State assignment of finite state machines for multilevel logic implementations using counters

Pernot, T.A.P.

Award date:
1989

EINDHOVEN UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING
DESIGN AUTOMATION GROUP

STATE ASSIGNMENT OF FINITE STATE MACHINES FOR MULTILEVEL LOGIC IMPLEMENTATIONS USING COUNTERS

T.A.P. Pernot

Master thesis
reporting on graduation work
performed from 26.02.89 to 10.10.89
by order of prof. dr. ing. J.A.G. Jess
and supervised by dr. ir. J.F.M. Theeuwen

SUMMARY

In this paper we present a method to encode the states of synchronous finite state machines (FSM). This method is targeted towards multilevel combinatorial logic and loadable counter implementations. We assume that an optimal state assignment is a state assignment which yields minimum area in the final implementation. The algorithm we have constructed is a combination of two other methods that minimize the surface needed for the resulting lay-out of a finite state machine.

The first method [1] replaces a large number of state transitions by counting transitions. Therefore an additional output, called count, is generated by the combinatorial logic of the FSM, and the feedback register of the FSM is adapted to be able to perform these counting transitions. The state transitions, that can be replaced by counting transitions are gathered in a set of chains.

The second method [2] is a state assignment algorithm that heuristically maximizes the number of common cubes in the encoded network so as to minimize the number of literals in the resulting combinatorial network after multilevel logic optimization. This method consists of two algorithms. The fanout-oriented algorithm attempts to maximize the size of the most frequently occurring common cubes in the encoded machine prior to optimization. The fanin-oriented algorithm attempts to maximize the number of occurrences of the largest common cubes in the encoded machine prior to optimization. The state assignment algorithms find pairs of clusters of states which, if kept minimally distant in the Boolean space representing the code, result in a large number of common sub-expressions in the Boolean network.

We have run our program, called CHAINS, with a wide range of benchmark examples. CHAINS produces the best results for the largest examples. The results for the small examples are comparable with or worse than the results obtained using other state assignment programs.

CONTENTS

1. INTRODUCTION

A finite state machine (FSM) is a sequential circuit, that can be defined from the behavioral point of view as a circuit whose output depends not only on the present inputs, but also on the past history of inputs, or can be defined from the constructional point of view as a circuit which contains at least one memory element or unit. We can represent a FSM schematically as shown in Fig.1.

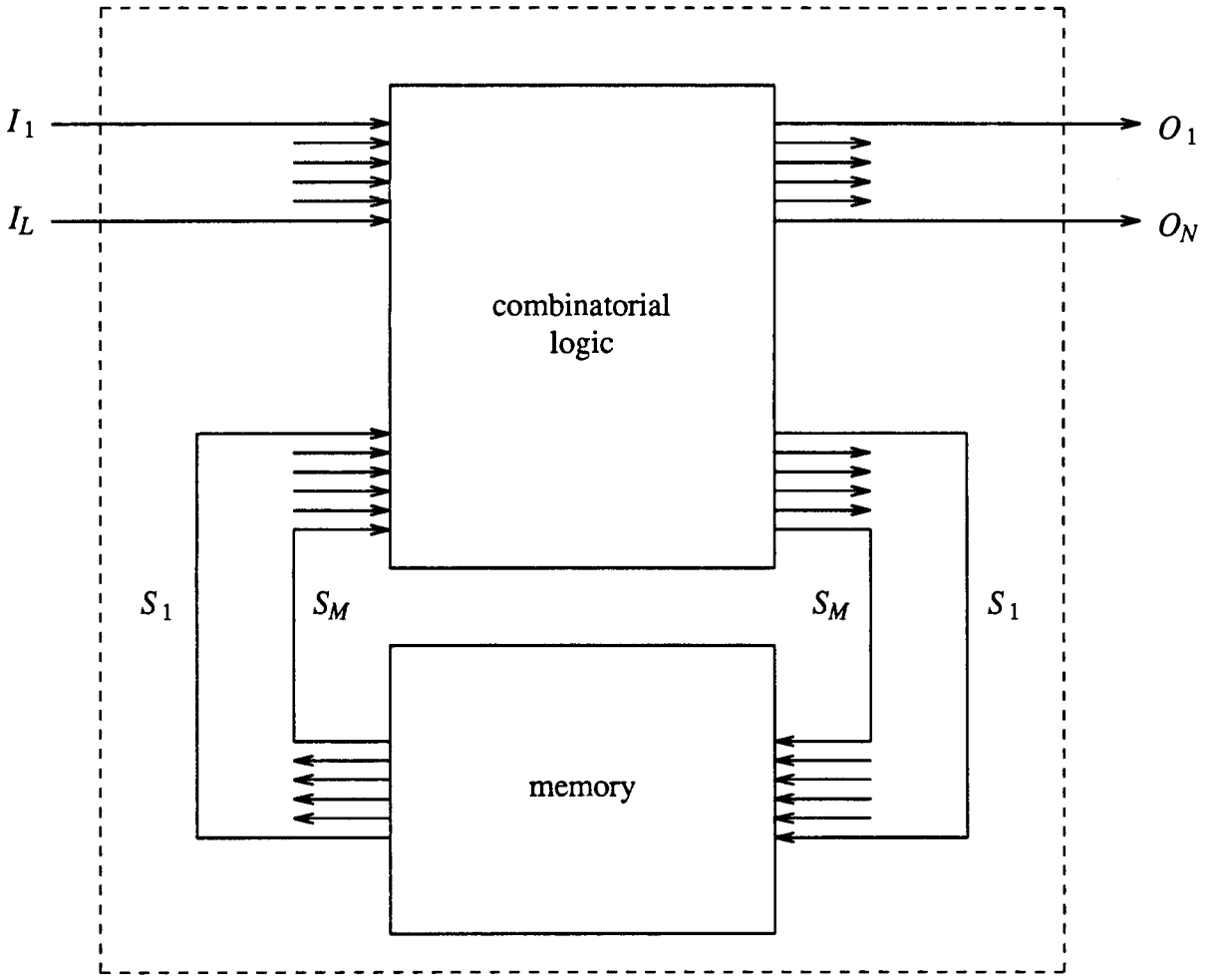

Figure 1. Schematic representation of a FSM.

The combinatorial logic is a $(L+M)$-input and $(N+M)$-output network, and the memory consists of M storage elements. From the behavioral point of view, we can consider the dotted line as a black box, which has L external inputs and N external outputs. This is why the state variables are also called internal variables.

In this paper we address the problem of encoding the states of synchronous finite state machines, targeted towards multilevel combinatorial logic and loadable counter implementations. We assume that an optimal state assignment is a state assignment which yields minimum area in the final implementation.

Most previous work in automatic FSM state assignments has been directed at the minimization of the number of product terms in a sum-of-products form of the combinatorial logic [3]-[6],[7]-[10] and hence, the results obtained are relevant for the cases where the combinatorial logic is implemented using programmable logic arrays (PLA's). In practice, most large FSM's cannot be synthesized as a single PLA for area and/or performance reasons. Multilevel logic implementations are generally used for smaller delays or smaller areas (or both). Results using manual state assignment have shown that existing automatic state assignment techniques are inadequate for producing optimal multilevel logic implementations [11].

In this paper we present a strategy for finding a state assignment of a FSM which heuristically minimizes the area used by a multilevel implementation of the combinatorial logic [2]. We also gather state transitions of the FSM in so called chains, to replace some state transitions by counting transitions [1]. This counting will be performed by a loadable counter that replaces the normal feedback register of the FSM.

In Section 2 some basic definitions and the formal definition of the problem are given. In Section 3 it is described how the set of chains is obtained, and in Section 4 it is described how the clusters are determined. A discussion about the length of the chains is found in Section 5. Then in Section 6 the actual state assignment is described. In Section 7 we work out an example, and in Section 8 we show results on the benchmark examples. Finally in Section 9 we present the conclusions.

2. *FORMAL DEFINITION OF THE PROBLEM*

In this section, we prepare some terminologies and give the formal definition of the problem.

2.1 Basic definitions

Definition 1: The finite state machine (FSM) is represented by a state transition table T(I,S,O), where:

1) $I = \{i_0, i_1, \ldots, i_l\}$, is the finite set of inputs,
2) $S = \{s_0, s_1, \ldots, s_m\}$ is the finite set of states, and
3) $O = \{o_0, o_1, \ldots, o_n\}$ is the finite set of outputs, of the FSM.
4) |T(I,S,O)| is the number of transitions, described in T(I,S,O).

Definition 2: Nb is the number of bits, used to encode the states of S.
$\log_2 (m+1) \leq Nb \leq \log_2 (m+1) + 1$.

Definition 3: $G_C(V, E_C, W(E_C))$ is a state transition graph. V is the set of vertices corresponding to the set of states S, |V| = |S|. $|E_C| \leq |T(I,S,O)|$. An edge $e = [s_i, s_j>$, $i \neq j$ and e in E_C, joins s_i to s_j if there exists at least one transition of the FSM from state s_i to state s_j. $W(E_C)$ is a set of labels attached to each edge, each label containing the weight of the edge. This edge-weight corresponds to the number of transitions from the start state s_i to the end state s_j of the edge. G_C is a directed graph with no self-loops and no parallel edges.

Definition 4: A chain *ch* is an ordered subset of E_C. The edges of *ch* form a path in the graph G_C. With |*ch*| we denote the number of states of *ch*. $|ch| \leq m + 1$.

Definition 5: $CH = \{ch_0, ch_1, \ldots, ch_k\}$, is a set of chains.

Definition 6: A cluster *cl* is a subset of S. The states contained in *cl* have to be coded minimally distant in the Boolean space representing the code. $|cl| \leq Nb$.

Definition 7: $CL = \{cl_0, cl_1, \ldots, cl_p\}$, $p \leq m$, is an ordered set of clusters. The label *j* of the cluster cl_j corresponds to the priority that is given to this cluster. If $i < j$, then the states of cl_i have to be coded before the states of cl_j.

Now we can make use of the definitions above to come to a formal definition of the state assignment problem.

2.2 Problem formulation

The state assignment problem consists of assigning a string of bits (code) to each of the states of S, so that no two states have an equivalent code. Furthermore the codes have to obey the chains of CH and the clusters of CL:

1) The chains of CH will be implemented as counting chains. This implies that the codes of an ordered set of states of a chain, also have to form an ordered set, according to the counting method that will be used.

2) The states of a cluster *cl* have to be coded minimally distant in the Boolean space representing the code. This means that the corresponding codes must have a minimum Hamming distance.

3. THE CHAINS

One way of minimizing the surface of the lay-out of a synchronous finite state machine, is replacing state transitions by counting transitions [1]. Therefore an additional output is introduced. If this output, called count, is "1", the new state is obtained from the oldstate by a counting transition. These counting transitions are gathered in the set CH. How these chains are derived is explained below.

3.1 Derivation of the graph $G_C(V,E_C,W(E_C))$

Before we can start with the actual derivation of the chains, we have to transform the state transition table STT, which we have stored in an array (see APPENDIX A), into a directed graph G_C. To determine the set of edges, E_C, we proceed as follows.

The state transition table is read line by line, ignoring the inputs and outputs. Every state transition (old state $\rightarrow$ new state) is checked with the already gathered edges. If there exists an edge corresponding to this transition, the edge weight of this edge is incremented by one. Else if there is no edge corresponding to this transition, a new edge, with an edge weight of one, is generated. Also, if the old state is equal to the new state of a state transition, this transition is skipped in the graph.

This leads to a directed graph $G_C(V,E_C,W(E_C))$, with no self-loops and no parallel edges, as defined in definition 3.

3.2 Definition of a legal arc

When chains are being built, the number of edges of E_C that can be used for further "chain-building" decreases. This is the result of two phenomena. The first and most logic one is that an edge that is placed in a chain can not be used again. The second one is the result of the increasing number of *illegal* edges.

Definition 8: A state is said to be *contained* if it is in the ordered set of a chain *ch* and is not the start or end state of this chain *ch*. The set of contained states is defined as follows:

$$CONT = \{s_i | s_i \text{ is contained in one of the chains}\}$$

Definition 9: An edge is called illegal if:

1) the start or end state of the edge is in the set CONT,
2) the start state of the edge is equal to the start state of one of the chains,
3) the end state of the edge is equal to the end state of one of the chains,
4) the start state of the edge is equal to the end state of a chain ch_i and the end state of the edge is equal to the start state of this chain ch_i.

Definition 10: An edge is called legal if it is not illegal.

3.3 Determination of the chains

The edges of the graph G_C are sorted on increasing edge weight [12]. Then the edges are connected into chains, beginning with the edges of the highest weight. Each edge is treated as follows.

We begin by checking whether the edge is legal or illegal. If the edge is illegal, it is skipped and the next edge is looked at. If the edge is legal, a possible connection to one of the chains is searched. If no suitable chain can be found to connect the edge, a new chain is added to the existing set of chains. This new chain is formed by the edge that was under consideration. Else if a chain is found to connect the edge, this chain is extended with this edge. Before we now look to the next edge, it is checked whether this extended chain can be connected to one of the other chains or not. If this connection is made or if no connection was possible, we will look to the next edge.

When all chains are determined, the state transition table is extended with an additional output, called count. The count output is set to one if the corresponding transition in the STT can be replaced by a counting transition. For all the remaining transitions the count output is set to zero.

3.4 Free states

If the set CH of chains with $|ch| > 1$ is determined, it is possible that CH doesn't cover the whole set of states, S. States that are not member of CH, are called free states. Because these states also have to be encoded, the set of chains is extended with a set of dummy chains. Such a dummy chain consists of a free state as start state and "-1" as end state of the chain.

4. THE CLUSTERS

The basic idea behind the clusters is to maximize the number and size of common cubes in the encoded network so as to minimize the number of literals in the resulting combinatorial logic network after multilevel optimization [2].

4.1 The graph $G_M(V,E_M,W(E_M))$

Definition 11: The state graph $G_M(V,E_M,W(E_M))$ is an undirected complete graph. V is the set of vertices corresponding to the set of states S, |V| = |S|. E_M is a complete set of edges. $W(E_M)$ represents the gains that can be achieved by coding the states joined by the corresponding arc as close as possible.

There ara determined two graphs of the type G_M from the STT (see APPENDIX A), using the algorithms presented in [2]. A critical part of this approach is the generation of $W(E_M)$. We have two algorithms: one assigns the weights to the edges by taking into consideration the old states and outputs of the STT, and henceforth is called *fanout-oriented.* The second algorithm assigns weights to the edges by taking into consideration the inputs and new states of the STT and is henceforth called *fanin-oriented.* So there will be determined two graphs of the type G_M; one using the fanout-oriented algorithm and one using the fanin-oriented algorithm.

The fanout-oriented algorithm attempts to maximize the size of the most frequently occurring common cubes in the encoded machine prior to optimization. The fanin-oriented algorithm attempts to maximize the number of occurrences of the largest common cubes in the encoded machine prior to optimization.

4.2 Determination of the clusters

Although in the actual program two sets of clusters are determined, one with the fanout-oriented graph and one with the fanin-oriented graph, we will explain how, in general, the set of clusters is determined from a graph G_M.

The algorithm proceeds as follows. Clusters of states with the cardinality of the cluster no greater than N_b+1 and consisting of edges of maximum weight are identified in G_M. Given G_M, the identification of these clusters is as follows.

A state s_i element of G_M, with the maximum sum of weights of any N_b connected edges is identified. The N_b states, $y_1, y_2, \ldots, y_{N_b}$ which correspond to the N_b edges from s_i are placed in the cluster cl_0, together with s_i. It is the intention to encode all the states of the cluster as close as possible to this "*maximum*" state s_i. After this, s_i and all the edges connected to s_i are deleted from G_M and the determination of the clusters cl_1, cl_2, etc is repeated till all the nodes are placed in some cluster.

5. *HOW TO COMBINE THE CLUSTERS AND CHAINS ?*

The algorithm we have constructed is a combination of two methods [1] and [2], that minimize the surface needed for the lay-out of a FSM. The first method is used to determine the set of chains, CH, and the second method is used to determine the ordered set of clusters, CL. Now it is the question how we can combine these two methods to come to a satisfactory new program.

We can take the set CH as starting point and adapt the set CL. This results in a set of clusters of chains. In case of a binary counter we can place two chains of a *chain–cluster* next to each other so the states of these two chains will get a code with the same most significant bits. As can be seen easily, when the chains of CH are long, the condition of the clusters of CL can not be met satisfactory. Therefore we have chosen for an other approach of this problem.

We start by determining chains of states with a maximal cardinality $|ch| = 2$, so consisting of only one edge of E_C. If $|ch| = 1$, the single state of this chain is a free state. When state s_i of a chain, is assigned a code, the other state, s_j,forming the "chain-partner" of s_i will immediately get a code according to counting sequence. Because the chains have a maximal length of two, the condition of coding states of a cluster *cl* minimally distant in the Boolean space, can now be met much more satisfactory.

Now it is the intention to form clusters of states, which have as few as possible internal chain connections, or indeed have no internal chain connections as shown in Fig.2.

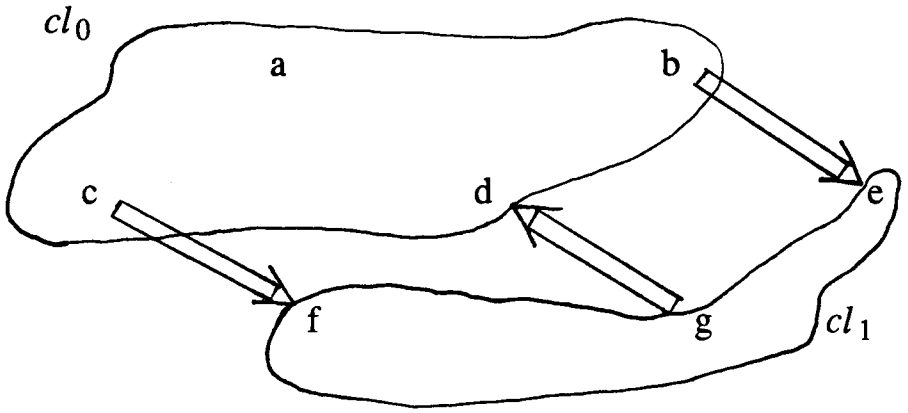

Figure 2. Ideal set of chains and clusters.

This means that there may only be chain connections between two states of a different cluster. The ideal situation shown in Fig.2. (chains are represented by arrows) can almost never be obtained, because the determination of the clusters is a rather rigid process. However, we can still influence this process. By deleting the chain transitions from the state transition table, and using the resulting STT for the determination of the clusters, we are able to get more satisfactory clusters, with respect to the chains.

6. *ENCODING OF THE STATES*

6.1 Coding with minimum number of bits

The encoding that is realized is based on the use of a binary counter. This binary counter generates strings of bits which have a least significant bit that is alternately equal to "0" or "1". Now it is the intention to assign codes to the states s_i and s_j of a chain *ch* in such a manner, that the start state of *ch*, s_i, is assigned a code with a least significant bit of "0", and the end state of *ch*, s_j, is assigned the succeeding code with least significant bit of "1". Doing this results in three advantages. The first advantage is, that the codes of the states of a chain always have the minimum Hamming distance of one bit. The second advantage is that we are able to encode all the states of the FSM with a minimum number of bits. This will be explained below.

Code table ($N_b = 3$)		
	fault	good
code	mark	mark
000	USED	USED
001	USED	USED
010	FREE	FREE
011	USED	FREE
100	USED	USED
101	FREE	USED
110	USED	FREE
111	USED	FREE

TABLE 1. Marking of the code table.

Every chain demands two successive codes of the code table (= the sequence of all codes of N_b bits generated by the binary counter). If the condition mentioned above is not satisfied, it is possible that the code table on a certain moment will look like pictured in the column *fault* of Table 1. A sequense of two used codes, one free code, two used codes, etc. Because a chain demands two successive codes, all the "gaps" of one free code can never be assigned. In the worst case this will result in 33% of the codes that can not be used. However when we meet with the condition, there will always be two successive free codes between the used codes, as can be seen in the column *good* of the table.

The third advantage is that we don't have to use a whole loadable counter to replace the feedback register of the FSM. It suffices to adapt the feedback register. This adaptation consists of inverting the least significant bit of the register, when the count signal is equal to "1". The other bits remain unchanged.

6.2 Assignment of the codes

Now that we have gathered the boundary conditions for the state assignment, we can start with the actual encoding of the states.

The code table is loaded with the sequence of codes of N_b bits. Every code is marked FREE, and each state is marked UNCODED. Then starting with cluster cl_0 (see definition 7) the algorithm proceeds as follows.

If s_i, the maximum state of cl_j, is UNCODED, the chain corresponding to state s_i is traced. If s_i is the start state of the chain, s_i is assigned a FREE code with a least significant bit equal to "0", and the corresponding end state of this chain is assigned the succeeding code, which will be FREE also, according to the previous paragraph. If s_i is the end state of the chain, s_i is assigned a FREE code with a least significant bit equal to "1", and the corresponding start state is assigned the preceding code. After s_i and his "chain-partner" are coded or if they were already coded before, we can proceed with coding the other states of cluster cl_j. The encoding of these states takes place a in similar way, with as only acception that a code is searched which has a minimum Hamming distance to the maximum state s_i. Whenever a code is assigned, the corresponding place in the code table is marked USED. If all states of cluster cl_j are coded, cluster cl_{j+1} is treated the same way.

6.3 An exception: the free states

Before a state is assigned a code, it is also checked wether this state is a free state. The chain corresponding to the state is traced and if the "end state" is equal to "-1", this state is a free state. For a free state of cl_j the encoding takes place in a slight different way. The code table is traced for a FREE code (with minimum Hamming distance from s_i) and if a FREE code is found, it is assigned to this free state. So there is no tracing of a FREE code with a specific least significant bit. In some cases this will result in a necessary extra code bit.

Because all the states of the real chains ($|ch| = 2$) are coded two by two, a code assignment to a free state ($|ch| = 1$) will always result in a gap in the code table. If the free state is assigned a code with a least significant bit equal to "0", the succeeding code can never be used to encode one of the states of a real chain. Likewise, if the free state is assigned a code with a least significant bit equal to "1", the preceding code can never be used to encode one of the states of a chain. This forces us to expand the number of bits with an additional code bit, if:

2 * (no_of_chains + no_of_free_states) > no_of_codes;

no_of_chains = number of chains with $|ch| = 2$,
no_of_free_states = number of free states,
no_of_codes = number of different codes with N_b bits.

7. *AN EXAMPLE*

In this section we will show how the program proceeds step by step. We take as input the STT of table 2.

Example STT				
inputs	old state	new state	outputs	count
0011	a	a	00	0
0001	b	a	10	0
0010	c	b	00	0
0011	b	c	11	0
0011	c	d	01	0
1000	c	f	10	1
0100	c	f	01	1
0100	b	e	11	1
1010	b	e	00	1
0111	d	g	10	1
1011	d	g	01	1
1100	e	g	01	0
1101	f	g	11	0
1101	g	e	10	0
1101	e	f	11	0

TABLE 2. Example STT

7.1 The chains

We first have to construct the graph G_C according to paragraph 3.1. This results in the graph shown in Fig. 3.

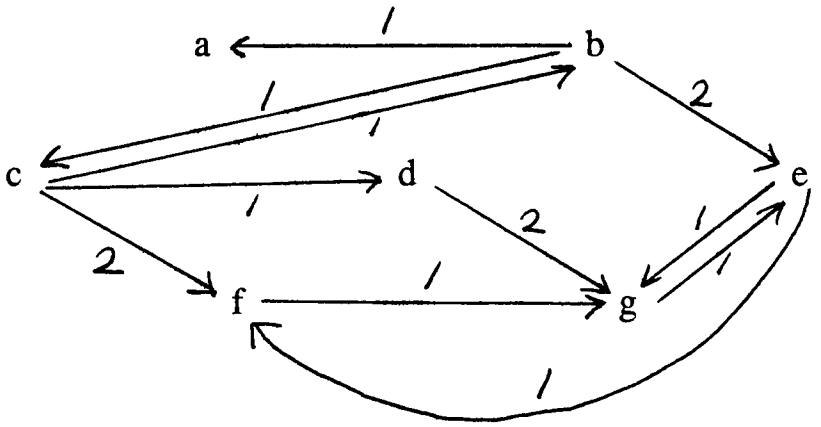

Figure 3. The graph G_C.

From this graph we determine the chains and dummy chains, according to paragraph 3.3 and 3.4. In Table 3 all the chains and the single dummy chain are shown (dummy chain has "-1" as end state !).

Chains		
	start state	end state
chain1	b	e
chain2	d	g
chain3	c	f
chain4	a	-1

TABLE 3. The chains

We use these chains to modify the STT. An extra outputsymbol, called count, is introduced. For every state transition that corresponds to one of the chain transitions, this additional output is set to "1". For the other state transitions the count variable is set to "0" (see fifth column of Table 2).

7.2 The clusters

In table 4 we have gathered all the edge weights for G_M, using the fanin-oriented algorithm mentioned in [2]. From this representation of the total graph we derive the clusters, shown in table 5.

Edge weights for fanin-oriented algorithm	
edge	edge weight
[a,b]	5
[a,c]	10
[a,d]	7
[a,f]	3
[a,e]	3
[a,g]	4
[b,c]	3
[b,d]	6
[b,f]	0
[b,e]	0
[b,g]	1
[c,d]	4
[c,f]	1
[c,e]	1
[c,g]	1
[d,f]	1
[d,e]	1
[d,g]	1
[f,e]	4
[f,g]	10
[e,g]	7

TABLE 4. Edge weights for fanin-oriented algorithm

Clusters				
	maxstate			
cluster1	a	c	d	b
cluster2	g	f	e	b

TABLE 5. The clusters for fanin-oriented algorithm

7.3 State assignment

We first check how many bits we need.

nr_of_codes = 8.
2 * (nr_of_chains + nr_of_free_states) = 8 = nr_of_codes.

So we can encode with the minimum number of bits (N_b = 3).

We start with the cluster with the lowest label; cluster0. The state with maximum sum of weights is state *a*. We trace the corresponding chain, this is chain4. Chain4 is a dummy chain (end state = "-1"), so we assign state *a* the first code of the code table, that is marked FREE, which is "000". Then we take state *c* and trace the chain: chain3. State *c* is the start state of this chain, so we search in the code table for a FREE code with a least significant bit equal to "0" and with a minimum Hamming ditance to the code of the maximum state *a*. State *c* is assigned the code "010" and the end state of chain3 , state *f*, is assigned code "011". How the code table is marked during the state assignment process can be seen in Table 6 (step 1,step 2,etc.)

Code assignment					
code		step 1	step 2	step 3	step 4
000	FREE	a	a	a	a
001	FREE	FREE	FREE	FREE	FREE
010	FREE	FREE	c	c	c
011	FREE	FREE	f	f	f
100	FREE	FREE	FREE	d	d
101	FREE	FREE	FREE	g	g
110	FREE	FREE	FREE	FREE	b
111	FREE	FREE	FREE	FREE	e

TABLE 6. Assignment of the codes, step by step

8. RESULTS

The general structure of the algorithm is given in figure 3.

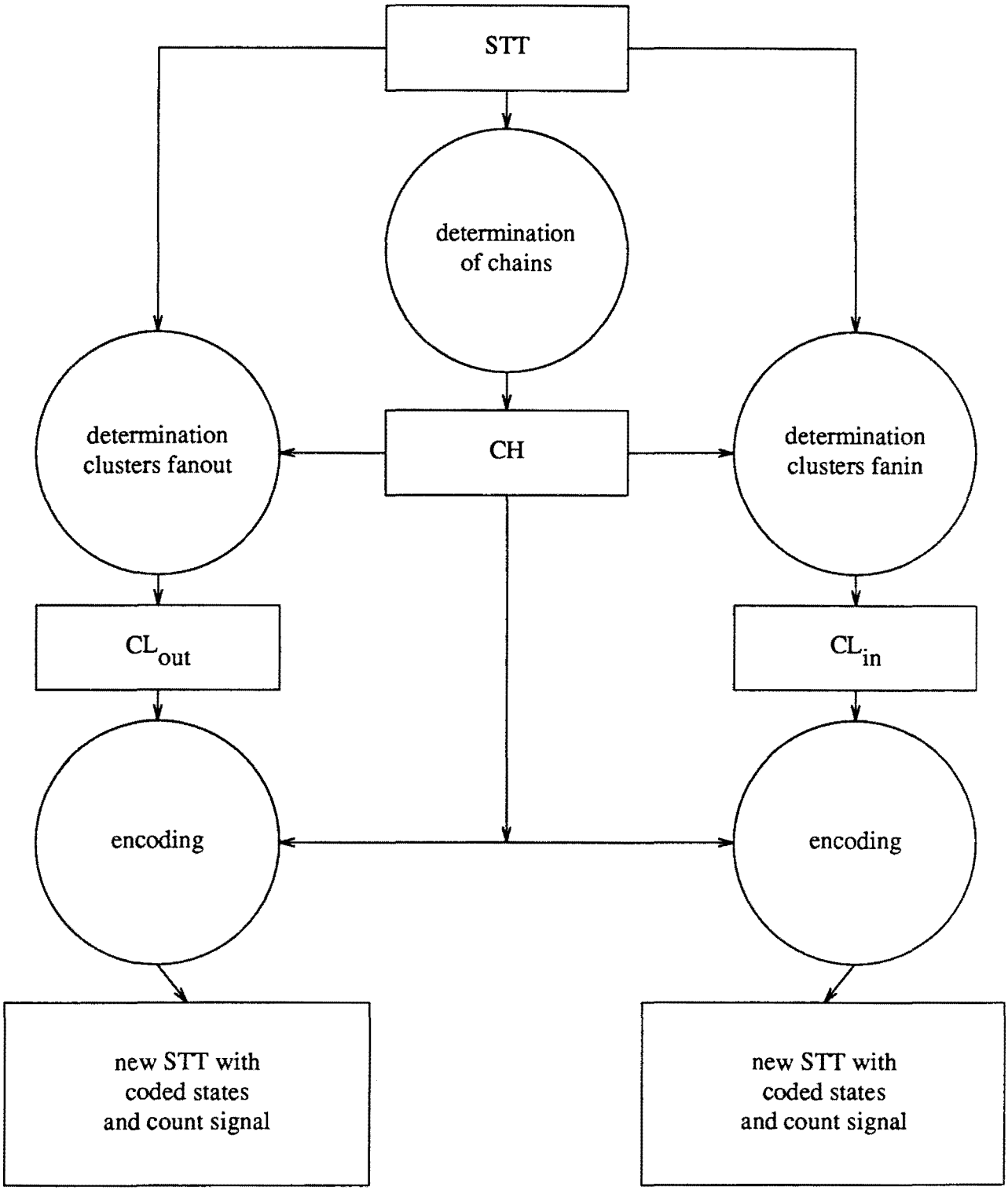

Figure 4. General structure of the algorithm.

As can be seen in the figure above the algorithm we have constructed produces two outputfiles, one fanin-oriented and one fanout-oriented.

We have run 19 benchmark examples representing a wide range of finite automata on different state assignment programs as well as on our own two algorithms. The size statistics of the examples are given in Table 7, with the minimum possible number of bits for each FSM indicated under the column #enc.

Statistics of benchmark examples				
Example	#inp	#out	#states	#enc
bbara	4	2	10	4
bbsse	7	7	16	4
bbtas	2	2	6	3
cse	7	7	16	4
dk15x	3	5	4	2
dk16x	2	3	27	5
keyb	2	1	19	5
lion	2	1	4	2
lion9	2	1	9	4
mark1	5	16	14	4
mc	3	5	4	2
modulo12	1	1	11	4
planet	7	19	48	6
s1	8	6	20	5
s1a	8	6	20	5
scf	27	56	128	7
shiftreg	1	1	8	3
tav	4	4	4	2
train11	2	1	10	4

TABLE 7. Statistics of benchmark examples.

The results obtained via random state assignment, using the state assignment program KISS, and the best result produced by either the fanout or the fanin-oriented algorithm of MUSTANG and our program CHAINS, are summarized in Table 8. The transistorcounts under RANDOM-A were obtained using a statistical average of five different random state assignments (using different starting seeds) on each example. RANDOM-B was the best result obtained in these different runs.

KISS typically uses a 1-3 bits more than the minimum encoding length. MUSTANG was run using the minimum possible bit encoding. Our program, called CHAINS, used a minimum number of bits for the encoding for all the examples, except for the examples *bbsse*, *cse*, *lion*, *mark1* and *scf*, for which it used one extra bit.

The program developed, CHAINS, produces the best results for the largest examples, as can be seen in the tables. The results for the small examples are comparable with or worse than the results obtained using the other state assignment programs. The time required by CHAINS for encoding these benchmarks varied between 0.1 CPU seconds for the small examples to 27 CPU seconds for the largest example, *scf*, on a HP 9000 series 800.

Results of other assignment methods and CHAINS					
Example	RANDOM-A	RANDOM-B	KISS	MUSTANG	CHAINS
bbara	120	91	103	81	115
bbsse	214	190	145	144	197
bbtas	37	26	34	32	32
cse	405	339	264	304	323
dk15x	122	109	91	104	95
dk16x	553	516	411	346	405
keyb	810	663	474	330	324
lion	20	18	21	18	32
lion9	61	52	37	20	68
mark1	112	89	114	87	124
mc	40	37	43	36	25
modulo12	43	40	49	36	44
planet	1063	1012	869	854	779
s1	852	805	690	200	461
s1a	649	583	382	162	315
scf	1674	1596	1441	1274	1158
shiftreg	37	32	8	2	31
tav	25	24	24	24	25
train11	67	53	46	50	89

TABLE 8. Resulting transistorcounts

9. CONCLUSIONS AND RECOMMENDATIONS

Although we took as a starting point an implementation of the FSM with a loadable binary counter as replacement for the feedback registers. it suffices to make only a small adjustment to the feedback register. This small adjustment consists of inverting the least significant bit of the register, when there is a count signal equal to "1".

It is also easy to adapt CHAINS to assign codes with more than the minimum number of bits.

As mentioned in Section 5, we have choosen for a maximum chain lenght of two. It must be possible to extend the program with a mechanism that links two or maybe more chains together, during the encoding. This "linking" can proceed as follows.

During the determination of the chains the number of legal edges decreases, but after all possible chains are formed, there will almost always remain several legal edges. These remaining legal edges can be seen as links between chains. We can use these links in the encoding part of the program. If we are searching for a FREE code in the code table, we can check if there exists a FREE code for a state which satisfies an additional condition. To explain this we will look to the encoding as if we were encoding chains. Suppose there exists a link between chain 1 and chain 2. Chain 1 is encoded already. If we trace the table for an encoding of chain 2, we now look first to the (two) code(s) of chain 1. If the code(s) next to chain 1 also satisfies the conditions stated in Section 6.2, this code will be assigned to chain 2. The transition corresponding to the made link, will be implemented as a new counting transition.

REFERENCES

[1] R.Amann, U.G. Baitinger,
"New state assignment algorithms for finite state machines using counters and multiple-PLA/ROM structures",
IEEE ICCAD-87.
Digest of Technical Papers, Santa Clara, CA, USA.
(IEEE Comput. Soc. Press 1987), pp. 20-23.

[2] Srinivas Devadas, Hi-Keung Ma, A. Richard Newton, A. Sangiovanni-Vincentelli,
"MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations",
IEEE Trans. on CAD, Vol. 7, No. 12(1988) pp. 1290-1299.

[3] D.B. Armstrong,
"A programmed algorithm for assigning internal codes to sequential machines",
IRE Trans. Electron. Comput., vol. EC-11, pp. 466-472, Aug. 1962.

[4] T.A. Dolotta and E.J. Mccluskey,
"The coding of internal states of Sequential Machines",
IEEE Trans. Electron. Comput., vol. EC-13, pp. 549-562, Oct 1964.

[5] H.C. Torng,
"An algorithlgorithm for finding secondary assignments of synchronous sequential circuits",
IEEE Trans. Computers, vol. C-17, pp.416-469, May 1968.

[6] G.D. Micheli, A. Sangiovanni-Vincentelli, and T. Villa,
"Computer-aided synthesis of PLA-based finite state machines",
in Proc. Int. Conf. on Computer-aided Design, Santa Clara, CA, November 1983, 154-156.

[7] G.D. Micheli, R.K. Brayton, and A. Sangiovanni-Vincentelli,
"Optimal state assignment of finite state machines",
IEEE Trans. Computer-Aided Design, vol. CAD-4, pp. 269-285, July 1985.

[8] A.J. Coppola,
"An implementation of a state assignment heuristic",
in Proc. 23rd Design Automation Conf.,Las Vegas, NV, July 1986.

[9] G.D. Micheli,
"Symbolic design of combinational and sequential logic circuits implemented by two-level macros",
IEEE Trans. Computer-Aided Design, vol. CAD-5, pp. 597-616, Oct. 1986.

[10] G. Saucier, M.C. Depaulet, and P. Sicard,
"ASYL: A rule-based system for controller synthesis",
IEEE Trans. Computer-Aided Design, vol. CAD-6, pp. 1088-1098, Nov. 1987.

[11] C. Tseng et al.,
"A versatile finite state machine synthesizer",
in Proc. Int. Conf. on Computer-Aided Design, Santa Clara, CA,pp.206-209, Nov. 1986.

[12] Prof.dr.ing. J.A.G. Jess,
"Informatica voor E-II",
Collegedictaat No.5610 (1987),pp. 1.28-1.36.

APPENDIX A

READING THE STATE TRANSITION TABLE

For reading the input we make use of the lexical analyser LEX. The input of the program consists of a file in espresso-format.

We start by reading the constants, which are used to read the actual state transition table. The constants of interest are the number of inputs and outputs, the number of states and the number of transition rules. When these constants are known, each transition rule is read and stored in an array.

A transition rule is built up out of four blocks: the inputs, the old state, the new state and the outputs. The inputs and outputs are copied as strings. The different state names are assigned positive integers and in a table the state name is written on a location corresponding to his integer representation. In the array of transition rules the integer representations of the old state and new state is stored. This is done, because integers are easier to manipulate than characterstrings. When a don't care state (ANY) is read in a transition rule, all the corresponding transitions to or from the other state can be generated, simply by generating transitions to or from this other state to all the integer representations of states. Even if we don't know all the state names, we can make transitions to or from them, because we know from the constants how many states there are.

When the total inputfile has been read, we have an array of transition rules, consisting of inputs, integer representations of the old and new state, and outputs. We also have a table, we can use to look up the real state name that belongs to an integer representation.