AUTOMATIC DESIGN OF FINITE STATE MACHINES WITH ELECTRICALLY PROGRAMMABLE DEVICES

Marek Perkowski, Associate Professor of Electrical Engineering, Portland State University, P.O. Box 751, Portland , OR 97207,
Hoang Uong, Software Engineer, Intel, 5200 N.E. Elam Young Pkwy, Hillsboro, OR,
Hoa Uong, Software Engineer,
Kentrox Industries Inc., 14375 N.W. Science Park Dr. OR.

SUMMARY

The paper describes a system for automatic design of large Finite State Machines and Boolean Functions with an aid of Electronically Programmable Devices. This system will be used as a preprocessor to standard logic minimization/fitting software systems available for EPLD's. It is particularly suitable to the optimization of large state machines and functions as well as partitioning them onto several devices. The design stages of the system are described and some algorithms are illustrated with examples.

1. INTRODUCTION

Software systems for automatic design of finite state machines (FSM) have been created since early 1960's, and are in industrial use, typically in large companies. However a reneved high interest has been recently created in designing digital circuits with state machines. It can be also noticed in small companies, universities, and hobbyists applications. It is due to the advantages created by the LSI technology that enables to design efficiently such machines with the Programmable Devices (PLD's). By PLD's, we understand after Coppola [Copp 86] any device which can be programmed by the user to realize a chunk of combinatorial or sequential logic: PAL's, PLE's (Monolitic Memories), EPLD's (Intel, Altera), EEPLD's (Lattice), FPLA's, FPLS's (Signetics). The PLD devices permit to design relatively large FSM's and logic functions (also multi-level ones) in single chips or just with a few of them. A lot of software packages have been implemented to aid the designers in these tasks. Such packages are available from some universities (like U.C. Berkeley) and also sold from companies like Intel, Altera, Signetics, Data I/O and others. To the best of our knowledge the packages have several drawbacks: they are not especially tuned for EPLD design (Berkeley), automatic multi-level Boolean minimization is not allowed, partitioning and decomposition of FSM's and Boolean functions is either absent or is not minimal, there is usually no state assignment or other optimization of abstract Finite State Machine descriptions.

This paper describes a CAD system that is under design in the Department of Electrical Engineering at Portland State University. The system is used for designing with EPLD's, but can be adopted for any generic PAL's with two-level AND/OR registered array logic. This system permits a state minimization of FSMs (both completely and incompletely specified machines), FSM state assignment and decomposition, multi-level logic design, Boolean decomposition of multi-output incompletely specified Boolean functions and machine type convertion (Mealy machine to Moore machine and vice versa).

The FSM design methodologies and some of our FSM design tools have been described in [Perk 86a]. The tools used for optimization of the PLD's, particularly from Intel, are presented in [Copp 86]. In this paper, we will briefly describe the entire PLD CAD system but the main topic of our interest are the new tools.

2. PARTITIONING OF THE HIGH LEVEL DESCRIPTIONS OF CIRCUITS WITH EPLD'S

The Electronically Programmable Devices [Inte 86] permit to design a variety of sequential and combinational circuits. However, when the circuit under design is too large, a user has to perform the fitting process either manually (see page 2-68 in [Inte 86]) or even to partitionate his/her design onto several devices. Sometimes the partitioning process is straightforward and originates directly from the description of the circuit, by being the composition of several small cooperating state machines and/or logic functions. However, when a single large machine is initially described, as there is a case for a microprogrammed control unit of a special processor, the partitioning process can be very difficult when performed manually.

There are basically three possible approaches to the partitioning of large Finite State Machines:

1) partitioning (decomposition) on the level of the machine's structure description, before logic realization or even before states' assignment.
2) decomposition of the Boolean functions which realize the excitation functions of the machine. This decomposition is based on the principles of the Boolean logic and is performed before or during the logic minimization.
3) partitioning of the realizations of the Boolean functions (these are usually the minimized

Boolean equations, the truth tables in the form of arrays of cubes or the netlists) based on the graph-theoretical methods (min-cut, maximum clique and other algorithms).

In this paper only the first approach will be illustrated. The second approach is described in [Perk 87b].

Each type of EPLD device comes with some technological constraints that has to be satisfied during a mapping of the machines and functions in the design process. For simplification, we will assume that the following constraints exist:

1. number of dedicated inputs to the device (denoted by NDI),
2. number of inputs/outputs to the device (NIO),
3. number of macrocells with OR gates and flip-flops (NMC),
4. number of AND gates on the inputs to the OR gates in the macrocells (number of products in the Sum-of-Products realizations) (NAG).

For instance, for some well-known Intel's EPLD's these parameters are:

for 5C031:
NDI = 10, NIO = 8, NMC = 8, NAG = 8.

for 5C060:
NDI = 4, NIO = 16, NMC = 16, NAG = 8.

for 5C090:
NDI = 12, NIO = 24, NMC = 24, NAG = 8.

for 5C180:
NDI = 16, NIO = 48, NMC = 48, NAG = 8.

3. DESIGN FLOW OF THE SYSTEM

The general data flow of our system is shown in Fig. 1.

There are several data input formats to our system.

The flow-diagram of a control unit can be read in the format *.flow similiar to the input file of the program Peg from University of California in Berkeley [Hama 84]. A description of the register-transfer flowchart in format *.flow can be converted to either a completely or an incompletely specified machine by program Conversion [Perk 87a]. These machines are described in the input format of the Kiss state assignment program from UC Berkeley [DeMi 85] (called *.kiss format). (This is a standard form used for benchmarking of the CAD programs that deal with the FSM's. Many examples of machines in this format can be received from SIGDA, UC Berkeley or other sources [MCNC 87].) The *.kiss format description is stored in a file, that can be converted to our state table format (called *.stab format) by program KissToStab.

A classical form of the FSM state-table with the rows corresponding to symbolic internal states and the columns corresponding to symbolic input states can be entered interactively by the user [Perk

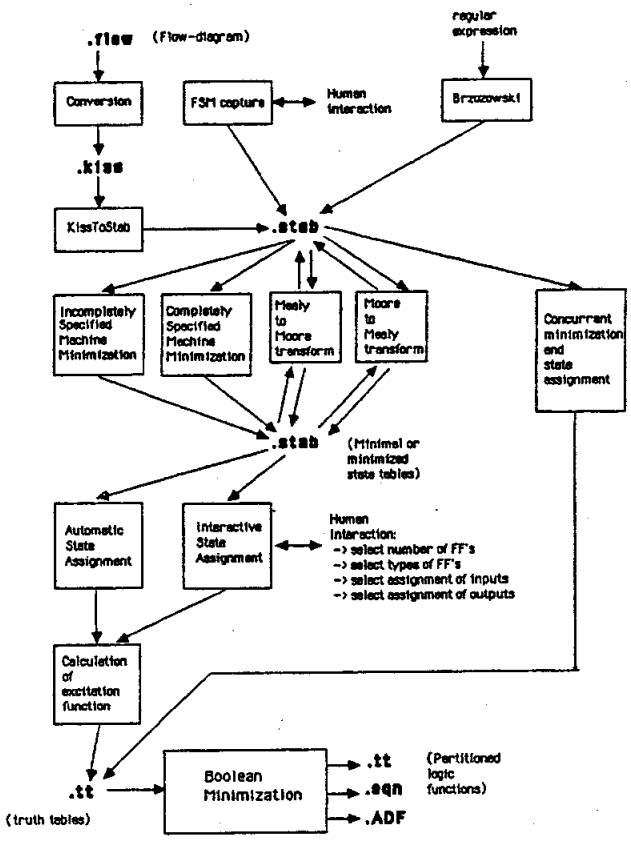


Fig.1.

85a], [Perk 86b]. A *.stab format file is created.

The textual ASCII file with the description of a regular expression or a nondeterministic Rabin-Scott machine can be read and converted to the state table with a use of the Prolog program [Perk 86]. The Brzozowski's method of the derivatives of regular expressions is applied. Machines produced by this program are not always minimal, since the fast equivalency check of regular expressions is applied, that can not recognize equivalency of some expressions. Hence, further minimization of the number of internal states is recommended.

The output from the system is in the form of a set of partitioned logic functions that is described in the truth table format (LIF, U.C. Berkeley) or as the logic equations format (Eqntott format from U.C. Berkeley). For the future, the ADF format of Intel's PLD software tools will be also generated. Hence, the system would be designed as a preprocessor to the existing university and commercially developed tools, especially those of U.C. Berkeley, Intel and Altera.

Our goal is to develop the integrated CAD system which will permit for completely automatic design of Finite State Machines, starting from various input descriptions (flow-graphs, graphs, tables, regular expressions) and producing different partitioned PLD-based realizations (also other than Sum-of-Products generic PAL architectures).

The design stages illustrated in the Figure 1 will be discussed in the next paragraphs.

4. STATE MACHINE DESIGN

4.1. STATE TABLES

By a Mealy machine we will understand an ordered 5-tuple
$M = < A, X, Y, \delta, \lambda >$,
where
A - is a set of internal states,
X - is a set of input states (symbols),
Y - is a set of output states,
δ: A x X -> A is a next-state (transition) function,
λ: A x X -> Y is an output function.

A state table of the machine is a 2-dimensional array with internal states as rows and input states as columns. A table's cell on the intersection of row A_i and column X_j includes next state A_r upon slash and a present output state Y_n below slash. This means that:
$\delta(A_i, X_j) = A_r$,
$\lambda(A_i, X_j) = Y_n$,

A Moore machine is defined as an ordered 5-tuple like a Mealy machine, the only difference being the fact that $\delta: A \rightarrow Y$, which means that the output states are functions of only the memory elements, and not memory elements and input signals as in the case of the Mealy machine.

4.2. THE ROLE OF THE DON'T CARES AND STATE TABLE MINIMIZATION

4.2.1. USING INVARIANTS TO INCREASE THE NUMBER OF DON'T CARES IN THE STATE TABLE

We have noticed that the decrease of the number of the PLD chips necessary to design a large machine can be obtained when the initial description (the FSM state table or the Boolean function) includes don't cares in one form or another. Therefore, our input formats permit to specify the don't cares directly or our algorithms generate them automatically from input descriptions. Direct specification of don't cares is done for the state tables (in format *.stab), for the state transitions of FSM's (in format *.kiss) and for the truth tables of Boolean functions (in format *.tt).

The flow-diagram is transformed to the Mealy machine's state table, whose columns correspond to disjoint cubes of input signals, and rows to the internal states of the machine [Perk 86b]. The number of states in this table can be now minimized, but better results are often obtained, when at first the number of the don't care cells in the table is increased. This can be done by using the invariants of the flow-diagram, which are inserted in the flow-diagram description by the user or are generated automatically from the flow-diagram [Perk 87a]. Increase in the number of don't care cells in the table permits in general case to minimize the number of the internal states of the machine better (which means less flip-flops), to find better state assignment and realization with smaller number of PLD devices. The number of internal states in the table can be minimized with use of one of the two algorithms: one of them for completely specified machines, and one for incompletely specified machines. The algorithm for the completely specified machines is a modification of one from [Koha 70]. The variant of the program that is to be applied is selected by the user. The branch-and-bound program for incompletely specified state table minimization is described in [Perk 85a]. The new, improved variant for incompletely specified machines is presented in [Perk 87]. It permits to optimize machines with up to 100 internal states. State minimization is especially useful when the state table is generated automatically from a high level description like a regular expression or a flow-diagram, or when the Mealy to Moore or the Moore to Mealy transformation had been previously executed.

One variant of our state minimization algorithm differs slightly from the classical approach. By symbol φ we denote "no operation" output which is treated during the state minimization as an output don't care. It means, that two internal states can be combined with respect to some column of the state table, when:
- they have compatible next states,
- they have consistant output states or "no operation" states and any other output state in this column.

After minimization of the state table and before the state assignment, the symbols φ are replaced with vectors of zeros, with as many zeros as the number of output signals of this machine. This replacement is done in order to prohibit creation of too large output implicants (i.e. the implicants which have a number of literals reduced too much) at the stage of Boolean minimization. Generation of such implicants could undesirably change the specification of the behavior of the state table (the behavior would be not equivalent to the initial specification).

Example 1.

A flow-diagram of the multiplying unit is shown in Fig.2. This unit calculates w = y * x by using only up- and down- counters in the data path. The multiplication by repetitive counter addition is applied.

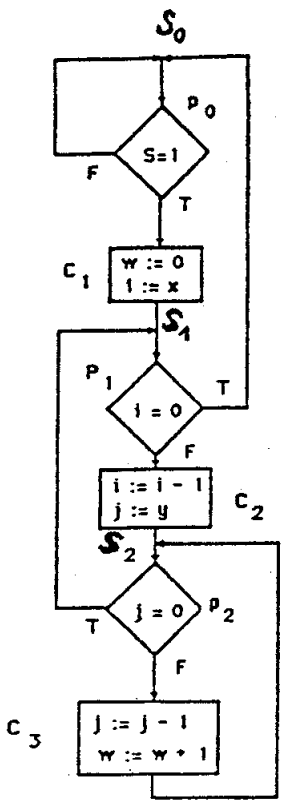


Fig. 2.

Let us assume at first (for comparison) that the invariants are not taken into account. From the flow-diagram of Fig. 2 the Mealy state table of Fig. 3 is created.

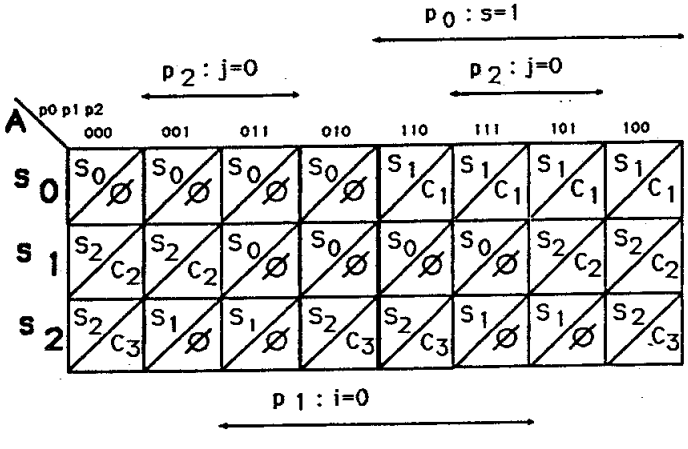


Fig. 3.

The number of internal states in this table cannot be minimized because any pair of states, like the pair of S_1 and S_2 or the pair of pair S_2 and S_0 have different outputs for the same input signal combinations (for instance S_0 and S_1 have outputs C_1 and C_2, respectively, for input combination $p_0 \overline{p_1}\,\overline{p_2}$ (corresponding to column 100 of the state table)).

The respective realization of excitation functions is shown in Figs. 4 a, b and output functions in Figs. 4 c, d.

a)

Q_1Q_2 \ $p_0p_1p_2$	000	001	011	010	110	111	101	100
00	00	00	00	00	01	01	01	01
01	10	10	00	00	00	00	10	10
11	-	-	-	-	-	-	-	-
10	10	01	01	10	10	01	01	10

b)

$$D_1 = Q_1\overline{p_2} + Q_2\overline{p_1}$$

$$D_2 = \overline{Q_1}\,\overline{Q_2}p_0 + Q_1p_2$$

c)

Q_1Q_2 \ $p_0p_1p_2$	000	001	011	010	110	111	101	100
00	Ø	Ø	Ø	Ø	C_1	C_1	C_1	C_1
01	C_2	C_2	Ø	Ø	Ø	Ø	C_2	C_2
11	—	—	—	—	—	—	—	—
10	C_3	Ø	Ø	C_3	C_3	Ø	Ø	C_3

d)

$$C_1 = p_0\overline{Q_1}\,\overline{Q_2}$$

$$C_2 = Q_2\overline{p_1}$$

$$C_3 = Q_1\overline{p_2}$$

Fig. 4.

As we see, two D flip-flops are required to realize excitation functions D_1 and D_2 of the sequential control unit.

Now, let us assume that the invariants of the flow-diagram from Fig. 2 are used. Since in S_0 the invariants (i=0) and (j=0) hold, then the transitions in the state table of Fig. 3 are specified for $p_1 = p_2 = 1$. Similiarly, since in S_1 the invariant (j=0) holds, in S_1 the transitions are specified for $p_2 = 1$. After introducing of all don't cares this way, the table of Fig. 3 takes the form of that of Fig. 5a.

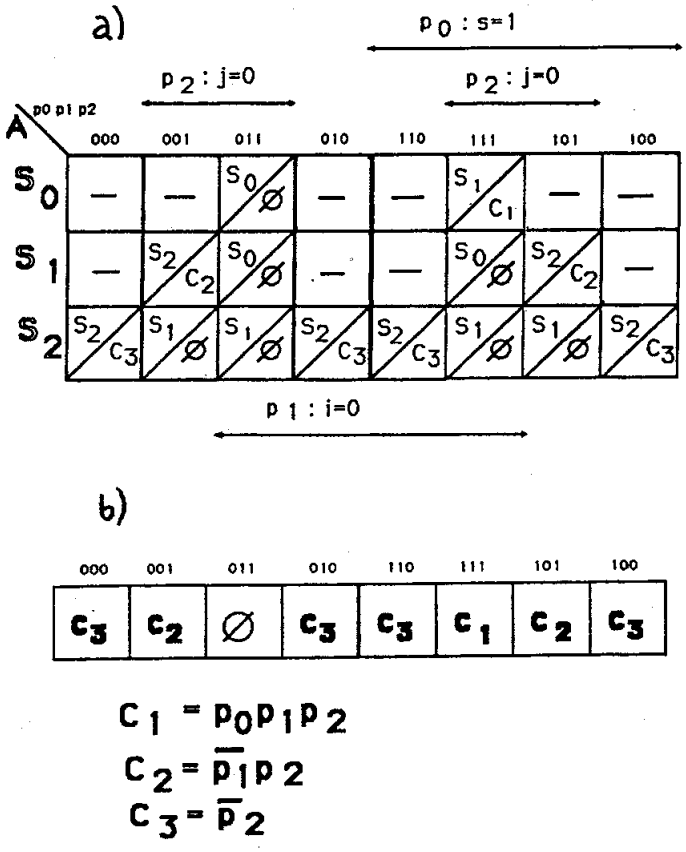


Fig. 5.

The Mealy table from Fig. 5a can be now minimized (internal states minimization) as in Fig. 5b. Let us observe, that the minimized machine from Fig. 5b has only one state! The output functions are:

$C_1 = p_0 \,\&\, p_1 \,\&\, p_2$, $C_2 = \overline{p_1}\, p_2$, $C_3 = \overline{p_2}$.

Hence, in this particular case the control unit has been reduced from a sequential to a purely combinational circuit.

Such deep reductions of control units are rare, but very substantial reduction in state machine complexity through generation of don't cares in the control unit description is very common and has been observed by us in many practical control units.

4.2.2. MINIMIZATION OF THE NUMBER OF INTERNAL STATES

Minimization of internal states of the state table consists in finding such groups of states that all states from each group can be joined together into one new state and the resultant machine with new states will be still behaviorally equivalent to the initial machine. This is achieved when the groups of states are *complete* and *closed*. Complete means, that all machine's internal states are included in (at least) one of the selected groups. Closed means, that each pair of states whose compatibility is implied by any pair of states from any selected group of states is also included in at least one of the selected groups (for these basic definitions see [Koha 70] or [Perk 85a]).

Example 2.

Given is a state table from Fig. 6.

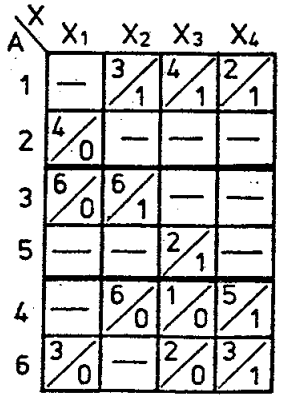


Fig. 6.

Now the groups of compatible states for this table are found according to the method from [Perk 85a]. They are shown as nodes of the implication graph in Fig. 7.

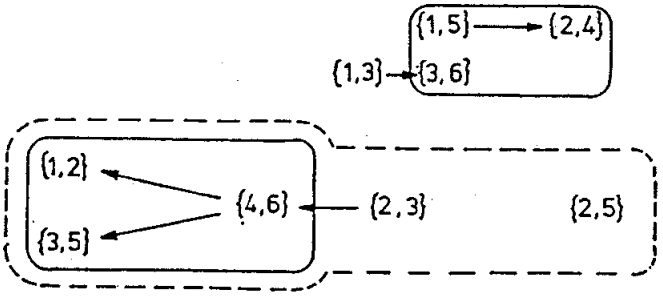


Fig. 7.

For instance, the states 2 and 5 are compatible, as

are also the states 2 and 3. Arrows of the graph denote the relation of the implied compatibility between the pairs of states. For instance arrows from pair {4, 6} to pair {1, 2} and {3, 5} mean that states 4 and 6 are compatible under condition that states 1 and 2 are compatible and states 3 and 5 are compatible. This can be observed in the table: when we want to merge states 4 and 6, the successors (next states) in column X_3 (states 1 and 2) and in column X_4 (states 5 and 3) must be compatible as well. The set of compatible groups {{1,5}, {2,4}, {3,6}} is compatible and closed, since there are no arrow going out of the respective subgraph (no pair is implied that do not belong to it). Similiarly compatible and closed are the following sets of of groups of internal states: {{1,2), {3,5), {4,6)} and {{1,2), {2,3,5), {4,6). (group {2,3,5} is created from pairs {2,3}, {3,5} and {2,5)). A set with three groups can be used to generate a machine with three states. All these three sets have three groups, so any of them is selected for further minimization (it can be proven that no better set exists). When the set {{1,2), {3,5), {4,6)} is selected, states 1 and 2, 3 and 5, 4 and 6 are merged (see Fig. 6 and Fig. 8a. After renaming states: {1,2} to A, {3,5} to B and {4,6} to C the new state table from Fig. 8b is created. The set of states that is included into other sets is renamed by the symbol of any set that includes it. For instance state 4 in the intersection of row {1,2} and column X_1 of table from Fig. 8 is replaced with C, since 4 is included in set {4,6} to which symbol C was assigned.

a)

A \ X	X_1	X_2	X_3	X_4
{1,2}	4/0	3/1	4/1	2/1
{3,5}	6/0	6/1	2/1	—
{4,6}	3/0	6/0	1,2/0	3,5/1

b)

A \ X	X_1	X_2	X_3	X_4
A	C/0	B/1	C/1	A/1
B	C/0	C/1	A/1	—
C	B/0	C/0	A/0	B/1

Fig. 8.

The number of the internal states of our machine has been then minimized from 6 to 3 and the number of flip-flops (assuming state assignment with the minimum number of flip-flops) from 3 to 2.

Such minimization permits for better fitting of large machines or groups of machines to PLD devices. The number of the devices can be reduced.

Minimization of the number of states can sometimes increase the complexity of the excitation and/or output functions. We cannot then treat the minimization stage dogmatically, as a must in the design process. What we wanted to achieve in our system, was rather to create an integrated environment in which the user can experiment with various optimization methods. These methods can be sometimes complementary, but for other machines the effects of transformations are mutually anihilating. Hence, the design trade-offs must be investigated in each particular case by the designer.

4.2.3. COLUMN MINIMIZATION OF STATE TABLES

Besides state minimization for internal states, discussed in many textbooks, another type of state minimization can be sometimes useful: minimization of the number of columns. Such minimization can be performed either before or after the minimization of number of the internal states. These two types of minimization can be iterated. Minimization of the number of columns is done using the method of the graph coloring of a graph, whose nodes correspond to the columns of the initial state table. We will call this graph a **column coloring graph.** An edge exists between two nodes if the corresponding columns are not compatible (not compatible are the columns that cannot be merged together into a single column). The graph is colored in a proper way: i.e. any two nodes linked with an edge should be assigned different colors. The number of different colors used in coloring is to be minimized. All columns colored with the same color are next merged together into a single column. If the number of colors in the coloring is smaller than the number of the nodes, then the number of columns of the machine is reduced.

Example 3.

Given is a state graph from Fig. 9. Such graph corresponds to the *.kiss format, obtained from a flow-diagram in *.flow format.

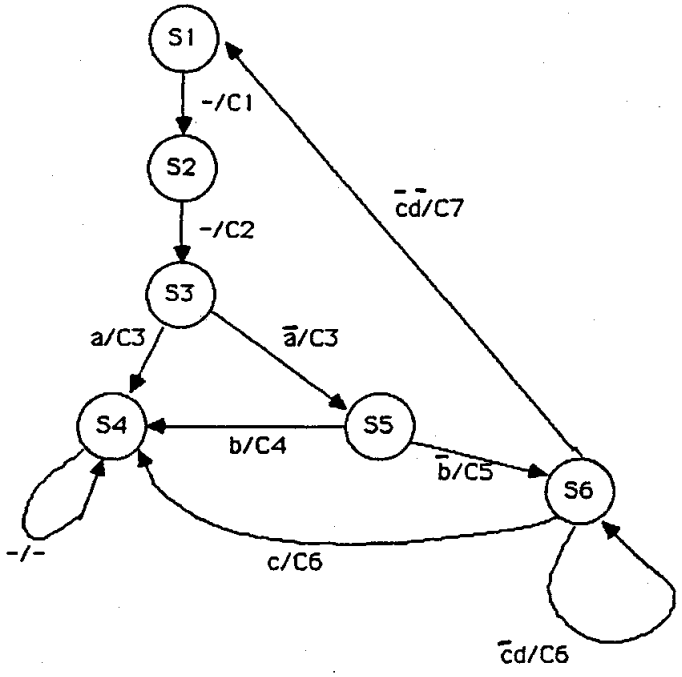


Fig. 9.

To find the cubes being the headers of the columns of the state machine a set of formulas is generated for each internal state of the graph, for which a branching exists. Such states are S3, S5 and S6. The formulas are:

for state S3 : $F(S3) = \{a, \bar{a}\}$,

for state S5 : $F(S5) = \{\bar{b}, b\}$,

for state S6 : $F(S6) = \{c, \bar{c}d, \bar{c}\bar{d}\}$.

To find a set of column cubes we calculate a formula:

$F(3) \,\&\, F(5) \,\&\, F(6) =$

$\{ a\bar{b}\bar{c}, a\bar{b}\bar{c}d, a\bar{b}\bar{c}\bar{d}, abc, abc\bar{d}, ab\bar{c}\bar{d}, \bar{a}\bar{b}\bar{c}, \bar{a}\bar{b}\bar{c}d, \bar{a}\bar{b}\bar{c}\bar{d}, \bar{a}bc, \bar{a}bc\bar{d}, \bar{a}b\bar{c}\bar{d} \}$.

Now the state table is created from the graph of Fig. 9 with the above cubes as columns (Fig. 10).

	abc	abcd	abcd	abc	abcd	abcd	abc	abcd	abcd	abc	abcd	abcd
S1	S2/C1	S2/C1	S2/C1	S2/C1	S2/C1	S2/C1	S2/C1	S2/C1	S2/C1	S2/C1	S2/C1	S2/C1
S2	S3/C2	S3/C2	S3/C2	S3/C2	S3/C2	S3/C2	S3/C2	S3/C2	S3/C2	S3/C2	S3/C2	S3/C2
S3	S4/C3	S4/C3	S4/C3	S4/C3	S4 /C3	S4/C3	S5/C3	S5/C3	S5/C3	S5/C3	S5/C3	S5/C3
S4	S4 /-	S4/-	S4 /-	S4/-	S4/-	S4/-	S4/-	S4/-	S4 /-	S4/-	S4 /-	S4/-
S5	S6/C5	S6/C5	S6/C5	S4 /C4	S4/C4	S4/C4	S6/C5	S6/C5	S6/C5	S4/C4	S4/C4	S4/C4
S6	S4/C6	S6/C6	S1/C7	S4/C6	S6/C6	S1/C7	S4/C6	S6/C6	S1/C7	S4/C6	S6/C6	S6/C7

Fig. 10.

Let us now assume, that the following invariants exist for the graph:
- for state S1: $a \neq b$,
- for state S2: $a = c$,
- for state S3: $b = 0$,
- for state S4: $a = 1$ & $b = 1$,
- for state S5: $(a = 1 \,\&\, c = 0)$ or $(a = 0 \,\&\, c = 1)$,
- for state S6: $b = 0$.

By a specified transition we understand a pair of the next state and the present output in a table's cell, where at least one of the symbols is other than a don't care. Let us discuss now how the specified transitions are calculated for internal state S1. By Boolean multiplication of all the header cubes of the table with the function $a\bar{b} + \bar{a}b$ (corresponding to the invariant of state S1) we get the following cubes:

$a\bar{b}\bar{c}, a\bar{b}\bar{c}d, a\bar{b}c\bar{d}, a\bar{b}\bar{c}\bar{d}, \bar{a}bc, \bar{a}bc\bar{d}, \bar{a}b\bar{c}d$.

Hence, the cells of the table being on the intersection of the columns with these cubes as headers and the row S1 remain as they have been specified in table of Fig. 10. Transitions from state S1 for columns corresponding to other input cubes are non-specified and are filled with don't cares symbols (dashes). By continuing this process for other internal states the table from Fig. 11 is created.

	1	2	3	4	5	6	7	8	9	10	11	12
	abc	abcd	abcd	abc	abcd	abcd	abc	abcd	abcd	abc	abcd	abcd
S1	S2/C1	S2/C1	S2/C1	--	--	--	--	--	--	S2/C1	S2/C1	S2/C1
S2	S3/C2	--	--	S3/C2	--	--	--	--	S3/C2	--	S3/C2	S3/C2
S3	S4/C3	S4/C3	S4/C3	--	--	--	S5/C3	S5/C3	S5/C3	--	--	--
S4	--	--	--	S4/-	S4/-	S4/-	--	--	--	--	--	--
S5	--	S6/C5	S6/C5	--	S4/C4	S4/C4	S6/C5	--	--	S4/C4	--	--
S6	S4/C6	S6/C6	S1/C7	--	--	--	S4/C6	S6/C6	S1/C7	--	--	--

Fig. 11.

To optimize this table we will first minimize the number of columns. Two columns are compatible and can be merged together when in each row one of the following conditions is satisfied:
- they have the same next states and outputs,
- there is a don't care in at least one of the two columns.

For the table from Fig. 11 a coloring graph is created (Fig. 12a). This graph is colored as in Fig. 12b.

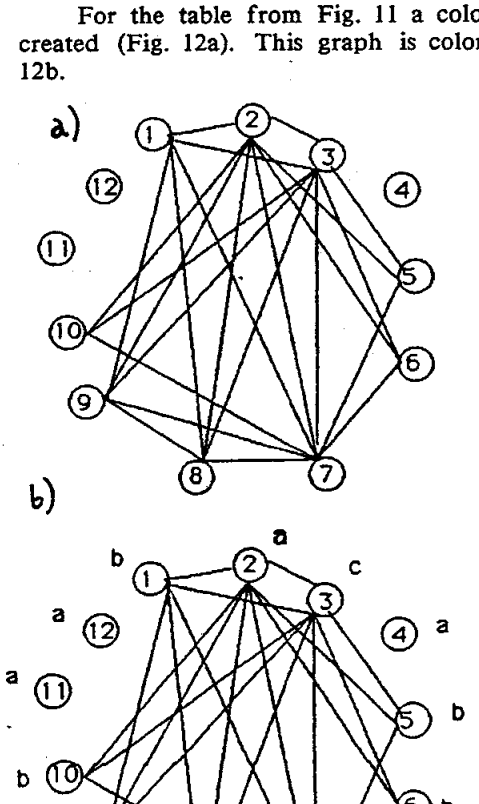


Fig. 12.

After merging columns of the same color the table of Fig. 13 is generated. For instance, the first column of this table originates from columns 1, 5, 6 and 10 merged together.

	1,5, 6,10	2,4, 11,12	3	7	8	9
S1	S2/C1	S2/C1	S2/C1	--	--	--
S2	S3/C2	S3/C2	--	--	--	S3/C2
S3	S4/C3	S4/C3	S4/C3	S5/C3	S5/C3	S5/C3
S4	S4/-	S4/-	--	--	--	--
S5	S4/C4	S6/C5	S6/C5	S6/C5	--	--
S6	S4/C6	S6/C6	S1/C7	S4/C6	S6/C6	S1/C7

Fig. 13.

This table is now minimized using standard methods for incompletely or completely specified machines (incompletely specified in this particular case). Comparison of all pairs of states (triangle table of Fig. 14a) produces the relation of implied compatiblity of states. For instance the triangle table shows, that states S1 and S4 are compatible under conditions that states S2 and S4 are compatible. Recursive process is used to mark all incompatible pairs of states with symbols X. (In this particular example no new symbols X are introduced).

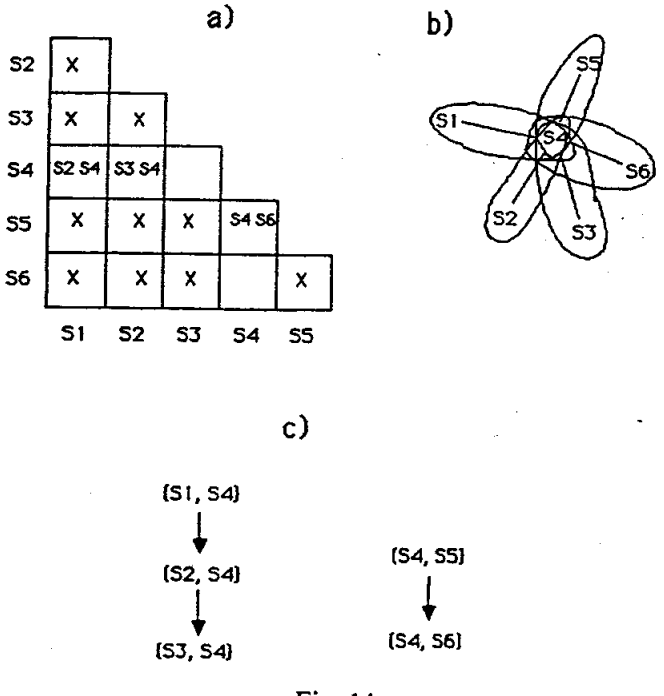


Fig. 14.

The merging graph created from the triangle table of Fig. 14a is presented in Fig. 14b. The nodes of the graph correspond to the internal states. An edge is created between two states, when these states in the triangle table are compatible (no symbol X exist in the corresponding cell of the table). The maximum cliques of the merging graph are: {(S1,S4), (S2,S4), (S3,S4), (S4,S5), (S4, S6)). The implication graph is shown in Fig. 14c. The closed and complete groups are in this example the same as the maximum cliques. As the result the table of Fig. 15 is obtained from the complete and closed groups and the table of Fig. 13. Now the column minimization process can be repeated. The column

	x	y	z	u	v	w
	1,5,6,10	2,4,11,12	3	7	8	9
S1,S4	S2,S4/C1	S2,S4/C1	S2,S4/C1	--	--	--
S2,S4	S3,S4/C2	S3,S4/C2	--	--	--	S3,S4/C2
S3,S4	S4,S5/C3	S4,S5/C3	S4,S5/C3	S4,S5/C3	S4,S5/C3	S4,S5/C3
S4,S5	S4,S6/C4	S4,S6/C5	S4,S6/C5	S4,S6/C5	--	--
S4,S6	S4,S6/C6	S4,S6/C6	S1,S4/C7	S4,S6/C6	S1,S4/C7	S1,S4/C7

Fig. 15.

coloring graph is shown in Fig. 16. The nodes (columns) x and v are colored with color a, the nodes y and u with color b, and the nodes z and w with color c. After merging the respective columns (x with v, y with u, and z with w) and renaming the states the table of Fig. 17 is created.

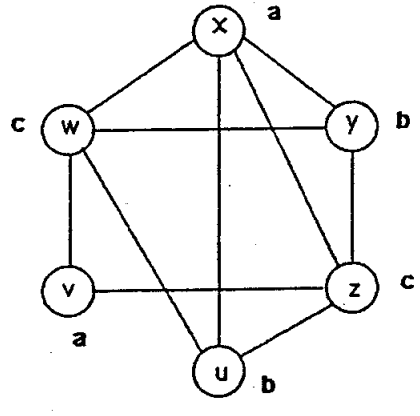


Fig. 16.

	x, v 1,5,6,10,8	y, u 2,4,11,12,7	z, w 3,9
A	B/C1	B /C1	B/C1
B	C /C2	C /C2	C/C2
C	D/C3	D /C3	D/C3
D	E/C4	E/C5	E/C5
E	E/C6	E/C6	A/C7

Fig. 17.

The resultant state table of Fig. 17 cannot be further minimized by state minimization. The first column of the table corresponds now to columns 1, 5, 6, 10, and 8 of the initial table. To design the state machine and the combinational inputs decoder the columns of the table of Fig. 17 are encoded

m , n	00	01	11	10
A	B/C1	B /C1	B/C1	--
B	C /C2	C /C2	C/C2	--
C	D/C3	D /C3	D/C3	--
D	E/C4	E/C5	E/C5	--
E	E/C6	E/C6	A/C7	--

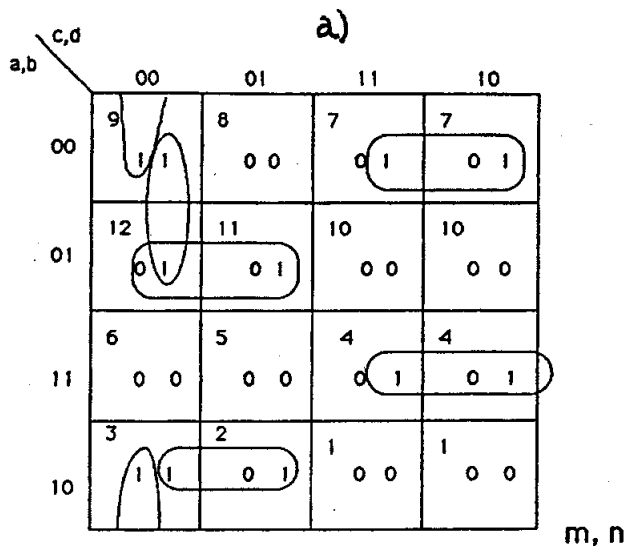
Fig. 18.

with values 00, 01, and 11 of signals m and n, respectively - see Fig. 18. Such assignment determines certain correspondence between signals a, b, c, d and signals m, n. This correspondence is represented as an array of cubes (truth table in *.tt format) and for illustration purposes is shown here in the form of a Karnaugh map (Fig. 19a). The equations:

$$m = \bar{b}\bar{c}\bar{d},$$

$$n = \bar{a}\bar{b}c + \bar{a}b\bar{c} + abc + ab\bar{c} + \bar{a}\bar{c}\bar{d}$$

are created by Boolean minimization. The structure of the entire realization of our machine is shown in Fig. 19b. The inputs decoder is described with the above equations and the FSM is described with the state table of Fig. 18. Similiarly, the output decoder can be created for signals C1 - C7.



b)

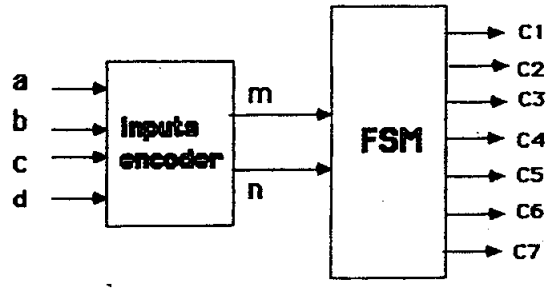


Fig. 19.

4.3. MEALY TO MOORE MACHINE TRANSFORMATION

It is in general case difficult to predict, which of the two possible realizations of the given machine is in the simpler form: a Mealy type, or an equivalent to it, Moore type. The Moore machine has a separate realization of output functions which can permit to fit it to a separate device. However, the possibilities of partitioning/minimizing/fitting of the transition functions are enhanced, since this functions have now less outputs and product terms (cubes) than the initial transitions/outputs functions. The equivalency of the machines is understood here in the sense of the same input/output behavior: being in equivalent states the machines respond with the same output sequences to any input sequence applicable to both of them (for the exact definitions see for instance [Koha 70]).

Let us observe, that the output states for the internal states of the Moore's machine are written in an additional column, as in Fig. 20d. Only the transition functions are written into the cells of the Moore maps, and not transitions (above slash) and output functions (below slash) as in the Mealy maps.

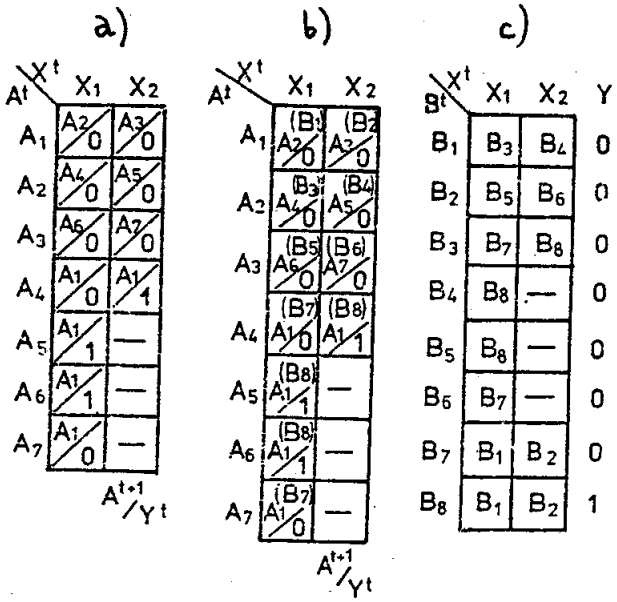


Fig. 20.

a) Mealy machine, b) allocation of states,
c) partially filled Moore table,
d) complete Moore table equivalent to the Mealy table of Fig. 20a.

Transformation of the state table of the Mealy machine into the table of an equivalent Moore machine is performed according to the following algorithm.

Algorithm to transform Mealy machine to an equivalent Moore machine.

1. Allocate the internal state symbol B_j of the new Moore machine to each different pair of next internal state and present output state (A_j, Y_j) from the cells of the Mealy machine's state table - identical states of the Moore machine will then correspond to identical pairs (Fig. 20b).
2. Create the Moore type state table, with as many rows as there are different state symbols B_j. Allocate to each state $B_j = (A_j, Y_j)$ the corresponding output signal Y_j from the pair (A_j, Y_j) as for the Moore type state output (Fig. 20c).
3. Fill the cells of the Moore table with the next state functions. The next states, B_r, for each X_i, are allocated to state $B_j = (A_j, Y_j)$ in the same manner as the state A_j has had, i.e. $(\delta_1(A_j, X_i), \lambda_1(A_j, X_i)) = (A_v, Y_z) = B_r$.

For instance, for Mealy state table of Fig. 20a the allocation of Moore machine's states is specified in Fig. 20b. We will allocate state B_1 to pair $A_2/0$, state B_2 to pair $A_3/0$, and so on. Because the pair $A_2/0$ corresponds to state B_1, the output signal for state will be equal 0 and the next states will be B_3 for input X_1 and B_4 for X_2, while these states exist in row A_2 of the state table from Fig. 20b. The complete state table of Moore machine is given in Fig. 20c.

The basis for the Moore to Mealy transformation shown above is the following theorem.

Theorem.

If $M_1 = < X, Y, A, \delta_1, \lambda_1 >$ is a Mealy machine then the Moore machine $M_2 = <X, Y, B_1, \delta_2, \lambda_2>$ is equivalent to it. The set B_1 and functions δ_2 and λ_2 are defined as follows:

$$B_1 = \{(A_j, Y_j) : (exists\ A_i \varepsilon A)(exists\ X_i \varepsilon X)$$
$$[A_j = \delta(A_i, X_i) \text{ and } (Y_j = \lambda_1(A_i, X_i))]\},$$
$$\delta_2(B_j, X_j) = \delta_2((A_j, Y_j), X_i) = (\delta_1(A_j, X_i), \lambda_1(A_j, X_i)),$$
$$\lambda_2(B_j) = \lambda_2((A_j, Y_j)) = Y_j.$$

Example 4.

State minimization is often necessary after execution of Mealy to Moore or Moore to Mealy transformations. Fig. 21 presents a Moore machine obtained from a Mealy machine of Fig. 3.

	000	001	011	010	110	111	101	100	
B1	B1	B1	B1	B1	B2	B2	B2	B2	∅
B2	B3	B3	B1	B1	B1	B1	B3	B3	C1
B3	B4	B5	B5	B4	B4	B5	B5	B4	C2
B4	B4	B5	B5	B4	B4	B5	B5	B4	C3
B5	B3	B3	B1	B1	B1	B1	B3	B3	∅

Fig. 21.

This new machine is not minimal and requires 3 flip-flops. In the process of minimization states B2 and B5 are merged and hence the minimized machine requires only two flip-flops. It is left to the reader to generate don't care cells for this machine, find realization and compare to the solution from Example 1. A machine from examples 1 and 4 is useful to demonstrate how combinations of various techniques, like invariants generation, state minimization, state assignment, Mealy <-> Moore transformations, and executed in different order, can essentially influence the quality and the form of the realization.

4.4. MOORE TO MEALY MACHINE TRANSFORMATION

The transformation of Moore state table into an equivalent Mealy state table consists in writing under slash, in each cell of the table with the next state A_i, the value of the Moore machine that corresponds to this state. Next the column of outputs is removed from the table. Fig. 22 illustrates a transformation of the Moore table (Fig. 22a) into an equivalent Mealy table (Fig. 22b).

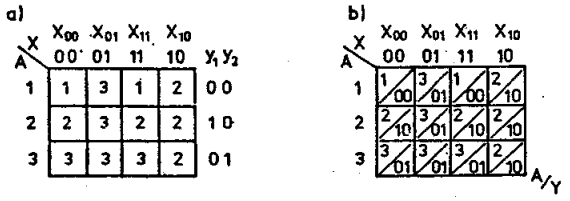


Fig. 22.

We have to bear in mind, that notation A_i / Y_i in the intersection of row A_j and column X_j means, that being in state A_j and receiving input state X_j the machine produces immediately the output state Y_j and only in the moment of time next (in the next clock's pulse) it transits to state A_i. Hence, let us assume, that the machine from Fig. 22a and the machine from Fig. 22b are both in their states 1 when input state X_{01} arrives. The Mealy machine produces an output state 01 immediately, while the Moore machine transits in the next pulse to state 3 and not earlier than then it produces an output 01. The Mealy and Moore machines are then equivalent, but the equivalency is with an accuracy of one clock pulse. We must remember this fact since it can sometimes have an influence on the design.

In our software package, the transformations between Moore and Mealy tables are possible for both before and after the state minimization. Different program pipelines can be organized by the user in the UNIX environment.

4.5. CONCURRENT MINIMIZATION AND STATE ASSIGNMENT

The currently used design approach is first to minimize the number of machine's internal states and follow it with the states' assignment. Both the state minimization and the state assignment problems are classical in the Automata Theory, and a number of approaches has been proposed. Most of the optimal state assignment algorithms are NP-hard, which means that they can practically find optimal solutions for only the small machines (12 - 15 states). The known approximate algorithms (like the approach of De Mitchelli, Sangiovanni-Vincentelli et al [DeMi 85]) yield non-minimal solutions, sometimes of poor quality. Minimization of the number of internal states results from the adopted assumption: "the more internal states, the more complicated is the realization, hence more memory elements are needed, there are more excitation functions, and therefore their realization is more complicated". Practical examples show the evidence, that we should neither seek a machine with the minimal number of internal states, nor the one that has excitation functions depending on the minimal number of variables - as lot of textbooks suggest, and as has been implemented in the current design automation systems. The examples of machines that have minimal realizations with the greater than minimum number of flip-flops can be easily found. Also, the attempt to find a realization of the set of excitation functions with minimal number of argument variables is often useless, because such realizations can have more chips than the other realizations of these functions. When these partitions are used for coding, the number of machine states are reduced, being a result of assigning the same codes to groups of compatible states. This corresponds to conducting a search for a machine, equivalent to the initial one, that minimizes the layout realization. Algorithms of this type, as well as algorithms for state assignment of minimal machines free from the above mentioned deficiencies, have been presented in [Lee 82], [Lee 84]. Each partition is evaluated separately: the value of its quality function is found by minimizing the corresponding Boolean function with multiple-valued input functions. Next, the branch-and-bound search in the space of sets of partitions is done, to minimize the value of the cost function. We use a method known from AI to guide this search by a calculated values of the quality function for nodes of the tree. The other advantages of the approach are the following ones: the cost of the output functions' realizations is taken into account as a part of the total cost; various types of flip-flops can be selected (D, JK, T - the best for each function); such an approach is especially advantageous for PLD devices in which various types of flip-flops are available.

Example 5.

A machine from Fig. 23a has been minimized to the form of Fig. 23c (states 6 and 7 were merged into new state 6). Next the machine from Fig. 23c has been optimally encoded as in Fig. 23d.

This state assignment leads to the following excitation and output functions:

$$D_1 = \overline{Q_1}\,\bar{X} + Q_1 x,$$
$$D_2 = Q_2\,\bar{x} + \overline{Q_2}\,x,$$
$$D_3 = \overline{Q_2} + \bar{x}\,\overline{Q_3} + \overline{Q_1}\,x,$$
$$z = \overline{Q_1}\,Q_2\,Q_3 + Q_1\overline{Q_2}\,Q_3 + \bar{x}\,\overline{Q_1}\,Q_2 + \bar{x}Q_1Q_2,$$

which requires 4-input OR gates in PLD (NAG ≥ 4) in the PLD.

Let us assume now that we use an algorithm

for concurrent state minimization and state assignment. For the non-minimized machine, the partitions are found:

$\tau_1 = \{\overline{1567}, \overline{2348}\}$,
$\tau_2 = \{\overline{1258}, \overline{3467}\}$,
$\tau_3 = \{\overline{1368}, \overline{2457}\}$.

See Fig. 23b.

These partitions are applied for state assignment of the machine from Fig. 23a and lead to the following excitation and output functions:

$D_1 = \overline{Q_1}$,
$D_2 = Q_2\bar{x} + \overline{Q_2}x$,

a)

A \ x	0	1
1	2/0	3/0
2	1/1	7/1
3	7/1	5/0
4	7/1	5/1
5	8/0	4/0
6	4/0	2/0
7	4/0	2/0
8	5/1	6/0

b)

	τ_1	τ_2	τ_3
1	0	0	0
2	1	0	1
3	1	1	0
4	1	1	1
5	0	0	1
6	0	1	0
7	0	1	1
8	1	0	0

c)

A \ x	0	1
1	2/0	3/0
2	1/1	6/1
3	6/1	5/0
4	6/1	5/1
5	8/0	4/0
6	4/0	2/0
8	5/1	6/0

d)

	τ_1	τ_2	τ_3
1	1	1	0
2	0	1	1
3	1	0	0
4	1	0	1
5	1	1	1
6	0	0	1
8	0	1	0

Fig. 23.

$D_3 = Q_2 + xQ_3 + \bar{x}\,\overline{Q_3}$,
$z = \bar{x}Q_1 + Q_1Q_3$.

This last realization requires OR gates with only three inputs. This example showes, that concurrent state minimization and state assignment (which corresponds to the not one-to-one assignment of an initial machine) can produce better results than the classical approach. This was confirmed on several practical examples. However, in general the designer has to try both approaches and compare the results.

4.6. STRUCTURE OF DECOMPOSED MACHINES AND ASSIGNMENT

Classical theory of state machines distinguishes two types of machines: Mealy and Moore machines. The structures that are used in order to implement such concepts in hardware are presented in Fig. 24a, b, c. The excitation (transition) and output functions of a Mealy machine can be realized either jointly (Fig. 24a) or separately (Fig. 24b). In both cases the functions are realized with PLA's, EPLD's, in TTL random logic or with any other method. In the case of EPLD's both realizations are used, depending on the size of the machine or machines that are to be mapped together.

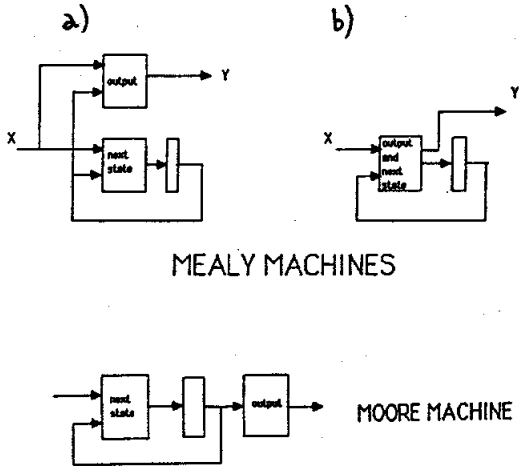


Fig. 24.

A Moore machine (Fig. 24c) can have the advantage of separate transition and output functions, as well as simpler realization of output function, which sometimes permits for better fitting of an abstract machine(s) to PLD devices.

To simplify the FSM realization, and especially to decrease number of inputs/outputs/flip-flops (depending on machine's type), realization of machines with separate logic for encoding of input and/or output signals is recommended. For

instance, Fig. 25 shows a Mealy machine with encoded inputs and encoded outputs. Such machines are designed by encoding combinations of input signals with symbolic input states and combinations of output signals with symbolic output states. Next, in the assignment process the assignments are found not for internal states only, as it is ususally done, but also for input and ouput symbolic states. In the phase of finding excitation functions and output functions of the internal machine the functions are created with encoded input signals as inputs and encoded output signals as outputs from the machine. Next descriptions of the two code converters are created in the form of truth tables:

- from original input signals to encoded input signals,
- from encoded output signals to original output signals,

MEALY MACHINE WITH INPUT AND OUTPUT ENCODING

Fig. 25.

These truth tables are minimized with standard tools and realized in EPLD's.

The applied by us assignment methods are very similiar for each of the three assignment processes:

- assignment of internal states,
- assignment of input states,
- assignment of output states.

Moreover, the order of these assignments can be arbitrary, and each next assignment process takes into account the results of the previously found assignments. This minimizes global constraints, and permits to investigate various constraints and compare many variants of structures and assignments.

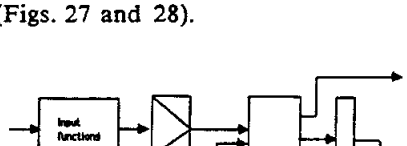
It can be observed, that in practise two types of FSM's are realized:

a) small (less than 16) number of input states, internal states and output states. All or almost all of combinations of input signals are allowed as input states in most of the internal states (like in a machine from the last example).

b) large (more than 30) number of input signals, output signals, and internal states. Most of the combinations of the input signals are however never used. Such machines are created from flow-diagrams (as in Example 1). They have limited branching factor, usually 2 (like predicates $p0$ and $p0$ in Example 1, or some small integer (for subsequent **if** statements or **case** statements in the flow-diagram description).

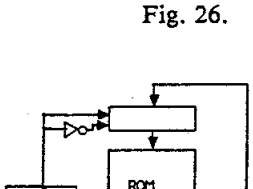
The machines of the first type are realized with the methods presented above.

The machines of the second type are realized in our system with various types of special units, like machines with selection of input signals (Fig. 26) or various types of microprogrammed units (Figs. 27 and 28).



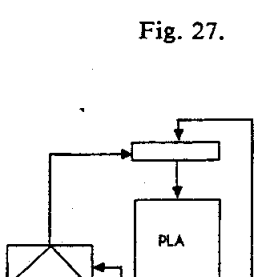
MEALY MACHINE WITH SELECTIONS OF INPUT SIGNALS

Fig. 26.



MICROPROGRAMMED MACHINE WITH ROM

Fig. 27.



MICROPROGRAMMED MACHINE WITH PLA SELECTED INPUTS AND ENCODED OUTPUTS

Fig. 28.

Although these machines are designed from the state graphs (tables with non-disjoint cubes as column headers) and not states tables (tables with disjoint cubes as column headers, as in the previous examples), many of the design methods for them are similiar to those presented here. They are based on the same graph-theoretical and combinatorial principles as graph coloring, set covering, maximum clique, and so on. The results produced are generally of worse quality, but the optimizations can be applied to the machines of much larger sizes.

The optimization methods include state minimization, decomposition, Mealy <-> Moore transformations, state assignment based on embedding to hypercubes, microcode placement in ROM, PLA minimization for microprogrammed FSM's and other.

5. BOOLEAN FUNCTION MINIMIZATION METHODS TO FIT A FUNCTION TO THE DEVICE(S)

The Boolean minimization process of the system includes at present basically four stages:

1. Reduction of the number of input variables to a function,
2. Decomposition of a Boolean function to combinational blocks,
3. Two-level Boolean minimization of each block.
4. Multi-level Boolean minimization of each block.

In the first stage a tree search algorithm is used to find all minimal sets of variables on which the initial function depends. This is done in order to fit functions to single devices, since the number of inputs NDI + NIO for devices is limited.

In the second stage we use new algorithms (see [Perk 87b]) which generalize ideas of the method presented recently by T. Sasao [Sasa 87]. The function is decomposed into chains of interconnected combinational blocks, each block being an arbitrary Boolean function of some subset of the input variables. Multi-wire connections between two blocks are allowed for better decomposability. The goal of the decomposition is to reduce the numbers of inputs to blocks in such a way, that each block will fit to a single PLD. We extend the approach of Sasao for the case of incompletely specified, multi-output functions. Some efficient coding methods for multiple-valued signals between logical blocks. have been also introduced.

In the third stage an algorithm described in [Nguy 86] and [Nguy 87] is used, for each single-output function of each block separately. When required by the user, the output polarity of any function can be inverted. If necessary (for instance when asynchronous machines are realized) a free of hazards solution is generated.

The fourth stage uses an algorithm of stage 3, modified to design multi-level functions.

6. CONCLUSION

Some techniques to minimize and partitionate Finite State Machines have been described. They are particularly useful in EPLD design but their applications are broader and include semi-custom and ASIC design, where gains like speed improvement or semiconductor area optimization can be obtained. These methods can be extended for asynchronous machines applying the approaches similiar to those from [Perk 79], [Perk 80], [Zaso 79] as well as microprogrammed synchronous, asynchronous and mixed synchronous/asynchronous machines as described in [Perk 70]. An important aspect is the attempt to build an integrated environment, where various methods of state table transformations can be applied and compared.

The algorithms are tested on the examples of industrial FSM's from the benchmarks of MCNC [MCNC 87]. These examples are in the *.kiss format so we are not able to understand their meaning. It is however more appropriate to understand the benchmark machine examples, which would permit to perform more interesting optimizations and partitions in comparison to the manual and the current automatic designs. The authors would be very obliged to obtain more examples of large Finite State Machines, especially those described with flow-diagrams and partitioned manually to EPLD's.

7. LITERATURE

[Copp 86] **Coppola, A.** : "Tools for Optimizing PLD Designs", Session 11, Record of Northcon 86, Seattle Sept.30 - Oct.2, 1986.

[DeMi 85] **De Micheli, G., Brayton, R., Sangiovanni-Vincentelli, A.L.** : "Optimal State Assignment for Finite State Machines", IEEE Transactions on Computer-Aided Design, Vol. CAD-4, No.3, pp.269-285, July 1985.

[Hama 84] **Hamachi, G.** : "Peg Tutorial", VLSI Tools, Univ. of California, Berkeley, 1984.

[Inte 86] **Intel,** : "User Defined Logic Handbook. EPLD Volume", 1986.

[Koha 70] **Kohavi, Z.** : "Switching and Finite Automata Theory", McGraw-Hill, New York, 1970.

[Lee 82] **Lee, E.B., Perkowski, M.** : "A New Approach to Structural Synthesis of Automata". University of Minnesota, Department of Electrical Engineering, report, 1982.

[Lee 84] **Lee, E.B., Perkowski, M.** : "Concurrent Minimization and State Assignment of Finite State Machines" . Proceedings of the 1984 Intern. Conf. on Systems, Man, and Cybernetics, IEEE, Halifax, Nova Scotia, Canada, October 9 - 12, 1984.

[MCNC 87] **Microelectronics Center of North Carolina** : "FSM benchmarks", International Workshop on Logic Synthesis", Research Triangle Park, North Carolina, May 12 - 15, 1987.

[Nguy 86] **Nguyen, L., Perkowski, M.** : "Boolean Minimization for PALs Using Graph Coloring on Personal Computer", Session 11, Record of Northcon 86, Seattle Sept.30 - Oct.2, 1986.

[Nguy 87] **Nguyen, L., Perkowski, M., Goldstein, N.B.** : "PALMINI - Fast Boolean Minimizer for Personal Computers", Proceedings of 24th Design Automation Conference, June 28 - July 1, 1987, Miami, Florida, Paper 33.3.

[Perk 70] **Perkowski, M.** : "Application of Logical Schemata of Algorithms to the Synthesis of Automata". Institute of Automatic Control, Technical University of Warsaw, 1970 (in Polish).

[Perk 79] **Perkowski, M., Zasowska, A.** : "Minimal Area MOS Asynchronous Automata". Proceedings of the International Symposium on Applied Aspects of Automata Theory, Warna, Bulgaria, 14-19 May 1979, pp. 284-298.

[Perk 80] **Perkowski, M.** : "The Method of Solving Combinatorial Problems in the Automatic Design of Digital Systems". Institute of Automatic Control, Technical University of Warsaw, 1980, (in Polish).

[Perk 85] **Perkowski, M.** : "Automatic Design of Finite State Machines". Seminar Electronics Laboratories, General Electric Company, Corporate Research and Development, Schenectady, N.Y. 12301, April 22, 1985.

[Perk 85a] **Perkowski, M., Nguyen, N.** : "Minimization of Finite State Machines in SuperPeg". Proceedings of the Midwest Symposium on Circuits and Systems. Luisville, Kentucky, 22-24 August 1985.

[Perk 86] **Perkowski, M., Smith, D., Krzywiec, R.** : "Logic Simulation/Design/Verification Environment in Prolog", Proceedings of the 17th Annual Pittsburgh Conference on Modeling and Simulation, April 24-25 1986, Univ. of Pittsburgh.

[Perk 86a] **Perkowski, M.** : "Digital Design Automation - Finite State Machine Design". Session 11, Record of Northcon 86, Seattle Sept.30 - Oct.2, 1986.

[Perk 86b] **Perkowski, M., Smith, D.** : "Intelligent User Interface for the DIADES Design Automation System", Session 11, Record of Northcon 86, Seattle Sept.30 - Oct.2, 1986.

[Perk 87] **Perkowski, M., Liu J.** : "A System for Fast Prototyping of Logic Design Programs", accepted to 1987 Midwest Conference on Circuits and Systems.

[Perk 87a] **Perkowski, M.** : "Application of Automatic Invariants Generation for Optimization of Control Units Realized as Finite State Machines", submitted to "INTEGRATION. The VLSI Journal". 1987.

[Perk 87b] **Perkowski, M., Uong, H.** : "Generalized Decompositions of Incompletely Specified Multi-Output Boolean Functions", PSU EE Dept. Report, 1987.

[Sasa 87] **Sasao, T.** : "Functional Decomposition of PLA's", Proceedings of the International Workshop on Logic Synthesis, Research Triangle Park, North Carolina, USA, May 12 - 15, 1987.

[Zaso 79] **Zasowska, A., Perkowski, M.** : "The Computer-oriented Method for Joint Minimization and State-assignment in Synchronous and Asynchronous Automata". Proceedings of the Conference "Application of digital computers in engineering design". Katowice 1979, pp. 31-42 (in Polish).