic AND-OR structures of combinational circuits are special parallel decompositions with partial machines $M_1,\ldots, M_k$ restricted to AND functions and with the output decoder Θ composed of exclusively OR functions).

Below, the general decomposition theorem will be interpreted for a number of important special cases. For reasons of simplicity in presentation, we consider the decompositions with only two machines and without local connections later in the section; however, the presented results can be very easily extended to n machines.

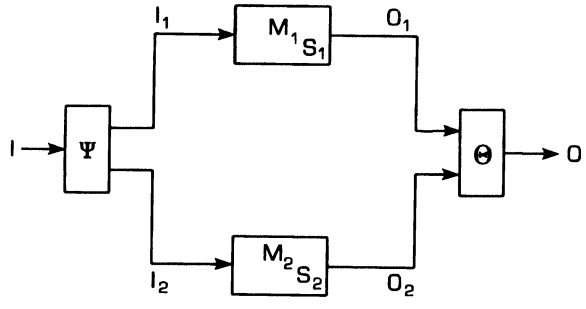

FIGURE 7 Parallel full-decomposition.

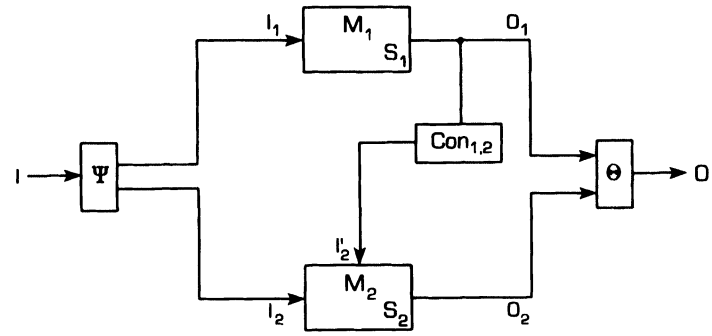

FIGURE 8 Serial full-decomposition

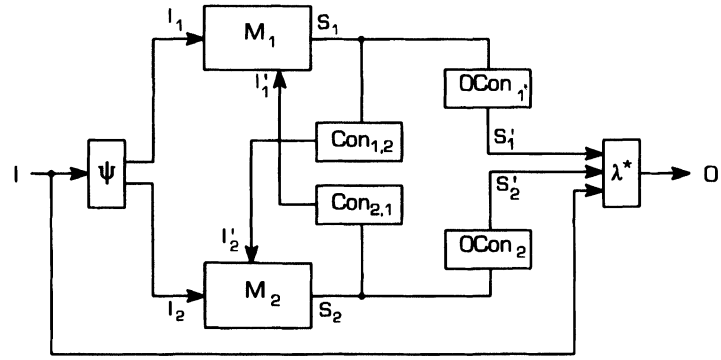

FIGURE 9 Full-decomposition with the separate realization of the next-state and output functions.

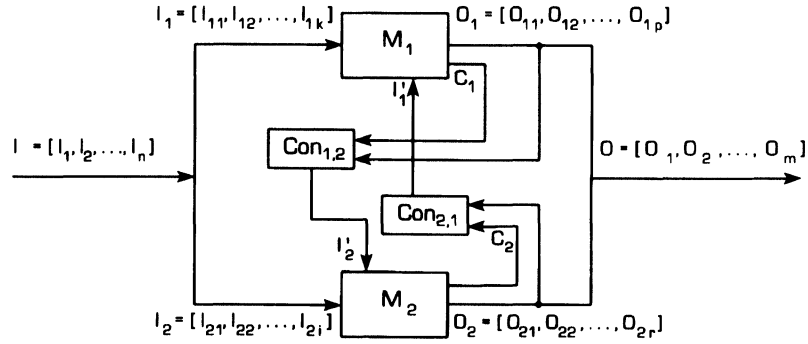

FIGURE 10 The bit general full-decomposition.

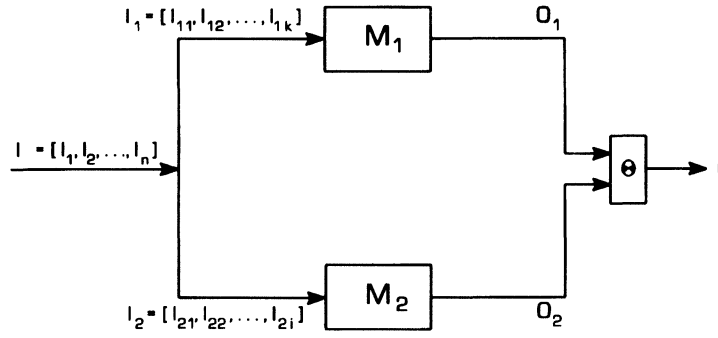

FIGURE 11 The input bit parallel full-decomposition.

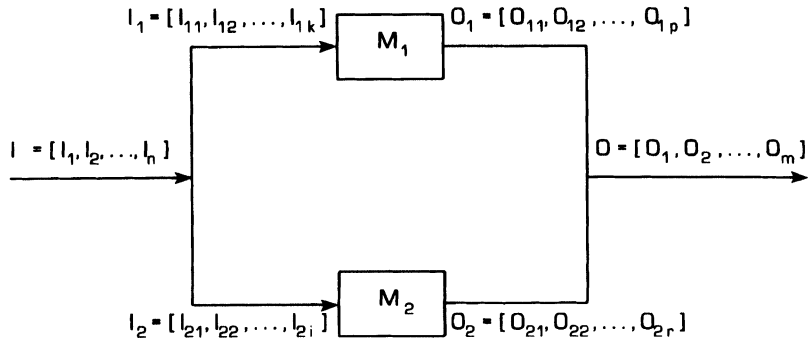

FIGURE 12 The bit parallel full-decomposition.

5.1 Sequential Machines

Let π_I, τ_I, π_S, τ_S, $\pi_{S\times I}$, $\tau_{S\times I}$ be partitions on M = (I,S,O,δ,λ), on I, S, and S × I, respectively. Let $\pi'_{S\times I}$ and $\tau'_{S\times I}$ be partitions on S × I, such that $\pi'_{S\times I} \geq \pi_{S\times I}$ and $\tau'_{S\times I} \geq \tau_{S\times I}$, and $\pi^I_{S\times I}$, $\tau^I_{S\times I}$, $\pi^S_{S\times I}$, $\tau^S_{S\times I}$ be partitions induced on S × I by π_I, τ_I, π_S, τ_S respectively.

Theorem 2 (general decomposition with two component machines and without local connections)

A sequential machine M = (I, S, O, δ, λ) has a **general full-decomposition** with the output behaviour realization if and only if two trinities of partitions (π_I, π_S, $\pi_{S\times I}$) and (τ_I, τ_S, $\tau_{S\times I}$) exist, so that:

(1) $(\pi^I_{S\times I} \cdot \pi^S_{S\times I} \cdot \tau'_{S\times I}, \pi_S)$ and $(\tau^I_{S\times I} \cdot \tau^S_{S\times I} \cdot \pi'_{S\times I}, \tau_S)$ are S × I - S partition pairs,

(2) $\pi^I_{S\times I} \cdot \pi^S_{S\times I} \cdot \tau'_{S\times I} \leq \pi_{S\times I}$ and $\tau^I_{S\times I} \cdot \tau^S_{S\times I} \cdot \pi'_{S\times I} \leq \tau_{S\times I}$,

(3) $\pi^I_{S\times I} \cdot \tau^I_{S\times I} \cdot \pi^S_{S\times I} \cdot \tau^S_{S\times I} \leq \pi'_{S\times I}$ and $\pi^I_{S\times I} \cdot \tau^I_{S\times I} \cdot \pi^S_{S\times I} \cdot \tau^S_{S\times I} \leq \tau'_{S\times I}$,

(4) $(\pi_{S\times I} \cdot \tau_{S\times I}, \pi_O(0))$ is an S × I - O partition pair.

Additionally, if

(5) $\pi_S \cdot \tau_S = \pi_S(0)$

is satisfied, then the state behaviour of M will also be realized.

Theorem 2 is interpreted graphically in Fig. 13. The primary input, state, and output sets of partial machines are defined as blocks of the appropriate partitions from the partition trinities (π_I, π_S, $\pi_{S\times I}$) and (τ_I, τ_S, $\tau_{S\times I}$). The interconnection circuits $Con_{1,2}$ and $Con_{2,1}$ compute the appropriate extra input information of each component machine from the output information of the other component machine, by computing the blocks $\pi'_{S\times I}$ and $\tau'_{S\times I}$ respectively. The next-state and output functions of partial machines map the blocks of their appropriate state and input partitions (primary and from the interconnections) into the blocks of their state and output partitions. The input decoder Ψ computes the blocks of π_I and τ_I from the input information of the original machine M. The state and output decoders Φ′ and Θ compute the state and output information of the original machine M from the state and output information of the partial machines represented by partitions π_S, τ_S, and $\pi_{S\times I}$ and $\tau_{S\times I}$, respectively.

In the case of the general decomposition with state connection (type S), $\pi_{S\times I}'$ and $\tau_{S\times I}'$ give partial information about the states of M_1 and M_2. This information can be described by state partitions π_S' and τ_S' induced on S by $\pi_{S\times I}'$ and $\tau_{S\times I}'$.

Now, it is possible to partly formulate the conditions (1), (2), and (3) of Theorem 2 directly in the terms of π_I, τ_I, π_S, τ_S, π_S', and τ_S' instead of formulating them in the terms of the appropriate partitions on S × I.

Theorem 3. A sequential machine M = (I,S,O,δ,λ) has a **general full-decomposition of type S** with the output

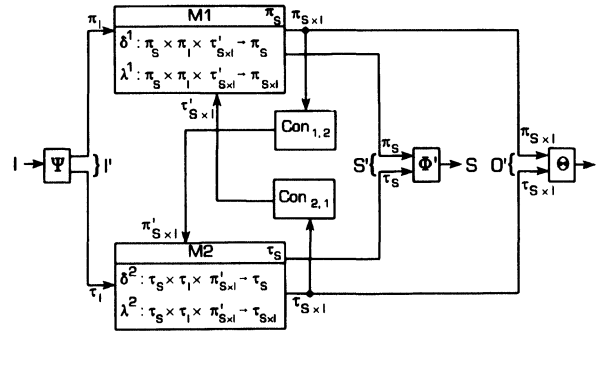

FIGURE 13 General full-decomposition of M with two component machines M_1 and M_2 defined by trinities (π_I, π_S, $\pi_{S\times I}$) and (τ_I, τ_S, $\tau_{S\times I}$).

behavior realization if and only if two trinities of partitions $(\pi_I,\pi_S,\pi_{S\times I})$ and $(\tau_I,\tau_S,\tau_{S\times I})$ exist, so that

(1a) (π_I, π_S) and (τ_I, τ_S) are I-S partition pairs
(1b) $(\pi_S \cdot \tau_S', \pi_S)$ and $(\tau_S \cdot \pi_S', \tau_S)$ are S-S partition pairs
(2) $\pi^I{}_{S\times I} \cdot \pi^S{}_{S\times I} \cdot \tau'^S{}_{S\times I} \le \pi_{S\times I}$ and $\tau^I{}_{S\times I} \cdot \tau^S{}_{S\times I} \cdot \pi'^S{}_{S\times I} \le \tau_{S\times I}$, where $\pi'^S{}_{S\times I} = \mathrm{ind}(\pi_S')$, and $\tau'^S{}_{S\times I} = \mathrm{ind}(\tau_S')$
(3) $\pi_S' \ge \pi_S \cdot \tau_S$ and $\tau_S' \ge \pi_S \cdot \tau_S$
(4) $(\pi_{S\times I} \cdot \tau_{S\times I}, \pi_O(O))$ is an S × I-O partition pair.

Additionally, if

(5) $\pi_S \cdot \tau_S = \pi_S(0)$ is also satisfied, then the state behaviour of M will also be realized.

In a parallel decomposition, no information flows between the partial machines. So, the partitions $\pi_{S\times I}'$, $\tau_{S\times I}'$, π_S', and τ_S' in Theorems 2 and 3 are reduced to $\pi_{S\times I}(1)$ and $\pi_S(1)$ respectively. In this manner, the Theorems 2 and 3 are reduced to the following theorem.

Theorem 4. A machine M has a **parallel full-decomposition** with output behaviour realization if and only if two partition trinities $(\pi_I,\pi_S,\pi_{S\times I})$ and $(\tau_I,\tau_S,\tau_{S\times I})$ exist that satisfy the following conditions:

(1) (π_I, π_S) and (τ_I, τ_S) are I-S partition pairs,
(2) π_S and τ_S are SP-partitions,
(3) $\pi^I{}_{S\times I} \cdot \pi^S{}_{S\times I} \le \pi_{S\times I}$ and $\tau^I{}_{S\times I} \cdot \tau^S{}_{S\times I} \le \tau_{S\times I}$,
(4) $(\pi_{S\times I} \cdot \tau_{S\times I}, \pi_O(0))$ is an S×I-O partition pair.

If the condition:

(5) $\pi_S \cdot \tau_S = \pi_S(0)$ is also satisfied, then the state behavior of M will also be realized.

In a serial decomposition only one of the component machines, say M_2, uses information of the output of the other machine (M_1). So, the partition $\tau_{S\times I}'$ in Theorem 2 is reduced to $\pi_{S\times I}(1)$.

In this manner Theorem 2 is reduced to the following theorem.

Theorem 5. A machine M has a **serial full-decomposition** with output behaviour realization if and only if two partition trinities $(\pi_I, \pi_S, \pi_{S\times I})$ and $(\tau_I,\tau_S,\tau_{S\times I})$ exist and the following conditions hold:

(1) (π_I, π_S) is an I-S partition pair, π_S is an SP-partition, and $(\tau^I{}_{S\times I} \cdot \tau^S{}_{S\times I} \cdot \pi'{}_{S\times I}, \tau_S)$ is an S×I-S partition pair,
(2) $\pi^I{}_{S\times I} \cdot \pi^S{}_{S\times I} \le \pi_{S\times I}$ and $\tau^I{}_{S\times I} \cdot \tau^S{}_{S\times I} \cdot \pi'{}_{S\times I} \le \tau_{S\times I}$,
(3) $(\pi_{S\times I} \cdot \tau_{S\times I}, \pi_O(0))$ is an S×I-O partition pair.

If the condition:

(4) $\pi_S \cdot \tau_S = \pi_S(0)$ is also satisfied, then the state behaviour of M will also be realized.

If $\tau'_S = \pi_S(1)$, Theorem 3 is reduced to the following theorem.

Theorem 6. A machine M has a **serial full-decomposition of type S** with output behaviour realization if and only if two partitions $(\pi_I, \pi_S, \pi_{S\times I})$ and $(\tau_I, \tau_S, \tau_{S\times I})$ exist that satisfy the following conditions:

(1) (π_I, π_S) and (τ_I, τ_S) are I-S partition pairs,
(2) π_S is an SP-partition and $(\tau_S \cdot \pi_S', \tau_S)$ is an S-S partition pair, where: $\pi_S' \ge \pi_S$,
(3) $\pi^I{}_{S\times I} \cdot \pi^S{}_{S\times I} \le \pi_{S\times I}$ and $\tau^I{}_{S\times I} \cdot \tau^S{}_{S\times I} \cdot \pi'^S{}_{S\times I} \le \tau_{S\times I}$, where $\pi'^S{}_{S\times I} = \mathrm{ind}(\pi_S')$,
(4) $(\pi_{S\times I} \cdot \tau_{S\times I}, \pi_O(0))$ is an S×I-O partition pair.

If the condition

(5) $\pi_S \cdot \tau_S = \pi_S(0)$ is also satisfied, then the state behaviour of M will also be realized.

Bit decomposition is a special case of symbol decomposition, where the input and output decoders Ψ and Θ are reduced to the appropriate distribution of the input and output bit lines (Figure 8). So, the partitions: π_I, τ_I, π_O, and τ_O in the decomposition theorems, should be replaced by the appropriate bit partitions π_{IB}, τ_{IB}, π_{OB}, and τ_{OB} on the set of input bits IB and the set of output bits OB. In this way, Theorem 2 for example, is transformed into the following theorem:

Theorem 7. A machine M has a **general bit full-decomposition** with output behaviour realization if and only if two bit partition trinities $(\pi_{IB}, \pi_S, \pi_{OB})$ and $(\tau_{IB}, \tau_S, \tau_{OB})$ exist that satisfy the following conditions:

(1) $(\pi^{IB}{}_{S\times I} \cdot \pi^S{}_{S\times I} \cdot \tau'{}_{S\times I}, \pi_S)$ and $(\tau^{IB}{}_{S\times I} \cdot \tau^S{}_{S\times I} \cdot \pi'{}_{S\times I}, \tau_S)$ are S × I-S partition pairs, where: $\tau^{IB}{}_{S\times I} = \mathrm{ind}(\pi_I)$, $\pi_I = \mathrm{ind}(\pi_{IB})$ and $\tau^{IB}{}_{S\times I} = \mathrm{ind}(\tau_I), \tau_I = \mathrm{ind}(\tau_{IB})$,
(2a) $\forall ob_k,\ ob_k \in OB\backslash dcb(\pi_{OB})$: $(\pi^{IB}{}_{S\times I} \cdot \pi^S{}_{S\times I} \cdot \tau'{}_{S\times I}, \pi_O(ob_k))$ is an S × I-O partition pair, and $\forall ob_k,\ ob_k \in OB\backslash dcb(\tau_{OB})$: $(\tau^{IB}{}_{S\times I} \cdot \tau^S{}_{S\times I} \cdot \pi'{}_{S\times I}, \tau_O(ob_k))$ is an S × I-O partition pair,
(2b) $\pi^{IB}{}_{S\times I} \cdot \pi^S{}_{S\times I} \cdot \tau'{}_{S\times I} \le \pi^{C1}{}_{S\times I}$ and $\tau^{IB}{}_{S\times I} \cdot \tau^S{}_{S\times I} \cdot \pi'{}_{S\times I} \le \tau^{C2}{}_{S\times I}$,
(2c) $\pi_{S\times I} = \left[\prod_{ob_k\in OB\backslash dcb(\pi_{OB})} \mathrm{ind}(\pi_O(ob_k))\right] \cdot \pi^{C1}{}_{S\times I}$, $\tau_{S\times I} = \left[\prod_{ob_k\in OB\backslash dcb(\tau_{OB})} \mathrm{ind}(\tau_O(ob_k))\right] \cdot \tau^{C2}{}_{S\times I}$,
(3) $\pi^{IB}{}_{S\times I} \cdot \tau^{IB}{}_{S\times I} \cdot \pi^S{}_{S\times I} \cdot \tau^S{}_{S\times I} \le \pi'{}_{S\times I}$ and $\pi^{IB}{}_{S\times I} \cdot \tau^{IB}{}_{S\times I} \cdot \pi^S{}_{S\times I} \cdot \tau^S{}_{S\times I} \le \tau'{}_{S\times I}$
(4) $\pi_{OB} \cdot \tau_{OB} = \pi_{OB}(0)$.

If the condition:

(5) $\pi_S \cdot \tau_S = \pi_S(0)$ is also satisfied, then the state behaviour of M will also be realized.

By replacing the input and output symbol partitions π_I, τ_I, π_O, and τ_O by the appropriate bit partitions π_{IB}, τ_{IB}, π_{OB}, τ_{OB}, the following theorem will be obtained from Theorem 4.

Theorem 8. A machine M has a **parallel bit full-decomposition** with output behaviour realization if and only if two bit partition trinities (π_{IB}, π_S, π_{OB}) and (τ_{IB}, τ_S, τ_{OB}) exist that satisfy the following conditions:

(1) (π_I, π_S) and (τ_I, τ_S) are IB-S partition pairs, where: $\pi_I = \text{ind}(\pi_{IB})$, $\tau_I = \text{ind}(\tau_{IB})$,
(2) π_S and τ_S are SP-partitions,
(3) $\forall ob_k$, $ob_k \in OB \backslash dcb(\pi_{OB})$: ($\pi_I$, $\pi_O(ob_k)$) and (π_S, $\pi_O(ob_k)$) are I-O and S-O partition pairs respectively,
(4) $\forall ob_k$, $ob_k \in OB \backslash dcb(\tau_{OB})$: ($\tau_I$, $\tau_O(ob_k)$) and (τ_S, $\tau_O(ob_k)$) are I-O and S-O partition pairs respectively,
(5) $\pi_{OB} \cdot \tau_{OB} = \pi_{OB}(0)$.

If the condition:

(6) $\pi_S \cdot \tau_S = \pi_S(0)$ is also satisfied, then the state behaviour of M will also be realized.

5.2 Combinational Machines

A combinational machine can be considered as a special case of a sequential machine with one state and the trivial next-state function. Therefore, the combinational machine is completely specified by: M=(I,O,λ), because the trivial state information can be eliminated from further consideration. In this way, Theorem 2 is reduced to the following theorem.

Theorem 9. A combinational machine M has a **general full-decomposition** if two partition doubles (π_I, π_I^*) and (τ_I, τ_I^*) exist that satisfy the following conditions:

(1) $\pi_I \cdot \tau_I' \leq \pi_I^*$ and $\tau_I \cdot \pi_I' \leq \tau_I^*$, where: $\tau_I' \geq \tau_I^*$, $\pi_I' \geq \pi_I^*$,
(2) $\pi_I \cdot \tau_I \leq \pi_I'$ and $\pi_I \cdot \tau_I \leq \tau_I'$,
(3) ($\pi_I^* \cdot \tau_I^*$, $\pi_O(0)$) is an I-O partition pair.

For parallel decomposition, Theorem 9 is reduced to the following theorem.

Theorem 10. A combinational machine M has a **parallel full-decomposition** if two partition doubles (π_I, π_I^*) and (τ_I, τ_I^*) exist that satisfy the following conditions:

(1) $\pi_I \leq \pi_I^*$, and $\tau_I \leq \tau_I^*$,
(2) ($\pi_I^* \cdot \tau_I^*$, $\pi_O(0)$) is an I-O partition pair.

For serial decomposition, the following theorem will result from Theorem 9.

Theorem 11. A combinational machine M has a **serial full-decomposition** if partition doubles (π_I, π_I^*) and (τ_I, τ_I^*) exist that satisfy the following conditions:

(1) $\pi_I \leq \pi_I^*$ and $\tau_I \cdot \pi_I' \leq \tau_I^*$, where: $\pi_I' \geq \pi_I^*$,
(2) ($\pi_I^* \cdot \tau_I^*$, $\pi_O(0)$) is an I-O partition pair.

By replacing in Theorem 11 the input partitions π_I and τ_I by the bit partitions π_{IB} and τ_{IB}, the following theorem can be obtained.

Theorem 12. A combinational circuit M has a non-trivial **input-bit parallel full-decomposition** if two partition doubles (π_{IB}, π_I^*) and (τ_{IB}, τ_I^*) exist that satisfy the conditions:

(1) $\pi_I \leq \pi_I^*$ and $\tau_I \leq \tau_I^*$, where: $\pi_I = \text{ind}(\pi_{IB})$, $\tau_I = \text{ind}(\tau_{IB})$,
(2) ($\pi_I^* \cdot \tau_I^*$, $\pi_O(0)$) is an I-O partition pair.

The following theorem can be obtained from Theorem 10 by replacing all partitions by bit partitions.

Theorem 13. A combinational circuit has a non-trivial **bit parallel full-decomposition** if two partition doubles (π_{IB}, π_{OB}) and (τ_{IB}, τ_{OB}) exist that satisfy the conditions:

(1) $\forall ob_k$, $ob_k \in OB \backslash dcb(\pi_{OB})$: ($\pi_I$, $\pi_O(ob_k)$) are I-O partition pairs, where: $\pi_I = \text{ind}(\pi_{IB})$,
(2) $\forall ob_k$, $ob_k \in OB \backslash dcb(\tau_{OB})$: ($\tau_I$, $\tau_O(ob_k)$) are I-O partition pairs, where: $\tau_I = \text{ind}(\tau_{IB})$,
(3) $\pi_{OB} \cdot \tau_{OB} = \pi_{OB}(0)$.

6. USING THE MODEL IN DIGITAL CIRCUIT SYNTHESIS

6.1 Decompositional Logic Synthesis

The aim of synthesis is to provide a circuit structure that realizes the specified behaviour, satisfies certain constraints and optimizes specific objectives.

In general, the constraints and objectives refer to the circuit's performance and how the various resources are used during the whole life-cycle of the circuit. They can be formulated along various dimensions such as time, area, inputs, outputs, power consumption, testability, reliability, maintainability, design time or cost etc.

In our methodology, the behaviour is specified in the form of an original sequential machine or Boolean function and the physical constraints and objectives are modelled as a constrained multi-objective optimization problem.

Decompositional synthesis consists of applying the general decomposition model and theorem, or their special cases, a number of times and in this way, adding the structural information to the design specifications until a directly implementable design description is obtained.

By repetitive use of the general full-decomposition model or its special cases, all possible implementation structures for sequential and combinatorial machines (all meaningful partial machine networks) can be obtained.

The appropriate decomposition theorems guarantee correctness by construction and limit the search for

solutions to the decompositional structures that realize the specified behaviour. The model, together with its theorems, forms a basis for decompositional synthesis. The model information on how the model was used during synthesis and allow us to check the correctness of the synthesis in a relatively easy way—by backward mapping the synthesis result into the specification. In this manner, checking the correctness of the human designer or automatic design tool behaviour is possible (see Section 6.2 for further information and Section 7 for an example).

For small sequential or combinational machines, the optimal decompositions can be found by implicit enumeration, limited only by the properties of the building blocks and the algebraic properties described by the appropriate decomposition theorems and partition pair theory. For large systems, the number of possible decompositions is so great than an implicit exhaustive search, performed using only the algebraic and building block properties, is impossible. It becomes necessary to construct the most promising decompositions using the theorems presented in this paper together with the appropriate heuristics. The heuristic evaluation functions and selection mechanisms must limit the search space to a manageable size and keep high-quality solutions in this limited space.

Our methodology guarantees "correctness by construction", easy post factum correctness verification and satisfaction of all the originally specified constraints. The objectives can be near-optimally satisfied, because the problems at hand are computationally complex and heuristic algorithms must be used.

6.2 Correctness Aspects

Currently, simulation and prototype testing are commonly used to validate designs, but this approach is not sufficient for complex circuits. The techniques of formal validation are more promising and therefore have been applied in our methodology. We will show that it is possible to use them very effectively and efficiently.

Proving correctness consists of providing evidence of the fact that the realization relation holds between an original specification and its implementation. Since synthesis involves adding detailed information to specification, proving correctness must involve the opposite i.e. abstracting from the information and in this way relating the detailed implementation description to its more abstract specification. The correctness-proving process associated with the decompositional logic synthesis is performed as a series of abstraction steps which are used to gradually relate the lower level design descriptions to the immediate higher level specifications, starting from the bottom level implementation and continuing until the original top level specification is reached.

Four types of abstraction are used in this process: structural abstraction (hiding the information about a circuit's internal structure by computing the behavioural description for the composition of partial machines); data abstraction (hiding the information about the implementation of data by replacing the binary data values with their symbolic abstract representations); behavioural abstraction (e.g. leaving unspecified behaviour for certain state/input combinations which will never occur in the operating environment ("don't cares")); and temporal abstraction (relating several units of lower level time to one unit of higher level time, e.g. relating the delays of structural implementation elements to the clock period of a sequential machine).

The structural, behavioural and data abstractions have actually been used to prove the general full-decomposition theorem. This theorem gives the necessary and sufficient conditions which must be fulfilled by each general composition of partial machines, in order to realize the functional behaviour as specified by the original machine. Once proved, the general full-decomposition theorem provides synthesis rules that are problem independent and guarantee functional correctness. The parametric correctness, in the sense of satisfying the hard physical constraints, is achieved with the class specific rules, which are distinct for different classes of target architectures (building blocks). These rules are obtained by modelling the synthesis problem as a multi-objective constrained optimization problem (see Section 6.3). The parametric optimization is guided by the rules that are problem-instance specific. These rules are constructed and selected automatically by the search algorithms, based on information about the characteristic features of a sequential machine related to the characteristics of building blocks and optimization aims [29]–[32].

Many researchers and designers are convinced that "correctness by construction" makes post-factum verification unnecessary. This is not true. Even if the construction rules are proved to be correct, their application can be faulty due to mistakes made by designers or errors in the synthesis tools.

The physical constraints are verified by estimating the parameters involved by using the abstract modelling, the lower level synthesis tools, or simulation, and checking the estimates against the constraints. Verification of the near-optimal satisfaction of the objectives consists of checking the performance of the synthesis tools by using benchmarking and statistical analysis of the synthesised designs [29]–[32]. The functional correctness is checked by repeatedly applying two elementary verification processes:

- checking of the correctness of each particular decomposition (transformation) computed by the syn thesis tools, i.e. verifying whether the proposed

system of partitions and associated state, input, and output mappings satisfy the conditions of the general full-decomposition theorem, and
- checking of if each particular planned decomposition has been applied successfully; this is performed by reverse mapping of the decompositional implementation structure into its specification.

In general, design verification is a complex process because one does not know the sequence of transformations which have to be performed, in order to show that an implementation satisfies its specification. In our methodology verification is simple, because the sequence of transformations results from the information produced during synthesis. If information about synthesis transformations to be performed is memorized, then finding the reverse transformations and performing the reverse mapping is very easy (see an example in Section 7). In place of verifying that a certain implementation is a realization of a given specification, as done by traditional verification methods, we check if the specific planned decompositional structure is correct and then we prove that the synthesised implementation is the planned realization of the specification. This results in a very efficient verification process. Of course, it requires prior knowledge of the class of correct structures and knowledge of decisions made for selecting a certain structure from the class of correct structures. The first part of the required knowledge is general and it is given in the form of a general full-decomposition model and its theorem proven in this paper. The second part of the knowledge is problem instance dependent; however, it must be found before constructing the required decomposition. Therefore, the only extra activity to be performed to enable the reverse mapping is keeping a record of the decisions taken during the construction of a certain decompositional structure, i.e. recording what instance of the model is intended to be used. This record can be kept in terms of (partial) machines and appropriate mapping functions. Since the highest level record represents the original machine and lowest level the resulting realization structure, the extra information recorded is limited to the tables of partial machines from the intermediate levels and appropriate mapping functions.

Since verification processes are performed by using reverse operators to those used during synthesis, the probability of masking the synthesis faults, by faults during verification, is negligible. Therefore, the verification performed in this way is very reliable. Synthesis faults can be rapidly detected and localized because the elementary verification processes can be immediately performed after finding or applying an elementary transformation.

The post factum verification by reverse mapping is much more efficient than a verification by traditional verification methods which do not use information from the synthesis. The time savings result from the fact that it is no longer required to find the sequence of the verification transformations, because this sequence is unambiguously defined by the sequence of the synthesis transformations. Therefore, the verification time is composed exclusively of time for performing the reverse transformations (which is comparable to time for performing synthesis transformations) and time for comparing the original specification with the result of the reverse mapping (which is proportional to the dimensions of the specification). Verification by reverse mapping is also much easier than proving correctness for the complex software of the synthesis tool and ensuring the correct functioning of its hardware. It is equivalent to showing that the synthesis tool has performed correctly for a particular case. An example of verification by reverse mapping can be found in Section 7.

The principle of reverse mapping is very general. It can be applied for off-line and on-line correctness verification of various systems in all cases where forward transformations are known. In particular, it can be applied for design verification of any kind of transformational design.

6.3 Search for Optimal Solutions

For large sequential or combinatorial systems, an exhaustive search for optimal decompositions is impossible. It is necessary to construct only the most promising decompositions when using heuristic search algorithms, and to choose the best of these. In our previous publications [29]–[32], some specific heuristic decomposition algorithms have been described and benchmark results from their software implementation have been presented. This section aims to describe the underlying principles of those algorithms. These principles can be used for construction of the heuristic search algorithms for various decomposition problems.

A specific decomposition problem can be modelled as a special multi-objective constrained optimization problem (MOCOP) [29][30][32]. A MOCOP can be characterized by four sets of components: a set $V = (V_1, V_2, \ldots V_n)$ of variables, a set $D = (D_1, D_2, \ldots D_n)$ of domains for the variables, a set $C = (C_1, C_2, \ldots C_k)$ of constraints, and a set $O = (O_1, O_2, \ldots O_p)$ of objectives. Each domain D_i: $D_i \in D$ represents a finite or infinite set of values. A constraint C_i: $C_i \in C$ is a m-ary relation on m of the n variables V_i: $V_i \in V$ and it can be viewed as a mapping from D to the set {T, F} (T = "true", F = "false"). An objective O_i: $O_i \in O$ is a function on the variables from V with values being real numbers, i.e. a function from D to R (R—the set of real numbers). To solve a MOCOP is to find an assignment of values to

variables V_j, j = 1...n, so that all constraints C_i, i = 1...k, are satisfied. Since a MOCOP involves several objectives, trade-off between them is possible. Therefore, the complete formulation of a MOCOP must include trade-off information. This trade-off information can be formulated in several different ways: as an order of the objectives, utility function, ranking information, local preference for small changes in values of the objectives etc. [43]. The choice of a specific formulation depends on the particular problem. To solve a MOCOP optimally is to solve it in such a way that the most preferred solution is found according to the objectives O_i, i = 1..p, and the actual trade-off information. When modelling the structural decomposition problems, the particular variables and domains correspond to various structural attributes (dimensions) of the implementation architecture such as inputs, outputs, memory elements, functional elements, interconnections, etc., and their possible values. The objectives and constraints are the functions and relations of those structural dimensions, such as the number of inputs (outputs, memory elements, etc.), area, speed, etc. or the limits imposed on such characteristics.

The solution of the MOCOP model of a specific decomposition problem can be found by defining a set of some elementary component machines (atomic computations), constructing a certain partition on the set of the elementary component machines and implementing each partition block as a separate component machine. If the model of a certain decomposition problem involves constraints, it can be solved by special multidimensional packing algorithms [29][30]. Models without hard constraints can be solved by special multi-objective clustering algorithms [32]. The partitioning process which produces partitions on the set of elementary component machines is preceded by analysis of characteristic features of an original machine related to the characteristics of building blocks. In particular, the input, output, and state information and their interrelations are analysed. This analysis enables us to distinguish elementary component machines and to characterize them and their correlations. Its results are used to guide the heuristic packing or clustering processes.

The partitioning problem is represented in terms of a space of states, where each state corresponds to a particular (partial) solution. A (partial) solution consists of a (partially constructed) partition. The tree of (partial) solutions has the form of an implicit tree, i.e. it is defined only by means of an initial state, the rules for generating the tree and the termination criteria. The rules describe how to generate successors to each partial solution (i.e. they define move operators that are (partial) mappings from states to states). Any state that meets a termination criterion is called a goal state. Partitions in packing algorithms are constructed by putting unallocated elements successively into the partition blocks [29][30]. Clustering algorithms construct the successor partitions by merging some of the partition blocks [32]. Heuristic algorithms are used to select the most promising partial solutions and to develop them further when applying only the best move operators.

A heuristic search algorithm can be effective and efficient, if it is able to appropriately compose a broad search of a solution space in many promising directions with a fast convergence to the (near-)optimal solutions. The fast convergence can result from using the knowledge cumulated in the previous search steps for selecting the most promising (partial) solutions and move operators.

A special heuristic double beam-search algorithm has been developed by us in order to efficiently construct a limited set of "sub-optimal" partitions. A **beam-search** is a variation of breadth-first search, where only a limited number of the most promising alternatives are explored in parallel. Beam-search requires two data structures: **current states** (that contain the set of states which have been constructed earlier and are considered to be extended presently), and **candidate states** (that contain the set of states which are being created as a direct extension of the current states). A third data structure, **final states**, contains the states with completed partitions.

Our **double beam algorithm** uses two **selection mechanisms**: Select Moves and Select States. Move operators are evaluated and selected in relation to a certain state (dynamically). Only a few of the most promising move operators are chosen for a certain state from current states by **Select Moves**, using a number of choice strategies and heuristic evaluation functions. By applying the selected move operators for each of the current states, a new set of candidate states is created. The work of the second selection mechanism **Select States,** is twofold: it scans the candidate states for completed partitions in order to include them into the final states and it examines the rest of the candidate states in order to include the best of them into the (new generation of) current states. The beam-search algorithm stops if the set of current states is empty.

The selection mechanisms Select Moves and Select States must ensure that a solution that violates the hard constraints will not be constructed and they will try to satisfy the objectives optimally by limited expenditure of computation time and memory space. In order to fulfil the first task, Select Moves will select only those move operators which, applied to a given state, do not lead to the violation of hard constraints. In order to fulfil the second task, Select Moves and Select States will select a number of the most promising operators or states, respectively, by using the estimations provided by some heuristic evaluation functions. The selection mechanisms and evaluation functions determine together the extent of the search and quality of the results.

The selection mechanisms use heuristic elaborations of one coherent decision rule: "in each state of the search, take a decision which has the greatest chance of leading to the optimal solution, i.e. a decision which is most certain according to the estimations given by the heuristic evaluation functions". If there are more decisions of the same or comparable quality, a number of them will be tried in parallel (beam-search).

According to the above rule, Select Moves will apply those move operators which maximize the choice certainty in a given current state and it will leave the operators which are open to doubts for future consideration. Since information contained in the partial solutions and used by the evaluation functions grows with the progress of computations, the uncertainty related to operators decreases. In each computation, Select Moves will maximize the conditional probability that the application of a certain move operator to a certain solution state leads to the optimal complete solution. Under this condition, it will maximize the growth of the information in the partial solution, which will then be used in the successive computations steps in order to estimate the quality of choices. The quality of the operator Q(op) is decided by these two factors.

Select Moves is controlled by two parameters: MAXMOVES (the maximum number of operator alternatives explored) and OQFACTOR (the quality factor for operators). Select Moves selects no more than MAXMOVES of the highest quality operators op, so that:

$$Q(op) \geq OQFACTOR * Qop_{max}$$

where Qop_{max} is the quality of the best alternative operator. Poor quality alternatives are not taken into account.

In addition to selecting the final states, the task of Select States is to choose the most promising candidate states for a new generation of current states. Select States is controlled by two parameters: MAXSTATES (the maximum number of state alternatives explored) and SQFACTOR (the state quality factor). Select States selects no more than MAXSTATES of the highest quality alternative states PS, for which:

$$Q(PS) \geq SQFACTOR * Q_{max}$$

where Q(PS) denotes the quality of an alternative PS and Q_{max} denotes the quality of the best alternative. Poor quality alternatives are not taken into account. Q(PS) can be computed by cumulating the qualities of the choices of operators that took place during the construction of PS and prediction of the quality of the best possible future choices on the way to the complete solution. Another possibility consists of predicting the quality of the best complete solution that can be achieved from a certain present state PS.

Generally, operators and partial solutions are estimated with some uncertainty. This uncertainty decreases with the progress of computations, because both the "sure" information contained in partial solutions and the quality of prediction grow with this progress. In the first phase of the search, the choices of operators can be done with much more certainty than the choices of partial solutions. In this phase, partial solutions almost do not exist or, in other words, they are far from being complete solutions and almost anything can happen to them on the way to achievable complete solutions.

Therefore in the first phase, the search should be performed almost exclusively based on the choices of operators and, with the progress of computations, more and more on the choices of partial solutions. In our algorithm, this is achieved by giving a relatively low value to MAXMOVES compared to MAXSTATES and a relatively high value to OQFACTOR compared to SQFACTOR.

Since the uncertainty of estimations decreases with the progress of computations, MAXMOVES and MAXSTATES can decrease and OQFACTOR and SQFACTOR can increase with the progress of computations, increasing the search efficiency in this way.

In the method described above, the double beam-search allows for effective and efficient decision-making under changing uncertainty.

In the first search phase the algorithm is divergent to high degree, i.e. a large number of the most promising directions in the search space are tried. In the second phase, when it is already possible to estimate the search directions and operators with a relatively high degree of certainty, the search becomes more and more convergent. The highly divergent character of the search in the first phase, composed with the continuous interplay between the partial solutions in the second phase, result in a global character of the double-beam algorithm.

The search method presented was implemented in a number of decomposition and state assignment programs and when tested on benchmarks, it efficiently produced very good results [29][30][31].

Of course, it is possible to use the solutions found by our constructive double-beam algorithm as good initial solutions for the search algorithms that perform search in the space of complete solutions (e.g. for local searches, simulated annealing, tabu search, or genetic algorithms). However, this was not necessary in the tested cases, because the double-beam constructed the strictly optimal solutions [29][30].

Complex multiple general decomposition problems can be solved by decomposing them into systems of more specific subproblems, which are easier to solve than the original problem, and then solving the systems of subproblems by using systems of cooperating subproblem-specific algorithms.

In this section, we have discussed only some very general principles of searching for the optimal decompositions. For each particular decomposition problem, the problem specific features should be used in order to distinguish the elementary component machines (atomic computations) and to perform the partitioning processes effectively and efficiently. In this way the generic packaging or clustering algorithms will be transformed into some problem specific algorithms. For example, in the case of a traditional two-level AND-OR decomposition of Boolean functions, the atomic computations can be defined as computations of minterms, the partial machines will be limited to AND circuits which will be able to compute product terms, and the output decoder will be limited to be an OR circuit. In this case, the decomposition problem can be viewed as clustering of minterms into larger terms. The aim will be to find the minimal number of clusters (terms, partial machines), that realize all the atomic computations (minterms).

7. EXAMPLE

The aim of the example is to illustrate the use of the proposed decompositional synthesis methodology for logic synthesis and correctness verification.

Consider a Moore machine M defined by the Table I and the following partitions on M:

- state partitions:
  $\pi_1 = \{\overline{1,3,5;2,4,6}\} = \{A,B\}$,
  $\pi_2 = \{\overline{1,2;3,4;5,6}\} = \{C,D,E\}$,
  $\pi_{2,1} = \{\overline{1,2,3,4;5,6}\} = \{F,G\}$,
  $\pi_{2,2} = \{\overline{1,2;3,4,5,6}\} = \{H,K\}$,
- input bit partition: $\pi_{IB1} = \{\overline{X_1};(\overline{X_2})\}$,
- output bit partitions:
  $\pi_{OB1} = \{\overline{y_1};(\overline{y_2,y_3})\}$,
  $\pi_{OB2} = \{\overline{y_2};(\overline{y_1,y_3})\}$,
  $\pi_{OB3} = \{\overline{y_3};(\overline{y_1,y_2})\}$,
  $\pi_{OB2,3} = \{\overline{y_2;y_3};(\overline{y_1})\}$.

The above partitions are found by analysing the information flow in the machine M; in particular, by finding the partitions describing the minimal state information necessary for computing the values of particular output bits (the $M_{S\text{-}O}(\pi_O(y_i))$ partitions for particular π_{OBi} partitions, i = 1, 2, 3: π_1, π_{21}, π_{22}) and then by finding the partitions describing the state and input information necessary to compute the previously found state information (π_1, π_{21}, π_{22}, π_{I1}, $\pi_I(0)$).

For the above partitions, the following statements are true:

(1) π_1 and π_2 are SP-partitions,
(2) $\pi_1 \cdot \pi_2 = \pi_S(0)$,
(3) (π_{I1}, π_1) is an I-S partition pair, where $\pi_{I1} = \text{ind}(\pi_{IB1}) = \pi_I(x_1)$,
(4) $(\pi_1, \pi_O(y_1))$ is an S-O partition pair,
(5) $(\pi_2, \pi_O(y_2))$ and $(\pi_2, \pi_O(y_3))$ are an S-O partition pairs,
(6) $\pi_{OB1} \cdot \pi_{OB2,3} = \pi_{OB}(0)$,
(7) $\pi_{2,1} \cdot \pi_{2,2} = \pi_2$,
(8) $(\pi_{2,1}, \pi_O(y_2))$ is an S-O partition pair,
(9) $(\pi_{2,2}, \pi_O(y_3))$ is an S-O partition pair,
(10) $\pi_{OB2} \cdot \pi_{OB3} = \pi_{OB2,3}$.

Since conditions (1)-(6) are fulfilled, the conditions of Theorem 8 are satisfied. Therefore, it is possible to construct a parallel bit full-decomposition of M into M_1 and M_2 with the state and output behavior realization as shown in Figure 14.

Since conditions (7)-(8) are fulfilled, the conditions of Theorem 7 are satisfied. So, it is possible to decompose M_2 further into $M_{2,1}$ and $M_{2,2}$, by constructing a general output-bit full-decomposition with the state and output behaviour realization as in Figure 14. The next-state and output tables of the partial machines from Figure 14 and the state decoding functions Φ' and Φ_2' are given in Tables II-VII.

Since each of the partial machines M_1, $M_{2,1}$ and $M_{2,2}$ has two states, only one two-state memory element is required for implementing the state memory for each of them. If D flip-flops are used and if we assign the binary state values according to the output values of each partial machine, then, the output logic will be reduced to identity functions and the following assignments and excitation functions will result (Tables VIII-X). Twice performed decomposition results in a state assignment for M.

Designing the decompositional implementation of M can proceed further with combinational logic synthesis and layout synthesis in order to optimize the specified excitation functions and to find an optimal layout. The combinational logic synthesis can also apply the decompositional paradigm. Since the decompositional realization of M, as given in the Tables II-X, has been obtained when using the previously proven correct ways of construction (Theorems 7 and 8), this realization is correct. However, it is correct if those correct ways of construction have been actually applied and not only planned to be used, i.e. if the human designer or automatic tool which actually applied these theorems performed fault-free. Since this cannot be guaranteed, the actual application of the correct construction must be checked for correctness.

Applying the concept of reverse mapping, it is possible to make a straightforward check whether the designed composition of partial machines realizes the specified behaviour of M.

In the first step, the decompositional structure of M_2 is mapped into its specification. From the tables of $M_{2,1}$ and $M_{2,2}$ (Tables V and VI), the table of the composition

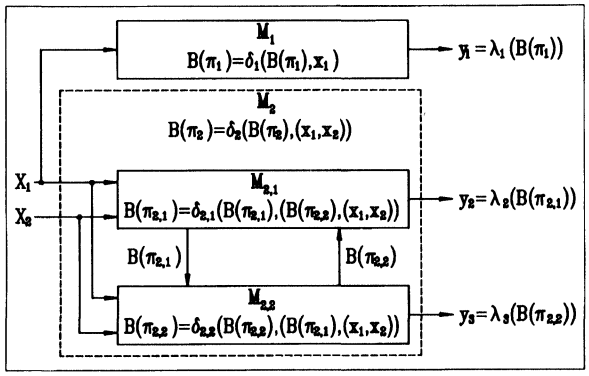

FIGURE 14 Decompositional realization structure for M.

machine $M_{2,1} \leftrightarrow M_{2,2}$ (Table XI) is computed and then the table for M_2 (Table III) is obtained by applying the state decoding function Φ'_2: $S_{2,1} \times S_{2,2} \rightarrow S_2$ (Table VII) to the table of the composition machine $M_{2,1} \leftrightarrow M_{2,2}$. The second step consists of mapping the composition of M_1 and M_2 into the specification of M. It can be performed in precisely the same way as the first step. Since the decomposition was correct, the reverse mapping did not uncover any faults.

Let us now introduce a fault, for example by replacing the correct state F in the first row and third column of the table of $M_{2,1}$ (Table V) by the faulty state G. In this way, we obtain Table XII. The table of the faulty composition machine $M_{2,1} \leftrightarrow M_{2,2}$ is shown as Table XIII. If we now apply the state decoding function Φ'_2: $S_{2,1} \times S_{2,2} \rightarrow S_2$ to Table XIII, we will obtain the table for M_2 (Table XIV). Simple comparison of the table for M_2 obtained from the reverse mapping (Table XIV) with the table for M_2 being the specification of M_2 obtained during the synthesis process (Table III) uncovers the fault. In one table in the first row and the third column state "C" appears and in the second "-" appears in the same place.

TABLE II
M_1

$S_1 \backslash x_1$	0	1	y_1
A	B	A	0
B	A	B	1

TABLE III
M_2

$S_2 \backslash x_2x_1$	00	01	10	y_3y_2
C	E	D	C	00
D	C	E	D	10
E	D	C	E	11

TABLE IV
Φ':$S_1 \times S_2 \rightarrow S$

$S_1 \backslash S_2$	C	D	E
A	1	3	5
B	2	4	6

TABLE V
$M_{2,1}$

$S_{2,2} \times x_2x_1$ / $S_{2,1}$	(H,00)	(H,01)	(H,10)	(K,00)	(K,01)	(K,10)	y_2
F	G	F	F	F	G	F	0
G	—	—	—	F	F	G	1

TABLE VI
$M_{2,2}$

$S_{2,1} \times x_2x_1$ / $S_{2,2}$	(F,00)	(F,01)	(F,10)	(G,00)	(G,01)	(G,10)	y_3
H	K	K	H	—	—	—	0
K	H	K	K	K	H	K	1

TABLE VII
Φ_2': $S_{2,1} \times S_{2,2} \rightarrow S_2$

$S_{2,1} \backslash S_{2,2}$	H	K
F	C	D
G	—	E

8. CONCLUSIONS

Implementation of a sequential machine or a Boolean function requires finding the composition of some structural elements, which allows realization of the input-output behaviour specified by a certain machine or function, and which satisfies a certain set of constraints and objectives. In general, the problem of finding an optimal implementation remains unsolved. Only the special case of two-level logic and unconstrained minimization (in the sense of the minimal term cover), can be processed by the exact techniques for designs up to about 20 inputs [11] and by the nearly optimal heuristic techniques for larger designs [6]. Constrained optimization or optimization for objectives other than minimum term cover remained unsolved even for the two-level logic structures which after all, form a small sub-class of all the possible implementation structures.

Contrary to traditional logic design methods, which deal with some very specific implementation structures, the decomposition methodology presented enables the modelling and construction of a large class of functionally correct digital circuit structures, which includes all structures known from the literature as its special cases. The decomposition theory presented shows that the structures tackled by the traditional logic synthesis methods form only a small subset of possible implementation structures. The traditional methods can of course, be used for solving the specific decomposition problems they were developed for (at least, if they solve them effectively and efficiently); however, they allow us to exploit

TABLE VIII
M_1 (D_1)

$S_1\backslash x_1$	0	1	y_1
A-0	1	0	0
B-1	0	1	1

TABLE IX
$M_{2.1}$ (D_2)

$S_{2.2}x_2x_1$ $S_{2.1}$	000	001	010	100	101	110	y_2
F-0	1	0	0	0	1	0	0
G-1	—	—	—	0	0	1	1

TABLE X
$M_{2.2}$ (D_3)

$S_{2.1}x_2x_1$ $S_{2.2}$	000	001	010	100	101	110	y_3
H-0	1	1	0	—	—	—	0
K-1	0	1	1	1	0	1	1

TABLE XI
$M_{2.1} \leftrightarrow M_{2.2}$

$S_{2.1} \times S_{2.2}\backslash x_2x_1$	00	01	10	y_3y_2
(F,H)	(G,K)	(F,K)	**(F,H)**	00
(F,K)	(F,H)	(G,K)	(F,K)	10
(G,K)	(F,K)	(F,H)	(G,K)	11

TABLE XII
$M_{2.1}$ with fault

$S_{2.2} \times x_2x_1$ $S_{2.1}$	(H,00)	(H,01)	(H,10)	(K,00)	(K,01)	(K,10)	y_2
F	G	F	**G**	F	G	F	0
G	—	—	—	F	F	G	1

TABLE XIII
$M_{2.1} \leftrightarrow M_{2.2}$

$S_{2.1} \times S_{2.2}\backslash x_2x_1$	00	01	10	y_3y_2
(F,H)	(G,K)	(F,K)	**(G,H)**	00
(F,K)	(F,H)	(G,K)	(F,K)	10
(G,K)	(F,K)	(F,H)	(G,K)	11

TABLE XIV
Faulty M_2

$S_2\backslash x_2x_1$	00	01	10	y_3y_2
C	E	D	-	00
D	C	E	D	10
E	D	C	E	11

only a small part of the opportunities created by modern microelectronic technology. The decompositional logic synthesis enables us to deal efficiently with design complexity and characteristic features of modern building blocks. It makes possible extensive examination of the solution space and allows exploitation of the machine's structural features in relation to a given set of objectives and constraints. Additionally, the partial circuits are smaller than the original ones and easier to design, optimize, implement and test. The decomposition methodology presented ensures "correctness by construction" and solves very effectively and efficiently the problem of design validation. As we have shown, "correctness by construction" and post-factum verification are two complementary validation approaches which should be used jointly during the design process. Designing with previously proven constructions can make not only synthesis but also post-factum verification much easier. The structural knowledge generated during the synthesis process is used for post-factum verification, raising its efficiency enormously. An appropriate structural knowledge in an appropriate form and appropriately applied is the main reason for an extremely efficient verification process using our methodology.

Many empirical data show superiority of the decompositional implementation regarding area, speed, etc. [1][2][3][8][9][12][13][26][35][39][40][44][45][46]. For example, Łuba et al. [35] applied the input-bit parallel decomposition to combinational circuits and saved on average more than 50% silicon area compared to traditional implementation, and the area saved was as high as 90% in one example. Similar results were reported in [8][9] and [46]. Decomposition results for sequential machines show that the overall silicon area for the decomposed machine is often much smaller than the area of the best monolithic implementation, and the component machines are always smaller than the original ones [2][26][39][40].

Recently, there has been substantial progress in developing efficient heuristic methods for various special cases of general decomposition and for different optimization objectives and constraints [9][29][30][32][37][44][45]; however, no overall method is yet available for efficient exploration of the entire space of multiple general decompositions in a systematic manner. This important research area remains open.

APPENDIX: THE PROOF OF THEOREM 1

First, it will be shown how to construct a general full-decomposition that will realize the output behaviour of M.

Let $M_i = (\pi_I^i \times \pi'_{S\times I}{}^i, \pi_S^i, \pi_{S\times I}^i, \delta^i, \lambda^i)$ be a component sequential machine for which the following conditions are satisfied:

(6) $(\pi_I^i, \pi_S^i, \pi_{S\times I}^i)$ satisfy the conditions of Theorem 1,

(7) $\forall A^i \in \pi_I^i \; \forall C'^i \in \pi'_{S\times I}{}^i \; \forall B^i \in \pi_S^i$:

$\delta^i(B^i, (A^i, C'^i)) = [\{\delta(s,x) \mid (s,x) \in \text{ind}^S(B^i) \wedge (s,x) \in \text{ind}^I(A^i) \wedge (s,x) \in C'^i\}]\pi_S^i$

$\lambda^i(B^i,(A^i,C'^i)) = [\{(s,x) \mid (s,x) \in \mathrm{ind}^S(B^i) \wedge (s,x) \in \mathrm{ind}^I(A^i) \wedge (s,x) \in C'^{i'})\}]\pi_{S\times I}{}^i$

Since the conditions (1) and (2) of Theorem 1 are satisfied,

$\{\delta(s,x) \mid (s,x) \in \mathrm{ind}^S(B^i) \wedge (s,x) \in \mathrm{ind}^I(A^i) \wedge (s,x) \in C'^i)\}$

$\{(s,x) \mid (s,x) \in \mathrm{ind}^S(B^i) \wedge (s,x) \in \mathrm{ind}^I(A^i) \wedge (s,x) \in C'^i)\}$

are included in just one block of $\pi_S{}^i$ and $\pi_{S\times I}{}^i$, respectively. This means, that $\delta^i(B^i,(A^i,C'^i))$ and $\lambda^i(B^i,(A^i,C'^i))$ are defined unambiguously, i.e. M_i are well-defined deterministic sequential machines which are able to compute their next-states and outputs from their present-states and inputs.

Let Con_i: $\times \pi_{S\times I}{}^j \to \pi'_{S\times I}{}^i$ be a connection rule of the component machine i and

(8) $\mathrm{Con}_i(C^1,\ldots,C^n) = [\{(s,x) \mid (s,x) \in C^1 \wedge \ldots \wedge (s,x) \in C^n\}]\pi'_{S\times I}{}^i$ (for the maximally pre-processed form) or Con_i: $\times \pi_{S\times I}{}^j \to \times \pi_{S\times I}{}^{j,i}$ be a connection rule of the component machine i and

(9) $\mathrm{Con}_i(C^1,\ldots,C^n) = ([\{(s,x) \mid (s,x) \in C^1\}]\pi_{S\times I}{}^{1,i},\ldots, [\{(s,x) \mid (s,x) \in C^n\}]\pi_{S\times I}{}^{n,i})$ (for the canonical form).

The definitions of connection rules for the partially pre-processed forms, lying between these two extremes, should be obvious.

Since $\pi'_{S\times I}{}^i = \prod_{j=1,\ldots,n} \pi_{S\times I}{}^{j,i}$ and $\pi_{S\times I}{}^{j,i} \geq \pi_{S\times I}{}^j$, $1 \leq i,j \leq n$, $(s,x) \in C^j$ are included in just one block of $\pi'_{S\times I}{}^i$. This means that the connection rules Con_i are in the both cases defined unambiguously. It is clear that the above construction of the machines M_i and connection rules C_i is a general composition of n sequential machines M_i. Since the condition (3) is satisfied, it is possible to construct the general composition of M_i as a legal composition. It is proved below that a legal general composition of the machines M_i defined above realizes the output behaviour of M.

Let: Ψ: $I \to \times \pi_I{}^i$ be a function, Φ: $S \to \times \pi_S{}^i$ be a function and Θ: $\times \pi_{S\times I}{}^i \to O$ be a surjective partial function, and

(10) $\Psi(x)$: $([x]\pi_I{}^1,\ldots,[x]\pi_I{}^n)$,
(11) $\Phi(s)$: $([s]\pi_S{}^1,\ldots,[s]\pi_S{}^n)$,
(12) $\Theta(C^1,\ldots,C^n) = \overline{\lambda}(C^1 \cap \ldots \cap C^n)$ if $C^1 \cap \ldots \cap C^n \neq \emptyset$.

Since $\left(\prod_i \pi_{S\times I}{}^i, \pi_O(0)\right)$ is an S × I-O partition pair (4), Θ is a surjective function on the subset of $\times \pi_{S\times I}{}^i$ which is mapped onto O and, therefore, for $C^1 \cap \ldots \cap C^n \neq \emptyset$:

(13) $\Theta(C^1,\ldots,C^n) \in O$.

So, $\forall s \in S\ \forall x \in I$:

$$\delta_{GC}(\Phi(s),\Psi(x)) =$$
$$= \delta_{GC}([s]\pi_S{}^1,\ldots,[s]\pi_S{}^n,[x]\pi_I{}^1,\ldots,[x]\pi_I{}^n)) = \quad (10)(11)$$
$$= (\delta^1([s]\pi_S{}^1, ([x]\pi_I{}^1,[(s,x)]\pi'_{S\times I}{}^1)),\ldots,$$
$$\delta^n([s]\pi_S{}^n, ([x]\pi_I{}^n,[(s,x)]\pi'_{S\times I}{}^n))) =$$
Def. 1, Def. 2, (7),(8)
$$= (\delta^1([s]\pi_S{}^1, ([x]\pi_I{}^1, ([(s,x)]\pi_{S\times I}{}^{1,1},\ldots,$$
$$[(s,x)]\pi_{S\times I}{}^{n,1}))),\ldots,$$
$$\delta^n([s]\pi_S{}^n, ([x]\pi_I{}^n, ([(s,x)]\pi_{S\times I}{}^{1,n},\ldots,$$
$$[(s,x)]\pi_{S\times I}{}^{n,n})))) = \quad \text{Def.1, Def.2, (7),(8),(9)}$$
$$= ([\{\delta(t,z) \mid (t,z)\in \mathrm{ind}^S([s]\pi_S{}^I) \wedge (t,z)\in \mathrm{ind}^I([x]\pi_I{}^1) \wedge$$
$$\wedge (t,z)\in[(s,x)]\pi_{S\times I}{}^{1,1} \wedge \ldots \wedge$$
$$(t,z)\in[(s,x)]\pi_{S\times I}{}^{n,1}\}]\pi_S{}^1,\ldots,$$
$$[\{\delta(t,z) \mid (t,z)\in \mathrm{ind}^S([s]\pi_S{}^n \wedge (t,z)\in \mathrm{ind}^I([x]\pi_I{}^n \wedge$$
$$\wedge (t,z)\in[(s,x)]\pi_{S\times I}{}^{1,n} \wedge \ldots \wedge$$
$$(t,z)\in[(s,x)]\pi_{S\times I}{}^{n,n}\}]\pi_S{}^n = \quad (7)$$
$$= ([\{\delta(t,z) \mid (t,z)=(s,x)\}]\pi_S{}^1,\ldots,$$
$$[\{\delta(t,z) \mid (t,z)=(s,x)\}]\pi_S{}^n) = \quad (1),(3),(6)$$
$$(s,x)\in \mathrm{ind}^S([s]\pi_S{}^i)$$
$$(s,x)\in \mathrm{ind}^I([x]\pi_I{}^i)$$
$$(s,x)\in[(s,x)\pi_{S\times I}{}^{ij}$$
$$= ([\delta(s,x)]\pi_S{}^1,\ldots,[\delta(s,x)]\pi_S{}^n =$$
$$= \Phi(\delta(s,x)) \quad (9)$$

and similarly:

$$\Theta(\lambda_{GC}\Phi((s),\Psi(x))) =$$
$$= \Theta(\lambda_{GC}([s]\pi_S{}^1,\ldots,[s]\pi_S{}^n),$$
$$([x]\pi_I{}^1,\ldots,[x]\pi_I{}^n))) = \quad (10),(11)$$
$$= \Theta(\lambda_1([s]\pi_S{}^1, ([x]\pi_I{}^1,[(s,x)]\pi'_{S\times I}{}^1)),\ldots,$$
$$\lambda^n([s]\pi_S{}^n, ([x]\pi_I{}^n, [(s,x)]\pi'_{S\times I}{}^n))) =$$
Def.1, Def.2, (7),(8)
$$= \Theta(\lambda^1([s]\pi_S{}^1, ([x]\pi_I{}^1, ([(s,x)]\pi_{S\times I}{}^{1,1},\ldots$$
$$[(s,x)]\pi_{S\times I}{}^{n,1}))),\ldots,$$
$$\lambda^n([s]\pi_S{}^n,([x]\pi_I{}^n, ([(s,x)]\pi_{S\times I}{}^{1,n},\ldots,[(s,x)]\pi_{S\times I}{}^{n,n}))))$$
$$= \quad \text{Def.1, Def.2, (7),(8),(9)}$$
$$= \overline{\lambda}(\lambda^1([s]\pi_S{}^1, ([x]\pi_I{}^1, ([(s,x)]\pi_{S\times I}{}^{1,1},\ldots,$$
$$[(s,x)]\pi_{S\times I}{}^{n,I}))) \cap \ldots \cap$$
$$\cap \lambda^n([s]\pi_S{}^n, ([x]\pi_I{}^n, ([s,x)]\pi_{S\times I}{}^{1,n},\ldots,$$
$$[(s,x)]\pi_{S\times I}{}^{n,n})))) = \quad (12)$$
$$= \overline{\lambda}([\{(t,z) \mid (t,z)\in \mathrm{ind}^S([s]\pi_S{}^1) \wedge (t,z)\in \mathrm{ind}^S([x]\pi_I{}^1) \wedge$$
$$\wedge (t,z)\in[(s,x)]\pi_{S\times I}{}^{1,1} \wedge \ldots \wedge (t,z)\in$$
$$[s,x)]\pi_{S\times I}{}^{n,I}\}]\pi_{S\times I'} \cap \ldots \cap$$
$$\cap [\{(t,z) \mid (t,z)\in \mathrm{ind}^S([s]\pi_S{}^n) \wedge (t,z)\in \mathrm{ind}^S([x]\pi_I{}^n) \wedge$$
$$\wedge (t,z)\in[(s,x)]\pi_{S\times I'}{}^{1,n} \wedge \ldots \wedge$$
$$(t,z) \in[(s,x)]\pi_{S\times I}{}^{n,n}\}]\pi_{S\times I}{}^n) = \quad (7)$$

$$= \overline{\lambda}([\{(t,z) \mid (t,z)=(s,x)\}]\pi_{S\times I}{}^{I} \cap \ldots \cap [\{(t,z) \mid (t,z)=(s,x)\}]\pi_{S\times I}{}^{n}) =$$

(1),(3),(6): $(s,x)\in ind^{S}([s]\pi_{S}{}^{I})$, $(s,x)\in ind^{I}([x]\pi_{I}{}^{i})$, $(s,x)\in[(s,x)]\pi_{S\times I}{}^{ij}$

$$= \lambda(s,x) \quad (4),\ (13)$$

From the calculations above, the definition of the output behaviour realization, and the definitions 2 and 3, it directly follows that the general composition of n component machines M_i will realize the output behaviour of M. If condition (5) of Theorem 1 is satisfied, then: $\bigcap_i [s]\pi_S{}^i = \{s\}$.

Therefore, it is possible to define such a surjective partial function Φ' that:

(14) Φ': $\pi_S{}^i \rightarrow S$ and $\Phi'(B^1,\ldots,B^n) = B^1 \ldots B^n$ if $B^1 \cap \ldots \cap B^n \neq \emptyset$.

Then $\forall(B^1,\ldots,B^n)$: $B^i \in \pi_S{}^i \wedge B^i \bigcap_i B^i \neq \emptyset$ (i.e. $\forall s \in S$) and $\forall x \in I$:

$$\Phi'(\delta_{GC}((B^1,\ldots,B^n, \Psi(x))) = \Phi'(\delta_{GC}((B^1,\ldots,B^n), ([x]\pi_I{}^1,\ldots,[x]\pi_I{}^n))) \quad (10)$$

$$= \Phi'(\delta^1(B^1, ([x]\pi_I{}^1,[(s,x)]\pi'_{S\times I}{}^1)),\ldots, \delta^n(B^n, ([x]\pi_I{}^n,[(s,x)]\pi'_{S\times I}{}^n))) =$$

Def.1, Def.2,(7),(8)

$$= \bigcap_i \delta^i(B^i, ([x]\pi_I{}^i,[(s,x)]\pi'_{S\times I}{}^i)) = \quad (14)$$

$$= \bigcap_i [\{\delta(t,z) \mid (t,z)\in ind^{S}(B^i) \wedge (t,z)\in ind^{I}([x]\pi_I{}^i) \wedge (t,z)\in[(s,x)]\pi'_{S\times I}{}^i\}]\pi_S{}^i = \quad (7)$$

$$= \bigcap_i [\delta(s,x) \mid (s,x)\in(\bigcap_i B^i,x)]\pi_S{}^i = \quad (3),\ (5)$$

$(\bigcap_i B^i,x)\in ind^{S}(B^i)$, $(\bigcap_i B^i,x)\in ind^{I}([x]\pi_I{}^i)$, $(\bigcap_i B^i,x)\in[(s,x)]\pi'_{S\times I}{}^i$

$$= \bigcap_i [\delta(\bigcap_i B^i,x)]\pi_S{}^i = \quad (5)$$

$$= \delta(\bigcap_i B^i,x) = \quad (5)$$

$$= \delta(\Phi'(B^1,\ldots,B^n), x) \quad (14)$$

Thus, if condition 5 of Theorem 1 is also satisfied, then from the above calculations, from the definition of the state and output behaviour realization and the definitions 2 and 3, it follows that a general composition of n component machines M_i will realize the state behaviour of M, i.e. machine M has a general full-decomposition with state and output behaviour realization.

This ends the proof in one direction. This part of the proof shows how to construct the partial machines and their general compositions so that they form decompositional structures which realize behaviour of a specified sequential machine. In the second part of the proof, it will be shown that the proposed structures are the only possible decomposition structures which solve the general decomposition problem.

Let the output behaviour of M be realized by a general composition M_{GC} of n machines $M_i = (I_i^*, S_i, O_i, \delta^i, \lambda^i)$, where:

$I_i^* = I_i' \times I_i$,

Con_i: $\times O_j \rightarrow I_i'$, $1 \leq i,j \leq n$, is a surjective function.

Let:

Ψ: $I \rightarrow \times I_i$ be a function,

Φ: $S \rightarrow \times S_i$ be a function,

Θ: $\times O_i \rightarrow O$ be a surjective partial function.

Then, the mappings Ψ and Φ introduce the following partitions (equivalence relations) on I and S, respectively:

$$\pi_I{}^i\text{: } [x]\pi_I{}^i = [z]\pi_I{}^i \text{ if and only if } x_i = z_i$$

where $(x_1,\ldots,x_i,\ldots,x_n)=\Psi(x)$ and $(z_1,\ldots,z_i,\ldots,z_n)=\Psi(z)$

$$\pi_S{}^i\text{: } [s]\pi_S{}^i = [t]\pi_S{}^i \text{ if and only if } s_i = t_i$$

where $(s_1,\ldots,s_i,\ldots,s_n)=\Phi(s)$ and $(t_1,\ldots,t_i,\ldots,t_n)=\Phi(t)$

The elements from I_i and S_i give partial (abstracted) information about the elements of I and S, respectively, according to the above formulated equivalence relations. The elements of I_i and S_i can thus be considered as names of the equivalence classes (partition blocks) of the appropriate equivalence relations (partitions). Therefore, knowing the element from I_i or S_i is equivalent to knowing the block of an appropriate partition on I or S, and computing the element from I_i or S_i is equivalent to computing the block of an appropriate partition.

Each component machine M_i of a general decomposition unambiguously computes its output information from its present-state and input (primary and from the other machines), by computing values of its output function: λ_i: $S_i \times I^*{}_i \rightarrow O_i$, with Φ: $S \rightarrow \times S_i$, $I^* = I_i \times I'{}_i$,
Ψ: $I \rightarrow \times I_i$, Con_i: $O_1,\ldots,O_n \rightarrow I'{}_i$.

If a general composition of M_i is legal, then the output function λ_i of each component machine computes its values from the original primary input and state information present in the composition machine (directly or indirectly). Therefore, λ_i can be considered as being a function λ_i: $S \times I \rightarrow O_i$. In this way, λ_i introduces the partition $\pi_{S\times I}{}^i$ on $S \times I$: $[(s,x)]\pi_{S\times I}{}^i = [(t,z)]\pi_{S\times I}{}^i$ if and only if $\lambda_i(s,x) = \lambda_i(t,z)$. The values of λ_i can be then considered as being the blocks of $\pi_{S\times I}{}^i$ or being the names of the blocks of $\pi_{S\times I}{}^i$. In a similar way the connection functions Con_i and $Con_{j,i}$ introduce the par-

titions $\pi'_{S\times I}{}^{i} = \prod_{i=1,\ldots,n} \pi_{S\times I}{}^{j,i}$ and $\pi_{S\times I}{}^{j,i}$, respectively, such that $\pi_{S\times I}{}^{j,i} \geq \pi_{S\times I}{}^{j}$.

Since λ_i: $S_i \times I_i \times I'_i \rightarrow O_i$ is a function, values of λ_i being the blocks of $\pi_{S\times I}{}^{i}$ must be unambiguously computed from $\pi_S{}^{i}$ ($\pi^s{}_{S\times I}{}^{i}$), $\pi_i{}^{i}$ ($\pi^I{}_{S\times I}{}^{i}$) and $\pi'_{S\times I}{}^{i}$. Therefore, condition (2) must be satisfied. Similarly, since δ_i: $S_i \times I_i \times I'_i \rightarrow S_i$ is a function, the blocks of $\pi_S{}^{i}$ must be unambiguously computed from $\pi_I{}^{i}$ ($\pi^i{}_{S\times I}{}^{i}$), $\pi_S{}^{i}$ ($\pi^S{}_{S\times I}{}^{i}$) and $\pi'_{S\times I}{}^{i}$. Therefore, condition (1) has to be satisfied.

To guarantee the compositions legality values of each connection function, and so the values of $\pi'_{S\times I}$, must be computed (directly or indirectly) from the original primary input and state information present in the composition. Since the total primary input information in the composition is defined by $\pi^I{}_{S\times I}$ and the total state information by $\pi^S{}_{S\times I}$, condition (3) must be satisfied.

The output information produced together by all the component machines and described by $\pi_{S\times I}$ has to enable the unambiguous computation of the output information of the original machine M. Since Θ is a surjective partial function, its value is undefined or gives an element from O, so that each element from O is defined unambiguously by n-tuples of elements from $\times O_i$ ($\times\pi_{S\times I}{}^{i}$). So, $(\prod_i \pi_{S\times I}{}^{i}, \pi_O(0))$ must be an S × I-O partition pair, i.e. condition (4) must be satisfied. If additionally the state behaviour must be realized, then the state information of all the component machines must enable the unambiguous computation of the state information for the original machine, i.e. a surjective partial function Φ': $\times S_i \rightarrow S$ must exist. Such a function introduces the following partition on S:

$$\pi_S{}^{i}\text{: } [s]\pi_S{}^{i} = [t]\pi_S{}^{i} \text{ if and only if } s_i = t_i$$
$$\text{where } s = \Phi'(s_1,\ldots,s_i,\ldots,s_n),\ t = \Phi'(t_1,\ldots,t_i,\ldots t_n),$$

Since Φ' is a surjective partial function, each element from S must be unambiguously defined by n-tuples of elements from $\times \pi_S{}^{i}$, i.e. $\prod_i \pi_S{}^{i} = \pi_S(0)$. So, condition (5) must be satisfied.

Summarizing, if a sequential machine M has a general full-decomposition then n trinities of partitions $(\pi_I{}^{i}, \pi_S{}^{i}, \pi_{S\times I}{}^{i})$ exist, and they satisfy conditions (1)–(5) of Theorem 1. This ends the proof.

Acknowledgements

The author is indebted to Paweł Konieczny and Ad Chamboné for help in preparing the text and figures.

References

[1] P. Ashar, S. Devadas, A.R. Newton: Optimum and Heuristic Algorithms for Finite State Machine Decomposition and Partitioning, International Conference on Computer-Aided Design, p. 216–219, Nov. 1989.

[2] P. Ashar, S. Devadas, A.R. Newton: A Unified Approach to the Decomposition and Re-decomposition of Sequential Machines, 27th ACM/IEEE Design Automation Conference, pp. 601–605, 1990.

[3] P.Ashar, S. Devadas, A.R. Newton: Sequential Logic Synthesis, Kluwer Academic Publishers, Boston/Dordrecht/London, 1992.

[4] R.L. Ashenhurst: The Decomposition of Switching Functions, International Symposium on Theory of Switching Functions, pp. 74–116, April 1955.

[5] R.K. Brayton, C.T. McMullen: The Decomposition and Factorisation of Boolean Expressions, International Symposium on Circuits and Systems, Rome, May 1982, pp. 49–54.

[6] R.K. Brayton, G.D. Hachel, C.T. McMullen, A.L. Sangiovanni-Vincentelli: Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, Boston/Dordrecht/- London, 1984.

[7] R.K. Brayton, R. Rudell, A. Sangiovenni-Vincentelli, A.R. Wang: MIS : Multiple-Level Logic Optimization System, IEEE Trans. on Computer-Aided Design, p. 1062–1081, Nov. 1987.

[8] K. Chen, J. Cong, Y. Ding, A.B. Kahng, P. Trajmar: DAG-Map: Graph-Based FPGA Technology Mapping for Delay Optimization, IEEE Design and Test of Computers, September 1992, pp. 7–20.

[9] M.J. Ciesielski, S. Yang: PLADE: A Two Stage PLA Decomposition, IEEE Transactions on CAD, vol. CAD -11, No 8, August 1992, pp. 943–954.

[10] H.A. Curtis: A New Approach to the Design of Switching Circuits, van Nostrand Co., Princeton 1962.

[11] M.R. Dagenais, V.K. Agarwal, N.C. Rumin: MCBOOL: A New Procedure for Exact Logic Minimization, IEEE Trans. on CAD, vol. CAD-5, No 1, Jan. 1986, pp. 229–238.

[12] S. Devadas et al: Decomposition and Factorization of Sequential Finite State Machines, International Conference on Computer-Aided Design, November 1988.

[13] S. Devadas: General Decomposition of Sequential Machines: Relationship to State Assignment, 26th Design Automation Conference, pp. 314–320, 1989.

[14] S. Devadas, A.R. Wang, A.R. Newton: Boolean Decomposition in Multi-Level Logic Optimization, Journal of Solid State Circuits, vol. 24, April 1989, pp. 399–408.

[15] W. Grass, M. Mutz: Modular Implementation of Finite State Machines Observing Topological Constraints, IFIP Conference on Computer Hardware Description Languages and Their Applications, Elsevier Science Publishers B.V. (North-Holland), pp. 183–196, 1990.

[16] J. Hartmanis: Symbolic Analysis of a Decomposition of Information Processing, Information and Control, vol. 3, pp. 154–178, June 1960.

[17] J. Hartmanis: Loop-free Structure of Sequential Machines, Information and Control, vol. 5, pp. 25–43, 1962.

[18] J. Hartmanis, R.E. Stearns: Algebraic Structure Theory of Sequential Machines, Englewood Cliffs, N.J.: Prentice-Hall, 1966.

[19] Y. Hou: Trinity Algebra and Full-Decompositions of Sequential Machines, Ph.D. thesis, Eindhoven University of Technology, the Netherlands, 1986.

[20] Y. Hou: Trinity Algebra and its Application to Machine Decompositions, Information Processing Letters, vol. 26, pp. 127–134, 1987.

[21] K. Jasiński, T. Łuba, J. Kalinowski: Parallel Decomposition in Logic Synthesis, 15th European Solid-State Circuits Conf., pp. 113–116, Sept. 1989.

[22] L. Jóźwiak: The Full Decomposition of Sequential Machines with the State and Output Behaviour Realization, EUT Report

88-E-188, Eindhoven University of Technology, the Netherlands, 1988.

[23] L. Jóźwiak: The Full Decomposition of Sequential Machines with the Output Behavior Realization, EUT Report 88-E-199, Eindhoven University of Technology, the Netherlands, 1988.

[24] L. Jóźwiak: The Full Decomposition of Sequential Machines with the Separate Realization of the Next-State and Output Functions, EUT Report 89-E-222, Eindhoven University of Technology, the Netherlands, 1989.

[25] L. Jóźwiak: The Bit Full-Decomposition of Sequential Machines, EUT Report 89-E-223, Eindhoven University of Technology, the Netherlands, 1989.

[26] L. Jóźwiak, F. Vankan: Bit Full-Decompositions of Sequential Machines—Algorithms and Results, Canadian Conference on Electrical and Computer Engineering, Montreal, Canada, 17–20 Sept. 1989.

[27] L. Jóźwiak: Simultaneous Decompositions of Sequential Machines, Microprocessing and Microprogramming 30 (1990), 305–312.

[28] L. Jóźwiak, T. Spassova-Kwaaitaal: Decompositional State Assignment with Reuse of Standard Designs, EUT Report 90-E-247, Eindhoven University of Technology, the Netherlands, 1990.

[29] L. Jóźwiak, J.C. Kolsteren: An Efficient Method for the Sequential General Decomposition of Sequential Machines, Microprocessing and Microprogramming, North-Holland, vol. 32, pp. 657–664, 1991.

[30] L. Jóźwiak, F. Volf: An Efficient Method for Decomposition of Multiple Output Boolean Functions and Assigned Sequential Machines, EDAC—The European Conference on Design Automation, Brussels, Belgium, March 16–19, 1992, pp. 114–122.

[31] L. Jóźwiak: An Efficient Heuristic Method for State Assignment of Large Sequential Machines: Journal of Circuits, Systems and Computers, vol. 2, No 1, 1992, pp 1–26.

[32] L. Jóźwiak, A.P.H. van Dijk: A Method for General Simultaneous Full-Decomposition of Sequential Machines: Algorithms and Implementation, EUT-Report 92-E-267, Eindhoven University of Technology, the Netherlands, 1992.

[33] B.G. Kim, D.L. Dietmeyer: Multilevel Logic Synthesis of Symmetric Switching Functions, IEEE Trans. on CAD, vol. CAD-10, No. 4, April 1991.

[34] K. Lam, S. Devadas: Performance-oriented Decomposition of Sequential Circuits, 1990 IEEE International Symposium on Circuits and Systems, pp. 2642–2645.

[35] T. Łuba, J. Kalinowski, K. Jasiński, H. Krasniewski: Combining Serial Decomposition with Topological Partitioning for Effective Multi-level PLA Implementations, IFIP Working conference on Logic and Architečture Synth., pp. 77–86, Paris, May 30–June 1, 1990.

[36] T. Łuba, K. Górski, L. Wroński: ROM-based Finite State Machines with PLA Address Modifiers, European Design Automation Conference, Hamburg, Germany, September 7–10, 1992, pp. 272–277.

[37] A.A. Malik, D. Harrison, R.K. Brayton: Three-Level Decomposition with Application to PLDs, IEEE International Conference on Computer Design: VLSI in Computers and Processors, Cambridge, Massachusetts, October 14–16, 1991, pp. 628–633.

[38] R.R. Munoz, C.E. Stroud: Automatic Partitioning of Programmable Logic Devices, VLSI System Design, pp. 74–79, October 1987.

[39] T. Muller—Wipperfurth, M. Geiger: Algebraic Decomposition of MCNC Benchmark FSMs for Logic Synthesis, EuroASIC Conference, 1991, pp. 146–151.

[40] M. Geiger, T. Muller-Wipperfurth: FSM Decomposition Revisited: Algebraic Structure Theory Applied to MCNC Benchmark FSM, 28 ACM/IEEE DAC, pp. 182–185, 1991.

[41] E.B. Pitty: A Critique of the GATEMAP Logic Synthesis System, Int. Workshop on Logic and Architecture Synth. for Silicon Compilers, pp. 65–84, May 1988.

[42] G. Saucier, P. Sicard, L. Bouchet: Multi-Level Synthesis on PALs, EDAC—European Design Automation Conference, pp. 542–546, Glasgow, March 1990.

[43] P. Vincke, M. Gassner, B. Roy: Multicriteria Decision Aid, John Wiley and Sons, Chichester, 1992).

[44] W. Wan, M.A. Perkowski: A New Approach to the Decomposition of Incompletely specified Multi-Output Functions Based on Graph Coloring and Local Transformation and Its Application to FPGA Mapping, European Design Automation Conference, Hamburg, Germany, September 7–10, 1992, pp. 230–237.

[45] S. Yang, M.J. Ciesielski: PLA Decomposition with Generalized Decoders, International Conference on Computer Aided Design, Santa Clara, California, November 1989, pp. 312–315.

[46] K. Yau-Hwang, W. Ruey-Rong, K. Ling-Yeung: Logic Design Using the PLAs with Limited I/O Pins and Product Terms, Microprocessing and Microprogramming, vol. 23, No 1, 1988, pp. 27–32.

Biographies

LECH JÓŹWIAK received his M. Sc. and Ph. D. degrees in Electronics from the Warsaw University of Technology, Poland, in 1976 and 1982, respectively. From 1976 to 1979, he worked at the Faculty of Electronics of this University. From 1979 to 1986, he was a chief of the project team in the Research Institute of Computers in Warsaw. From 1986 till now, he is an associate professor in the Department of Digital Information Systems, Eindhoven University of Technology, the Netherlands. His research interests include design theory, circuit and system theory, design theory and artificial intelligence in design and correctness verification. He is an author of numerous research reports and papers.