Multiobjective Finite State Machine Encoding using Non-Deterministic Evolutionary Algorithms

Aiman El-Maleh and Sadiq M. Sait
College of Computer Sciences & Engineering
King Fahd University of Petroleum & Minerals
KFUPM # 673, Dhahran-31261, Saudi Arabia
e-mail: {aimane,sadiq}@ccse.kfupm.edu.sa

November 24, 2004

Abstract

The rapid increase in complexity of VLSI circuits along with their proliferation in new domains have posed new challenges to the VLSI CAD industry. Mobile devices have given rise to new requirements such as low power in addition to conventional area (size) and Performance (timing) goals. The increase in chip complexity (size) has further increased the complexity of VLSI devices. With ever shrinking design to market windows, there has always been a strong need to develop synthesis tools to address such multi-objectives efficiently. One central problem in the synthesis of digital systems is controller synthesis which can be accomplished via FSMs, microprograms, etc. The complexity of FSM implementation depends on its state assignment. State assignment of FSMs for efficient area implementation alone is an NP-hard problem. The problem is further involved if we consider additional objectives such as low-power and ease of testability. Non-deterministic evolutionary heuristics such as Genetic Algorithms, Tabu Search and Simulated Evolution have shown to yield good results in solving such multiobjective hard combinatorial optimization problems in other areas of design automation. The objective of this work is to develop efficient tools for FSM state assignment to address area, power and testability issues. The main focus will be (a) to design suitable and efficient cost functions to evaluate different generated solutions, and (b) to effectively engineer

evolutionary heuristics for FSM state assignment in order to yield controllers with small area, low power dissipation, and ease of testability.

Keywords: VLSI Design, Iterative Algorithms, Low-Power, FSM Synthesis, State Assignment, Testability, Multiobjective Optimization, Meta-heuristics.

1 Introduction and Motivation

Technological advancements in Very Large Scale Integration (VLSI) have empowered the industry to integrate millions of transistors on a single chip. The contemporary approach adopted to address design of devices with high complexity is by following concepts of structured designing and design abstraction [1]. Abstraction is used to utilize the efforts of designers at higher levels. This allows fast initial prototyping with refinements left to be added at lower stages using detailed circuit information. Typical levels of abstraction, together with their corresponding functionalities, are illustrated in Figure 1. Computer Aided Design (CAD) tools automate the VLSI design process at all levels of design abstraction.

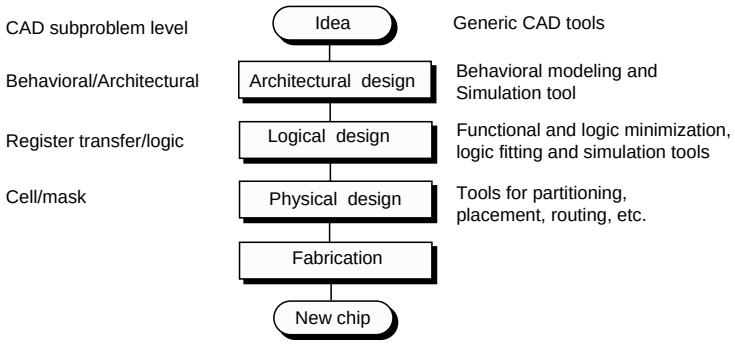

Figure 1: Levels of abstraction and corresponding design steps [2].

With the rapid increase in system functionality, new paradigms such as mobile computing have emerged. Mobile computing have added a new facet of power efficiency in the complexity of VLSI design [3]. At the same time, the increasingly complex VLSI devices are proving more and more difficult to test. Efficient testing is no longer the sole responsibility of test engineers and the focus is now on better design strategies to make the device more easily testable [4]. Thus, testability of a VLSI chip adds another objective to the increasingly complex task of designing VLSI circuits.

1.1 Motivation

The complexity of today's digital systems is tackled by partitioning and automating the design process using CAD tools. Digital Systems are broadly composed of two sub-components namely a controller and a datapath. The datapath performs all the arithmetic and logical operations required on the

data. The controller is responsible for controlling the sequence of operations on the datapath and is generally implemented as a finite state machine (FSM).

Traditionally, synthesis of FSMs was targeted for area minimization which itself is an NP-hard problem [5]. Various heuristics have been proposed and employed to address the NP-hard nature of the problem. The inclusion of power and testability optimization objectives, increases the difficulty of FSM synthesis. Therefore, efficient heuristics that address this hard multiobjective combinatorial optimization problem are needed.

Non-deterministic evolutionary heuristics/meta-heuristics like Genetic Algorithms, Simulated Evolution and Tabu Search have shown good results in solving various combinatorial optimization problems [6]. These heuristics try to optimize user defined goals of the problem encapsulated within a cost function. The quality of the solution depends on how closely the problem is modeled using the cost function.

The focus of this work is to address the complex problem of FSM state assignment targeting small area, low power and testability at a higher level, where there is greater degree of flexibility for trying out different alternatives. Design of various cost metrics will be investigated to develop models for estimating the quality of solutions that satisfy our objectives.

2 Finite State Machine State Assignment

State assignment in FSMs [5] is one of the main problems in the synthesis of sequential machines. The complexity of FSMs combinational circuit, hence the area, depends on its state assignment or encoding. Also, power dissipation, and testability are functions of state assignment. Thus, depending on the requirements, the assignment of states can be subject to different constraints. Gaining insight into the problem of assigning states can be fruitful in coming up with solutions which will lead to structures and complexity that will satisfy the above mentioned objectives and constraints. Below we discuss the relevant theory.

2.1 Encoding and Partitioning

The state assignment problem of an FSM can be viewed as a coding problem or as a partitioning problem [7, 8, 5]. The coding problem requires each state to be assigned a unique binary pattern. From the partitioning point of view, each state variable, y_i (one of the bits of the memory part of FSM), partitions

the assigned states into two sets. All states in one set are those for which y_i is 1, and those in the other set for which y_i is 0.

Therefore a partition on a set S of states is a collection of disjoint subsets whose set union is S. The disjoint subsets are called the *blocks* of the partition. A partition is called an m-block partition if the number of blocks in it are m.

The partition induced by a state variable y_i is represented with the Greek symbol Tau, $\tau(y_i)$. As an example, consider a machine M with four states (A,B,C,D) and a single input (x) as given in Table 1. The above state machine

PS	NS	
	x=0	x=1
A	A	D
B	A	C
C	C	B
D	C	A

Table 1: State Machine - 1

with a state assignment is shown in Table 2.

y_1y_2	Y_1Y_2	
	x=0	x=1
A $\rightarrow$ 00	00	10
B $\rightarrow$ 01	00	11
C $\rightarrow$ 11	11	01
D $\rightarrow$ 10	11	00

Table 2: A sample encoding for State Machine - 1

In the assignment used in Table 2, $y_1 = 0$ for states A and B, and $y_1 = 1$ for states C and D. Therefore, y_1 induces a 2-block partition $\tau(y_1) = (\overline{AB}; \overline{CD})$. Similarly, y_2 induces another 2-block partition $\tau(y_2) = (\overline{AD}; \overline{BC})$, on the states of machine M.

If every state of the state machine is assigned a unique code then the product of all the partitions is a partition that has as many blocks as the number of states. We call such a partition as a *zero partition* represented by $\pi(0)$. Mathematically

$$\prod_{i=1}^{k} \tau(y_i) = \pi(0) \tag{1}$$

where k is number of partitions (which is also the number of state variables).

For example, the product of the partitions induced by coding of Table 2, $\tau(y_1)$ and $\tau(y_2)$ is given as

$$\tau(y_1) \cdot \tau(y_2) = (\overline{A}; \overline{B}; \overline{C}; \overline{D}) = \pi(0) \tag{2}$$

where the dot operator $(\cdot)$ refers to the intersection operation on the states of blocks in the individual partitions.

The problem of state assignment is to find a set of partitions such that (1) is true.

Closed Partition

A partition is said to be closed if for any two states S_i and S_j which are in the same block, and for any input I_k, the next states denoted by $I_k.S_i$ and $I_k.S_j$ are in a common block of the partition. This condition must be true for all pairs of states in every block. Such a partition is said to be closed and is represented by π. For example, partition $\tau(y_1)$ in Table-2 is a closed partition. A closed partition is a special form of a partition in which the next block can be uniquely determined from the knowledge of present block and inputs. For example, suppose that r-state variables are assigned to a closed partition, where $r = \lceil \log_2 |\pi| \rceil$ ($|\pi|$ is the number of blocks in a closed partition) among k-state variables of the sequential machine, where $k = \lceil \log_2(n) \rceil$, n being the number of states. Then, according to the definition of a closed partition, the r state variables are independent of the remaining $k - r$ state variables. Closed partition is thus referred to as zero-dependency condition. In the above example, y_1 is independent of y_2. The equation for Y_1 is

$$Y_1 = \overline{X} Y_1 + X \overline{Y_1} \tag{3}$$

Note that the partition $\tau(y_2)$ is not a closed partition

Parallel and Serial Decompositions

The presence of closed partition indicates that some of the state variables can be independently determined irrespective of the other state variables. Thus, if we can find a set of closed partitions such that Condition (1) above is satisfied, then the machine can be decomposed into parallel sub-machines, equal to

PS	$y_1y_2y_3$	NS x1x2			
		00	01	10	11
A	000	A	C	D	F
B	111	C	B	F	E
C	010	A	B	F	D
D	110	E	F	B	C
E	100	E	D	C	B
F	111	D	F	B	A

Table 3: State Machine - 2

the number of closed partitions in the set, operating independently. Such a decomposition is referred to as parallel decomposition. Mathematically

$$\pi(1).\pi(2)\cdots\pi(k) = \pi(0) \tag{4}$$

However, if such a set of closed partitions could not be found, we need to find a partition denoted by T such that Condition (1) can be satisfied, i.e.,

$$\pi(1)\cdot\pi(2)\cdots\pi(v)\cdot T = \pi(0); \quad (v < k) \tag{5}$$

In such a case, the partitions $\pi(1)$ to $\pi(v)$ are still closed and so self-dependent. However, the partition T is not closed and so is dependent on state variables other than those assigned to itself. This yields a serial decomposition of a state machine in which independent subsets of the state machine feed the dependencies required for dependent subset of the machine.

Partition Pairs

The structure of sequential machines is much more complicated than a bunch of parallel or serially connected sub-machines. There are sub-machines that are cross dependent. The concept of partition-pairs helps analyze such dependencies.

A partition pair *(T, T')* on the states of a sequential machine **M** is an ordered pair of partitions such that, if S_i and S_j are in the same block of T, then for every input I_k in I, $I_k.S_i$ and $I_k.S_j$ are in the same block of T'. The partition T is called the predecessor partition and T' the successor partition.

Consider a state machine with a state assignment shown in Table-3. Partitions induced by state variables, y_2 and y_3 are given as

$$\tau(y_2) = \tau_1 = (\ \overline{A, E};\ \overline{B, C, D, F}) \tag{6}$$

$$\tau(y_3) = \tau_2 = (\ \overline{A, C, D, E};\ \overline{B, F}) \tag{7}$$

Clearly, (τ_1, τ_2) form a partition pair since the next state at any input for a pair of states in a block in τ_1 lie in some block of τ_2. τ_1 is said to be predecessor partition and τ_2 the successor. Thus, to uniquely determine the next block in the successor partition, one needs to know the present block in the predecessor partition along with inputs. That is to say that the successor partition is dependent on the information of the state variables that induce the predecessor partition. Thus, a partition pair can be thought of as one (or single) dependency condition.

P-Dependency Condition

A P-dependency condition, where P is greater than one, can be derived in a similar manner. This requires the computation of what is known as *Mm-pairs* [5].

A partition *M(T')* is the summation (union) of all partitions T_i such that *(T_i, T')* is a partition pair. Thus, *M(T')* is the largest partition, i.e., a partition containing the biggest blocks whose successor blocks are contained in *T'*. Similarly, a partition *m(T)* is the product (intersection) of all partitions $T_{i'}$ such that *(T, $T_{i'}$)* is a partition pair; where *m(T)* is the smallest partition, i.e., a partition containing the smallest blocks that can be the successors of the blocks of *T*.

The P-dependency condition states that if the next-state variable Y_i can be computed from the external inputs and a subset P_i of the state variables, then the product of partitions induced by the subset P_i should be contained within M of the partition induced by y_i. Mathematically

$$\prod_{y_j \in P_i} \tau(y_j) \subseteq M[\tau(y_i)] \tag{8}$$

where $\tau(y_k)$ represents the partition induced by variable y_k. The product is taken over all $\tau(y_j)$, such that y_j is contained in the subset P_i.

The P-dependency condition is also referred to as information flow inequality [5]. The condition can be used to find the number of dependencies required for state variables.

Consider the state machine whose state table is given in Table-4. We begin by finding the smallest partitions implied by pairs of states. Let τ_{AB} be parti-

tion that includes a block (AB) and leaves all other states in separate blocks ($\tau_{AB} = (\overline{AB}; \overline{C}; \overline{D}; \overline{E})$. Then, by definition, the smallest partition containing the block implied by τ_{AB} is $m(\tau_{AB})$, which can be determined by looking at the successive or next states of states in τ_{AB}.

PS	NS				z
	I_0	I_1	I_3	I_2	
A	C	A	D	B	0
B	E	C	B	D	0
C	C	D	C	E	0
D	E	A	D	B	0
E	E	D	C	E	1

Table 4: State Machine - 3

$$m(\tau_{AB}) = \{\ \overline{A, C, E}; \overline{B, D}\ \} = \tau_1'$$

Clearly, $(\tau_{AB}, m(\tau_{AB}))$ is a partition pair.

Similarly, the rest of the smallest partitions implied by other pair of states can be found.

$$\begin{aligned}
m(\tau_{AC}) = m(\tau_{DE}) &= (\overline{A, C, D}; \overline{B, E}) = \tau_2' \\
m(\tau_{AD}) = m(\tau_{CE}) &= (\overline{A}; \overline{B}; \overline{C, E}; \overline{D}) = \tau_3' \\
m(\tau_{AE}) = m(\tau_{CD}) &= (\overline{A, B, C, D, E}) = \pi(I) \\
m(\tau_{BC}) = m(\tau_{BE}) &= (\overline{A}; \overline{B, C, D, E}) = \tau_4' \\
m(\tau_{BD}) &= (\overline{A, C}; \overline{B, D}; \overline{E}) = \tau_4'
\end{aligned}$$

Let the three state variables needed to encode the 8-states be $y1$, $y2$ and $y3$ and their partitions represented as τ_{y1}, τ_{y2}, τ_{y3} respectively. Then the problem of state assignment is to encode $y1$, $y2$ and $y3$ such that

$$\tau_{y1} . \tau_{y2} . \tau_{y3} = \pi(0)$$

One such state assignment can be

$$\begin{aligned}
\tau_{y1} &= (\overline{A, C, E}; \overline{B, D}) \\
\tau_{y2} &= (\overline{A, B, D}; \overline{C, E}) \\
\tau_{y3} &= (\overline{A, C, D}; \overline{B, E})
\end{aligned}$$

The corresponding M of the above partitions are found out as follows

$$M(\tau_{y1}) = \tau_{AB} + \tau_{AD} + \tau_{CE} + \tau_{BD} = (\overline{A, B, D}; \overline{C, E})$$
$$M(\tau_{y2}) = \tau_{AD} + \tau_{CE} = (\overline{A, D}; \overline{B}; \overline{C, E})$$
$$M(\tau_{y3}) = \tau_{AC} + \tau_{DE} = (\overline{A, C}; \overline{B}; \overline{D, E})$$

where the operator + is the union of two partitions defined as union of every two blocks in the two partitions provided that the intersection of the two blocks is not empty.

We can now use information flow inequality 8 to find out dependencies of state variables for the given state assignment. The inequality states that dependency of a state variable inducing partition τ_{yi} is equal to the smallest subset of the product of partitions $\tau_{y1}\tau_{y2}\tau_{y3}$ that is lesser or equal to $M(\tau_{yi})$. Thus, we see that

$$\tau_{y2} = M(\tau_{y1})$$
$$\tau_{y2}.\tau_{y3} \subset M(\tau_{y2})$$
$$\tau_{y1}.\tau_{y3} \subset M(\tau_{y3})$$

Consequently, Y1 is dependent on y2 while Y2 depends on the information supplied by y2 and y3. Similarly, Y3 receives its inputs from y1 and y3. Thus,

$$Y_1 = f_1(Inputs, y_2)$$
$$Y_2 = f_2(Inputs, y_2, y_3)$$
$$Y_3 = f_3(Inputs, y_1, y_3)$$

3 Literature Review

In this section, a detailed survey of heuristics for FSM synthesis for area, power and testability is reported. FSM synthesis for area is usually targeted independently for two-level and multi-level realizations. FSM synthesis for power involves calculating transition probabilities between states and to reduce switching involved. FSM synthesis for testability is concerned with reducing the sequential depth and the number of loops in the synthesized circuit. Description of various cost models to model the optimization objectives at a higher level are also detailed.

Review for FSM synthesis strategies is followed by a survey of iterative heuristics, namely genetic algorithm, tabu search and simulated evolution. These heuristics have shown good results in optimizing hard combinatorial problems.

3.1 FSM Encoding for Area

The encoding for a finite state machine (FSM) determines its combinational component. The number of storage bits n_b used to store the state assignment also affects the encoding and so the FSM's complexity. The area of an FSM is also a function of the type of flip-flop being employed. Encoding for finite state machines have traditionally been targeted for reducing the complexity of its combinational part. A good survey of FSM encoding for area can be found in [9].

The use of D-type flip-flops is most prevalent in VLSI circuits today. This work will also be implicitly using D-type flip flops for storing the finite state machine's state assignment.

The minimum number of state variables needed for state assignment is given as

$$r_0 = \lceil \log_2(n) \rceil \tag{9}$$

where n is equal to the number of states of the FSM.

An assignment using the minimum number of state variables has the benefit of using the minimum number of storage devices. However, with such an assignment, there is a potential of reduced flexibility in satisfying the number of encoding constraints (discussed later). The problem is further investigated in [10, 11, 12, 13, 14, 15]

Even if we consider minimal state assignments with D type flip flop, the number of possible combinations is exhaustively large [16].

$$N = \frac{(2^{n_b} - 1)!}{(2^{n_b} - n)! r_0!} \tag{10}$$

Thus, exhaustive evaluation is not feasible and heuristics are generally employed to tackle the problem of FSM encoding.

Logic minimization aims to optimize the combinational logic of an FSM. This in turn depends on the degree of freedom provided by an efficient FSM encoding. A good encoding can help the logic minimizer to achieve a better realization in terms of logic cost. Logic minimizers employ different heuristics for two-level and multi-level circuits as their cost measures differ.

A two level implementation realizes a logic function as a sum of product terms. The circuit complexity of such a representation is related to the number of inputs, outputs, number of product terms and number of variables utilized in a product term, i.e. the number of literals.

The simplest way to encode an FSM is by assigning 1-hot state codes. In 1-hot encoding for a state, the corresponding code bit for a state is set to 1 and all others to 0. Thus, 1-hot encoding is a case of non-minimal state assignment such that the number of variables required is equal to the number of states. It is further noticed [17, 18] that such an encoding is poor to minimizing the size in sum of products representation.

An objective of state encoding could be to reduce dependencies among states [19, 20]. The rationale is that by having dependencies reduced, literal count will decrease and so will interconnect. However, reduced dependencies correlate weakly with the minimality of sum of products representation.

The complexity of a two level realization can be reduced by using mechanisms such as implicant merging, code covering and disjunctive coding [9]. The idea behind ***Implicant Merging*** (See Table-5) is to assign adjacent codes to states that produce either same next-state or output or both at similar input conditions. This yields bigger cubes while doing Karnaugh minimization, and results in a simpler final expression. Implicant merging requires adjacency constraints to be met by the state assignment algorithm. ***Code Covering*** involves a code word of a state covering a code word of some other state(s), i.e. all the bit positions for which the second code word is 1, correspond to 1 in the first code word. An example utilizing code covering is illustrated in Table-6. Assume that S_1 is encoded with 110 and S_2 with 100. In this case, the input condition (S, 001) can be treated as don't care condition for the next state S_2, reducing the cover cardinality for S_2 from three to two cubes. Covering constraints produce covering codewords. Reducing cover cardinality using ***Disjunctive Coding*** is illustrated in Table-7. Disjunctive constraints require that the disjunction of state codes is equal to some other state code. In the example shown, the states are encoded such that the code for S_2 is the disjunction of the state codes for S_1 and S_3. As such, the second implicant with the input field 101 gets contained in other input conditions and thus is completely saved.

The major difficulty for 2-level realization of an FSM is the simultaneous consideration of all the types of constraints [15]. In general, it is not possible to satisfy all the coding conditions with a code using the minimum number of bits r_0. By increasing the number of code bits to $r > r_0$, more coding constraints can be satisfied. The increase in the number of storage elements and state signals to be generated has to be justified against the potential of reducing combinational logic by satisfying additional coding constraints.

The problem with many approaches to two-level assignment is that no exact predictions are possible, as to how the satisfaction of coding conditions

PS	I	NS	z
S_1	i	S	o
S_2	i	S	o
S_3	i	S	o
0--	i	S	o

Table 5: Implicant Merging

PS	I	NS	z
S	001	S_1	o
S	000	S_2	o
S	01-	S_2	o
S	001	110	o
S	0--	100	o

Table 6: Code Covering

affects the complexity of the resulting combinational logic, since the different coding conditions interact with each other in a complex way. The application of coding constraints and finding out their effect would be excessively costly as it would require a huge number of logic minimizations. To mitigate this problem, [12] proposed an elegant solution of ***symbolic minimization***. By using symbolic minimization techniques, it is possible to optimize the function independently of the encoding and determine the codes at a later time. This requires performing the minimization at the symbolic level, before the encoding.

In contrast to two level circuits, multiple level circuits provide much more degree of freedom in optimizing combinational network. This is because of the flexibility provided due to operations such as common sub-expression extraction and factorization. Unfortunately, it comes with an increase in the

PS	I	NS	z
S	001	S_1	o
S	101	S_2	o
S	111	S_3	o
S	-01	100	o
S	1-1	010	o

Table 7: Disjunctive Coding

difficulty of modeling and optimizing the multi-level network themselves.

The complexity measure for multi-level circuits is the encoding length and the number of literals in the optimized logic network. Since encoding length is mostly taken constant, literal saving by extracting common sub-expressions has been the focus of most of the work done for multi-level FSM optimization. This involves finding the state pairs which when encoded carefully can result in extracting common sub-expressions. In contrast to two level circuits, state pairs in multi-level implementations do not necessarily have to be given adjacent codes. If two states have n state bits in common, the combination of the two states result in a common sub-expression with n literals. To identify the states to assign close codes, two heuristics proposed by [21, 22] standout. The first called fanout oriented, tries to assign closer codes to the states that have same next state transition. The rationale is to maximize the size of common cube by assigning closer codes (lesser hamming distance) to such states. The second approach is referred as fanin oriented in which state pairs with incoming transitions from the same states are given high weights for closer code assignment. Here the motivation is to maximize the frequency of common cubes in the encoded next state function. The schemes are improved upon in [23]. Rules for detecting potential common cubes and formulae for more precise evaluation of literal savings have been proposed in [24]

There have been a few attempts of utilizing genetic algorithm for solving state assignment problem [25, 26]. Almaini et al [25] utilize ESPRESSO tool in SIS for their cost calculation which though being accurate is computationally infeasible. Amaral et.al use a cost function proposed by Armstrong [27]. The cost model tries to combine the properties of fanin and fanout oriented

algorithms. The contribution in the above work is in the design of genetic algorithm for state assignment problem. However, the authors did not try to take advantage of the availability of the state codes in cost function computation.

In Jedi [23], the encoding affinity cost is modeled as a function of how many times a pair of states are represented in next state and output functions. The cost function of Jedi is given in equation-11.

$$J^P_{k,l} = \sum_{i=1}^{m_0}(P^o_{k,i} + P^o_{l,i}) + \frac{n_E}{2}\sum_{i=1}^{n_s}(P^s_{k,i} + P^s_{l,i}) \tag{11}$$

where,

$P^o_{k,i}$ is the number of times state k is represented in output i,

$P^s_{k,i}$ is number of times state k is represented in state i,

m_o is the number of outputs,

n_s is the number of states,

n_E is the number of encoding bits.

For example, consider the state machine in Table-4. The next state equations for states A-E are given as.

$$\begin{aligned} A &= A.I_1 + D.I_1 \\ B &= A.I_2 + B.I_3 + D.I_2 \\ C &= A.I_0 + B.I_1 + C.I_0 + C.I_3 + E.I_3 \\ D &= A.I_3 + B.I_2 + C.I_1 + D.I_3 + E.I_1 \\ E &= B.I_0 + C.I_2 + D.I_0 + E.I_0 + E.I2 \end{aligned}$$

and the output equation is given as

$$O_0 = E$$

Here, $P^s_{C,D}$ is number of times state C is present in next-state equation of state D which is equal to one. Similarly, $P^s_{E,E}$ is two. $P^o_{C,0_0} = 0$ and $P^o_{E,0_0} = 1$

Once the affinity cost for every state is calculated, Jedi uses Simulated Annealing to minimize 12

$$\sum_{i=1}^{n_s}\sum_{j=1}^{n_s} = J^P_{i,j}.\Delta(i,j) \tag{12}$$

where $\Delta(i,j)$ is the Hamming distance between codes of state i and j.

Mustang [22], observed that if $P^{s/o}_{k,i} = 50$ and $P^{s/o}_{l,i} = 2$, states k and l are less strongly connected than they would be when $P^{s/o}_{k,i} = 26$ and $P^{s/o}_{l,i} = 26$, even though the sums are the same. They thus proposed use of multiplication in place of addition to represent encoding affinity. The cost function of Mustang is given in equation-13

$$M^P_{k,l} = \sum_{i=1}^{m_0} (P^o_{k,i} * P^o_{l,i}) + \frac{n_E}{2} \sum_{i=1}^{n_s} (P^s_{k,i} * P^s_{l,i}) \quad (13)$$

Jedi and Mustang both try to reduce the Hamming distance between highly recurring states in the next state equations. However, since it is the flip flop equations that are synthesized and not the next state equations, the measures contain inherent inaccuracy. Moreover, Jedi does not give weightage to pair of states that appear together in next state function. This may yield codes for a pair of states that though being closer in Hamming distance, appear in different flip-flop functions and so the reduction in Hamming distance could not be utilized in reducing the logic. Mustang, by the virtue of its multiplication operation, is immune to such a situation. Moreover, both Jedi and Mustang do not try to reduce two-level logic in their cost models.

In this work we will evaluate the various cost functions and proposed in the literature with regards to their correlation with the multi-level implementation cost. Furthermore, we will investigate an efficient cost measure that correlates well with the multi-level implementation cost.

3.2 FSM Encoding for Low Power

Power dissipation has always been one of the major concerns in logic circuits design. Excessive power dissipation often causes chip run-time failure, reduction in chip life-time, and costs more expensive packaging. In recent times, portable electronics applications have given power-aware computing a whole new importance. This is due to the fact that limitations in battery capacities and progress trail far behind the ever increasing computing requirements. Power consumption is thus constrained and optimized at all levels of design hierarchy including technology selection, architectural transformation, logic synthesis and physical design [3]. VLSI designers have thus been faced with another optimization parameter of low power. Recently, a lot of work is reported in the literature to automate the exploration of low power solutions at different levels of VLSI hierarchy [3, 28]

Power Estimation for FSMs

The exact power consumption of a VLSI device is a complex function of many parameters and thus can only be accurately found out by running numerous power simulations on the final device. However a simpler measure for power dissipation by a CMOS logic gate can be found out by the following equation.

$$P_{ave} = \frac{C_L V dd^2 E_{SW}}{2.T_{cyc}} \tag{14}$$

where T_{cyc} is the cycle time, C_L the load capacitance of a CMOS gate and E_{SW} being the expected switching activity at the gate's outputs.

The above equation shows that by reducing switching, supply voltage or capacitance seen by the gate, the power consumption of a CMOS device can be reduced.

There is a rich amount of work reported in the literature for power estimation of sequential circuits [29, 30, 31, 32]. The power estimation techniques can be broadly classified into statistical [33] or probabilistic [34]. Both the approaches are implemented in SIS [35] version 1.2. The statistical approaches work by simulating the state machine using the user provided input vectors and determining the state probabilities based on it. Probabilistic approaches on the other hand try to correlate the various probabilities in order to calculate state probabilities if the FSM is simulated for infinite amount of time. Statistical techniques can be fast and accurate if a short representative sequence for an FSM can be determined. However, determining such a sequence is an open research problem. [36] reports a statistical power estimation technique using randomly generated input sequences until a desired accuracy is achieved. Najm et al in [37] propose a technique to estimate power within a desirable accuracy of an FSM by simulating fraction of a large input set. The technique tries to simulate FSM repeatedly by blocks of consecutive vectors at random until a desired accuracy is achieved. A Monte-Carlo approach for power estimation for sequential circuits is also proposed [38]. The technique generates mutually independent power samples using multiple copies of the circuit that are simulated in parallel with mutually independent input vector streams. Samples are collectively analyzed to check for the terminating condition.

The power estimation problem is addressed even at a more higher level using entropy as power estimating function [39, 40]. The rationale is that since entropy is a measure of information-carrying capacity, a higher entropy on a state line means higher number of transitions on it. The maximum transition can be attributed when the probability on a line is exactly half and corresponds to its maximum entropy value.

A state transition graph (STG) is denoted by *G(V, E)* where a vertex $S_i \in V$ represents a state of the FSM and an edge $e_{i,j} \in E$ represents a transition from state S_i to S_j. Let P_{Si} denote the state probability, that is, the probability of finding the state machine in S_i at any given time, and p_{ij}

denotes the conditional (state) transition probability, which is the probability of the machine making a transition from state S_i to state S_j, that is

$$p_{ij} = Probability(Next = S_j | Present = S_i) \tag{15}$$

A STG can be interpreted as a Markov chain. A Markov chain is a representation of a finite state Markov process [41]. A Markovian process is termed as memory-less since the probability distribution at any time depends only on the present time and not on how the process arrived till that period. For a large class of Markovian processes for which our STG is also a member, the probability of a state is the limiting value approached as it is run for infinite amount of time. This is termed as *limiting state probability theorem* [42]. Mathematically

$$limit_{n->\infty} p_{ij}(n) = P_{Sj} \tag{16}$$

The above can be iteratively found out by solving Chapman-Kolmogorov equations [43] as follows

$$\begin{aligned} P_{Si}(n+1) &= \sum_{j \in In\_State(i)} p_{ji} P_{Sj}(n) \\ & i = 1, 2, \cdots, M-1 \\ 1 &= \sum_{j} P_{Sj}(n+1) \end{aligned} \tag{17}$$

where n is iteration number and $In\_State(i)$ is the set of fanin states of i in the STG.

The process is terminated once state probabilities converge so that the difference between successive iterations is within a user defined tolerance value. To tackle the complexity of solving the above system of equations, approximate methods have been proposed in [44, 45]

The *Total State Transition Probability* for a transition from a state S_i to state S_j is the probability that the machine transits to state S_j from state S_i. The total state transition probability can thus be calculated [46] as follows

$$P_{ij} = p_{ij}.P_{Si} \tag{18}$$

where P_{ij} is the total state transition probability from state S_i to state S_j.

The sum of total state transition probabilities in between two states indicates the amount of switching in between them. This sum can be treated as a weight between the two states attributed on a single edge connecting them.

$$W_{ij} = P_{ij} + P_{ji} \tag{19}$$

A STG in which all the transitions between two states are replaced with a weighted edge is called a *weighted graph*. The weight on an edge indicates the relative proximity in the state assignment of the two connected states on that edge. By assigning shorter distance codes to states connected with higher weights, i.e higher transition probability, the overall switching on the state lines of the FSM can be minimized. Thus a cost model for minimizing power consumption can be to have *Minimum Weighted Hamming Distance* (MWHD). Mathematically

$$\sum_{S_i S_j \in S} W_{ij} H(S_i, Sj) \tag{20}$$

Previous Work

Most of the work reported in the literature [47, 48, 49] tries to achieve minimum weighted hamming distance by optimizing the above equation for low power realization of FSMs.

However, as (14) shows, power consumption depends on how much capacitance is switched. A reduced amount of switching on greatly increased load capacitance may well offset any savings achieved. Thus, by reducing the switching activity, the problem is only half solved. However, the knowledge of the gate loading can only be accurately found out once the design is synthesized and mapped on a specific library.

Kang et al in [50] try to take into account area into their cost equation for low power FSM realization. The cost function used is a linear combination of minimum weighted hamming distance for power and the literal savings by Jedi for area. However, since there is no correlation between the two terms, the technique does not aim at minimizing switched capacitance but merely tries to achieve a low power and area FSM solution. The rationale being that a low area solution will anyhow contribute towards a low power solution. The problem is solved using genetic local search algorithm.

Suresh et al [51] describe a modification of MWHD scheme. The algorithm tries to identify code swaps between states such that the final cost in terms of weighted switching can be reduced. The authors define base switching as the minimum amount of switching that is possible if the all the states are assigned a unidistance code. Relative switching is defined as a measure of goodness of how close the average switching is to the minimal possible base switching value. The algorithm then identifies 'slack' values which is the amount by

which the cost can be decreased if two state codes are exchanged. Edges with high slack values are first identified, then sorted and finally those that yield better costs are exchanged. The algorithm terminates when there is no more good exchanges remaining. The algorithm suffers from complexity of $O(n^3)$ as for every exchange, it has to take care of its effect on other edges connected to the two nodes in focus. Moreover, the greedy algorithm proposed is vulnerable to get stuck in a local minima.

Roy et al addressed the problem of minimizing power in sequential circuits in Syclop [52]. The authors use conditional transition probabilities in place of steady state probabilities while solving a MWHD solution. The hard nature of the problem is addressed by using simulated annealing algorithm. Once the state codes with reduced MWHD cost are found out, constrained multi-level logic synthesis is performed. A set of kernels are computed for each logic expression and a non-trivial intersection of kernels is selected so that fanout for nodes having high transition density can be reduced. The rationale is that reduced fanout on highly switched state lines will result in low switched capacitance.

A MWHD scheme is employed for non-minimal state encoding by Koegst et al in [53]. The authors advocate the use of a user specified input sequence for measuring total state transition probabilities, and thus weights, instead of equation-17. Koegst in [54] used a multi-criteria non-minimal state assignment for low power where assignment helps deactivating idle parts of FSM along with reducing MWHD.

A novel technique for low power state assignment is proposed by Majid et al [55]. The authors note that an optimal solution for MWHD problem can be obtained using Integer Linear Programming (ILP). However, any such technique suffers from exponential complexity of ILP itself. This can be mitigated if ILP has to be applied on small finite sets. They thus proposed a semi-gray encoding technique in which the states are partitioned into small groups in decreasing order of their weights. The states within a group are then assigned gray codes using ILP.

A low power FSM realization is proposed using Huffman style to provide non-uniform state codes in [56]. The technique proposes shorter codes for states with higher switching activities and more for lesser switched states. The rationale being that lesser state lines will yield lesser weighted switching as well as switched capacitance. However, the overhead of the scheme barred the authors to implement it as it is. Instead, the state set is encoded using only two different code lengths. Moreover, a logic is proposed to shut off clock for the inactive set.

Another interesting variation in MWHD approach is proposed by Silvano et al [57]. The authors note that the state assignment procedure can be broken down into state ordering and state encoding sub-steps. For state ordering, various techniques have been proposed so that a chain of highly probable states is formed. The rationale in doing so is that consecutive states from a highly probable state are more likely to be visited than stand alone nodes with high probability. Once states are ordered, they are encoded using encoding techniques described in [58]. The sate encoding techniques try to reduce hamming distance between consecutive states in the state ordering list as well as the states that are connected to those states.

Benini [59] proposed state assignment technique for low power based on total state probabilistic MWHD algorithm. The authors propose the use of a greedy variation of column-based encoding [18, 60]. The cost function also tries to minimize area using cost metrics of multi-level minimizers of Jedi and Mustang. However, the cost function again lacks any correlation between area being saved and power attributed to it. Thus, the technique is essentially aiming for low power and low area solutions independently.

Pedram et al in [61] describe a novel technique for low power state assignment by introducing the concept of literal power savings in MWHD algorithm. The authors propose the usage of area saved times its power savings as the cost function. Power cost models for both two and multi-level logic implementation are described. The cost models are power extension for area cost models for two-level and multi-level circuits. Simulated Annealing algorithm is used as search strategy.

Another interesting work for power and area minimization is presented in [62] by Chao et al. The authors use entropy measure to calculate the probability distribution of an FSM. They then distribute the number of possible codes into groups such that the codes within a group have equal number of ones. Each state is then assigned to a group so as to minimize the overall switching. A state is finally assigned a unique code within a group using literal saving estimates.

In some recent work, Almaini et al [63] have employed Genetic Algorithm [6] to solve the problem of MWHD. The cost function aims at optimizing both area and power separately. The area estimate is the number of cubes in a synthesized circuit. Pomeranz et al [64] have also used Genetic Algorithm to partition the FSM such that inactive partitions be turned off to reduce power. They propose to do state encoding such that it can also determine the partition as well as state assignment.

In this work we will investigate the use of efficient power cost estimators

including the one proposed in [61].

3.3 FSM Encoding for Testability

Testability of a VLSI circuit is attributed to how efficiently the various faults in the circuit can be excited and observed. This involves generation and application of test sets at primary inputs of a circuit to excite its various faults and observing them at the outputs. The test sets can either be manually generated or using automatic CAD tools. Automatic test generation tools are efficient in terms of cost and effectiveness and so are generally employed to find the test patters. This work will also consider the use of Computer Aided Automatic Test Pattern Generation (ATPG) tools.

ATPG tools use both random and deterministic techniques to build the test set. Deterministic test set takes into account the behavior and structure of the circuit under test to build its test set. They thus yield higher fault coverage though being more computationally expensive. The complexity and type of a circuit, whether combinational or sequential, determines how efficiently automatic test pattern generator performs.

Test generation for combinational circuits is known to be NP-hard problem [65]. The worst case size of the search space is bounded by 2^i, where i is equal to the number of inputs. However, techniques have been developed to reduce this large search space by an intelligent search of the primary input combinations. These techniques include D-algorithm [66, 67], PODEM [68], and FAN [69]. ATPG tools based on these algorithms are quite efficient in finding test patterns to detect all the testable faults in an integrated circuit.

Automatic test pattern generation for sequential circuit is much more involved than combinational circuits. Unlike combinational ATPG, existing sequential ATPG tools may not produce satisfactory results for some class of circuits due to their complexity. For this reason, design for test techniques like partial scan [70] have been used to improve the testability of the circuit. The increased complexity of sequential circuits arises from memory feature in their behavior. Thus, to excite a fault, the memory elements have to be first initialized and then the fault has to be propagated to the primary outputs. Finally, a justification sequence is to be derived that traverses the circuit from the initialized state to the current state of the circuit. These sequences require processing the sequential machine in multiple clock periods. Thus, sequential test generation involves a time domain component. To cope with this difficulty, *Iterative Array Model* was proposed [4]. The model transforms the time domain aspect of sequential circuit into space domain by unrolling the sequen-

tial behavior into multiple iterations of its combinational circuit, effectively making it as a large combinational circuit. The iterative model thus permits the automatic test pattern generation algorithms for combinational circuit to be extended to sequential logic.

A sequential circuit can be classified as cyclic or acyclic. If a node can be revisited after starting from that node in the forward direction without visiting any other node again, then a cycle is said to be present in the sequential circuit. The length of the cycle (cycle length) is said to be the number of sequential elements encountered during the traversal. Sequential depth refers to the number of sequential elements from primary input to the primary output.

The complexity of an ATPG can be attributed to the time it takes to attain the required level of test completeness. This in turn is a strong function of the complexity of the circuit. The upper bound on the number of vectors needed to test all testable faults in an acyclic sequential circuit with i inputs and sequential depth d is $d * 2^i$ [71], which is comparable to the complexity of a combinational circuit. However, a cyclic sequential circuit may require an initialization sequence to test the combinational logic in a given state. This initialization sequence can be as long as the $M-1$, M being the number of possible states for the state machine. Thus the upper bound for a cyclic sequential circuit having i primary inputs and M states, using s number of sequential elements is $M * 2^{s+I} = 2^{2s+I}$ [71]. It clearly shows that the complexity for ATPG of sequential circuit increases exponentially with the number and length of the cycles. The complexity of sequential ATPG is investigated by Lioy et al in [72]. The authors contend that that the complexity of sequential ATPG depends on the number of flip-flop per loop (FF/L) and the number of loops per flip-flop (L/FF). The former estimates the cyclic structure of the circuit and the latter predicts how much the design is 'winded up' on itself or how much interdependence exists between the loops. The authors note that the test generation complexity increases with FF/L and L/FF, while increasing the number of state-controlling inputs reduces its complexity. The authors further propose a formal algorithm to identify the loops within a sequential circuit. Marchok et al in [73] also note that the complexity of sequential ATPG varies with re-timing. Furthermore, the authors cite a new factor, *density of encoding*, which gives the measure of degree of valid states compared to the number of possible states in the state machine, to be key indicator in the complexity of structural sequential ATPG. The complexity of ATPG is carefully investigated in [74].

As described earlier, the nature of encoding strongly determines the structure of the sequential circuit, its various dependencies, cycles and intercon-

nections. The information flow inequality of state machines (8) enables us to quickly realize its structure prior to its synthesis. This can help provide an accurate measure of complexity of a sequential circuit at a higher level of abstraction.

Pomeranz et al [75] explored the possibility of controlling more state lines in order to increase the testability of a sequential circuit. They have proposed a synthesis technique that evaluates some state variable functions using primary inputs or primary output functions. The motivation is that since primary outputs are directly observable and primary inputs directly controllable, an increase in testability can be achieved.

Cheng et al [76] have proposed a novel method of encoding that reduces the feedbacks in a sequential circuit. The motivation is to reduce the cyclic nature of the sequential circuit. The authors propose state encoding by following states merging according to some rules. The first rule tries to maximize the number of blocks in a partition while the second tries to merge two states having the same next state. The rationale for the former rule is that by having a large number of blocks in a partition, more information can be derived from primary inputs alone for the next state function, that in turn reduces the number of feedbacks. The latter rule aims at area minimization by incorporating the commonly used cost metrics used in multi-level area minimization for a sequential circuit.

Mo-hat et al in [77] try to take into account the testability for PLA-based FSMs. The authors propose K-hot encoding scheme to deal with various types of PLA faults. In K-hot code, exactly K-bits are set equal to 1. The rationale is that many types of PLA faults can be easily detected if exactly K lines are not high.

Prinetto et al in [78] discuss testability measure for inputs and outputs of an FSM. The authors note that optimal testability is achieved when outputs are high for half of the possible inputs and low for the other. They further note that such a condition runs counter to power minimization condition where the aim is to have reduced switching.

In this work we will use testability measures based on loop count and sequential depth as cost estimators.

3.4 Iterative Algorithms

A number of iterative algorithms are proposed in the literature. The motivation for using iterative algorithms becomes clear when recalling the hard nature of the FSM encoding problem as mentioned above. These algorithms

are capable of efficiently searching for a near optimal solution in a large solution space and have been very successful in solving a number of combinatorial optimization problems in various disciplines of science and engineering. In the following, a brief description of genetic algorithm (GA), tabu search (TS), and simulated evolution (SimE) is presented.

Genetic Algorithm (GA)

GA is an elegant search technique that emulates the process of natural evolution as a means of progressing towards the optimal solution. A high level algorithmic description of GA is given in Figure 2 [6]. GA uses an encoded representation of a solution in the form of a string made up of symbols called *genes*. The string of genes is called *chromosome*. The algorithm starts with a set of initial solutions called *population* that may be generated randomly or taken from the results of a constructive algorithm. Then, in each iteration (*known as generation in GA terminology*), all the individual chromosomes in the population are evaluated using a *fitness function*. Then, in the *selection* step, two of the above chromosomes at a time are selected from the population. The individuals having higher fitness values are more likely to be selected. After the selection step, different operators namely *crossover, mutation,* and *inversion* act on the selected individuals for evolving new individuals called *offsprings*. These genetic operators are described below.

Crossover is an important genetic operator. It is applied on two individuals that are selected in the selection step to generate an offspring. The generated offspring inherits some characteristics from both parents in a way similar to natural evolution. There are different crossover operators namely *simple, order, partially mapped, and cycle*. The simple crossover operation for instance, works by choosing a random cut point in both parent chromosomes (the cut point should be the same in both parents) and generating the offspring by combining the segment of one parent to the left of the cut point with the segment of the other parent to the right of the cut [6]. For description of other crossover operators, see [79, 6, 80].

The *mutation* operator is used to introduce new random information in the population. It helps to prevent the search process from trapping in local minima. An example of mutation operation is the swapping of two randomly selected genes of a chromosome. The importance of this operation is that it can introduce a desired characteristic in the solution that could not be introduced by the application of the crossover operator alone. However, mutation is applied with a low rate so that GA does not turn into a memory-less search process [79].

```
Algorithm (Genetic_Algorithm)
  (N_p = Population Size)
  (N_g = Number of Generations)
  (N_o = Number of Offsprings)
  (P_i = Inversion Probability)
  (P_μ = Mutation Probabilty)
  Begin
    (Construct initial population)
    Construct_Population(N_p);
    For j = 1 to N_p
      Evaluate_Fitness (Population[j])
    EndFor;
    For i = 1 to N_g
      For j = 1 to N_o
(Choose parents with probability proportional to fitness value)
        (x,y) ← Choose_parents;
        (Perform crossover to generate offsprings)
        offspring[j] ← Crossover(x,y)
        For k = 1 to N_p
          With probability P_μ apply Mutation (Population[k])
          With probability P_i apply Inversion (Population[k])
        EndFor;
        Evaluate Fitness(offspring[j])
    EndFor;
    Population ← Select(Population, offspring, N_p)
  EndFor;
  Return highest scoring configuration in population
End. (Genetic Algorithm)
```

Figure 2: Outline of simple Genetic Algorithm [6].

The quality of the solution obtained from GA is dependent on the choice of certain parameters such as population size, number of generations, crossover and mutation rates and also the type of crossover used. The selection of values for these parameters is problem specific and so there are no hard and fast rules for this purpose. The choice of these parameters is left to the conception and intuition of the person applying GA to a specific problem.

Tabu Search (TS)

Tabu search is an iterative heuristic that has been applied for solving a range of combinatorial optimization problems in different fields [6]. Tabu search starts from an initial feasible solution and carries out its search by making a sequence of random moves or perturbations. A *tabu list* is maintained that stores the attributes of a number of previous moves. This list prevents bringing the search process back to already visited states. In each iteration, a subset of *neighbor* solutions is generated by making a certain number of moves and the best move (the move that resulted in the best solution) is accepted, provided it is not in the tabu list. Otherwise, if the said move is in the tabu list, the best solution is checked against an *aspiration criterion* and if satisfied, the move is accepted. Thus, the aspiration criterion can override the tabu list restrictions. It is desirable in certain conditions to accept a move even if it is in the tabu list, because it may take the search into a new region due to the effect of intermediate moves. The behavior of tabu search heavily depends on the size of tabu list as well as on the chosen aspiration criterion. Different sizes of tabu list result in short-term, intermediate term, and long-term memory components that can be used for intensifying or diversifying the search. The aspiration criterion determines the extent to which the tabu list can restrict the possible moves. If a tabu move satisfies aspiration criterion, then the move is accepted and tabu restriction is overridden. The structure of TS is given in Figure 3. The detailed description of tabu search can be found in [6].

Simulated Evolution (SimE)

SimE is a general iterative heuristic proposed by Ralph Kling [81, 82, 83]. It falls in the category of algorithms which emphasize the behavioral link between parents and offspring, or between reproductive population, rather than the genetic link [84]. This scheme combines iterative improvement and constructive perturbation and saves itself from getting trapped in local minima by following a stochastic perturbation approach.

The algorithm for Simulated Evolution is given in Figure-4. As can be seen, simulated evolution iteratively operates a sequence of evaluation, se-

Algorithm *Tabu_Search*

Ω : Set of feasible solutions
S : Current solution
S* : Best solution
Cost: Objective function
N(S): Neighborhood of S ∈ Ω
V* : Sample of neighborhood solutions
T : Tabu list
AL : Aspirartion level

Begin

Start with an initial feasible solution S ∈ Ω
Initialize tabu list and aspiration level
For fixed number of iterations **Do**
    Generate neighbor solutions V* ⊂ N(S)
    Find best S* ∈ V*
    If move S to S* is not in T **Then**
        Accept move and update best solution
        Update T and AL
    Else
        If Cost(S*) < AL **Then**
            Accept move and update best solution
            Update T and AL
        End If
    End If
End For

End.

Figure 3: Outline of Tabu Search algorithm [6].

lection, and allocation steps till the stopping criteria is met. Unlike genetic algorithm and tabu search, simulated evolution algorithm works on only one solution(population). The algorithm proceeds with evaluating goodness of its members. The goodness value is then used to partition the members in two sets, Ps and Pr. Pr contains those members that have achieved a desired level of goodness. The rest are placed in Ps. The idea behind selection is to save the state of good members from further perturbations. Members that did not satisfy required goodness, i.e. members in set Ps, are then perturbed in the allocation step. The exact nature of perturbation is problem-specific and usually requires designer's ingenuity. The goal of allocation is to favor improvements over the previous generation.

Non-determinism in simulated evolution resides at selection stage. A member that has achieved desired goodness level still has a nonzero chance to be assigned to Ps set. It is this non-determinism that provides simulated evolution algorithm the capability of uphill climbing.

From the results reported in the literature, SimE algorithm is a sound and robust randomized search heuristic. It is guaranteed to converge to the optimal solution if given enough time [6].

```
ALGORITHM Simulated_Evolution(M, L);
/* M: Set of movable elements; */
/* L: Set of locations; */
/* B: Selection bias; */
/* Stopping criteria and selection bias can be automatically adjusted; */
INITIALIZATION;
Repeat
   EVALUATION:
      ForEach  m ∈ M Do g_m = O_m / C_m EndForEach;

   SELECTION:
      ForEach  m ∈ M Do
         If Selection(m,B) Then  P_s = P_s ∪ {m}
            Else  P_r = P_r ∪ {m}
         EndIf;
      EndForEach;
   Sort the elements of P_s;

   ALLOCATION:
      ForEach  m ∈ P_s Do Allocation(m) EndForEach;
Until Stopping-criteria are met;
Return (BestSolution);
End Simulated_Evolution.
```

Figure 4: Outline of Simulated Evolution Algorithm [6].

4 Problem and Cost Functions Formulation

In this project, a high level tool for finite state machine synthesis will be designed and implemented with the objective of optimizing power, area and testability.

4.1 Problem Statement

The finite state machine synthesis problem can be stated as follows: Given a state machine with a number of inputs, outputs and states, define a state encoding that promises maximum potential in minimizing the objective functions while working at a higher level.

4.2 Cost Function

Various cost functions for area, power and testability for FSMs have been described in detail in section-3. One of the goals of the proposed work is to define a cost function that best formulates the given problem at a higher level, with an efficient execution time.

5 Project Objectives

The proposed research work will aim to develop a CAD tool that has the potential to efficiently solve the multiobjective problem of FSM encoding for area, power and testability objectives. The work entails investigation and fine tuning of cost functions for area, power and testability of an FSM. The cost functions that best correlate with the respective parameter will then be utilized to solve the multiobjective problem using some aggregating function. The quality of the solution obtained based on single-objective optimization will be compared to those obtained based on multiple-objective optimization. The work will be utilizing iterative algorithms like Genetic Algorithm, Simulated Evolution and Tabu Search to explore the search-space for an efficient solution.

6 Tasks Outline

The details of the major tasks to be carried out during the project work are enumerated as follows:

Task 1: Study and evaluation of cost estimators for area. In particular, cost estimators proposed by Jedi, Mustang, and others will be studied and evaluated. New efficient area cost estimators will be investigated.

Task 2: Employ and integrate suitable testability measures with area costs.

Task 3: Investigate the incorporation of power dissipation in the cost function and model a multiobjective cost function for reducing area, reducing power dissipation, and improving testability.

Task 4: Engineer GA for solving the multiobjective optimization problem for state assignment. This will include design of suitable chromosomes, choice of operators, tuning of parameters, etc.

Task 5: Employ TS for the above problem. This will include choice of suitable moves and their attributes, aspiration criteria, choice of parameters, etc.

Task 6: Tailor SE to solve the above multiobjective optimization problem. This task will include design of suitable *goodness* functions for the assignment of a state to a code, and choice of appropriate operators (selection and allocation).

Task 7: Experimentation with benchmarks and comparison of the 3 algorithms with other heuristics proposed in the literature.

Task 8: Documentation of implemented software, documentation of findings, and Publication of results.

7 Schedule & Management Plan

The work schedule is shown in the table below. All investigators and graduate students will be involved in all the tasks of the project for the entire duration.

	(Months)					
	01-03	04-06	07-09	10-12	13-15	16-18
Task 01	—	—				
Task 02		—				
Task 03		—	—			
Task 04			—	—		
Task 05				—	—	
Task 06					—	—
Task 07		—		—		—
Task 08		—		—		—

The project team will consist of a principal investigator, a co-investigators, and one part-time graduate-student. Responsibilities will be divided among the three senior investigators on the basis of their previous experience and background.

Dr. Aiman El-Maleh holds a B.Sc. in Computer Engineering, with first honors, from King Fahd University of Petroleum & Minerals in 1989, a M.A.SC. in Electrical Engineering from University of Victoria, Canada, in 1991, and a Ph.D in Electrical Engineering, with dean's honor list, from McGill University, Canada, in 1995. Dr. El-Maleh is an Assistant Professor in the Computer Engineering Department at King Fahd University of Petroleum & Minerals since September 1998. He was a member of scientific staff with Mentor Graphics Corp., a leader in design automation, from 1995-1998. His research interests are in the areas of synthesis, testing, and verification of digital systems. In addition, Dr. El-Maleh has research interests in VLSI design, design automation, and computer arithmetic. He is the winner of the best paper award for the most outstanding contribution in the field of test for 1995 at the European Design & Test Conference. His paper presented at the 1995 Design Automation Conference was also nominated for best paper award. He holds one US patent. Dr. El-Maleh was a member of the program committee of the Design Automation and Test in Europe Conference (DATE'98).

Dr. Sadiq M. Sait has major interests in VLSI Design automation, and, in engineering and applications of computers. He has published several papers in the area of VLSI physical design automation. He has co-authored two books, which are directly related to the project: (a) *VLSI Physical Design Automation: Theory and Practice*, McGraw-Hill Book Co., Europe, December 1994. Also Co-published by **IEEE Press**, USA, January 1995 (Hard bound edition), and, (b) *Iterative Computer Algorithms with Applications in*

Engineering: Solving Combinatorial Optimization Problems. December 1999, IEEE Computer Society Press, California.

All investigators will take the responsibility of the overall management of the project. Students will be computer engineering/science graduates with good programming background and will work under the guidance of investigators. The investigators will hold meetings as often as necessary to coordinate their work and to make necessary decisions. Frequent meetings will be necessary, minimum once a week.

8 Utility Value of the Project and Deliverables

The utility value of this project is many-fold, namely:

1. One of the main objectives of this project is to train graduate students in VLSI synthesis, algorithmic research, and in non-deterministic algorithms and their applications.
2. A new lab, recently setup, will be enhanced to provide support for future research projects in high-level synthesis, and help in research work of faculty and graduate students.
3. It is expected that the findings of our investigation of non-deterministic search for near optimal solutions will help in addressing other similar NP-hard engineering problems.
4. The results of this work can be used by other investigators in academia and industry to enhance existing methods. The project will contribute to advancing state-of-art techniques.

9 Detailed Budget

Senior Investigators

The senior investigators will work for 18 months during the regular semesters for the entire duration of the project. They will receive payments as per the university regulations. A graduate student will assist in the implementation aspects of the research, and will be required in the preparation of setup, literature search, experimentation, etc. Their total compensation will be as follows:

Dr. Aiman El-Maleh (PI)	SR 1200/- * 18 =	SR 21,600
Dr. Sadiq M. Sait (CO-I)	SR 1000/- * 18 =	SR 18,000
Graduate Student	SR 600/- *18 =	SR 10,800
Total		SR 50,400/-

Equipment, Materials & Supplies, and Other Expenses

Facilities available at KFUPM will be used at no charge to the project. No additional equipment is requested for this project as PCs and peripherals obtained for the previous projects will be used. Expenses for simple peripherals (such as CD writers, flash memories, hard-disks, remote keyboard and mouse, wireless devices, etc.,), consumables such as floppies, tapes, zip drives, CDs, printer toner, etc., will amount to SR 1,000/-, purchase of stationary, literature and books, etc., will require SR 500/-. Other miscellaneous and other incidental expenses may amount to a maximum of SR 1,000/-

Total cost of consumables and supplies is estimated to be

SR 2,500/-

A secretary will work for the total duration of the project, SR 1,800/- for payment will be required. Individual items are as summarized below.

Manpower	SR	50,400/-
Consumables:		
Expenses for peripherals (such as CD writers, flash memories, hard-disks, remote keyboard and mouse, wireless devices, etc.,)		
Floppies, CDs, tapes, zip drives, printer toner, etc	SR	1,000/-
Books, other literature, Stationary, etc.,	SR	500/-
Secretary	SR	1,800/-
Miscellaneous and other incidental expenses	SR	1,000/-
Conference Attendance (2 Required Trips)	SR	20,000/-
Total	SR	74,700/-

The total cost[1] of the project is estimated to be SR 74,700/-

[1] SR 74,700/-= US $ 19,920/-

References

[1] N. H. E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design: A systems perspective, Second edition.* Addison-Wesley, 1993.

[2] Sadiq M. Sait and Habib Youssef. VLSI Physical Design Automation: Theory and Practice. *World Scientific Pubishers*, 2001.

[3] M. Pedram. Design technologies for low power VLSI. *In Encyclopedia of Computer Science and Technology, Marcel Dekker, Inc.*, pages 73–96, 1997.

[4] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman. *Digital Systems Testing And Testable Design.* IEEE Press, 1990.

[5] Z. Kohavi. *Switching and Finite Automata Theory, Second edition.* McGraw-Hill Book Co., 1978.

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

[7] R. E. Stearns and J. Hartmanis. On the state assignment problem for sequential machines II. *IRE Transaction EC-10*, 1961.

[8] H. Allen Curtis. Multiple reduction of variable dependency of sequential machines. *Journal of ACM*, 9(3):324–344, 1962.

[9] Bernhard Eschermann. State assignment for hardwired VLSI control units. *ACM Computing Survey*, 25(4):415–436, 1993.

[10] R. Amann and U. G. Baitinger. Optimal state chains and state codes in finite state machines. *IEEE Transaction Computer Aided Design*, CAD-8:153–170, 1989.

[11] Demicheli G., Brayton R. K., and Sangiovanni Vincenteli A. Optimal state assignment for Finite State Machines. *IEEE Transaction on Computer Aided Design*, CAD-4:3, 269–285, 1985.

[12] G. DeMicheli. Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros. *IEEE Transaction on Computer Aided Design*, CAD-5:597–616, 1986.

[13] J. L. Huertas and J. M. Quintana. A new method for the efficient state-assigmnent of PLA-based sequential machines. *In the Internatzonal Conference on Computer-Aided Digital Design*, pages 156–159, 1988.

[14] J. L. Huertas and J. M. Quintana. Efficiency of state assignment methods for PLA based sequential circuits. *IEE Proceedings E, Computer and Digital Techniques*, pages 247–253, 1989.

[15] T. Villa and A. Sangiovanni-Vincentelli. Nova: state assignment of finite state machines for optimal two-level logic implementations. *Proceedings of the 1989 26th ACM/IEEE conference on Design automation conference*, pages 327–332, 1989.

[16] P. Weiner and Smith E. J. On the number of distinct state assignments for synchronous sequential machines. *IEEE Transaction on Elec Cornput. EC-16*, pages 220–221, 1967.

[17] G. DeMicheli, Brayton R.K., and Sangiovanni-Vincentelli A. Optimal State Assignment for Finite State Machines. *IEEE Transaction on CAD*, CAD-4(3):269– 284, July 1985.

[18] G. DeMicheli. Symbolic Design of Combinational and Sequential Logic Circuits Implemented by Two-level Logic Macros. *IEEE Trans. on CAD*, CAD-5(4):597–616, October 1986.

[19] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Prentice Hall, Englewood Cliffs, 1966.

[20] G. Saucier, M. Crastes de Paulet, and P. Sicard. ASYL: A Rule-based System for Controller Synthesis. *IEEE Transaction on CAD/ICAS*, Vol. CAD-6, No. 6:1088–1097, November 1987.

[21] P. Ashar, S. Devadas, and A. Newton. *Sequential Logic Synthesis*. Kluwer Academic Publishers, Boston, MA, 1992.

[22] S. Devadas, H.T. Ma, A.R. Newton, and Sangiovanni-Vincentelli. MUSTANG: State Assignment of Finite State Machines for Optimal Multi-Level Logic Implememations. *ICCAD*, November 1987.

[23] B. Lin and A. R. Newton. Synthesis of multiple-level logic from symbolic high-level description languages. *IFIP International Conference on Very Large Scale Integration*, pages 187–196, August 1989.

[24] X. Du, G. Hachtel, B. Lin, and A. R. Newton. MUSE: A Multilevel symbolic encoding algorithm for state assignment. *IEEE Trans. Computer Aided Design*, CAD-10:1.28–38, 1991.

[25] A. Almaini, J. Miller, P. Thomson, and S. Billina. State Assignment of state machine using Genetic Algorithm. *IEEE Proc. on Computer and Digital Techniques*, 142:279 – 286, 1995.

[26] J. Amaral, K. Turner, and J. Ghosh. Designing Genetic Algorithm for State Assignment Problem. *IEEE Trans on SMC*, 25:659 – 694, 1995.

[27] D.B Armstrong. A programmed algorithm for assigning internal codes to sequential machines. *IRE Transactions on Electronic Computers*, pages 466 – 472, 1962.

[28] Massoud Pedram. Power minimization in IC design: Principles and applications. *ACM Transaction on Design Automation of Electronic Systems*, 1(1):3–56, 1996.

[29] F.N. Najm. A survey of power estimation techniques in VLSI circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2:446 – 455, 1994.

[30] F.N. Najm. Power estimation techniques for integrated circuits. *IEEE/ACM International Conference on Computer-Aided Design*, 2:492 – 499, 1995.

[31] Chi-Ying Tsui, J. Monteiro, Massoud Pedram, Srinivas Devadas, A.M. Despain, and B. Lin. Power estimation methods for sequential logic circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3:404 – 416, 1995.

[32] Devadas S., Malik S., Keutzer K., and White J. Estimation of Average Switching Activity in Combinational and Sequential Circuits. *29th DAC*, pages 253–259, 1992.

[33] Devadas S. and Omnteiro J. Techniques for the Power Estimation of Sequential Logic Circuits Under User-Specified Input Sequences and Programs. *Int. Symp. on Low Power Design*, 1995.

[34] Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Probabilistic Analysis of Large Finite State Machines. *proceedings of DAC*, pages 270–275, 1994.

[35] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, University of California, Berkeley, 1992.

[36] Farid N. Najm, Shashank Goel, and Ibrahim N. Hajj. Power estimation in sequential circuits. In *Proceedings of the 32nd ACM/IEEE conference on Design automation conference*, pages 635–640. ACM Press, 1995.

[37] Joseph N. Kozhaya and Farid N. Najm. Accurate power estimation for large sequential circuits. In *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, pages 488–493. IEEE Computer Society, 1997.

[38] V. Saxena, F. N. Najm, and I. N. Hajj. Monte-Carlo approach for power estimation in sequential circuits. *European Design and Test Conference*, pages 416 – 420, 1997.

[39] Mahadevamury Nemani and Farid N. Najm. Towards a High-Level Power Estimation Capability. *IEEE Transactions on Computer Aided Design of Integrated Cicuits and Systems*, pages 588 – 598, 1996.

[40] M. Nemani and F.N. Najm. High-level area and power estimation for VLSI circuits. *IEEE Transactions on Computer Aided Design of Integrated Cicuits and Systems*, 18:697 – 713, 1999.

[41] Gary D. Hatchel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Markovian Analysis of Large Finite State Machines. *IEEE Transactions on CAD*, Vol. 15, Num. 12:1479–1493, Dec. 1996.

[42] A. W. Drake. *Fundamentals of Applied Probability Theory.* McGraw-Hill, 1967.

[43] A. Papoulis. *Random Variables and Stochastic Processes.* McGraw-Hill, 1984.

[44] Chi-Ying Tsui, Massoud Pedram, and Alvin M. Despain. Exact and approximate methods for calculating signal and transition probabilities in FSMs. In *Proceedings of the 31st annual conference on Design automation conference*, pages 18–23. ACM Press, 1994.

[45] Jose Monteiro, Srinivas Devadas, and Bill Lin. A methodology for efficient estimation of switching activity in sequential logic circuits. In *Proceedings of the 31st annual conference on Design automation conference*, pages 12–17. ACM Press, 1994.

[46] K. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications.* Prentice-Hall, 1982.

[47] Tzong-Dar Her, Wei Kang Tsai, F. Kurdahi, and Yulin Chen. Low-power driven state assignment of finite state machines. *IEEE Asia-Pacific Conference on Circuits and Systems*, pages 454 –459, 1994.

[48] S. K. Hong, I. C. Park, S. H. Hwang, and C. M. Kyung. State assignment in finite state machines for minimal switching power consumption. *IEEE Electronics Letters*, 30 Issue: 8:627 –629, 1994.

[49] S. J. Wang and M. D. Horng. State assignment of finite state machines for low power applications. *IEEE Electronics Letters*, 32 Issue: 25:2323 –2324, 1996.

[50] E. Olson and S.M. Kang. State assignment for low-power FSM synthesis using genetic local search. *IEEE Custom Integrated Circuits Conference*, pages 140 –143, 1994.

[51] V. Vamshi, T. Akhilesh, and R. Suresh. Re-encoding for Low Power State Assignment of FSMs. *ACM International Symposium on Low Power Design*, 1995.

[52] K. Roy and S. Prasad. SYCLOP: synthesis of CMOS logic for low power applications . *ICCD*, pages 464 –467, 1992.

[53] I. Lemberski, M. Koegst, S. Cotofana, and B. Juurlink. FSM non-minimal state encoding for low power. *23rd International Conference on Microelectronics*, pages 605 –608, 2002.

[54] M. Koegst, G. Franke, and K. Feske. State assignment for FSM low power design. *EURO-DAC*, pages 28 –33, 1996.

[55] Chunhong Chen, Jiang Zhao, and Majid Ahmadi. A semi-Gray encoding algorithm for low-power state assignment. *International Symposium on Circuits and Systems, ISCAS*, 5:V–389 –V–392, 2003.

[56] P. Surti, L.F. Chao, and A. Tyagi. Low power FSM design using Huffman-style encoding. *European Design and Test Conference, ED&TC*, pages 521 –525, 1997.

[57] P. Bacchetta, L. Daldoss, D. Sciuto, and C. Silvano. Low-power state assignment techniques for finite state machines. *International Symposium on Circuits and Systems*, 2:641 –644, 2000.

[58] L. Daldoss, D. Sciuto, and C. Silvano. State encoding for low power embedded controllers. *International Symposium on Circuits and Systems, ISCAS*, 2:421 –424, 1998.

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

[60] T. Dolotta and E. McCluskey. The coding of internal states of sequential machines. *IEEE Trans. Electron. Ecomput.*, EC-13:549–562, 1964.

[61] Chi-Ymg Tsui, M. Pedram, Chih-Ang Chen, and A. M. Despain. Low Power State Assignment Targeting Two And Multi-level Logic Implementations. *IEEE/ACM International Conference on Computer-Aided Design*, pages 82–87, 1994.

[62] I. Bhupathi and L. F. Chao. High-level area and power estimation for VLSI circuits. *ISCAS*, 4:759 – 762, 1996.

[63] Y. Xia and A. E. A. Almaini. Genetic algorithm based state assignment for power and area optimisation. *IEEE Proceedings Computers and Digital Techniques*, 149:128 – 133, 2002.

[64] G. Venkataraman, S. M. Reddy, and I. Pomeranz. GALLOP: genetic algorithm based low power FSM synthesis by simultaneous partitioning and state assignment. *IEEE 16th International Conference on VLSI Design*, pages 533 – 538, 2003.

[65] O. H. Ibarra and S. Sahni. Polynomially Complete Fault Detection Problems. *IEEE Transaction on Computers*, C-24, No. 3:242–249, 1975.

[66] J. P. Roth. Diagnosis of Automata Failure: A Calculus and a method. *IBM Journal of Research and Development*, 10, No. 4:278–291, 1966.

[67] J. P. Roth, W. G. Bouricius, and P. R. Schneider. Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Cicuits. *IEEE Transactions on Electronic Computers*, EC-16, No. 10:567–580, 1967.

[68] P. Goel and B. C. Rosales. PODEM-X: an automatic test generation system for VLSI logic structures. *Proceedings of the Design Automation Conference, IEEE Computer Society Press*, pages 260–268, 1981.

[69] H. Faujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, C-32, No. 12:1137–1144, 1983.

[70] T. Trischler. Incomplete scan pat with an automatic test gernaeration methodology. *Proceedings of the International Test Conference*, pages 153 – 162, 1980.

[71] A. Miczo. *Digital Logic Testing and Simulation*. Harper & Row Publishers, 1986.

[72] A. Lioy, P. L. Montessoro, and S. Gai. A complexity analysis of sequential ATPG. *IEEE International Symposium on Circuits and Systems*, pages 1946 – 1949, 1989.

[73] T. E. Marchok, A. El-Maleh, W. Maly, and J. Rajski. A complexity analysis of sequential ATPG. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15:1409 – 1423, 1996.

[74] Thomas Edward Marchok. *Modelling the difficulty of Automatic Test Pattern Generation for Sequential Circuits*. PhD thesis, Carnegie-Mellon University, 1995.

[75] I. Pomeranz and K. T. Cheng. State assignment using input/output functions. *29th ACM/IEEE Design Automation Conference, 1992. Proceedings.*, pages 573 – 577, 1992.

[76] K. T. Cheng and V. D. Agrawal. Design of sequential machines for efficient test generation. *ICCAD*, pages 358 – 361, 1989.

[77] C. R. Mohan and P. P. Chakrabarti. EARTH: combined state assignment of pla-based fsm's targeting area and testability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 727 – 731, 1996.

[78] S. Chiusano, F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda. Guaranteeing testability in re-encoding for low power. *Sixth Asian Test Symposium*, pages 30 – 35, 1997.

[79] K. Shahookar and P. Mazumder. VLSI Cell Placement Techniques. *ACM Computing Surveys*, 2(23):143–220, June 1991.

[80] J. P. Cohoon and W. D. Paris. Genetic placement. *IEEE Transactions on Computer Aided Design*, pages 956–964, 1987.

[81] R. M. Kling. *Optimization by Simulated Evolution and its Application to Cell Placement.* PhD thesis, University of Illinois, Urbana, 1990.

[82] R. M. Kling and P. Banerjee. ESP: Placement by Simulated Evolution. *IEEE Transaction on Computer-Aided Design*, 3(8):245–255, March 1989.

[83] R. M. Kling and P. Banerjee. Empirical and Theoretical Studies of the Simulated Evolution Method Applied to Standard Cell Placement. *IEEE Transaction on Computer-Aided Design*, 10(10):1303–1315, October 1991.

[84] D. B. Fogel. An Introduction to Simulated Evolutionary Optimization. *IEEE Transaction on Neural Networks*, 5(1):3–14, January 1994.