Decompositional state assignment with reuse of standard designs : using counters as sub-machines and using the method of maximal adjacencies to select the state chains and the state codes

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

Eindhoven
University of Technology
Netherlands

Faculty of Electrical Engineering

Decompositional State Assignment with Reuse of Standard Designs:

Using Counters as Sub-Machines and Using the Method of Maximal Adjacencies to Select the State Chains and the State Codes

by
L. Jóźwiak and T. Spassova-Kwaaitaal

EUT Report 90-E-247
ISBN 90-6144-247-8
September 1990

Eindhoven University of Technology Research Reports
EINDHOVEN UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering
Eindhoven The Netherlands

ISSN 0167-9708 Coden:TEUEDE

DECOMPOSITIONAL STATE ASSIGNMENT WITH REUSE OF STANDARD DESIGNS: USING COUNTERS AS SUB-MACHINES AND USING THE METHOD OF MAXIMAL ADJACENCIES TO SELECT THE STATE CHAINS AND THE STATE CODES

by

L. Józwiak
and
T. Spassova-Kwaaitaal

EUT Report 90-E-247
ISBN 90-6144-247-8

Eindhoven
September 1990

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Jóźwiak, L.

Decompositional state assignments with reuse of standard designs: using counters as sub-machines and using the method of maximal adjacencies to select the state chains and the state codes / by L. Jóźwiak and T. Spassova-Kwaaitaal. - Eindhoven: Eindhoven University of Technology, Faculty of Electrical Engineering. - Fig., tab. - (EUT report, ISSN 0167-9708; 90-E-247)
Met lit. opg., reg.
ISBN 90-6144-247-8
SISO 664 UDC 681.325.65:519.6 NUGI 832
Trefw.: automatentheorie.

DECOMPOSITIONAL STATE ASSIGNMENT WITH REUSE OF STANDARD DESIGNS: USING COUNTERS AS SUB-MACHINES AND USING THE METHOD OF MAXIMAL ADJACENCIES TO SELECT THE STATE CHAINS AND THE STATE CODES

Lech Jóźwiak, T.Spassova-Kwaaitaal

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

Abstract - One of the most important steps in the design of finite state machines is the assignment of values to the binary state variables to represent the symbolic internal states of the machine. The complexity of the resulting implementation can vary extensively from assignment to assignment.

From our experiments with more than 20 sequential machines follows, that the silicon area for the best assignment that we found, was typically about half that for the worst assignment.

The problem of finding an optimal state assignment is computationally complex. It is NP-hard. In a strict sense, it has never been solved, except for exhaustive search, which for large machines is unpractical or impossible, even using a computer. In this situation, some approximated heuristic approaches must be used. Using some knowledge about the internal structure of a sequential machine, these approaches try to reduce the search space to a manageable size and to keep the high quality solutions in that reduced space. They produce often very good solutions, but they do not guarantee the strict optimality for them.

Most of the known heuristic state assignment methods work better for small than for large machines.

For the above reasons, decompositional state assignment approaches are interesting.

If the specification of a given sequential machine (or its part) is strongly similar to the specification of a given standard machine then there is a great chance to reach a better solution by decompositional implementation.

Constructing a modified version of the method of maximal adjacencies [1], we answered the question: how to find the (sub-) optimal sequential decomposition of a given sequential machine into a number of sub-machines defining counters and a small general sequential sub-machine.

The precise algorithm for computing the (sub-) optimal state chains and the (sub-) optimal state codes is described in the report and illustrated with examples.

Index Terms - Automata theory, logic minimization, logic system design, sequential machines.

Acknowledgement - The authors are indebted to Prof.ir.A.Heetman and Prof.ir.M.P.J.Stevens for making it possible to perform this work.

CONTENTS

1. Introduction

The methodology for digital circuits have changed because of the growing complexity of modern IC's. To manage this complexity computer-aided synthesis techniques are used. They ensure functional, logical, electrical and geometrical correctness and allow a decrease in the design time. But often these techniques lead to integrated circuits, which require a large silicon area. Therefore design optimization procedures should be used to yield area-effective circuits.

Basic architecture of a digital system consists of two parts: a processing unit and a control unit. The control unit can often require more than half of the total area, so it is very important to reduce the amount of hardware used by it. Serial processing units constitute also an important class of digital hardware.

Control units and serial processing units can be represented by a finite state machine (FSM) (sequential machine, finite automaton).

Traditional hardware implementation of a FSM consists of two parts: a combinational logic and a state memory (fig. 1.1).

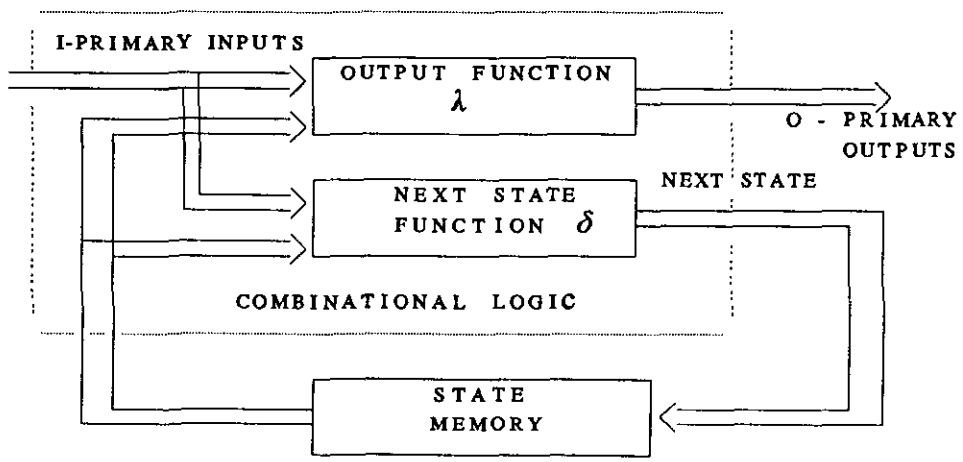

Fig.1.1 General model of a hardware implementation for a sequential machine

The combinational logic realizes the next-state function δ and the output function λ. Depending on the present state and the values

of primary inputs, δ generates the next state and λ generates the values for the primary outputs.

Because of the regular structure of the PLAs they are often used to implement the combinational logic of the FSM in modern designs.

State memory is implemented through binary memory elements (flip-flops).

One of the most important steps in the design of finite state machines is the assignment of values to the binary state variables to represent the symbolic internal states of the machine. The complexity of the resulting implementation can vary extensively from assignment to assignment.

From our experiments with more than 20 sequential machines follows, that the silicon area for the best assignment that we found, was typically about half that for the worst assignment.

The problem of finding an optimal state assignment is computationally complex. It is NP-hard. In a strict sense, it has never been solved, except for exhaustive search, which for large machines is unpractical or impossible, even using a computer. In this situation, some approximated heuristic approaches must be used. Using some knowledge about the internal structure of a sequential machine, these approaches try to reduce the search space to a manageable size and to keep the high quality solutions in that reduced space. They produce often very good solutions, but they do not guarantee the strict optimality for them.

Most of the known heuristic state assignment methods work better for small than for large machines.

For the above reasons, decompositional state assignment approaches are interesting.

2. Basic definitions

A sequential machine is a 5-tuple
$M = (I,S,O,\delta,\lambda)$,
with the following specifications:
I - finite nonempty set of inputs ;
S - finite nonempty set of internal states ;
O - finite set of outputs ;
δ - a mapping, called the next state function,
δ: S x I $\longrightarrow$ S ;
λ - a mapping, called the output function,
λ: S x I $\longrightarrow$ O (a Mealy machine) ;
λ: S $\longrightarrow$ O (a Moore machine).

Sequential machines can be represented by their graphs. The states are represented by the nodes of the graph and the transitions - by the arcs. The arcs are directed. Two nodes may be connected by more than one arc. When two or more arcs have the same start node and the same end node, they are called multiple and the transitions which they represent, are called **multiple transitions.**

Multiple transitions are checked and the largest input subspaces which contain exclusively the inputs from a given multiple transition and don't have any common elements between each other, are found. The multiple transitions between two given states will be implemented together: all of them will be realized by a counter or all by PLA.

When two arcs connect two nodes and have opposite directions, they are called opposite arcs. The nodes which are connected through opposite arcs are called directly interconnected nodes and the states, which they represent are called **directly interconnected states.**

Every node can be described as a **start node**, when only the arcs that are coming out of it are concerned. When, only the arcs that are coming into the nodes are concerned, then the node is described as a **terminal node**.

All the arcs coming out of a certain start node form the **start set** of the node. All the arcs that are coming into a certain terminal node form the **end set** of the node.

The two opposite arcs, which connect two directly interconnected nodes form directly **interconnected pair**.

When an arc starts and ends at the same node, it is called cyclic, and the transition - **cyclic transition**.

Every transition can be described by a start node - S_i, a terminal node - T_i and an input vector - X_i.

So, single transitions are described by a triple of parameters (S_i, T_i, X_i). Multiple transitions are described by a start node S_i, a terminal node T_i and a group of separate input subspaces $X_i...X_m$. So, multiple transitions are described by the n-tuple (S_i, T_i, $X_i...X_m$).

An example for a graph structure of a sequential machine is given on fig.2.1.

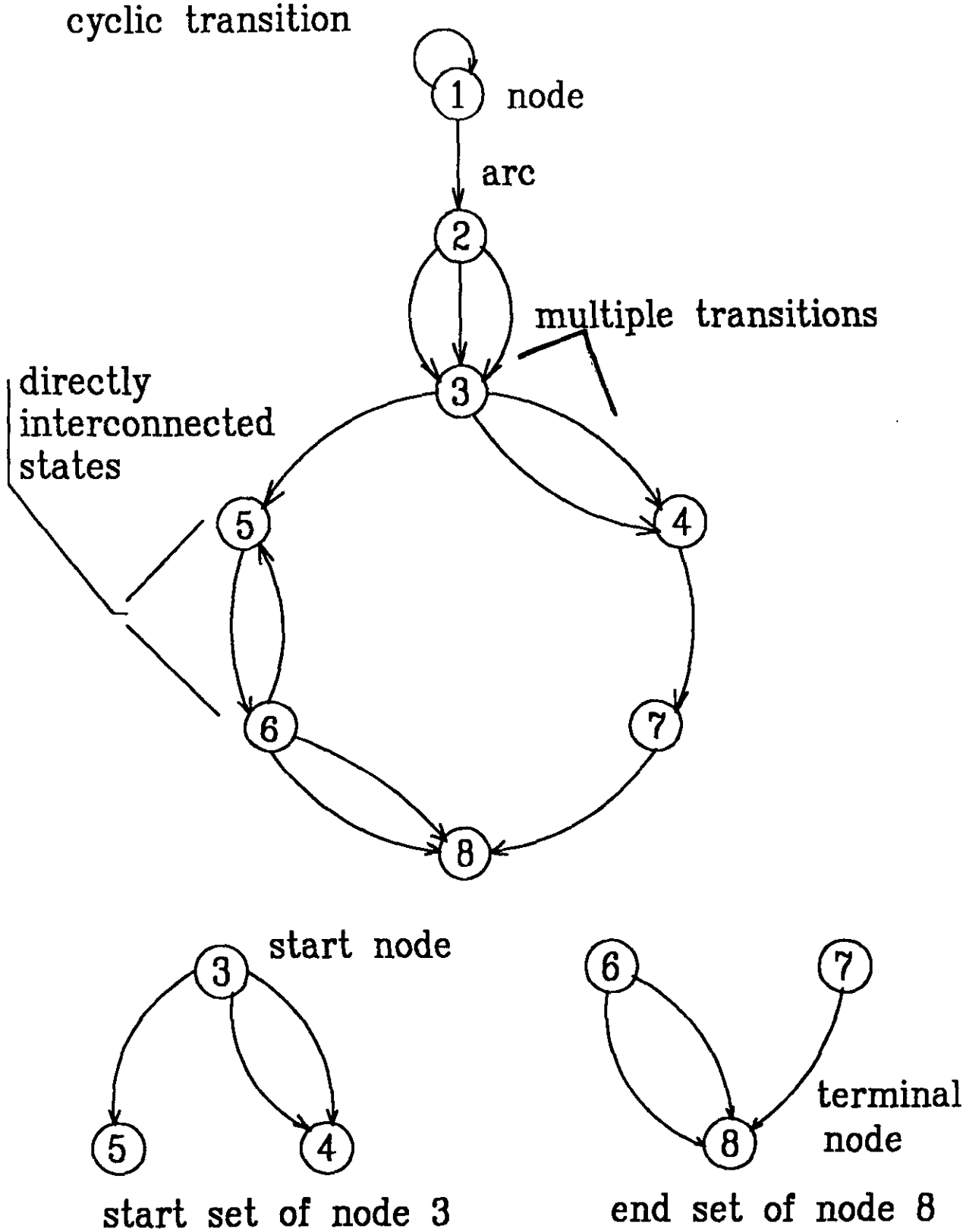

Fig.2.1 Graph structure of a sequential machine

3. Decompositional implementation for a sequential machine

Two sorts of decompositions are feasible for sequential machines:

- simultaneous decompositions;

and

- sequential decompositions.

Simultaneous decompositions divide the process described by a given sequential machine into a number of interacting parallel processes, each implemented by one partial machine. All the partial processes are active simultaneously and together they realize the decomposed process.

Sequential decomposition divide the process described by a given sequential machine into a number of sequential sub-processes, each implemented by one sub-machine. Only one of the sub-processes is active at a given time and all the sub-processes together realize the decomposed process.

In our previous works, we considered simultaneous decompositions with **general partial machines** [4],[5],[6],[7],[8]. The term "general" means that a partial machine can be any sequential machine. We considered also sequential decompositions with general partial machines. Our results in the last field will be published in the nearest future.

In this work , we will consider some decompositions with **special partial machines**, i.e. partial machines which are (more or less) predefined.

This sort of decompositions is very important, because it allows reuse of earlier designs.

A limited reuse of earlier designs can be obtained by using parameterized generators, which describe some classes of circuits, e.g. k-bit long counter of the natural binary code, k-bit Gray-code counter, parameterized filter, parameterized CRC coder or checker etc. In this case, reuse of a design is limited to a given class of

circuits defined by a given generator.

Often the specification of a circuit to design (or its part) does not meet completely the specification of a special machine designed previously, but it is only very similar to the special machine. In such case, the earlier design can still be reused but not as the only circuit. The second circuit must be constructed. The newly designed circuit together with a given special machine designed earlier must realize a machine which meets the specification of a circuit to design.

Generally, in order to reuse the earlier designs a given sequential machine must be decomposed into a number of special partial machines (representing the earlier designs) and a number of general partial machines (representing the new part of the design). All the partial machines together must realize the decomposed machine.

Reuse of earlier designs is important not only because it decreases the design time and design costs. It is specially important to reuse the standard designs which are designed very carefully. Such designs ensure the correctness and are optimal or near-optimal from the complexity point of view.

If a large enough part of a circuit will be constructed using standard optimal designs and a small part of it will be constructed in the form of a general machine, it is a great chance to reach an optimal or near optimal solution. There is a chance to reach better solution than that offered by the heuristic state assignment tools applied to a general sequential machine on a whole. One reason for it is the fact that one large part of a circuit will be optimal due to using the optimal standard design; the second one is the fact that the second (general) part will be much smaller than the whole circuit and therefore easier to optimize. However, one must remember that the standard part and the general part will be optimized in separation. So, the eventual common parts of the logic cannot be shared in this case. It means, that having ideal optimization tools (e.g. state assignment tools), it will be always possible to find no worse general solution than any solution obtained by reusing of earlier designs.

So, the only reason, for the solution obtained by reusing the earlier designs to be better (from the complexity point of view) than the best solution found in a general way, is the unideal heuristic character of the optimization tools.

If only a small part of a circuit will be implemented by reusing the standard designs, then the chance to reach a better solution will be small. The impossibility to share the logic by the standard and the general part as well as the necessity to implement a piece of hardware that will ensure the proper cooperation of those two parts, will probably cause the hardware overhead greater than the overhead caused by the unideal character of the optimization tools.

So, the following two problems have to be solved:
- to discover that the specification of a given design (or a part of it) is strongly similar to the specification of a given standard design (i.e. the chance to reach a better solution is high enough);
- to find the optimal decomposition of a given design into standard and general parts.

In this work, we tried to analyze and to solve the above described problems using a counter as an example of a standard (special) machine. The choice of this example was not random. Counters itself constitute a very important class of sequential circuits and, what is more important in this context, many practical controllers can be designed as modified counters quite well.

We were interested in the decompositional implementation for a sequential machine using sequential decomposition into a number of counters (standard machines) and one general machine. (Since we developed earlier a method for sequential decomposition of a general machine into a number of general machines, the above formulated problem is general enough).

Sequential decomposition consists in partitioning a given sequential machine into a number of sub-machines and ensuring the proper cooperation of this sub-machines in order to realize the behaviour of the machine to decompose.

We considered the one state realization of the state and output behaviour in the sense of the definition given in [4]. "Sub-machine" is understood here in the sense defined by the algebraic theory of sequential machines.

A machine $M' = (I', S', O', \delta', \lambda')$ is a **sub-machine** of a machine $M = (I, S, O, \delta, \lambda)$ if and only if:

$S' \leq S$, $I' \leq I$, $O' \leq O$,

$\delta' = \delta$ restricted to $S' \times I'$ ($\delta' : S' \times I' \longrightarrow S'$),

$\lambda' = \lambda$ restricted to $S' \times I'$ or S'

($\lambda' : S' \times I' \longrightarrow O'$ or $\lambda' : S' \longrightarrow O'$).

In other words, sequential decomposition partitions the graph of a given sequential machine into a number of sub-graphs. Each sub-graph is implemented then as a separated sub-machine. One must ensure that only one of the sub-machines is active at a given time. So, if a given sub-machine is in one of its active states and from this state for a given input value another sub-machine should be activated, the active sub-machine must suspend his work and, at the same time, it must activate another sub-machine. From this time on, the activated sub-machine must keep itself active up to a similar situation as described above (i.e. it must perform its normal work) and each suspended sub-machine (also the newly suspended) will be suspended up to the activation by a sub-machine which is active at the moment of the activation.

Since sequential decomposition partitions the graph of a sequential machine into a number of sub-graphs, the graph of a sequential machine will be a very useful tool in considering sequential decompositions.

The machine's graph is defined as

$MG = < V, E, L(V), L(E) >$

where

V - the set of vertices corresponding to the set of machine's states;

E - the set of edges corresponding to the set of transitions between the states;

L(V) - a set of labels attached to each vertex; this set is empty for a Mealy machine and each label represents the output value in a given state for a Moore machine;

L(E) - a set of labels attached to edges; each label represents the input value of the machine (for Moore machine) or the input and the output value (for Mealy machine) corresponding to a given transition represented by the edge.

The transitions in the machine's graph represent the actions performed by a sequential machine.

The only action that can be performed by a counter is "go to the next state". So, the only type of a sequential machine that can be realized using a counter consists of a sequence of successive states and is represented by a path in the machine's graph.

Since we considered here only the one state realizations, only one state can be made successive and only one state can be made previous to a given state. In general, we can have more than one next state for a given state and different inputs ("fork" splitting in a machine's graph) and transitions to a given state can have an origin in more than one state ("join" action in a machine's graph). It follows immediately that not every sequential machine can be implemented by composing only some counters (only counters can be implemented using only counters) and that at most one transition from a given "fork" or "join" group can be implemented by a counter.

So, in order to construct a one state realization of a sequential machine in the decompositional form using a sequential decomposition into counters we must use also a general sequential machine as one of the sub-machines (the only exception is the case of a counter itself).

The optimal choice of transitions from the "fork" and "join" groups to be implemented by a counter will be considered in the next chapters. All the transitions which will be realized by a counter will be referred to as **counting transitions**. The counting transitions can form a number of counting chains (paths in a graph). Each counting chain describes a sub-machine that can be realized by a counter. However, instead of implementing each counter representing a sub-machine in separate, we will implement all of them together using one larger counter and differentiate its sub-ranges to implement different counters - sub-machines.

A general sub-machine can be implemented using a PLA (for a combinational logic) and a register (for a state memory). However, in a sequential decomposition only one of the sub-machines is active at a given time. So, it is possible to share the flip-flops of a counter and to use them one time in the counter configuration (if one of the counters - sub-machines is active) and another time in a register configuration (if a general sub-machine is active). In consequence, in order to implement the counter - sub-machines and the memory of the general sub-machine a loadable binary counter will be used. The load signal (L) will distinguish between the active states of a general sub-machine and the active states of one of the counter - sub-machines.

The aim of our decomposition is to minimize the total silicon area for implementation.

The silicon area for PLA grows with the number of state variables k and with the number of product terms used for realizing the next-state and output functions [1].

The silicon area for a counter grows with the number of state variables which for the minimum-length assignments grows logarithmic with the number of states implemented in a counter ($k = \lceil \log_2 |S| \rceil$).

We decide to use a counter if most of the transitions (and states) can be implemented using it. So, we can expect in the practice at most one-bit growth of a counter due to the extra transitions and states of a general sub-machine. Therefore, we will not consider minimization of a counter.

In the sequel we present a method which heuristically minimizes the area used for the PLA implementation of the combinatorial logic by replacing some of the state transitions by counting transitions. These counting transitions will then be realized by a loadable binary counter, which replaces the traditional feedback register of the FSM. Selecting of the counting transitions has been considered for the first time in [2]; however, it has been done in relation to the very low quality state assignment algorithm - KISS [3]. In order to select the counting transitions we developed a special edition of the method of maximal adjacencies [1] . Figure 3.1 shows the basic counter-based PLA structure using a loadable counter.

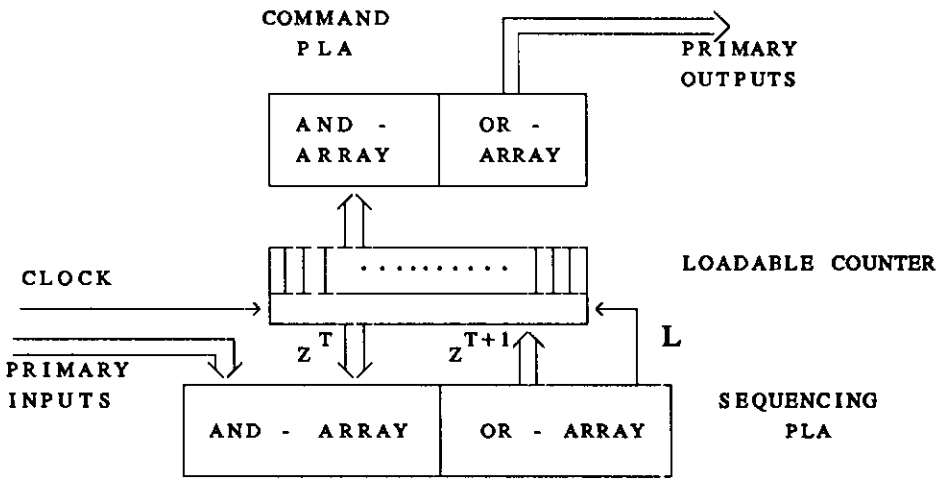

Fig.3.1 Basic counter-based PLA structure

This structure implements FSM's with Moore specification, since the output vector depends only on the actual state Z^T; however, very similar structure implementing Mealy machine can be constructed also.

In the next chapters, the suboptimal selection of counting transitions and suboptimal state assignment will be considered.

4. A new method for state assignment

To solve the problem of sub-optimal state assignment for an implementation of a sequential machine using a counter and a PLA, we have developed a method, which is a combination of the method of maximal adjacencies (MMA) [1] with the concept of using a counter as a state memory .

The MMA [1] has been developed, based upon the observation, that the information contained in the next-state and the output tables of sequential machines instructs the input-state, present state- next state and output-state dependencies for adjacency conditions. The adjacency conditions are ordered according to the number of adjacent "1"s and "0"s in the binary functions representing δ and λ reached

when a given condition is satisfied by the assignment. Then, the conditions are considered and combined in their order constructing some "suboptimal" assignments. Combining the maximal number of the best adjacency conditions makes an order in the Karnaugh tables which represent the binary next state and output functions by accumulating "1"s together and "0"s together, i.e. by constructing small number of large product terms. These "suboptimal" assignments lead to small PLA area.

Using a counter [2] makes it possible to realize a number of state transitions by a counter. Counting transitions need not be implemented by a PLA. This saves the PLA area. The feedback register of the FSM is replaced by a loadable binary counter, which is controlled by a load variable L. Hence, an extra output called "load" is generated by the combinatorial logic of the FSM. In each cycle the counter offers a binary code word. If the next state of the sequential machine is identical to the next counting state, the counter has only to be incremented, while L = 0. Otherwise, the counter has to be loaded from the PLA due to L = 1.

Our method uses the basic ideas of the method of maximal adjacencies in order to find the transitions that should be realized by the counter in order to minimize the PLA area.

Two main objectives are taken in mind:

1. Since each transition, which is generated by the counter allows "don't cares" in the next state table for the transitions that are left to be realized by PLA and therefore an additional potential for further minimization, we have to look for as many counting transitions as possible.

2. In order to be realized by PLA, each transition has a certain cost. The idea is to implement the transitions as cheap as possible. That means, that the most expensive transitions should be realized by the counter and the cheapest ones - by PLA.

We consider that when we choose a certain transition to be realized by a counter, the neighbouring transitions (from the same start set, the same end set and the same directly interconnected pair) should be realized by PLA. On that base we calculate 5 parameters.

1. C_{cmax_i} - maximal cost of a given transition i, when realized by a counter.

$$C_{cmax_i} = \sum_{j=1}^{n} C_{PLAmax_j}$$

C_{PLAmax_j} - maximal cost of a given transition to be realized in PLA

n - number of neighbouring transitions

The maximal cost is calculated as a sum of the maximal costs of the neighbouring transitions, which are left to be realized in PLA. It is calculated as the number of product terms that are necessary to be realized for a given transition without satisfying any restrictions or adjacency conditions.

2. C_{cmin_i} - minimal cost of a given transition i when realized by a counter.

$$C_{cmin_i} = \sum_{j=1}^{n} C_{PLAmin_j}$$

n - number of neighbouring transitions

C_{PLAmin_j} - minimal cost of a given transition j to be realized in PLA

C_{cmin} is calculated as a sum of the minimal costs of the neighbouring transitions which should be realized in PLA. The minimal cost is calculated as the number of product terms, that are necessary to be realized for a given transition when certain adjacency conditions and restrictions are satisfied.

3. C_{PLAmax_i} - maximal cost of a given transition i to be realized in PLA

It is calculated as the number of product terms that are necessary to realize a given transition without satisfying any restrictions or adjacency conditions.

4. C_{PLAmin_i} - minimal cost of a given transition i to be realized in PLA

It is calculated as the number of product terms that are necessary to realize a given transition when certain adjacency conditions and restrictions are satisfied.

5. AdjRestr - adjacency restrictions

It is defined as the number of adjacency restrictions which should be satisfied in order to reach the calculated C_{cmin} for a given transition.

These 5 parameters are estimated and based on their values and the weights preliminary given for each one of the parameters, an order of the transitions is made using a multicriterial optimization method.

The quality factor which is calculated for each transition is highest in value for the transition which is best to be realized by counter. So in the final list, the transitions are ordered according to their quality factors and the transitions with the highest quality factor is first on the list.

From all the transitions in the list the one that should be realized by the counter are chosen. They are called counting transitions, because they are not implemented by a PLA but they are realized by only incrementing the value in the counter.

When choosing the counting transitions two objectives are taken in mind:

1. Every transition realized by a counter saves the PLA area. So, as much as possible transitions should be realized by the counter, i.e. the number of the counting transitions should be as large as possible.

2. From all the transitions in the list, for counting transitions should be chosen the one with the highest quality factors, that means, the one which are the most expensive when implemented in PLA.

The basic activities of a sequential machine can be described formally as a "sequence", "spread choice" and "join choice".

A "sequence" is when for a given present state there is only one next state and both states are not interconnected. That means that the two nodes are connected only by one arc. In this case, the transition is always realized by a counter which gives the code of the next state.

"Spread choice" is an activity when from a given present state there is a possibility to go to two or more different next states.
In this case one of the transitions can be realized by the counter and the rest are implemented by a decision logic in PLA which gives the code of the next state.

"Join choice" is an activity when a given state can be reached by a number (more then one) previous states. In this case one of the transitions can be realized by the counter and the rest are implemented by a decision logic in PLA which gives the code of the next state.

A sequential machine can be decomposed into a number of partial machines which realize the activities described above.

We assume that the "sequence" machines and some of the transitions of "spread choice" and "join choice" machines are realized by one counter. The rest of the transitions of "spread choice" and "join choice" machines are realized with one combinational logic block.

Determining the counting transitions

For every state we can form two sets.

Previous state set which is defined as follows:

$PREVSS = \{ PREVS_i \mid PREVS_i$ is a previous state of the given state $\}$

and

Next state set which is defined as:

$NSS = \{ NEXTS_i \mid NEXTS_i$ is a next state of the given state $\}$

From these two sets for a given state we have to choose one previous state and one next state, which define the two transitions from the "join choice" and "spread choice" machines which will be realized by the counter.

Using the list with the ordered transitions according to their quality factor and a beam search algorithm one or more lists of counting transitions are made.

In the beam search algorithm two parameters are used:

MaxBeams - maximum number of beams

QC - quality coefficient

$$QC = \frac{QFalt}{QFbestalt}$$ where

QFalt - quality factor for the alternative transition

QFbestalt - quality factor for the best alternative transition

MaxBeams and QC should be chosen experimentally. After the first experiments good results were obtained when

MaxBeams = 3 and QC = 0.9

When a given transition is chosen to be realized by a counter, some of the neighbouring transitions become noncounting, i.e. they can't be used as counting transitions. Noncounting transitions are the transitions from the same NSS for the start node of the chosen counting transition and the same PRESS for the end node of the transition and the other transition from the interconnected pair.

The set of counting transitions is determined as

SCT = { CT_i | CT_i is a counting transition }

and is created in the following manner:

1. All the transitions which construct the "sequence" machines are put in the set SCT and are taken away from the list of the ordered transitions.
2. The best transition from the list of ordered transitions is taken and put into the SCT.
3. The noncounting transitions are determined and are removed from the list of the ordered transitions.
4. The algorithm continues with the next best transition until the list of ordered transitions is empty.

Choosing the type of the counter

In general, each type of counter can be used to implement the counting transitions and the best counter should be chosen; however, counters of different types impose different requirements on "successive" states and codes. For example, for the natural binary code counter, "successive" means the next binary coded number. In the future, we are going to give possibility to use different counters; however,at the moment, the reflected binary (Gray code) counter is used. This decision is imposed by the fact that in the Gray code only one digit change occurs when passing from any one combination to the next. That means that the successive states are also adjacent, or the succession condition for the counting transitions from one chain is given by the adjacency

condition for the successive states from the chain.

These adjacency conditions should be satisfied obligatory and they are called **primary adjacency conditions.**

Using the adjacency conditions instead of succession conditions allows applying a slightly modified version of the method of maximal adjacencies [1] for the state assignment.

Constructing the chains

The counting transitions are combined in chains in the following manner.

We begin by taking one of the counting transitions. It forms the first chain. We try to connect to it the next of the counting transitions. If this is not possible we form a new chain. When a counting transition is connected to already existing chain, before proceeding with the next transition we check whether the extended chain can be connected to any of the other chains.

When all chains are constructed they are checked if they are closed, i.e. if the start node of the chain is connected to the end node of the same. If such transition exists it is removed from the chain.

Using MAXAD for state assignment

MAXAD is a program which has been developed on the base of method of maximal adjacencies (MMA). This method [1] creates for a given assignment length k, a set of final families of partitions that maximize the adjacency level of "1"s and "0"s of Boolean functions obtained with a given family used for state assignment. Calculations are based only on the information from the next state and output tables. First the adjacency conditions for input - state, present state - next state and state - output dependencies are determined. Then these three sorts of adjacency conditions are combined together and ordered, according to the offered level of adjacency, forming the ordered list of adjacency conditions. The last step consists of creating the final families of partitions based on the ordered list of adjacency conditions. In this step the adjacency conditions are considered in the order of their ordered list and the final families of partitions are created

that satisfy the greatest number of compatible adjacency conditions.

After the determination of the counting transitions the state transition table, describing the sequential machine is changed as all the transitions which will be realized by the counter are replaced by don't cares. Further an additional variable called "load" is included in the state transition table. The load variable is set to one only for the transitions, which cannot be replaced by counting transitions. The next state code word is provided by the PLA and is loaded into the counter. For all counting transitions L is zero and the next state code is provided by simply incrementing the counter.

The modified description of the sequential machine is used as an input file for MAXAD. The adjacency conditions for input-state, present state-next state and state-output dependencies are determined. The adjacency conditions are combined and an ordered list is formed. These adjacency conditions are additional to the primary adjacency conditions which follow from the already generated counting chains and they are called **secondary adjacency conditions.** The primary adjacency conditions should be satisfied obligatory, while from the secondary adjacency conditions we have to satisfy as much as possible.

The satisfaction of the adjacency conditions is done while we construct the final families of partitions(FFP). While combining the adjacency conditions in order to form FFP's the following constraints should be taken into account[1]:

1. If two states $S_k|S_l$ have to be adjacent, then they must be contained in two different blocks of just one two-block partition, which is a member of a FFP. In all other partitions from the FFP, they must be contained in one block.

2. Each pair of incompatible states (S_m,S_n) must be separated in at least one partition from a FFP (separate condition).

3. Only proper partitions are useful for state assignment and only they can be members of FFP's.

4. The FFP for a minimal machine is an orthogonal family of proper partitions.

5. Each state $S_k:S_k \in S$ may be adjacent with at most k other states $S_l:S_l \in S$.

The mechanism of constructing a limited set of near optimal families of final partitions for an assignment length k, SNOFFP(k) is described in [1].

Each FFP $\in$ SNOFFP(k) must contain k proper partitions i.e. k two-block partitions containing at most 2^{k-1} elements in each of their blocks.

We start by constructing partial proper partitions which satisfy all primary adjacency conditions. And after that by trying to satisfy as much as possible secondary adjacency conditions we build the SNOFFP. For every FFP from this set by changing the blocks within the partitions and the places of the partitions we find a construction for which the states which are members of the counting chains can be assigned with codes which correspond to the Gray code sequence.

5. Evaluation of transitions in relation to adjacency conditions.

At present, we evaluate transitions taking into account only the adjacency conditions for the next-state function; however, the adjacency conditions for the output function can be taken into account in a very similar way.

We estimate the cost to realize in PLA for each transition in relation to the surrounding transitions. Cyclic transitions must be realized by PLA, because it's not possible to realize them by counter. If some of the other transitions are related to them, they can be realized in PLA cost free or for a very low cost. In this case some adjacency conditions should be fulfilled.

Single transitions are described by the triple of parameters $(S_iT_iX_i)$ where
- S_i - start node
- T_i - terminal node
- X_i - input vector

Multiple transitions are described by the n-tuple of parameters $(S_iT_iX_i...X_m)$ where
- S_i - start node
- T_i - terminal node
- $X_i...X_m$ - separate input subspaces

In connection to adjacency conditions the following cases are possible:

5.1 $S_i \equiv T_i$ and $S_i \equiv S_j$

5.1.1 The transitions are $S_iT_iX_i$ and $S_jT_jX_j$

If $X_i \mid X_j$ cost = 1

5.1.2 The transitions are $S_iT_iX_i...X_m$ and $S_jT_jX_j...X_n$

5.1.2.1 If $X_i...X_m$ and $X_j...X_n$ form one subspace

cost = n

5.1.2.2 If $X_i \mid X_j$ cost = k(n-1)+1

k - number of variables used for state coding

5.1.2.3 If $X_i \mid X_j$ ... and $X_m \mid X_n$ *l*- adjacencies

cost = k(n-*l*)+*l*

For these cases the adjacency condition is $S_j \mid T_j$

5.2 $S_i \equiv S_j$

5.2.1 The transitions are $S_iT_iX_i$ and $S_jT_jX_j$

If $X_i \mid X_j$ cost = $\frac{k+1}{2}$

5.2.2 The transitions are $S_iT_iX_i$ and $S_jT_jX_j...X_n$

5.2.2.1 If $X_i \mid X_j$ cost = $\frac{k\,n+1}{n+1}$

5.2.2.2 If X_i and $X_j \ldots X_n$ form one subspace

$$\text{cost} = \frac{k+n}{1+n}$$

5.2.3 The transitions are $S_iT_iX_i \ldots X_m$ and $S_jT_jX_j \ldots X_n$

5.2.3.1 If $X_i \mid X_j$ $\quad$ cost $= \frac{k(m+n)-k+1}{m+n}$

5.2.3.2 If $X_i \mid X_j \ldots$ and $X_m \mid X_n$ $\quad$ l- adjacencies

$$\text{cost} = \left(\frac{k(m+n)-lk+l}{m+n}\right) m$$

5.2.3.3 If $X_i \ldots X_m$ and $X_j \ldots X_n$ form one subspace

$$\text{cost} = \left(\frac{k-1+m+n}{m+n}\right) m$$

For these cases the adjacency condition is $T_i \mid T_j$

5.3 $S_i \equiv T_i$ and $T_i \equiv T_j$

5.3.1 The transitions are $S_iT_iX_i$ and $S_jT_jX_j$

If $X_i \equiv X_j$ $\quad$ cost $= 0$

5.3.2 The transitions are $S_iT_iX_i \ldots X_m$ and $S_jT_jX_j \ldots X_n$

If $X_i \equiv X_j \ldots\ldots X_m \equiv X_n$

$$\text{cost} = 0$$

The adjacency condition for these cases is $S_i \mid S_j$

5.4 $S_i \equiv T_j$ $\quad$ $T_i \equiv S_j$

5.4.1 The transitions are $S_iT_iX_i$ and $S_jT_jX_j$

If $X_i \equiv X_j$ cost = $\frac{k+1}{2}$

5.4.2 The transitions are $S_iT_iX_i$ and $S_jT_jX_j...X_n$

If $X_i \equiv X_j$ cost = $\frac{kn+1}{n+1}$

5.4.3 The transitions are $S_iT_iX_i...X_m$ and $S_jT_jX_j...X_n$

If $X_i \equiv X_j$ ..... $X_m \equiv X_n$ in l cases

cost = $(\frac{k(m-l+n)+l}{n+m})$ m

The adjacency condition for these cases is $S_i | S_j$

5.5 $T_i \equiv T_j$

5.5.1 The transitions are $S_iT_iX_i$ and $S_jT_jX_j$

If $X_i \equiv X_j$ cost = $\frac{k}{2}$

5.5.2 The transitions are $S_iT_iX_i$ and $S_jT_jX_j...X_n$

If $X_i \equiv X_j$ cost = $\frac{kn}{n+1}$

5.5.3 The transitions are $S_iT_iX_i...X_m$ and $S_jT_jX_j...X_n$

If $X_i \equiv X_j$ ..... $X_m \equiv X_n$ in l cases

cost = $(\frac{(m-l+n)k}{m+n})$ m

The adjacency condition for these cases is $S_i | S_j$

6. Algorithm description.

6.1 Read the transition table.

6.2 Check the multiple transitions. Find the largest input subspaces which contain exclusively the inputs from a given multiple transition and don't have any common elements between each other.

6.3 For each state a set of next states to which that state leads is found. If S is the number of states, N_s sets are constructed .

For i = 1 to S

Next_state_set(i) = {$next\_state_j$,$times_j$,$input\_vectors_j$ |
$next\_state_j$-next state for present state i,
$times_j$-number of $input\_vectors_j$,
$input\_vectors_j$-input vectors for a transition
from $present\_state_i$ to $next\_state_j$}

6.4 For each state, a set of previous states leading to that state is found. N_s sets are constructed.

For i = 1 to S

Previous_state_set (i) = {$previous\_state_j$,$times_j$,$input\_vectors_j$ |
$previous\_state_j$-previous state of $state_i$,
$times_j$-number of $input\_vectors_j$,
$input\_vectors_j$-input vectors for a transition from $previous\_state_j$ to $state_i$}

6.5 Check for each state if there exists a directly interconnected state.

If $S_i \equiv T_j$ and $S_j \equiv T_i$

then make $Interconnected\_pair_{i,j}$

$Interconnected\_pair_{i,j}$ = {$state_i$,$state_j$,input_vectors;
$state_j$,$state_i$,input_vectors}

6.6 Make a set of cyclic transitions.

Cyclic_transition_set$_i$ = {state$_i$,input_vectors$_i$|state$_i$-state with cyclic transition}

6.7 Generation of subgraphs.

6.7.1 Read one transition.

6.7.2 For the start state of the transition take the next_state_set.

6.7.3 For the end state of the transition take the previous_state_set.

6.7.4 Check for directly interconnected pairs.

6.7.5 Combine the three sets. They form one subgraph.

6.8 Calculation of the cost of each transition to be realized in PLA.

6.8.1 Calculation of the maximal cost.

6.8.2 Calculation of the minimal cost.

It is based on the condition that some of the states are coded with adjacent codes and is calculated according to p. 5 .

For every transition check if it is related to

1. Cyclic transition - $T_i \equiv S_i$
   a) check if $T_i \equiv T_j$ and $X_i \equiv X_j$
      then $S_i \mid S_j$
   b) check if $S_i \equiv S_j$ and $X_i \mid X_j$
      then $S_j \mid T_j$

2. Another transition
   a) check if $S_i \equiv S_j$ and $X_i \mid X_j$
      then $T_i \mid T_j$
   b) check if $T_i \equiv T_j$ and $X_i \equiv X_j$
      then $S_i \mid S_j$

3. To directly interconnected transitions
   Check if $S_i \equiv T_j$ and $T_j \equiv S_j$ and $X_i \equiv X_j$
   then $S_i \mid S_j$

6.9 Calculation of the cost for each transition to be realized by a counter.

6.9.1 Calculation of maximal cost - it is calculated as a sum of the maximal costs that are necessary for all noncyclic transitions from the same subgraph to be realized by PLA.

6.9.1.1 Take one subgraph.

6.9.1.2 Choose one transition.

6.9.1.3 For the rest of the transitions calculate the maximal cost C_{cmax_i} as follows

$$C_{cmax_i} = \sum_{j=1}^{n} C_{PLAmax_j}$$

C_{PLAmax_j} - maximal cost of a given transition to be realized in PLA

n - number of neighbouring transitions

6.9.2 Calculation of minimal cost

6.10 Calculation of the number of adjacency restrictions - AdjRestr

6.11 Estimate C_{cmax}, C_{cmin}, C_{PLAmax}, C_{PLAmin} and AdjRestr for every transition and using a multicriterial optimization method make one suboptimal order of the transitions. The quality factor will be highest for the transition which is best to be realized by a counter.

6.12 Using a beam search algorithm construct one or more suboptimal lists of transitions for the counter containing as many transitions with the highest quality factors as possible

6.13 Construct the counting chains and derive the primary adjacency conditions.

6.14 In the next state table describing the sequential machine replace with don't cares all the transitions which are members of the counting chains.

6.15 Include an additional variable called "load" in the state table of the machine and set it to 1 only for the transitions which cannot be replaced by counting transitions. For the rest of the transitions it is set to 0.

6.16 Use the modified description of the machine as an input file for MAXAD and determine the adjacency conditions for input-state, present state-next state and state-output dependencies.

6.17 Construct partial proper partitions satisfying all primary adjacency conditions.

6.18 Build the SNOFFP by trying to satisfy as much as possible secondary adjacency conditions.

6.19. By changing the blocks within the partitions and the places of the partitions find an appropriate constructions for the FFP from the SNOFFP for which the counting states can be assigned with codes which correspond to the Gray code sequence.

7. EXAMPLES

All steps of the algorithm will be explained using the following two examples.

7.1 Example I

7.1.1 The sequential machine is described by the following next-state table.

Table 7.1.1

S \ I	x_0 00	x_1 01	x_2 11	x_3 10
1	2	1	3	3
2	2	3	3	4
3	1	1	1	4
4	2	2	2	2

Next-state table

7.1.2 The multiple transitions are checked and the largest input subspaces which contain exclusively the inputs from a given multiple transition and don't have any common elements between each other are found. By the integer 2 don't care bits in the input vectors are represented. The input vectors stand for the largest input subspaces and they are given in Table 7.1.2.

Table 7.1.2

present state	next state	input vector
1	3	12
1	1	01
1	2	00
2	3	21
2	4	10
2	2	00
3	4	10
3	1	02 21
4	2	22

7.1.3 For each state a set of next states to which that state leads is found. For S=4 (S - number of states) N_4 sets are constructed.

N_1 = { 3,1,12; 1,1,01; 2,1,00 }
N_2 = { 3,1,21; 4,1,10; 2,1,00 }
N_3 = { 4,1,10; 1,2,02,21 }
N_4 = { 2,1,22 }

7.1.4 For each state, a set of previous states leading to that state is found. For S = 4, P_4 sets are constructed.

P_1 = { 3,2,02,21; 1,1,01 }
P_2 = { 4,1,22; 2,1,00; 1,1,00}
P_3 = { 2,1,21; 1,1,12 }
P_4 = { 3,1,10; 2,1,10 }

7.1.5 Check for each state if there exists a directly interconnected state and make interconnected pairs.

IP_{13} = { 1,3,12; 3,1,02,21 }
IP_{24} = { 2,4,10;4,2,22 }

7.1.6 Make a set of cyclic transitions

There are cyclic transitions for states 1 and 2 and the sets are the following.

C_1 = { 1,01 }
C_2 = { 2,00 }

7.1.7 Generation of subgraphs

For the easier description of the algorithm we shall give names to the transitions. They are shown in Table 7.1.3.

Table 7.1.3

present state	next state	input vector	transition name
1	2	00	A
1	1	01	B
1	3	12	C
2	2	00	D
2	4	10	E
2	3	21	F
3	1	02	G
3	1	21	I
3	4	10	H
4	2	22	J

By combining the next state set for the start state with the previous state set for the end state of each noncyclic transition and the directly interconnected pairs, the following subgraphs are generated. Multiple transitions are considered together and the generated subgraph is one for them.

$SG_A \longrightarrow N_1$ and P_2 are combined

$SG_C \longrightarrow N_1$ and P_3 and IP_{13} are combined

$SG_E \longrightarrow N_2$ and P_4 and IP_{24} are combined

$SG_F \longrightarrow N_2$ and P_3 are combined

$SG_{GI} \longrightarrow N_3$ and P_1 and IP_{13} are combined

$SG_H \longrightarrow N_3$ and P_4 are combined

$SG_J \longrightarrow N_4$ and P_2 and IP_{24} are combined

7.1.8 Calculation of the cost of each transition to be realized in PLA.

7.1.8.1 Calculation of maximal cost.

It is calculated as the number of product terms that are necessary to realize a given noncyclic transition without satisfying any restrictions or adjacency conditions.

The following maximal costs are calculated:

$C_{PLAMAX_A} = 2$

$C_{PLAMAX_C} = 2$

$C_{PLAMAX_E} = 2$

$C_{PLAMAX_F} = 2$

$C_{PLAMAX_{GI}} = 4$

$C_{PLAMAX_H} = 2$

$C_{PLAMAX_J} = 2$

7.1.8.2 Calculation of minimal cost

It is calculated as the number of product terms that are necessary to realize a given noncyclic transition when certain adjacency conditions and restrictions are satisfied.

The evaluation of transitions in connection to adjacency conditions is explained in p.5.

The following minimal costs are calculated for the different transitions:

C_{PLAMIN_A} = 0 (related to transition D, adjacency condition - 1|2)

C_{PLAMIN_C} = 2 (not related to any other transition)

C_{PLAMIN_E} = 1 (related to transition D, adjacency condition - 2|4)

C_{PLAMIN_F} = 2 (not related to any other transition)

$C_{PLAMIN_{GI}}$ = 2,7 (related to transition H, adjacency condition - 1|4)

C_{PLAMIN_H} = 1 (related to transition E, adjacency condition - 2|3)

C_{PLAMIN_J} = 2 (not related to any other transition)

7.1.9 Calculation of the cost for each transition to be realized by a counter.

7.1.9.1 Calculation of maximal cost

It is calculated as a sum of the maximal costs that are necessary for all noncyclic transitions from the same subgraph to be realized by PLA.

$$C_{cmax_i} = \sum_{j=1}^{n} C_{PLAmax_j}$$

C_{PLAmax_j} - maximal cost of a given transition to be realized in PLA

n - number of neighbouring transitions

The maximal cost for the transitions is calculated as follows:

C_{CMAX_A} = 4 (in SG_A noncyclic transitions G and J are left)

C_{CMAX_C} = 8 (in SG_c noncyclic transitions A,F,G and I are left)

C_{CMAX_E} = 6 (in SG_E noncyclic transitions F,H and J are left)

C_{CMAX_F} = 4 (in SG_F noncyclic transitions E and C are left)

$C_{CMAX_{GI}}$ = 4 (in SG_{GI} noncyclic transitions H and C are left)

C_{CMAX_H} = 6 (in SG_H noncyclic transitions G,I and E are left)

C_{CMAX_J} = 4 (in SG_J noncyclic transitions A and E are left)

7.1.9.2 Calculation of minimal cost

It is calculated as a sum of the minimal costs that are necessary for all noncyclic transitions from the same subgraph to be realized by PLA.

$$C_{cmin_i} = \sum_{j=1}^{n} C_{PLAmin_j}$$

n - number of noncyclic transitions

C_{PLAmin_j} - minimal cost of a given transition j to be realized in PLA

The following minimal costs are calculated:

C_{CMIN_A} = 4 (in SG_A noncyclic transitions C and J are left)

C_{CMIN_C} = 4.7 (in SG_C noncyclic transitions A,F,G and I are left; adjacency conditions 1|2, 1|4)

C_{CMIN_E} = 5.3 (in SG_E noncyclic transitions F,H and J are left; adjacency condition 1|4)

C_{CMIN_F} = 3 (in SG_F noncyclic transitions E and C are left; adjacency condition 2|4)

$C_{CMIN_{GI}}$ = 3 (in SG_{GI} noncyclic transitions C and H are left; adjacency condition 2|3)

C_{CMIN_H} = 5 (in SG_H noncyclic transitions G,I and E are left; adjacency condition 2|4)

C_{CMIN_J} = 1 (in SG_J noncyclic transitions A and E are left; adjacency conditions 1|2, 2|4)

Remark: When we assume that a certain transition should be realized by a counter, we should not calculate the minimal cost of any neighbouring transition in relation to that transition. We should search for relation with another transition. This is done in connection with the calculation of the costs for the transitions E and H.

7.1.10 Calculation of the number of the adjacency restrictions - NAR

It is calculated as the number of the adjacency restrictions, which should be satisfied in order to reach the calculated C_{CMIN} for a given transition.

$NAR_A = 0$

$NAR_C = 2$

$NAR_E = 1$

$NAR_F = 1$

$NAR_{GI} = 1$

$NAR_H = 1$

$NAR_J = 2$

7.1.11 For every transition we have calculated five parameters. By using a multicriterial optimization method we make an order of the transitions such that the transitions which are best to be realized by a counter will be at the top of the list.

For the example we have reached the following order.

present state	next state	transition
3	1	GI
2	3	F
4	2	J
1	2	A
3	4	H
1	3	C
2	4	E

7.1.12 By using a beam search algorithm we construct the following suboptimal list of counting transitions.

present state	next state	transition
3	1	GI
2	3	F
4	2	J

7.1.13 The constructed counting chain is as follows:

$$4 \longrightarrow 2 \longrightarrow 3 \longrightarrow 1$$

The following primary adjacency conditions are derived.

$$2 \mid 4 \ , \ 2 \mid 3 \ , \ 1 \mid 3$$

We see that all the states of the machine are members of the counting chain and they should be coded in such a way as to form a Gray code sequence.

7.1.14 In the next state table we replace by don't cares all the transitions which are members of the counting chains.

S \ I	x_0 00	x_1 01	x_2 11	x_3 10
1	2	1	3	3
2	2	–	–	4
3	–	–	–	4
4	–	–	–	–

7.1.15 Set the additional variable called "load" to 1 only for the transitions which cannot be replaced by counting transitions. For the rest - set it to zero.

S \ I	x_0 00	x_1 01	x_2 11	x_3 10
1	1	1	1	1
2	1	0	0	1
3	0	0	0	1
4	0	0	0	0

L

7.1.16 Calculate the adjacency conditions for input-state, present state-next state and state-output dependencies using the next state table with the counting transitions replaced by "don't cares". This is done according to the method described in [1].

For our example we obtained the following ordered list of adjacency conditions.

pair of adjacent states	pairs of next states and number of their occurencies	number of unconditionally reached adjacencies	estimation of the total number of adjacencies	number of "don't cares"
2|3	———	9	9	4
2|4	———	9	9	4
3|4	———	8	8	4
1|4	———	8	8	4
1|3	3|4 / 1	7	8	3
1|2	3|4 / 1	7	8	3

Ordered list of adjacency conditions

7.1.17 Construct partial proper partitions satisfying all primary adjacency conditions

According to 7.1.13 the following primary adjacency conditions were calculated

$$2 \mid 4 \ , \ 2 \mid 3 \ , \ 1 \mid 3$$

Each FFP $\in$ SNOFFP must contain 2 proper partitions i.e. 2 two - block partitions containing at most 2 elements in each of their blocks. We consider the adjacency conditions one after the other and construct the following FFP.

$2|4 : \ \{ \overline{2} , \overline{4} \} \{ \overline{2,4} , \overline{0} \}$

$2|3 : \ \{ \overline{2,3} , \overline{4} \} \{ \overline{2,4} , \overline{3} \}$

$1|3 : \ \{ \overline{2,3} , \overline{1,4} \} \{ \overline{2,4} , \overline{1,3} \}$

We have reached only one FFP so it is not possible to satisfy any of the secondary adjacency conditions.

7.1.18 By changing the blocks within the partitions and the places of the partitions we find an appropriate construction for the FFP for which the counting states can be assigned with codes which correspond to the Gray code sequence.

One possible construction is:

$\{ \overline{2,4} , \overline{1,3} \} \{ \overline{1,4} , \overline{2,3} \}$

For the assignment of blocks with

$\overline{2,4}$ - 0; $\overline{1,3}$ - 1; $\overline{1,4}$ - 0; $\overline{2.3}$ - 1

the following assignment of states is reached:

4 - 00
2 - 01
3 - 11
1 - 10

The Gray code sequence between the counting states is fulfilled.

The assigned next state table is as follows:

	S^I	x_0 00	x_1 01	x_2 11	x_3 10
1 -	10	01	10	11	11
2 -	01	01	--	--	00
3 -	11	--	--	--	00
4 -	00	--	--	--	--
					S_1S_2

In the assigned next state table we add an additional variable L which has the value of 1 for the transitions which cannot be replaced by counting transitions, and 0 otherwise.

	S \ I	x_0 00	x_1 01	x_2 11	x_3 10
1 -	10	011	101	111	111
2 -	01	011	--0	--0	001
3 -	11	--0	--0	--0	001
4 -	00	--0	--0	--0	--0
					S_1S_2 L

We make the Karnaugh maps in order to calculate the number of terms which are necessary to realize S_1, S_2 and L

S \ I	I_1I_2 00	I_1I_2 01	I_1I_2 11	I_1I_2 10
00	-	-	-	-
01	0	-	-	0
11	-	-	-	0
10	0	1	1	1

2 terms

Karnaugh map of S_1

S \ I	I_1I_2 00	I_1I_2 01	I_1I_2 11	I_1I_2 10
00	-	-	-	-
01	1	-	-	0
11	-	-	-	0
10	1	0	1	1

2 terms

Karnaugh map of S_2

S \ I	I_1I_2 00	I_1I_2 01	I_1I_2 11	I_1I_2 10
00	0	0	0	0
01	1	0	0	1
11	0	0	0	1
10	1	1	1	1

3 terms

Karnaugh map of L

By using a counter-based PLA structure the next-state functions can be implemented with 7 terms as shown on fig.7.1.

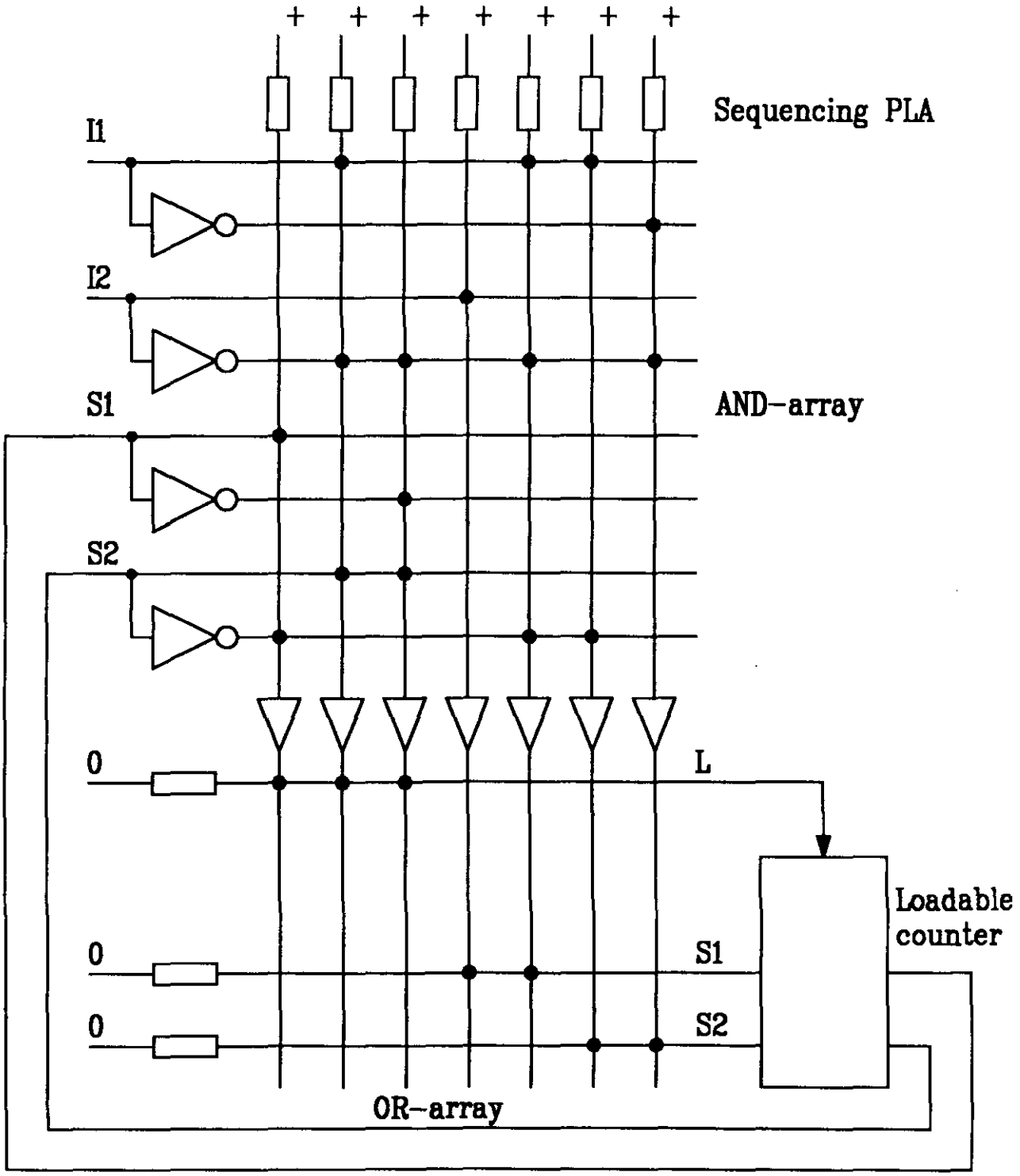

Fig. 7.1 Implementation by using counter based PLA structure.

7.2 Example II

7.2.1 The sequential machine is described by the following next-state table.

Table 7.2.1

S \ I	x_0 00	x_1 01	x_2 11	x_3 10
1	2	2	2	2
2	3	4	5	3
3	3	3	4	4
4	1	4	5	1
5	6	6	5	6
6	7	7	9	9
7	1	8	8	1
8	9	9	9	9
9	1	1	9	9

Next-state table

7.2.2 The multiple transitions are checked and the largest input subspaces which contain exclusively the inputs from a given multiple transition and don't have any common elements between each other are found. By the integer 2 don't care bits in the input vectors are represented. The input vectors stand for the largest input subspaces and they are given in Table 7.2.2.

Table 7.2.2

present state	next state	input vector
1	2	22
2	5	11
2	4	01
2	3	20
3	4	12
3	3	02
4	5	11
4	4	01
4	1	20
5	5	11
5	6	20 02
6	9	12
6	7	02
7	8	21
7	1	20
8	9	22
9	9	12
9	1	02

7.2.3 For each state a set of next states to which that state leads is found. For S=9 (S - number of states) N_9 sets are constructed.

N_1 = { 2,1,22 }
N_2 = { 5,1,11; 4,1,01; 3,1,20 }
N_3 = { 4,1,12; 3,1,02 }
N_4 = { 5,1,11; 4,1,01; 1,1,20 }
N_5 = { 5,1,11; 6,2,20,02 }
N_6 = { 9,1,12; 7,1,02 }
N_7 = { 8,1,21; 1,1,20 }
N_8 = { 9,1,22 }
N_9 = { 9,1,12; 1,1,02 }

7.2.4 For each state, a set of previous states leading to that state is found. For S = 9, P_9 sets are constructed.

P_1 = { 9,1,02; 7,1,20; 4,1,20 }
P_2 = { 1,1,22 }
P_3 = { 3,1,02; 2,1,20 }
P_4 = { 4,1,01; 3,1,12; 2,1,01 }
P_5 = { 5,1,11; 4,1,11; 2,1,11 }
P_6 = { 5,2,20,02 }
P_7 = { 6,1,02 }
P_8 = { 7,1.21 }
P_9 = { 9,1,12; 8,1,22; 6,1,12 }

7.2.5 Check for each state if there exists a directly interconnected state and make interconnected pairs.
For this example directly interconnected pairs don't exist.

7.2.6 Make a set of cyclic transitions

There are cyclic transitions for states 3,4,5 and 9 and the sets are the following.

$C_3 = \{ 3,02 \}$
$C_4 = \{ 4,01 \}$
$C_5 = \{ 5,11 \}$
$C_9 = \{ 9,12 \}$

7.2.7 Generation of subgraphs

For the easier description of the algorithm we shall give names to the transitions. They are shown in Table 7.2.3.

Table 7.2.3

present state	next state	input vector	transition name
1	2	22	A
2	5	11	D
2	4	01	C
2	3	20	B
3	4	12	F
3	3	02	E
4	5	11	I
4	4	01	H
4	1	20	G
5	5	11	L
5	6	20	J
5	6	02	K
6	9	12	N
6	7	02	M
7	8	21	P
7	1	20	O
8	9	22	Q
9	9	12	S
9	1	02	R

By combining the next state set for the start state with the previous state set for the end state of each noncyclic transition and the directly interconnected pairs, the following subgraphs are generated. Multiple transitions are considered together and the generated subgraph is one for them.

$SG_A \longrightarrow N_1$ and P_2 are combined

$SG_B \longrightarrow N_2$ and P_3 are combined

$SG_C \longrightarrow N_2$ and P_4 are combined

$SG_D \longrightarrow N_2$ and P_5 are combined

$SG_F \longrightarrow N_3$ and P_4 are combined

$SG_G \longrightarrow N_4$ and P_1 are combined

$SG_I \longrightarrow N_4$ and P_5 are combined

$SG_{JK} \longrightarrow N_5$ and P_6 are combined

$SG_M \longrightarrow N_6$ and P_7 are combined

$SG_N \longrightarrow N_6$ and P_9 are combined

$SG_O \longrightarrow N_7$ and P_1 are combined

$SG_P \longrightarrow N_7$ and P_8 are combined

$SG_Q \longrightarrow N_8$ and P_9 are combined

$SG_R \longrightarrow N_9$ and P_1 are combined

7.2.8 Calculation of the cost of each transition to be realized in PLA.

7.2.8.1 Calculation of maximal cost.

It is calculated as the number of product terms that are necessary to realize a given noncyclic transition without satisfying any restrictions or adjacency conditions.

The following maximal costs are calculated:

$C_{PLAMAX_A} = 4$

$C_{PLAMAX_B} = 4$

$C_{PLAMAX_C} = 4$

$C_{PLAMAX_D} = 4$

$C_{PLAMAX_F} = 4$

$C_{PLAMAX_G} = 4$

$C_{PLAMAX_I} = 4$

$C_{PLAMAX_{JK}} = 8$

$C_{PLAMAX_M} = 4$

$C_{PLAMAX_N} = 4$

$C_{PLAMAX_O} = 4$

$C_{PLAMAX_P} = 4$

$C_{PLAMAX_Q} = 4$

$C_{PLAMAX_R} = 4$

7.2.8.2 Calculation of minimal cost

It is calculated as the number of product terms that are necessary to realize a given noncyclic transition when certain adjacency conditions and restrictions are satisfied.

The evaluation of transitions in connection to adjacency conditions is explained in p.5.

The following minimal costs are calculated for the different transitions:

$C_{PLAMIN_A} = 4$ (not related to any other transition)

$C_{PLAMIN_B} = 4$ (not related to any other transition)

$C_{PLAMIN_C} = 0$ (related to transition H, adjacency condition - 2|4)

$C_{PLAMIN_D} = 0$ (related to transition L, adjacency condition - 2|5)

$C_{PLAMIN_F} = 1$ (related to transition E, adjacency condition - 3|4)

$C_{PLAMIN_G} = 2$ (related to transition O, adjacency condition - 4|7)

$C_{PLAMIN_I} = 0$ (related to transition L, adjacency condition - 4|5)

$C_{PLAMIN_{JK}} = 2$ (related to transition L, adjacency condition - 5|6)

$C_{PLAMIN_M} = 2.5$ (related to transition N, adjacency condition - 7|9)

C_{PLAMIN_N} = 0 (related to transition S, adjacency condition - 6|9)

C_{PLAMIN_O} = 2 (related to transition G, adjacency condition - 7|4)

C_{PLAMIN_P} = 2.5 (related to transition O, adjacency condition - 8|1)

C_{PLAMIN_Q} = 4 (not related to any other transition)

C_{PLAMIN_R} = 1 (related to transition S, adjacency condition - 1|9)

7.2.9 Calculation of the cost for each transition to be realized by a counter.

7.2.9.1 Calculation of maximal cost

It is calculated as a sum of the maximal costs that are necessary for all noncyclic transitions from the same subgraph to be realized by PLA.

$$C_{cmax_i} = \sum_{j=1}^{n} C_{PLAmax_j}$$

C_{PLAmax_j} - maximal cost of a given transition to be realized in PLA

n - number of neighbouring transitions

The maximal cost for the transitions is calculated as follows:

C_{CMAX_A} = 0 (SG_A contains only the transition A)

C_{CMAX_B} = 8 (in SG_B noncyclic transitions C and D are left)

C_{CMAX_C} = 12 (in SG_C noncyclic transitions B,F,D are left)

C_{CMAX_D} = 12 (in SG_D noncyclic transitions B,C and I are left)

C_{CMAX_F} = 4 (in SG_F noncyclic transition C is left)

C_{CMAX_G} = 12(in SG_G noncyclic transitions I,O and R are left)

C_{CMAX_I} = 8 (in SG_I noncyclic transitions G and D are left)

$C_{CMAX_{JK}}$ = 0 (SG_{JK} contains only the transition IK)

C_{CMAX_M} = 4 (in SG_M noncyclic transition N is left)

C_{CMAX_N} = 8 (in SG_N noncyclic transitions M and Q are left)

C_{CMAX_O} =12 (in SG_O noncyclic transitions P,G and R are left)

C_{CMAX_P} = 4 (in SG_P noncyclic transition O is left)

C_{CMAX_R} = 8 (in SG_R noncyclic transitions G and O are left)

C_{CMAX_Q} = 4 (in SG_Q noncyclic transition N is left)

7.2.9.2 Calculation of minimal cost

It is calculated as a sum of the minimal costs that are necessary for all noncyclic transitions from the same subgraph to be realized by PLA.

$$C_{cmin_i} = \sum_{j=1}^{n} C_{PLAmin_j}$$

n - number of noncyclic transitions

C_{PLAmin_j} - minimal cost of a given transition j to be realized in PLA

The following minimal costs are calculated:

C_{CMIN_A} = 0 (SG_A contains only the transition A)

C_{CMIN_B} = 0 (in SG_B noncyclic transitions C and D are left; adjacency conditions 2|4, 2|5)

C_{CMIN_C} = 5 (in SG_C noncyclic transitions B,D and F are left; adjacency conditions 2|5, 3|4)

C_{CMIN_D} = 4 (in SG_D noncyclic transitions B,C and I are left; adjacency conditions 2|4, 4|5)

C_{CMIN_F} = 0 (in SG_F noncyclic transition C is left; adjacency condition 2|4)

C_{CMIN_G} = 3,5 (in SG_G noncyclic transitions I,O and R are left; adjacency conditions 4|5, 1|8, 1|9)

C_{CMIN_I} = 2 (in SG_I noncyclic transitions G and D are left; adjacency condition 4|7, 2|5)

$C_{CMIN_{JK}}$ = 0 (SG_{JK} contains only the transition JK)

C_{CMIN_M} = 0 (in SG_M noncyclic transition N is left; adjacency condition 6|9)

C_{CMIN_N} = 8 (in SG_N noncyclic transitions M and Q are left;

C_{CMIN_O} = 9 (in SG_O noncyclic transitions P,G and R are left;
adjacency condition 1|9)

C_{CMIN_P} = 2 (in SG_P noncyclic transition O is left;
adjacency condition 7|4)

C_{CMIN_Q} = 0 (in SG_Q noncyclic transition N is left;
adjacency condition 6|9)

C_{CMIN_R} = 4 (in SG_R noncyclic transitions G and O are left;
adjacency condition 7|4)

Remark: When we assume that a certain transition should be realized by a counter, we should not calculate the minimal cost of any neighbouring transition in relation to that transition. We should search for relation with another transition. This is done in connection with the calculation of the costs for the transitions G, N and O.

7.2.10 Calculation of the number of the adjacency restrictions - NAR

It is calculated as the number of the adjacency restrictions, which should be satisfied in order to reach the calculated C_{CMIN} for a given transition.

$NAR_A = 0$

$NAR_B = 2$

$NAR_C = 2$

$NAR_D = 2$

$NAR_F = 1$

$NAR_G = 3$

$NAR_I = 2$

$NAR_{JK} = 0$

$NAR_M = 1$

$NAR_N = 0$

$NAR_O = 1$

$NAR_P = 1$

$NAR_Q = 1$

$NAR_R = 1$

7.2.11 For every transition we have calculated five parameters. By using a multicriterial optimization method we make an order of the transitions such that the transitions which are best to be realized by a counter will be at the top of the list.

For the example we have reached the following order.

present state	next state	tran s i tion
1	2	A
5	6	J K
8	9	Q
6	7	M
7	8	P
2	3	B
3	4	F
9	1	R
4	5	I
6	9	N
4	1	G
7	1	O
2	5	D
2	4	C

7.2.12 By using a beam search algorithm we construct the following suboptimal list of counting transitions.

present state	next state	tran s i tion
5	6	J K
6	7	M
7	8	P
8	9	Q
9	1	R
1	2	A
2	3	B
3	4	F

7.2.13 The constructed counting chain is as follows:

$5 \longrightarrow 6 \longrightarrow 7 \longrightarrow 8 \longrightarrow 9 \longrightarrow 1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4$

The following primary adjacency conditions are derived.

5 | 6 , 6 | 7 , 7 | 8 , 8 | 9 , 9 | 1 , 1 | 2 , 2 | 3, 3 | 4

We see that all the states of the machine are members of the counting chain and they should be coded in such a way as to form a Gray code sequence.

7.2.14 In the next state table we replace by don't cares all the transitions which are members of the counting chains.

S \ I	x_0 00	x_1 01	x_2 11	x_3 10
1	-	-	-	-
2	-	4	5	-
3	3	3	-	-
4	1	4	5	1
5	-	-	5	-
6	-	-	9	9
7	1	-	-	1
8	-	-	-	-
9	-	-	9	9

7.2.15 Set the additional variable called "load" to 1 only for the transitions which cannot be replaced by counting transitions. For the rest - set it to zero.

S \ I	x_0 00	x_1 01	x_2 11	x_3 10
1	0	0	0	0
2	0	1	1	0
3	1	1	0	0
4	1	1	1	1
5	0	0	1	0
6	1	1	0	0
7	1	0	0	1
8	0	0	0	0
9	0	0	1	1

L

7.2.16 Calculate the adjacency conditions for input-state, present state-next state and state-output dependencies using the next state table with the counting transitions replaced by "don't cares". This is done according to the method described in [1].

For this example we obtained the following ordered list of adjacency conditions.

pair of adjacent states	pairs of next states and number of their occurencies	number of unconditionally reached adjacencies	estimation of the total number of adjacencies	number of "don't cares"
4|9	1,4 | 1,9 2 | 3	6	16	2
4|6	1,9 3	9	15	3
6|7	1,9 3	9	15	3
7|9	1,9 3	9	15	3
1|9	——	14	14	4
1|4	——	13	13	4
1|2	——	12	12	4
1|3	——	12	12	4
1|5	——	12	12	4
1|6	——	12	12	4

Ordered list of adjacency conditions

pair of adjacent states	pairs of next states and number of their occurencies	number of uncondi-tionally reached adjacencies	estimation of the total number of adja-cencies	number of "don't cares"
1|7	———	12	12	4
1|8	———	12	12	4
2|4	———	12	12	4
2|5	———	12	12	4
2|7	———	12	12	4
2|8	———	12	12	4
3|5	———	12	12	4
3|8	———	12	12	4
3|6	———	12	12	4
7|8	———	12	12	4

Ordered list of adjacency conditions (cont.)

pair of adjacent states	pairs of next states and number of their occurencies	number of unconditionally reached adjacencies	estimation of the total number of adjacencies	number of "don't cares"
4|5	——	12	12	4
4|7	——	12	12	4
4|8	——	12	12	4
5|7	——	12	12	4
5|8	——	12	12	4
6|8	——	12	12	4
6|9	——	12	12	4
8|9	——	12	12	4
2|9	1,4 | 5,9 2 | 1	6	12	2
2|3	3,4 1	9	11	2

Ordered list of adjacency conditions (cont.)

pair of adjacent states	pairs of next states and number of their occurencies	number of unconditionally reached adjacencies	estimation of the total number of adjacencies	number of "don't cares"
2|6	5, 9 / 1	9	11	3
3|7	1, 3 / 1	9	11	3
5|6	5, 9 / 1	9	11	3
5|9	—	11	11	3
3|9	1, 3 / 2	6	10	2
3|4	1|3 / 1	7	9	2

Ordered list of adjacency conditions (cont.)

7.2.17 Construct partial proper partitions satisfying all primary adjacency conditions.

According to 7.2.13 the following primary adjacency conditions were calculated

5 | 6 , 6 | 7 , 7 | 8 , 8 | 9 , 9 | 1 , 1 | 2 , 2 | 3, 3 | 4

Each FFP $\in$ SNOFFP must contain 4 proper partitions i.e. 4 two-block partitions containing at most 8 elements in each of their blocks. We consider the adjacency conditions one after the other and construct the following FFP.

(3,4,5,9,8 1,2,6,7)(1,7,9,8 2,3,4,5,6)(1,2,3,4,9 5,6,7,8)(4 1,2,3,5,6,7,9,8)
(4,5,9,8 1,2,3,6,7)(1,7,9,8 2,3,4,5,6)(1,2,3,4,9 5,6,7,8)(3,4 1,2,5,6,7,9,8)
(5,9,8 1,2,3,4,6,7)(1,4,7,9,8 2,3,5,6)(1,2,3,4,9 5,6,7,8)(3,4 1,2,5,6,7,9,8)
(5,9,8 1,2,3,4,6,7)(1,7,9,8 2,3,4,5,6)(1,2,3,9 4,5,6,7,8)(3,4 1,2,5,6,7,9,8)
(3,4,5,9,8 1,2,6,7)(1,2,3,7,9,8 4,5,6)(1,2,3,4,9 5,6,7,8)(2,3,4 1,5,6,7,9,8)
(3,4,5,9,8 1,2,6,7)(1,2,3,4,7,9,8 5,6)(1,2,3,9 4,5,6,7,8)(2,3,4 1,5,6,7,9,8)
(4,5,9,8 1,2,3,6,7)(1,2,7,9,8 3,4,5,6)(1,2,3,4,9 5,6,7,8)(2,3,4 1,5,6,7,9,8)
(5,9,8 1,2,3,4,6,7)(1,2,7,9,8 3,4,5,6)(1,2,3,9 4,5,6,7,8)(2,3,4 1,5,6,7,9,8)
(5,9,8 1,2,3,4,6,7)(1,2,7,9,8 3,4,5,6)(1,2,3,4,9 5,6,7,8)(2,3 1,4,5,6,7,9,8)
(4,5,9,8 1,2,3,6,7)(1,2,3,4,7,9,8 5,6)(1,2,9 3,4,5,6,7,8)(2,3,4 1,5,6,7,9,8)
(5,9,8 1,2,3,4,6,7)(1,2,3,7,9,8 4,5,6)(1,2,9 3,4,5,6,7,8)(2,3,4 1,5,6,7,9,8)
(1,5,9,8 2,3,4,6,7)(3,4,7,9,8 1,2,5,6)(1,2,3,4,9 5,6,7,8)(4 1,2,3,5,6,7,9,8)
(1,4,5,9,8 2,3,6,7)(7,9,8 1,2,3,4,5,6)(1,2,3,4,9 5,6,7,8)(3,4 1,2,5,6,7,9,8)
(1,5,9,8 2,3,4,6,7)(4,7,9,8 1,2,3,5,6)(1,2,3,4,9 5,6,7,8)(3,4 1,2,5,6,7,9,8)
(1,5,9,8 2,3,4,6,7)(7,9,8 1,2,3,4,5,6)(1,2,3,9 4,5,6,7,8)(3,4 1,2,5,6,7,9,8)
(1,2,5,9,8 3,4,6,7)(4,7,9,8 1,2,3,5,6)(1,2,3,4,9 5,6,7,8)(2,3,4 1,5,6,7,9,8)
(1,2,5,9,8 3,4,6,7)(7,9,8 1,2,3,4,5,6)(1,2,3,9 4,5,6,7,8)(2,3,4 1,5,6,7,9,8)
(1,2,5,9,8 3,4,6,7)(7,9,8 1,2,3,4,5,6)(1,2,3,4,9 5,6,7,8)(2,3 1,4,5,6,7,9,8)
(1,2,3,5,9,8 4,6,7)(3,4,7,9,8 1,2,5,6)(1,2,3,4,9 5,6,7,8)(2,3,4 1,5,6,7,9,8)
(1,2,3,4,5,9,8 6,7)(3,4,7,9,8 1,2,5,6)(1,2,3,9 4,5,6,7,8)(2,3,4 1,5,6,7,9,8)
(1,2,3,5,9,8 4,6,7)(7,9,8 1,2,3,4,5,6)(1,2,9 3,4,5,6,7,8)(2,3,4 1,5,6,7,9,8)
(1,2,3,4,5,9,8 6,7)(4,7,9,8 1,2,3,5,6)(1,2,9 3,4,5,6,7,8)(2,3,4 1,5,6,7,9,8)
(1,4,5,9,8 2,3,6,7)(1,2,7,9,8 3,4,5,6)(1,2,3,4,9 5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,5,9,8 2,3,4,6,7)(1,2,7,9,8 3,4,5,6)(1,2,3,9 4,5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,5,9,8 2,3,4,6,7)(1,2,7,9,8 3,4,5,6)(1,2,3,4,9 5,6,7,8)(1,2,3 4,5,6,7,9,8)
(1,4,5,9,8 2,3,6,7)(1,2,3,4,7,9,8 5,6)(1,2,9 3,4,5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,5,9,8 2,3,4,6,7)(1,2,3,7,9,8 4,5,6)(1,2,9 3,4,5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,5,9,8 2,3,4,6,7)(1,2,3,7,9,8 4,5,6)(1,2,3,4,9 5,6,7,8)(1,2 3,4,5,6,7,9,8)
(1,2,5,9,8 3,4,6,7)(1,4,7,9,8 2,3,5,6)(1,2,3,4,9 5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,2,5,9,8 3,4,6,7)(1,7,9,8 2,3,4,5,6)(1,2,3,9 4,5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,2,5,9,8 3,4,6,7)(1,7,9,8 2,3,4,5,6)(1,2,3,4,9 5,6,7,8)(1,2,3 4,5,6,7,9,8)
(1,2,3,5,9,8 4,6,7)(1,7,9,8 2,3,4,5,6)(1,2,9 3,4,5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,2,3,4,5,9,8 6,7)(1,4,7,9,8 2,3,5,6)(1,2,9 3,4,5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,2,3,5,9,8 4,6,7)(1,7,9,8 2,3,4,5,6)(1,2,3,4,9 5,6,7,8)(1,2 3,4,5,6,7,9,8)
(1,2,5,9,8 3,4,6,7)(1,2,3,7,9,8 4,5,6)(1,9 2,3,4,5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,2,5,9,8 3,4,6,7)(1,2,3,4,7,9,8 5,6)(1,4,9 2,3,5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,2,3,5,9,8 4,6,7)(1,2,7,9,8 3,4,5,6)(1,9 2,3,4,5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,2,3,4,5,9,8 6,7)(1,2,7,9,8 3,4,5,6)(1,4,9 2,3,5,6,7,8)(1,2,3,4 5,6,7,9,8)
(1,4,5,9 2,3,6,7,8)(7,9,8 1,2,3,4,5,6)(1,2,3,4,9,8 5,6,7)(3,4 1,2,5,6,7,9,8)
(1,5,9 2,3,4,6,7,8)(4,7,9,8 1,2,3,5,6)(1,2,3,4,9,8 5,6,7)(3,4 1,2,5,6,7,9,8)
(1,5,9 2,3,4,6,7,8)(7,9,8 1,2,3,4,5,6)(1,2,3,9,8 4,5,6,7)(3,4 1,2,5,6,7,9,8)
(1,2,5,9 3,4,6,7,8)(4,7,9,8 1,2,3,5,6)(1,2,3,4,9,8 5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,5,9 3,4,6,7,8)(7,9,8 1,2,3,4,5,6)(1,2,3,9,8 4,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,5,9 3,4,6,7,8)(7,9,8 1,2,3,4,5,6)(1,2,3,4,9,8 5,6,7)(2,3 1,4,5,6,7,9,8)
(1,2,3,5,9 4,6,7,8)(3,4,7,9,8 1,2,5,6)(1,2,3,4,9,8 5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,3,4,5,9 6,7,8)(3,4,7,9,8 1,2,5,6)(1,2,3,9,8 4,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,3,5,9 4,6,7,8)(7,9,8 1,2,3,4,5,6)(1,2,9,8 3,4,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,3,4,5,9 6,7,8)(4,7,9,8 1,2,3,5,6)(1,2,9,8 3,4,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,5,9 3,4,6,7,8)(1,2,3,7,9,8 4,5,6)(9,8 1,2,3,4,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,5,9 3,4,6,7,8)(1,2,3,4,7,9,8 5,6)(4,9,8 1,2,3,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,3,5,9 4,6,7,8)(1,2,7,9,8 3,4,5,6)(9,8 1,2,3,4,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,3,4,5,9 6,7,8)(1,2,7,9,8 3,4,5,6)(4,9,8 1,2,3,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,3,5,9 4,6,7,8)(1,2,3,4,7,9,8 5,6)(3,4,9,8 1,2,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,3,4,5,9 6,7,8)(1,2,3,7,9,8 4,5,6)(3,4,9,8 1,2,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,4,5,9 2,3,6,7,8)(1,2,7,9,8 3,4,5,6)(1,2,3,4,9,8 5,6,7)(1,2,3,4 5,6,7,9,8)
(1,5,9 2,3,4,6,7,8)(1,2,7,9,8 3,4,5,6)(1,2,3,9,8 4,5,6,7)(1,2,3,4 5,6,7,9,8)
(1,5,9 2,3,4,6,7,8)(1,2,7,9,8 3,4,5,6)(1,2,3,4,9,8 5,6,7)(1,2,3 4,5,6,7,9,8)
(1,4,5,9 2,3,6,7,8)(1,2,3,4,7,9,8 5,6)(1,2,9,8 3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(1,5,9 2,3,4,6,7,8)(1,2,3,7,9,8 4,5,6)(1,2,9,8 3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(1,2,5,9 3,4,6,7,8)(1,4,7,9,8 2,3,5,6)(1,2,3,4,9,8 5,6,7)(1,2,3,4 5,6,7,9,8)
(1,2,5,9 3,4,6,7,8)(1,7,9,8 2,3,4,5,6)(1,2,3,9,8 4,5,6,7)(1,2,3,4 5,6,7,9,8)
(1,2,5,9 3,4,6,7,8)(1,7,9,8 2,3,4,5,6)(1,2,3,4,9,8 5,6,7)(1,2,3 4,5,6,7,9,8)
(1,2,3,5,9 4,6,7,8)(1,7,9,8 2,3,4,5,6)(1,2,9,8 3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(1,2,3,4,5,9 6,7,8)(1,4,7,9,8 2,3,5,6)(1,2,9,8 3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(1,2,3,5,9 4,6,7,8)(1,7,9,8 2,3,4,5,6)(1,2,3,4,9,8 5,6,7)(1,2 3,4,5,6,7,9,8)

(1,2,5,9 3,4,6,7,8)(1,2,3,7,9,8 4,5,6)(1,9,8 2,3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(1,2,5,9 3,4,6,7,8)(1,2,3,4,7,9,8 5,6)(1,4,9,8 2,3,5,6,7)(1,2,3,4 5,6,7,9,8)
(1,2,3,5,9 4,6,7,8)(1,2,7,9,8 3,4,5,6)(1,9,8 2,3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(1,2,3,4,5,9 6,7,8)(1,2,7,9,8 3,4,5,6)(1,4,9,8 2,3,5,6,7)(1,2,3,4 5,6,7,9,8)
(1,2,3,4,5 6,7,9,8)(2,3,4,7,8 1,5,6,9)(1,2,9,8 3,4,5,6,7)(4 1,2,3,5,6,7,9,8)
(1,2,3,5 4,6,7,9,8)(2,3,4,7,8 1,5,6,9)(1,2,3,4,9,8 5,6,7)(3,4 1,2,5,6,7,9,8)
(1,2,3,4,5 6,7,9,8)(2,3,7,8 1,4,5,6,9)(1,2,3,4,9,8 5,6,7)(3,4 1,2,5,6,7,9,8)
(1,2,3,4,5 6,7,9,8)(2,3,4,7,8 1,5,6,9)(1,2,3,9,8 4,5,6,7)(3,4 1,2,5,6,7,9,8)
(1,2,5 3,4,6,7,9,8)(4,7,8 1,2,3,5,6,9)(1,2,3,4,9,8 5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,5 3,4,6,7,9,8)(7,8 1,2,3,4,5,6,9)(1,2,3,9,8 4,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,3,5 4,6,7,9,8)(3,4,7,8 1,2,5,6,9)(1,2,3,4,9,8 5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,3,4,5 6,7,9,8)(3,4,7,8 1,2,5,6,9)(1,2,3,9,8 4,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,3,4,5 6,7,9,8)(3,4,7,8 1,2,5,6,9)(1,2,3,4,9,8 5,6,7)(2,3 1,4,5,6,7,9,8)
(1,2,3,5 4,6,7,9,8)(7,8 1,2,3,4,5,6,9)(1,2,9,8 3,4,5,6,7)(2,3,4 1,5,6,7,9,8)
(1,2,3,4,5 6,7,9,8)(4,7,8 1,2,3,5,6,9)(1,2,9,8 3,4,5,6,7)(2,3,4 1,5,6,7,9,8)
(2,3,5 1,4,6,7,9,8)(3,4,7,8 1,2,5,6,9)(1,2,3,4,9,8 5,6,7)(1,2,3,4 5,6,7,9,8)
(2,3,4,5 1,6,7,9,8)(3,4,7,8 1,2,5,6,9)(1,2,3,9,8 4,5,6,7)(1,2,3,4 5,6,7,9,8)
(2,3,4,5 1,6,7,9,8)(3,4,7,8 1,2,5,6,9)(1,2,3,4,9,8 5,6,7)(1,2,3 4,5,6,7,9,8)
(2,3,5 1,4,6,7,9,8)(7,8 1,2,3,4,5,6,9)(1,2,9,8 3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(2,3,4,5 1,6,7,9,8)(4,7,8 1,2,3,5,6,9)(1,2,9,8 3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(2,3,4,5 1,6,7,9,8)(4,7,8 1,2,3,5,6,9)(1,2,3,4,9,8 5,6,7)(1,2 3,4,5,6,7,9,8)
(3,4,5 1,2,6,7,9,8)(2,3,7,8 1,4,5,6,9)(1,2,3,4,9,8 5,6,7)(1,2,3,4 5,6,7,9,8)
(3,4,5 1,2,6,7,9,8)(2,3,4,7,8 1,5,6,9)(1,2,3,9,8 4,5,6,7)(1,2,3,4 5,6,7,9,8)
(3,4,5 1,2,6,7,9,8)(2,3,4,7,8 1,5,6,9)(1,2,3,4,9,8 5,6,7)(1,2,3 4,5,6,7,9,8)
(4,5 1,2,3,6,7,9,8)(2,3,4,7,8 1,5,6,9)(1,2,9,8 3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(5 1,2,3,4,6,7,9,8)(2,3,7,8 1,4,5,6,9)(1,2,9,8 3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(3,4,5 1,2,6,7,9,8)(4,7,8 1,2,3,5,6,9)(1,9,8 2,3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(3,4,5 1,2,6,7,9,8)(7,8 1,2,3,4,5,6,9)(1,4,9,8 2,3,5,6,7)(1,2,3,4 5,6,7,9,8)
(4,5 1,2,3,6,7,9,8)(3,4,7,8 1,2,5,6,9)(1,9,8 2,3,4,5,6,7)(1,2,3,4 5,6,7,9,8)
(5 1,2,3,4,6,7,9,8)(3,4,7,8 1,2,5,6,9)(1,4,9,8 2,3,5,6,7)(1,2,3,4 5,6,7,9,8)
(1,2,5 3,4,6,7,9,8)(1,7,9,8 2,3,4,5,6)(1,2,3,9,8 4,5,6,7)(1,2,3,4,9 5,6,7,8)
(1,2,5 3,4,6,7,9,8)(1,7,9,8 2,3,4,5,6)(1,2,3,4,9,8 5,6,7)(1,2,3,9 4,5,6,7,8)
(1,2,3,5 4,6,7,9,8)(1,7,9,8 2,3,4,5,6)(1,2,9,8 3,4,5,6,7)(1,2,3,4,9 5,6,7,8)
(1,2,3,4,5 6,7,9,8)(1,4,7,9,8 2,3,5,6)(1,2,9,8 3,4,5,6,7)(1,2,3,4,9 5,6,7,8)
(1,2,3,5 4,6,7,9,8)(1,7,9,8 2,3,4,5,6)(1,2,9,8 3,4,5,6,7)(1,2,9 3,4,5,6,7,8)
(1,2,3,4,5 6,7,9,8)(1,4,7,9,8 2,3,5,6)(1,2,3,4,9,8 5,6,7)(1,2,9 3,4,5,6,7,8)
(1,2,5 3,4,6,7,9,8)(1,2,3,7,9,8 4,5,6)(1,9,8 2,3,4,5,6,7)(1,2,3,4,9 5,6,7,8)
(1,2,3,5 4,6,7,9,8)(1,2,7,9,8 3,4,5,6)(1,9,8 2,3,4,5,6,7)(1,2,3,4,9 5,6,7,8)
(1,2,3,4,5 6,7,9,8)(1,2,7,9,8 3,4,5,6)(1,4,9,8 2,3,5,6,7)(1,2,3,4,9 5,6,7,8)
(1,2,3,4,5 6,7,9,8)(1,2,3,4,7,9,8 5,6)(1,4,9,8 2,3,5,6,7)(1,2,9 3,4,5,6,7,8)
(1,2,3,5 4,6,7,9,8)(1,2,7,9,8 3,4,5,6)(1,2,3,4,9,8 5,6,7)(1,9 2,3,4,5,6,7,8)
(1,2,3,4,5 6,7,9,8)(1,2,7,9,8 3,4,5,6)(1,2,3,4,9,8 5,6,7)(1,4,9 2,3,5,6,7,8)
(1,2,3,4,5 6,7,9,8)(1,2,3,4,7,9,8 5,6)(1,2,9,8 3,4,5,6,7)(1,4,9 2,3,5,6,7,8)
(2,3,4,5 1,6,7,9,8)(3,4,7,9,8 1,2,5,6)(1,2,3,9,8 4,5,6,7)(1,2,3,4,9 5,6,7,8)
(2,3,4,5 1,6,7,9,8)(3,4,7,9,8 1,2,5,6)(1,2,3,4,9,8 5,6,7)(1,2,3,9 4,5,6,7,8)
(2,3,5 1,4,6,7,9,8)(7,9,8 1,2,3,4,5,6)(1,2,9,8 3,4,5,6,7)(1,2,3,4,9 5,6,7,8)
(2,3,4,5 1,6,7,9,8)(4,7,9,8 1,2,3,5,6)(1,2,9,8 3,4,5,6,7)(1,2,3,4,9 5,6,7,8)
(3,4,5 1,2,6,7,9,8)(4,7,9,8 1,2,3,5,6)(1,9,8 2,3,4,5,6,7)(1,2,3,4,9 5,6,7,8)
(3,4,5 1,2,6,7,9,8)(7,9,8 1,2,3,4,5,6)(1,4,9,8 2,3,5,6,7)(1,2,3,4,9 5,6,7,8)
(4,5 1,2,3,6,7,9,8)(3,4,7,9,8 1,2,5,6)(1,9,8 2,3,4,5,6,7)(1,2,3,4,9 5,6,7,8)
(3,4,5 1,2,6,7,9,8)(4,7,9,8 1,2,3,5,6)(1,2,3,4,9,8 5,6,7)(1,9 2,3,4,5,6,7,8)
(3,4,5 1,2,6,7,9,8)(7,9,8 1,2,3,4,5,6)(1,2,3,4,9,8 5,6,7)(1,4,9 2,3,5,6,7,8)
(2,3,5 1,4,6,7,9,8)(1,2,7,9,8 3,4,5,6)(9,8 1,2,3,4,5,6,7)(1,2,3,4,9 5,6,7,8)
(2,3,4,5 1,6,7,9,8)(1,2,7,9,8 3,4,5,6)(4,9,8 1,2,3,5,6,7)(1,2,3,4,9 5,6,7,8)
(2,3,4,5 1,6,7,9,8)(1,2,3,7,9,8 4,5,6)(3,4,9,8 1,2,5,6,7)(1,2,3,4,9 5,6,7,8)
(2,3,4,5 1,6,7,9,8)(1,2,3,4,7,9,8 5,6)(3,4,9,8 1,2,5,6,7)(1,2,3,9 4,5,6,7,8)
(2,3,4,5 1,6,7,9,8)(1,2,3,4,7,9,8 5,6)(4,9,8 1,2,3,5,6,7)(1,2,9 3,4,5,6,7,8)
(3,4,5 1,2,6,7,9,8)(1,4,7,9,8 2,3,5,6)(9,8 1,2,3,4,5,6,7)(1,2,3,4,9 5,6,7,8)
(3,4,5 1,2,6,7,9,8)(1,7,9,8 2,3,4,5,6)(4,9,8 1,2,3,5,6,7)(1,2,3,4,9 5,6,7,8)
(4,5 1,2,3,6,7,9,8)(1,7,9,8 2,3,4,5,6)(3,4,9,8 1,2,5,6,7)(1,2,3,4,9 5,6,7,8)
(5 1,2,3,4,6,7,9,8)(1,7,9,8 2,3,4,5,6)(3,4,9,8 1,2,5,6,7)(1,2,3,9 4,5,6,7,8)

7.2.18 By taking into account the calculated secondary adjacency conditions and by changing the blocks within the partitions and the places of the partitions we find an appropriate construction for the FFP for which the counting states can be assigned with codes which correspond to the Gray code sequence.

One FFP which can be used for state assignment is:

(3,4,5,8,9 1,2,6,7)(1,7,8,9 2,3,4,5,6)

(1,2,3,4,9 5,6,7,8)(4 1,2,3,5,6,7,8,9)

We change the places of the partitions in the following manner:

(4 1,2,3,5,6,7,8,9)(1,2,3,4,9 5,6,7,8)

(1,7,8,9 2,3,4,5,6)(3,4,5,8,9 1,2,6,7)

For the assignment of blocks with :

4	- 1
1,2,3,5,6,7,8,9	- 0
1,2,3,4,9	- 1
5,6,7,8	- 0
1,7,8,9	- 1
2,3,4,5,6	- 0
3,4,5,8,9	- 0
1,2,6,7	- 1

The following assignment of states is reached:

5 - 0000
6 - 0001
7 - 0011
8 - 0010
9 - 0110
1 - 0111
2 - 0101
3 - 0100
4 - 1100

The Gray code sequence between the counting states is fulfilled.

The assigned next state table is as follows:

S I	00	01	11	10
1 - 0111	----	----	----	----
2 - 0101	----	1100	0000	----
3 - 0100	0100	0100	----	----
4 - 1100	0111	1100	0000	0111
5 - 0000	----	----	0000	----
6 - 0001	----	----	0110	0110
7 - 0011	0111	----	----	0111
8 - 0010	----	----	----	----
9 - 0110	----	----	0110	0110
				S_1 S_2 S_3 S_4

In the assigned next state table we add an additional variable L which has the value of 1 for the transitions which cannot be replaced by counting transitions, and 0 otherwise.

S I	00	01	11	10
1 - 0111	----0	----0	----0	----0
2 - 0101	----0	11001	00001	----0
3 - 0100	01001	01001	----0	----0
4 - 1100	01111	11001	00001	01111
5 - 0000	----0	----0	00001	----0
6 - 0001	----0	----0	01101	01101
7 - 0011	01111	----0	----0	01111
8 - 0010	----0	----0	----0	----0
9 - 0110	----0	----0	01101	01101
			S_1 S_2 S_3 S_4 L	

By using a counter-based PLA structure S_1, S_2, S_3, S_4 and L should be implemented as shown in example I.

8.Conclusions

In the report the decompositional state assignment with reuse of standard designs has been discussed. Since the state assignment methods that consider a sequential machine on the whole are all heuristic, they produce often good solutions but they do not guarantee the strict optimality for them and they can fail sometimes. Most of them work better for small than for large machines. For this reason, decompositional implementations of sequential machines with reuse of standard carefully optimized sub-machines can sometimes be superior. If the specification of a given sequential machine (or its part) is strongly similar to the specification of a given standard machine then there is a great chance to reach a better solution by decompositional implementation.

In the second part of the report, we focused our attention on counters as standard machines. We made this choice, because many practical controllers can be designed quite well as modified counters.

Constructing a modified version of the method of maximal adjacencies [1], we answered the question: how to find the (sub-) optimal sequential decomposition of a given sequential machine into a number of sub-machines defining counters and a small general sequential sub-machine.

The precise algorithm for computing the (sub-) optimal state chains and the (sub-) optimal state codes is described in the report and illustrated with examples.

We are now developing software that will implement this algorithms. In the appendix, the first results are provided which were obtained using the first part of the software in order to process the machine from the examples.

The work described in this report should be continued in the following directions:

- finishing the implementation of the software and checking the practical usefulness of the decompositional state assignment with reuse of standard designs;

- considering other types of standard sub-machines than counters;

- considering the simultaneous decompositions with reuse of standard designs.

LITERATURE

[1] Jozwiak, L.
Minimal realization of sequential machines:
The method of maximal adjacencies.
EUT Report 88-E-209, Eindhoven University of Technology,
The Netherlands, 1988.

[2] Amann, R. and U.G.Baitinger
Optimal state chains and state codes in finite state machines.
IEEE Trans.Comp.-Aided Des.,Vol.8,No 2,February 1989,p.153-170.

[3] De Micheli, G. and R.K.Brayton, A.Sangiovanni-Vincentelli
Optimal state assignment for finite state machines.
IEEE Trans.Comp.-Aided Des.,Vol.CAD-4,No 3,July 1985,p.269-285.

[4] Jozwiak, L.
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.

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

[6] Jozwiak, L.
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.

[7] Jozwiak, L.
The bit full-decomposition of sequential machines.
EUT Report 89-E-223, Eindhoven University of Technology, The Netherlands, 1989.

[8] Jozwiak, L. and F.Vankan
Bit full-decomposition of sequential machines. Algorithms and results.
Canadian conference on electrical and computer engineering, Montreal, Canada, September 17-20, 1989.

[9] Hartmanis, J.
On the state assignment problems for sequential machines I.
IRE Trans.Electron.Comput., Vol.EC-10(1961),p.157-165.

[10] Stearns, R.E. and J.Hartmanis
On the state assignment problems for sequential machines II.
IRE Trans.Electron.Comput., Vol.EC-10(1961),p.593-603.

[11] Armstrong, D.B.
On the efficient assignment of internal codes to sequential machines.
IRE Trans.Electron.Comput., Vol.EC-11(1962),p.611-622.

[12] Armstrong, D.B.
A programmed algorithm for assigning internal codes to sequential machines.
IRE Trans.Electron.Comput., Vol.EC-11(1962),p.466-472.

APPENDIX

Enter FSM definition file name : example1

StateRange : 4

InputBitRange : 2

ProductTerms : 10

STATENAMETABLE (It gives the relationship between the internal and the external names of the machine. Further the internal names are used.)

1	1
2	2
3	3
4	4

NEXTSTATETABLE

presentstate	nextstate	times	inputvector
1	3	1	12
1	1	1	01
1	2	1	00

presentstate	nextstate	times	inputvector
2	3	1	21
2	4	1	10
2	2	1	00

presentstate	nextstate	times	inputvector
3	4	1	10
3	1	2	02
			21

presentstate	nextstate	times	inputvector
4	2	1	22

PREVSTATETABLE

presentstate	prevstate	times	inputvector
1	3	2	02
			21

1	1	1	01

presentstate	prevstate	times	inputvector
2	4	1	22
2	2	1	00
2	1	1	00

presentstate	prevstate	times	inputvector
3	2	1	21
3	1	1	12

presentstate	prevstate	times	inputvector
4	3	1	10
4	2	1	10

INTERCONNECTIONTABLE

```
state  1-> 3
subspace(s): 12
state  1<- 3
subspace(s): 02
             21

state  2-> 4
subspace(s): 10
state  2<- 4
subspace(s): 22
```

CYCLICTRANTABLE

```
state:  1
subspace(s): 01

state:  2
subspace(s): 00
```

COSTTABLE

```
presentstate =  1 nextstate =  2

MaxPLA      =  2.0E+00
MinPLA      =  0.0E+00
Adj.States  :  1 2
MaxCount    =  4.0E+00
MinCount    =  4.0E+00
```

```
Adj.States :

presentstate =  1 nextstate =  3

MaxPLA      =  2.0E+00
MinPLA      =  2.0E+00
Adj.States :
MaxCount    =  8.0E+00
MinCount    =  4.7E+00
Adj.States :  1 2
              1 4


presentstate =  2 nextstate =  4

MaxPLA      =  2.0E+00
MinPLA      =  1.0E+00
Adj.States :  2 4
MaxCount    =  6.0E+00
MinCount    =  5.3E+00
Adj.States :  4 1


presentstate =  2 nextstate =  3

MaxPLA      =  2.0E+00
MinPLA      =  2.0E+00
Adj.States :
MaxCount    =  4.0E+00
MinCount    =  3.0E+00
Adj.States :  2 4


presentstate =  3 nextstate =  1

MaxPLA      =  4.0E+00
MinPLA      =  2.7E+00
Adj.States :  1 4
MaxCount    =  4.0E+00
MinCount    =  3.0E+00
Adj.States :  3 2


presentstate =  3 nextstate =  4

MaxPLA      =  2.0E+00
MinPLA      =  1.0E+00
Adj.States :  3 2
MaxCount    =  6.0E+00
MinCount    =  5.0E+00
Adj.States :  2 4


presentstate =  4 nextstate =  2

MaxPLA      =  2.0E+00
MinPLA      =  2.0E+00
Adj.States :
MaxCount    =  4.0E+00
MinCount    =  1.0E+00
Adj.States :  1 2
              2 4
```

```
Enter the weight for MaxPLA    :  1.2E+00
Enter the weight for MinPLA    :  1.0E+00
Enter the weight for MaxCount  :  1.2E+00
Enter the weight for MinCount  :  1.0E+00
Enter the weight for AdjNr     :  8.0E-01

These are the ordered transistions according to ELECTRE I  :

 presentstate :   3  nextstate :  1        1.4E+01
 presentstate :   2  nextstate :  3        1.2E+01
 presentstate :   4  nextstate :  2        1.0E+01
 presentstate :   1  nextstate :  2        8.0E+00
 presentstate :   3  nextstate :  4        6.0E+00
 presentstate :   1  nextstate :  3        4.0E+00
 presentstate :   2  nextstate :  4        2.0E+00


These are the ordered transistions according to ELECTRE II :

 presentstate :   3  nextstate :  1        2.0E+01
 presentstate :   2  nextstate :  3        1.7E+01
 presentstate :   4  nextstate :  2        1.7E+01
 presentstate :   1  nextstate :  2        1.2E+01
 presentstate :   1  nextstate :  3        7.0E+00
 presentstate :   3  nextstate :  4        7.0E+00
 presentstate :   2  nextstate :  4        4.0E+00



VALUES CALCULATED BY ELECTRE-II
Enter the max number of beams :    3
Enter the min qualityfactor   : 0.90


The contents of the transitiontable :

Preset State        Next State          Value
-------------------------------------------

      3                  1                20.
      4                  2                17.
      2                  3                17.
      1                  2                12.
      3                  4                 7.
      1                  3                 7.
      2                  4                 4.

The contents of the Chaintable :

Present State        Next State
- ----------------------------

Chain          1

         4                  2
```

```
          2                   3
          3                   1
```

```
Adjacency : 4|2
Adjacency : 2|3
Adjacency : 3|1
```

The created FFP's are :

(1,4 2,3)(1,3 2,4)

****** ASSIGNMENTFOUND ******

The assignments are :

State 1 --> 10
State 2 --> 01
State 3 --> 11
State 4 --> 00

execution time : 0 h 1 m 22 s

END OF THE PROGRAM

Enter FSM definition file name : example2

StateRange : 9

InputBitRange : 2

ProductTerms : 19

STATENAMETABLE (It gives the relationship between the internal and the external names of the machine. Further the internal names are used.)

5	5
6	6
7	7
8	9
9	8
1	1
2	2
3	3
4	4

NEXTSTATETABLE

presentstate	nextstate	times	inputvector
1	2	1	22

presentstate	nextstate	times	inputvector
2	5	1	11
2	4	1	01
2	3	1	20

presentstate	nextstate	times	inputvector
3	4	1	12
3	3	1	02

presentstate	nextstate	times	inputvector
4	5	1	11
4	4	1	01
4	1	1	20

presentstate	nextstate	times	inputvector
5	5	1	11
5	6	2	02
			20

presentstate	nextstate	times	inputvector
6	8	1	12
6	7	1	02

presentstate	nextstate	times	inputvector
7	9	1	21
7	1	1	20

presentstate	nextstate	times	inputvector
8	8	1	12
8	1	1	02

presentstate	nextstate	times	inputvector
9	8	1	22

PREVSTATETABLE

presentstate	prevstate	times	inputvector
1	8	1	02
1	7	1	20
1	4	1	20

presentstate	prevstate	times	inputvector
2	1	1	22

presentstate	prevstate	times	inputvector
3	3	1	02
3	2	1	20

presentstate	prevstate	times	inputvector

4	4	1	01
4	3	1	12
4	2	1	01

presentstate	prevstate	times	inputvector
5	5	1	11
5	4	1	11
5	3	1	11

presentstate	prevstate	times	inputvector
6	5	2	02 20

presentstate	prevstate	times	inputvector
7	6	1	02

presentstate	prevstate	times	inputvector
8	8	1	12
8	9	1	22
8	6	1	12

presentstate	prevstate	times	inputvector
9	7	1	21

INTERCONNECTIONTABLE

CYCLICTRANTABLE

state: 3
subspace(s): 02

state: 4
subspace(s): 01

state: 5
subspace(s): 11

state: 8
subspace(s): 12

COSTTABLE

presentstate = 1 nextstate = 2

MaxPLA = 4.0E+00
MinPLA = 4.0E+00
Adj.States :
MaxCount = 0.0E+00
MinCount = 0.0E+00
Adj.States :

presentstate = 2 nextstate = 3

MaxPLA = 4.0E+00
MinPLA = 4.0E+00
Adj.States :
MaxCount = 8.0E+00
MinCount = 0.0E+00
Adj.States : 2 4
2 5

presentstate = 2 nextstate = 4

MaxPLA = 4.0E+00
MinPLA = 0.0E+00
Adj.States : 2 4
MaxCount = 1.2E+01
MinCount = 5.0E+00
Adj.States : 2 5
3 4

presentstate = 2 nextstate = 5

MaxPLA = 4.0E+00
MinPLA = 0.0E+00
Adj.States : 2 5
MaxCount = 1.2E+01
MinCount = 4.0E+00
Adj.States : 2 4
4 5

presentstate = 3 nextstate = 4

MaxPLA = 4.0E+00
MinPLA = 1.0E+00
Adj.States : 3 4
MaxCount = 4.0E+00
MinCount = 0.0E+00
Adj.States : 2 4

presentstate = 4 nextstate = 1

MaxPLA = 4.0E+00
MinPLA = 2.0E+00
Adj.States : 4 7
MaxCount = 1.2E+01
MinCount = 3.5E+00
Adj.States : 4 5
8 1
1 9

```
presentstate =  4 nextstate =  5

MaxPLA      =  4.0E+00
MinPLA      =  0.0E+00
Adj.States  :  4 5
MaxCount    =  8.0E+00
MinCount    =  2.0E+00
Adj.States  :  4 7
               2 5


presentstate =  5 nextstate =  6

MaxPLA      =  8.0E+00
MinPLA      =  2.0E+00
Adj.States  :  5 6
MaxCount    =  0.0E+00
MinCount    =  0.0E+00
Adj.States  :

presentstate =  6 nextstate =  7

MaxPLA      =  4.0E+00
MinPLA      =  2.5E+00
Adj.States  :  7 8
MaxCount    =  4.0E+00
MinCount    =  0.0E+00
Adj.States  :  6 8


presentstate =  6 nextstate =  8

MaxPLA      =  4.0E+00
MinPLA      =  0.0E+00
Adj.States  :  6 8
MaxCount    =  8.0E+00
MinCount    =  8.0E+00
Adj.States  :

presentstate =  7 nextstate =  1

MaxPLA      =  4.0E+00
MinPLA      =  2.0E+00
Adj.States  :  7 4
MaxCount    =  1.2E+01
MinCount    =  9.0E+00
Adj.States  :  8 1


presentstate =  7 nextstate =  9

MaxPLA      =  4.0E+00
MinPLA      =  2.5E+00
Adj.States  :  9 1
MaxCount    =  4.0E+00
MinCount    =  2.0E+00
Adj.States  :  7 4


presentstate =  8 nextstate =  1

MaxPLA      =  4.0E+00
MinPLA      =  1.0E+00
```

```
 Adj.States :  8 1
 MaxCount   =  8.0E+00
 MinCount   =  4.0E+00
 Adj.States :  7 4


 presentstate =  9 nextstate =  8

 MaxPLA     =  4.0E+00
 MinPLA     =  4.0E+00
 Adj.States :
 MaxCount   =  4.0E+00
 MinCount   =  0.0E+00
 Adj.States :  6 8




Enter the weight for MaxPLA    :  1.2E+00
Enter the weight for MinPLA    :  1.0E+00
Enter the weight for MaxCount  :  1.2E+00
Enter the weight for MinCount  :  1.0E+00
Enter the weight for AdjNr     :  8.0E-01

These are the ordered transistions according to ELECTRE I  :

 presentstate :   1  nextstate :  2       2.7E+01
 presentstate :   5  nextstate :  6       2.7E+01
 presentstate :   9  nextstate :  8       2.4E+01
 presentstate :   6  nextstate :  7       2.2E+01
 presentstate :   7  nextstate :  9       1.9E+01
 presentstate :   2  nextstate :  3       1.8E+01
 presentstate :   3  nextstate :  4       1.7E+01
 presentstate :   8  nextstate :  1       1.4E+01
 presentstate :   4  nextstate :  5       1.1E+01
 presentstate :   6  nextstate :  8       1.1E+01
 presentstate :   4  nextstate :  1       7.0E+00
 presentstate :   7  nextstate :  1       7.0E+00
 presentstate :   2  nextstate :  5       4.0E+00
 presentstate :   2  nextstate :  4       2.0E+00


These are the ordered transistions according to ELECTRE II :

 presentstate :   5  nextstate :  6       4.1E+01
 presentstate :   1  nextstate :  2       3.3E+01
 presentstate :   6  nextstate :  7       3.3E+01
 presentstate :   9  nextstate :  8       3.2E+01
 presentstate :   3  nextstate :  4       3.0E+01
 presentstate :   2  nextstate :  3       2.7E+01
 presentstate :   7  nextstate :  9       2.6E+01
 presentstate :   8  nextstate :  1       2.6E+01
 presentstate :   4  nextstate :  5       2.1E+01
 presentstate :   4  nextstate :  1       1.2E+01
 presentstate :   6  nextstate :  8       1.2E+01
 presentstate :   7  nextstate :  1       9.0E+00
 presentstate :   2  nextstate :  5       7.0E+00
 presentstate :   2  nextstate :  4       6.0E+00
```

VALUES CALCULATED BY ELECTRE-II
Enter the max number of beams : 3
Enter the min qualityfactor : 0.90

The contents of the transitiontable :

Preset State	Next State	Value
5	6	41.
6	7	33.
1	2	33.
9	8	32.
3	4	30.
2	3	27.
8	1	26.
7	9	26.
4	5	21.
6	8	12.
4	1	12.
7	1	9.
2	5	7.
2	4	6.

The contents of the Chaintable :

Present State	Next State
Chain 1	
5	6
6	7
7	9
9	8
8	1
1	2
2	3
3	4

Adjacency : 5|6
Adjacency : 6|7
Adjacency : 7|9
Adjacency : 9|8
Adjacency : 8|1
Adjacency : 1|2
Adjacency : 2|3
Adjacency : 3|4

The created EFP's are :

(3,4,5,8,9 1,2,6,7)(1,7,8,9 2,3,4,5,6)(1,2,3,4,8 5,6,7,9)(4 1,2,3,5,6,7,8,9)

(4,5,8,9 1,2,3,6,7)(1,7,8,9 2,3,4,5,6)(1,2,3,4,8 5,6,7,9)(3,4 1,2,5,6,7,8,9)
(5,8,9 1,2,3,4,6,7)(1,4,7,8,9 2,3,5,6)(1,2,3,4,8 5,6,7,9)(3,4 1,2,5,6,7,8,9)
(5,8,9 1,2,3,4,6,7)(1,7,8,9 2,3,4,5,6)(1,2,3,8 4,5,6,7,9)(3,4 1,2,5,6,7,8,9)
(3,4,5,8,9 1,2,6,7)(1,2,3,7,8,9 4,5,6)(1,2,3,4,8 5,6,7,9)(2,3,4 1,5,6,7,8,9)
(3,4,5,8,9 1,2,6,7)(1,2,3,4,7,8,9 5,6)(1,2,3,8 4,5,6,7,9)(2,3,4 1,5,6,7,8,9)
(4,5,8,9 1,2,3,6,7)(1,2,7,8,9 3,4,5,6)(1,2,3,4,8 5,6,7,9)(2,3,4 1,5,6,7,8,9)
(5,8,9 1,2,3,4,6,7)(1,2,7,8,9 3,4,5,6)(1,2,3,8 4,5,6,7,9)(2,3,4 1,5,6,7,8,9)
(5,8,9 1,2,3,4,6,7)(1,2,7,8,9 3,4,5,6)(1,2,3,4,8 5,6,7,9)(2,3 1,4,5,6,7,8,9)
(4,5,8,9 1,2,3,6,7)(1,2,3,4,7,8,9 5,6)(1,2,8 3,4,5,6,7,9)(2,3,4 1,5,6,7,8,9)
(5,8,9 1,2,3,4,6,7)(1,2,3,7,8,9 4,5,6)(1,2,8 3,4,5,6,7,9)(2,3,4 1,5,6,7,8,9)
(1,5,8,9 2,3,4,6,7)(3,4,7,8,9 1,2,5,6)(1,2,3,4,8 5,6,7,9)(4 1,2,3,5,6,7,8,9)
(1,4,5,8,9 2,3,6,7)(7,8,9 1,2,3,4,5,6)(1,2,3,4,8 5,6,7,9)(3,4 1,2,5,6,7,8,9)
(1,5,8,9 2,3,4,6,7)(4,7,8,9 1,2,3,5,6)(1,2,3,4,8 5,6,7,9)(3,4 1,2,5,6,7,8,9)
(1,5,8,9 2,3,4,6,7)(7,8,9 1,2,3,4,5,6)(1,2,3,8 4,5,6,7,9)(3,4 1,2,5,6,7,8,9)
(1,2,5,8,9 3,4,6,7)(4,7,8,9 1,2,3,5,6)(1,2,3,4,8 5,6,7,9)(2,3,4 1,5,6,7,8,9)
(1,2,5,8,9 3,4,6,7)(7,8,9 1,2,3,4,5,6)(1,2,3,8 4,5,6,7,9)(2,3,4 1,5,6,7,8,9)
(1,2,5,8,9 3,4,6,7)(7,8,9 1,2,3,4,5,6)(1,2,3,4,8 5,6,7,9)(2,3 1,4,5,6,7,8,9)
(1,2,3,5,8,9 4,6,7)(3,4,7,8,9 1,2,5,6)(1,2,3,4,8 5,6,7,9)(2,3,4 1,5,6,7,8,9)
(1,2,3,4,5,8,9 6,7)(3,4,7,8,9 1,2,5,6)(1,2,3,8 4,5,6,7,9)(2,3,4 1,5,6,7,8,9)
(1,2,3,5,8,9 4,6,7)(7,8,9 1,2,3,4,5,6)(1,2,8 3,4,5,6,7,9)(2,3,4 1,5,6,7,8,9)
(1,2,3,4,5,8,9 6,7)(4,7,8,9 1,2,3,5,6)(1,2,8 3,4,5,6,7,9)(2,3,4 1,5,6,7,8,9)
(1,4,5,8,9 2,3,6,7)(1,2,7,8,9 3,4,5,6)(1,2,3,4,8 5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,5,8,9 2,3,4,6,7)(1,2,7,8,9 3,4,5,6)(1,2,3,8 4,5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,5,8,9 2,3,4,6,7)(1,2,7,8,9 3,4,5,6)(1,2,3,4,8 5,6,7,9)(1,2,3 4,5,6,7,8,9)
(1,4,5,8,9 2,3,6,7)(1,2,3,4,7,8,9 5,6)(1,2,8 3,4,5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,5,8,9 2,3,4,6,7)(1,2,3,7,8,9 4,5,6)(1,2,8 3,4,5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,5,8,9 2,3,4,6,7)(1,2,3,7,8,9 4,5,6)(1,2,3,4,8 5,6,7,9)(1,2 3,4,5,6,7,8,9)
(1,2,5,8,9 3,4,6,7)(1,4,7,8,9 2,3,5,6)(1,2,3,4,8 5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,2,5,8,9 3,4,6,7)(1,7,8,9 2,3,4,5,6)(1,2,3,8 4,5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,2,5,8,9 3,4,6,7)(1,7,8,9 2,3,4,5,6)(1,2,3,4,8 5,6,7,9)(1,2,3 4,5,6,7,8,9)
(1,2,3,5,8,9 4,6,7)(1,7,8,9 2,3,4,5,6)(1,2,8 3,4,5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,2,3,4,5,8,9 6,7)(1,4,7,8,9 2,3,5,6)(1,2,8 3,4,5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,2,3,5,8,9 4,6,7)(1,7,8,9 2,3,4,5,6)(1,2,3,4,8 5,6,7,9)(1,2 3,4,5,6,7,8,9)
(1,2,5,8,9 3,4,6,7)(1,2,3,7,8,9 4,5,6)(1,8 2,3,4,5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,2,5,8,9 3,4,6,7)(1,2,3,4,7,8,9 5,6)(1,4,8 2,3,5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,2,3,5,8,9 4,6,7)(1,2,7,8,9 3,4,5,6)(1,8 2,3,4,5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,2,3,4,5,8,9 6,7)(1,2,7,8,9 3,4,5,6)(1,4,8 2,3,5,6,7,9)(1,2,3,4 5,6,7,8,9)
(1,4,5,8 2,3,6,7,9)(7,8,9 1,2,3,4,5,6)(1,2,3,4,8,9 5,6,7)(3,4 1,2,5,6,7,8,9)
(1,5,8 2,3,4,6,7,9)(4,7,8,9 1,2,3,5,6)(1,2,3,4,8,9 5,6,7)(3,4 1,2,5,6,7,8,9)
(1,5,8 2,3,4,6,7,9)(7,8,9 1,2,3,4,5,6)(1,2,3,8,9 4,5,6,7)(3,4 1,2,5,6,7,8,9)
(1,2,5,8 3,4,6,7,9)(4,7,8,9 1,2,3,5,6)(1,2,3,4,8,9 5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,5,8 3,4,6,7,9)(7,8,9 1,2,3,4,5,6)(1,2,3,8,9 4,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,5,8 3,4,6,7,9)(7,8,9 1,2,3,4,5,6)(1,2,3,4,8,9 5,6,7)(2,3 1,4,5,6,7,8,9)
(1,2,3,5,8 4,6,7,9)(3,4,7,8,9 1,2,5,6)(1,2,3,4,8,9 5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,3,4,5,8 6,7,9)(3,4,7,8,9 1,2,5,6)(1,2,3,8,9 4,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,3,5,8 4,6,7,9)(7,8,9 1,2,3,4,5,6)(1,2,8,9 3,4,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,3,4,5,8 6,7,9)(4,7,8,9 1,2,3,5,6)(1,2,8,9 3,4,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,5,8 3,4,6,7,9)(1,2,3,7,8,9 4,5,6)(8,9 1,2,3,4,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,5,8 3,4,6,7,9)(1,2,3,4,7,8,9 5,6)(4,8,9 1,2,3,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,3,5,8 4,6,7,9)(1,2,7,8,9 3,4,5,6)(8,9 1,2,3,4,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,3,4,5,8 6,7,9)(1,2,7,8,9 3,4,5,6)(4,8,9 1,2,3,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,3,5,8 4,6,7,9)(1,2,3,4,7,8,9 5,6)(3,4,8,9 1,2,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,3,4,5,8 6,7,9)(1,2,3,7,8,9 4,5,6)(3,4,8,9 1,2,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,4,5,8 2,3,6,7,9)(1,2,7,8,9 3,4,5,6)(1,2,3,4,8,9 5,6,7)(1,2,3,4 5,6,7,8,9)
(1,5,8 2,3,4,6,7,9)(1,2,7,8,9 3,4,5,6)(1,2,3,8,9 4,5,6,7)(1,2,3,4 5,6,7,8,9)
(1,5,8 2,3,4,6,7,9)(1,2,7,8,9 3,4,5,6)(1,2,3,4,8,9 5,6,7)(1,2,3 4,5,6,7,8,9)
(1,4,5,8 2,3,6,7,9)(1,2,3,4,7,8,9 5,6)(1,2,8,9 3,4,5,6,7)(1,2,3,4 5,6,7,8,9)
(1,5,8 2,3,4,6,7,9)(1,2,3,7,8,9 4,5,6)(1,2,8,9 3,4,5,6,7)(1,2,3,4 5,6,7,8,9)
(1,2,5,8 3,4,6,7,9)(1,4,7,8,9 2,3,5,6)(1,2,3,4,8,9 5,6,7)(1,2,3,4 5,6,7,8,9)
(1,2,5,8 3,4,6,7,9)(1,7,8,9 2,3,4,5,6)(1,2,3,8,9 4,5,6,7)(1,2,3,4 5,6,7,8,9)
(1,2,5,8 3,4,6,7,9)(1,7,8,9 2,3,4,5,6)(1,2,3,4,8,9 5,6,7)(1,2,3 4,5,6,7,8,9)
(1,2,3,5,8 4,6,7,9)(1,7,8,9 2,3,4,5,6)(1,2,8,9 3,4,5,6,7)(1,2,3,4 5,6,7,8,9)
(1,2,3,4,5,8 6,7,9)(1,4,7,8,9 2,3,5,6)(1,2,8,9 3,4,5,6,7)(1,2,3,4 5,6,7,8,9)
(1,2,3,5,8 4,6,7,9)(1,7,8,9 2,3,4,5,6)(1,2,3,4,8,9 5,6,7)(1,2 3,4,5,6,7,8,9)
(1,2,5,8 3,4,6,7,9)(1,2,3,7,8,9 4,5,6)(1,8,9 2,3,4,5,6,7)(1,2,3,4 5,6,7,8,9)

(1,2,5,8 3,4,6,7,9)(1,2,3,4,7,8,9 5,6)(1,4,8,9 2,3,5,6,7)(1,2,3,4 5,6,7,8,9)
(1,2,3,5,8 4,6,7,9)(1,2,7,8,9 3,4,5,6)(1,8,9 2,3,4,5,6,7)(1,2,3,4 5,6,7,8,9)
(1,2,3,4,5,8 6,7,9)(1,2,7,8,9 3,4,5,6)(1,4,8,9 2,3,5,6,7)(1,2,3,4 5,6,7,8,9)
(1,2,3,4,5 6,7,8,9)(2,3,4,7,9 1,5,6,8)(1,2,8,9 3,4,5,6,7)(4 1,2,3,5,6,7,8,9)
(1,2,3,5 4,6,7,8,9)(2,3,4,7,9 1,5,6,8)(1,2,3,4,8,9 5,6,7)(3,4 1,2,5,6,7,8,9)
(1,2,3,4,5 6,7,8,9)(2,3,7,9 1,4,5,6,8)(1,2,3,4,8,9 5,6,7)(3,4 1,2,5,6,7,8,9)
(1,2,3,4,5 6,7,8,9)(2,3,4,7,9 1,5,6,8)(1,2,3,8,9 4,5,6,7)(3,4 1,2,5,6,7,8,9)
(1,2,5 3,4,6,7,8,9)(4,7,9 1,2,3,5,6,8)(1,2,3,4,8,9 5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,5 3,4,6,7,8,9)(7,9 1,2,3,4,5,6,8)(1,2,3,8,9 4,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,3,5 4,6,7,8,9)(3,4,7,9 1,2,5,6,8)(1,2,3,4,8,9 5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,3,4,5 6,7,8,9)(3,4,7,9 1,2,5,6,8)(1,2,3,8,9 4,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,3,4,5 6,7,8,9)(3,4,7,9 1,2,5,6,8)(1,2,3,4,8,9 5,6,7)(2,3 1,4,5,6,7,8,9)
(1,2,3,5 4,6,7,8,9)(7,9 1,2,3,4,5,6,8)(1,2,8,9 3,4,5,6,7)(2,3,4 1,5,6,7,8,9)
(1,2,3,4,5 6,7,8,9)(4,7,9 1,2,3,5,6,8)(1,2,8,9 3,4,5,6,7)(2,3,4 1,5,6,7,8,9)
(2,3,5 1,4,6,7,8,9)(3,4,7,9 1,2,5,6,8)(1,2,3,4,8,9 5,6,7)(1,2,3,4 5,6,7,8,9)
(2,3,4,5 1,6,7,8,9)(3,4,7,9 1,2,5,6,8)(1,2,3,8,9 4,5,6,7)(1,2,3,4 5,6,7,8,9)
(2,3,4,5 1,6,7,8,9)(3,4,7,9 1,2,5,6,8)(1,2,3,4,8,9 5,6,7)(1,2,3 4,5,6,7,8,9)
(2,3,5 1,4,6,7,8,9)(7,9 1,2,3,4,5,6,8)(1,2,8,9 3,4,5,6,7)(1,2,3,4 5,6,7,8,9)
(2,3,4,5 1,6,7,8,9)(4,7,9 1,2,3,5,6,8)(1,2,8,9 3,4,5,6,7)(1,2,3,4 5,6,7,8,9)
(2,3,4,5 1,6,7,8,9)(4,7,9 1,2,3,5,6,8)(1,2,3,4,8,9 5,6,7)(1,2 3,4,5,6,7,8,9)
(3,4,5 1,2,6,7,8,9)(2,3,7,9 1,4,5,6,8)(1,2,3,4,8,9 5,6,7)(1,2,3,4 5,6,7,8,9)
(3,4,5 1,2,6,7,8,9)(2,3,4,7,9 1,5,6,8)(1,2,3,8,9 4,5,6,7)(1,2,3,4 5,6,7,8,9)
(3,4,5 1,2,6,7,8,9)(2,3,4,7,9 1,5,6,8)(1,2,3,4,8,9 5,6,7)(1,2,3 4,5,6,7,8,9)
(4,5 1,2,3,6,7,8,9)(2,3,4,7,9 1,5,6,8)(1,2,8,9 3,4,5,6,7)(1,2,3,4 5,6,7,8,9)
(5 1,2,3,4,6,7,8,9)(2,3,7,9 1,4,5,6,8)(1,2,8,9 3,4,5,6,7)(1,2,3,4 5,6,7,8,9)
(3,4,5 1,2,6,7,8,9)(4,7,9 1,2,3,5,6,8)(1,8,9 2,3,4,5,6,7)(1,2,3,4 5,6,7,8,9)
(3,4,5 1,2,6,7,8,9)(7,9 1,2,3,4,5,6,8)(1,4,8,9 2,3,5,6,7)(1,2,3,4 5,6,7,8,9)
(4,5 1,2,3,6,7,8,9)(3,4,7,9 1,2,5,6,8)(1,8,9 2,3,4,5,6,7)(1,2,3,4 5,6,7,8,9)
(5 1,2,3,4,6,7,8,9)(3,4,7,9 1,2,5,6,8)(1,4,8,9 2,3,5,6,7)(1,2,3,4 5,6,7,8,9)
(1,2,5 3,4,6,7,8,9)(1,7,8,9 2,3,4,5,6)(1,2,3,8,9 4,5,6,7)(1,2,3,4,8 5,6,7,9)
(1,2,5 3,4,6,7,8,9)(1,7,8,9 2,3,4,5,6)(1,2,3,4,8,9 5,6,7)(1,2,3,8 4,5,6,7,9)
(1,2,3,5 4,6,7,8,9)(1,7,8,9 2,3,4,5,6)(1,2,8,9 3,4,5,6,7)(1,2,3,4,8 5,6,7,9)
(1,2,3,4,5 6,7,8,9)(1,4,7,8,9 2,3,5,6)(1,2,8,9 3,4,5,6,7)(1,2,3,4,8 5,6,7,9)
(1,2,3,5 4,6,7,8,9)(1,7,8,9 2,3,4,5,6)(1,2,3,4,8,9 5,6,7)(1,2,8 3,4,5,6,7,9)
(1,2,3,4,5 6,7,8,9)(1,4,7,8,9 2,3,5,6)(1,2,3,4,8,9 5,6,7)(1,2,8 3,4,5,6,7,9)
(1,2,5 3,4,6,7,8,9)(1,2,3,7,8,9 4,5,6)(1,8,9 2,3,4,5,6,7)(1,2,3,4,8 5,6,7,9)
(1,2,3,5 4,6,7,8,9)(1,2,7,8,9 3,4,5,6)(1,8,9 2,3,4,5,6,7)(1,2,3,4,8 5,6,7,9)
(1,2,3,4,5 6,7,8,9)(1,2,7,8,9 3,4,5,6)(1,4,8,9 2,3,5,6,7)(1,2,3,4,8 5,6,7,9)
(1,2,3,4,5 6,7,8,9)(1,2,3,4,7,8,9 5,6)(1,4,8,9 2,3,5,6,7)(1,2,8 3,4,5,6,7,9)
(1,2,3,5 4,6,7,8,9)(1,2,7,8,9 3,4,5,6)(1,2,3,4,8,9 5,6,7)(1,8 2,3,4,5,6,7,9)
(1,2,3,4,5 6,7,8,9)(1,2,7,8,9 3,4,5,6)(1,2,3,4,8,9 5,6,7)(1,4,8 2,3,5,6,7,9)
(1,2,3,4,5 6,7,8,9)(1,2,3,4,7,8,9 5,6)(1,2,8,9 3,4,5,6,7)(1,4,8 2,3,5,6,7,9)
(2,3,4,5 1,6,7,8,9)(3,4,7,8,9 1,2,5,6)(1,2,3,8,9 4,5,6,7)(1,2,3,4,8 5,6,7,9)
(2,3,4,5 1,6,7,8,9)(3,4,7,8,9 1,2,5,6)(1,2,3,4,8,9 5,6,7)(1,2,3,8 4,5,6,7,9)
(2,3,5 1,4,6,7,8,9)(7,8,9 1,2,3,4,5,6)(1,2,8,9 3,4,5,6,7)(1,2,3,4,8 5,6,7,9)
(2,3,4,5 1,6,7,8,9)(4,7,8,9 1,2,3,5,6)(1,2,8,9 3,4,5,6,7)(1,2,3,4,8 5,6,7,9)
(2,3,5 1,4,6,7,8,9)(7,8,9 1,2,3,4,5,6)(1,2,3,4,8,9 5,6,7)(1,2,8 3,4,5,6,7,9)
(2,3,4,5 1,6,7,8,9)(4,7,8,9 1,2,3,5,6)(1,2,3,4,8,9 5,6,7)(1,2,8 3,4,5,6,7,9)
(3,4,5 1,2,6,7,8,9)(4,7,8,9 1,2,3,5,6)(1,8,9 2,3,4,5,6,7)(1,2,3,4,8 5,6,7,9)
(3,4,5 1,2,6,7,8,9)(7,8,9 1,2,3,4,5,6)(1,4,8,9 2,3,5,6,7)(1,2,3,4,8 5,6,7,9)
(4,5 1,2,3,6,7,8,9)(3,4,7,8,9 1,2,5,6)(1,8,9 2,3,4,5,6,7)(1,2,3,4,8 5,6,7,9)
(3,4,5 1,2,6,7,8,9)(4,7,8,9 1,2,3,5,6)(1,8,9 2,3,4,5,6,7)(1,2,3,4,8 5,6,7,9)
(2,3,5 1,4,6,7,8,9)(1,2,7,8,9 3,4,5,6)(8,9 1,2,3,4,5,6,7)(1,2,3,4,8 5,6,7,9)
(2,3,4,5 1,6,7,8,9)(1,2,7,8,9 3,4,5,6)(4,8,9 1,2,3,5,6,7)(1,2,3,4,8 5,6,7,9)
(2,3,4,5 1,6,7,8,9)(1,2,3,7,8,9 4,5,6)(3,4,8,9 1,2,5,6,7)(1,2,3,4,8 5,6,7,9)
(2,3,4,5 1,6,7,8,9)(1,2,3,4,7,8,9 5,6)(3,4,8,9 1,2,5,6,7)(1,2,3,8 4,5,6,7,9)
(2,3,4,5 1,6,7,8,9)(1,2,3,4,7,8,9 5,6)(4,8,9 1,2,3,5,6,7)(1,2,8 3,4,5,6,7,9)
(3,4,5 1,2,6,7,8,9)(1,4,7,8,9 2,3,5,6)(8,9 1,2,3,4,5,6,7)(1,2,3,4,8 5,6,7,9)
(3,4,5 1,2,6,7,8,9)(1,7,8,9 2,3,4,5,6)(4,8,9 1,2,3,5,6,7)(1,2,3,4,8 5,6,7,9)
(4,5 1,2,3,6,7,8,9)(1,7,8,9 2,3,4,5,6)(3,4,8,9 1,2,5,6,7)(1,2,3,4,8 5,6,7,9)
(5 1,2,3,4,6,7,8,9)(1,7,8,9 2,3,4,5,6)(3,4,8,9 1,2,5,6,7)(1,2,3,8 4,5,6,7,9)

****** ASSIGNMENTFOUND ******

The assignments are :

State 1 --> 0111
State 2 --> 0101
State 3 --> 0100
State 4 --> 1100
State 5 --> 0000
State 6 --> 0001
State 7 --> 0011
State 8 --> 0110
State 9 --> 0010

execution time : 0 h 0 m 28 s

END OF THE PROGRAM

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

(224) Book of abstracts of the first Benelux-Japan Workshop on Information and Communication Theory, Eindhoven, The Netherlands, 3-5 September 1989.
Ed. by Han Vinck.
EUT Report 89-E-224. 1989. ISBN 90-6144-224-9

(225) Hoeijmakers, M.J.
A POSSIBILITY TO INCORPORATE SATURATION IN THE SIMPLE, GLOBAL MODEL OF A SYNCHRONOUS MACHINE WITH RECTIFIER.
EUT Report 89-E-225. 1989. ISBN 90-6144-225-7

(226) Dahiya, R.P. and E.M. van Veldhuizen, W.R. Rutgers, L.H.Th. Rietjens
EXPERIMENTS ON INITIAL BEHAVIOUR OF CORONA GENERATED WITH ELECTRICAL PULSES SUPERIMPOSED ON DC BIAS.
EUT Report 89-E-226. 1989. ISBN 90-6144-226-5

(227) Bastings, R.H.A.
TOWARD THE DEVELOPMENT OF AN INTELLIGENT ALARM SYSTEM IN ANESTHESIA.
EUT Report 89-E-227. 1989. ISBN 90-6144-227-3

(228) Hekker, J.J.
COMPUTER ANIMATED GRAPHICS AS A TEACHING TOOL FOR THE ANESTHESIA MACHINE SIMULATOR.
EUT Report 89-E-228. 1989. ISBN 90-6144-228-1

(229) Oostrom, J.H.M. van
INTELLIGENT ALARMS IN ANESTHESIA: An implementation.
EUT Report 89-E-229. 1989. ISBN 90-6144-229-X

(230) Winter, M.R.M.
DESIGN OF A UNIVERSAL PROTOCOL SUBSYSTEM ARCHITECTURE: Specification of functions and services.
EUT Report 89-E-230. 1989. ISBN 90-6144-230-3

(231) Schemmann, M.F.C. and H.C. Heyker, J.J.M. Kwaspen, Th.G. van de Roer
MOUNTING AND DC TO 18 GHz CHARACTERISATION OF DOUBLE BARRIER RESONANT TUNNELING DEVICES.
EUT Report 89-E-231. 1989. ISBN 90-6144-231-1

(232) Sarma, A.D. and M.H.A.J. Herben
DATA ACQUISITION AND SIGNAL PROCESSING/ANALYSIS OF SCINTILLATION EVENTS FOR THE OLYMPUS PROPAGATION EXPERIMENT.
EUT Report 89-E-232. 1989. ISBN 90-6144-232-X

(233) Nederstigt, J.A.
DESIGN AND IMPLEMENTATION OF A SECOND PROTOTYPE OF THE INTELLIGENT ALARM SYSTEM IN ANESTHESIA.
EUT Report 90-E-233. 1990. ISBN 90-6144-233-8

(234) Philippens, E.H.J.
DESIGNING DEBUGGING TOOLS FOR SIMPLEXYS EXPERT SYSTEMS.
EUT Report 90-E-234. 1990. ISBN 90-6144-234-6

(235) Heffels, J.J.M.
A PATIENT SIMULATOR FOR ANESTHESIA TRAINING: A mechanical lung model and a physiological software model.
EUT Report 90-E-235. 1990. ISBN 90-6144-235-4

(236) Lammers, J.O.
KNOWLEDGE BASED ADAPTIVE BLOOD PRESSURE CONTROL: A Simplexys expert system application.
EUT Report 90-E-236. 1990. ISBN 90-6144-236-2

(237) Ren Qingchang
PREDICTION ERROR METHOD FOR IDENTIFICATION OF A HEAT EXCHANGER.
EUT Report 90-E-237. 1990. ISBN 90-6144-237-0

Eindhoven University of Technology Research Reports
Faculty of Electrical Engineering

ISSN 0167-9708
Coden: TEUEDE

(238) Lammers, J.O.
THE USE OF PETRI NET THEORY FOR SIMPLEXYS EXPERT SYSTEMS PROTOCOL CHECKING.
EUT Report 90-E-238. 1990. ISBN 90-6144-238-9

(239) Wang, X.
PRELIMINARY INVESTIGATIONS ON TACTILE PERCEPTION OF GRAPHICAL PATTERNS.
EUT Report 90-E-239. 1990. ISBN 90-6144-239-7

(240) Lutgens, J.M.A.
KNOWLEDGE BASE CORRECTNESS CHECKING FOR SIMPLEXYS EXPERT SYSTEMS.
EUT Eeport 90-E-240. 1990. ISBN 90-6144-240-0

(241) Brinker, A.C. den
A MEMBRANE MODEL FOR SPATIOTEMPORAL COUPLING.
EUT Report 90-E-241. 1990. ISBN 90-6144-241-9

(242) Demarteau, J.I.M. and H.C. Heyker, J.J.M. Kwaspen, Th.G. van de Roer
MICROWAVE NOISE MEASUREMENTS ON DOUBLE BARRIER RESONANT TUNNELING DIODES.
EUT Report 90-E-242. 1990. ISBN 90-6144-242-7

(243) Massee, P. and H.A.L.M. de Graaf, W.J.M. Balemans, H.G. Knoopers, H.H.J. ten Kate
PREDESIGN OF AN EXPERIMENTAL (5-10 MWt) DISK MHD FACILITY AND PROSPECTS OF COMMERCIAL (1000 MWt) MHD/STEAM SYSTEMS.
EUT Report 90-E-243. 1990. ISBN 90-6144-243-5

(244) Klompstra, Martin and Ton van den Boom, Ad Damen
A COMPARTSON OF CLASSICAL AND MODERN CONTROLLER DESIGN: A case study.
EUT Report 90-E-244. 1990. ISBN 90-6144-244-3

(245) Berg, P.H.G. van de
ON THE ACCURACY OF RADIOWAVE PROPAGATION MEASUREMENTS: Olympus propagation experiment.
EUT Report 90-E-245. 1990. ISBN 90-6144-245-1