The bit full-decomposition of sequential machines

tu/e
Research Report
ISSN 0167-9708
Coden: TEUEDE

Eindhoven
University of Technology
Netherlands

Faculty of Electrical Engineering

The Bit Full-Decomposition of Sequential Machines

by
L. Jóźwiak

EUT Report 89-E-223
ISBN 90-6144-223-0
May 1989

Eindhoven University of Technology Research Reports

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering

Eindhoven The Netherlands

ISSN 0167-9708 Coden: TEUEDE

THE BIT FULL-DECOMPOSITION OF SEQUENTIAL MACHINES

by

L. Jóźwiak

EUT Report 89-E-223

ISBN 90-6144-223-0

Eindhoven

May 1989

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Jóźwiak, L.

The bit full-decomposition of sequential machines /
by L. Jóźwiak. - Eindhoven: Eindhoven University of
Technology, Faculty of Electrical Engineering. - Fig. -
(EUT report, ISSN 0167-9708; 89-E-223)
Met lit. opg., reg.
ISBN 90-6144-223-0
SISO 664 UDC 681.325.65:519.6 NUGI 832
Trefw.: automatentheorie

The bit full-decomposition of sequential machines.

L. Jóźwiak

Digital Systems Group, Faculty Electrical Engineering,
Eindhoven University of Technology, The Netherlands

Abstract - Control units and serial processing units of today's information processing systems must realize complex processes, which are usually described in the form of a sequential machine or a number of cooperating sequential machines. Large machines are difficult to: design, optimize, implement and verify. Therefore, there is a real need for CAD tools, which could decompose a complex sequential machine into a number of smaller and less complicated partial machines.

For many years, the decomposition of only the internal states of sequential machines has been studied. However, this sort of decomposition is not a sufficient solution. The complexity of a circuit implementing a sequential machine is a function not only of machine's internal states but as well of inputs and outputs. Furthermore, the possibility to implement a machine with today's array logic building blocks depends not only on the number of internal states but as well on inputs and outputs. So, there is a real need for decompositions upon the states, inputs and outputs of a sequential machine, i.e. for full-decompositions.

During the full-decomposition process, the input and/or state and/or output symbols (values) can be decomposed or the input and/or state and/or output bits. So, it is possible to perform the symbol full-decomposition or the bit full-decomposition.

This report provides the classification of full-decompositions and describes briefly the theoretical foundations of bit full-decomposition.

Comparing to the symbol full-decomposition,the bit full-decomposition has the following advantage: input and output decoders are reduced to an appropriate distribution of the primary input and output bits between the partial machines.

In the report, definitions of a bit partition and bit partition pairs are introduced and their usefulness to bit full-decompositions is shown. It is proved, that the bit full-decomposition can be treated as a special case of the symbol full-decomposition; therefore, no new decomposition theory is needed for this case, but the symbol full-decomposition theory together with the theorems introduced here constitute the theory of bit full-decomposition.

Finally, a comparison is made between the symbol and the bit full-decompositions and some practical conclusions and remarks are presented.

In the appendix, an example is provided that illustrates the possibility and the practical usefulness of bit full-decomposition.

Based on the developed theory, the CAD algorithms calculating different bit full-decompositions have been developed and implemented. Those algorithms and the practical results are presented and estimated in the separate paper [5].

Index Terms - Automata theory, decomposition, logic design, sequential machines.

Acknowledgements - The author is indebted to Prof.ir. A. Heetman and Prof.ir. M.P.J. Stevens for making it possible to perform this work, to Dr. P.R. Attwood for making corrections to the English text and to mr. C. van de Watering for typing the text.

CONTENTS

	page
1. Introduction	1
2. Types of full-decomposition	2
3. Partition pairs and bit full-decompositions	5
4. Comparison of different sorts of full-decomposition	12
5. CAD algorithms and practical results	13
References	15
Appendix (Example)	16

1. Introduction.

Control units and serial processing units of today's information processing systems must realize complex processes, which are usually described in the form of a sequential machine or a number of cooperating sequential machines. Large and complicated sequential machines are difficult to: design, optimize, implement and verify. Therefore, there is a real need for CAD tools, which could decompose a complex sequential machine into a number of smaller and less complicated partial machines. Array logic implementation techniques dictate also the requirements for decomposition. One of possible approaches to the decomposition of sequential machines is the algebraic approach.

For many years, the algebraic decomposition of only the internal states of sequential machines has been studied [6]+[17]. However, this sort of decomposition is not a sufficient solution. The most important parameters such as the complexity, speed, testability, power consumption etc., of a circuit implementing a sequential machine are functions not only on machine's internal states but as well on inputs and outputs. Furthermore, the possibility to implement a machine with today's array logic building blocks depends not only on the number of internal states but as well on inputs and outputs. So, there is a real need for decompositions upon the states, inputs and outputs of a sequential machine, i.e. for full-decompositions [1]+[4]. Algebraic full-decompositions can be used in order: to make it possible to implement a given sequential machine with existing building blocks or inside a limited silicon area; to improve some design parameters (speed, testability, ...); to minimize partially the resultant circuit and to make it possible to optimize the separate partial machines, although it may be impossible to optimize the whole machine.

In this report, the classification of full-decompositions is provided, the theoretical foundations of bit full-decompositions are briefly described and a comparison of different sorts of full-decompositions is made.

In the appendix, an example is provided that illustrates the possibility and the practical usefulness of bit full-decomposition.

Based on the developed theory, the CAD algorithms that

calculate different bit full-decompositions have been developed and implemented. Those algorithms and the practical results are presented and estimated in the separate paper [5].

We close our presentations with conclusions about the practical usefulness of full-decomposition and the CAD algorithms developed by us.

2. Types of full-decomposition.

DEFINITION 1 A ***sequential machine*** M is an algebraic system defined as follows:

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

where:

- I - a finite non-empty set of inputs,
- S - a finite non-empty set of internal states,
- O - a finite set of outputs,
- δ - the next-state function: δ: SxI $\longrightarrow$ S,
- λ - the output function, λ: SxI $\longrightarrow$ O (a *Mealy* machine), or λ: S $\longrightarrow$ O (a *Moore* machine).

When the output set, O, and the output function, λ, are not defined, the sequential machine M = (I, S, δ) is called a ***state machine***.

Let M = (I, S, O, δ, λ) be the sequential machine to be decomposed. In [3][4], such a full-decomposition is presented, where it is necessary to find two partial sequential machines, M_1= $(I_1,S_1,O_1,\delta^1,\lambda^1)$ and M_2= $(I_2,S_2,O_2,\delta^2,\lambda^2)$, each having fewer states and/or inputs and/or outputs than M. Each of them can calculate its next-states and outputs using only the information about its own input and its own state and, in combination, they form a sequential machine M' that has the same input-output behaviour or input-state and input-output behaviour as M (common realization of the next-state and output functions - Fig. 1).

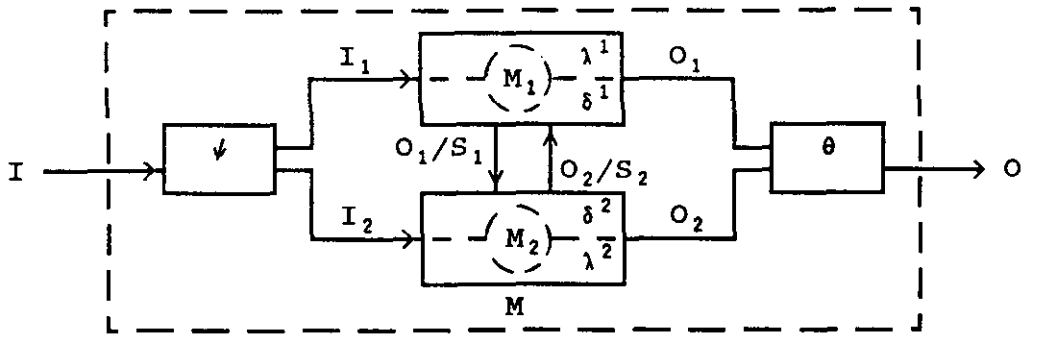


Fig. 1 The full-decomposition of a sequential machine M with two partial sequential machines M_1 and M_2. (common realization of the next-state and output functions).

Instead of considering the realization of a machine M as a whole, the realization of the next-state function, δ, can be considered separately from the output function, λ.

It is possible to abstract from the output function λ and to decompose the state machine which is defined by I, S and the next-state function δ. Then, it is possible to realize the output function λ, where λ is treated as a function of the primary inputs to a sequential machine M (in the Mealy case), and the states of partial state machines M_1 and M_2 that are obtained from a full-decomposition of the state machine defined by I, S and δ (separate realization of the next-state and output functions - Fig. 2).

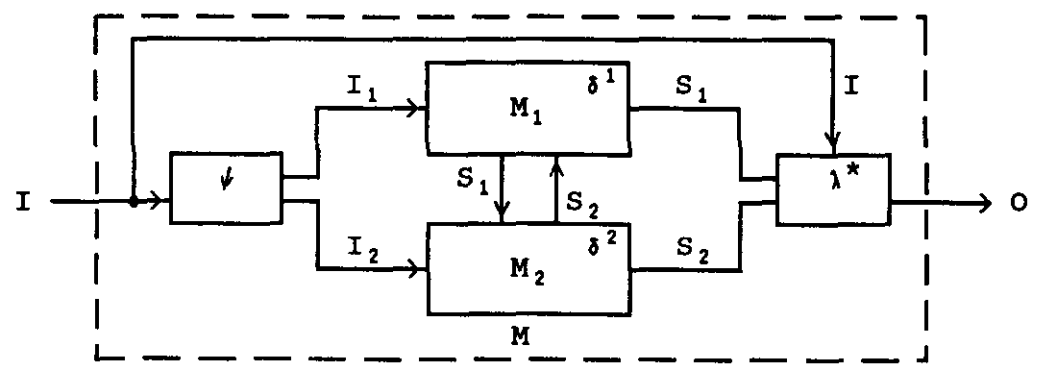


Fig. 2 The full-decomposition of a sequential machine M with the separate realization of the next-state and output functions.

Both types of the full-decomposition above can be considered as decompositions realizing the state and output behaviour of a machine M, but the first type may be considered also as a decomposition realizing only the output behaviour of M [3][4].

From the viewpoint of connections between the component machines, it is possible to distinguish the following types of full-decompositions:

- a parallel full-decomposition - each of the component machines can calculate its own next-states and outputs independently of the other component machines and only from the information about its own internal state and the partial information about the inputs;
- a serial full-decomposition - one of the component machines, which is called the tail or dependent machine (M_2), uses information about the states or outputs of the second machine, which is called the head or independent machine (M_1), plus the information about its own state and the partial information about the inputs in order to calculate its own next-states and outputs;
- a general full-decomposition - each of the component machines uses information about the states or outputs of the other machine, plus the information about its own state and the partial information about the inputs in order to calculate its own next states and outputs.

From the viewpoint of the kind of information available about a given submachine and used by another submachine in order to calculate its next-states and outputs, the following two types of a full-decomposition can be distinguished:

- a decomposition with information about the states (type S);
- a decomposition with information about the outputs (type O).

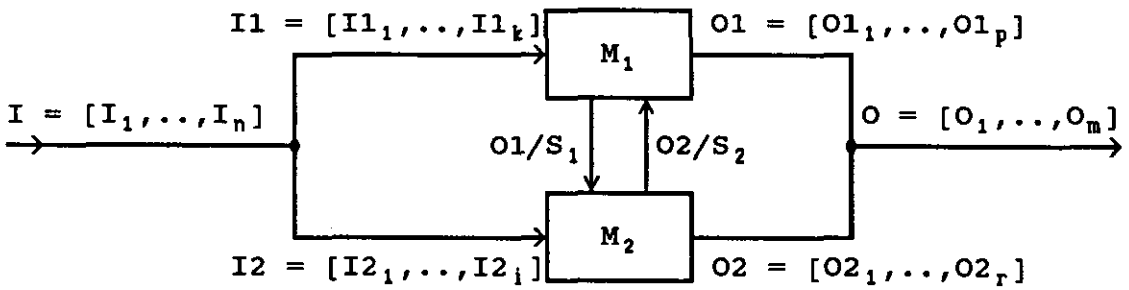
A given submachine can use the information about the "present" or the "next" state or output of the other submachine; consequently, the class P (present) and the class N (next) of decompositions can be distinguished.

The sets I, S and O of inputs, states and outputs can be treated as sets of symbols, but for the sets I and O, there is another treatment too.

Contrary to the states, which are given in the form of symbols,

in most cases, and for which codes have to be chosen, the inputs and outputs of a sequential machine are usually pre-assigned. In most cases, the inputs and outputs are given in the form of vectors of the input/output bit values, because inputs comprise direct signals from the surroundings of the machine, while outputs are the direct control signals sent by the machine to the surroundings. Of course, input and output vectors can also be treated as symbols, but the vector view of them is often useful in relation to the full-decomposition, because it allows the input and output bits to be decomposed between the partial machines instead of the input and output symbols.

In this case, the input and output decoders, ψ and θ are reduced to the appropriate distribution of the input and output bit lines. So, each of the types of full-decomposition considered previously can be considered as either a symbol full-decomposition or a bit full-decomposition.



$\{I1_1,..,I1_k\} \subseteq \{I_1,..,I_n\}$, $\{I2_1,..,I2_j\} \subseteq \{I_1,..,I_n\}$,
$\{O1_1,..,O1_p\} \subseteq \{O_1,..,O_m\}$, $\{O2_1,..,O2_r\} \subseteq \{O_1,..,O_m\}$.
$O1 \cup O2 = O$.

Fig. 2 The bit full-decomposition of a sequential machine M.

3. Partition pairs and bit full-decomposition.

The concepts of partitions and partition pairs introduced by Hartmanis [9][10][11][12] are useful tools for analyzing the information flow in and between machines; therefore, they were used in this work.

Let S be any set of elements.

DEFINITION 3.1 Partition π on S is defined as follows:

$$\pi = \{B_i \mid B_i \subseteq S \text{ and } B_i \cap B_j = 0 \text{ for } i \neq j \text{ and } \bigcup_i B_i = S\},$$

i.e. a partition π on S is a set of disjoint subsets of S whose set union is S.

For a given $s \in S$, the block of a partition π containing s is denoted as: $[s]\pi$ and $[s]\pi = [t]\pi$ is written to denote that s and t are in the same block of π. Similarly, the block of a 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 denoted by $\pi_S(0)$. The partition containing all the elements of S in one block is called an ***identity*** or ***one partition*** and is denoted by $\pi_S(I)$.

Let π_1 and π_2 be two partitions on S.

DEFINITION 3.2 ***Partition product*** $\pi_1 \cdot \pi_2$ is the 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 $[s]\pi_2 = [t]\pi_2$.

DEFINITION 3.3 ***Partition sum*** $\pi_1 + \pi_2$ is the 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 $[s_i]\pi_1 = [s_{i+1}]\pi_1$ either $[s_i]\pi_2 = [s_{i+1}]\pi_2$, $0 \leq i \leq n-1$.

DEFINITION 3.4 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* $\pi_1 \cdot \pi_2 = \pi_1$ *if and only if* $\pi_1 + \pi_2 = \pi_2$.

Let π_S, τ_S, π_I, π_0 be the partitions on $M = (I, S, O, \delta, \lambda)$, in particular: π_S, τ_S on S, π_I on I, π_0 on O.

DEFINITION 3.7

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

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

(iii) (π_S, π_0) is an ***S-O partition pair*** *if and only if*
$\forall B \in \pi_S \ \forall x \in I : B\bar{\lambda}_x \subseteq C , C \in \pi_0$ (Mealy case)
or
$\forall B \in \pi_S : B\bar{\lambda} \subseteq C , C \in \pi_0$ (Moore case).

(iv) (π_I, π_O) is an ***I-O partition pair*** *if and only if*

$\forall A \epsilon \pi_I \ \forall s \epsilon S : s\bar{\lambda}_A \subseteq C , C \epsilon \pi_O$ (Mealy case)

or

$\forall A \epsilon \pi_I \ \forall s \epsilon S : s\lambda \subseteq C , C \epsilon \pi_O$ (Moore case).

The practical 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. Thus, if the block of π_S which contains the present state of the machine M is known as well as the present input of M, it is possible to compute unambiguously the block of τ_S which contains the next state of M for the states from a given block of π_S and a given input, i.e. the input and the block of π_S determine unambiguously the block of τ_S. Interpreting the notions of I-S, S-O and I-O partition pairs is similar.

In the case of a Moore machine, the definition of an I-O pair is trivial, because each (π_I, π_S) will satisfy it (the output of M is defined by the state of M unambiguously).

DEFINITION 3.8 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.

For the purpose of bit full-decomposition, the concepts of bit partitions (as a special case of partitions) and bit partition pairs has been introduced by us.

Let B be a set of input or output bits: $B = \{b_1, b_2, \ldots, b_{|B|}\}$. Let $T = \{t_1, t_2, \ldots, t_{|T|}\}$ be a set of input/output symbols.

Each input/output bit b_k: $b_k \epsilon B$, introduces a two block partition $\pi_T(b_k)$ on the set of input/output symbols T. In one block of $\pi_T(b_k)$, these symbols are contained for which bit b_k has the value $\emptyset$; in the second block of $\pi_T(b_k)$ are the symbols for which b_k has the value 1. The product of partitions $\pi_T(b_k)$ for all the bits b_k: $b_k \epsilon B$ defines unambiguously the set of all input/output symbols, i.e.

$$\prod_{b_k \epsilon B} \pi_T(b_k) = \pi_T(\emptyset).$$

DEFINITION 3.9 A partition

$\pi_B = \{b_1, b_2, \ldots, b_k, (b_{k+1}, \ldots, b_{|B|})\}$ on the set of bits B, where:

- important bits : $b_1, b_2, \ldots, b_k$ are kept in separate blocks,
- don't care bits: $b_{k+1}, \ldots b_n$ are kept in a single block called a don't care block and denoted by $dcb(\pi)$,

is called a bit partition on B.

The product ($\cdot$) and sum (+) operation and the ordering relation ($\leq$) for bit partions are normal partition operations and ordering relations, 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 partition $\pi_B(0)$ is defined as a bit partition with an empty don't care block, i.e. $\pi_B = \pi_B(0)$ *if and only if* $dcb(\pi_B) = \emptyset$.

Let π_{IB} be a bit partition on the set of input bits $IB = \{ib_1, \ldots, ib_{|IB|}\}$. Let π_{OB} be a bit partition on the set of output bits $OB = \{ob_1, \ldots, ob_{|OB|}\}$ and let τ_S be a (symbol) partition on the set of states S.

DEFINITION 3.10 (π_{IB}, τ_S) is an IB-S partition pair *if and only if* $\forall s \in S \quad \forall ib_k \in dcb(\pi_{IB})$:

$$[s\delta_{(ib_1, \ldots, ib_{(k-1)}, 0, ib_{(k+1)}, \ldots, ib_{|IB|})}]\tau_S =$$

$$= [s\delta_{(ib_1, \ldots, ib_{(k-1)}, 1, ib_{(k+1)}, \ldots, ib_{|IB|})}]\tau_S,$$

i.e. for each state $s \in S$, the next sates are included in the same block of τ_S independently of the values of all the bits ib_k: $ib_k \in dcb(IB)$.

Let $\pi_0(ob_k)$ be the two block partition that is introduced by the output bit ob_k: $ob_k \in OB$ on the set of output symbols O.

DEFINITION 3.11 (τ_S, π_{OB}) is an S-OB partition pair *if and only if* $\forall x \epsilon I \quad \forall s,t \epsilon S \wedge [s]\tau_S = [t]\tau_S \quad \forall ob_k \notin dcb(\pi_{OB})$:

$$[s\lambda_k]\pi_0(ob_k) = [t\lambda_x]\pi_0(ob_k) \ ,$$

i.e. the input value $x \epsilon I$ and the block $B\tau_S \epsilon B$ define unambiguously the value of each output bit ob_k: $ob_k \notin dcb(\pi_{OB})$.

DEFINITION 3.12 (π_{IB}, π_{OB}) is an IB-OB partition pair *if and only if* $\forall s \epsilon S \quad \forall ib_k \epsilon dcb(\pi_{IB}) \quad \forall ob_k \notin dcb(\pi_{OB})$:

$$[s\lambda_{[ib_1,\ldots,ib(k-1),0,ib(k+1),\ldots,ib_{|IB|}]}]\pi_0(ob_k) =$$

$$= [s\lambda_{[ib_1,\ldots,ib(k-1),1,ib(k+1),\ldots,ib_{|IB|}]}]\pi_0(ob_k) \ ,$$

i.e. for each state s, the values of all the output bits $ob_k \notin dcb(\pi_{OB})$ are independent of the values of all the input bits $ib_k \epsilon dcb(\pi_{IB})$.

Let π_I be a partition that is introduced on the set of input symbols I by a set of input bits IB-dcb(π_{IB}), i.e.:

$$\pi_I = \prod_{ib_k \epsilon IB\text{-}dcb(\pi_{IB})} \pi_I(ib_k) \ .$$

Let π_I' be a partition introduced on I by the set of "don't care" input bits $dcb(\pi_{IB})$,

i.e. $$\pi_I' = \prod_{ib_k \epsilon dcb(\pi_{IB})} \pi_I(ib_k) \ .$$

It is obvious that $\pi_I \cdot \pi_I' = \pi_I(\emptyset)$

THEOREM 3.1

If (π_{IB}, τ_S) is an IB-S partition pair and π_I is the partition on I that is introduced by the set of input bits IB-dcb(π_{IB}), then:
(π_I, π_S) is an I-S partition pair.

Proof.

From the definition of an IB-S partition pair, it follows immediately that the block of a partition τ_S that contains the next-state $s\delta_x$ for a given state $s \epsilon S$ and a given input $x \epsilon I$, is independent of the block of a partition $\pi_I(ib_k)$ containing the current input x, for all $ib_k \epsilon dcb(\pi_{IB})$. Therefore, the block of

τ_s, containing the next-state $s\delta_x$ depends only on s and the blocks of partitions $\pi_I(ib_k)$ for ib_k: $ib_k \not\in dcb(\pi_{IB})$, i.e. the block of τ_s containing the next-state $s\delta_x$ is determined unambiguously by the present state S and the block of a partition π_I which represents the product of partitions $\pi_I(ib_k)$ for all ib_k: $ib_k \in$ IB-dcb(π_{IB}). So, the partitions, π_I and τ_s, constitute an I-S partition pair.
The following two theorems can be proved in a similar way.

THEOREM 3.2

If (τ_s, π_{OB}) is an S-OB partition pair and π_O is an output partition on O that is introduced by the set of output bits OB-dcb(π_{OB}),

i.e. $$\pi_O = \prod_{ob_k \in OB\text{-}dcb(\pi_{OB})} \pi_O(ob_k)$$

then, (τ_s, π_O) is a S-O partition pair.

THEOREM 3.3

If (π_{IB}, π_{OB}) is an IB-OB partition pair, πI represents a partition on I that is introduced by the set of input bits IB-dcb(π_{IB})
and
π_O represents a partition on O that is introduced by the set of output bits OB-dcb(π_{OB}),
then:
(π_I, π_O) is an I-O partition pair.

Let π'_B and π''_B be two partitions on the set of input/output bits B and let π'_T and π''_T be two partitions on the set of input/output symbols T such that :-

$$\pi'_T = \prod_{b_k \in B\text{-}dcb(\pi'_B)} \pi_T(b_k) \quad \text{and} \quad \pi''_T = \prod_{b_k \in B\text{-}dcb(\pi''_B)} \pi_T(b_k) \ .$$

THEOREM 3.4

If two bit partitions π'_B and π''_B are orthogonal, then, the symbol partitions, π'_T and π''_T, introduced by them, are orthogonal too:
i.e. if $\pi'_B \cdot \pi''_B = \pi_B(0)$ then: $\pi'_T \cdot \pi''_T = \pi_T(0)$.

Proof.

If $\pi'_B \cdot \pi''_B = \pi_B(0)$, then: $dcb(\pi'_B \cdot \pi''_B) = dcb(\pi'_B) \cdot dcb(\pi''_B) = \emptyset$

(from the definition of a zero bit partition).

$$\pi'_T \cdot \pi''_T = \prod_{b_k \in B - dcb(\pi'_B)} \pi_T(b_k) \cdot \prod_{b_k \in dcb(\pi''_B)} \pi_T(b_k) =$$

$$= \prod_{b_k \in B - (dcb(\pi'_B) \cap dcb(\pi''_B))} \pi_T(b_T) = \prod_{b_k \in B} \pi_T(b_k) = \pi_T(0).$$

Similar definitions and similar theorems can be introduced and proved for weak partition pairs.

In [3] and [4], a set of constructive theorems, concerning the existence of different kinds of symbol full-decompositions has been proved. Each of these theorems stated: if, for a machine M, a given system of I-S, S-S, S-O and I-O partition pairs exists and some partitions from these pairs satisfy the appropriate orthogonality conditions, then, a given type of a symbol full-decomposition of M will result.

For instance, if for a machine M, two trinities of partitions: (π_I, π_S, π_O) and (τ_I, τ_S, τ_O) exist, that:
- π_S and τ_S are SP-partitions,
- (π_I, π_S) and (τ_I, τ_S) are I-S partition pairs,
- (π_I, π_O) and (τ_I, τ_O) are I-O partition pairs,
- (π_S, π_O) and (τ_S, τ_O) are S-O partition pairs,

and
- $\pi_O \cdot \tau_O = \pi_O(\emptyset)$,

then:
a parallel symbol full-decomposition of M with the realization of the output behaviour will result. If additionally $\pi_S \cdot \tau_S = \pi_S(\emptyset)$ then the state behaviour of M will also be realized.

Those facts have the following interpretation:
Let the partial machine M_1 in the parallel symbol full-decomposition be constructed according to the trinity (π_I, π_S, π_O) and the partial machine M_2 acoording to the trinity (τ_I, τ_S, τ_O). Let blocks of π_I, π_S and π_O be adequately the inputs, states and outputs of M_1 and the blocks of τ_I, τ_S and τ_O be adequately the inputs, states and outputs of M_2.

Since π_I, π_s, π_0 and τ_I, τ_s, τ_0 form the listed above partition pairs, based only on the information about the block of π_I containing the input of M and the block of π_s containing the present-state of M (i.e. information about the input and present-state of M_1), machine M_1 can calculate unambiguously the block of π_s in which the next-state of M is contained, as well as, the block of π_0 that contains the output of M for the input from a given block of π_I and for the present-state from a given block of π_s (i.e. M_1 can calculate its next-state and output). Similarly, machine M_2 based only on the information about its input and present-state can calculate its own next-state and output. Since $\pi_0 \cdot \tau_0 = \pi_0(\emptyset)$, the knowledge of the block of π_0 and the block of τ_0 in which the output of M is contained, makes it possible to calculate this output. So, if $\pi_0 \cdot \tau_0 = \pi_0(\emptyset)$, the machines M_1 and M_2 together can calculate the state of M unambiguously. That means, that the machines M_1 and M_2 operate independently of each other and they realize together the output or the state and output behaviour of M, i.e. M has a parallel symbol full-decomposition.

From theorems 3.1 - 3.3, it follows that: if certain bit partition pairs exist, then, the appropriate symbol partition pairs will exist and, from theorem 3.4, it follows that: if two bit partitions are orthogonal, then, the appropriate symbol partitions are orthogonal too.

So, the bit full-decomposition can be considered as a special case of symbol full-decomposition. No new theory for the bit full-decomposition needs to be developed; since, the theory for the symbol full-decomposition described in [3][4] and supplemented with the theorems provided in this report, can be utilized directly for bit full-decomposition.

4. Comparison of different sorts of full-decomposition.

Symbol-full-decomposition is general while bit-full-decomposition is a special case, i.e. a given type of bit-full-decomposition cannot exist, whereas, that of symbol-full-decomposition can. However, for symbol-full-decomposition input and output decoders must be realized in the form of combinational circuits whereas for bit-ful-decomposition they are reduced to the appropriate distribution of input and output bits between the

partial machines.

From the practical point of view, full-decompositions of type N are not so attractive as decompositions of type P, because in decompositions of type N, one of the component machines has to be able to compute its next-state or output, before the second component machine, using the information about the computed next-state or output of the first machine, can compute its own next-state or output. In this situation, the frequency of input signals needs to be limited and a two-phase clock is required.

The decompositions with the separate realization of the next-state and output functions are easier to find than the decompositions with the common realization, but, using them the suboptimal solutions can be found only, because the common parts of the next-state and output logic cannot be shared.

In the case of serial and general decompositions, connections between partial machines have to be implemented whereas for parallel decompositions no connections are needed. The complexity of combinational logic of the component machines is also usually low for parallel decompositions (reduced dependencies). Therefore, solving the practical cases starts with trying to find an appropriate parallel full-decomposition which satisfies some requirements.

5. CAD algorithms and practical results.

Based on the theory of full-decomposition provided in [1][2][3][4] and in this report, the CAD algorithms, that calculate different parallel and serial full-decompositions, have been developed and implemented.

The practical aspects of full-decompositions are described more precisely in a separate paper [5].

We close our presentation with some conclusions about the practical usefulness of full-decompositions and the CAD-algorithms and programs developed by us.

For a benchmark of 43 medium and large (number of input bits ≥ 10, number of output bits ≥ 10, number of states ≥ 20) practical sequential machines we got from out colleagues, we run programs for bit full-decompositions implemented following the concept of

weak partition pairs.

We found good parallel bit full-decompositions for 30% of the examples and we found good serial bit full-decompositions for 50% of the machines. A good decomposition means: reduction of the silicon area used for implementing a sequential machine to be decomposed or a small increase of the silicon area, but each of the partial machines is substantially smaller than the original machine (improvement of the other design parameters).

Since some machines do not possess any parallel and/or serial full-decompositions, many machines do not possess good parallel and/or serial full-decompositions and every machine possesses general decompositions, we are now busy developing CAD tools for general full-decompositions.

For some large sequential machines with special internal features (e.g. a lot of "don't cares"), the number of SP-partitions and/or partition pairs which have to be generated and checked in order to find useful parallel or serial full-decompositions can be so high, that, with the use of our programs and computers, we are not able to calculate the decompositions in reasonable time (two cases from our benchmark); however, for many large machines we reached good results.

We are now busy developing faster full-decomposition tools according to the concept of labelled partition pairs.

REFERENCES

[1] Y. Hou : Trinity algebra and full-decompositions of sequential machines, Ph.D. thesis, Eindhoven University of Technology, The Netherlands, 1986.

[2] Y. Hou : Trinity algebra and its application to machine decompositions, Information Processing Letters, vol.26, p.127-134, 1987.

[3] 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.

[4] L. Jóźwiak : The full decomposition of sequential machines with the output behaviour realization, EUT Report 88-E-199, Eindhoven University of Technology, The Netherlands, 1988.

[5] L. Jóźwiak, F. Vankan: Bit full-decompositions of sequential machines - algorithms and results, to be published in the Proceedings of the Canadian Conference on Electrical and Computer Engineering, Montreal, September 1989.

[6] G. Cioffi, E. Constantini, S. de Julio : A new approach to the decomposition of sequential systems, Digital Processes, vol.3, p. 35-48, 1977.

[7] G. Cioffi, S. de Julio, M. Lucertini : Optimal decomposition of sequential machines via integer nonlinear programming: A computational algorithm, Digital Processes, vol.5, p. 27-41, 1979.

[8] A. Ginzburg : Algebraic theory of automata, N.Y.: Academic Press, 1968.

[9] J. Hartmanis : Loop-free structure of sequential machines, Inf. & Control, vol.5, p.25-43, 1962.

[10] J. Hartmanis : Further results on the structure of sequential machines, J. Assoc. Comput. Mach., vol.10,p.78-88, 1963.

[11] J. Hartmanis, R.E. Stearns : Pair algebra and its application to automata theory, Inf. & Control, vol.7, p.485-507, 1964.

[12] J. Hartmanis, R.E. Stearns : Algebraic structure theory of sequential machines, Englewood Cliffs, N.J.: Prentice-Hall, 1966.

[13] W.M.L. Holcombe : Algebraic Automata Theory, Cambridge University Press, 1982. (Cambridge studies in advanced mathematics, vol.1).

[14] Yu.V. Pottosin, E.A. Shestakov : Approximate algorithms for parallel decomposition of automata, Autom. Contr. & Comput. Sci., vol.15, No 2, p.24-31, 1981. (Translation of: Avtom. & Vytchisl. Techn.).

[15] Yu.V. Pottosin, E.A. Shestakov E.A. : Decomposition of an automaton into a two-component network with constraints on internal connections, Autom. Contr. & Comput. Sci., vol.16, No 6, p.24-31, 1982.

[16] M. Yoeli : The cascade decomposition of sequential machines, IRE Trans. Electron. Comput., vol.EC-10, p.587-592, 1961.

[17] M. Yoeli : Cascade-parallel decompositions of sequential machines, IEEE Trans. Electron. Comput., vol.EC-12, p.322-324, 1963.

APPENDIX

Example.

Task: implement machine s1.kis given below with a minimum number of PLA's having 8-bit outputs.

Since the number of output bits of the machine is NOB = 6 and the minimal number of bits needed in order to implement the internal states of the machine is $\lceil \log_2 NS \rceil = 5$ (number of states NS=20), it is impossible to implement the machine with one PLA having 8 bit outputs ($NOB + \lceil \log_2 NS \rceil = 11 > 8$).

So, we have to use at least two such PLA's and to decompose the machine into two submachines.

We performed the task using our decomposition programs. Below, the results reached by the programs for computing the bit serial full-decomposition (a special case of the serial full-decomposition without input and output decoders, but with input and output bits distributed in an appropriate manner among the submachines) are presented.

We reached two submachines:

M_1 (the head machine) with NS = 16 states and NOB = 2 output bits ($NOB + \lceil \log_2 NS \rceil = 6$ bits)

and

M_2 (the tail machine) with NS = 2 states and NOB = 4 output bits ($NOB + \lceil \log_2 NS \rceil = 5$ bits).

Each of these submachines is implementable with PLA having an 8-bit output.

We reached this decomposition in 30 seconds at the APOLLO workstation DN4000.

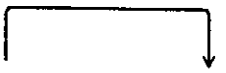
***** MAPPING : (M1⟶M2 ⟹M) *****

Mapping between states S1 and S2 of M1 and M2 and states of s1.kis

S1	S2	
	1	2
1	1	x
2	2	x
3	3	x
4	4	6
5	5	15
6	7	x
7	8	17
8	9	10
9	11	x
10	12	x
11	13	x
12	14	x
13	16	x
14	18	x
15	19	x
16	20	x

* entry = x for don't care

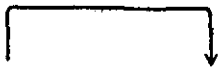
***** MACHINE s1.kis *****



input-vector	present state	next-state	output-vector
-1-00---	1	1	000001
00--0---	1	1	000001
-0--1---	1	2	000011
-1-01---	1	2	000011
01-10---	1	3	001001
11-10---	1	4	011001
-1-11---	1	5	001011
10--0---	1	6	010001
-0------	2	7	000101
-1-0----	2	7	000101
-1-1----	2	8	001101
0---0---	3	3	001001
----1---	3	5	001011
1---0---	3	4	011001
--------	5	8	001101
--0-----	6	9	100001
--1-----	6	10	101001
--------	4	10	101001
-0--1---	7	7	000101
-1-01---	7	7	000101
-1-11---	7	8	001101
00--0---	7	11	000000
-1-00---	7	11	000000
11-10---	7	12	011000
10--0---	7	13	010000
01-10---	7	14	001000
----1---	8	8	001101
0---0---	8	14	001000
1---0---	8	12	011000
00--00--	11	11	000000
00---1-0	11	11	000000
-1-000--	11	11	000000
-1-0-1-0	11	11	000000
00--01-1	11	1	000001
-1-001-1	11	1	000001
-0--11-1	11	2	000011
-1-011-1	11	2	000011
10--01-1	11	6	010001
01-100--	11	14	001000
01-1-1--	11	14	001000
01-110--	11	15	001010
11-1----	11	12	011000
100-10--	11	16	000010
-1-010--	11	16	000010
101-101-	11	16	000010
00--10--	11	16	000010
10--00--	11	13	010000
10---1-0	11	13	010000
101-100-	11	13	010000
0---00--	14	14	001000
0----1-0	14	14	001000
0---01-1	14	3	001001
0---10--	14	15	001010
0---11-1	14	5	001011
1----0--	14	12	011000
1----1-0	14	12	011000
1----1-1	14	4	011001
------0-	15	17	001100
------1-	15	8	001101
-----1-1	12	10	101001
-----0--	12	18	101000
-----1-0	12	18	101000
1-0-----	9	9	100001
1-1-----	9	10	101001
0---1---	9	2	000011
0---0---	9	1	000001
1-------	10	10	101001
0---0---	10	1	000001
0---1---	10	2	000011
---0--1-	16	7	000101
---0--0-	16	19	000100
-0-1----	16	19	000100
-1-1----	16	17	001100
--0--0--	13	20	100000
--0--1-0	13	20	100000
--0--1-1	13	9	100001
--1-----	13	18	101000
----1-0-	17	17	001100
----1-1-	17	8	001101
1---0---	17	12	011000
0---0---	17	14	001000
1----0--	18	18	101000
1----1-0	18	18	101000
0---00--	18	11	000000
0----1-0	18	11	000000
0---01-1	18	1	000001
0---11-1	18	2	000011
1----1-1	18	10	101001
0---10--	18	16	000010
----1-1-	19	7	000101
00--0---	19	11	000000
-1-00---	19	11	000000
01-10---	19	14	001000
11-10---	19	12	011000
10--0---	19	13	010000
-1-11-0-	19	17	001100
-0--1-0-	19	19	000100
-1-01-0-	19	19	000100
1-0--0--	20	20	100000
1-0--1-0	20	20	100000
0---00--	20	11	000000
0----1-0	20	11	000000
0---01-1	20	1	000001
0---10--	20	16	000010
0---11-1	20	2	000011
1-0--1-1	20	9	100001
1-1-----	20	18	101000

***** SUBMACHINE M1 *****

((1),(2),(3),(4,6),(5,15),(7),(8,17),(9,10),(11),(12),(13),(14),(16),(18),(19),(20))



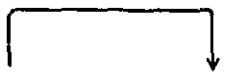
input-vector	present state	next-state	output-vector

inputvector : I1 I2 I3 I4 I5 I6 I7 I8
outputvector : O2 O5

10--0---	1	4	10
-1-11---	1	5	01
11-10---	1	4	10
01-10---	1	3	00
-1-01---	1	2	01
-0--1---	1	2	01
00--0---	1	1	00
-1-00---	1	1	00
-1-1----	2	7	00
-1-0----	2	6	00
-0------	2	6	00
1---0---	3	4	10
----1---	3	5	01
0---0---	3	3	00
--0-----	4	8	00
--1-----	4	8	00
------0-	5	7	00
------1-	5	7	00
01-10---	6	12	00
10--0---	6	11	10
11-10---	6	10	10
-1-00---	6	9	00
00--0---	6	9	00
-1-11---	6	7	00
-1-01---	6	6	00
-0--1---	6	6	00
----1-0-	7	7	00
----1-1-	7	7	00
1---0---	7	10	10
0---0---	7	12	00
1-0-----	8	8	00
1-1-----	8	8	00
0---0---	8	1	00
0---1---	8	2	01
101-100-	9	11	10
10---1-0	9	11	10
10--00--	9	11	10
00--10--	9	13	01
101-101-	9	13	01
-1-010--	9	13	01
100-10--	9	13	01
11-1----	9	10	10
01-110--	9	5	01
01-1-1--	9	12	00
01-100--	9	12	00
10--01-1	9	4	10
-1-011-1	9	2	01
-0--11-1	9	2	01
-1-001-1	9	1	00
00--01-1	9	1	00
-1-0-1-0	9	9	00
-1-000--	9	9	00
00---1-0	9	9	00
00--00--	9	9	00
-----1-0	10	14	00
-----0--	10	14	00
-----1-1	10	8	00
--1-----	11	14	00
--0--1-1	11	8	00
--0--1-0	11	16	00
--0--0--	11	16	00
1----1-1	12	4	10
1----1-0	12	10	10
1----0--	12	10	10
0---11-1	12	5	01
0---10--	12	5	01
0---01-1	12	3	00
0----1-0	12	12	00
0---00--	12	12	00
-1-1----	13	7	00
-0-1----	13	15	00
---0-0--	13	15	00
---0--1-	13	6	00
0---10--	14	13	01
1----1-1	14	8	00
0---11-1	14	2	01
0---01-1	14	1	00
0----1-0	14	9	00
0---00--	14	9	00
1----1-0	14	14	00
1----0--	14	14	00
-1-01-0-	15	15	00
-0--1-0-	15	15	00
-1-11-0-	15	7	00
10--0---	15	11	10
11-10---	15	10	10
01-10---	15	12	00
-1-00---	15	9	00
00--0---	15	9	00
----1-1-	15	6	00
1-1-----	16	14	00
1-0--1-1	16	8	00
0---11-1	16	2	01
0---10--	16	13	01
0---01-1	16	1	01
0----1-0	16	9	00
0---00-0	16	9	00
1-0--1-0	16	16	00
1-0--0--	16	16	00

***** SUBMACHINE M2 *****

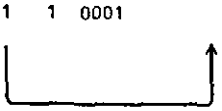
((1,2,3,4,5,7,8,9,11,12,13,14,16,18,19,20),(6,10,15,17))



S1	S2	input-vector	next-state	output-vector

inputvector : 11 12 13 14 15 16 17 18
outputvector : 01 03 04 06

1	1	10--0---	2	0001
1	1	-1-11---	1	0101
1	1	11-10---	1	0101
1	1	01-10---	1	0101
1	1	-1-01---	1	0001
1	1	-0--1---	1	0001
1	1	00--0---	1	0001
1	1	-1-00---	1	0001
1	2	--------	*	----
2	1	-1-1----	1	0111
2	1	-1-0----	1	0011
2	1	-0------	1	0011
2	2	--------	*	----
3	1	1---0---	1	0101
3	1	----1---	1	0101
3	1	0---0---	1	0101
3	2	--------	*	----
4	1	--------	2	1101
4	2	--1-----	2	1101
4	2	--0-----	1	1001
5	1	--------	1	0111
5	2	------1-	1	0111
5	2	------0-	2	0110
6	1	01-10---	1	0100
6	1	10--0---	1	0000
6	1	11-10---	1	0100
6	1	-1-00---	1	0000
6	1	00--0---	1	0000
6	1	-1-11---	1	0111
6	1	-1-01---	1	0011
6	1	-0--1---	1	0011
6	2	--------	*	----
7	1	1---0---	1	0100
7	1	0---0---	1	0100
7	1	----1---	1	0111
7	2	0---0---	1	0100
7	2	1---0---	1	0100
7	2	----1-1-	1	0111
7	2	----1-0-	2	0110
8	1	0---0---	1	0001
8	1	0---1---	1	0001
8	1	1-1-----	2	1101
8	1	1-0-----	1	1001
8	2	0---1---	1	0001
8	2	0---0---	1	0001
8	2	1-------	2	1101
9	1	101-100-	1	0000
9	1	10---1-0	1	0000
9	1	10--00--	1	0000
9	1	00--10--	1	0000
9	1	101-101-	1	0000
9	1	-1-010--	1	0000
9	1	100-10--	1	0000
9	1	11-1----	1	0100
9	1	01-110--	2	0100
9	1	01-1-1--	1	0100
9	1	01-100--	1	0100
9	1	10--01-1	2	0001
9	1	-1-011-1	1	0001
9	1	-0--11-1	1	0001
9	1	-1-001-1	1	0001
9	1	00--01-1	1	0001
9	1	-1-0-1-0	1	0000
9	1	-1-000--	1	0000
9	1	00---1-0	1	0000
9	1	00--00--	1	0000
9	2	--------	*	----
10	1	-----1-0	1	1100
10	1	-----0--	1	1100
10	1	-----1-1	2	1101
10	2	--------	*	----
11	1	--1-----	1	1100
11	1	--0--1-1	1	1001
11	1	--0--1-0	1	1000
11	1	--0--0--	1	1000
11	2	--------	*	----
12	1	1----1-1	1	0101
12	1	1----1-0	1	0100
12	1	1---0---	1	0100
12	1	0---11-1	1	0101
12	1	0---10--	2	0100
12	1	0---01-1	1	0101
12	1	0----1-0	1	0100
12	1	0---00--	1	0100
12	2	--------	*	----
13	1	-1-1----	2	0110
13	1	-0-1----	1	0010
13	1	---0--0-	1	0010
13	1	---0--1-	1	0011
13	2	--------	*	----
14	1	0---10--	1	0000
14	1	1----1-1	2	1101
14	1	0---11-1	1	0001
14	1	0---01-1	1	0001
14	1	0----1-0	1	0000
14	1	0---00--	1	0000
14	1	1----1-0	1	1100
14	1	1----0--	1	1100
14	2	--------	*	----
15	1	-1-01-0-	1	0010
15	1	-0--1-0-	1	0010
15	1	-1-11-0-	2	0110
15	1	10--0---	1	0000
15	1	11-10---	1	0100
15	1	01-10---	1	0100
15	1	-1-00---	1	0000
15	1	00--0---	1	0000
15	1	----1-1-	1	0011
15	2	--------	*	----
16	1	1-1-----	1	1100
16	1	1-0--1-1	1	1001
16	1	0---11-1	1	0001
16	1	0---10--	1	0000
16	1	0---01-1	1	0001
16	1	0----1-0	1	0000
16	1	0---00--	1	0000
16	1	1-0--1-0	1	1000
16	1	1-0--0--	1	1000
16	2	--------	*	----



Eindhoven University of Technology Research Reports
Faculty of Electrical Engineering

ISSN 0167-9708
Coden: TEUEDE

(205) Butterweck, H.J. and J.H.F. Ritzerfeld, M.J. Werter
FINITE WORDLENGTH EFFECTS IN DIGITAL FILTERS: A review.
EUT Report 88-E-205. 1988. ISBN 90-6144-205-2

(206) Bollen, M.H.J. and G.A.P. Jacobs
EXTENSIVE TESTING OF AN ALGORITHM FOR TRAVELLING-WAVE-BASED DIRECTIONAL DETECTION AND PHASE-SELECTION BY USING TWONFIL AND EMTP.
EUT Report 88-E-206. 1988. ISBN 90-6144-206-0

(207) Schuurman, W. and M.P.H. Weenink
STABILITY OF A TAYLOR-RELAXED CYLINDRICAL PLASMA SEPARATED FROM THE WALL BY A VACUUM LAYER.
EUT Report 88-E-207. 1988. ISBN 90-6144-207-9

(208) Lucassen, F.H.R. and H.H. van de Ven
A NOTATION CONVENTION IN RIGID ROBOT MODELLING.
EUT Report 88-E-208. 1988. ISBN 90-6144-208-7

(209) Jóźwiak, L.
MINIMAL REALIZATION OF SEQUENTIAL MACHINES: The method of maximal adjacencies.
EUT Report 88-E-209. 1988. ISBN 90-6144-209-5

(210) Lucassen, F.H.R. and H.H. van de Ven
OPTIMAL BODY FIXED COORDINATE SYSTEMS IN NEWTON/EULER MODELLING.
EUT Report 88-E-210. 1988. ISBN 90-6144-210-9

(211) Boom, A.J.J. van den
H_∞-CONTROL: An exploratory study.
EUT Report 88-E-211. 1988. ISBN 90-6144-211-7

(212) Zhu Yu-Cai
ON THE ROBUST STABILITY OF MIMO LINEAR FEEDBACK SYSTEMS.
EUT Report 88-E-212. 1988. ISBN 90-6144-212-5

(213) Zhu Yu-Cai, M.H. Driessen, A.A.H. Damen and P. Eykhoff
A NEW SCHEME FOR IDENTIFICATION AND CONTROL.
EUT Report 88-E-213. 1988. ISBN 90-6144-213-3

(214) Bollen, M.H.J. and G.A.P. Jacobs
IMPLEMENTATION OF AN ALGORITHM FOR TRAVELLING-WAVE-BASED DIRECTIONAL DETECTION.
EUT Report 89-E-214. 1989. ISBN 90-6144-214-1

(215) Hoeijmakers, M.J. en J.M. Vleeshouwers
EEN MODEL VAN DE SYNCHRONE MACHINE MET GELIJKRICHTER, GESCHIKT VOOR REGELDOELEINDEN.
EUT Report 89-E-215. 1989. ISBN 90-6144-215-X

(216) Pineda de Gyvez, J.
LASER: A LAyout Sensitivity ExploreR. Report and user's manual.
EUT Report 89-E-216. 1989. ISBN 90-6144-216-8

(217) Duarte, J.L.
MINAS: An algorithm for systematic state assignment of sequential machines - computational aspects and results.
EUT Report 89-E-217. 1989. ISBN 90-6144-217-6

(218) Kamp, M.M.J.L. van de
SOFTWARE SET-UP FOR DATA PROCESSING OF DEPOLARIZATION DUE TO RAIN AND ICE CRYSTALS IN THE OLYMPUS PROJECT.
EUT Report 89-E-218. 1989. ISBN 90-6144-218-4

(219) Koster, G.J.P. and L. Stok
FROM NETWORK TO ARTWORK: Automatic schematic diagram generation.
EUT Report 89-E-219. 1989. ISBN 90-6144-219-2

(220) Willems, F.M.J.
CONVERSES FOR WRITE-UNIDIRECTIONAL MEMORIES.
EUT Report 89-E-220. 1989. ISBN 90-6144-220-6

(221) Kalasek, V.K.I. and W.M.C. van den Heuvel
L-SWITCH: A PC-program for computing transient voltages and currents during switching off three-phase inductances.
EUT Report 89-E-221. 1989. ISBN 90-6144-221-4

Eindhoven University of Technology Research Reports
Faculty of Electrical Engineering

ISSN 0167-9708
Coden: TEUEDE

(222) Jóźwiak, L.
THE FULL-DECOMPOSITION OF SEQUENTIAL MACHINES WITH THE SEPARATE REALIZATION OF THE NEXT-STATE AND OUTPUT FUNCTIONS.
EUT Report 89-E-222. 1989. ISBN 90-6144-222-2

(223) Jóźwiak, L.
THE BIT FULL-DECOMPOSITION OF SEQUENTIAL MACHINES.
EUT Report 89-E-223. 1989. ISBN 90-6144-223-0