SMAS: A PROGRAM FOR THE CONCURRENT STATE REDUCTION AND STATE ASSIGNMENT OF FINITE STATE MACHINES

M.J. Avedillo J.M. Quintana J.L. Huertas

Dept. of Design of Analog Circuits, Centro Nacional de Microelectrónica, Sevilla, SPAIN

ABSTRACT

In this paper we describe a state assignment algorithm for PLA-based machines which produces an assignment of non necessarily distinct, and eventually incompletely specified codes. In this new approach, state reduction and state assignment are concurrently dealt with, and a restricted state splitting technique is explored. The algorithm is particularly appropriate for machines with compatibility relations among its states because the potentials of state merging are exploited during the state assignment step. The input to SMAS, the programme implementing the algorithm, is a symbolic cover of the FSM. The output is a boolean representation of both next state and output functions suitable to be minimized with ESPRESSO [Bray84]. The machines in the MCNC[Lisa89] benchmark set are used to test the new algorithm and to compare it with a well known state assignment program.

I INTRODUCTION

One main task in logic system synthesis is the design of Finite Sequential Machines (FSMs). This process includes state reduction, state assignment and logic minimization. Classically, the two first steps have received an independent treatment. Afterwards, in more recent references on automatic FSM design systems, it has been stated that the classical assumptions of structural design methods are no longer valid for modern technologies [Lee84]. Lee recommends not to seek a FSM with the minimum number of states. Rudell [Rude85] does not include state minimization within FSM synthesis, but he points out the need of a more general transformation of the FSM aimed at obtaining an equivalent machine that is easier to build.

In [Lee84] it is suggested to replace the stages of state minimization and state assignment with one stage of joint minimization and state assignment. In this paper, a novel approach to the concurrent state minimization and state assignment is described. It is also explored a restricted state splitting technique. The need of state splitting for optimal state assignment has been pointed out in [Hart62], [Deva90].

II RATIONAL OF THE NEW APPROACH

Let us briefly show how the goals of state reduction and state splitting are achieved by a coding process which allows a single code to be assigned to a group of states and the use of incompletely specified codes.

Assigning a single code to a group of states is equivalent to the transformation of the FSM description by the substitution of a group of states with one single state (state reduction) and the assignment of this internal state in the new description. From classical state reduction theory we know that, in order to specify the same external behavior than original machine, the states which are merged into one state must be compatible and closure constraints must be satisfied [Unger69], [Grass68]. These conditions are used as constraints in our encoding process. This is, the state reduction is achieved during the assignment process.

Assigning an incompletely specified code [Koha78] (group of codes) to a single state is equivalent to the transformation of the FSM symbolic description consisting in the substitution of a single state by a group of states (state splitting) and the assignment of this new description..There are two reasons to allow incompletely specified codes:

1.- It is important to cope with the concurrent state reduction and state assignment of incompletely specified sequential machines. In general, for this kind of FSMs, a closed set of compatibles covering a state table (solution to the classic state reduction phase) is not disjoint. But a non disjoint gathering of states cannot be achieved via state assignment if incompletely specified codes are not allowed. Thus, concurrent state reduction and state assignment will fail to reach a part of the space of solutions so, precluding the possibility of finding some efficient ones.
2.- The boolean cover of the machine which is supplied to the logic minimizer, can have extra flexibility in this way, because some next state entries have been substituted by a group of states. The minimizer can take advantage by choosing each time the best state of the group to achieve minimization. In [Tseng86] it is reported that allowing "don't care" bits in a state assignment often results in a significant reduction of the combinational component.

Nevertheless, this state splitting is restricted, because the codes assigned to the set of states resulting from the splitting of an original one are constrained to form a cube.

II.1 Concurrent state reduction and state assignment

From previous paragraphs, it is clear that our assignment process is equivalent to both the transformation of the symbolic description of the FSM and to the assignment of this new description. Some constraints need to be imposed to the encoding so that the behavior of the machine is not changed by the state reduction achieved concurrently to the assignment.

Let us introduce some definitions in order to formulate the constraints the assignment has to satisfy. We say two states have **compatible assignments** if the intersection of their codes is not empty, (notice that "don't care" bits are allowed in the codes). For example, assume the state S_1 has been given the code (001−) and the state S_2 the code (0010). S_1 and S_2 have compatible assignments because $(001-) \cap (0010) = (0010)$. **Identical assignments** is a particular case of compatible assignments.

We say two states have **discriminated assignments** if the intersection of their codes is empty.

In order to guarantee that an encoding is valid (the derived logic implements the desired behavior) the following constraints must be satisfied:

1.- A pair of states with compatible assignments must be compatible states or, a pair of incompatible states must be given discriminated assignments.

2.- For each input sequence, the pairs of states which are implied by a pair of states with compatible assignments must have compatible assignments too or, the pairs of states which imply a pair of states with discriminated assignments must be given discriminated assignments too.

III THE ALGORITHM

Given a state table describing the external behavior of a FSM, the state assignment phase tries to find the binary representation of the internal states of the machine corresponding to a PLA of minimal area. In our approach a valid binary representation is built up by exploring both the potentials of state merging and state splitting.

Given *ns* internal states in the symbolic description to be assigned, we assume that $ns \times nb$ bits are going to be assigned to a value from the set {0, 1, −}, where *nb* is the number of bits of the generated codes and so it is not known until the algorithm has finished the assignment process. We start with the number of bits needed to encode as many states as there are in the maximal incompatible of highest cardinality (nb_0). Afterwards, the length of the codes will be incremented by one bit, each time the actual intermediate assignment cannot be turned into a valid one with the current number of bits in the encoding.

Initially the codes of all the states are completely unspecified, the algorithm assigns bits to 0 or 1 until the assignment is valid according to conditions 1 and 2. In order to describe how the algorithm works, we introduce the concept of **pair of compulsory discrimination** as those pairs of states which cannot have compatible assignments (incompatible states, and those pairs which imply a pair with discriminated assignments, are **pairs of compulsory discrimination**). Initially the only pairs of compulsory discrimination are the pairs of incompatible states. Figure 1 shows the main data structures and a Pidgin-C description of the algorithm.

III.1 Procedure Initialize

The lists DAP, IAP, IIP are initialized to the empty set. The list of pairs of compulsory discrimination (CDP) is initialized to the set of

Data Structures

DAP: list of pairs with discriminated assignments.
CDP: list of pairs of compulsory discrimination.
IIP: list of pairs implied by pairs of states with identical assignments.
IAP: list of pairs with identical assignments.

```
Main()
{  Read  FSM();
   Initialize();
   for ( each pair (si,sj) in CDP)
   {  Discriminate(si,sj);
   }
}
```

Figure 1

pairs of incompatible states. The initial length of the codes (nb_0) is determined. A state (Sinit) is selected to have nb_0 bits assigned to 0.

III.2 Procedure Discriminate

Basic operation in the process is the discrimination of the assignments of a pair of states. Among all the different assignments of bits leading to the discrimination, the best one according to a cost function is chosen. The lists DAP, IAP, IIP,CDP are then updated. for example, for DCP to be updated pairs which have been discriminated are removed and those pair implying other pairs just discriminated are added. The discrimination of some pairs can require that the code length is incremented as we said before. In this case the lists *IIP, IAP* are emptied.

Theorem: When there are no pairs of compulsory discrimination left, the actual assignment satisfies conditions 1 and 2 and so it is valid.

Proof: a) Suppose it is not a legal assignment because there are pairs with compatible assignments which are no compatible states. This would mean an incompatible pair has not been discriminated yet and so, the list of pairs of compulsory discrimination would not be empty as it is the condition for the process to finish.

b) Suppose it is not a legal assignment because there are pairs which have compatible codes but they imply pairs with discriminated assignments. This is not possible because when discriminating those implied pairs, the implying pairs would have been added to the list of pairs of compulsory discrimination.

III.3 Considerations about order strategies and cost function

It is worth noting that the straight application of our algorithm does not guarantee that the length of the codes is minimum. However, several heuristics have been developed in order to achieve more efficient assignments. For example, the order in which pairs of states are discriminated and the cost function are critical. Very efficient order strategies have been introduced so that, in practice, codes longer than minimum ones are not usually generated. Moreover, the potential of state merging leads, in some cases, to assignments with a number of state variables which is less than the minimum needed to code the number of states in the initial symbolic description. The cost function has been designed to force the number of bits to keep low. At the same time, it aims at reducing the area of final implementations by adding extra flexibility to next state functions and by taking into account the fulfillment of the adjacencies derived from Humphrey's rules.

IV EXPERIMENTAL RESULTS

The algorithm is particularly suited for those machines for which state merging applies and so a subset of the MCNC machines, those with compatibles or equivalent states, has been used to test the algorithm. We have also tried many examples picked out from Switching Circuit Textbooks and journal papers about minimization (all of them referenced in [Reus86]). We will focus on two figures of merits: the area occupied by the combinational component in the PLA implementation of the FSM and the time which is required for design.

Tables I and II depict the results for both groups of machines. The number of primary inputs (ni), primary outputs (no) and states (ns) are shown for each of the machines to which SMAS (C implementation of the new algorithm) and NOVA [Vila88] (hybrid algorithm) have been applied. The number of state variables (nv), the number of product terms (tp) after logic minimization and the size ((2ni + 3nv + no)tp) of the PLA implementing the combinational component for the assignments obtained with each program are also shown in Tables I and II. Times are given in seconds in a SUN WorkStation 3/260. A column with the ratio of sizes, Ar, is included

$$Ar = \frac{\textit{Size with SMAS } (size_S)}{\textit{Size with NOVA } (size_N)}$$

From both Tables, we can see that in all the cases the number of state variables in SMAS' assignments is less than or equal to the number of state variables in NOVA's assignments. Concerning the first group of machines, SMAS obtains a more efficient implementation than NOVA does in 16 of the 18 machines we have tried. In particular, let us focus on the set formed by *donfile, modulo12, s1a* and *s8*. These machines do not need to be implemented as FSMs. NOVA is not able of detecting it and supplies an assignment for them. This leads to realizations which are expensive in terms of area. None of the machines in the second group (Table II) is more efficiently assigned with NOVA than it is with SMAS.

In Tables III and IV, the arithmetic average of the ratios of sizes and of the ratios of times (it is defined similar to the ratios of sizes) between our algorithm and different algorithms in NOVA are shown. The default random option that we use, tries as many random encodings as states in the machine. The hybrid algorithm from NOVA has been chosen because it is said to provide the best tradeoff between area and time. Table III summarizes results for group 1 while Table IV is devoted to group 2. As we can observe from Table III, area savings up to 50% are achieved with SMAS versus the hybrid algorithm in NOVA, the average of time ratios being near to 1. Even more area and time savings are achieved by SMAS on the second group of machines.

Finally, although this algorithm is not tailored to the assignment of minimized machines, when SMAS is applied to the subset of the MCNC benchmark corresponding to minimized machines, there is only an increment of around 11% in area with respect to the i-hybrid option in NOVA.

CONCLUSIONS

We have developed and programmed an state assignment algorithm which, concurrently to the assignment of binary codes in the symbolic representation, exploits the potentials of state merging and state splitting when it is possible.

Up to now, the algorithm is extremely advantageous for the assignment of non minimized machines because of the power of state merging. It achieves slightly worse results than NOVA for minimized machines. Nevertheless, this is a preliminary version of the work. In our opinion, paying more attention to the state splitting technique and the tuning of the cost function will lead to a more efficient assignment of machines with a minimum number of states. In any case, the algorithm is original and paves the way to a new treatment of state assignment.

REFERENCES

[Bray84] R. K. Brayton, G.D. Hatchel, C. McMullen, and A.L. Sangiovanni, "Logic Minimization Algorithms for VLSI Synthesis". Hingham, MA, Kluwer Academic Pub., 1984.

[Deva90] S. Devadas, H. T. Ma, A.R. Newton, A. Sangiovanni-Vincentelli, "Irredundant Sequential Machines Via Optimal Logic Synthesis", IEEE Trans. on CAD, Vol 9, pp 8-17, January 1990.

[Gras68] A. Grasselli and F. Luccio, "Some covering Problems in Switching Theory", "Network and Switching Theory", Academic Press, 1968.

[Hart62] J. Hartmanis and R.E. Stearns, "Some Dangers in the State Reduction of Sequential Machines", Inform. Cont., pp 252-260, Sept. 1962.

[Koha78] Z. Kohavi, "Switching and Finite Automata Theory". New York, MCGraw Hill, 1978.

[Lee84] E.B. Lee and M. A. Perkowski "Concurrent Minimization and State Assignment of Finite State Machines", *Proc. 1984 Intern. Conf. on Syst. Man. and Cyb. IEEE, Halifax, October 1984.*

[Lisa89] R. Lisanke, "Introduction of Synthesis benchmark", Int. Workshop on Logic Synthesis, North Carolina, 1989.

[Reus86] B. Reusch and W. Merzenich, "Minimal Coverings for Incompletely Specified Sequential Machines", *Acta Informatica,* No. 22, pp. 663-678, 1986.

[Rude85] R. Rudell, A. Sangiovanni-Vincentelli, G. De Micheli, "A Finite-State Machine Synthesis System", *Proceedings of ISCAS 1985*, pp.647-650, May 1985.

[Tsen86] C Tseng, A.M: Prabhu, C. Li, Z. Mehmood, M.M. Tong, "A Versatile Finite State Machine Synthesizer", *Proceedings of ISCAS 1986, pp 206-209. 1986.*

[Unger69] S. H. Unger, "Asynchronous Sequential Switching Circuits", Wiley-Interscience, New York, 1969.

[Vila88] T. Vila,"NOVA", in *Octtools User's Manual*, UCB, March, 1988.

				NOVA				SMAS				Ar
	ni	*no*	*ns*	*nv*	*tp*	$size_N$	*time*	*nv*	*tp*	$size_S$	*time*	
bbara	4	2	10	4	25	550	4.6	3	22	418	1.95	0.76
bbsse	7	7	16	4	30	990	6.9	4	29	957	9.6	0.97
beecount	3	4	7	3	13	247	2.5	3	10	190	1.13	0.77
donfile	2	1	24	5	35	700	1306.6	0	0	0	0.98	0.00
ex1	9	19	20	5	48	2496	94.3	5	56	2912	87.15	1.17
ex2	2	2	19	5	29	609	8.4	4	11	198	5.18	0.32
ex3	2	2	10	4	18	324	3.1	3	9	135	1.28	0.42
ex5	2	2	9	4	14	252	9.6	2	9	108	1.05	0.43
ex7	2	2	10	4	17	306	5.7	2	7	84	1.07	0.27
lion9	2	1	9	4	8	136	5.6	3	8	112	1.05	0.82
mark1	5	16	15	4	21	798	56.8	4	19	722	6.75	0.90
modulo12	1	1	12	4	11	165	1.63	0	0	0	0.38	0.00
opus	5	6	10	4	16	448	2.1	4	19	532	2.7	1.19
s1a	8	6	20	5	76	2812	12.63	0	0	0	37.88	0.00
s8	4	1	5	3	10	180	2.555	0	0	0	0.23	0.00
sse	7	7	16	4	30	990	7.0	4	29	957	10.05	0.97
tbk	6	3	32	5	151	4530	2155.8	4	55	1485	113.18	0.33
train11	2	1	11	4	9	153	8.0	2	6	66	1.2	0.43

Table I

				NOVA				SMAS				Ar
	ni	*no*	ns	*nv*	*tp*	$size_N$	*time*	*nv*	*tp*	$size_S$	*time*	
FSM1	2	1	5	3	8	112	4.0	2	3	33	0.4	0.29
FSM2	2	1	9	4	12	204	4.4	3	8	112	1.2	0.55
FSM3	2	1	7	3	11	154	3.1	2	5	55	0.6	0.36
FSM4	2	1	6	3	8	112	3.3	2	4	44	0.5	0.39
FSM5	2	1	6	3	10	140	4.0	2	6	66	0.7	0.47
FSM6	2	1	22	5	31	620	378.0	4	16	272	10.7	0.44
FSM7	2	1	9	4	15	255	40.3	2	8	88	1.0	0.345
FSM8	2	1	6	3	9	126	3.3	2	7	77	0.6	0.61
FSM9	3	1	9	4	15	255	14.0	2	7	91	0.9	0.36
FSM10	3	1	8	3	18	288	8.8	2	8	104	1.1	0.36
FSM11	2	1	6	3	10	140	2.5	3	9	126	0.9	0.90
FSM12	2	1	9	4	13	221	7.9	2	7	77	0.8	0.35
FSM13	3	1	6	3	13	208	3.4	2	8	104	0.8	0.50
FSM14	2	1	6	3	7	98	2.1	1	3	24	0.6	0.24
FSM15	3	1	10	4	11	209	13.3	2	4	52	1.3	0.25

Table II

ratios	random #ns	NOVA hybrid
area	0.48	0.54
time	0.12	0.61

Table III

ratios	random #ns	NOVA hybrid
area	0.44	0.43
time	0.08	0.16

Table IV