Microprocessing and Microprogramming 35 (1992) 151-158
North-Holland

Estimators for Logic Minimization and Implementation Selection of Finite State Machines

A.J.W.M. ten Berg

University of Twente
Faculty of Computer Science
P.O.Box 217, NL 7500 AE Enschede, The Netherlands

This paper considers two estimation problems which occur during the implementation design for a finite state machine (FSM). The first is a precise estimation of the reduction of a programmed logic array implementation (PLA) for a FSM by logic minimization. The second concerns selection of implementation alternatives based on such estimations. Estimations give the designer a quick overview of the impact of an optimization method for FSM implementation without running the actual time-consuming algorithms. The method uses curve-fitting on results found in literature for logic minimization preceded by state-assignment. Our estimations correlate by 0.97 to those results.State-graph statistics can also be used for selection of the most profitable optimization from a set of alternatives. We tested selection between a counter based implementation, partial state coding, state-assignment and topological partitioning. The goal is selection of the alternative which has the highest probability to deliver the largest minimization of the FSM. This selection method is also empirically verified by comparing its results with results obtained by running specific optimization algorithms on machines of the MCNC benchmark set.

1 Introduction

A considerable problem in synthesis is the selection of the minimal implementation alternative for Finite State Machines. Many implementation alternatives are known [2][5] for specific hardware configurations. Each of these implementations has its specific optimization or minimization algorithms [2],[3],[12]. Thus, for a proper comparison of alternatives, the designer must execute all optimization algorithms. Unfortunately, this takes large computational efforts. Equal problems arise also in design planning [15], were design plans must be build. Quick estimations of optimizations can be of great help in such situations. Estimations improve the efficiency of the design process. One can identify two types of estimations for evaluation of design alternatives.

The first estimation type is the prediction of results of complex and therefore time-consuming optimizations. Such optimizations are usually bound to one implementation. An important example of such an optimization is the two-level logic minimization problem for Finite State Machines (FSM) implemented in a single PLA. However, only if the state-assignment problem [3] , [4] is solved, the minimal PLA is obtained. The state-assignment assigns binary codes to internal FSM states such that a two-level or a multi-level logic optimization has its maximal effect. In this case we consider only two-level logic ie. PLA implementations. The exact amount of minimization is not the most interesting to know. In most cases the relative minimization potential is far more useful for the designer. In other words, whether application of logic minimization with state-assignment will reduce the machine size by 20%, 50% or maybe even 70%.

The second estimation type concerns the selection problem among implementation alternatives. For selection, not the accuracy of estimation for each individual alternative is important, but its selectivity. Therefore, such estimations must have a reasonable correlation with the results of their optimization algorithm and preferably a low correlation with the results of other optimizations. Selection estimations deliver the alternative with the highest probability of obtaining the minimal implementation.

In order to test the estimators, we ran optimization algorithms for each alternative on the MCNC benchmark set of FSM's. This benchmark set is used widely for the comparison of state-assignment algorithms. The correlation coefficient between estimations and algorithm results provides us an empirical verification. The set of alternatives contains a counter, partial state code generation, state-assignment and topological partitioning. Section 2 discusses the curve-fit estimation method for

state-assignment. Section 3 explains the optimizations for each alternative and also the statistical measures used for the selection.

2 State-assignment result estimation

What we try to estimate is the maximal reduction of a PLA implementation of a FSM by logic minimization. This problem includes the state-assignment problem, which must be solved optimal in order to obtain the maximal profit out of logic minimization. The state-assignment problem is very complex and as problem hard to estimate. The FSM structure however is much simpler and therefore we can estimate the maximal profit of logic minimization for it.
In other engineering sciences as for example in chemistry, curve-fitting is usually applied to model complex physical processes. In chemistry, the behaviour of reactors is modelled by volume, pressure and temperature parameters, which are very global compared to the actual molecular reactions. Then, curve-fit methods can supply understanding and prediction of the macro-molecular behaviour of the complete reactor. Therefore, we investigate a curve-fitting method for the estimation of two level logic minimization. Analogue to the example we have to identify global FSM parameters as input for the estimation polynomial.

The curve-fit method delivers the coefficients C for a polynomial in the form of (1). The input data for the curve-fit are the results from NOVA [9]. NOVA performs near-optimal state-assignments for PLA implementations, and it obtains its minimization results with ESPRESSO [11], which is a widely known two level logic minimizer. The number of coefficients of the polynomial minus one is the order of the polynomial. We applied mathematical APL programs for regression analysis [1] to compute the coefficients C of the polynomial curve. These programs apply a least square's method for this computation.

Polynomial of order 4 :

$$POLY(x) = C1.x^{**}4 + C2.x^{**}3 + C3.x^{**}2 + C4.x + C5 \qquad (1)$$

The first problem is to determine the parameter(s) x of the FSM that give a curve close to the NOVA/ESPRESSO results. The second problem is the order of the polynomial. This order must be small compared to the set of data from which the polynomial is derived. NOVA [9] provides results for 25 machines from the MCNC benchmark set (table 1). For a proper curve, the polynomial order must be small compared to the number of samples. Therefore we use a polynomial of a third order, which is small enough.

2.1 Finite State Machine parameters.

Next, we identify some curve parameters, for which we need first some FSM definitions. Cardinality of a set of elements is denoted by #. For example: #I is the number of symbols in the input alphabet. A binary coded symbol is called a word. A Finite State Machine is defined by the quintuplet FSM = <I, O, S, OUT, TRS > given by :

$I = \{i_1..i_k\}$	Input alphabet where i_x is a binary coded symbol
$O = \{o_1..o_m\}$	Output alphabet where o_x is a binary coded symbol
$S = \{s_1..s_v\}$	State alphabet, where s_x is symbolic.
$o := OUT(s, i)$	Mealy Output function
$o := OUT(s)$	Moore Output function
$s' := TRS(s, i)$	Transition function

A second specification of a FSM is given by a set PTS={$pt_1..pt_n$} of four tuples <input word, state, next state, output word> denoted as pt=<i,s,s',o> which are called productterms. The FSM's in the benchmark set are all of the Mealy type. Now we can write the estimation of the minimized number of productterms $\#PTS_{min}$ in (2).

$$\#PTS_{min} = POLY(x) * \#PTS$$

We select various parameters, and combinations of them, for the x in (2) and tested their quality. First we compute the polynomial coefficients. Then, the estimations of $\#PTS_{min}$ are compared with the NOVA/ESPRESSO [9] results. This comparison includes the correlation coefficient and the average difference between the algorithm minimization and the estimated minimization $\#PTS_{min}$. Examination of the FSM structure in relation with the NOVA [9] results shows two important FSM parameters. The first parameter is the number of productterms per state #PTS/#S. One expects that for logic minimization, a high number of productterms for a state causes a relatively high minimization potential. Because, if more productterms share the same state code then larger cubes can be expected. A second parameter is the number of output variables #O. Roughly, the higher this number, the more difficult it becomes to combine productterms. Thus the number of output variables contributes in a reverse way to minimization. Therefore we use this parameter inverted (1/#O) when we combine it with other parameters as #PTS/#S. Furthermore, the number of input variables #I may be interesting. Although #I is mostly related to #PTS/#S and therefore only a limited effect can be expected from its use.

A combination of more FSM parameters in x requires a weighting mechanism. This to make sure that value ranges of parameters become overlapping. Thus, we presume an equal importance of both parameters. The value range of the #PTS/#S parameter is roughly from 1 to 10. The 1/#O parameter is always between 0 and 1. Therefore this last parameter is multiplied with a factor 10. Another alternative would be the use of complex curve fitting, which can obtain a polynomial for sets of parameters. However, the results of parameter balancing are quite good in this case, which makes complex curve-fitting superfluous here.

	#I	#O	#PTS	#S
bbara	4	2	60	10
bbsse	7	7	56	16
bbtas	2	2	24	6
beect	3	4	28	7
cse	7	7	91	16
dk14	3	5	56	7
dk15	3	5	32	4
dk16	2	3	108	27
dk17	2	3	32	8
dk27	1	2	14	7
dk512	1	3	30	15
donfl	2	1	96	24
ex1	9	19	138	20
ex2	2	2	72	19
ex3	2	2	36	10
ex5	2	2	32	9
ex6	5	8	34	8
keyb	7	2	170	19
plan	7	19	115	48
s1	8	6	107	20
sand	11	9	184	32
scf	27	56	166	121
shift	1	1	16	8
styr	9	10	166	30
tra11	2	1	25	11

Table 1. Data of MCNC FSM set.

Table 1 gives the MCNC FSM data and Table 2 shows a comparison of FSM parameters and some combinations of them. Clearly x=(10/#O) + (#PTS/#S) delivers the best result with a correlation coefficient of 0.97 with the NOVA results and an average difference of 6.1% in $\#PTS_{min}$

parameter : x	Corr.Coeff.	Av.Diff.%
#PTS/#S	0.94	10.3
#O	0.94	9.5
(10/#O)+(#PTS/#S)	0.97	6.1
#I+(10/#O)+(#PTS/#S)	0.92	11.2
#I	0.92	11.2
#I+(10/#O)	0.93	10.4
#I+(#PTS/#S)	0.92	12.1

Table 2. Estimations for several FSM parameters compared.

compared with the NOVA results. Then, Table 3 gives the coefficients of the polynomial for x=(10/#O)+(#PTS/#S). Table 4 shows in the second column the productterms after minimization $\#PTS_{min}$, computed with the polynomial. The first column shows the number of productterms obtained by NOVA/ESPRESSO. Table 5 gives the percentile differences between estimation and algorithm, related to the original #PTS of the machines.

coefficient	order
-3.23 E -4	x**3
9.60 E -3	x**2
-0.13	x**1
1.03	1

Table 3. Coefficients of estimation polynomial.

These tables show the remarkable precision of the estimations. Certainly, because the estimations are based on only two parameters of the FSM and a third order polynomial. This gives sufficient confidence in the usefulness of this type of estimations for prediction of the impact of logic minimization including state-assignment. The two largest exceptions are machines s1 and ex1. We analysed these machines into further detail but did not identify a clear cause. Further research concerns estimations for other optimization algorithms used in

	#PTS	
	NOVA	estim.
bbara	24	23
bbsse	27	34
bbtas	8	11
beect	11	15
cse	44	45
dk14	22	23
dk15	16	13
dk16	49	53
dk17	17	16
dk27	8	7
dk512	17	18
donfl	23	25
ex1	40	67
ex2	29	32
ex3	17	17
ex5	15	15
ex6	23	20
keyb	47	44
plan	87	86
s1	80	54
sand	89	93
scf	138	140
shift	4	6
styr	89	86
tra11	9	8

Table 4. NOVA results with estimations.

	%Diff.
ex5	0.0
mark1	0.0
tra11	0.0
styr	1.2
bbara	1.6
keyb	1.7
dk14	-1.8
donfl	-2.1
sand	-2.2
cse	-2.2
scf	-2.5
plan	2.6
ex3	2.7
dk17	3.1
dk512	-3.4
dk16	-3.8
ex2	-5.6
dk27	7.1
ex6	8.8
dk15	9.3
bbsse	-12.5
bbtas	-12.5
shift	-12.5
beect	-14.3
ex1	-19.6
s1	24.2

Table 5. Percentile differences.

microarchitecture synthesis. The next section discusses a different type of estimations that can select a microarchitecture from several alternatives. This requires a selective character instead of a high precision.

3 Implementation alternative selection by estimations.

The second type of estimation problem is the selection among implementation alternatives. In most cases, the designer has to choose from a set of alternative implementations. Unfortunately, each implementation usually requires its own specific optimization. Therefore a proper comparison between alternatives requires execution of all the optimization algorithms. Estimations can provide the designer quickly with information on the alternatives. Then, the designer can deduct which of the alternatives are most promising for a more detailed comparison. We implemented a set of implementation alternatives and computed results of their optimization algorithms for the machines in the MCNC benchmark set. Next is the FSM analysis which is the base for the estimations. The problem is to identify those FSM parameters that have a selective character. Therefore, these parameters must provide a high correlation between the estimation and its optimization algorithm results and a low correlation with the other alternatives.

We focus mainly on the transition function in the FSM, because quite a number of different alternatives is available to implement it. First we explain each of the alternatives shortly with theirs related optimization problem. Some optimization algorithms originate from literature and some where necessarily self-developed. The alternatives set contains a counter implementation, partial state coding, state assignment and topological partitioning. These are all quite common to FSM's and not restricted to very specific types of FSM's as other more dedicated optimizations [7],[8] are.

3.1 A counter implementation.

Algorithms for optimal embedding of a counter in the transition function of a FSM are studied extensively in [2]. The counter implements the countable transitions. This makes these transitions redundant in the transition PLA and therefore removable. The PLA generates the signal for counter activation by default in case an input pattern does not match any of its productterms. The counter must be loadable to accept codes generated by the PLA in case of non-countable transitions. The counter connects to the transition function output, thus it affects only transition codes.

The optimization coupled to this implementation is the coding of transitions such that the countable transitions cover the highest possible number of productterms (PTS). Thus, the algorithm must identify a collection of transition chains that cover a maximum number of productterms. Such an algorithm is given in [2]. That algorithm handles only Moore type machines and incorporates also other constraints for the code assignment which make it less suitable for our purpose. Because the MCNC benchmark set contains Mealy machines, we developed a different algorithm better suited to Mealy machines.

Countable transition chain Ct_i :

$$Ct_i = trs_1..trs_k \mid Code(trs_{i+1})=Code(trs_i)+1 \quad \text{where } 1<= i <= k \quad (3)$$

The total number of productterms implemented by a counter is then :

$$PTSC = \sum_{i=1}^{n} \#pt(Ct_i)$$ where n is the number of countable chains

The objective of the algorithm is to code TRS such that PTSC is maximal. Therefore all transitions get a weight assigned representing their number of productterms : $w_i = \#pt(trs_i)$. The algorithm removes first all transitions of which the start state is equal to the next state, because transition cycles are not countable. It sorts the transitions to decreasing weights. In the next phase, the algorithm builds chains of transitions. It starts with the first unmarked transition and scans the list for follow-up transitions. Thus, the transitions with the highest weight are selected first. The algorithm proceeds with this until it encounters a cycle, or it finds no follow-up transition. Furthermore, it marks only the state which is at the head of a chain. Through this way of state marking a state may occur in more than one chain. It occurs however only once as head of a chain. This redundancy avoids local optima. The algorithm generates new chains as long as there exist unmarked states that have unmarked successors. Next, the chains are sorted according their cumulative weights, again in a decreasing order. To retrieve the final collection of countable chains, the algorithm takes chains from the head of the list. Chains that contain states which do also occur in previously selected chains are rejected. The algorithm codes the states according the chains selected. This algorithm iterates until no improvement occurs in the PTSC. Table 6 shows the results for this algorithm on the MCNC FSM benchmark set. Column d.Cntr1 shows the number of countable productterms before optimization, d.Cntr2 after optimization. These results are also in table 7, column #PTS_c.

	#PTS	d.Cntr1	d.Cntr2
bbara	60	11	15
bbsse	56	8	11
bbtas	24	10	11
beect	28	4	7
cse	91	4	14
dk14	56	5	7
dk15	32	3	5
dk16	108	11	26
dk17	32	7	9
dk27	14	3	5
dk512	30	7	12
donfil	96	7	13
ex1	138	3	21
ex2	72	6	18
ex3	36	3	9
ex4	21	11	13
ex5	32	4	8
ex6	34	3	9
ex7	36	6	10
keyb	170	12	44
lion	11	3	3
lion9	25	6	6
plan	115	6	50
s1	107	6	23
sand	184	2	82
scf	166	31	120
shift	16	4	8
styr	166	8	56
tav	49	2	23
tra04	14	5	6
tra11	25	1	8

Table 6. Improvement in countable productterms.

3.2 Partial state coding

Partial state coding is not a common optimization type in controlpath synthesis systems. However, many manual designs of processor controlpaths [10] use this method to implement state transitions. It reduces the number of productterms in case several states have identical transition structures and active input variable sets. The transition function generates just a part of the next address (state code), often only the two least significant bits (LSB). The output function generates the other, the most significant, bits (MSB) of the next address. By this, these MSB bits must be identical in all next states of a certain state. This state code generation is feasible because states usually have only a small number of next states as shown in figure 1. That makes it possible to code the local selection of the next state in a few bits. Then the MSB bits of the state code can be equal among all next states of a state. That makes it possible to generate the MSB part with the output function. In case several states have identical sets of input vectors and transitions, they can share a set of productterms in the transition PLA. This reduces the number of productterms and output variables in the transition PLA. Note the cost of additional output variables for the MSB bits in the output function. The optimization problem is to find the maximal number of compatible transition sets. As stated before, transition sets are compatible in case all inputs are identical and the sets of next states contain equal subsets of states. Furthermore these sets may contain each state only once to prevent multiple codes for a single state. In case not all states are partially codeable, the transition function decomposes into two PLA's, of which one generates the usual complete next state code for all state transitions that cannot be implemented by partial state coding.

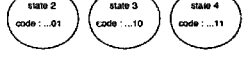

Figure 1. Example of partial coding of sucessor states.

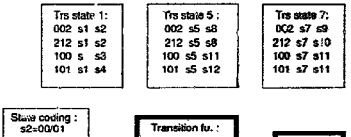

Figure 2. Example of productterm reduction by partial codes

The example of figure 2 shows a reduction in productterms. In this example, the transition set of state 7 is not identical with the other two sets because its next state structure has a different pattern. Both the input word set and the transitions have to be identical, as is the case for states 1 and 5. After state coding, the transitions of states 1 and 5 can be merged. However, the transitions from state 7 cannot be implemented in the same PLA as the merged set of states 1 and 5 . The output function generates the static code part, in case of state 1 it generates 00, of state 5 : 10, and for state 7 this code is don't care : 22.

Optimization

The next algorithm detects the number of redundant productterms in a FSM for a dbits wide LSB part of the state code. Firsts it removes all transitions starting from states that have more than 2**dbits of next states. Then it generates maximal clusters of states with identical input vectors. From these clusters the algorithm makes new ones that contain the next state collections in a matrix form with one row of next states for each state. For partial state coding, a cluster should contain only unique states. Therefore the algorithm searches for a maximal unique sub-matrix. It discards non-unique states from a state/transition matrix one by one until a sub-matrix with unique states remains. The reduction in transitions is then obtained by counting the number of states in the unique submatrix minus one row, because one row is actually implemented. From this follows also the number of redundant productterms. In table 7, column #PTS_a gives the reduction of productterms of the transition function. This method is suited best for control functions with a few transitions per state and disjoint next-state sets.

	#PTS	#S	#PTS_a	#PTS_c	#PTS_s	#PTS_t
bbara	60	10	6	15	36	0
bbsse	56	16	3	11	29	7
bbtas	24	6	8	11	16	0
beecf	28	7	0	7	17	0
cse	91	16	0	14	47	6
dk14	56	7	0	7	34	0
dk15	32	4	0	5	16	0
dk16	108	27	68	26	59	0
dk17	32	8	8	9	15	0
dk27	14	7	2	5	6	0
dk512	30	15	16	12	13	0
donfl	96	24	8	13	73	0
ex1	138	20	2	21	98	31
ex2	72	19	28	18	44	0
ex3	36	10	12	9	19	0
ex4	21	14	2	13	-	1
ex5	32	9	8	8	17	0
ex6	34	8	0	9	11	0
ex7	36	10	16	10	-	0
keyb	170	19	0	44	123	6
lion	11	4	3	3	-	0
lion9	25	9	0	6	-	0
plan	115	48	34	50	28	18
s1	107	20	3	23	27	5
sand	184	32	34	82	95	36
scf	166	121	10	120	28	78
shift	16	8	14	8	12	0
styr	166	30	0	56	77	26
tav	49	4	0	23	-	0
tra04	14	4	0	6	-	0
tra11	25	11	0	8	16	0

- : no data available from []

#PTS_a = productterm reduction for partial addressing
#PTS_c = productterm reduction for a counter
#PTS_s = productterm reduction by state-assignment
#PTS_t = productterm reduction equivalent of topol. part.

Table 7. Algorithm results : productterm reduction.

Microprogramming methods [10] use that knowledge extensively to generate the controlpath function which can be implemented efficiently with this technique.

3.3 State Assignment.

As is clear from section 2, logic minimization with state assignment reduce also the number of productterms of a PLA implementation. Therefore, we consider it as another design alternative. Table 7 shows the results of NOVA/ESPRESSO [9] as #PTS - $\#PTS_{min}$ in column #PTS_s. Unfortunately NOVA does not supply data for all machines in the MCNC set.

3.4 Topological Partitioning.

This problem concerns the division of a single PLA into several smaller PLA's such that the AND- and OR-planes of the PLA's become densely used, which reduces the total silicon area occupied. Several algorithms are known for this problem [12], [13], [14]. We selected the algorithm of Hennessy [13] and extended it to FSM's by viewing all productterms in a state as an entity, which simplifies implementation and reduces computational complexity. For more details on the algorithm we refer to [14]. One variant of this topological partitioning algorithm minimizes the total area of the PLA's, if possible. It accounts also for input/output buffers, power and ground overhead. The last column of table 7, called #PTS_t, shows the converted results. The original results are in terms area reduction. These were converted to numbers of productterms which take the equal amount of area, to make topological partitioning results comparable with the results of the other algorithms.

3.5 FSM analysis.

This section explains how the estimations for selection are made. Both the state-graph structure and I and O word densities are analysed. The most obvious analysis concerns the division of numbers of transitions per state. We started with simple estimations based on observations and computed the correlation coefficients. Then we tried to improve these correlations by exploring closely related alternatives. The tables do not show these alternatives for sake of clarity. We discuss only the final estimations in the remainder of this section. Table 8 shows the reduction of productterms #PTS-$\#PTS_{min}$ for each estimation.

The column 'subset1' of table 8 estimates countable transitions. It estimates the average number of countable transitions as the number of states times the average number of productterms for a transition (#S*#PTS/#TRS). We know that just one countable transition can occur for

each state and also the average number of productterms in a transition. This gives the average expectation of the number of productterms to be saved.

	subset1	subset2	subset3	subset4
bbara	16	0	60	1
bbsse	21	12	38	12
bbtas	12	24	24	0
beect	8	28	28	0
cse	26	8	91	7
dk14	14	0	56	1
dk15	10	0	32	0
dk16	28	108	108	2
dk17	11	32	32	0
dk27	11	8	0	0
dk512	15	30	0	0
donfl	24	96	96	1
ex1	37	12	130	24
ex2	24	72	72	1
ex3	12	36	36	1
ex4	16	10	0	7
ex5	11	32	32	0
ex6	8	16	31	3
ex7	13	36	36	1
keyb	70	12	158	19
lion	4	2	0	0
lion9	9	4	0	0
plan	77	46	52	52
s1	26	20	86	22
sand	65	44	138	69
scf	133	32	34	133
shift	8	16	0	0
styr	54	4	153	37
tav	49	0	49	0
tra04	7	8	8	0
tra11	11	16	0	0

Table 8. Productterm reduction estimations.

Column 'subset2' is for partial state coding. The number of productterms per state relates strongly to the number of code bits generated by the transition function. We choose for two bits. This allows at most four next states in a state. Therefore we count all productterms of states which have exactly two or four productterms (#{pt leaving s} = 2,4).

Next, column 'subset3' is for state assignment. A high number of productterms per state gives on average better logic minimization results. We counted the total number of productterms of states with four or more productterms (#{pt leaving s} >= 4). After some experiments, this number gave the highest correlation with state-assignment results.

The last column 'subset4' of table 8 shows the productterm reduction equivalent of the area saved with topological partitioning. This equivalent gives compatibility with the other subsets, which is needed for a correct determination of the correlation coefficients.
This number of productterms is derived from the average number of input/output variables active per state. This number lies between 0 and 1. The lower this number the higher the possible area savings by topological partitioning. The next formula was derived experimentally to correspond with the optimization results from 3.4. #PTS =1/(#Is/#S + C* #Os/#S) were #Is is the total of active input variables counted over all states. #Os is the output variable equivalent of this. C=50 is an experimentally derived constant to balance both parameters.

3.6 Results.

Table 9 lists the computed correlation coefficients between the estimated reduction in #PTS of table 8 and the algorithm results in table 7. Table 9 shows quite high correlations on the main diagonal and reasonable low correlation coefficients with other optimization estimations. However, there is one exception, which is the correlation between counter and topological partitioning. Both the counter and topological partitioning estimations correlate highly with each others algorithm results.

opt.method	subset1	subset2	subset3	subset4
counter	0.95	0.17	0.46	0.95
part.address	0.22	0.78	0.24	0.18
state-assign.	0.45	0.18	0.93	0.27
topol.partit.	0.86	0.03	0.34	0.97

Table 9. Correlation coefficients of estimations.

One can give two explanations for this. The first is that we measure the wrong parameters for both estimations. But, because the subsets 1 and 4 are based one totally different parameters which are independent by nature, this explanation is not founded. The second explanation is that both optimizations are highly correlated for the FSM's in this benchmark set. To check this assumption, we computed also all the correlations between the results of the optimization algorithms. Table 10 lists them. All coefficients on the main diagonal are one, as expected. Furthermore, this table shows that the second explanation is correct, because the counter and topological partitioning algorithm results are here also highly correlated. In fact, the coefficients in table 9 resemble those in table 10 quite closely, which shows that our estimations for alternative selection are quite reliable.

opt.method	counter	part.addr	state-ass.	topol.part.
counter	1.00	0.28	0.41	0.91
part.address	0.28	1.00	0.15	0.10
state-assign.	0.41	0.15	1.00	0.32
topol.partit.	0.91	0.10	0.32	1.00

Table 10. Correlations between optimization results.

How must we interpret FSM estimations on alternatives with regard to these correlations ?. In case the estimations for a FSM result in just one high estimation number, the selection is evident. When equal or nearly equal estimation figures occur for alternatives, one selects the alternative which has the higher correlation coefficient in table 9, because the correlation coefficient expresses the probability to be the best alternative. But, if the FSM has equal estimation figures for subsets 1 and 4. no selection can be made based the correlation coefficients. Thus, only in this case one has to run both optimization algorithms. It is likely that both alternatives have about equal minimization potentials. Concluding, for most cases our estimations give a clear indication of the probably best alternative, which helps the designer to choose among alternative implementations.

4 Conclusions.

For the estimation of logic minimization including state-assignment, we conclude from the correlation coefficient of 0.97 with an average size difference of 6% compared to the algorithm results, that it is reliable enough to apply it in synthesis decisions. It gives a good expectation of what size reductions are feasible with logic minimization for a FSM without the need for extensive computations. Although the high state-assignment problem complexity, the FSM structure is less complex, which explains the high quality of our results.

The selection estimators, also based on FSM statistics, show correlation coefficients around 0.9 and cross-correlation coefficients that are typically below 0.4. Only the highly correlated results of two optimizations in the alternatives set distort these results partially. On average, these correlations help to retrieve information on which design alternatives are the most suitable for a certain FSM. This selection method saves detailed computations for all optimizations for the alternatives. This makes this type of estimation methods also suitable for planning based design systems [15]. At this moment we extend our research on curve-fitting to other implementation techniques as multi-level logic implementations. Also, on the field of implementation selection, research for more elaborated sets of alternative implementations is carried out.

References

[1] Thomson, N., APL Programs for the Mathematics Classroom., (Springer Verlag, New York, 1989).

[2] Amann, R., Algorithmische entwurfsverfahren fuer kombinierte pla/rom-steuerwerke unter verwendung van zaehlern, Dissertation, (VDI Verlag, Duesseldorf, 1987).

[3] DeMicheli, G., Optimal State Assignment for Finite State Machines, IEEE Trans. Comp. Aided Design, vol. CAD-4, (1985) pp. 269-284.

[4] Devadas, S. and Newton, R., Exact Algorithms for Output Encoding, State Assignment, and Four-Level Boolean Minimization, IEEE Trans. on CAD, Vol.10. No.1. January 1991, pp. 13-27.

[5] Obrebska, M., Efficiency and Performance Comparison of Different Design Methodologies for Control Parts of Microprocessors, Microprocessing and Microprogramming 10, (1982) pp.163-178.

[6] Tredennick, N., The "Cultures" of Microprogramming, Proc. of the 15th annual workshop on Microprogramming- Micro-15, 1982, pp. 79-83.

[7] Paulin, P.G., Horizontal Partitioning of PLA-based Finite State Machines, Proc. of the 26th ACM/IEEE Design Automation Conference, 1989, pp. 333-338.

[8] Tarroux, G. et. al., Optimization of Micro-Controllers by Partitioning, Proc. EDAC 1991, Amsterdam, Feb. 1991, pp. 368-373.

[9] Villa, T. and Sangiovanni-Vincentelli A., NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic implementation, IEEE Trans. on CAD, vol. 9 no. 9. sept 1990, pp. 905-924.

[10] Husson, S.S., Microprogrammin Principles and Practices, Prentice Hall, 1970.

[11] Brayton, R.K., et. al., Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publ., 1984.

[12] De Micheli, G. and Santomauro, M., "Smile ; A Computer Program for partitioning of programmed logic arrays", Computer Aided design, vol 15 No. 2, March 1983, pp. 89-97.

[13] Hennessy,J., "Partitioning Programmable Logic Arrays Summary", IEEE Proc. Int. Conf. on Computer Aided Design, IEEE 1983, pp. 180-181.

[14] ten Berg, A.J.W.M., "Floorplan Optimized Topological Partitioning of Programmed Logic Arrays" accepted for publication at the WG 10.5 IFIP Workshop on Synthesis, Generation and Portability of Library Blocks for ASIC design. Grenoble, March 1992.

[15] Knapp, D.W. and Parker, A.C., "The ADAM Design Planning Engine", IEEE Trans. on Comp. Aided Des., Vol. 10, No. 7, July 1991, pp. 829 - 845.