FSM State-Encoding for Area and Power Minimization Using Simulated Evolution Algorithm

Sadiq M. Sait*[1], F. C. Oughali[2], A. M. Arafeh[3]

[1,2,3]Department of Computer Engineering
[1]Center for Communications and IT Research, Research Institute
King Fahd University of Petroleum & Minerals
Dhahran, Saudi Arabia.
*sadiq@kfupm.edu.sa

ABSTRACT
In this paper we describe the engineering of a non-deterministic iterative heuristic [1] known as simulated evolution (SimE) to solve the well-known NP-hard state assignment problem (SAP). Each assignment of a code to a state is given a Goodness value derived from a matrix representation of the desired adjacency graph (DAG) proposed by Amaral et.al [2]. We use the (DAG_a) proposed in previous studies to optimize the area, and propose a new DAG_p and employ it to reduce the power dissipation. In the process of evolution, those states that have high Goodness have a smaller probability of getting perturbed, while those with lower Goodness can be easily reallocated. States are assigned to cells of a Karnaugh-map, in a way that those states that have to be close in terms of Hamming distance are assigned adjacent cells. Ordered weighed average (OWA) operator proposed by Yager [3] is used to combine the two objectives. Results are compared with those published in previous studies, for circuits obtained from the MCNC benchmark suite. It was found that the SimE heuristic produces better quality results in most cases, and/or in lesser time, when compared to both deterministic heuristics and non-deterministic iterative heuristics such as Genetic Algorithm.

Keywords: EDA, FSM Synthesis, State Encoding, Simulated Evolution, Multiobjective Optimization, Non-Deterministic Algorithms, Desired Adjacency Graphs, Fuzzy Logic.

1. Introduction

Most tasks involved in designing VLSI systems employ CAD tools. Digital systems are designed based on the separation of data path and control logic. The control logic is implemented by synthesizing finite state machines (FSMs). Automated design of FSMs with area and power constraints has been of considerable interest to the CAD community. The complexity of FSM implementation lies in its storage elements and combinational logic realization. It is possible to synthesize an FSM using a minimum number of encoding bits ($\log_2\lceil N_s \rceil$, where N_s is the number of states in the FSM); however, using additional bits could be justified if combinational logic was reduced thereby reducing the overall area and/or power consumption. If n_b is the number of bits used to encode each state, then the number of possible assignments (hence the size of the search space [4]) is given by

$$S = \frac{(2^{n_b})!}{(2^{n_b} - N_s)!} \qquad (1)$$

For example, in a two state machine, if two bits are used for encoding, then there are 12 possible assignments. Different state assignments will results in different area and power requirements. For large FSMs, exploring all possible encoding solutions for optimization is an extensive task. At times, it may be possible to reduce the amount of combinational logic by increasing the number of bits per state (incompletely specified machines), but this obviously increases the size of the search space. Clearly, the state assignment problem (SAP) is an NP-hard combinatorial optimization problem similar to the travelling salesman problem. In this work, we engineer another iterative heuristic known as Simulated Evolution (SimE) to search for better solutions to solve the SAP for FSMs.

Previous research on two-level combinational logic realization of FSMs using well designed deterministic heuristics for area minimization employed mechanisms such as implicant merging, code covering and disjunctive coding [5]. Finding states assignment which resulted in common expressions and maximum literal savings was one of the objectives in the state assignment problem (SAP). Devadas et.al., proposed two algorithms [6] [7], the first of which is fan-out oriented and assigns close codes, in terms of Hamming distances, to state pairs that have similar next state transitions. The second algorithm, which is fan-in oriented (also called Mustang), looks for state pairs with higher number of incoming transitions from the same states. Higher weights are given to those pairs of states to be assigned close codes. The motivation in this case is to maximize the frequency of common cubes in the encoded next-state functions. Another heuristic, similar to this one, is Jedi [8] where state assignment are made in a way similar to that proposed by Devadas' fan-out algorithm which calculates the encoding affinity cost as a function of how frequently a pair of states is represented in next-state and output functions. The use of a semiformal specification for code implementation along with formal verification of finite state machines (FSM) was reported by Torres et al. [9].

Non-deterministic general iterative heuristics such as simulated annealing, particle swarm and tabu search has been used to solve a variety of combinatorial optimization problems [10, 11, 12]. For a state assignment problem, a genetic algorithm-based state encoding, targeting minimization of area and power, was proposed by Chaudhury et.al. [13]. They used a unified approach targeting static power and dynamic power along with area trade-off. Other attempts to use GAs to solve the SAP include the work by Aly [14], Almaini et al [13], and others [16], [17]. Another attempt is by Amaral et al [2] who used GA with cost function proposed by Armstrong [18]. Armstrong's measure combines fan-in, fan-out and output costs for measuring literal savings. Amaral proposed a matrix representation as genotype, and a desired adjacency graph (DAG_a) as a tool for applying heuristic rules on FSM [19].

Power consumption in CMOS circuits is mainly attributed to charging and discharging of circuit's capacitance. Reducing power consumption is done by reducing switching activity, logic area (capacitance), or their product. Switching activities in sequential circuits are due to logic transition on flip-flops and primary inputs. Most of the work reported in previous works [20] [21] tries to minimize total switching on the flip-flops.

The aim of this work is to engineer another evolutionary non-deterministic heuristics commonly known as SimE. One key requirement of SimE is to define a Goodness measure of the current assignment of a movable element, in our case the assignment of a binary pattern to a state. In the process of evolution, elements with high Goodness are given a lower probability of moving from their current assignments. The DAG proposed by Amaral [2] is exploited in the design of cost estimators and Goodness measures required by the SimE algorithm.

The rest of the paper is organized as follows. In section 2 we discuss the construction of the desired adjacency graphs (DAG) as proposed by Amaral [2] for area minimization, and we propose a new DAG for power minimization. In section 3 we present the SimE heuristic for the state assignment problem (SAP), using desired adjacency graph (DAG) proposed in [2]. In section 4 we present problem formulation, proposed Goodness measures and an allocation function for SimE. Experiments and results are reported in section 5. Finally, we provide some final conclusions.

2. Desired adjacency graphs (DAGs)

The assignment of codes to states is a combinatorial optimization problem with the size of the search space given by equation 1. Searching all possible encoding solutions is an extensive work that requires sub-optimal search methods. Table 2 shows a description of a small finite state machine with two possible assignments. Assignment 1 has a cost of 47 literals (in SOP form when synthesized by "SIS" [22]) while assignment 2 has a cost of only 4 literals. Efficient techniques are required to find the right assignment of codes to states to reduce cost.

Present state	**Next state**		**Output Z0**	**Assign #1**	**Assign #2**
	I0=0	**I0=1**			
S0	S0	S4	0	101	010
S1	S0	S4	1	110	000
S2	S1	S5	0	001	011
S3	S1	S5	1	100	001
S4	S2	S6	0	011	110
S5	S2	S6	1	000	100
S6	S3	S7	0	111	111
S7	S3	S7	1	010	101

Table 1. Two assignments of an FSM for “shiftreg” MCNC benchmark requiring different number of literals.

	S0	S1	S2	S3	S4	S5	S6	S7
S0	0	9	2	0	9	0	2	0
S1	9	0	1	3	1	8	0	2
S2	2	1	0	8	3	2	8	0
S3	0	3	8	0	0	3	1	9
S4	9	1	3	0	0	8	3	0
S5	0	8	2	3	8	0	1	2
S6	2	0	8	1	3	1	0	9
S7	0	2	0	9	0	2	9	0

Table 2. DAGa for “shiftreg” MCNC benchmark for area optimization.

As discussed earlier, many heuristic techniques have been proposed for the state assignment problem [2, 18, 23]. Based on a paper by Armstrong [2], Amaral introduced a tool called desired adjacency graph (*DAG$_a$*) which can be used for applying heuristic rules to any given FSM. The heuristic used suggest that reduction in literal count, and thereby the cost of synchronous sequential circuits (SSC) is possible by minimizing the Hamming distance between predefined sets of states which may be chosen as follows. If they are:

- Rule i: In the same set of successors of a given state.
- Rule ii: In the same set of predecessors of a given state with a given input condition.
- Rule iii: In the same partition for a given output.

2.1 DAG$_a$ for an area minimization

Using the above, Amaral et al [2] proposed a desired adjacency graph (*DAG$_a$*), which is a weighted graph that represents the strength of connection between states (nodes of the graph). Simply, it indicates the desirability of having states close to each other. In order to have a low area SSC, it is necessary to minimize the distance between states that are strongly connected in the *DAG$_a$*. (Please refer to [2] for equations used to build a *DAG$_a$*).

As an example, given the previously described FSM which represents the shiftreg benchmark circuit of MCNC, table 2 shows the *DAG$_a$* obtained using equations reported in [2]. For example, we obtain DAG (0,4) = R1 + R1 + R3 + R4 = 3 + 3 + 2 + 1 = 9. The two factors R1 are produced because states S0 and S4 are common successors of states S0 and S1 (Rule i), the factor R3 appears because Z0(S0) = Z0(S4) (Rule iii), and R4 is added because there is one transition from state S0 to S4.

2.2 DAG_p for power minimization

Since our objective is to minimize both area and power, we propose a DAG_p that can be used for the power minimization. As is well known, the switching activity in a circuit has a direct influence on the power dissipated. The switching activities in a finite state machine can be modeled as a state transition graph (STG) G(V, E), where a vertex S_i in V represents a state of the FSM and an edge E_{ij} in E represents a transition from state S_i to S_j . Let P_{si} denote the probability of finding the state machine in state S_i, and p_{ij} denotes the transition probability from state S_i to state S_j. Interpreting the STG as a Markov Chain, P_{si} is the steady states probabilities.

The steady states probabilities can be iteratively calculated by solving Chapman-Kolmogorov equations. The process is terminated once the state probabilities converge so that the difference between successive iterations is within a user defined tolerance value. Thus, the total transition probability from a state S_i to state S_j is the probability that the machine in state S_i multiplied by the transition probability from state S_i to state S_j.

$$P_{ij} = p_{ij}.P_{sj} \qquad (2)$$

Where P_{ij} is the total state transition probability from state S_i to state S_j. The sum of the total state transition probabilities between two states indicates the amount of switching between them.

$$W_{ij} = P_{ij} + P_{ji} \qquad (3)$$

Based on the transition values calculated in STG, a desired adjacency graph (DAG_p) for power minimization is formulated. The values of weights in the DAG_p indicate the desired relative proximity in the state assignment of two states. By assigning shorter distance codes to states connected with higher values, (i.e., higher transition probability), the overall switching on the state lines of the FSM can be minimized.

As an example, table 3 shows the transition probabilities p_{ij} from state S_i to state S_j for the *bbara* circuit in MCNC benchmark used in this paper.

The steady state probabilities are calculated using the script in *Matlab* that iteratively solves the Chapman-Kolmogorov equations. Table 4 shows the steady state probabilities for the *bbara* circuit in MCNC benchmark.

The total transition probability from state S_i to state S_j is the probability that the machine in state S_i (steady state of S_i) multiplied by the transition probability from state S_i to state S_j. The sum of the total state transition probabilities between two states indicates the amount of switching between them, which equals to the value of our proposed DAG_p for power minimization. As an example, the transition probability from $S0$ to $S1$ equals to 0.125 and the transition probability from $S1$ to $S0$ equals to 0.0625. Thus, we obtain $DAG_p(1,2) = P_{0,1} + P_{1,0} = (p_{0,1} \times P_{s0}) + (p_{1,0} \times P_{s1}) = (0.125 \times 0.155242) + (0.0625 \times 0.266667) = 0.036072$. Where $p_{0,1}$ and $p_{1,0}$ are the transition probabilities from $S0$ to $S1$ and from $S1$ to $S0$ respectively. And P_{s0} and P_{s1} are the steady state probabilities of $S0$ and $S1$. Table 5 gives the DAG_p for “bbara” MCNC benchmark that can be used in the design of Goodness measures required by SimE for power optimization.

3. Simulated evolution

The SimE algorithm is a general search strategy for solving a variety of combinatorial optimization problems which seek to find a global optimum of some real valued cost function cost Ω → R defined over a discrete set Ω. The set Ω is called the state

	S0	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**
S0	0.8125	0.125	0	0	0.0625	0	0	0	0	0
S1	0.0625	0.75	0.125	0	0.0625	0	0	0	0	0
S2	0	0.0625	0.75	0.125	0.0625	0	0	0	0	0
S3	0	0	0	0.875	0.0625	0	0	0.0625	0	0
S4	0.0625	0.125	0	0	0.75	0.0625	0	0	0	0
S5	0	0.125	0	0	0.0625	0.75	0.0625	0	0	0
S6	0	0.125	0	0	0	0	0.8125	0.0625	0	0
S7	0	0.125	0	0	0.0625	0	0	0.75	0.0625	0
S8	0	0.125	0	0	0.0625	0	0	0	0.75	0.0625
S9	0.0625	0.125	0	0	0.0625	0	0	0	0	0.75

Table 3. State transitions probabilities of “bbara” circuit for MCNC benchmark.

S0	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**
0.1552	0.2666	0.1333	0.1333	0.1967	0.0498	0.0163	0.0374	0.0093	0.0023

Table 4. Steady state probabilities of “bbara” circuit for MCNC benchmark.

space and its elements are referred to as states (do not confuse with the states of FSM). A state space Ω together with an underlying neighborhood structure (the way one state can be reached from another state) form the solution space. Combinatorial optimization problems can be modeled in a number of ways. A generic formulation suggested by Saab and Rao [24] is the following: Given a finite set M of distinct movable elements and a finite set L of locations, a state is defined as an assignment function S: M → L satisfying certain constraints.

Many of the combinatorial problems can be formulated according to this generic model. In our case the assignment is of binary patterns of fixed length to states of an FSM.

The structure of the SimE algorithm is shown in Figure 1. SimE assumes that there is a solution Ø of a set M of n (movable) elements or modules. The algorithm starts from an initial assignment $Ø_{initial}$, and then, following an evolution based approach, seeks to reach better assignments from one generation to the next by perturbing some ill-suited components and retaining the near-optimal ones. A cost function cost associates with each assignment of movable element m_i a cost C_i. The cost C_i is used to compute the Goodness (fitness) g_i of an element m_i, for each $m_i \in M$. The algorithm has one main loop consisting of three basic steps, evaluation, selection, and allocation. The three steps are executed in sequence until the solution’s average Goodness reaches a maximum value, or no noticeable improvement to the solution fitness is observed after a number of iterations. The evaluation step consists of evaluating the Goodness g_i of each element m_i of the solution Ø. The Goodness measure must be a single number expressible in the range [0, 1]. It is defined as:

$$g_i = \frac{O_i}{C_i} \tag{4}$$

Where O_i is an estimate of the optimal cost of the element m_i, and C_i is the actual cost of m_i in its current location.

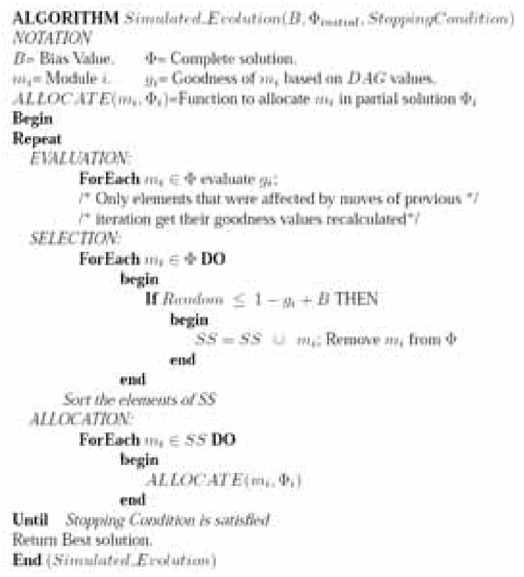
The above equation assumes a minimization problem (maximization of Goodness). Notice that, according to the above definition, the O_i’s do not change from generation to generation, and therefore, are computed only once during the

	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
S0	0	0.0361	0	0	0.0220	0	0	0	0	0.0001
S1	0.0361	0	0.0417	0	0.0413	0.0061	0.0020	0.0047	0.0012	0.0003
S2	0	0.0417	0	0.0167	0.0083	0	0	0	0	0
S3	0	0	0.0167	0	0.0083	0	0	0.0083	0	0
S4	0.0220	0.0413	0.0083	0.0083	0	0.0154	0	0.0023	0.0006	0.0001
S5	0	0.0061	0	0	0.0154	0	0.0031	0	0	0
S6	0	0.0020	0	0	0	0.0031	0	0.0010	0	0
S7	0	0.0047	0	0.0083	0.0023	0	0.0010	0	0.0023	0
S8	0	0.0012	0	0	0.0006	0	0	0.0023	0	0.0006
S9	0.0001	0.0003	0	0	0.0001	0	0	0	0.0006	0

Table 5. DAGp for "bbara" MCNC benchmark for power optimization.

initialization step. Hence only the C_i's have to be recomputed at each call to the evaluation function. Empirical evidence [25] shows that the accuracy of the estimation of O_i is not very crucial to the successful application of SimE. However, the Goodness measure must be strongly related to the target objective of the given problem.

The second step of the SimE algorithm is selection. Selection takes as input a bias value B, the solution Ø together with the estimated



```
ALGORITHM Simulated_Evolution(B, $\Phi_{initial}$, StoppingCondition)
NOTATION
B= Bias Value.        $\Phi$= Complete solution.
$m_i$= Module i.      $g_i$= Goodness of $m_i$ based on DAG values.
ALLOCATE($m_i$, $\Phi_i$)=Function to allocate $m_i$ in partial solution $\Phi_i$
Begin
Repeat
  EVALUATION:
      ForEach $m_i \in \Phi$ evaluate $g_i$;
      /* Only elements that were affected by moves of previous */
      /* iteration get their goodness values recalculated*/
  SELECTION:
      ForEach $m_i \in \Phi$ DO
          begin
            If Random $\leq 1 - g_i + B$ THEN
              begin
                $SS = SS \cup m_i$; Remove $m_i$ from $\Phi$
              end
          end
    Sort the elements of SS
  ALLOCATION:
      ForEach $m_i \in SS$ DO
          begin
            ALLOCATE($m_i$, $\Phi_i$)
          end
Until  Stopping Condition is satisfied
Return Best solution.
End (Simulated_Evolution)
```

Figure 1. Structure of the SimE algorithm [1].

Goodness of each element. It partitions Ø into two disjointed sets; a selection set SS and a partial solution $Ø_p$ of the remaining elements of the solution Ø. Each element in the solution is considered separately from all other elements. The decision whether to assign an element m_i to the set SS is based solely on its Goodness g_i. The selection operator has a non-deterministic nature, i.e., an individual with a high Goodness (close to one) still has a non-zero probability of being assigned to the selection set *SS*. It is this element of non-determinism that gives SimE the capability of escaping local minima. Allocation is the SimE operator that has the most impact on the quality of solution. Allocation takes as input the set *SS* and the partial solution $Ø_p$ and generates a new complete solution Ø' with the elements of set *SS* mutated according to an allocation function. The goal of allocation is to favor improvements over the previous generation, without being too greedy.

4. Problem formulation

This section describes how SAP is formulated in SimE. This includes population representation, the use of DAGs in the development of Goodness measures to be used in evaluating each individual of the population, and the allocation procedure.

4.1 Solution representation

States assignment problem can be seen as a linear or 2-D placement problem. In table 6, states are placed in suitable locations associated with different Gray codes. A data structure similar to Karnaugh maps is employed where adjacent slots have a Hamming distance of one. As it is preferred that pairs of states with high DAG values have close codes, an attempt was made to get them as close to each other as in the 2-D structure.

Gray coding	00	01	11	10
Locations	L0	L1	L2	L3

Table 6. Locations associated with Gray Sequence.

We can think of the set of movable elements as the states, and the set of locations as the squares of the K-map. In order to employ the SimE algorithm we need to define the population and a Goodness measure for individuals. Since the simplicity of the population representation determines the complexity of the allocation function, the population is represented as a linear or 2-D array. The number of array elements equals the number of FSM states. Each element contains index of one Gray sequence code. As shown in the example in table 7, the first element in the population corresponds to the state (S0) and contains the index of the corresponding Gray code.

S0	**S1**	**S2**	**S3**
L0	L2	L3	L1

Table 7. Population representation.

Giving a finite state machine with 4 states, the final solution is the SimE population that results in minimum cost. The algorithm will try to place highly connected states in DAG into adjacent locations (i.e., assign them close codes). Table 8 depicts the final solution where S0 assigned code (00) which corresponds to the location (L0) in Gray sequence. Similarly, S1 is assigned code (11) which corresponds to location (L2) in Gray sequence, and so on.

Gray coding	00	01	11	10
State assign.	S0	S3	S1	S2

Table 8. Final solution sepresentation.

4.2 Goodness measure–I

Goodness measure in SimE (equation 4) consists of two elements; O_i an estimation of optimal cost and C_i the actual cost of individual S_i in the population. Goodness value of individual S_i is defined as the sum over all S_j states of Hamming distance between state S_i and state S_j multiplied by DAG_{ij}. Pairs of states with high DAG values are preferable to have close codes (minimum Hamming distance). Estimation of the optimal Goodness O_i is calculated as in equation 5. The distance between individual S_i and all other states is assumed to be equal to 1.

$$O_i = \sum_{j=0}^{s-1} DAG_{ij} \qquad (5)$$

Although this estimation of O_i is not possible to be achieved, it is still a good estimation for an optimal case. Empirical evidence shows that the accuracy of O_i is not very crucial to the successful application of SimE [25].

4.3 Goodness measure–II

Goodness value of individual S_i is defined as the sum over all states of Hamming distance between state S_i and state S_j multiplied by DAG_{ij}. Estimation of the optimal Goodness O_i is calculated as in equation 6, where, the distance between state i and other states is assumed to be based on the value of W_j .

$$O_i = \sum_{j=0}^{s-1} DAG_{ij}.W_{ij} \qquad (6)$$

In any binary encoding of finite-state machines, the number of codes that are a distance d from any other code is calculated using mathematical combinations. Table 9 shows an example of 3, 4 and 5-bits encoding. The number of codes that are of Hamming distances d = 1, 2, 3, 4, 5 are listed.

Encoding Length	**Number of codes of distance (d) from any given code**				
	d=1	d=2	d=3	d=4	d=5
3-bits	3	3	1		
4-bits	4	6	4	1	
5-bits	5	10	10	5	1

Table 9. Hamming distance in n-bit encoding.

Thus, for any encoding with n-bits, the total number of codes that are of a distance (d) from any other code is given by the following equation:

$$N_c = \binom{n_b}{d} = \frac{n_b!}{d!(n_b - d)!} \qquad (7)$$

Weight vector W is built according to the above definition. For example, in 3-bits encoding, weight vector will be W= [1, 1, 1, 2, 2, 2, 3]. O_i is then calculated by sorting DAG_i in descending order and applying equation 6. In Goodness measure II, states with strong connections are multiplied by small weights (distance) values, while weakly connected states will be multiplied by higher distance values. This calculation provides a more accurate assessment of the optimal value O_i.

The actual cost C_i for individual S_i is expressed in equation 8. $D(S_i, S_j)$ is the Hamming distance between codes of state S_i and state S_j. DAG_{ij} is the corresponding weight between state S_i and state S_j. DAG could be related to the area or power. C_i will be low when pairs of states with high DAG_{ij} have a small distance D (i.e., assigned adjacent codes due to their strong relation represented in DAG). As O_i values are fixed for all generations, the Goodness will be determined by the value of C_i. As C_i approaches the estimated optimal value O_i, the Goodness of individual S_i increases.

$$C_i = \sum_{j=0}^{s-1} D(S_i, S_j).DAG_{ij} \qquad (8)$$

As the heuristics suggest, an assignment with maximum Goodness value for each individual should result in an SSC with minimal cost.

4.4 Initialization

Initialization includes specifying bias for selection procedure, stopping criteria which have been chosen to be a fixed number of generations, computing O_i for each individual in the population, and constructing an initial population by random code assignment.

4.5 Selection

In the selection step, each member of the population is considered separately for selection; selection function is used with its original description in SimE [1]. Individuals with low Goodness are more likely to be selected for mutation in the next generation. On the other hand, individuals with high Goodness have higher chance of retaining their assigned codes in next generation. However, they still have nonzero probability to be assigned to selection set *SS*. The value of Bias *B* is a function of how realistic is the estimate of optimal cost O_i of individual S_i. In case O_i is a tight lower bound on the actual cost C_i, then a value of (*B=0*) is a reasonable choice. However, if O_i is a loose lower bound for C_i, i.e., if O_i cannot possibly be achieved (like in our case), then a small negative value for *B* should be chosen to compensate for the lack of accuracy of O_i.

Selected elements in set *SS* are sorted in an ascending order based on their Goodness value, where elements with lower Goodness value are processed first at allocation step.

Example	inputs	outputs	states
bbara	4	2	10
bbsse	7	7	16
dk14	3	5	7
donfile	2	1	24
train11	2	1	11
lion9	2	1	9
s1	8	6	20
shiftreg	1	1	8
tav	4	4	4
Sand	11	9	32

Table 10. Selected MCNC benchmark circuits used for comparison.

Avg. # selected	14	13	12	10
Bias	0	-0.05	-0.1	-0.15

Table 11. Bias Vs. average no. of selected individuals.

Example	GA	NOVA1	NOVA2	Jedi	SimE-1	SimE-2
bbara	130	134	154	124	109	99
bbsse	345	312	381	289	257	275
dk14	252	252	268	346	208	203
donfile	408	321	280	169	260	174
train11	53	79	48	40	43	38
lion9	22	51	39	30	23	21
shiftreg	10	9	3	18	4	4
tav	32	35	35	35	32	32
Avg.	157	149	151	131	120	112

Table 12. SAP results comparison for area minimization (in no. of literals SOP).

Benchmark	SimE		Jedi	
	Assign.	Cost lit(SOP)	Assign.	Cost lit(SOP)
bbara	6-3-15-11-2-0-8-10-14-7	92	6-10-11-14-2-3-0-8-12-4	124
bbsse	14-12-6-4-2-0-11-3-13-5-8-15-7-1-9-10	248	15-2-12-8-4-5-0-1-14-10-3-6-7-13-11-9	289
dk14	7-3-5-1-2-6-0	205	4-0-1-2-5-3-7	346
donfile	19-22-27-17-18-16-3-6-15-24-26-30-32-21-31-29-20-28-2-0-10-14-4-12	213	12-13-15-14-29-31-8-25-11-10-9-27-20-28-4-6-30-22-16-24-0-18-26-2	169
train11	4-1-3-0-6-9-5-2-7-11-15	38	3-5-11-7-10-13-15-2-0-9-1	40
lion9	0-1-3-11-15-7-5-4-6	19	15-13-9-11-14-12-8-10-6	30
shiftreg	7-3-6-2-5-1-4-0	4	7-4-6-5-3-0-2-1	18
tav	1-0-2-3	32	1-2-3-0	35
Avg.	-	106	-	131

Table 13. States assignment for SimE, Jedi.

4.6 Allocation

After selecting and sorting individuals in an ascending order of their Goodness values, they are processed. Each state in the selection set *SS* needs to be assigned a new code, where it should have better Goodness compared to the previous one. All codes assigned originally to the selected states are now free. First state to be reallocated can have any of these nonassigned codes, while next states will have remaining free codes. The state to be processed will have its Goodness evaluated for each available code. The evaluation process will take into consideration the remaining states in $Ø_p$, in addition to states which already have been reassigned new codes. The code that maximizes the state Goodness will be chosen.

5. Implementation and results

The experiments compare performance of SimE for the area minimization with Nova-1[1] Nova-2[2] [26] and genetic algorithm reported by Amaral [2]. Results for Jedi [8] tool were also included in the comparison. MCNC benchmarks were used for reporting and comparing the results in this paper,

[1] Nova-1 is NOVA executed with the default option -e ig.
[2] Nove-2 is NOVA executed with options -e ioh -r.

Example	GA (MWHD)	Jedi - Best	SimE-1	SimE-2
bbara	214.7	156.5	135.98	137.6
bbsse	446.1	496.6	393.14	393.06
dk14	661.2	628.1	502.56	493.04
train11	180.4	207.1	157.18	151.94
lion9	142	145.6	117.64	107.78
s1	1165.1	1087.2	971.3	992.12
shiftreg	163.3	96.3	144.02	135.2
Avg.	424.61	360.71	323.93	321.55

Table 14. SAP Results comparison for power minimization (μW).

Benchmark	SimE	
	Assign.	Cost lit (μW)
bbara	0-2-6-7-4-12-5-1-9-8	92
bbsse	0-4-5-1-6-7-3-2-10-14-15-8-12	248
dk14	1-3-0-4-2-7-5	205
train11	13-15-11-7-5-14-12-3-1-10-9	38
lion9	6-2-8-0-4-5-1-3-7	19
s1	0-2-6-4-1-20-3-7-19-23-31-22-9-16-27-17-15-18-11-25	4
shiftreg	0-2-1-3-4-6-5-7	32
Avg.	-	106

Table 15. State assignments for SimE.

details of selected circuits in terms of the number of states, etc., are given in table 10. The cost is obtained using “SIS” tools developed by UC Berkeley [22], and it is the number of literals in SOP form, which are required for implementing the FSM.

GA solutions reported by Amaral [2], were obtained with a population size of 200, and run for 800 generations [2]. In our implementation, we fixed the number of iterations to eight hundred, and bias value of (-0.15) for Goodness measure–I and 0 for Goodness measure II. Table 11 shows relation between bias value and number of selected individuals of the population. It is evident that the increased value of bias (i.e. from -0.15 to 0) results in an increase in the average number of selected individuals (i.e., from 10 to 14). This behavior is due to the use of loose bound estimation of Goodness O_i, namely Goodness measure-I. A high (positive) bias will increase the number of selected elements in each iteration, which allows the algorithm to search harder. In contrast, a negative value of bias will have the effect of reducing the number of selected elements for mutation.

Comparison of results is reported in table 12. Cost reported by “SIS” tool is taken without any further minimization to the logic obtained. Reported numbers are averaged results as they vary due to the random behavior of the non-deterministic iterative heuristic. Clearly, SimE outperforms other tools and algorithms in most cases, (except in a few, for example results for donfile circuit when compared to results by Jedi). While, GA is known to be very time consuming heuristic, SimE has a much shorter runtime (i.e., 80 to 90 sec) for used benchmarks. It was also noted that time did not increase rapidly with the increase of number of states due to the compact population representation. We can notice that SimE-2 achieveed better results compared to SimE-1 as the optimal cost estimation used in Goodness measure-II is more realistic.

Validation of obtained state assignments was conducted by matching the obtained logic with the original description of the state machine. This validation ensures that state assignment done by SimE doesn't change circuits' behavior. Table 13 shows final state assignment and literal count produced by SimE compared to Jedi assignment. For GA and NOVA assignments you can refer to [2]. These assignments are provided for the sake of verification.

Power minimization results are compared with results reported in [27] and Jedi tool using assignment options that gives best results. In [27] the authors are reporting power minimization using two methods; first by using MWHD and then by a measure called Fan-out which accounts for the size of the logic cones. Table 14 shows that SimE with Goodness measure–II is outperforming all other methods. In most cases, SimE results are better than those reported in [27] using MWHD (except for one circuit (shiftreg) when compared to results reported using Jedi). The values reported in table 14 are the average of five different runs. All the results for power minimization were calculated using (power_estimate -t sequential) call in SIS synthesis tool. The power values reported are in microwatts assuming 20 MHz clock and 5 voltage power supply. Table 15 shows the assignments of codes for the states of the benchmark circuits when optimized for power. Note that numbers in different tables vary slightly due to the non-deterministic nature of the algorithm. But as it can be observed, the final results are close indicating the stability of the heuristic and its implementation.

Figure 2 shows the behavior of a SimE run. The darker line in the plot is the average Goodness measure, and the lighter line above depicts the FSM cost. As it can be observed, with iterations, the cost decreases and the Goodness increases. Also observe the hill-climbing phenomenon of SimE.

Our problem being solved consists of two objectives to be optimized. Balancing different objectives by a weighted cost functions is not sufficient to reach the desired solution. One convenient vehicle available for representing multi-objective cost functions is fuzzy logic which provides a required formal algebra to express and combine trade-off objective criteria. Functions for each objective are used (called membership functions) which map the numerical value of objectives to the interval [0,1] [1]. To obtain a fuzzy logic definition of the above multi-criteria objective function, two linguistic variables area and power are introduced and a linguistic value for each variable is defined, in our case *small* for *area*, and *low* for *power*. These linguistic values characterize the degree of satisfaction of the designer with the values of objectives $f_i(x)$, $(i=a,p)$. These degrees of satisfaction are described by membership functions $\mu_i(.)$ on fuzzy sets of linguistic values.

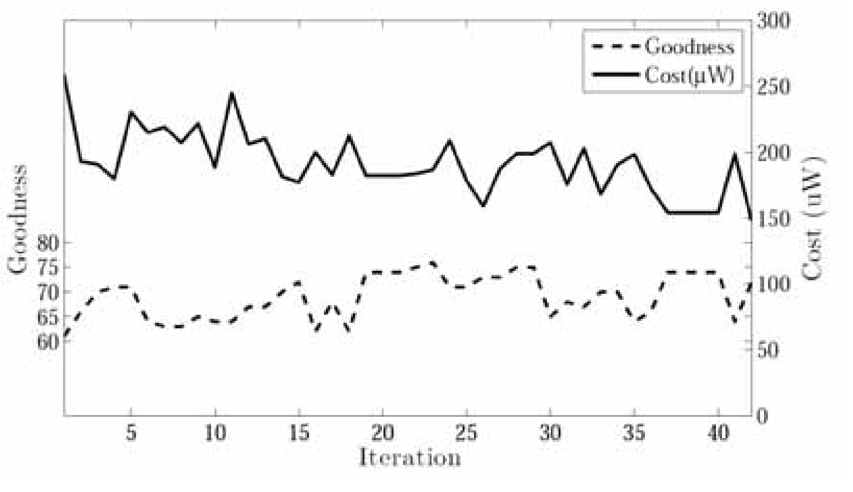


Figure 2. SimE Behavior.

Membership functions for small area and low power are built. These are non-increasing functions, since the smaller the area $f_a(.)$ and lower the power $f_b(x)$, the higher is the degree of satisfaction $\mu_a(.)$, and $\mu_b(.)$. The most desirable solution is the one with the highest membership in the fuzzy subsets *small area*, and *low power*. However, such a solution most likely does not exist. Therefore, one has to trade-off these individual criteria with each other. This trade-off is conveniently specified in linguistic terms in the form of the following fuzzy logic rule.

Let the fuzzy subset of good solutions be characterized by the following fuzzy rule:

R.0 **If** (small area) OR (low power)
Then good solution.

We implement the fuzzy OR above using the *orlike* Ordered Weighted Averaging (OWA) operator proposed by Yager [3] where the degree of ORing is controlled by a parameter β between [0,1]. According to the *orlike* operation, the above fuzzy logic rule R.0 evaluates the following.

$$\mu(x) = \beta \times \max(\mu_a, \mu_b) + (1-\beta) \times \frac{1}{2}(\mu_a + \mu_b) \tag{9}$$

where β is a parameter between 0 and 1 indicating the degree of nearness of this *orlike* operator to the strict meaning of the **max** operator. Table 16 shows results obtained using fuzzy operators compared to results obtained for area or power individually. Comparison between fan-out [23], Jedi and SimE are reported in Table 17. SimE with fuzzy logic outperforms other methods in five circuits and performs poorly in two circuits.

6. Conclusions

In this paper we presented the engineering of an evolutionary heuristic [1] to find better solutions for the NP-hard state assignment problem. Solutions in simulated evolution heuristic evolve based on the current Goodness value of their assignments. SimE accommodates the domain knowledge of the designer in the design of Goodness measure which plays an important role in the perturbation of solutions during their process of their evolution. Two Goodness measures are proposed, one incorporating more domain knowledge, and this is reflected in the performance of the heuristic in the solutions obtained. It is evident that the more the engineer puts in his domain knowledge in the design of Goodness measures, the better is the performance of the heuristic. Goodness measures

Benchmark	**Area Heuristic**		**Power Heuristic**		**Fuzzy (MAX)**	
	Area	Power	Area	Power	Area	Power
bbara	58	162.56	63	132.46	58	136.26
bbsse	124	463.26	134	393.06	123	400.84
dk14	104	531.88	104	493.04	104	493.3
train11	20	101.84	31	151.94	21	97.64
lion9	14	86.76	16	107.78	15	80.74
s1	352	1312.88	296	992.12	294	993.2
shiftreg	4	108.04	17	135.2	4	108.04
tav	24	167.3	24	161.96	24	161.96
Avg.	88	366.82	86	320.95	80	309

Table 16. Fuzzy logic results comparison.

Benchmark	GA - Fanout (MAX)		Jedi -best		SimE - fuzzy (MAX)	
	Area	Power	Area	Power	Area	Power
bbara	58	181.2	73	156.5	58	136.26
bbsse	123	437.1	134	496.6	123	400.84
dk14	101	551.3	108	628.1	104	493.3
train11	23	122.2	34	207.1	21	97.64
lion9	16	105.3	19	145.6	15	80.74
s1	191	751.2	282	1087.2	294	993.2
shiftreg	2	96.3	2	96.3	4	108.04
Avg.	73	320.66	93	402.49	88	330

Table 17. Literature comparison (area & power).

in this work exploit the desired adjacency graphs available in the literature for area optimization [2]. We propose a new desired adjacency graph for power optimization. In order to assign codes which are close in terms of Hamming distance, the problem is treated as an assignment problem with a difference. The 2-D structure to which they are assigned is similar to Karnaugh maps, and the heuristic seeks to find adjacent squares for pairs of states that have a minimal Hamming distance. The two objectives for area and power are combined using the fuzzy ordered weighted operator proposed by Yager [3]. Results are compared to those published in literature and it is seen that both in terms of quality of solutions and the required run-time, the performance of the SimE heuristic with the proposed Goodness measures is excellent and can be used for other similar NP-hard problems.

Acknowledgment

The authors acknowledge King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia for support. This work was also supported under KFUPM funded project # FT-2005/63.

References

[1] Sadiq M. Sait and Habib Youssef. Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems. IEEE Computer Society Press, California, December 1999.

[2] J. N. Amaral, K. Tumer, and J. Ghosh. Designing genetic algorithms for the state assignment problem. IEEE Transactions on Systems, Man and Cybernetics, 25(4):687 –694, April 1995.

[3] Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. IEEE Transaction on Systems, MAN, and Cybernetics, 18(1), January 1988.

[4] Peter Weiner and Edward J. Smith. On the number of distinct state assignments for synchronous sequential machines. IEEE Transactions on Electronic Computers, EC-16(2):220–221, April. 1967.

[5] Bernhard Eschermann. State assignment for hardwired VLSI control units. ACM Computer Survey, 25:415 436, December 1993.

[6] Pranav Ashar, Srinivas Devadas, and A. Richard Newton. Sequential Logic Synthesis. Kluwer Academic Publishers, Norwell, MA, USA, 1992.

[7] A. R. Newton S. Devadas, H. T. Ma. Mustang: State assignment of finite state machines for optimal multi-level logic implemementations. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, November 1987.

[8] B. Lin and A. R. Newton. Synthesis of multiple level logic from symbolic high-level description languages. In Very Large Scale Integration, 1990.

[9] R.E. Gonzalez D.Torres, J.Cortez. Semi-formal specifications and formal verification improving the digital design: some statistics. Journal of Applied Research and Technology, 7(1):15-40, April, 2009.

[10] A. E. Ylmaz F. Yaman. Impacts of genetic algorithm parameters on the solution performance for the uniform circular antenna array pattern synthesis problem. Journal of Applied Research and Technology, 8(3):378-394, December, 2010.

[11] Nareli Cruz-Cortes Ricardo Barron-Fernandez Jesus A. Alvarez-Cedillo Gerardo A. Laguna-Sanchez, Mauricio Olguin-Carbajal. Comparative study of parallel variants for a particle swarm optimization algorithm implemented on a multithreading GPU. Journal of Applied Research and Technology, 7(3):292-309, December, 2009.

[12] A. Miranda-Vitela F. Lara-Rosano J. L. Perez-Silva, A. Garces-Madrigal. Dynamic fuzzy logic functor. Journal of Applied Research and Technology, 6(2):84-94, August, 2008.

[13] Saurabh Chaudhury, Krishna Teja Sistla, and Santanu Chattopadhyay. Genetic algorithm-based FSM synthesis with area-power trade-offs. Integr. VLSI J., 42:376–384, June 2009.

[14] Walid M. Aly. Solving the state assignment problem using stochastic search aided with simulated annealing. American Journal of Engineering and Applied Sciences, 2:703–707, 2009.

[15] A. E. A. Almaini, J. F. Miller, P. Thomson, and S. Billina. State assignment of finite state machines using a genetic algorithm. IEE Proceedings - Computers and Digital Techniques, 142(4):279 –286, Jul 1995.

[16] M. Chyzy and W. Kosinski. Evolutionary algorithm for state assignment of finite state machines. In Proceedings of Euromicro Symposium on Digital System Design, pages 359 – 362, 2002.

[17] N. Nedjah and Luiza de Macedo Mourelle. Evolutionary synthesis of synchronous finite state machines. In The 2006 International Conference on Computer Engineering and Systems, pages 19 –24, November 2006.

[18] D. B. Armstrong. A programmed algorithm for assigning internal codes to sequential machines. IRE Transactions on Electronic Computers, EC-11(4):466 – 472, Augest 1962.

[19] J. N. Amaral and Wagner C. Cunha. State assignment algorithm for incompletely specified finite state machines. In Fifth Congress of the Brazilian Society of Microelectronics, 1990, pages 174–183, 1990.

[20] L. Benini and G. DeMicheli. State encoding for low power embedded controllers. IEEE Journal of Solid State Circuits, 30:258 – 268, 1995.

[21] S. Chattopadhyay and P. N. Reddy. Finite state machine state assignment targeting low power consumption. IEE Proceedings - Computers and Digital Techniques, 151(1):61 – 70, January 2004.

[22] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. SIS A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, EECS Department, University of California, Berkeley, 1992.

[23] D. J. Comer. Digital Logic and State Machine Design. Saunders College Publishing, 3rd edition, 1995.

[24] Y. Saab and V. Rao. Stochastic Evolution: A Fast Effective Heuristic for some Generic Layout Problems. In 27th ACM/IEEE Design Automation Conference, pages 26–31, 1990.

[25] R. M. Kling and Prithviraj Banerjee. Optimization by simulated evolution with applications to standard cell placement. In Proceedings of the 27th ACM/IEEE Design Automation Conference, DAC '90, pages 20 25, New York, NY, USA, 1990. ACM.

[26] T. Villa and A. Sangiovanni-Vincentelli. Nova: state assignment of finite state machines for optimal two level logic implementation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 9(9):905 –924, Sep 1990.

[27] A. El-Maleh, Sadiq M. Sait, and F. Nawaz Khan. Finite state machine state assignment for area and power minimization. In Proceedings of IEEE International Symposium on Circuits and Systems, 2006.