Efficiency of state assignment methods for PLA-based sequential circuits

Professor J.L. Huertas
J.M. Quintana

Indexing terms: Digital control, Integrated circuits

Abstract: The implementation of finite sequential machines by using a programmable array logic to synthesise their combinational part is considered. A critical view of the efficiency of existing methods to carry out the state assignment of these machines is given, and it is shown that we can derive a bound on the number of state variables beyond which even an arbitrary coding usually leads to better results in terms of area occupation. It is suggested in the paper that this bound can still be found when more refined area estimates are used.

1 Introduction

Computer-aided synthesis of sequential circuits is an area of active research [1–4] because of the important role they play in the design of complex digital systems. To be efficient, a design procedure for such circuits must resort to regular structures; in particular, the programmable logic array (PLA) is considered a natural candidate for a design methodology coping with design complexity.

A PLA implementation of a finite state machine (FSM) is shown in Fig. 1, where the PLA size depends on

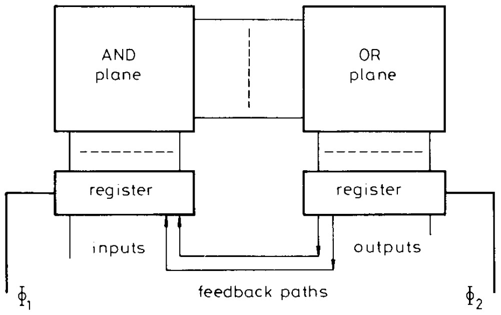

Fig. 1 *PLA implementation of FSM*

both the number of required state variables ns and the number of product terms tp of the next-state equations describing the FSM.

Classically, the synthesis of a sequential circuit must be partitioned into several tasks. Here we will consider the optimal state assignment problem, which is probably the step that most critically influences the final cost of the circuit. Methods to deal with this problem have been reported in the past that are specifically said to be tailored to PLA-based implementations [5–10]. All of these approaches rely on strategies that are focussed on obtaining assignments with either a minimum or a quasi-minimum number of product terms. The price to be paid is that they require a number of state variables (equivalently a register length) which is higher than the minimum. Although all of those methods produce very nice results when applied to FSM with a few states, we have realised that for large machines they lead to assignments far away from an optimum. In general, all of them lead to area wasting when the number of involved variables increases.

The objective of this paper is twofold. First of all, we will use a conventional figure to estimate the silicon area occupation for a given FSM. Then, based on such an estimation, we will show the inefficiency of the known methods, and we will conclude that the usual strategy of minimising the next-state function cardinality (i.e. the number of product terms) leads to worse assignments than minimising the number of state variables in many cases. As a consequence, an *a priori* estimation of the final area occupation will be proven to be of interest. Finally, a discussion on more-accurate area estimates is included. Although the occupied area is more-precisely evaluated, all of those estimates allow us to arrive at the same conclusions as the originally used area-occupation figure.

2 Background

2.1 Basic cost criterion

When designing an FSM by using a PLA to implement its combinational component the main concerns for efficiency are the area occupation and the final operating speed. Since the latter is strongly influenced by the length of the input-output paths, a compact implementation occupying as small an area as possible usually allows us to fulfill both basic requirements. Concerning the silicon area, it depends on the relative positioning of the different PLA elements. For the sake of simplicity, we will consider here the arrangement shown in Fig. 2. There the array length is proportional to nI, the number of inputs to the AND array of the PLA, and to nO, the number of outputs to the OR array, its depth being proportional to the cardinality of the combinational function to be realised, i.e. to the number of product terms tp in the AND array. We will consider initially a dynamic system, this fact affecting only the physical dimensions of the register bank. Since our results can be extended to other cases

Paper 6695E (C1, C2), first received 5th April 1988 and in revised form 27th January 1989

Professor Huertas and Dr. Quintana are at the Departamento de Electrónica y Electromagnetismo, Facultad de Fisica, Universidad de Sevilla, Avda. Reina Mercedes s/n, Sevilla, 41012-Sevilla, Spain

[17], we will restrict ourselves to consider the case where the state variables are the output variables of the FSM. With this in mind, the area can be expressed as [19]

$$A = k_0 tp[k_1(ni + ns) + k_2 ns] \quad (1)$$

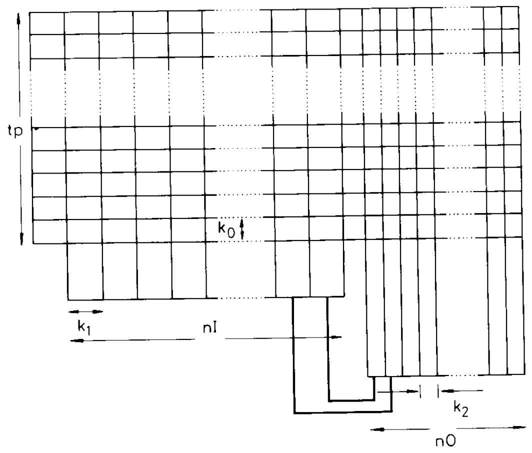

Fig. 2 *Topological representation of PLA*

where ni is the number of the FSM inputs, ns is the number of state variables, k_0 is a constant that depends on the technology, and k_1 and k_2 are constants that depend on the detailed design of the input decoders and the output buffers, respectively. Because of the technology we are currently using in our laboratory we will take $k_1 = 2k_0$ and $k_2 = k_0$, to simplify the maths. Then

$$A = k_0 tp[2ni + 3ns] \quad (2)$$

2.2 Existing assignment strategies

Coding methods for PLA-based FSMs fall into one of two categories. The approach from De Micheli *et al.* (henceforth referred to as DM) [7, 9–12] uses a symbolic minimisation of the FSM combinational part, followed by the solution of a constrained encoding problem. Symbolic minimisation is carried out by multiple-valued minimisation and gives a minimal representation of the next-state function of the FSM which does not depend on the particular state assignment to be chosen. This technique reduces the number of product terms implementing the FSM combinational component.

The second approach is represented by the work of Papachristou and Sharma (P-S) [6] and Acha and Calvo (A-C) [8]. Both methods try to find an efficient state coding resulting in a reduced (ideally minimal) number of product terms, by decomposing the state table into state block partitions according to some heuristics rules [13]. The method in Reference 6 is a Liu procedure [14, 15] with a particular set lumping strategy. On the other hand, the method in Reference 8 is a generalised Liu procedure that through the solution of a covering problem determines a minimal set of maximal compatibles [16]. The former method does not give all the minimum Liu assignments and even might fail in obtaining one, although it generally leads to near-minimal results without high computational overheads. On the contrary, the latter method allows the designer to get all the minimum Liu codings. However, there is a big difference in terms of CPU time, the P-S approach being usually much faster than the A-C approach.

All of these methods [6–9] try to find a minimal cardinality implementation of the FSM next-state function, i.e. an implementation with the minimum number of product terms. Of course, the price to be paid is that the number of state variables (in other words, the register length) is not the minimum, and is usually far away from such a minimum.

It was shown in Reference 16 that given the state table describing a FSM, the minimum number of product terms for its next-state function can be easily calculated. We will call henceforth the Liu number tp_L of a machine to this minimum. Hence, we can predict *a priori* the minimal cardinality of the combinational part which might be attainable during the assignment process. We will show in the next Section how this number can be used (in conjunction with the minimum number of state variables) to determine a lower bound to the usefulness of any of the methods revised in the present Section.

3 Formal comparison of algorithms

3.1 Previous example

Let us begin by considering a few examples as a motivation for our work. Tables 1 and 2 show the state table for two FSMs of different sizes. Table 1 has been selected from the literature [6] and Table 2 has been randomly generated. A third Table from elsewhere [17] has been considered too, this one corresponding to a randomly

Table 1: State table of FSM1

Present state	Input state				
	I1	I2	I3	I4	I5
s1	s3	s2	s2	s7	s3
s2	s5	s2	s1	s7	s3
s3	s5	s4	s3	s6	s4
s4	s6	s5	s4	s2	s4
s5	s7	s5	s4	s1	s6
s6	s1	s6	s4	s1	s6
s7	s1	s2	s3	s3	s6

Table 2: State table of FSM2

Present state	Input state					
	I1	I2	I3	I4	I5	I6
s1	s1	s1	s4	s1	s2	s3
s2	s3	s3	s4	s1	s4	s3
s3	s4	s1	s1	s1	s2	s5
s4	s2	s1	s4	s2	s6	s3
s5	s5	s3	s1	s2	s2	s7
s6	sA	sA	sB	sE	s6	s3
s7	s8	sC	s1	sE	s2	s7
s8	sA	s9	s9	sE	sA	s4
s9	sC	s9	s8	sE	sA	s4
sA	sC	sB	sA	sD	sB	s3
sB	sD	sC	sB	s9	sB	s7
sC	sE	sC	sB	s8	sD	s3
sD	s8	sD	sB	s8	sD	s7
sE	s8	s9	sA	sA	sD	s8

Table 3: Area assignments using different algorithms

FSM	Dimension		Area			
	I	S	A-C	P-S	DM	Random
FSM1	5	7	306	306	306	330
FSM2	6	14	Not Practical	1404	1296	918
FSM3	15	80	Not Practical	86275	Not Practical	29464

I applies for the number of input columns
S applies for the number of states

generated FSM with 80 states and 15 input columns. Because of its size, we refer the interested reader to [17] for the details of this state table. We have applied to these machines the three state-assignment methods referenced above, as well as a random search. The values for the occupied area appear in Table 3, this estimation corresponding to an n-MOS technology we have available, with $k_0 = \lambda$. Although these figures will vary when moving from one technology to another, they are illustrative enough for showing the relative value of the different algorithms. The inefficiency of the methods should be clear when dealing with those three examples. With the exception of the smallest machine, an arbitrary assignment is always better than the assignments generated by the algorithms. Entries of 'Not Practical' apply for a method in Table 3 when the CPU time was found to be at least two orders of magnitude higher than a random search.

3.2 Bounds on usefulness of a coding algorithm

Let us consider a sequential machine whose internal states can be coded using as many as ns_{min} state variables. Let us call tp_{min} the minimum number of product terms for realising the combinational part of that FSM; in general, when using tp_{min}, the required number of state variables will be higher than ns_{min}. Taking into account the area occupation formulae derived above, we should try to evaluate whether or not a given assignment is better than other in terms of silicon area. In particular we will compare an assignment using the minimum number of state variables with an assignment with the minimal cardinality. Let us call A to a state assignment of the former class and B to an assignment of the latter. To be precise

a class A assignment has $tp = tp_{min}$, and $ns \geqslant ns_{min}$

a class B assignment has $tp \geqslant tp_{min}$, and $ns = ns_{min}$

To be considered advantageous, assignment A must occupy less area than assignment B. As we have pointed out before, we can estimate a lower bound for tp. Hence, presuming this bound is always attainable (i.e. $tp_{min} = tp_L$), we can say that for any class A assignment to be 'better' than any other using a minimum number of state variables (class B assignments) the following inequality must be fulfilled:

$$(2ni + 3ns)tp_L \leqslant (2ni + 3ns_{min})tp \tag{3}$$

We can represent eqn. 3 in a design space, whose coordinates are ns/ns_{min} and tp/tp_L. The region of this space where class A assignments are advantageous corresponds to

$$\frac{tp}{tp_L} \geqslant \frac{2ni + 3ns}{2ni + 3ns_{min}} \tag{4}$$

The design space is also bounded by other two considerations: first, no solution to the coding problem can be found requiring less than ns_{min}; secondly, we can define a maximum value for tp, tp_{max}. A rough evaluation for such a maximum could be

$$tp_{max} = (\text{no. of states})(\text{no. of inputs}) \tag{5}$$

A better estimate can be derived by using any randomly generated assignment whose cardinality will be a more accurate bound. Anyhow, only the shadowed triangle in Fig. 3 gives rise to minimal-cardinality assignments with a lower cost than an assignment with the minimum number of state variables.

From Fig. 3, the maximum value of the number for state variables ns_{max} can be calculated.

$$ns_{max} = \frac{(2ni + 3ns_{min})tp_{max} - 2nitp_L}{3tp_L} \tag{6}$$

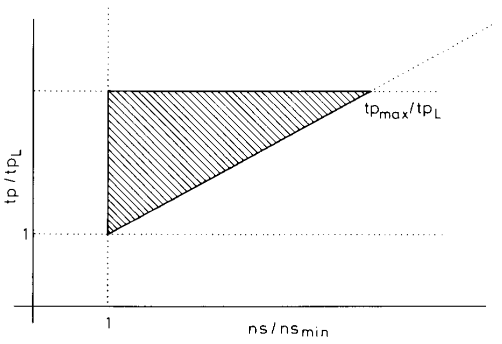

Fig. 3 *Design space for PLA-based FSM*

Note that this bound is rather optimistic for the minimal cardinality assignment, especially if we use the value of tp_{max} in eqn. 5.

To understand more clearly the practical consequence of eqn. 6 we can rewrite eqn. 2 using

$$tp = tp_L + \Delta tp$$

$$ns = ns_{min} + \Delta ns$$

which gives a figure for the influence of Δtp and Δns on the silicon area.

$$\Delta A_1 = k_0^2(2ni + 3ns)\Delta tp \tag{7}$$

$$\Delta A_2 = 3k_0^2 \, tp\Delta ns \tag{8}$$

Since $3tp$ is usually much greater than $(2ni + 3ns)$, eqns. 7 and 8 mean that increasing the number of state variables is usually much more dramatic than increasing the number of product terms. This is especially true for FSM of medium and bigger sizes.

With this result in mind, it is worth considering the problem of predicting whether or not a given method could give a good result prior to performing the coding process. Note that for a large FSM this is very interesting because of the high computation times required to carry out the state assignment.

Turning back to eqn. 3, and after some manipulations, we can write

$$\eta = \frac{ns_{max}}{ns_{min}} = \frac{2ni + 3ns_{min}}{3ns_{min}} \frac{tp/tp_{max}}{tp_L/tp_{max}} - \frac{2ni}{3ns_{min}} \tag{9}$$

Given the state table of a FSM, we can represent eqn. 9 in the plane of Fig. 4 using an auxiliary parameter, $\sigma = tp/tp_{max}$. On this Figure we can explain the basis for an evaluation procedure. For any machine, we will determine the value of tp_L. Then, using a plot like the one in Fig. 4, we will estimate the maximum number of state variables to be allowed so that a minimal cardinality assignment could be area-efficient. Of course this maximum value for ns will depend on the value of σ we consider, but since the curves tend to be flat above $tp_L/tp_{max} = 0.3$, the influence of σ can be disregarded in many practical situations and hence we can incorporate into any coding algorithm a stopping mechanism when ns_{max} is surpassed.

Two additional considerations are necessary. First, the usual value of σ could be expected to lie between 0.6 and 0.9, thus reducing the curves in Fig. 4 to be taken into account. Secondly, most FSMs have a Liu number $\xi = tp_L/tp_{max}$, between 0.4 and 0.8, which means an additional restriction of the interesting region in Fig. 4.

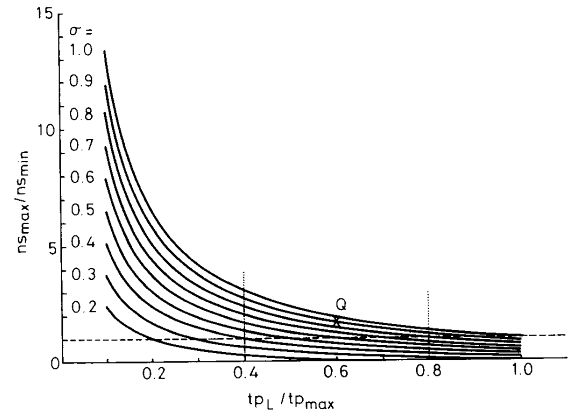

Fig. 4 *Maximum value of ns as function of* σ
$ni = 15, ns = 80$

As an illustration, Table 4 shows the Liu number for the three FSMs in Tables 1 and 2, and the state table of Reference 17 as well as the value of σ obtained by a randomly generated coding of cardinality tp_r. In Table 4, ns_{stop} represents the maximum value of ns still giving a better area occupation than that corresponding to ns_{min}, and ns_{actual} corresponds to the value of ns obtained when the P-S method is employed. It should be clear that a search for a minimal cardinality state coding could be stopped largely before the encoding algorithm is completed. Also, Fig. 5 illustrates our assertion about the 'typical' value for ξ. This parameter has been represented in Fig. 5 for one hundred FSMs of medium size. All of those machines in Fig. 5 have a value of tp_L/tp_{max} around 0.6. Also, Table 5 represents in more detail the value of the different parameters for these FSMs. To save space only 30 are shown in this Table. The CPU time is explicitly indicated and corresponds to a VAX 11/750 under ULTRIX-32*.

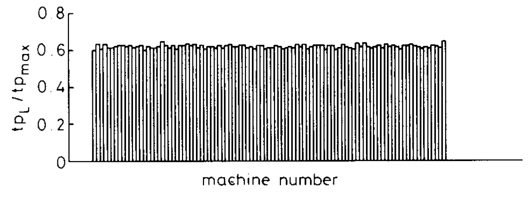

Fig. 5 *tp_L/tp_{max} for 100 FSMs of medium size*

Nevertheless, our results must not be misunderstood. We cannot conclude that minimal cardinality methods have to be avoided, since in some cases they can lead to good solutions. For example, in machines admitting a small value of tp_L, these methods could give a better performance. See, for instance, the FSM described by Table 6 [18]. That machine has a very low value ot tp_L, thus allowing us to explore for a good coding. Table 6 also gives a coding obtained by the P-S method, which has minimal cardinality (7 in this case). Unfortunately, machines with such a low value for tp_L/tp_{max} are rarely found. Moreover, even in this case, an assignment can be derived using the minimum number of state variables (3) and a higher cardinality (8 product terms), this coding occupying slightly less silicon area.

In summary, using any minimal cardinality state encoding technique must be inefficient and it is worth including an evaluation mechanism to show whether or not such a technique has to be disregarded in favour of a

Table 5: Parameters of 30 FSMs of medium size

	KISS [9]				P-S [6]			
	tps	*nb*	Area	Time	*tps*	*nb*	Area	Time, s
FSM1	46	9	1518	260.9	46	9	1518	11.2
FSM2	47	8	1410	233.5	47	10	1692	13.1
FSM3	50	9	1650	284.9	50	9	1650	11.1
FSM4	51	8	1530	244.9	51	8	1530	11.4
FSM5	51	8	1530	216.2	51	9	1683	10.3
FSM6	43	11	1677	518.8	43	11	1677	10.5
FSM7	50	9	1650	263.3	50	9	1650	10.4
FSM8	44	9	1452	251.3	44	10	1584	9.9
FSM9	42	8	1260	242.5	42	10	1512	10.1
FSM10	45	8	1350	234.1	45	10	1620	10.5
FSM11	50	7	1350	218.4	50	9	1650	9.9
FSM12	50	6	1200	204.1	50	9	1650	10.4
FSM13	46	10	1656	371.5	46	9	1518	9.9
FSM14	48	11	1872	347.9	48	10	1728	10.4
FSM15	47	10	1692	289.8	47	8	1410	9.9
FSM16	48	11	1872	362.9	48	9	1584	10.2
FSM17	49	10	1764	266.1	49	10	1764	10.5
FSM18	45	9	1485	297.5	45	10	1620	10.3
FSM19	57	6	1368	208.7	57	8	1710	10.2
FSM20	51	7	1377	215.7	51	11	1989	10.3
FSM21	53	7	1431	224.5	53	9	1749	10.1
FSM22	43	13	1935	103.3	43	8	1290	10.0
FSM23	45	11	1755	484.8	45	10	1620	10.2
FSM24	48	6	1152	203.1	48	9	1584	10.1
FSM25	53	6	1272	210.7	53	8	1590	10.2
FSM26	48	10	1728	320.4	48	8	1440	10.2
FSM27	51	6	1224	204.0	51	10	1836	10.2
FSM28	43	11	1677	429.8	43	9	1419	10.1
FSM29	47	9	1551	224.3	47	9	1551	10.1
FSM30	45	9	1485	261.2	45	7	1215	9.9

Table 6: Example of FSM with low ξ ($\xi = 0.25$)

Present state	Input state				Assignment
	I1	I2	I3	I4	
s1	s2	s3	s4	s1	s1 → 1100
s2	s5	s6	s7	s7	s2 → 0010
s3	s5	s6	s2	s7	s3 → 0000
s4	s2	s3	s4	s1	s4 → 1100
s5	s7	s3	s7	s1	s5 → 1010
s6	s5	s6	s4	s7	s6 → 0100
s7	s5	s6	s2	s7	s7 → 0001

Table 4: Design parameters for FSM1, FSM2 and FSM3

FSM	tp_L	tp_r	tp_{max}	$\xi = tp_L/tp_{max}$	$\sigma = tp_r/tp_{max}$	ns_{min}	ns_{stop}	ns_{actual}
FSM1	17	22	35	0.48	0.63	3	4	4
FSM2	36	51	84	0.40	0.61	4	6	11
FSM3	725	1016	1200	0.60	0.85	7	10	37

* VAX and ULTRIX are trademarks of DEC.

method based on a minimum number of state variables. This evaluation mechanism can be incorporated as an stopping routine into any coding computer program or can simply give an idea of how large is the probability of obtaining a good solution before going into any coding process. For instance, in the third example of Table 4, it should be clear using Fig. 4 (see point Q in this Figure) that we are very unlikely to find a good solution by employing any minimal cardinality algorithm because the bound for ns_{max} is 10, which is rather lower than expected to be attained by any of the referred-to methods. In fact, Table 4 shows that the number of state variables obtained is very much higher (37 variables).

4 More-accurate area estimates

The conclusions we have drawn above, seem to be based on the area estimation we have taken from the literature [19]. Unfortunately, although eqn. 1 has been typically used for comparing different implementations of a sequential machine using a PLA [17], this expression is a partial figure of merit, which may be misleading when referred to an integrated realisation. Hence, since much more meaningful expressions can be derived instead of eqn. 1, there remains the question of whether or not our results in Section 3 are still valid when we perform a more careful evaluation of the area occupation for a given state assignment. In this Section, we will derive more-precise expressions for the area and we will manipulate them at the same way we have done with eqn. 1 in the previous discussion. To carry out such a derivation, we will consider separately the case for static and dynamic flip-flops. These new figures can be used at the same way we have used eqn. 1 in the previous discussion.

It is useful to mention the effects of topological optimisation on our considerations, specifically PLA folding. In the case of combinational circuits, PLA folding has been successfully applied [20, 21]; however, where sequential circuits are concerned, the existence of feedback paths severely reduces the probability of finding practical solutions. For that reason, we have not considered the incidence of folding on our conclusions.

4.1 Figure of merit for internal area occupation

Let us consider the symbolic layout of a FSM shown in Fig. 2. It should be clear that there exists an area occupation that is not taken into account for the conventional cost criterion in Section. 2.1. The terms not considered in eqn. 1 are detailed in Fig. 6. The length of the array is increased by a constant term k_5 due to the pull-up devices, and its depth is augmented by a term $k_3\,ns$, which is due to the wiring that feeds back the state variables to the PLA input and by a constant term k_4 due to the output buffer. The resulting cost expression for the FSM in Fig. 6 will be

$$A_1 = [(k_1(ni + ns) + k_2\,ns + k_5) \times (k_0\,tp + k_3\,ns + k_4)] \quad (10)$$

A similar expression can be derived when the FSM is implemented by using static flip-flops (Fig. 7). If this is the case we will find

$$A_2 = [(k_1(ni + ns) + k_2\,ns + k_5) \times (k_0\,tp + k_4)] + KC + KR \quad (11)$$

where KC and KR correspond to the channel and register areas, respectively.

4.2 External area occupation

An even more-accurate estimate must consider the area due to the inclusion of the FSM into a chip, where it has

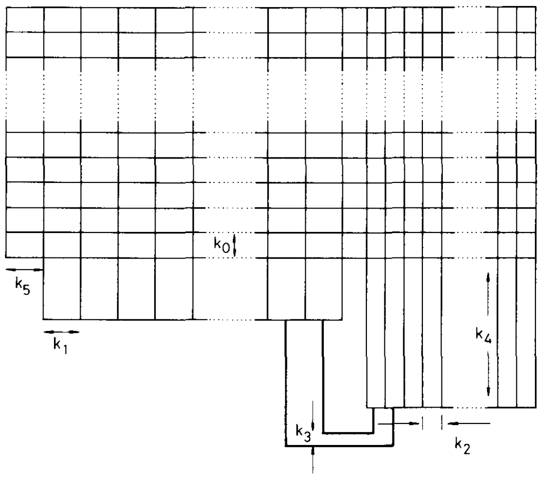

Fig. 6 *Topological representation of FSM when dynamic flip-flops are used*

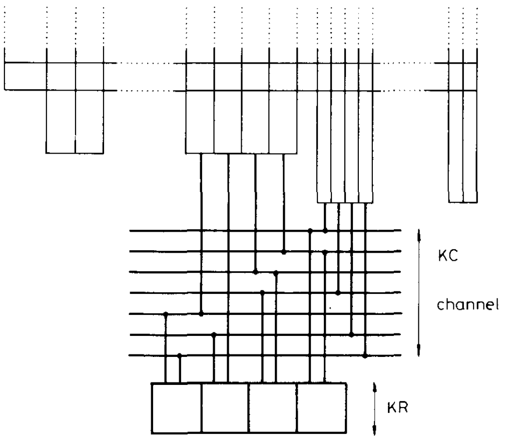

Fig. 7 *Detail of PLA-based FSM using static register*

to communicate with other subcircuits. Making such an estimation is impossible in a quite general way, since it will depend strongly on the freedom of the designer to select an arbitrary encoding for all the subsystems.

Because of the difficulty of defining a general cost criterion, we will assume several simplifications here. Thus, the influence of a given assignment of a FSM on its on-chip environment will be represented by a bus of depth proportional to its number of state variables and of length equal to the internal PLA length plus the internal PLA depth (see Fig. 8). Hence, this area evaluation includes the surroundings of the FSM, giving a value of

$$A_3 = [(k_1(ni + ns) + k_2\,ns + k_3\,ns + k_5) \times (k_0\,tp + k_3\,ns + k_4)] \quad (12)$$

for the dynamic case, and

$$A_4 = [(k_1(ni + ns) + k_2\,ns + k_3\,ns + k_5) \times (k_0\,tp + k_4)] + KC + KR \quad (13)$$

for the case of using a static register.

4.3 Comparing the area occupation figures

The different area evaluations considered above can be applied to determine the cost of a given FSM. Of course,

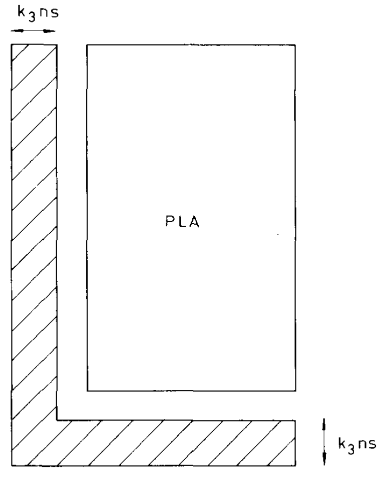

Fig. 8 *PLA area including surroundings of FSM*

they are orientative and we will use them here to show that the conclusions we reached when eqn. 1 is employed are confirmed even for more-precise area evaluations. For any FSM, we can plot all of those estimates as we have done in Fig. 9. The value of the occupied area can

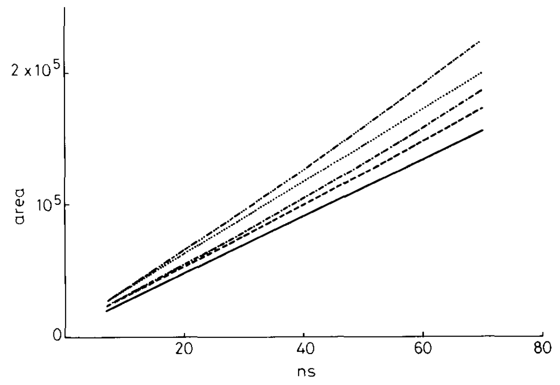

Fig. 9 *Area occupation for every case considered in text*
$ni = 15$, $ns = 80$, $\sigma = 0.6$
——— Eqn. 1
- - - - Eqn. 10
—·—· Eqn. 11
······· Eqn. 12
···–··· Eqn. 13

be seen to increase monotonically with *ns* and remain in the same relative position for the five different expressions we have derived. As a consequence, a procedure similar to that applied in Section 3 can be followed, hence attaining a maximum value for *ns* beyond which any minimal cardinality coding is not area-efficient. The actual maximum will depend on the expression we consider to be more accurate for evaluating the silicon area occupation.

However, concerning the value of ns_{max}/ns_{min}, (i.e. the maximum number of state variables a minimal cardinality assignment can have for being efficient), the more restrictive estimate does not always correspond to the same expression. This can be illustrated by Fig. 10 where three of those area measures have been plotted. In the Figure, it should be clear that the higher value for ns_{max}/ns_{min} sometimes corresponds to eqn. 1 and sometimes to eqn. 10. In the particular example of Fig. 10

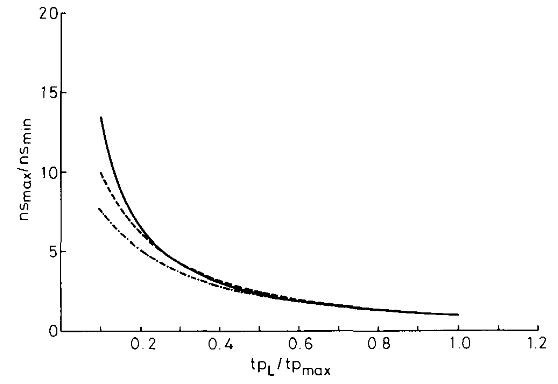

Fig. 10 *Evolution of ns_{max}/ns_{min} for minimal cardinality assignment*
$ni = 15$, $ns = 80$
——— Eqn. 1
- - - - Eqn. 10
·—·— Eqn. 11

there is not a big difference, but it suggests care if an accurate evaluation of ns_{max} is required.

Nevertheless, what is shown in Fig. 10 is very representative of the behaviour we have observed for the different area estimations. Only for low values of tp_L/tp_{max}, is there significance in the particular model we use for representing the area. This is especially true when a static register bank is employed, since in this case the value for ns_{max} can be further reduced.

5 Conclusions

In this paper we have given a critical view of the reported methods for performing the state assignment process in a FSM to be implemented by using a PLA. Generally speaking, we have shown the need for carrying out an evaluation of the maximum number of state variables required for any coding procedure using a minimal cardinality (or quasiminimal) assignment. Such an evaluation has to be performed before the coding since it might reduce the CPU time to be spent in the process. A method has been given to check whether a minimal cardinality assignment is cost-effective or not. Also we have proposed a few alternative expressions for evaluating the silicon area occupation for PLA-based sequential machines. It has been proved that our checking method can be applied independently of the area estimate to be used.

6 References

1 SOBOL, R.: 'The universal synchronous machine'. VLSI Design, pp. 60–66, Nov. 1983
2 AGRAWAL, P., and MEYER, M.J.: 'Automation in the design of finite state machines'. VLSI Design, pp. 74–84, Sept. 1984
3 BROWN, D.W.: 'A state-machine synthesis-SMS'. Proc. 18th Design Automation Conference, Nashville, TN, 1981, pp. 301–305
4 MEYER, M.J., AGRAWAL, P., and PFISTER, R.G.: 'A VLSI FSM design system'. Proc. 21st Design Automation Conference, Albuquerque, NM
5 KANG, S.: 'Automated synthesis of PLA based systems'. PhD dissertation, Stanford University, 1981
6 PAPACHRISTOU, C.A., and SARMA, D.: 'An approach to sequential circuit construction in LSI programmable logic arrays', *IEE Proc. E, Comput. & Digital Tech.*, 1983, **130**, (5)
7 DE MICHELI, G.: 'Computer-aided synthesis of PLA-based systems'. PhD dissertation, Berkeley University, 1984

8 ACHA, J.I., and CALVO, J.: 'On the implementation of sequential circuits with PLA modules', *IEE Proc. E, Comput. & Digital Tech.*, 1985, **132**, (5), pp. 518–522

9 DE MICHELI, G., BRAYTON, R.K., and SANGIOVANNI-VINCENTELLI, A.: 'Optimal state assignment for finite state machines', *IEEE Trans. Comput.-Aided Des.*, 1985, **CAD-4**, pp. 269–284

10 DE MICHELI, G.: 'Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros', *IEEE Trans., Comput.-Aided Des.*, 1986, **CAD-5**, (4), pp. 597–615

11 DE MICHELI, G., SANGIOVANNI-VINCENTELLI, A., and VILLA, T.: 'Computer-aided synthesis of PLA-based finite state machines'. Proc. 1983 Int. Conf. on CAD, pp. 154–157

12 DE MICHELI, G., BRAYTON, R.K., and SANGIOVANNI-VINCENTELLI, A.: 'KISS: A program for optimal state assignment of finite state machines'. Proc. 1984 Int. Conf. on CAD, pp. 209–211

13 ARMSTRONG, D.B.: 'A programmed algorithm for assigning internal codes to sequential machines', *IRE Trans. Elect. Comput.*, 1962, **EC-11**, pp. 611–622

14 LIU, C.N.: 'A state variable assignment method for asynchronous sequential machines', *J. ACM*, 1963, **10**, pp. 209–216

15 TAN, C.J.: 'State assignments for asynchronous sequential machines', *IEEE Trans. Comput.*, 1971, **C-20**, (4), pp. 382–391

16 UNGER, S.H.: 'Theory of asynchronous sequential machines' (McGraw-Hill Book Company Inc., New York 1969)

17 QUINTANA-TOLEDO, J.M.: 'Una contribución al diseño automático de circuitos digitales usando PLAs'. PhD dissertation, Sevilla University, 1987 (in spanish)

18 CURTISS, H.A.: 'Tan-like state assignments for synchronous sequential machines', *IEEE Trans. Comput.*, 1973, **C-22**, pp. 181–187

19 KAMBAYASI, Y.: 'Logic design of programmable logic arrays', *IEEE Trans. Comput.*, 1979, **C-28**, (9), pp. 609–617

20 HACHTEL, G.D., NEWTON, A.R., and SANGIOVANNI-VINCENTELLI, A.L.: 'An algorithm for optimal PLA folding', *IEEE Trans. Comput.-Aided Des.*, 1982, **CAD-1**, (2), pp. 63–76

21 DE MICHELI, G., and SANGIOVANNI-VINCENTELLI, A.L.: 'Multiple constrained folding of programmable logic arrays: theory and applications', *IEEE Trans. Comput.-Aided Des.*, 1982, **CAD-2**, (3), pp. 151–167

Efficient realisation of discrete Fourier transforms using the recursive discrete Hartley transform

W.C. Siu
K.L. Wong

Indexing terms: Algorithms, Signal processing

Abstract: In the paper, we present the results of our study using a recursive discrete Hartley transform technique to compute discrete Fourier transforms. We also introduce an improved in-place and in-order prime-factor mapping to effectively realise composite-length DFTs. In using these new techniques, the speed of computation is comparable to that of the Winograd Fourier transform algorithm (WFTA), whereas the program size of the present approach is much smaller than that of the WFTA. This approach is most suitable for the cases where there are restrictions on program lengths.

1 Introduction

The discrete Fourier transform (DFT) is one of the most important tools in modern digital signal processing. Much research effort has been dedicated to its efficient realisations. The fast Fourier transform (FFT) [1] proposed by Cooley and Tukey reduces the number of multiplications from N^2 to $(N/2)(\log_2 N)$, where N is the transform length. Good [2] showed that a single dimension DFT can be converted to a multidimensional form if the transform length is composed of relatively prime factors. Winograd [3] and Kolba and Parks [4] made use of Rader's theorem [5] to construct efficient algorithms of this class. Compared with FFT, these latter approaches further reduce the number of multiplications by one-half to two-thirds. Siu and Constantinides [6], made use of Rader's algorithm to propose the very fast discrete Fourier transform (VFDFT) based on the number theoretic transform (NTT) which achieved having only one multiplication per point.

Recently, Siu [7] proposed a nesting algorithm for the VFDFT which recursively decomposed the discrete Fourier transform into short modules that just required two very efficient primary DFT modules of length 2 and 4. This paper extends this work by developing another efficient algorithm making use of the discrete Hartley transform (DHT) [8]. The present work solves the problem of a prime-factor mapping for DHT which is an essential step for the recursive decomposition. Sorensen *et al.* [9] proposed a modified prime-factor mapping technique based upon Good's algorithm [2] to construct composite-length DHT using prime-length modules. The control structure of this approach is complicated since data from different short transforms within the same stage are coupled by some extra additions. In this paper, we use a simpler but more general approach in which a recursive DHT technique is used to formulate a fast algorithm for the computation of DFT. An improved in-place in-order prime-factor mapping is also developed for the construction of long composite length transforms.

Paper 6710E (C1, C2), first received 26th July 1988 and in revised form 6th February 1989

The authors are with the Department of Electronic Engineering, Hong Kong Polytechnic, Hung Hom, Kowloon, Hong Kong

2 Recursive discrete Hartley transform

The discrete Hartley transform of a real data sequence $\{r(n): n = 0, 1, \ldots, N-1\}$ is defined as

$$Q(k) = \sum_{n=0}^{N-1} r(n) \operatorname{cas}(2\pi nk/N) \tag{1}$$

where $\operatorname{cas}\beta = \cos\beta + \sin\beta$.

Note that the DHT is structurally very similar to the DFT. Most importantly, it retains the cyclic property which was made used by Rader [5] and Siu and Constantinides [6] to convert prime-length DFTs into cyclic correlation and cyclic convolution forms, respectively. This enables the same conversion to be made on the DHT. More precisely, if $N = P$, where P is a prime number with a primitive root equal to g, eqn. 1 can be written as

$$Q(0) = \sum_{n=0}^{P-1} r(n) \tag{2}$$

and

$$Q_k = R_k + r(0) \tag{3}$$

where

$$Q_k = Q(\langle g^{k-1}\rangle_P) \tag{4}$$

$$R_k = \sum_{n=0}^{P-2} r_n A_{k-n} \tag{5}$$

$$A_n = \operatorname{cas}(2\pi\langle g^{n-2}\rangle_P/P) \tag{6}$$

$$r_n = r(\langle g^{-(n+1)}\rangle_P) \qquad \text{for } k, n = 0, 1, \ldots, P-2 \tag{7}$$

The expression $\langle C\rangle_P$ means the residue of the number C modulo P. Eqn. 5 is a length-$(P-1)$ cyclic convolution. A previous effort [7] made use of a slightly different permutation to convert a prime-length DFT into this form and evaluated the convolution based on length-$(P-1)$ DFTs. As the DHT also possesses convolution properties [8], eqn. 5 can be realised using length-$(P-1)$ DHTs.