An Algorithm for Area and Delay Optimization of Sequential Machines through Decomposition

Aurobindo Dasgupta
Dept. of Electrical and Computer Engg.
University of Massachusetts
Amherst, MA 01003

Israel Koren
Dept. of Electrical and Computer Engg.
University of Massachusetts,
Amherst, MA 01003

Presented by Wayne Burleson

Abstract

A new algorithm is presented for decomposing a Finite State Machine(FSM) to reduce the area and delay. For a known state encoding, it partitions the next state and output bits so as to decrease the cost, which is a function of the area and delay. A state encoding which yields a low cost is derived by using a heuristic algorithm. This algorithm differs from existing algorithms in that it uses a measure for the area that is more reflective of the actual area of the final chip. For most examples, the results obtained on a set of benchmarks are better than those of [1] and [18] .

1 Introduction

Sequential circuits play a major role in the control of digital systems and efficient tools for their design are essential for computer-aided design of VLSI circuits. Area and performance optimization involves the development of algorithms for state assignment[6, 12] and decomposition of finite state machines (FSMs) [1, 7, 8, 18, 19, 20].

High quality algorithms for state assignment, targeting two-level and multilevel logic implementations have been developed in [4, 12, 20] and their relationship to state assignment has been investigated in [5].

It is desirable to decompose FSMs into smaller interacting machines for a number of reasons:

1. The implementation of sequential circuits as interacting FSMs improves performance as a result of the reduction in the longest path between the latch inputs and outputs. This might improve the performance of FSM controllers, which often dictate the required duration of the system clock[1].

2. Decomposing an FSM into smaller interacting FSMs can lead to a decrease in chip area. This is partly due to the partitioning of the logic and partly due to the fact that both algorithms for state assignment and logic partitioning work better for smaller machines. In [1, 18, 7], the area is calculated as the sum of the areas of the smaller interacting FSMs.

3. Smaller machines may be required when there are restrictions on the number of product terms or on the number of input and output lines (for example, in PLA-based FPGAs). A technique to optimally create several smaller machines instead of one big machine is useful in these cases.

2 Layout Topology

Figure 1 shows the proposed layout of a decomposed finite state machine. Every input is available to all submachines. The states of each submachine are made available to the others if necessary. The outputs are generated directly by the individual submachines. Note that the state information is shared and that, in general, the submachines are not independent of each other. The submachines are implemented as PLAs. The area, which is the product of the width and height, also includes unused/wasted area of the chip.

3 Objectives of the Decomposition

The motivation for FSM decomposition is to optimise the performance and area. For a PLA implementation of the submachines, these measures may be estimated as follows:

- **Performance:** The performance of a circuit is the delay along the longest path. Based on the delay model presented in [7], the delay for a PLA implementation can be estimated as

$$D_{PLA} = max_{i,j}(f_{I_i} + f_{P_j})$$

where f_{I_i} is the fanout for the ith input line I_i and f_{P_j} is the fanout for the line corresponding to the jth product term.

Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences, 1994

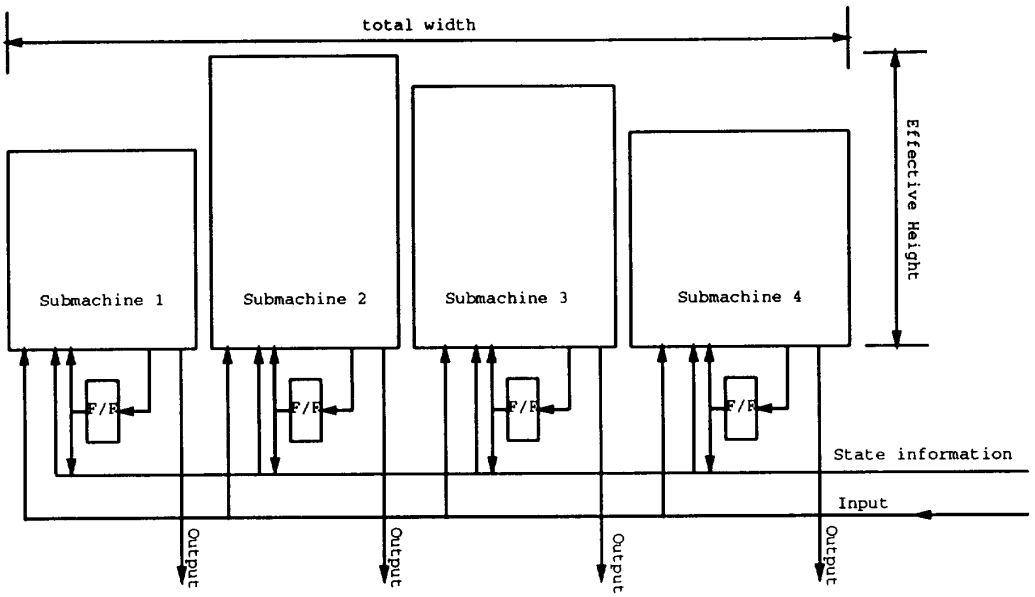

Figure 1: The proposed layout of the decomposed finite state machine.

Additional delay is due to the storage elements and the interconnections between the machines, and is not considered here. Since each submachine is implemented as a PLA, the submachine with the longest delay will determine the period of the clock or the delay of the decomposed FSM.

• **Area:** The measure used for calculating area in [1], [7] and [18] is the sum of the areas of the individual submachines. We estimate the area of the decomposed FSM according to the layout shown in Fig. 1. This assumes that the submachines of the FSM are placed next to each other. Approximating the area as that occupied by the AND-OR planes of the constituent submachines yields

$$Area = P_{max} \times width$$

where P_{max} is the maximum number of product terms among all the submachines and
$width = \Sigma(I+O)$ over all submachines where I and O are the inputs and outputs, respectively, in which each includes the state bits(present state bits for I and next state bits for O). If the submachines are placed next to each other according to the layout in Figure 1 then this proposed measure is more reflective of the actual area of the chip than those of [1], [7] and [18] because it includes the unused(wasted) area. This measure for area would be important when the submachines of the FSM are not distributed in a chip to be closer to the logic that they control. The justification for this measure for area appears in Appendix A.

4 Partitioning the state and output bits

An algorithm is proposed to distribute the next state and output bits among the submachines with the primary goal being to reduce some function of the area and delay for the decomposed FSM. This is achieved by attempting to decrease the total number of product terms in all submachines simultaneously while keeping the number of submachines small. In this method, FSM decomposition is done in two steps. The first step is to estimate the number of product terms required for each of the next state and output bits, when

implemented as a single PLA. This corresponds to a decomposition where all submachines have two states each. This can be done using any 2-level logic minimiser (e.g., ESPRESSO [2]).

The next step is to combine the bits of the next state and output and assign them to submachines so that the maximum number of product terms among the submachines is minimised. This method ensures that the empty or wasted area in the layout is minimised. The choice of the bits allocated to a submachine will depend on the number of product terms required for each bit, as calculated in the first step. The number of product terms in the resultant submachine can never exceed the sum of the product terms associated with each bit of the next state and output which is assigned to that submachine.

The pseudo code for the algorithm for assigning state and output bits to submachines and calculating the cost is shown below.

```
Calculate_Cost()
{
Find minimum Cost for number_plas = 1 to number of state and output bits
 {
  Distribute_State_And_Output_Bits(number_plas)
  Effective_Height = height of submachine with maximum number of product terms
  Effective_Width = (2 · number_plas · (# of primary inputs+# of state bits)) + (# of state bits + output)
  Area = Effective_Height · Effective_Width
  Calculate delay of FSM
  Cost = Area · Delay^(Delay weight/Area weight)
  }
}

Distribute_State_And_Output_Bits(number_plas)
{
average_height
=(Sum of pdt. terms of all st. & out. bits when implemented separately) ÷ number_plas
No bit is assigned to any pla.
While there exists an unassigned state bit and output bit
 {
 Free_pla=pla with maximum free space(average height-prod_terms assigned to that pla)
 Assign the bit which reduces the most free space to Free_pla
 }
}
```

We have used a greedy approach to arrive at a solution in procedure Distribute_State_And_Output_Bits. This gives a reasonably good solution. To get a better solution an approximation algorithm could be used but it would require a higher execution time. This is because the algorithm would have to search different solutions (every possible assignment of the bits to submachines) to arrive at a result with minimum overall area. This problem is similar to the *Bin Packing* problem (which is NP-hard) [13].

Example 1: Consider an FSM with 1 bit, 3 bits and 2 bits for coding the input, state and output, respectively. Let the bits for the next state and the output be represented as $s0, s1, s2$ and $o1, o2$, respectively. Let the minimum number of product terms associated with each bit when implemented as a PLA be as shown in the table below.

Bit position	Number of Product terms
s0	5
s1	7
s2	8
o1	6
o2	5

If bits $s0, s1$ and $o1$ are to be computed in one PLA then the minimum number of product terms can never exceed 5+7+6 = 18, which is the sum of the minimum number of product terms required to implement $s0, s1$ and $o1$ separately in individual PLAs. We choose the configuration which partitions the bits such that the upper limit of the area calculated, using the method just described, is minimum. The total number of next state and output bits is 5, the value for each bit being calculated in exactly one submachine. The total number of lines, true and inverted, entering the input plane of each PLA is $2*(1+3) = 8$. Hence the width of the decomposed FSM is $8*n+5$, where n is the number of interconnected FSMs (submachines) which are implemented as PLAs. The area upper bounds ($(8*n+5)*$ *upper limit of product terms*) for decompositions corresponding to different numbers of submachines are shown in Table 1.

We choose the decomposition that corresponds to the least upper bound for the area. The exact area of this decomposition is then calculated by minimizing the two-level logic of each of the submachines. The submachine with the largest number of product terms determines the effective height (see Figure 1) of the decomposed FSM. In this example, decomposing the

Table 1: Area upper bounds for different decompositions of the FSM in Example 1.

Number of submachines	Partition of next state & output bits	Area Upper bound
1	31 $(s0, s1, s2, o1, o2)$	31×13=403
2	18 $(s0, s2, o2)$ 13 $(s1, o1)$	18×21=378
3	8 $(s2)$ 12 $(s0, s1)$ 11 $(o1, o2)$	12×29=348
4	8 $(s2)$ 7 $(s1)$ 6 $(o1)$ 10 $(s0, o2)$	10×37=370
5	8 $(s2)$ 7 $(s1)$ 6 $(o1)$ 5 $(s0)$ 5 $(o2)$	8×45=360

FSM into three interacting FSMs with the assigned bits $s2$ and $s0, s1$ and $o1, o2$, respectively, results in the least upper bound of the area for the chip. This is how we determine the decomposition for an FSM with a given state encoding.

5 A Heuristic Algorithm to Obtain a Good Encoding

Given a method which arrives at the decomposition for a known state encoding, the problem of arriving at a state encoding which minimises a cost function which depends, in general, on area and delay, is of exponential complexity. To arrive at a state encoding which minimises the value of the cost function, we will have to calculate the cost function for every possible state assignment. We use the minimum code length for encoding the states of the FSM. The minimum code length required for an FSM with N states is $\lceil \log_2 N \rceil$ bits. $2^{\lceil \log_2 N \rceil}$ is the total number of codes that could be assigned to a single state. Therefore, the number of different possible state assignments for an FSM with N states is $2^{\lceil \log_2 N \rceil}!/(2^{\lceil \log_2 N \rceil} - N)!$, which is an exponential function. To find the optimal solution to this problem would be intractable. Hence, we have decided to use a heuristic algorithm to obtain a good, though not necessarily optimal, state encoding. A heuristic algorithm that seems to be well suited for this problem is simulated annealing [9, 17].

The simulated annealing algorithm starts with an initial solution and then examines another solution (not necessarily a neighboring solution) with the purpose of reducing the cost. It also allows occasional "uphill moves" (moves that worsen the current solution) in an attempt to reduce the probability of being stuck at a locally optimal solution. These uphill moves are controlled probabilistically by a parameter called the temperature T, and become less and less likely toward the end of the annealing process, as the value of T decreases. In our algorithm a specific state encoding corresponds to a solution in the solution space. The solution space consists of configurations (solutions) which correspond to all the different possible state encodings.

5.1 Annealing Schedule

Our temperature schedule is of the form $T_k = r \cdot T_{k-1}, k = 1, 2, 3....$ A typical value of r, the temperature reduction rate, is 0.85. The initial temperature T_0 is determined by performing a sequence of random moves and computing the quantity Δ_{avg}, the average value of the magnitude of the change in cost per move. The probability of acceptance is $\exp^{-\Delta/T}$, where Δ is the change in cost per move at temperature T. The algorithm can start at any initial encoding which can be far from optimal.

At each temperature, enough moves are attempted until either there are M downhill moves or the total number of moves exceeds $2M$, where $M = k^2$, k being the minimum number of bits required to encode the states. We found through repeated experiments that choosing a function like $M = k$ resulted in too few moves and a function like $M = k^3$ required a very large number of moves for any particular temperature. Too few moves would result in very few configurations being investigated, lowering the quality of the result, and a large number of moves would increase the execution time of the program excessively. The annealing process is terminated [17] when the number of accepted moves is less than 5% of all moves made at a certain temperature or the temperature is sufficiently low. Experimental results on the effect of changing the annealing parameters and a justification for the final values of the parameters that are chosen are given in Appendix B.

6 Results

The algorithm described above was implemented as a C program. The algorithm was tested on a number of examples obtained from the MCNC FSM benchmark suite [11]. The motivation for FSM decomposition was elucidated in some detail in the previous

sections. Based on the properties desired, the efficacy of a decomposition algorithm can be judged from the following criteria.

The comparison between the areas of the two-level implementation of the encoded submachines and the two-level implementation of the prototype machine (using NOVA [16]) presented in Table 2.

PI denotes the number of Primary Inputs (Input bits),
S denotes the number of States,
PO denotes the number of Primary Outputs (Output bits),
P denotes the number of Product terms.
Area1 (in column 10) is the area calculated using the cost function in section 3
Area2 (in column 11) is the area in Area1 + the routing area. This was calculated by generating the layout of the FSM.
The cost function used here is the *area* of the decomposed machine. The area and delay of the prototype machine were derived using the state encoding obtained from NOVA [16]. The delay was calculated using the method described in [7]. As expected, the delay of the decomposed machine is much less than the delay of the prototype machine in all examples. The area of the decomposed machine is the smallest rectangle enclosing all the submachines when they are placed next to each other. This area which is more reflective of the actual chip area was compared to the area (using NOVA) of the prototype machine. We assume that, in general, all input bits are required in every submachine because it is unlikely for any next state or output bit to be independent of an input bit. Hence, the effective width of the layout of a decomposed machine will be larger for an example with a relatively large number of input bits compared to the number of state bits, thus resulting in a larger area when decomposed. This explains why in examples like *s1a*, *ex4* and *ex6* where the number of input bits is greater than the number of state bits, we found that the area of the best decomposed machine is greater than the area of the prototype machine as calculated using NOVA [16]. This area is not shown in Table 2 because in all such examples the program discovered a state encoding which when used to calculate the area for a single undecomposed machine would result in a smaller area than the area of the prototype machine, which is shown in Table 2. The algorithm in the program can very well be expected to arrive at a decomposition, if it exists, with less area than the prototype machine.

Our program also investigates the decomposition based on the encoding derived from NOVA. In all the examples, in Table 2, our algorithm found an encoding different from that derived using NOVA because the encoding resulted in a lower cost function. In four examples (*dk15*, *ex6*, *lion* and *train4*) the areas were equal to that derived from NOVA but the delay of the FSM using our encoding was less.

In Table 3 we show the results of our program when using the cost functions $Area \times Delay^{0.25}$ and $Area \times Delay^{0.5}$. As expected, the FSMs tend to decompose (decreasing the delay) when the weight for the *Delay* is increased relative to the *Area*. In these cases too, *Area* includes the unused or wasted area of the decomposed FSM. When using the cost function $Area \times Delay^{0.5}$ we found that in most examples the FSM was decomposed into submachines with one output each (i.e, two-state submachines). In this case, the FSM cannot be decomposed further and its delay cannot be reduced anymore. So using other cost functions like $Area \times Delay$ where the ratio of the exponents of the *Delay* to *Area* is greater than 0.5, may not produce results which are better than $Area \times Delay^{0.5}$.

The comparison of the results of our algorithm with two recent works [1] and [18] in this field is shown in Table 4. The product terms and delay estimates were not available in [1] for comparison.The area and delay in most examples show an improvement in our algorithm. It should be noted that the area, in column 4 and 6, is defined in [1] and [18] as the sum of the areas of the submachines only, whereas in our definition, the area, in column 2, also includes the unused or wasted area. If we had considered only the sum of the areas of the submachines, then our results would have been even better. An estimate of the unused/wasted area using the implementation in [18] can be made from the product terms of the decomposed submachines shown in column 5. The average percentage improvement of our result when compared to the results of [1, 18] are also shown.

The delay of an FSM is determined by the longest delay among the delays of the submachines. Our algorithm tries to ensure that all submachines have a nearly equal number of product terms (by minimizing wasted area) as much as possible. This forces the values of the delays of the submachines to be close to each other. This might not be true in other schemes where the resultant FSM has submachines such that the difference between the number of product terms is high.

Table 2: Comparison of the encoded prototype machine and our results.

Example	Prototype FSM				NOVA [16]		Our Results(cost function is *Area*)				
	PI	S	PO	P	Area	Delay	P	Submachine area	Area1	Area2	Delay
bbara	4	10	2	26	572	15	9/9/8	162/162/144	468	586	2
bbtas	2	6	2	13	195	11	5/5	60/65	125	148	3
beecount	3	7	4	12	228	13	10	190	190	190	7
dk14	3	7	5	30	600	18	26	520	520	520	8
dk15	3	4	5	18	306	13	18	306	306	306	7
dk27	1	7	2	8	104	7	3/3/3	27/30/30	87	98	2
dk512	1	15	3	19	323	7	10/10	130/140	270	303	4
ex3	2	10	2	18	324	13	17	306	306	306	6
ex4	6	14	9	14	465	15	13	429	429	429	13
ex5	2	9	2	15	270	20	12	216	216	216	6
ex6	5	8	8	25	675	18	25	675	675	675	11
ex7	2	10	2	17	612	19	10/10	150/150	300	341	3
lion	2	4	1	6	66	5	6	66	66	66	3
lion9	2	9	1	15	153	6	7	119	119	119	5
modulo12	1	12	1	15	585	8	5/5	65/60	125	146	3
s1a	8	20	0	65	2015	65	53	1643	1643	1643	6
shiftreg	1	8	1	8	216	5	1/1/1/1	9/9/9/9	36	42	1
styr	9	30	10	94	4042	63	87	3741	3741	3741	15
tav	4	4	4	11	198	6	2/2/2/2/2	26/26/26/26/28	132	166	2
train4	2	4	1	6	66	6	6	66	66	66	3
Avg. % Redtn									22.07	18.06	58.23
% STD. DEV.									23.43	22.71	21.24

This is why our FSM has a smaller delay than [18]. In only one example *s1a*, the delay is larger because our results in Table 4 were calculated with area as the only cost function. However in Table 3 with the cost function $Area \cdot Delay^{0.5}$, an encoding was found such that the delay of the FSM *s1a* is 1 (the lowest possible). In this case decomposition consists of 5 submachines.

Comparison of the area of the decomposed machine with that of the prototype machine, when implemented in multilevel logic is shown in Table 5. The multilevel area of the prototype machine is calculated for the state encoding obtained from NOVA [16](which targets two level implementations) , MUSTANG [4] and JEDI [10](which targets multilevel implementations). The multilevel area in column 2 is calculated for the decomposed machine which uses a state encoding (derived using our algorithm described in the previous sections) intended for a good two-level implementation. The multilevel areas were calculated using MISII [3]. An improved script [14] was used for logic minimisation. A standard cell layout was then obtained separately for each submachine and the prototype machine via the program WOLFE [15]. The decomposed machines were then laid out so that the area of the smallest enclosing rectangle is minimum. This is the area shown in column 2. This area too, thus includes the unused or wasted area. The areas shown in column 4, 6 and 8 are the multilevel areas of the prototype machine using NOVA, MUSTANG and JEDI. We have then shown the percentage area improvement (column 10) and delay improvement (column 11) when compared to the best area and best delay using NOVA or MUSTANG or JEDI, respectively. In five of the ten examples the multilevel area for the decomposed machine was less than the multilevel area of the prototype machine. There was an improvement in the delay in all but one case. The delay was calculated using a gate delay model (every gate in the circuit was associated with a rise time delay and a fall time delay). The proposed algorithm, though designed for a good decomposition in two-level logic, also results in a good decomposition in multilevel logic.

The figures for delay in Tables 1-4 were derived using the method in [7] so that it is possible to compare our results to that in [18]. The method in [7] is used there. The figures for delays in Table 5 have different units and were calculated using MISII[3]. This is because the method in [7] can be used for PLAs only and not for multilevel circuits.

Table 3: Results obtained for two different cost functions.

Example	$Area \times Delay^{0.25}$			$Area \times Delay^{0.5}$		
	Number of Submachines	Area	Delay	Number of Submachines	Area	Delay
bbara	3	468	2	6	612	1
bbtas	4	135	2	4	135	2
beecount	5	268	2	7	364	1
dk14	6	640	2	8	728	1
dk15	4	376	2	6	462	1
dk27	5	90	1	5	135	1
dk512	4	329	2	7	385	1
ex4	7	765	2	9	772	2
ex5	6	234	6	6	390	1
ex6	1	675	11	6	1177	2
ex7	6	390	1	6	390	1
lion	3	81	1	3	81	1
lion9	3	159	2	5	195	1
modulo12	4	135	2	4	188	1
s1a	1	1984	6	5	3381	1
shiftreg	3	56	2	4	36	1
tav	6	156	1	6	156	1
train4	3	56	2	3	81	1
AVERAGE % REDUCTION		8.76	77.58		-15.23	88.16
% STD. DEV.		32.59	13.19		44.88	5.55

7 Conclusion

Described in this paper is an approach to decomposing finite state machines to reduce area and delay. Given a state encoding, it indicates how to group the next state and output bits to minimize area and delay. A heuristic is then proposed to arrive at a state encoding which minimizes the cost function. Results for both 2-level and multilevel implementations of the decomposed machines have been compared to that of prototype machines. Comparisons with [1] and [18] have been made. An estimate for the area which is more reflective than previous works of the actual chip area has been made. The execution time of the program ranges from a few minutes to 100 CPU minutes on a DEC station 5000. Methods to reduce the execution time by tuning the parameters of the simulated annealing algorithm while maintaining the quality of the results are being investigated (some directions of investigation have been explained in Appendix B). The execution times in [1] and [18] were not available for comparison.

8 Acknowledgements

This work was supported in part by a grant from NSF under contract number MIP-9013013. Discussions with Wayne Burleson and Maceij Ciesielski are gratefully acknowledged.

References

[1] P. Ashar, S. Devadas and A. R. Newton, "Optimum and heuristic algorithms for an approach to Finite State Machine decomposition", *IEEE Trans. Computer-Aided Design*, Vol.10, March 1991.

[2] R. K. Brayton, G. D. Hatchel, C. T. McMullen and A. Sangiovanni-Vincentelli, *Logic Minimization algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[3] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A. Wang,"MIS: A multiple-level logic optimization system", *IEEE Transactions on Computer-Aided Design*, Nov. 1987.

[4] S. Devadas, H-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State assignment of finite state machines targeting multi-level logic implementations", *IEEE Trans. Computer-Aided Design*, vol. 7, pp 1290-1300, Dec. 1988.

Table 4: Comparison of our results with [1, 18].

Example	Our Results		Result of [1]	Result of [18]		
	Area	Delay	Area	Prod. terms	Area	Delay
dk27	87	2	85	4/4/3	110	4
dk512	270	4	342	5/5/7/6	276	5
s1a	1643	31	1862	29/29/44	3060	26
ex4	429	13	453	-	-	-
modulo12	125	3	161	-	-	-
styr	3741	15	4317	-	-	-
tav	132	2	150	-	-	-
ex3	306	6	-	6/13	298	11
bbara	468	2	-	9/9/9	495	8
beecount	190	7	-	7/6/3	237	7
AVERAGE % IMPROVEMENT			7		14.73	28.54
% STD. DEV.			7.94		16.77	31.81

Table 5: Comparison of Multilevel Areas.

Circt.	Decmpd. machine Area	Delay	Prototype FSM Area (NOVA)	Delay	Prototype FSM Area [4]	Delay	Prototype FSM Area (JEDI)[10]	Delay	% Area	% Delay
bbtas	72896	6.99	74296	8.64	98392	9.50	66096	8.8	-9.30	19.10
dk14	230880	8.39	245496	21.10	229392	21.62	261232	32.4	-0.60	60.24
dk512	162240	5.29	213776	22.45	242104	16.31	182736	12.8	12.63	67.57
ex5	172640	6.93	209808	17.47	153032	17.35	172640	14.8	-12.80	53.17
lion	45312	4.41	46176	7.07	58368	7.25	31744	6.0	-29.90	26.50
lion9	64152	5.14	88000	11.88	81168	11.88	50760	4.4	-20.87	-16.80
modulo12	62952	4.63	91168	10.44	70448	11.01	96120	6.2	10.70	33.90
s1a	615760	9.33	1011096	34.21	936600	40.68	565640	34.6	-8.13	72.73
styr	1734792	24.71	1705072	49.83	1747000	64.06	1634800	64.0	-5.70	50.42
ex4	165784	5.58	186120	11.61	203832	13.90	213776	16.6	11.00	51.94
Average	332741	8.14	387101	19.47	382034	21.4	327554	20.1	-5.29	41.87
Std. Dv.	493342	5.73	513756	12.75	516776	16.8	459775	17.7	13.39	25.52

[5] S. Devadas and A. R. Newton, "Decomposition and factorization of sequential finite state machines", *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 1206-1217, Nov. 1989

[6] J. Hatmanis and R. E. Stearns, *Algebriac Structure Theory of Sequential Machines*, Prentice Hall, Eaglewood Cliffs, NJ,1966.

[7] Z. Hasan and M. J. Ciesielski, "FSM decomposition for performance optimization", Technical Report TR-91-CSE-14, Department of Electrical & Computer Engineering, University of Massachusetts, Amherst,1991.

[8] F. C. Hennie, *Finite State Models for Logical Machines*, Wiley, New York, 1968.

[9] S. Kirkpatrick, C. D. Gelatt Jr., and M.P.Vecchi, "Optimization by simulated annealing", Science, 220(4598), pp 671-680, 1983.

[10] B. Lin, A. R. Newton, "Synthesis of Multiple level logic from Symbolic High-Level Description Languages", *Proc. IFIP Int. Conf. on Very Large Scale Integration (VLSI'89)*, pp. 187-196, North-Holland, Amsterdam, 1990.

[11] R. Lisanke, ed. "FSM Benchmark suite", Microelectronics Center of North Carolina, Research Triangle Park, NC, 1987.

[12] G. De 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.

[13] T. H. Cormen, C. E. Lieserson and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Company, 1990.

[14] H. Savoj, H. Wang and R. K Brayton, "Improved Scripts in MIS-II for Logic Minimization of Combinational Circuits", *International workshop on logic synthesis*, MCNC, 1991.

[15] C. Sechen and A. Sangiovanni-Vincentelli, "The Timber Wolf Placement and Routing Package", *IEEE Journal of Solid State Circuits*, April 1985.

[16] T. Villa and A. Sangiovanni-Vincentelli, "NOVA:State assignment of finite state machines for optimal two-level logic implementations", *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 905-924, Sept.1990.

[17] D. F. Wong, H. W. Leong and C. L. Liu, *Simulated Annealing for VLSI Design.* Kluwer Academic Publishers, 1988.

[18] M. K. Yajnik and M. J. Ciesielski,"Finite State Machine Decomposition using Multiway Partitioning", *International Conference on Computer Design*, 1992.

[19] M. Yeoli,"The cascade decomposition of sequential machines", *IRE Trans. Electron Computing*, pp 587-592, Apr. 1961.

[20] H. P. Zeiger,"Loop-free synthesis of finite state machines", Ph.D dissertation, Dept. of Electrical Engineering, MIT, Sept. 1964.

Appendix A: Justification of the area measure

The area measure used in this paper (in Section 3) is based on the layout of the submachines shown in Figure 1. If the submachines are distributed in a chip to be closer to the data flow logic that they control, then the area measure of [1], [7] and [18] would be suitable. However, in our results (in Section 6) we have shown that using our algorithm on benchmarks, the area (total submachine area + wasted area) using the area measure in Section 3 is less than the area (total submachine area) in [1], [7] and [18]. So even when the submachines are distributed in a chip the encoding generated by our algorithm has yielded good results.

Also, distributing the submachines of the FSM such that they are closer to the logic that they control will increase the routing area because all the state bits (present state bits) and the primary inputs which are required by all submachines will then have to be made available to the submachines in different parts of the chip. For a layout like that shown in Figure 1 the routing area is considerably less than in [1, 7, 18] because the submachines are close to each other.

The table below shows the fraction of the routing area in some examples using our algorithm and the proposed layout. Primary inputs and state bits are required by all submachines and outputs are not. We would expect those FSMs with a large number of primary inputs and state bits compared to primary outputs to have a greater percentage for the routing area. This would also depend on the complexity of the Boolean functions that realize the primary outputs and the next state bits. We have calculated the fraction of the routing area using our encoding algorithm (for the cost function $Area \cdot Delay^{0.5}$, see Table 3) and the layout of Figure 1 on a few benchmark examples. The fraction of the routing area was calculated from the layouts of these examples. In all these examples the fraction for the routing area was for the worst case since the FSM was decomposed into the maximum possible number of submachines. The results are shown below.

Circ.	I	State bits	O	Number of Submch.	% routing area
s1a	8	5	0	5	15
dk15	3	2	5	6	13
ex7	2	4	2	6	16
tav	4	2	4	6	20

As in the calculation of area in [1], [7] and [18], our calculated area does not include the routing area. However, in our scheme, since the layout of the submachines are such that they are always next to one another, our scheme probably requires the smallest fraction for the routing area. The actual area (including the routing area) is 1.25 times (for the layout in Figure

1) the area estimated using our equation in the worst case for the examples shown above.

Decomposition may not reduce the area of FSMs if the number of inputs is large when compared to the number of outputs. Let IS be the total number of inputs (complemented and uncomplemented primary inputs and state bits)in the prototype machine, OS be the total number of outputs(number of primary outputs and state bits) in the prototype machine, $Prod_{pr}$ be the number of product terms in the prototype machine and $Prod_{D2}$ be the maximum number of product terms when the machine is decomposed into 2 submachines. If decomosition is uneconomical with the area as the cost function, then

$$(I+O)\cdot Prod_{pr} < (2\cdot I+O)\cdot Prod_{D2}$$

$$\Rightarrow \frac{Prod_{pr}-Prod_{D2}}{Prod_{D2}} < \frac{I}{I+O}$$

If $I \gg O$ then it is likely that the condition is satisfied which would result in a decomposition with a larger area when compared to the prototype machine.

Appendix B: Experiments on the parameters of Simulated Annealing

The tables above show the effect of change in the temperature reduction rate (r) and the number of downhill moves allowed at any temperature (M) on the Area and Execution time for four examples.

For higher values of r (the decrease in temperature is smaller) more state assignments are searched (the program goes through more levels of temperatures) which requires a longer execution time and yields a lower area. This is supported by the results shown in Table 6. A deviation from the expected behavior was found in the example *shiftreg*. This anomalous behavior could be attributed to the fact that *shiftreg* has a simple and regular state transition table.

When M increases we would expect longer execution times and smaller areas because a greater number of state assignments are searched at every temperature. This was contrary to the results we observed (shown in Table 7). In these cases a higher value of M yielded a low area state assignment(high powers of k, close to the initial high temperature) which terminated the algorithm at the next lower temperature. The number of accepted moves was less than 5% of all moves at that temperature [17]. This also resulted in a decrease in execution times for higher values of M.

Table 6: Effect of change in temperature reduction rate (r) on the area and execution time

Example	AREA			EXEC. TIME		
	0.80	0.85	0.90	0.80	0.85	0.90
bbtas	135	135	125	549	573	940
beecount	209	190	190	427	486	749
dk27	91	87	87	427	486	749
shiftreg	56	36	56	421	394	616

Table 7: Effect of change in the number of downhill moves allowed (M) at any temperature on the area and execution time

Example	AREA			EXEC. TIME		
	k	k^2	k^3	k	k^2	k^3
bbtas	140	140	135	582	573	267
beecount	209	190	190	724	486	427
dk27	104	87	91	494	486	192
shiftreg	36	36	56	392	394	204

Table 8: Effect of change in the value of p on the area and execution time

Example	AREA			EXEC. TIME		
	0%	2%	5%	0%	2%	5%
bbtas	140	140	140	594	577	560
beecount	209	209	209	1108	1074	1057
dk27	87	87	87	523	504	488
shiftreg	36	36	36	403	395	368

Table 8 shows the effect of a change in the parameter of the terminating condition, p, which is the percentage such that the program will stop when the fraction of the number of moves accepted over all moves at a particular temperature is less than p. When p increases the execution time should decrease and the area should increase. But for small values of p like in Table 8 the area will not change appreciably. This is because the algorithm has found a low value for the area just before termination for small values of p.

The figures for the areas shown in the tables were the figures that appeared in 50-70% of the number of times (10-15 times) the program was run. The values for r, M and p which resulted in the greatest reduction in cost (in this case $AREA$) without high execution times were chosen. The values that we chose for r, M and p were 0.85,k^2 and 5%, respectively.