Applied Soft Computing 13 (2013) 4832–4840

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Binary particle swarm optimization (BPSO) based state assignment for area minimization of sequential circuits

Aiman H. El-Maleh [a,*], Ahmad T. Sheikh [b], Sadiq M. Sait [c]

[a] Computer Engineering Department, KFUPM, Dhahran, Saudi Arabia
[b] College of Computer Science & Engineering, KFUPM, Dhahran, Saudi Arabia
[c] Department of Computer Engineering and Center for Communications and IT Research, Research Institute, KFUPM, Dhahran, Saudi Arabia

ARTICLE INFO

Article history:
Received 4 May 2012
Received in revised form 4 June 2013
Accepted 5 August 2013
Available online 12 September 2013

Keywords:
State assignment (SA)
Area minimization
Non-determinism
Heuristics
PSO
Binary PSO

ABSTRACT

State assignment (SA) for finite state machines (FSMs) is one of the main optimization problems in the synthesis of sequential circuits. It determines the complexity of its combinational circuit and thus area, delay, testability and power dissipation of its implementation. Particle swarm optimization (PSO) is a non-deterministic heuristic that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. PSO optimizes a problem by having a population of candidate solutions called particles, and moving them around in the search-space according to a simple mathematical formulae. In this paper, we propose an improved binary particle swarm optimization (BPSO) algorithm and demonstrate its effectiveness in solving the state assignment problem in sequential circuit synthesis targeting area optimization. It will be an evident that the proposed BPSO algorithm overcomes the drawbacks of the original BPSO algorithm. Experimental results demonstrate the effectiveness of the proposed BPSO algorithm in comparison to other BPSO variants reported in the literature and in comparison to Genetic Algorithm (GA), Simulated Evolution (SimE) and deterministic algorithms like Jedi and Nova.

1. Introduction

We are living in an era of smart phones and smart devices that have the computing power of desktop PCs of few years back, yet, they can be occupied in our palms. This shrinking of area is still continuing, posing challenges to the design of efficient synthesis tools. Digital systems are comprised of a data path unit and a control unit. The data path unit is concerned with arithmetic and logic operations and the movement/storage of data among registers in the digital system. The control unit is the brain of any digital system and it controls what operations should occur at which point in the life cycle of a digital system. Control units are designed based on Finite state machines (FSMs). Subsequent to the design of a FSM from an given specification, each state in the FSM is binary encoded. Encoding plays a vital role in the design of digital systems. Different assignment of state codes will result in different combinational logic implementations and different cost values. It has been observed that minute changes in state codes have a significant effect on area, power and testability of a circuit. Automated design of FSMs to optimize a given cost such as area, power, performance and delay, etc. has been of considerable interest to the Computer Aided Design (CAD) community. This has not only posed new challenges but also opened new avenues of research for Very Large Scale Integrated (VLSI) CAD industry.

The state assignment (SA) of an FSM is a problem that maps $f: S \rightarrow B^n$, where n is the code length, $n \geq \lceil \log_2 |S| \rceil$, B^n is an n-dimensional Boolean hypercube and $|S|$ is the number of states. To encode S states using k bits, the number of possible state assignment combinations is given by Eq. (1).

$$\frac{(2^k)!}{(2^k - |S|)!} \tag{1}$$

For example, if we have an FSM with 10 states, then each state will require 4 bits for unique encoding. The number of possible encodings will be 29,059,430,400. Exhaustively assigning each combination of code and looking for the one that will optimize the given objective will require large computational time. The SA of FSM is an NP (nondeterministic-polynomial time)-hard problem [15], which implies exponential complexity in the worst case.

In area optimization of two-level circuits, the objective is to reduce the number of Sum of Product (SOP) terms, while in multi-level circuit optimization the objective is to reduce the number of literals (i.e. variables in true or complement form) in the expressions of the synthesized circuit. Several optimization techniques have been proposed to solve the SA problem. Previous research on two-level combinational realization of FSMs using deterministic

* Corresponding author. Tel.: +966 540950326.
E-mail address: aimane@kfupm.edu.sa (A.H. El-Maleh).

1568-4946/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2013.08.004

heuristics for area minimization employed mechanisms such as implicit merging, code covering and disjunctive coding [9]. Finding a state assignment which resulted in common expressions and maximum literal savings was one of the objectives in state assignment problem. Devadas et al. proposed two algorithms [3,8], the first one is fan-out oriented and assigns close codes, from the perspective of minimum hamming distance, to the state pairs that have similar next state transitions. The second algorithm, that 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 the state pairs that are assigned close codes in order to maximize the frequency of common cubes in the encoded next-state functions. Deterministic algorithms for area minimization of two-level circuits are also proposed in Jedi [2] and Nova [5]. SIS: Sequential Interactive Synthesis [28] is an interactive tool for the synthesis and optimization of sequential circuits. SIS can synthesize combinational, synchronous and asynchronous circuits generating optimized two-level (Sum of Products form) or multi-level (factorized) equations, which can then further be mapped onto a user-defined component library representing gates, flip-flops, standard cells, etc. In a nutshell, SIS can be used as a framework to test and compare different algorithms, as a cost evaluator and also as a tool for automatic synthesis and optimization of sequential circuits. The focus in this paper is to optimize the area of multi-level circuits, and for that matter SIS tool has been utilized to compute the area requirements.

Additional efforts have been made to to solve this problem deterministically using various objective functions. Sagahyroon et al. formulated the SA problem as an integer linear programming (ILP) problem for power minimization of FSMs [27]. Another approach for low power design is proposed by Salauyou et al. [23] which is based on sequentially assigning states with highest tranition probability state codes with Hamming distance of 1. In [4,5,7] multi-level area minimization has been modeled as constrained graph-embedding problem on Boolean hypercubes.

Due to the complexity of SA problem, and the limitations of existing deterministic algorithms, there has been a flurry of interest in the use of non-deterministic heuristic algorithms like Genetic Algorithm (GA) [10,12,16,18,20], Simulated Annealing [2,26], Tabu Search (TS) [19], Simulation Evolution (SimE) algorithm [30] and using other Evolutionary Algorithms [17,21]. The GA has been used to optimize power consumption by dividing a state machine into sub FSMs, and then in each partition, the states have code with minimum hamming distance [24]. Later on, power consumption is reduced by selectively activating the FSM partitions. In another approach, Chaudhury et al. [25] proposed GA-based methodology to minimize area, leakage (static) power as well as dynamic power during synthesis of finite state machines. A multi-objective GA is proposed [29] to minimize the power and area requirements of completely and incompletely specified sequential circuits. Amaral et al. [12] used GA with cost function proposed by Armstrong [1]. Armstrong 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) as a tool for applying heuristic rules on FSM [6]. The obvious benefit of these algorithms is that they search for an optimal solution in a large search space and tend to produce promising solutions for hard combinatorial optimization problems given the tuning parameters are chosen carefully. The element of randomness in these algorithms tries to avoid local minima.

In this work, the use of particle swarm optimization, a non-deterministic evolutionary algorithm based on the movements of birds, to solve SA for FSM problem is explored. An improved binary particle swarm optimization is proposed and SA of FSM is taken as a test case to compare the results of the proposed method with other variants of PSO, evolutionary and deterministic algorithms.

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
S1	S3	S3	0	0
S2	S3	S1	0	1
S3	S5	S4	0	1
S4	S2	S3	1	0
S5	S2	S5	1	0

Fig. 1. A finite state machine example.

The comparison criterion is the literal count in synthesized multi-level circuits. The rest of the paper is organized as follows. Section 2 discusses a simple example to illustrate the concept and importance of state assignment problem. In Section 3, we give a brief background of PSO and its evolution. In Section 4, we discuss our proposed binary PSO (BPSO) algorithm. An illustrative example is discussed in Section 5 to give a gist of particles evolution with time/iterations. Experimental setup and simulation results are discussed in Section 6, while conclusions are given in Section 7.

2. State assignment problem

To understand the state assignment problem, we now discuss a simple example. Fig. 1 describes the states of an FSM, their transition to the next state and the output produced during transition from one state to the other depending on the input value. To understand the table in Fig. 1, consider a case when *Present State* = *S*2. If input $X = 0$ then *Next State* = *S*3 and *Output* = 0 but if $X = 1$ then *Next State* = *S*1 and *Output* = 1.

Since there are 5 states, a 3-bit code is required for encoding each state. Table 1 shows two different state assignment codes labeled as "Code 1" and "Code 2". The resulting area cost i.e., the number of literals computed using SIS [28], is 22 for "Code 1". However, "Code 2" has significantly smaller area, resulting in 12 literals as compared to "Code 1". The number of literals is a cost measure that correlates with the number of transistors in the circuit. For example, the equation $y = ab + \bar{a}c$ has 4 literals $(a, \bar{a}, b, c)$.

This example demonstrates the significant impact of state assignment on the area of a synthesized sequential circuit.

3. PSO background

Particle swarm optimization proposed by Kennedy and Eberhart [11], is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. PSO optimizes a problem by having a population of candidate solutions called particles, and moving these particles around in the search-space according to simple mathematical formulae. The movements of the particles are guided by their own best known position (solution) in the search-space as well as the swarm's entire best known position. As improved positions are being discovered, they will be employed to guide the movements of the swarm.

Table 1
State assignments with resulting area cost.

State	Code 1	Code 2
S1	101	111
S2	011	101
S3	001	001
S4	010	100
S5	111	000
Area (no. of literals)	22	12

Initially, PSO algorithm was developed for problems that were continuous in nature. Later on, due to the discrete intrinsic nature of most of the real life problems, the original authors proposed discrete version of PSO [14]. In the following, we will take a brief look into both the continuous and discrete versions of PSO.

3.1. Continuous particle swarm optimization

The continuous PSO consists of the following two equations:

$$V_i(t+1) = \omega_t \times V_i(t) + c_1 r_1 (pBest_i - X_i(t)) + c_2 r_2 (gBest - X_i(t)) \quad (2)$$

$$X_i(t+1) = x_i(t) + V_i(t+1) \quad (3)$$

In the above equations, $V_i(t+1)$ and $X_i(t+1)$ are the new velocity and the new position of particle i at time $(t+1)$, respectively; c_1 and c_2 are arbitrary constants, and r_1 and r_2 are random variables $\in(0, 1)$ of uniform distribution. $pBest_i$ is the best solution of particle i until time t while $gBest$ is the overall global best solution among all the particles until time t. The $pBest_i$ of a particle is the exploitation factor, while $gBest$ helps in the exploration of the solution space. This way particles try not only to come closer to their own best solution, but also to the global best solution with a certain probability and weight factor. ω is a linearly decreasing inertia factor that gives less weight to the previous velocity over the period of iterations. An upper bound on velocity V_{max} is normally placed to make sure that particles remain in a finite search space. In continuous PSO, large value of V_{max} tends to perform more exploration, while in binary PSO small value of V_{max} is preferred. The next position of a particle is computed by the addition of newly computed velocity and the previous position.

3.2. Binary particle swarm optimization

Proposed by Kennedy and Eberhart [14], the binary version of Particle Swarm Optimization implements the decision making of a particle based on discrete decision i.e., "true" or "false". In contrary to the continuous PSO, this algorithm represents each state of a particle in the form of discrete binary "0" and "1" numbers. The velocities in binary version are defined as probabilities with which a certain bit of particle's solution will change to "0" or to "1". This way, velocity is clamped between the range [0,1]. Sigmoid function is used to map the continuous valued velocity given by Eq. (4) to the range [0,1] [14], as shown in Eq. (5).

$$V_{ij}(t+1) = \omega_t \times V_{ij}(t) + c_1 r_1 (pBest_{ij} - X_{ij}(t)) + c_2 r_2 (gBest_j - X_{ij}(t)) \quad (4)$$

$$V'_{ij}(t+1) = sig(V_{ij}(t+1)) = \frac{1}{1 + e^{-V_{ij}(t+1)}} \quad (5)$$

The subscript i, j refers to particle number i and the particular bit j of that particle's velocity, respectively. The equation used to update position of a particle, originally proposed, is given below.

$$X_{ij}(t+1) = \begin{cases} 1 & \text{if } r_{ij} < sig(V_{ij}(t+1)) \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

where r_{ij} is a uniformly distributed random number in the range (0,1).

To understand the functionality of this approach, consider a case when velocity is limited in a range from $[-4, 4]$, with $V_{max} = 4$ and $V_{min} = -4$. The velocity component defines the probability with which a particle's bit should change its bit value. For example, if $V_{ij} = 4$, then using Eq. (5) will imply $sig(V_{ij}) = 0.98$, which is the probability that X_{ij} will have bit value "1". Similarly, $V_{ij} = -4$ implies $sig(V_{ij}) = 0.018$, which is the probability that X_{ij} will have bit value "1". Finally, for $V_{ij} = 0$ we have $sig(V_{ij}) = 0.5$, in this case the search turns into pure random search and X_{ij} can be changed to either "1" or "0" with equal probability.

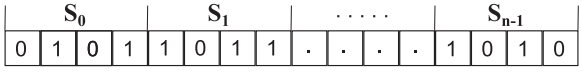


Fig. 2. An example of a particle.

Now, consider the case when $pBest_{ij} = gBest_j = 1$ and $X_{ij} = 0$ at time t, then Eq. (4) will increase the velocity of the jth bit of particle i, i.e., the probability of $X_{ij}(t) = 1$ is increased. For a second case where $pBest_{ij} = gBest_j = 0$ and $X_{ij} = 1$ at time t, then Eq. (4) will decrease the velocity of the jth bit of particle i, i.e., the probability of $X_{ij}(t) = 0$ is increased. Both of these cases are working well as expected i.e., when a certain bit in a particle is different from both the local best and the global best solutions, the velocity should either increase or decrease to maximize the probability of getting a bit similar to that of the global and local best solutions. This not only highlights the importance of velocity in PSO, but also quantifies the importance of constraining the velocity to a certain range.

The limitations to the original algorithm occurs when either the current bit is the same as the local and global best solutions bits or when the global and local best solutions bits are different. When $X_{ij} = pBest_{ij} = gBest_j$ at time t, in that case the velocity becomes a mere function of its previous velocity component, whereas, the velocity component should have been either decreased or increased depending on whether $pBest_{ij} = gBest_j = 0$ or $pBest_{ij} = gBest_j = 1$, respectively. The other limitation is when $pBest_{ij} \neq gBest_j$. In that case, if $X_{ij} = pBest_{ij}$ or $X_{ij} = gBest_j$ then the particle will end up following either its personal best or global best solution without improving the overall solution quality.

In this paper, we propose an improved BPSO algorithm that will rectify the drawbacks of the original BPSO, discussed above. We then compare our approach to another modification proposed by Khanesar et al. [22] that uses the concept of "rate of change of velocity".

4. Proposed BPSO algorithm

We propose a formulation that will utilize binary PSO (BPSO) algorithm to solve the SA problem. In our formulation, each particle will hold a complete solution i.e., each particle possesses codes for all states of a circuit. Fig. 2 shows the contents of a particle, where for the purpose of illustration, a $4-bit$ code is assigned to each state. The size of a particle is the number of states in a circuit times the number of bits required to encode each state.

In our proposed BPSO algorithm, we have modified the original algorithm to overcome its drawbacks. The following equation computes the velocity of the proposed algorithm. The velocity equation is separated into three different cases, which are mentioned below.

$$V_{ij}(t+1) = \begin{cases} \omega_t \times V_{ij}(t) + c_1 r_1 + c_2 r_2 & \text{if } gB_j = pB_{ij} = 1 \\ \omega_t \times V_{ij}(t) - c_1 r_1 - c_2 r_2 & \text{if } gB_j = pB_{ij} = 0 \\ \omega_t \times V_{ij}(t) & \text{otherwise} \end{cases} \quad (7)$$

where gB_j and pB_{ij} are bits of global best and local best solutions of a particle. The position of a particle is updated using Eq. (6), whereas, ω_t is a linearly decreasing inertia factor calculated using Eq. (8). $\omega_{t=max}$ and $\omega_{t=min}$ define the maximum and minimum value of ω respectively, with t being the iteration number.

$$\omega_t = \omega_{t=max} - \frac{\omega_{t=max} - \omega_{t=min}}{t = max} \times t \quad (8)$$

As can be observed, Eq. (7) presents three different scenarios of updating the velocity component. These will rectify the problems of the original BPSO algorithm. For the cases when $gBest_j = pBest_{ij} = 1$

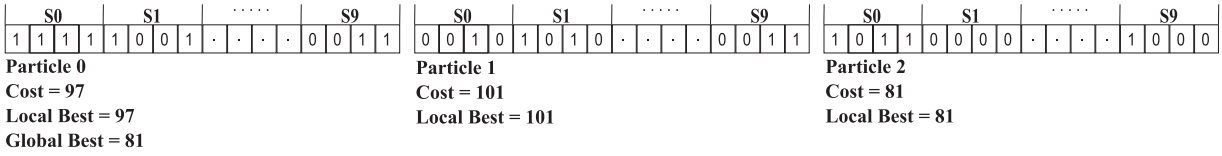


Fig. 3. Initial state of particles.

or $gBest_j = pBest_{ij} = 0$, the velocity component should be increased or decreased respectively, as both the global and local best solutions have a consensus. For the case when $gBest_j \neq pBest_{ij}$, the velocity remains unchanged and depends only on the previous velocity. This is justified as there is no agreement between the global best and local best solutions and hence none of them is favored.

Algorithm 1 shows the pseudo-code of BPSO. It starts with initialization of particles and then successively updating velocity, position, local best and global best solution with each iteration. We did not specifically employ any local minima avoidance scheme and rely on the intrinsic random behavior of BPSO to avoid any such situation.

Algorithm 1. Binary PSO (BPSO)

Require: No. of Particles, MaxIter, InputFile
1: Read Input File
2: Initialize Particles
3: **while** ($iter \leq MaxIter$) **do**
4: **for all** (Particles i) **do**
5: **Update** Velocity V_i of Particle i
6: **Update** Position X_i of Particle i
7: **Validate** X_i for duplicate codes
8: **if** ($currentSolution_i < pBest_i$) **then**
9: $pBest_i = currentSolution_i$
10: **end if**
11: **end for**
12: $gBest_{iter} = \min_{\forall i}(pBest_i, gBest_{iter-1})$
13: $iter$++
14: **end while**

The algorithm takes as an input three arguments, *No. of Particles, MaxIter and InputFile*. Particles velocity and positions are then randomly initialized. Newly initialized positions are a set of codes that can be assigned to the states of a circuit under optimization. Before computing cost, state codes must be validated. In validation we check for duplicate codes, which are highly possible due to random initialization, and if any duplicate code is found then it is replaced with an unused code. Minimum Hamming Distance criterion is used to pick a code from a set of unused codes. For example, if we have 10 states in a circuit then we require 4 bits at minimum to encode them. So in this case we have 10 codes that are assigned to states and the remaining 6 codes unassigned. The personal best of each particle is the initial cost computed by random initialization. The global best is the minimum cost among personal bests.

The velocity and positions are then updated using Eqs. (5), (6) and (7). Post processing is performed every time a position changes in order to replace any duplicate codes.

5. Illustrative example

We now discuss a simple example to illustrate the functionality of the proposed algorithm. The proposed algorithm is used to minimize the area of a small benchmark circuit. Our example consists of a case where *number of particles* = 3, *Max. iterations* = 20, $c_1 = c_2 = 1.5$. The considered *benchmark* is *bbara . kiss2* which has 10 states; therefore 4 bits are required to encode each state. The abbreviations used in the example are shown below along with their definitions.

$CV_{i,j}$: current velocity of jth bit of a particle i computed using Eq. (7).
$PV_{i,j}$: Previous Velocity of jth bit of a particle i.
$X_{i,j}$: jth bit of current solution of a particle i.
$pB_{i,j}$: jth bit of personal best of a particle i.
gB_j: jth bit of global best among all particles.
Expr: $sgn(CV_{ij})$.

The initial state of particles is shown in Fig. 3. Since, there are 10 states and 4 bits are required to encode each state, therefore, the size of the particle is 40 bits. The current velocity $CV_{i,j}$ and previous velocity $PV_{i,j}$ vectors also have size 40 and are initialized to "0". As evident from Fig. 3, each item in a particle has a binary value of either "0" or "1" and four items in sequence collectively form the code of a state.

After random initialization, *Particle* 2 holds the global best solution and the personal/local best solution of each particle is its initial solution. We will now go through few iterations to observe the evolution of particles more closely.

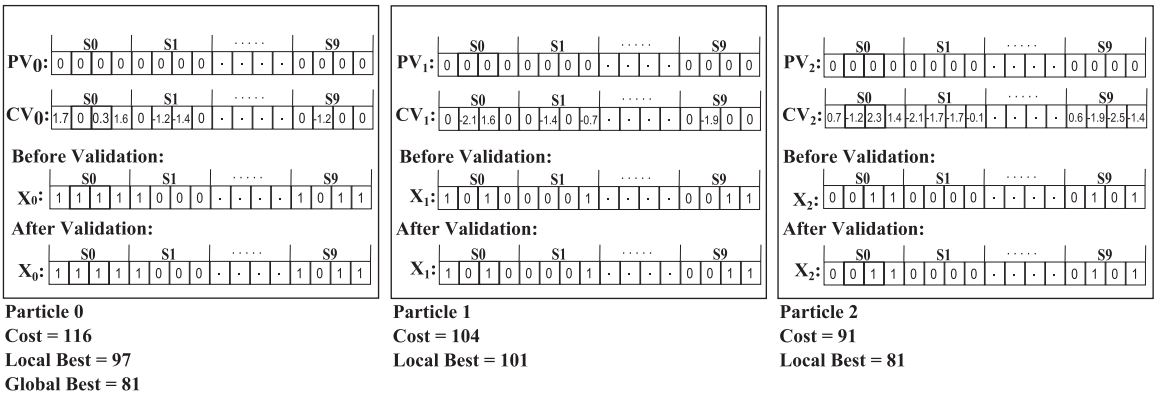


Fig. 4. After iteration 1, for $\omega = 0.82$.

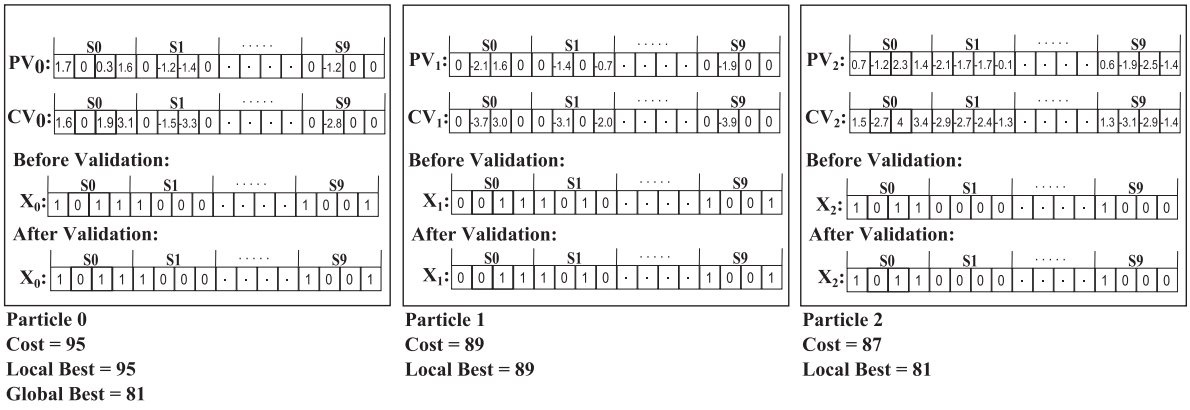


Fig. 5. After iteration 2, for $\omega = 0.74$.

The state of particles after first iteration are shown in Fig. 4. To elaborate the computation of current velocity vector, consider encoding of *state*0(*S*0) of *Particle* 1. From Fig. 3 we can observe that $X_{1,3} = 1$, $PB_{1,3} = 1$, $GB_3 = 1$, $PV_{1,3} = 0$. The ω has a range between $0.9 \rightarrow 0.1$. Therefore, using Eq. (8), $\omega = 0.82$ in first iteration. In the first step of a particle's evolution we compute the velocity, and in the proposed algorithm each bit in a particle's solution has an individual velocity component. In order to compute $CV_{1,3}$ i.e., current velocity of bit at index 3 of particle 1, Eq. (7) will be utilized. Here, we have a condition $GB_3 = PB_{1,3} = 1$, therefore,

$$\begin{aligned} V_{1,3}(1) &= \omega_t \times V_{1,3}(0) + c_1 r_1 + c_2 r_2 \\ &= c_1 r_1 + c_2 r_2 \\ &= 1.69 \quad \text{for } r_1 = 0.91, r_2 = 0.86 \end{aligned}$$

Then, $sig(V_{1,3}(1.69)) = 0.85$. And finally, r_{13} a random number, is generated and gets a value of 0.64. Since $r_{13} < 0.85$, therefore $X_{1,3} = 1$, otherwise it would be $X_{1,3} = 0$. This way we compute the current velocity of each bit of a particle's current position and in the next iteration the current velocity of particle becomes its previous velocity. One important thing to discuss here is that after we change the current position of a particle, more than one states might end up with having the same code. To solve this, a validation check is performed each time a particle changes it position. In validation check, we seek for the duplicate codes and replace them with the unused codes with which they have minimum hamming distance. In current example, 4 bit codes are used, so the set of codes that can be assigned are 16, ranging from $0000 \rightarrow 1111$. Since, there are 10 states, a set of 6 unassigned codes is available to replace the duplicate codes based on minimum hamming distance criterion. After first iteration there is no change in the overall best solution, therefore the global best solution retains its initial solution.

Figs. 5 and 6 show the second and third iterations. After second iteration there is no change in the global best solution but the local best solutions of *Particle* 0 and *Particle* 1 are improved. After third iteration, the local best solutions of *Particle* 0 and *Particle* 2 are improved, resulting in *Particle* 2 holding a new global best solution with *Cost* = 76. Continuing in this fashion, after 20 iterations *Particle* 1 holds the best solution with best *Cost* = 70.

To augment the discussion on the severity of state assignment problem in Section 2, Fig. 7 shows the evolution of two randomly chosen particles for the first 50 iterations. The spikes in the figure show a sudden change in the cost with a slight change in state assignments.

6. Experimental results

The code for the proposed BPSO algorithm was implemented and executed on Linux machine with Quad-core processors and 4 GB of RAM. MCNC/LGSynth [13] benchmark circuits are used to test the efficacy of the algorithm. These benchmark circuits have varying complexity. Table 2 shows the benchmark circuits along with their number of states, inputs and outputs, used in this work. For synthesis, *sis*1.3 package was used with *stg_to_network*-e2 (generates an optimized two-level logic network) and *fx* (fast extraction for multilevel circuit optimization) options to perform multilevel optimization in order to compute the area of a synthesized circuit with assigned set of state codes. Based on simulation results, the constants c_1 and c_2 were fixed at 1.5. The initial value of the inertia factor ω is also empirically derived and has a range of $0.9 \rightarrow 0.1$.

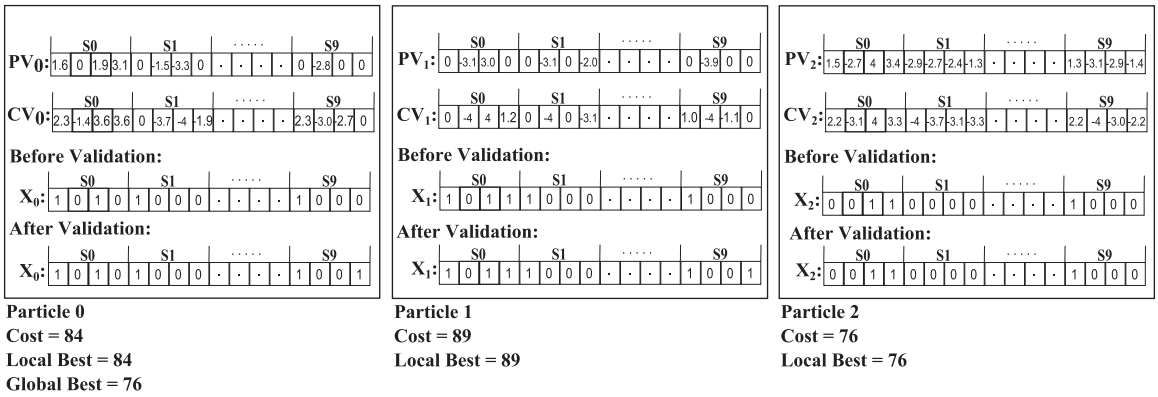
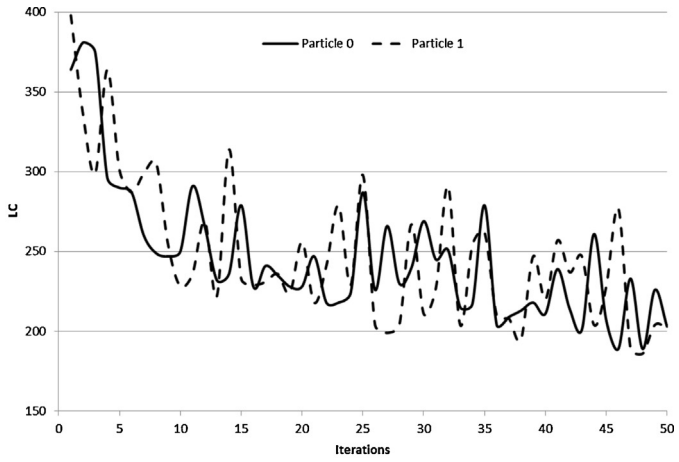
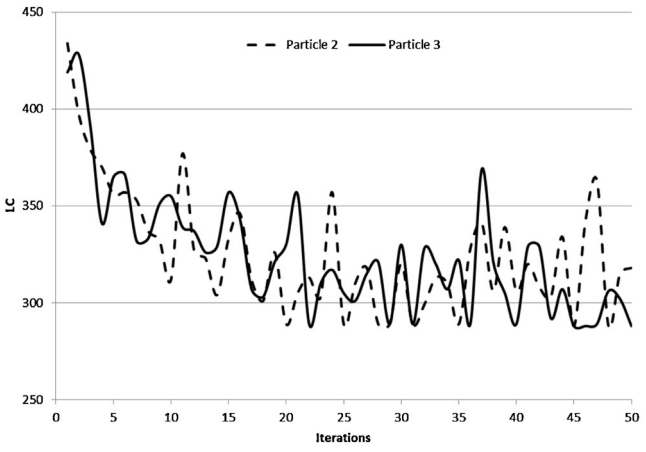


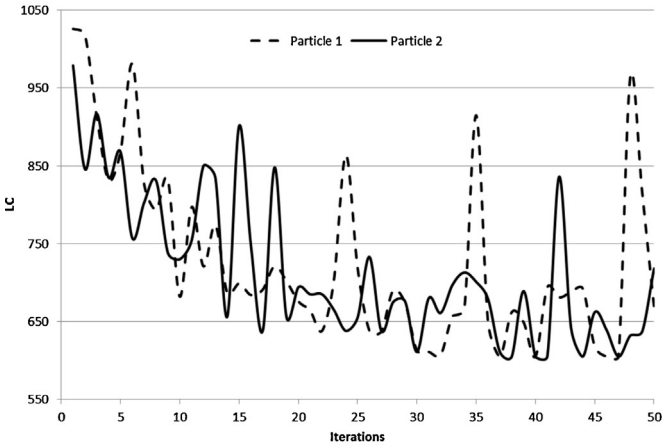
Fig. 6. After iteration 3, for $\omega = 0.66$.



(a) *keyb*



(b) *s832*



(c) *tbk*

Fig. 7. Particle's evolution: literal count (LC) versus iterations of two particles for few circuits.

Table 2
Benchmarks characteristics.

Circuits	States	Inputs	Outputs
bbara	10	4	2
bbsse	16	7	7
cse	16	7	7
dk14	7	3	5
ex2	19	2	2
ex3	10	2	2
keyb	19	7	2
lion9	9	2	1
planet	48	7	19
pma	24	8	8
s1	20	8	6
s1494	48	8	19
s832	25	18	19
sand	32	11	9
styr	30	9	10
tav	4	4	4
tbk	32	6	3
train11	11	2	1

Simulations were carried out for two different scenarios and are separately discussed in the following subsections.

6.1. Comparison with other BPSO techniques

Our main objective of this research is to compare the performance of proposed BPSO algorithm with the already existing BPSO algorithms. For that sake, the original BPSO algorithm given by Eqs. (2), (5) and (6) along with its enhanced version in [22] are also implemented for the State Assignment problem. For each algorithm, we assumed the same number of particles and the same number of maximum iterations. Each algorithm is executed for 350 iterations and the number of particles is taken to be 64. Due to randomness in these heuristic algorithms, each is executed 10 times in order to have a better picture regarding their performance. The results are reported in terms of *Best*, *Average* and *Worst* cost achieved over 10 runs. Table 3 shows the results achieved for different variants of BPSO algorithm.

The average time of execution for all algorithms is almost the same, so we can eliminate this factor from performance comparison. The evaluation criterion focus are the individual costs achieved for each circuit and the total cost, which is the sum of individual costs. As can be observed from Table 3, the total number of literals for each of the best, average and worst category achieved by the proposed *BPSO-proposed* algorithm are far better than the original *BPSO-VNORM* [14] and slightly better than *BPSO-VCHANGE* [22] algorithm. The proposed algorithm outperforms *BPSO-VNORM* for all benchmark circuits and in every performance evaluation category. In Comparison to *BPSO-VCHANGE*, the proposed *BPSO* outperforms it in all evaluation categories for 9 benchmarks which are *dk14*, *ex3*, *keyb*, *planet*, *s1*, *s832*, *sand*, *styr* and *tbk*. For the remaining benchmarks, the *BPSO-proposed* algorithm achieves same *Best* results for *bbara*, *cse*, *ex2*, *lion9* and *train11*, while better *Average* and *Worst* literal counts. For *bbsse*, *pma* and *s1494* benchmarks, *BPSO-VCHANGE* performs better in all performance categories as compared to the proposed BPSO algorithm. Fig. 8 shows plots of how the proposed and other binary PSO algorithms achieved their best solutions for few benchmark circuits.

6.2. Comparison with genetic algorithm, simulated evolution and other deterministic algorithms

Table 4 shows the comparison of the proposed BPSO algorithm with Genetic Algorithm (GA) ([20]) using the same parameters i.e. number of genes and number of generations as used for the BPSO, with the recently proposed Simulated Evolution (SimE) algorithm

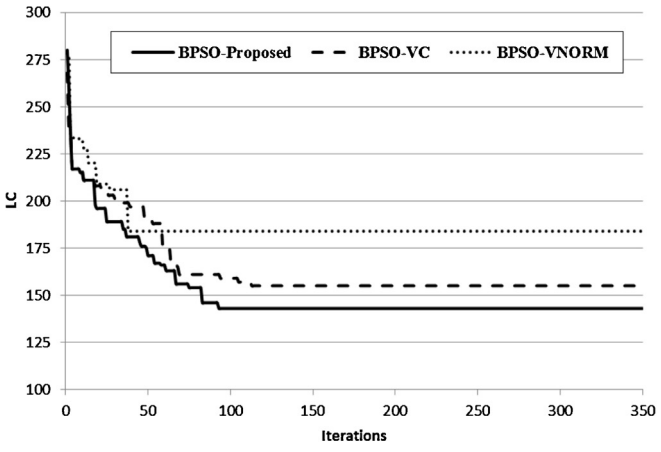
Table 3
Comparison of proposed BPSO with other BPSO techniques.

Circuits	BPSO-proposed				BPSO-VCHANGE [22]				BPSO-VNORM [14]			
	Best	Avg	Worst	Avg time[a]	Best	Avg	Worst	Avg time	Best	Avg	Worst	Avg time
bbara	49	52.9	55	792.6	49	55.0	58	778.7	58	59.5	61	776.3
bbsse	102	107.1	111	823.1	99	106.6	115	823.4	114	118.0	122	836.4
cse	184	191.0	198	987.4	183	191.5	203	970.2	192	202.0	210	1045.3
dk14	98	99.8	102	783.6	100	101.6	103	778.5	98	99.1	101	785.9
ex2	66	99.5	117	861.4	66	101.2	119	860.1	137	140.8	146	871.6
ex3	51	54.0	56	796.4	53	53.6	56	789.8	54	57.3	60	768.7
keyb	143	163.8	178	1479.6	155	167.7	184	1361.4	184	208.2	217	1883.4
lion9	10	11.7	13	785.6	10	11.7	14	788.3	11	12.6	14	763.7
planet	494	526.1	561	2288.4	516	534.1	553	2219.3	577	590.4	603	2385.2
pma	155	164.6	181	1163.3	143	163.8	178	1159.4	198	205.0	213	1251.3
s1	173	231.7	278	1591.7	180	235.2	281	1464.6	362	372.0	388	1873.4
s1494	588	602.0	628	3073.4	564	589.1	630	2987.4	658	665.9	675	3201.1
s832	216	245.4	267	1750.5	232	249.1	278	1663.6	290	311.1	328	1916.2
sand	488	510.1	527	2374.4	490	510.6	546	2300.2	555	580.6	595	2601.0
styr	412	437.5	455	2326.2	427	447.3	475	2153.4	539	555.2	571	2598.7
tav	24	24.0	24	744.0	24	24.0	24	770.2	24	24.0	24	802.1
tbk	261	368.2	458	6671.1	292	381.6	468	5952.5	669	692.1	715	9155.1
train11	12	13.7	15	773.4	12	14.3	19	782.1	14	16.1	12	780.9
Total	3502	3878.7	4200	–	3571	3915.0	4280	–	4710	4885.9	5040	–

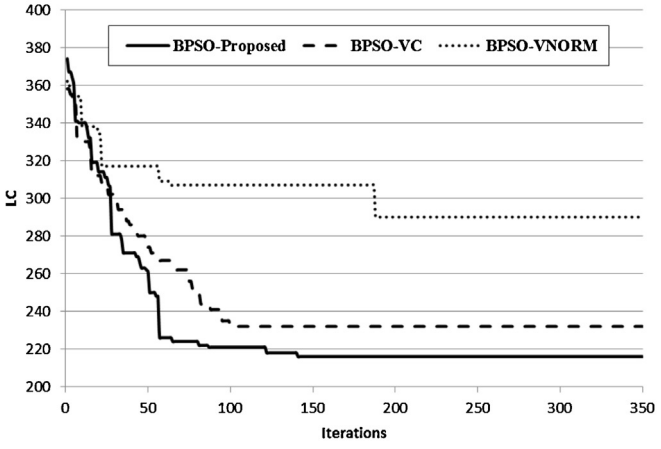
[a] In seconds.

Table 4
Comparison of proposed BPSO with other state assignment techniques.

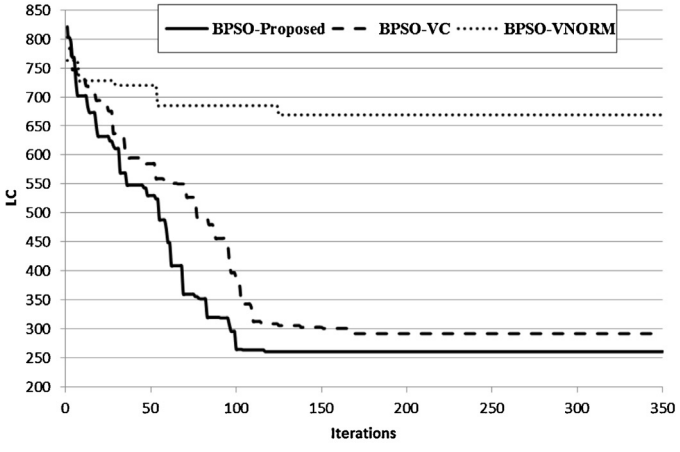
Benchmark	BPSO-proposed				GA [20]				SimE [30]	Jedi [2]	Nova [5]
	Best	Avg	Worst	Avg time	Best	Avg	Worst	Avg time			
bbara	49	52.5	55	792.6	49	49.4	58	1204.3	59	73	57
bbsse	102	107.1	111	823.1	101	100.5	106	1350.7	117	134	140
cse	184	191.0	198	987.4	179	184.2	190	1743.7	–	240	214
dk14	98	99.8	102	783.6	102	103.1	105	1543.2	161	108	111
ex2	66	99.5	117	861.4	64	90.1	120	1373.7	–	123	127
ex3	51	54.0	56	796.4	54	54.7	58	1149.1	–	65	71
keyb	143	163.8	178	1479.6	142	152.4	165	2096.1	–	260	201
lion9	10	11.7	13	785.6	10	10.0	10	1083.7	17	19	27
planet	494	526.1	561	2288.4	466	505.4	560	3775.0	–	603	591
pma	155	164.6	181	1163.3	160	165.3	180	1676.5	–	263	241
s1	173	231.7	278	1591.7	132	216.2	300	2270.9	–	282	340
s1494	588	602.0	628	3073.4	570	591.1	620	3885.1	–	679	715
s832	216	245.4	267	1750.5	230	256.7	280	2254.9	–	357	274
sand	488	510.1	527	2374.4	498	519.8	557	3002.6	–	554	558
styr	412	437.5	455	2326.2	405	422.5	520	3506.2	–	518	502
tav	24	24.0	24	744.0	24	24.0	24	893.8	24	305	365
tbk	261	368.2	458	6671.1	410	462.1	560	8060.8	–	305	365
train11	12	13.7	15	773.4	18	18.5	20	1102.0	27	34	32
Total	3502	3878.7	4200	–	3590	3902.0	4409	–		4617	4566



(a) *keyb*



(b) *s832*



(c) *tbk*

Fig. 8. Literal count (LC) versus iterations of best cases for few benchmarks.

[30] and with well known deterministic algorithms like Nova [5] and Jedi [2]. Compared to Jedi and Nova, the proposed algorithm achieves far better results for all benchmark circuits. Compared to the results reported by SimE, the proposed algorithm outperforms it in all evaluation parameters. The total literal count achieved by the proposed *BPSO* for *Best*, *Average* and *Worst* categories are also better than GA. It can also be observed that the time taken by GA is exorbitantly high for each circuit as compared to the proposed algorithm. Furthermore, in comparison to the work reported in [21], the literals count achieved by three reported benchmarks *Shiftreg*, *Lion9* and *Train11* is 13, 27 and 39, whereas the number of literals achieved by the proposed algorithm are 2, 10 and 12, respectively. All these results demonstrate the competitiveness of proposed algorithm in comparison to existing algorithms reported in the literature.

7. Conclusion

In this work, an improved binary particle swarm optimization (BPSO) algorithm is proposed that overcomes the drawbacks of the originally proposed BPSO algorithm. The effectiveness of the proposed algorithm is demonstrated by its application to the state assignment problem in sequential circuit synthesis targeting area optimization. Experimental results on benchmark circuits demonstrate the effectiveness of the proposed BPSO algorithm in terms of lower area in comparison to other BPSO variants reported in the literature. Better results are also achieved in comparison to Genetic Algorithm (GA) and deterministic state encoding algorithms (Nova and Jedi). It is worth mentioning that the proposed BPSO approach can be used to solve multitude of optimization problems like routing and scheduling, FPGA cell placement, etc.

Acknowledgment

This work is supported by King Fahd University of Petroleum & Minerals under project # SAB-2005/09.

References

[1] D.B. Armstrong, A programmed algorithm for assigning internal codes to sequential machines, IRE Transactions on Electronic Computers EC-11 (4) (August 1962) 466–472.
[2] B. Lin, A.R. Newton, Synthesis of multiple-level logic from symbolic high-level description languages, in: IFIP International Conference on Very Large Scale Integration, 1989 August, pp. 187–196.
[3] S. Devadas, H.T. Ma, A.R. Newton, A. Sangiovanni-Vincentelli, MUSTANG: state assignment of finite state machines for optimal multilevel logic implementations, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 7 (1988 December) 1290–1300.
[4] G. DeMicheli, R.K. Brayton, A. Sangiovanni-Vincenteli, Optimal state assignment for finite state machines, IEEE Transaction on Computer Aided Design CAD-4 (3) (1985) 269–285.
[5] T. Villa, A. Sangiovanni-Vincentelli, Nova: state assignment of finite state machines for optimal two-level logic implementations, in: Proceedings of the 1989 26th ACM/IEEE conference on Design automation conference, 1989, pp. 327–332.
[6] J.N. Amaral, W.C. Cunha, State assignment algorithm for incompletely specified state machines, in: Fifth Congress of the Brazilian Society of Microelectronics, 1990, pp. 174–183.
[7] X. Du, G. Hachtel, B. Lin, A.R. Newton, MUSE: a multilevel symbolic encoding algorithm for state assignment, IEEE Transactions on Computer Aided Design CAD-10 (1991) 28–38.
[8] P. Ashar, S. Devadas, A. Richard Newton, Sequential Logic Synthesis, Kluwer Academic Publishers, Norwell, MA, USA, 1992.
[9] B. Eschermann, State assignment hardwired VLSI control units, ACM Computer Survey 25 (1993 December) 415–436.
[10] A.E.A. Almaini, J.F. Miller, P. Thomson, S. Billina, State assignment of finite state machines using a genetic algorithm, Comptuers and Digital Techiniques, IEE Processings 142 (4) (1995 July) 279–286.
[11] J. Kennedy, R. Eberhart, Particle swarm optimization, in: IEEE International Conference on Neural Networks, vol. 4, 1995 November/December, pp. 1942–1948.
[12] J. Amaral, K. Turner, J. Ghosh, Designing genetic algorithm for state assignment problem, IEEE Transactions on SMC 25 (1995) 659–694.
[13] http://www.cbl.ncsu.edu:16080/benchmarks/LGSynth89/fsmexamples
[14] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, in: IEEE International Conference on Computational Cybernetics and Simulation, vol. 5, 1997 October, pp. 4104–4108.
[15] S.M. Sait, H. Youssef, Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems, IEEE Computer Society Press, CA, 1999 December.
[16] Y. Xia, A.E.A. Almaini, Genetic algorithm based state assignment for power and area optimization, IEEE Proceedings Computers and Digital Techniques 149 (2002) 128–133.

[17] M. Chyzy, W. Kosinski, Evolutionary algorithm for state assignment of finite machines, in: Euromicro Symposium on Digital System Design, 2002, pp. 359–362.
[18] S. Chattopadhyay, A. Chetry, S. Biswas, State assignment and selection of types and polarities of flip-flops, for finite state machine synthesis, in: India Annual Conference, 2004. Proceedings of the IEEE INDICON 2004, First, pp. 27, 30, 20–22 December, 2004.
[19] F.N. Khan, FSM State Assignment for Area, Power and Testability using Non-Deterministic Evolutionary Heuristics, KFUPM, Saudi Arabia, 2005 (Master's thesis), Computer Aided Design, CAD-10:28–38, 1991.
[20] A. El-Maleh, S.M. Sait, F. Nawaz Khan, Finite state machine state assignment for area and power minimization, in: IEEE International Symposium on Circuits and Systems, ISCAS 2006, 21–24 May, 2006.
[21] N. Nedjah, L. de Macedo Mourelle, Evolutionary synthesis of synchronous finite state machines, in: International Conference on Computer Engineering and Systems, 2006 November, pp. 19–24.
[22] M.A. Khanesar, M. Teshnehlab, M.A. Shoorehdeli, A novel binary particle swarm optimization, in: Mediterranean Conference on Control & Automation, MED'07. vol. 1, 27–29 June, 2007, pp. 1–6.
[23] V. Salauyou, T. Grzes, FSM state assignment methods for low-power design, in: 6th International Conference on Computer Information Systems and Industrial Management Applications, CISIM'07, 28–30 June, 2007, pp. 345–350.
[24] S.N. Pradhan, M.T. Kumar, S. Chattopadhyay, Integrated power-gating and state assignment for low power FSM synthesis, in: IEEE Computer Society Annual Symposium on VLSI, ISVLSI'08, 7–9 April, 2008, pp. 269–274.
[25] S. Chaudhury, K. Teja Sistla, S. Chattopadhyay, Genetic algorithm-based FSM synthesis with area-power trade-offs, Integration, the VLSI Journal 42 (3) (2009 June) 376–384.
[26] W.M. Aly, Solving the state assignment problem using stochastic search aided with simulated annealing, American Journal of Engineering and Applied Sciences 2 (2009) 703–707.
[27] A. Sagahyroon, F.A. Aloul, A. Sudnitson, Low power state assignment using ILP techniques, in: 15th IEEE Mediterranean Electrotechnical Conference, 26–28 April, 2010, pp. 850–855.
[28] Sentovich, E.M., Singh, K.J., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P.R., Brayton, R.K., Sangiovanni-Vincentelli, A.L., SIS: A System for Sequential Circuit Synthesis, EECS Department, University of California, Berkeley, 1992. http://www.eecs.berkeley.edu/Pubs/TechRpts/1992/2010.html
[29] B.A. Al Jassani, N. Urquhart, A.E.A. Almaini, State assignment for sequential circuits using multi-objective genetic algorithm, IET Computers & Digital Techniques 5 (4) (2011 July) 296–305.
[30] F.C. Oughali, Sait, M. Sadiq, A.M. Arafeh, FSM state-encoding for area and power minimization using simulated evolution algorithm, Journal of Applied Research and Technology 10 (2012) 845–858.