GALLOP: Genetic Algorithm based Low Power FSM Synthesis by Simultaneous Partitioning and State Assignment

Ganesh Venkataraman [*] Sudhakar M. Reddy [*]
ECE Department
University of Iowa
Iowa City, IA 52242
gvenkata@engineering.uiowa.edu
reddy@engineering.uiowa.edu

Irith Pomeranz [†]
School of ECE
Purdue University
West Lafayette, IN 47907
pomeranz@ecn.purdue.edu

Abstract

Partitioning has been shown to be an effective method for synthesis of low power finite state machines. In this approach, an FSM is partitioned into two or more coupled sub-machines such that most of the time only one of the sub-machines is active. In this paper, we present a GA based approach for simultaneous partitioning and state assignment of finite state machines with power reduction as the objective. Experimental results obtained compare favorably with previous works on FSM partitioning, low power state assignment as well as using GA for partitioning alone.

1. Introduction

Low power synthesis has gained significant attention in recent years. It is well known that state assignment has a significant impact on both area and power dissipation of Finite State Machines (FSMs). Early works on state assignment [2, 3] concentrated on area minimization. Recent works have investigated low power designs [4]-[10]. In [5], GA was used for both state encoding and selection of flip-flops (D or T).

Partitioning has been shown to be a very effective technique for reducing power in FSMs [6]-[9]. Partitioning decomposes a given FSM into two or more coupled sub-machines. When one submachine is active, the inputs to the other sub-machines are turned off to reduce power dissipation. Hence, except when there is a transition between two different sub-machines, only one submachine is active at a time. Synthesis of FSMs for low power using partitioning is done in two steps. The first step finds a good partition and the second step finds good state assignments for the coupled sub-machines. Different optimization procedures are used to obtain partitions [6]-[9]. In [7] genetic optimization is used to derive the desired partition. After partitioning, known state assignment procedures were used to synthesize the FSM.

It is important to note that low power design procedures briefly reviewed above as well as the procedure presented in this paper achieve low power dissipation at the expense of increased circuit area.

The main contribution of the work reported in this paper is the *concurrent* consideration of partitioning and state assignment. We use a GA based approach in which partitioning and state-assignment are considered simultaneously. Experiments on the MCNC91 benchmark circuits show that the proposed approach yields designs that dissipate lower power than those obtained by earlier procedures. Thus, the proposed procedure can be used in power-constrained designs when other procedures do not yield suitable circuits.

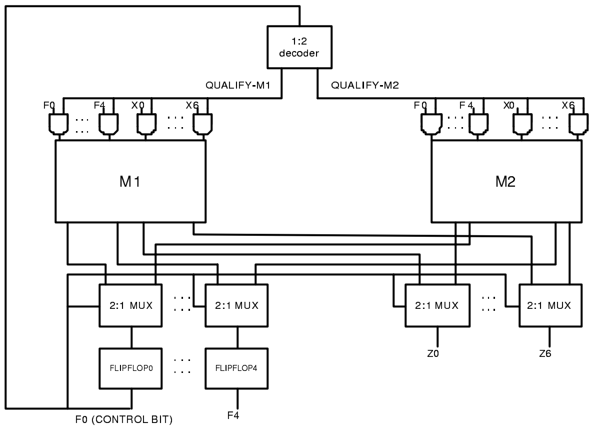
2. FSM Partitioning

A good description of the methods used to realize a partitioned FSM is given in [9]. In this section, we briefly describe the structure of a low power partitioned FSM with the help of an example. We also discuss the model we use for estimating power dissipated in a partitioned FSM. Using this model, we derive the cost function that will be used in our GA formulation.

Our partitioning scheme partitions the set of states of a machine into two subsets implemented by two submachines. This is different from the classical notion of

[*] Research supported in part by NSF Grant No. CCR-0097905 and in part by SRC Grant No. 2001-TJ-949
[†] Research supported in part by NSF Grant No. CCR-0098091 and in part by SRC Grant No. 2001-TJ-950

Figure 1. **Circuit for the partitioned state machine**



decomposition where a state of the machine is obtained by concatenating the states of the two sub-machines. The partitioning used in this work can be viewed as a partitioning of the combinational logic of the sequential circuit into two or more sub-circuits each of which computes the sequential circuit outputs and next states for different state-transitions. Thus, in the proposed partition one does not see two separate sequential sub-circuits with their own sets of flip-flops. However, for the sake of simplicity, in the sequel, we do discuss the partition in terms of two sub-machines. We next give an example and illustrated in Figure 1 together with an explanation of how the partitioned circuit operates and reduces dissipated power.

Consider the MCNC91 benchmark FSM BBSSE. Let M denote the original machine. It has 7 inputs, 7 outputs and 16 states. Let S represent the set of states in BBSSE. Let this set be partitioned into two subsets: $S_1 = (s_{10}, s_{11}, s_{12}, s_{13})$ and $S_2 = (s_{20}, s_{21}, s_{22}, \ldots, s_{211})$ realized using two sub-machines M1 and M2. These two sub-machines are dependent on each other (or coupled) since there could be a transition from a state in one submachine to a state in the other submachine. Such transitions are referred to as *cross transitions*. A transition happening between two states within the same submachine is referred to as a *self transition*. Since there are 16 states, we need a minimum of 4 bits to encode them. In order to simplify the design of the control logic, 5 bits are used to represent the states with the most significant bit (MSB) acting as the control bit that distinguishes the states in S_1 from those in S_2.

Both M1 and M2 take the current state and the primary inputs as the input and compute the next state and the primary outputs. We say that a submachine is *active* if the current state is in the set of states of the submachine. If a submachine is not active we say that it is *idle*. In CMOS, power is dissipated whenever the output of a gate changes. Therefore, if we could hold the inputs to an idle submachine to a fixed value (thereby holding the outputs to a fixed value), then power would not be consumed in it.

We use the MSB of the encoded states as a control bit to indicate whether the machine is in a state in S_1 or S_2. Assume that when the control bit is 0 the machine state is in S_1 and when the control-bit is 1 the state is in S_2. A 1:2 decoder takes the control-bit as input and produces two outputs QUALIFY-M1 (equals 1 when control bit is 0) and QUALIFY-M2 (equals the value of the control bit). All the inputs (present state and primary inputs) to M1 are qualified along with QUALIFY-M1 via 2-input AND gates. A similar design is used for M2 employing QUALIFY-M2. The next state outputs that come from both M1 and M2 are connected to the same set of flip-flops via 2:1 multiplexers, which use the control-bit as their select input. Thus, when M1 is active, the control-bit is low and the next state output of M1 is fed to the flip-flops and the next-state output of M2 is ignored. The same process is carried out for the primary outputs as well. The entire circuit is shown in Figure 1 (here Z0-Z6 denote the primary outputs and X0-X6 the primary inputs). Note that the classical notion of decomposition implies that independent sets of flip-flops are used for the submachines. In this case, one can turn off the clock to the idle submachine by using the control bit. We did not employ this method in our experiments.

A sequence of events in the partitioned FSM occurs as following. Assume that we are currently at clock cycle T and M1 is active. The inputs to M2 are all held at zero. Consequently power is not consumed in M2. M1 computes the next state and the primary outputs. Suppose this happens to be a self transition, then the control-bit remains zero, M1 remains active and M2 stays idle in clock cycle T+1. If the transition happens to be a cross transition, M1 sets the control-bit high. Consequently, in clock cycle T+1, the inputs to M1 are held at zero and M2 becomes active receiving the primary inputs and the present state (which was computed by M1 and stored in the flip-flops during clock cycle T). Thus, after the new state is latched during clock cycle T, the inputs to M1 change from the inputs at the start of clock cycle T to all zeros and the inputs to M2 change from all zeros to the inputs for clock cycle T+1. Consequently during a cross transition both M1 and M2 see input changes and hence power is dissipated in both the machines.

Next we develop a model to estimate the power dissipated in the composite machine shown in Figure 1. We point out that we need this model only during the execution of the GA based partitioning and state assignment procedure we used. We will use a more accurate power estimation tool (available in SIS [15]) on the final solution we obtain. To develop the model, we identify four possible events based on the possible transitions and derive an estimate of the power dissipated during each one of these events. Let *Pr(Event-i)* and *Power(Event-i)* represent the probability of occurrence of event i and the power consumed when the event occurs, respectively. Let *P(s)* denote the steady state probability of the machine M being in state *s*, where $s \epsilon S$. In the following we assume that all the inputs to M are equally likely.

Event1: Represents a self transition within M1. For this to happen, the current state should $\epsilon\ S_1$ and the primary inputs are such that the next state also $\epsilon\ S_1$. Power is consumed in M1 and the control logic (CL). Let the sum of the probabilities of occurrences of all the input signals that will bring a state s_{1j} to any state s_{1k} be PIM_{1j}, then

$$Pr(Event-1) = \sum_{j=0}^{3} P(s_{1j}) * PIM_{1j} \tag{1}$$

Event2: Represents a self transition within M2. This is similar to Event-1 except that we are now dealing with M2.

$$Pr(Event-2) = \sum_{j=0}^{11} P(s_{2j}) * PIM_{2j} \tag{2}$$

Event3: Represents a cross transition from M1 to M2. Power is dissipated in M1, M2 and CL. The current state should $\epsilon\ S_1$ and the primary inputs are such that the next state $\epsilon\ S_2$. Let the sum of the probabilities of occurrences of all the input signals that will bring a state s_{1j} to any state s_{2k} be $PI^{'}M_{1j}$, then

$$Pr(Event-3) = \sum_{j=0}^{3} P(s_{1j}) * PI^{'}M_{1j} \tag{3}$$

Event4: Represents a cross transition from M2 to M1. This similar to Event-3 except that M1 and M2 are interchanged.

$$Pr(Event-4) = \sum_{j=0}^{11} P(s_{2j}) * PI^{'}M_{2j} \tag{4}$$

The average power dissipated can be estimated using the following equation. In deriving this equation we use the approximation that the power dissipated during an event is independent of the specific transition causing the event.

$$Power_{avg} = \sum_{i=1}^{4} Pr(Event-i) * Power(Event-i) \tag{5}$$

Let *Power(M1)*, *Power(M2)* and *Power(CL)* represent the power dissipated in M1, M2 and the control logic respectively. We assume that the power consumed in M1, M2 and CL are independent of the transitions. Now substituting equations (1)-(4) in (5):

$$\begin{gathered} Power_{avg} = Pr(Event-1) * Power(M1) + Pr(Event-2) * Power(M2) + \\ (Pr(Event-1)+Pr(Event-2)) * (Power(M1)+Power(M2)) + Power(CL) \end{gathered} \tag{6}$$

Since the power dissipated in the control logic is almost independent of the partitioning (it could cause a slight variation in the number of flip-flops, but we can safely ignore this for our purposes), this term could be dropped from the cost function. Additionally, area is a good first order approximation of power in combinational circuits and hence we can express the average power dissipated as given in equation (7).

$$\begin{gathered} Power_{avg} = Pr(Event-1) * Area(M1) + Pr(Event-2) * Area(M2) + \\ (Pr(Event-3) + Pr(Event-4)) * (Area(M1) + Area(M2)) \end{gathered} \tag{7}$$

Equation (7) is used as the cost function for evaluating the fitness of individuals in the GA based procedure we propose. We use the number of product terms derived by ESPRESSO for the combinational logic of M1 and M2 as an estimate of the area and in Equation (7) as an estimate of the power dissipated. It should be pointed out that it is possible to use more direct and more accurate estimate of power dissipated in a candidate partitioned realization of the FSM to determine its fitness. However, the run time of the resulting partitioning and state assignment procedures could be very high and for this reason we use the approximation employed in Equation (7).

3. Genetic Algorithm for simultaneous Partitioning and State Assignment

Genetic Algorithms [11] have been shown to be effective in finding good approximate solutions to a wide range of optimization problems that are NP-complete. In general, a GA-based problem formulation and optimization goes through the following steps: An encoding to represent the solution, generation of an initial random population, evaluation of the fitness function, selection of chromosomes that should participate in the production of the next generation and a mutation operator to ensure that the GA does not get stuck in a local minima. This section describes our implementation of the steps given above.

3.1. Encoding

The encoding should represent both the partition and the state assignment. To represent the partition we use an array called the partition array of n elements representing n states of the FSM. Each element in this array is a 0 or 1, 0 representing that the state is in M1 and 1 representing that the state is in M2. For example, if the machine has 5 states (state 0 to state 4) and the partition array is $< 0, 1, 1, 0, 1 >$, then M1 includes the states 0 and 3 and M2 includes the states 1, 2 and 4. To represent the codes assigned to the states, we use the codes array suggested in [14]. Each element of this array could vary between 0 and *(nmax-1)* where *nmax* is equal to 2^r with minimum value of r such that *nmax* is larger than or equal to *n*. Our objective is to map the set of states (which vary from 0 to *n-1*) uniquely to a set of codes (which vary from 0 to *nmax-1*). For example if $n = 5$ and codes array $= < 2, 4, 7, 0, 4 >$ would map the states $< 0, 1, 2, 3, 4 >$ to codes $< 2, 5, 1, 0, 3 >$. Interested reader could refer to [14] for a detailed explanation of this procedure.

3.2. Generation of the initial population

Each individual has two components: the partition array and the codes array. The codes-array for each individual is generated randomly with each element equally likely to take any value between 0 and $(nmax - 1)$. A different procedure that uses the steady state probabilities of the states of M is used to generate the partition array. This is because we prefer that M1 would contain as many states with high probabilities as possible. We decided on this heuristic after comparing the results for benchmark FSMs with smaller numbers of states using an initial population that is randomly chosen and when the suggested heuristic is used to pick the initial population. We found that the proposed heuristic resulted in considerably better solutions in shorter time. To achieve the objective, we first identify the states whose steady state probability is higher than the average probability (which is 1/n). For these we set the probability of partitioning array element to be 0 (probability that the state will be in M1) to be equal to the difference between its steady state probability and 1/n added with 0.50. For the rest of the states this value is set to its value of the steady state probability itself.

3.3. Fitness Function

Equation (7) of Section II gives an estimate of the power dissipated in a candidate partitioned machine or in other words its cost. The cost represents how bad the corresponding individual is. We derive the fitness [12] of an individual from this cost. Fitness F of an individual is computed from its power cost C by the following procedure. Let C_{avg} denote the average cost of the population. Define a reference worst cost C_w as follows:

$$C_w = C_{avg} + \gamma * \sigma \tag{8}$$

where σ denotes the variance of the population and γ is a user defined scaling factor. For all individuals with cost less than C_w, F is computed as $C_w - C$ and for the others F is set to zero.

3.4. Selection and Crossover

The selection process must ensure that the individuals with higher fitness are more likely to get selected. In GALLOP, we employed tournament selection with replacement. The process never selects an individual with zero fitness so that members with cost more than C_w never get passed on to the next generation. The process of selection and crossover can be summarized as follows:

1. Select 2 individuals random with non-zero fitness. Choose the fitter one to be the first parent p1.
2. Repeat (1) to get the second parent p2
3. Choose a crossover point (crpoint) at random.
4. Offsping1.partition array[1 to n] = p1.partition array[1 to crpoint], p2.partition array[(crpoint+1) to n]
5. Offsping1.codes array[1 to n] = p1.codes array[1 to crpoint], p2.codesarray[(crpoint+1) to n]
6. Offsping2.partition array[1 to n] = p2.partition array[1 to crpoint], p1.partition array[(crpoint+1) to n]
7. Offsping2.codes array[1 to n] = p2.codes array[1 to crpoint], p1.codes array[(crpoint+1) to n]

3.5. Mutation

Mutation consists of a random modification of some members of the current population followed by acceptance or rejection of the mutated individual depending on its fitness. With a probability P_m, a member is chosen to undergo mutation. Then one element of the partition array and the codes array of the individual, chosen at random, is modified to another random value. The randomly chosen element in the partition array could take a value of 0 or 1, with equal probability, and the element in the codes array could take any value between 0 and $(nmax-1)$, with equal probability. We could have just complemented the selected partition array element. But we chose 0 or 1 with equal probability. This makes the effective mutation probability for the partition array to be $0.5 * P_m$. After mutation, the cost of the new individual is computed. If there is a decrease in cost (meaning that we now have an individual better than the one before mutation), the mutation is accepted. If not, the mutation is accepted with a certain probability P_{accept} given by:

$$P_{accept} = 1 - (1 - P_{av}) * \Delta C / \Delta_{av} \qquad (9)$$

where ΔC is the change in cost for the current move, Δ_{av} is the average of the cost increasing moves so far and P_{av} is a user defined constant. Δ_{av} is initially set to zero thereby rejecting the initial cost increasing moves.

3.6. The complete procedure used

The complete flow of the procedure we employed is summarized below.

1. Compute the steady state probabilities of the states of the FSM.
2. Generate an initial random population with partition array and codes array generated as described in Section 3.1.
3. Compute the fitness of each individual.
4. Pass on the 2 best individuals directly to the next generation to ensure that the best individual in each generation is never lost.
5. Generate the remaining set of individuals by cross over operations on the individuals in the existing population by the procedure described in Section 3.2.
6. Perform mutation with a probability of P_m.
7. Repeat steps 3-6 for the specified number of generations
8. Return the best individual.

4. Experimental Results

The proposed GA based synthesis procedure to realize low power FSMs called GALLOP was implemented in C and PERL. The procedure was run on a PC with a 400 MHz Pentium II and 512 MB RAM. After partitioning and state assignment, the resulting circuit was synthesized using SIS [15]. The standard script was used for logic optimization followed by the "full_simplify" command. Power consumption for the entire circuit, including the control logic and flip-flops was estimated using the "power_estimate -t S" command in SIS [15]. Supply voltage and frequency were set to the default values of 5V and 20MHz respectively and zero delay-model was used.

The following nomenclature is used in all the tables in this section: AI for area increase, PR for power reduction and LIT for literal count in the realized circuits.

Table 1 summarizes the results obtained using GALLOP and compares it to the realization synthesized using JEDI (with the option "jedi -e c") (the data for JEDI was taken from [9]) and FSMD [9]. For each circuit we ran GALLOP four times with different random seeds. The best and the average results are reported. For all circuits we ran the GA for 30 generations. Mutation probability was set to 0.10. After the circuit name the number of states in the circuit is given followed by the literal count and the average power dissipated in the circuit realized using JEDI and FSMD for state assignment of the given machine. For example, the power dissipation values for the FSM BBSSE are 373.2 μW and 381.1 μW when implemented using JEDI and FSMD respectively. In the next set of two columns we present the literal count and power dissipation for the circuits obtained using GALLOP. GALLOP-BEST represents the best power instance and the AVG gives the average over 4 runs of GALLOP. For BBSSE these values are 308.1 and 316.25 μW, respectively. This is followed by 2 sets of 2 columns comparing GALLOP with JEDI and FSMD for both the best case and the average over 4 runs of GALLOP. These results are given in percentage increase in area (AI) and percentage decrease in power dissipation (PR) for GALLOP compared to the area and power dissipation using JEDI and FSMD. For example, for BBSSE, GALLOP-BEST results in a $(373.2 - 308.1)/373.2 = 17.44\%$ power reduction and $(103 - 173)/103 = 67.96\%$ area increase over JEDI. Similar data is given with respect to FSMD in the last two columns.

From Table 1 it can be seen that using GALLOP results in an average power reduction of 57% while on average increasing the area (measured by literal count) by 77.23% over the designs obtained when FSMs are synthesized to minimize area by using JEDI for state assignment. The results are particularly good for circuits with larger number of states. For example for the circuits with 48 states an average power reduction of 88.14% is obtained. It can be seen that except for one circuit SAND, GALLOP reduces power for all other benchmarks as compared with FSMD. Also the area increases as expected. For the circuits with 48 states the reduction in power compared to FSMD is over 75%.

Table 1. Comparisons of GALLOP with the design using JEDI

CIRCUIT	n	JEDI		FSMD		GALLOP-BEST, AVG		Against JEDI		Against FSMD	
		#LIT	Power	#LIT	Power	#LIT	Power	%AI	%PR	%AI	%PR
BBSSE	16	103	373.2	152	381.1	173, 173.25	308.1, 316.25	67.96, 68.2	17.44, 15.26	13.82, 13.98	3.14, 0.58
CSE	16	186	422.8	245	257.7	266, 267.75	235.1, 245.45	43.01, 43.95	44.39, 41.95	8.57, 9.29	8.77, 4.75
EX1	20	214	789.8	257	409.8	367, 344.5	351.4, 357.23	71.5, 60.98	55.51, 54.77	42.8, 34.05	14.25, 12.83
KEYB	19	168	599.4	246	492.4	352, 341.25	355.1,383.23	109.52, 103.13	40.76, 36.07	43.09, 38.72	27.88, 22.17
PLANET	48	472	2375.5	536	1337.9	848, 828.25	332.8, 384.03	79.66, 75.48	85.99, 83.83	58.21, 54.52	75.13, 71.3
PMA	24	202	965.1	237	622.2	467, 461.25	159.9, 186.03	131.19, 128.34	83.43, 80.72	97.05, 94.62	74.3, 70.1
S1488	48	503	1448.6	593	592.9	907, 900.75	90, 96.18	80.32, 79.08	93.79, 93.36	52.95, 51.9	84.82, 83.78
S1494	48	514	1276.2	625	938.3	926, 909.25	196.2, 203.63	80.16, 76.9	84.63, 84.04	48.16, 45.48	79.09, 78.3
SAND	32	527	1525.5	488	634.7	878, 836.5	804.9, 813.45	66.6, 58.73	47.24, 46.68	79.92, 71.41	-26.82, -28.16
SSE	16	103	373.2	152	318.1	173, 173.25	308.1, 316.25	67.96, 68.2	17.44, 15.26	13.82, 13.98	3.14, 0.58
STYR	30	407	1073	501	695.8	617, 611.25	467.6, 477.65	51.6, 50.18	56.42, 55.48	23.15, 22.01	32.8, 31.35
AVERAGE								77.23, 73.92	57, 55.22	43.78, 40.91	34.23, 31.6

In Table 2 we compare the results of using GALLOP with LPSA [4] which used a new state assignment procedure targeting low power two level and multilevel implementations. Out of the circuits simulated in Table 1, the results of [4] were available only for those in Table 2. Additionally the results in [4] are given relative to the circuit obtained by default options in JEDI. Default options in JEDI yield different results from those obtained in Table 1. However, in order to make a fair comparison, we report the percentage increase in the area and percentage decrease in power when GALLOP is used relative to the circuit obtained using the default option in JEDI. We obtained these results by synthesizing the FSMs using the default options in JEDI and then determining percentage differences in area and power using GALLOP (JEDI was called for state assignment inside SIS using the ''state_assign jedi" command). From Table 2 it can be seen that using GALLOP the power reduces on average by over 36% relative to the circuit when it is synthesized using default option in JEDI, where as on the average LPSA based designs yield an improvement in power of about 16% only.

Table 2. Comparisons of GALLOP with LPSA

CIRCUIT	LPSA		G-BEST, G-AVG	
	#LIT	POWER	%AI	%PR
BBSSE	6.56	18.37	38.4, 38.6	27.28, 25.36
CSE	-1.41	12.15	11.76, 12.5	39.86, 37.21
EX1	10.88	17.48	39.02, 30.49	53.91, 30.49
SSE	6.56	18.37	38.4, 38.6	27.28, 25.36
SAND	16.12	10.52	38.05, 31.53	43.26, 42.66
AVERAGE	7.74	15.38	33.13, 30.34	38.32, 36.75

We cannot directly compare the results of using GALLOP with the results in [7], which also used a GA based circuit partitioning followed by standard state assignment. The reason is that [7] did not report the clock frequency and supply voltage used. Additionally no information on how the circuits that did not use partitioning were synthesized is available in [7]. However it may be noted that [7] reports an average power reduction of 31% and area increase of 48% compared to the results using GALLOP realizing 57% power reduction and 77% increase in area.

The GALLOP run-times for all the circuits were less than 45 minutes except for $SAND$ (about 1 hour), and $S1488$ and $S1494$ (about 2 hours each).

5. Concluding Remarks

In this work we presented GALLOP - an FSM synthesis tool targeting low power. It differs from previous works by performing state assignment and partitioning simultaneously using GA and using steady state probabilities to bias the initial population to wards giving better results. Experimental results presented show that GALLOP yields circuits which consume considerably less power than those obtained by the use of earlier known low power synthesis procedures. In the future we plan to consider using more accurate measures of power dissipated in a candidate circuit to measure its fitness. We will also consider a fitness function that may allow simultaneously optimizing power and area.

References

[1] K. Keutzer and P. Vanbekbergen, "The Impact of CAD on the design of low power digital circuits", *Dig. of Papers, IEEE Symp.*, pp. 42-45, 1994

[2] T. Villa and A. Sangiovinni-Vincentilli, "NOVA: State Assignment of finite state machines for optimal two-level logic implementation", *IEEE Trans. on CAD*, pp.905-924, Sept 1990

[3] S. Devadas, H-K. T. Ma, A. R. Newton, and Sangiovinni Vincentilli, "MUSTANG: State Assignment of finite state machines targeting multi-level implementations", *IEEE Trans. on CAD*, pp.1290-1300, Dec. 1988

[4] Chi-Ying Tsui, Massoud Pedram and Alvin Despain, "Low-Power State Assignment Targeting Two and Multilevel Implementations", *IEEE Trans. on CAD*, pp. 1281-1291, Dec. 1998

[5] S. Chattopadhyay, "Low power state assignment and flipflop selection for finite state machine synthesis -a genetic algorithmic approach", *IEE Proceedings*, pp. 147-151, July-Sept. 2001

[6] Jose C. Monteiro and Arlindo L. Oliveira, "FSM decomposition by direct circuit manipulation applied to low power design", *ASP-DAC 2000*, pp:351-358

[7] Luca Benini, Giovanni De Micheli and Frederik Vermulen, "Finite State Machine Partitioning for Low Power", *ISCAS 98*, pp:5-8

[8] Wai-Kwong Lee and Chi-Ying Tsui, "Finite State Machine Partitioning for Low Power", *ISCAS 99*, pp: 306-309

[9] Sue-Hing Chow, Yi-Cheng Ho and Tingting Hwang, "Low Power Realization of Finite State Machines-A Decomposition Approach", *ACM Trans. on Design Automation of Electronic Systems*, pp. 315-340, July 1996

[10] Luca Benini and Giovanni De Micheli, "Automatic Synthesis of Low-Power Gated-Clock Finite-State Machines", *IEEE Trans. on CAD*, pp. 630-643, June 1996

[11] D. Goldberg, "Genetic Algorithms in search, optimization and machine learning", *Addison-Welsey*, 1989

[12] Pinaki Mazumder and Elizabeth M. Rudnick, "Genetic Algorithms in VLSI Design, Layout and Test Automation", *Prentice Hall PTR*, 1999

[13] Shanq-Jang Ruan, Rung-Ji Shang, Feipei Lai and Kun-Lin Tsai, "A Bipolar-Codec Architecture to reduce Power in Pipelined Circuits", *IEEE Trans. on CAD*, pp:343-348, Feb 2001.

[14] A. E. A Almaini, J. F. Miller, P. Thomson and S. Billina, "State Assignment of finite state machines using a genetic algorithm", *Computers and Digital Techniques, IEE Proceedings*, pp. 279- 286, July 1995

[15] E. Sentovich, K. Singh, et. al. "SIS: a system for sequential circuit synthesis", *Tech. Rep. M92/41, Electronic Research Laboratory, College of Engineering, University of California, Berkeley*, 1992.