Chapter 9
Synchronous Finite State Machines Design with Quantum-Inspired Evolutionary Computation*

Abstract. Synchronous finite state machines are very important for digital sequential designs. Among other important aspects, they represent a powerful way for synchronizing hardware components so that these components may cooperate adequately in the fulfillment of the main objective of the hardware design. In this chapter, we propose an evolutionary methodology based on the principles of quantum computing to synthesize finite state machines. First, we optimally solve the state assignment *NP*-complete problem, which is inherent to designing any synchronous finite state machines. This is motivated by the fact that with an optimal state assignment, one can physically implement the state machine in question using a minimal hardware area and response time. Second, with the optimal state assignment provided, we propose to use the same evolutionary methodology to yield an optimal evolutionary hardware that implements the state machine control component. The evolved hardware requires a minimal hardware area and imposes a minimal propagation delay on the machine output signals.

9.1 Introduction

Sequential digital systems or simply finite state machines (FSMs) have two main characteristics: there is at least one feedback path from the system output signal to the system input signals; and there is a memory capability that allows the system to determine current and future output signal values based on the previous input and output signal values [1].

Traditionally, the design process of a state machine passes through five main steps, wherein the second and third steps may repeated several times as shown in Figure 9.1.

1. the specification of the sequential system, which should determine the next states and outputs of every present state of the machine. This is done using state tables and state diagrams;

* This chapter was developed together with Marcos Paulo Mello Araujo.

N. Nedjah and L. de Macedo Mourelle, *Hardware for Soft Computing and Soft Computing for Hardware*, Studies in Computational Intelligence 529,
DOI: 10.1007/978-3-319-03110-1_9,

2. the state reduction, which should reduce the number of present states using equivalence and output class grouping;
3. the state assignment, which should assign a distinct combination to every present state. This may be done using Armstrong-Humphrey heuristics [1, 2, 3];
4. the minimization of the control combinational logic using K-maps and transition maps;
5. finally, the implementation of the state machine, using gates and flip-flops.

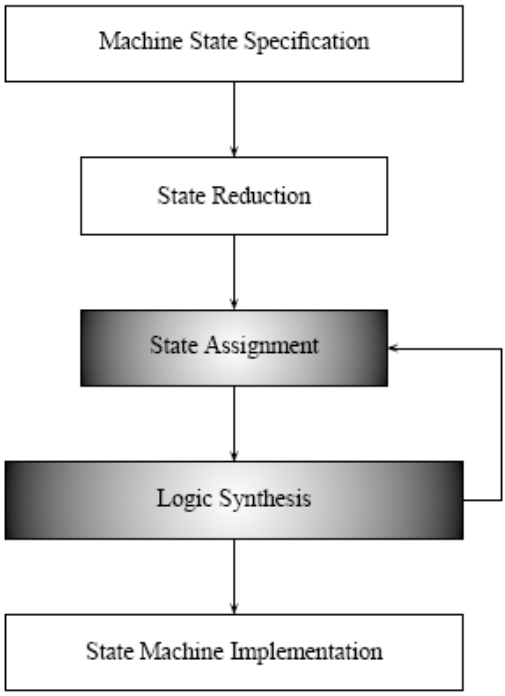


Fig. 9.1 Design methodology for sequential circuits

In this chapter, we concentrate on the third and forth steps of the design process, i.e. the state assignment and the control logic minimization problems. We present a quantum-inspired genetic algorithm designed to find a state assignment of a given synchronous finite state machine, which attempts to minimize the cost related to the state transitions. Then, we adapt the same quantum-inspired evolutionary algorithm to evolve the circuit that controls the machine current and next states.

The problems involved in state machine synthesis have been extensively studied in the past [2, 3, 1, 4]. These studies can be applied to state machine with a limited complexity, i.e. few state and transitions to control. Furthermore, the evolutionary

principle in the form of genetic algorithms and genetic programming has been explored to solve these problems [5, 6, 7, 8]. The application of this principle allowed designers to synthesize more complex, state machines, i.e. with a little more states and transitions, and without much design effort. However, when the complexity of the state machine at hand goes beyond a certain limit, this applications fails to yield interesting synthesis results and also, the execution extends over hours of evolution One of the attractive properties of quantum computing is the possibility of massive parallelism, as it will be detailed later in the next sections of this chapter. This parallelism is not explicit. Instead, it is embedded within the information representation.

The use of both the evolutionary principle combined with that of quantum computing should allow us to improve further the synthesis process both in terms of improving the quality of the yielded results and also in synthesizing more complex state machine with no design effort and with shorter evolution time. The results presented towards the end of this chapter prove that the use of the quantum-inspired evolutionary process is very efficient. Using the proposed algorithm, we were able to synthesize automatically and evolutionary very complex state machines in a record time. In practical terms, our algorithm can be embedded in hardware synthesis tools to improve the quality of the synthesis result and generate those result efficiently.

The remainder of this chapter is organized into six sections. In Section 9.2, we introduce the problems that face the designer of finite state machine, which are mainly the state assignment problem and the design of the required control logic. In Section 9.3, we show that a well chosen assignment improves considerably the cost of the control logic. In Section 9.4, we give a thorough overview on the principles of quantum computing. In Section 9.5, we design a quantum-inspired genetic algorithm, which we call *QIGA* for evolving innovative solutions to hard *NP*-complete problems. In Section 9.6, we apply QIGA to the state assignment problem and we describe the genetic operators used as well as the fitness function, which determines whether a state assignment is better that another and how much. Subsequently, in Section 9.7, we present a quantum-inspired synthesizer for evolving efficient control logic circuit given the state assignment for the specification of the state machine in question. Then, we describe the circuit encoding, quantum gates used as well as the fitness function, which determines whether a control logic design is better than another and how much. Towards the end of this chapter, in Section 9.8, we present the results evolved by QIGA for some well-known FSM benchmarks. Then we compare the obtained results with those obtained by another genetic algorithm described in [7, 6] as well as with NOVA^{TM}, which uses well established but non-evolutionary methods [9]. We also provide the area and time requirements of the designs evolved through our evolutionary synthesizer for those benchmarks and compare the yielded results with those obtained using the traditional method to design state machines [9]. Last but no least, in Section 9.9, we draw some conclusions about this study and give some directions for future work.

9.2 Design Methodology of Synchronous Finite State Machines

Digital systems can be classified as *combinational systems* or *sequential systems*. A combinational system must obey the following restrictions [1]:

1. The values 0/1 of the output signals must depend on the actual values 0/1 of the input signals only.
2. There should be no feedback of the output signals to the input signals.

The aforementioned two restrictions make the design and analysis of combinational systems a straightforward task. Each output signal can be expressed as a Boolean function of the input signals. For a combinational system of n input signals and m output signals, we can have:

$$o_j = \phi_j(i_1, i_2, \ldots, i_n), \quad j = 1, 2, \ldots, m \tag{9.1}$$

wherein $i_1, i_2, \ldots, i_n$ area the input signals, $o_1, o_2, \ldots, o_m$ are the output signals and ϕ_1, ϕ_2, …, ϕ_m form the necessary m Boolean function that yield the output signals.

In many digital systems, the output signal behavior cannot be determined knowing only the actual behavior of the input signals. In this case, the history of the input and output signals must be used to do so. Sequential systems are fundamentally different from the combinational ones, in spite of the fact that the former also include a combinational part. The term *sequential* is commonly used to describe this distinction. Sequential systems present two main characteristics:

1. There exists at least on path of feedback between the output and input signals of the system;
2. The circuit has the ability of remember information about the system past, in such a way that previous values of the output signals could be used to determine their respective next values.

The removal of the combinational restrictions allows for a larger spectrum for the application of digital systems. The use of memory elements and the feedback feature allow for the consideration of the time element as a parameter in the definition of the system behavior. Therefore, the information related to past events can be used to determine the behavior of the output signals. Moreover, information about both the past and the present can be captured as to plan and specify some future activities.

A clear advantage that can be observed through the comparison of sequential and purely combinational systems is the reduction of the hardware required due to the repetitive nature of sequential systems. However, a sequential system almost always requires more time to execute tasks [1]. The generic architecture of a Mealy finite state machine is given in Figure 9.2.

The input signals of a sequential system can be divided into two groups: *primary input signals* ($i_1, i_2, \ldots, i_n$) and *secondary input signals* ($p_1, p_2, \ldots, p_k$). The behavior of the primary input signals define the actual value of the system input, which can be one of the 2^n different possible combinations. The behavior of the secondary input signals reflects the past history of the sequential system. These signals are also called *current state signals* and whose values are read from the system memory.

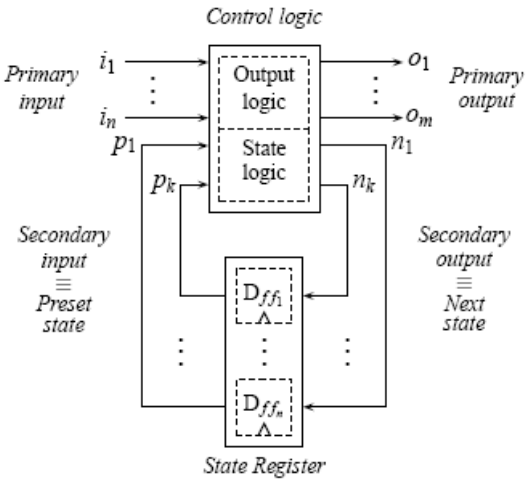


Fig. 9.2 A structural description of a Mealy state machine

The system ability to remember information about the past can be implemented through the utilization of *flip-flops* or *latches* [10]. The set of flip-flops used is generally called the *state register*. The k signal values of the secondary input form what is commonly known as the *present state* of the system. Therefore, the system may have 2^k distinct possible states. For this reason, sequential systems are also commonly called as *finite state systems* [10]. The *total state* of the system is defined as the union of the two sets of primary and secondary input signals. So, there are 2^{n+k} different total states.

The output signals can also be divided into two groups: *primary output signals* (o_1, o_2, …, o_m) and *secondary output signals* ($n_1, n_2, \ldots, n_k$). The primary output signals form the control signals that are sent to the environment in which the sequential system is embedded. The secondary output signals form the data for the sequential system memory. These signals present the new value that will be saved into the system memory as soon as the next cycle of operation starts. Therefore, the secondary output signals are commonly called the *next state* of the system. In the same moment that the next state signals are written into the state register, the system passes to show this state as the present state. The primary and secondary output signals of the system are yield by combinational operations on the total state signals.

The design methodology of a state machine that controls the behavior of a given digital system may be subdivided into the following main steps:

- **Machine Specification:** The relationship between the present state signals and the primary input signals and that between the next state signals and the primary output signals describes the behavior of the sequential system. This relationship

can be represented in many different ways. The most commonly used representations are the *state transition diagram* and the *state transition table*.

- **State Reduction:** States that produce the same output signal and have the same next state behavior are identified as *equivalent* and so are combined into a single state that acts in substitution to all these equivalent states. Equation 9.2 suggests that the total number of states that are necessary during the operation of the sequential system, say n, determine the minimal number of the state signals in that system implementation. Therefore, reducing the number of the included states yields a reduction in the state register size and also may lead to a reduction in the complexity of the control logic required. Some techniques used for the identification of equivalent states and the simplification of the state machine model can be found in [4].

$$K = \lceil \log_2(n) \rceil \tag{9.2}$$

- **State Assignment:** Once the specification and the state reduction steps have been completed, the following step consists then of assigning a code to each state present in the machine. It is clear that if the machine has N distinct states then one needs N distinct combinations of 0s and 1s. So, one needs K flip-flops to store the machine current state, wherein K is the smallest positive integer such that $2^K \geq N$. The state assignment problem consists of finding the best assignment of the flip-flop combinations to the machine states. Since a machine state is nothing but a counting device, a combinational control logic is necessary to activate the flip-flops in the desired sequence. A generic architecture of a machine state is shown in Figure 9.2, wherein the feedback signals constitute the machine state, the control logic is a combinational circuit that computes the state machine primary output signals from the current state signals and the primary input signals. It also produces the signals of the machine next state.

  Let n be the number of states in a given machine and so $b = \lceil \log_2 n \rceil$ flip-flops are needed to store the machine state. A state assignment consists of identifying the 2^b binary codes that should be used to identify the machine n states. The number of possible distinct state assignments $f(n,b)$ [11] is given in Equation 9.3.

$$f(n,b) = \frac{2^b}{(2^b - n)} \tag{9.3}$$

  Table 9.2 shows the values obtained for f when applied to some specific values of n and b. For instance, if the evaluation of an assignment as to its impact on the state machine implementation lasts say 100 μs, then 66 years would be needed to test all possible assignments, which cannot be done. Therefore, it is essential to use heuristics to overcome this problem.

- **Logic Synthesis:** The control logic component in a state machine is responsible for generating the primary output signals as well as the signal that form the next state. It does so using the primary input signals and the signals that constitute the current state (see Figure 9.2). Traditionally, the combinational circuit of the control logic is obtained using the transition maps of the flip-flops [1]. Given a state transition function, it is expected that the complexity, in terms of area and time,

Table 9.1 Number of possible state assignments

n	b	$f(n,b)$
2	1	2
3	2	24
4	2	24
5	3	6720
6	3	20160
7	3	40320
8	3	40320
9	4	$\approx 4 \cdot 10^9$
10	4	$\approx 3 \cdot 10^{10}$
11	4	$\approx 2 \cdot 10^{11}$
12	4	$\approx 9 \cdot 10^{11}$
13	4	$\approx 3 \cdot 10^{12}$
14	4	$\approx 1 \cdot 10^{13}$
15	4	$\approx 2 \cdot 10^{13}$

and so the cost of the control logic will vary for different assignments of flip-flop combinations to the allowed states. Consequently, the designer should seek the assignment that minimizes the complexity and so the cost of the combinational logic required to control the state transitions.

9.3 Impact of State Assignment

Given a state transition function, the requirements of area and time vary with respect to the state assignment used. Therefore, the designer or the computer-aided design tool for circuit synthesis needs always to select carefully the state assignment to be used. Existing techniques for state assignment can be listed as follows:

- *One-hot*: This technique associates a bit in the state register to each one of the existing state. This simplifies a great deal the synthesis flux as the control logic circuit can be obtained on-the-fly. However, it requires a register state whose size is defined by the number of states in the machine [10].
- *Heuristics*: These techniques attempt to identify a "good" assignment based on some heuristics. For instance, in [2] and [3], a heuristic based on state code adjacency, which attempts to assign adjacent codes to states that are "close" considering the state transition function. Two states are said to be *close* if one is the next state to the other and two binary codes are said to be *adjacent* if these are distinct in one single position. The idea behind this heuristic is the fact that adjacent binary codes will appear next to each other in Karnaugh maps and therefore would allow larger grouping, when necessary.

- *Meta-heuristics*: Evolutionary algorithms are used to evolve efficient assignments, rendering the assignment problem to an optimization one [6, 12]. These algorithms have been proven very efficient, very robust and the results obtained are far superior to those yield by the heuristic-based techniques

In order to demonstrate the impact of the chosen state assignment on the control logic complexity in terms of area and response time, let us consider the state machine described in Table 9.2 and try two different state codifications, which are $assignment_1 = \{00,11,01,10\}$ and $assignment_2 = \{00,01,11,10\}$. The circuit schematics for the state machine using $assignment_1$ and $assignment_2$ are shown in Figure 9.3 and 9.4 respectively.

Table 9.2 Example of state transition table

present state	next state		output (O)	
	$I=0$	$I=1$	$I=0$	$I=1$
s_0	s_0	s_1	0	0
s_1	s_2	s_1	0	1
s_2	s_0	s_3	1	0
s_3	s_2	s_1	1	1

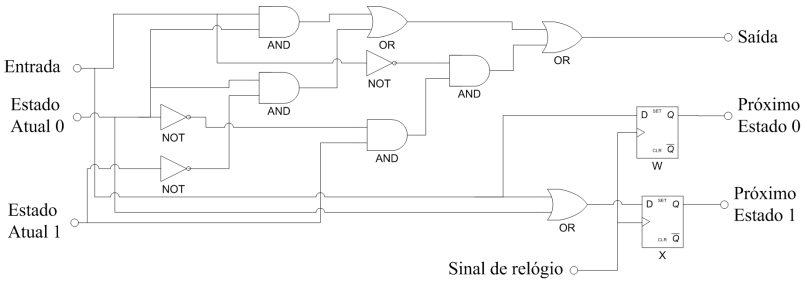


Fig. 9.3 Circuit schematics for the state machine using $assignment_1$

This example proves that the appropriate state assignment can reduce the implementation cost of the machine. The cost is defined here as the number of gates NOT, AND and OR of two one-bit inputs used. The inverted output signal of the flip-flops are considered of cost zero for the circuit implementation as these are available as output from the flip-flops. Assuming that the implementation cost of a given circuit is defined as the number of logic gates included, then Table 9.3 summarizes this cost for several possible state assignments, including $assignment_1$ and $assignment_2$. The afore-described example is an illustration of the fact that the choice of state assignment can reduce considerably the cost of state machine implementations, if chosen carefully.

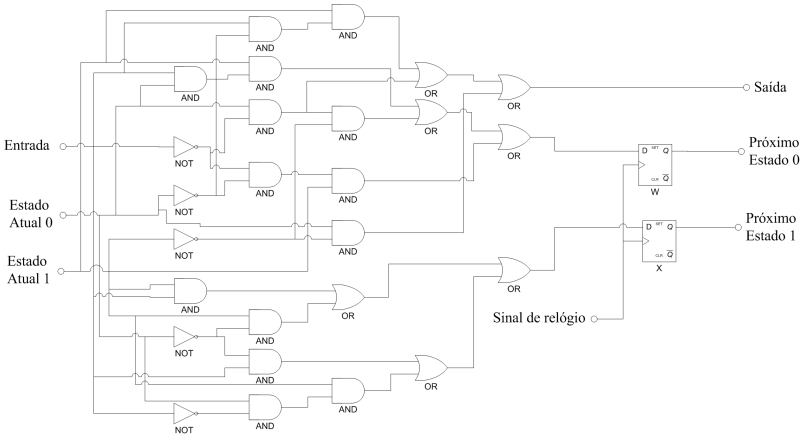


Fig. 9.4 Circuit schematics for the state machine using $assignment_1$

Table 9.3 Comparison of the number of logic gates for several possible state assignments

assignment	#AND	#OR	#NOT	Total
[00,11,01,10]	4	3	1	8
[00,01,10,11]	5	2	1	8
[00,10,01,11]	5	2	1	8
[00,11,10,01]	5	3	1	9
[11,00,01,10]	5	3	1	9
[00,01,11,10]	10	7	1	18
[00,10,11,01]	11	6	1	18

In Section 9.6, we concentrate on the third step of the design process, i.e. the state assignment problem. We present a quantum-inspired genetic algorithm, designed for finding a state assignment of a given synchronous finite state machine, which attempts to minimize the cost related to the state transitions. In Section 9.7, we focus on evolving minimal control logics for state machines for a given state assignment and using an adapted version of the quantum-inspired genetic algorithm. Before getting to that, however, we first give an introduction to quantum computing and then we sketch the proposed algorithm.

9.4 Principles of Quantum Computation

Quantum computing is based on the concepts of quantum mechanics and is expected to be one of the main pillars of next generation computers. Many researchers are already using the principles of quantum computing to develop new techniques and algorithms to take advantage of the underlaying benefits [13, 14]. The basic

elements of quantum computing are: *quantum bits*, *quantum registers*, *quantum gates* and *quantum circuits*. These concepts are defined in the remainder of this section.

9.4.1 Quantum Bit

In quantum computing, the smallest unit of information stored in a two-state system is called a quantum bit or *qubit* [15]. The 0 and 1 states of a classical bit, are replaced by the state vectors $|0\rangle$ and $|1\rangle$ of a qubit. This vectors are usually written using the *bracket* notation, introduced by Paul Dirac in [16]. The state vectors of a qubit are represented as in Equation 9.4:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{e} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{9.4}$$

While the classical bit can be in only one of the two basic states that are mutually exclusive, the generic state of one qubit can be represented by a linear combination of the state vectors $|0\rangle$ and $|1\rangle$, as in Equation 9.5:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle, \tag{9.5}$$

wherein α and β are complex numbers. The state vectors $|0\rangle$ and $|1\rangle$ form a canonical base and the vector $|\psi\rangle$ represents the superposition of this vectors, with α and β amplitudes. The unit normalization of the state of the qubit ensures that Equation 9.6 is true:

$$|\alpha|^2 + |\beta|^2 = 1. \tag{9.6}$$

The phase of a qubit is defined by an angle ζ, defined as in Equation 9.7:

$$\zeta = \arctan(\beta/\alpha), \tag{9.7}$$

and tthe quadrant of qubit phase ζ is defined as in (9.8). If d is positive, the phase ζ lies in the first or third quadrant; otherwise, the phase ζ lies in the second or fourth quadrant [17].

$$d = \alpha \cdot \beta, \tag{9.8}$$

The physical interpretation of the qubit is that it may be simultaneously in the states $|0\rangle$ and $|1\rangle$, which allows for an infinite amount of information to be stored in state $|\psi\rangle$. However, during the act of observing the state of a qubit, it collapses to a single state, i.e. either $|0\rangle$ or $|1\rangle$ [18]. The qubit collapses to state $|0\rangle$, with probability $|\alpha|^2$ or state $|1\rangle$, with probability $|\beta|^2$.

9.4.2 Quantum Registers

A system with m qubits contains information on 2^m states. The linear superposition of possible states can be represented as in Equation 9.9:

$$|\psi\rangle = \sum_{k=1}^{2^m} C_k |S_k\rangle , \tag{9.9}$$

wherein C_k specifies the probability amplitude of the corresponding states $|S_k\rangle$ and subjects to the normalization condition of Equation 9.10.

$$|C_1|^2 + |C_2|^2 + ... + |C_{2^m}|^2 = 1 \tag{9.10}$$

9.4.3 Quantum Gates

The state of a qubit can be changed by the operation of a quantum gate or q-gate. The q-gates applies a unitary operation U on a qubit in the state $|\psi\rangle$, making it evolve to the state $U|\psi\rangle$, maintaining the probabilities interpretation defined in Equation 9.6. There are several q-gates, such as the *NOT* gate, *Controlled-NOT* gate, *Hadamard* gate, *rotation* gate [15].

9.5 Quantum-Inspired Genetic Algorithms

Since the emerging of evolutionary computation field, many new hybridized algorithms and technique based on the main concepts of evolution have been developed. Just to name few, we can cite multi-objective evolutionary algorithms [20, 21], swarm-based techniques [19], differential evolution [22] and quantum-inspired evolutionary algorithm [23]. As any evolutionary algorithms, the latter is based on a population of solutions which is maintained through many generations. It seeks the best fitted solution to the problem, by evaluating the characteristics of those included in the current population. In the next section, we describe the quantum-inspired representation of the individual and the underlaying computational process.

9.5.1 Solution Representation

Evolutionary algorithms, like genetic algorithms, for instance, can use several representations that have been used with success: binary, integer, real or even symbolic [24]. The quantum-inspired evolutionary algorithms use a new probabilistic representation, that is based on the concept of qubits as defined in Equation 9.5 and q-individuals, which consist of a string of qubits. A q-individual, say p, can be viewed as in Equation 9.11, wherein $|\alpha_i|^2 + |\beta_i|^2 = 1$, for $i = 1, 2, 3, ..., m$.

$$p = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_m \\ \\ \beta_1 & \beta_2 & \beta_3 & \cdots & \beta_m \end{bmatrix} \tag{9.11}$$

The advantage of the representation of the individuals using qubits instead of the classical representation of bits is the ability of representing the linear superpositions

of all possible states. For instance, an individual represented with three qubits ($m = 3$) can be depicted as in Quation 9.12:

$$p = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{1}{2} \\ \\ \frac{1}{\sqrt{2}} & \sqrt{\frac{2}{3}} & \frac{\sqrt{3}}{2} \end{bmatrix}, \tag{9.12}$$

or viewed in the alternative way of Equation 9.13,

$$p = \frac{1}{2\sqrt{6}}|000\rangle + \frac{1}{2\sqrt{2}}|001\rangle + \frac{1}{2\sqrt{3}}|010\rangle + \frac{1}{2}|011\rangle + \frac{1}{2\sqrt{6}}|100\rangle + \frac{1}{2\sqrt{2}}|101\rangle + \frac{1}{2\sqrt{3}}|110\rangle + \frac{1}{2}|111\rangle \tag{9.13}$$

The numbers in Equation 9.13 represent the amplitudes whose square-roots indicate the probabilities of observing states $|000\rangle$, $|001\rangle$, $|010\rangle$, $|011\rangle$, $|100\rangle$, $|101\rangle$, $|110\rangle$ and $|111\rangle$, which are $\frac{1}{24}$, $\frac{1}{8}$, $\frac{1}{24}$, $\frac{1}{12}$, $\frac{1}{24}$, $\frac{1}{8}$, $\frac{1}{24}$ and $\frac{1}{12}$, respectively.

The evolutionary algorithms with the quantum-inspired representation of individuals should permit a population diversity better than other representations, since the included individuals can represent linear superpositions of all possible states [25, 23]. For instance, the single q-individual of Equation 9.12 is enough to represent eight states. When using the classical representation of bits, eight individuals would be necessary to encode the same information.

9.5.2 Algorithm Description

The basic structure of the quantum-inspired evolutionary algorithm used in this chapter is described by Algorithm 9.1 [26].

The quantum-inspired evolutionary algorithm maintains a population of q-individuals, $P(g) = \left\{\mathrm{p}_1^g, \mathrm{p}_2^g, ..., \mathrm{p}_n^g\right\}$ at generation g, where n is the size of population, and p_j^g is a q-individual defined as in Equation 9.14:

$$\mathrm{p}_j^g = \begin{bmatrix} \alpha_{j_1}^g & \alpha_{j_2}^g & \alpha_{j_3}^g & \cdots & \alpha_{j_m}^g \\ \\ \beta_{j_1}^g & \beta_{j_2}^g & \beta_{j_3}^g & \cdots & \beta_{j_m}^g \end{bmatrix}, \tag{9.14}$$

where m is the number of qubits, which defines the string length of the q-individual, and $j = 1, 2, ..., n$.

The initial population of n individuals is generated setting $\alpha_i^0 = \beta_i^0 = 1/\sqrt{2}$ ($i = 1, 2, ..., m$) of all $\mathrm{p}_j^0 = \mathrm{p}_j^g\,|_{g=0}$ for $j = 1, 2, ..., n$. This allows each q-individual to be the superposition of all possible states with the same probability.

The binary solutions in S_g are obtained by an observation process of the states of every q-individual in P_g. Let $S_g = \left\{\mathrm{s}_1^g, \mathrm{s}_2^g, ..., \mathrm{s}_n^g\right\}$ at generation g. Each solution, s_i^g for $i = 1, 2, ..., n$, is a binary string with the length m, that is, $\mathrm{s}_i^g = s_1 s_2 ... s_m$, where s_j is either 0 or 1.

Algorithm 9.1. Quantum-Inspired Genetic Algorithm – QIGA

```
g := 0;
generate P_0 with n individuals
observe P_0 into S_0
evaluate the fitness of every solution in S_0
store S_0 into B_0
while (not termination condition) do
   g := g + 1;
   observe P_{g-1} into S_g
   evaluate the fitness of every solution in S_g
   update P_g using a q-gate and apply probability constraints
   store best solutions of B_{g-1}, S_g in B_g
   store the best solution in B_g into b
   if (no improvement for many generation) then
      replace all the solution of B_g by b
   end if
end while
return b
```

The observation process is implemented using random probability: for each pair of amplitudes $[\alpha_k, \beta_k]^T$ for $k = 1, 2, ..., n \times m$ of every qubit in the population P_g, a random number r in the range $[0, 1]$ is generated. If $r < |\beta_k|^2$, the observed qubit is 1; otherwise, it is 0.

The q-individuals in P_g are updated using a q-gate, which is detailed later in the next section. We impose some probability constraints such that the variation operation performed by the q-gate avoid a premature convergence of a qubit to either to 0 or 1. This is done by allowing neither of $|\alpha|^2$ nor $|\beta|^2$ to reach 0 or 1. For this purpose, the probability $|\alpha|^2$ and $|\beta|^2$ are constrained to 0.02 as a minimum and 0.98 as a maximum. Such constraints allowed the algorithm to escape local minimum. This variation is one of the contribution of the chapter and has not been introduced in the original version of the algorithm [23].

After a given number of generations, if the best solution b does not improve, all the solutions stored into B_g are replaced by b. This step can induce a variation of the probabilities of the qubits within the q-individuals. This operation is also performed in order to escape local minimum and avoid the stagnant state.

9.6 State Assignment with QIGA

The identification of a good state assignment has been thoroughly studied over the years. In particular, Armstrong [2] and Humphrey [3] have pointed out that an assignment is good if it respects three rules, which consist of the following:

- two or more machine states that have the same next state should be given adjacent binary codes;

- two or more states that are the next states of the same state should be given adjacent binary codes.
- the first rule should have precedence over the second.

State adjacency means that the states appear next to each other in the mapped representation. In other terms, the combination assigned to the states should differ in only one position;

Now we concentrate on the assignment encoding and the fitness function. Given two different state assignments, the fitness function allows us to decide which is fitter.

9.6.1 State Assignment Encoding

In this case, a q-individual represents a state assignment. Each q-individual consists of an array of $2 \times N \lceil (\log_2 N) \rceil$ entries, wherein each set of $2 \times \lceil \log_2 N \rceil$ entries are the qubits associated to a single machine state. For instance, Figure 9.5 represents a q-individual and a possible assignment for a machine with 4 states obtained after the observation of the qubits.

S_0	S_1	S_2	S_3
α_1^0 α_2^0	α_1^1 α_2^1	α_1^2 α_2^2	α_1^3 α_2^3
β_1^0 β_2^0	β_1^1 β_2^1	β_1^2 β_2^2	β_1^3 β_2^3
1 1	0 1	0 0	1 0

Fig. 9.5 Example of state assignment encoding

Note that when an observation occurs, one code might be used to represent two or more distinct states. Such a state assignment is not possible. In order to discourage the selection of such an assignment, we apply a penalty every time a code is used more than once within the considered assignment. This will be further discussed in the next section where the fitness function is described.

9.6.2 Q-Gate for State Assignment

To drive the individuals towards better solutions, a q-gate is used as a variation operator of the quantum-inspired evolutionary algorithm presented at this chapter. After an update operation, the qubit must always satisfy the normalization condition $|\alpha'|^2 + |\beta'|^2 = 1$, where α' and β' are the amplitudes of the updated qubit.

Initially, each q-individual represents all possible states with the same probability. As the probability of every qubit approaches either 1 or 0 as a result of many applications of the q-gate, the q-individual converges to a single state and the diversity property disappears gradually. By this mechanism, the quantum-inspired evolutionary algorithm can treat the balance between exploration and exploitation [23]. The q-gate used is inspired by a quantum rotation gate. This is defined in Equation 9.15.

$$\begin{bmatrix} \alpha' \\ \beta' \end{bmatrix} = \begin{bmatrix} cos(\Delta\theta) & -sin(\Delta\theta) \\ sin(\Delta\theta) & cos(\Delta\theta) \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \tag{9.15}$$

where $\Delta\theta$ is the rotation angle of each qubit towards either of the states 0 or 1, depending on the amplitude signs. The angle $\Delta\theta$ should be adjusted according to problem at hand.

The value of the angle $\Delta\theta$ can be selected from the Table 9.4, where $f(\mathtt{s}_i^g)$ and $f(\mathtt{b}_i^g)$ are the fitness values of $\mathtt{s}_i^g$ and $\mathtt{b}_i^g$, and s_j and b_j are the jth bits of the observed solutions $\mathtt{s}_i^g$ and the best solutions $\mathtt{b}_i^g$, respectively. The rotation gate allows changing the amplitudes of the considered qubit, as follows:

1. If s_j and b_j are 0 and 1, respectively, and if $f(\mathtt{s}_i^g) \geq f(\mathtt{b}_i^g)$ is false then:
   - if the qubit is located in the first or third quadrant as defined in Equation 9.8, $\Delta\theta = \theta_3$ is set to a positive value to increase the probability of the state $|1\rangle$;
   - if the qubit is located in the second or fourth quadrant, $\Delta\theta = -\theta_3$ should be used to increase the probability of the state $|1\rangle$.
2. If s_j and b_j are 1 and 0, respectively, and if $f(\mathtt{s}_i^g) \geq f(\mathtt{b}_i^g)$ is false:
   - if the qubit is located in the first or third quadrant, $\Delta\theta = \theta_5$ is set to a negative value to increase the probability of the state $|0\rangle$;
   - if the qubit is located in the second or fourth quadrant, $\Delta\theta = -\theta_5$ should be used to increase the probability of the state $|0\rangle$.

Table 9.4 Look-up table of $\Delta\theta$

s_j	b_j	$f(\mathtt{s}_i^g) \geq f(\mathtt{b}_i^g)$	$\Delta\theta$
0	0	false	θ_1
0	0	true	θ_2
0	1	false	θ_3
0	1	true	θ_4
1	0	false	θ_5
1	0	true	θ_6
1	1	false	θ_7
1	1	true	θ_8

When it is ambiguous to select a positive or negative number for the angle parameter, we set its value to zero as recommended in [23]. The magnitude of $\Delta\theta$ has an effect on the speed of convergence. If it is too big, the search grid of the algorithm would be large and the solutions may diverge or converge prematurely to a local optimum. If it is too small, the search grid of the algorithm would be small and the algorithm may stagnate. Hence, the magnitude of $\Delta\theta$ varies and the corresponding values depend on the application problem. In the state assignment problem, we experimentally discovered that these values should be set as follows: $\theta_1 = \theta_2 = \theta_4 = \theta_6 = \theta_7 = \theta_8 = 0$, $\theta_3 = 0.05\pi$, and $\theta_5 = -0.05\pi$.

9.6.3 State Assignment Fitness

This step of the quantum-inspired evolutionary algorithm evaluates the fitness of each binary solutions obtained from the observation of the states of the q-individuals. The fitness evaluation of state assignments is performed with respect to the rules of Armstrong [2] and Humphrey [3]:

- how much a given state assignment adheres to the first rule, i.e. how many states in the assignment, which have the same next state but have no adjacent state codes;
- how much a given state in the assignment adheres to the second rule, i.e. how many states in the assignment, which are the next states of the same state but have no adjacent state codes.

In order to efficiently compute the fitness of a given state assignment, we use an $N \times N$ *adjacency matrix*, wherein N is the number of the machine states. The triangular bottom part of the matrix holds the expected adjacency of the states with respect to the first rule while the triangular top part of it holds the expected adjacency of the states with respect to the second rule. The matrix entries are calculated as described in Equation 9.16, wherein *AM* stands for the *A*djacency *M*atrix, functions $next(\sigma)$ and $prev(\sigma)$ yield the set of states that are next and previous to state σ, respectively. For instance, the 4×4 adjacency matrix for the state machine presented in Table tab:estados is shown in Figure 9.6.

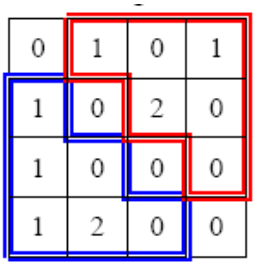


Fig. 9.6 Example of adjacency matrix

$$AM_{i,j} = \begin{cases} \#(next(q_i) \cup next(q_j)) & \text{If } i > j \\ \#(prev(q_i) \cup prev(q_j)) & \text{If } i < j \\ 0 & \text{If } i = j \end{cases} \tag{9.16}$$

Using the adjacency matrix AM as defined in Equation 9.16, the fitness function applies a penalty of 2 or 1, every time the first or second rule are broken, respectively. The penalty of breaking first rule is higher than that associated with the second rule to maintain the higher priority of the former over the latter. Equation 9.17 shows the details of the fitness function applied to a state assignment σ, wherein function $na(q,p)$ returns 0 if the codes representing states q and p are adjacent and 1 otherwise. Note that if assignment σ associates two distinct states to the same binary code, the σ is penalized by adding the constant ψ to the corresponding fitness value.

$$f(\sigma) = \sum_{i \neq j \ \& \ \sigma_i = \sigma_j} \psi + \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} (AM_{i,j} + 2 \times AM_{j,i}) \times na(\sigma_i, \sigma_j) \tag{9.17}$$

For instance, considering the state machine whose adjacency matrix is described in Figure 9.6, the state assignment $\{s_0 \equiv 00, s_1 \equiv 10, s_2 \equiv 01, s_3 \equiv 11\}$ has a fitness of 5 as the codes of states s_0 and s_3 are not adjacent but $AM_{0,3} = AM_{3,0} = 1$ and the codes of states s_1 and s_2 are not adjacent but $AM_{1,2} = 2$ while the assignments $\{s_0 \equiv 00, s_1 \equiv 11, s_2 \equiv 01, s_3 \equiv 10\}$ has a fitness of 3 as the codes of states s_0 and s_1 are not adjacent but $AM_{0,1} = AM_{1,0} = 1$.

The objective of the quantum-inspired evolutionary algorithm is to find the assignment that minimizes the fitness function as described in Equation 9.17. Assignments with fitness 0 satisfy all the adjacency constraints. Note that such an assignment may not exist for some state machines.

9.7 Logic Synthesis with QIGA

Exploiting the quantum-inspired evolutionary algorithm, we can automatically generate novel control logic circuits that are reduced with respect to area and time requirements. The allowed gates are NOT, AND, OR, XOR, NAND, XNOR and WIRE, as shown in Table 9.5. The last row represents a physical wire and thus, the absence of a gate.

Table 9.5 Gate name, gate code, gate-equivalent and average propagation delay (ns)

Name	Code	Area	Delay
NOT	000	1	0.0625
AND	001	2	0.2090
OR	010	2	0.2160
XOR	011	3	0.2120
NAND	100	1	0.1300
NOR	101	1	0.1560
XNOR	110	3	0.2110
WIRE	111	0	0.0000

9.7.1 Circuit Codification

We encode circuit designs using a matrix of cells that may be interconnected. A cell may or may not be involved in the circuit schematics and consists of two inputs, a logical gate and a single output. A cell draws its input signals from the outputs of the gates of the previous column. The cells located in the first column draw their inputs from the circuit global input signals. Each cell is encoded with a number of qubits, enough to represent the allowed gates and the signals that may be connected in each input of the cell gate. Note that the total number of qubits may vary depending on the number of outputs of the previous column and the number of primary inputs in the case of the first column [27]. An example of a matrix of cells with respect to this encoding is given in Figure 9.7.

For instance, the first part of Figure 9.8 represents a cell encoding and a possible observation of the qubits states while the second part indicates the correspondent circuit encoded by this cell, that is composed by an *AND* gate with its input A and B connected to the first and third element of its previous column.

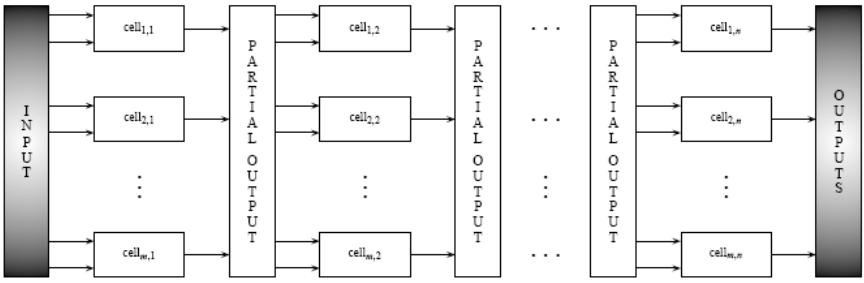


Fig. 9.7 Circuit representation

	Gate			Input A		Input B	
Cell	α_1 β_1	α_2 β_2	α_3 β_3	α_4 β_4	α_5 β_5	α_6 β_6	α_7 β_7
Observation	0	0	1	0	0	1	0

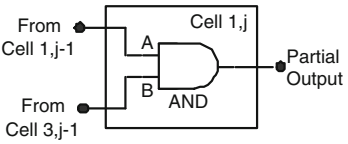


Fig. 9.8 Example of a cell considering that it has 4 outputs

When the observation of the qubits that define the gate yields 111, i.e. WIRE, then the signal connected to the cell's *A* input appears in the partial output of the cell. When the number of partial outputs of a column or the global inputs are not a power of 2, some of them are repeated in order to avoid that a cell be mapped to an inexistent input signal. The circuit primary output signals are the output signals of the cells in the last column of the matrix. If the number of global outputs are less than the number of cells in the last column, then some of the output signal are not used in the evolutionary process.

The power of the quantum-inspired representation can be evidenced in the drawing of Figure 9.9, which shows that all possible circuits can be represented with only one q-individual in a probabilistic way, as explained in the Section 9.5.1.

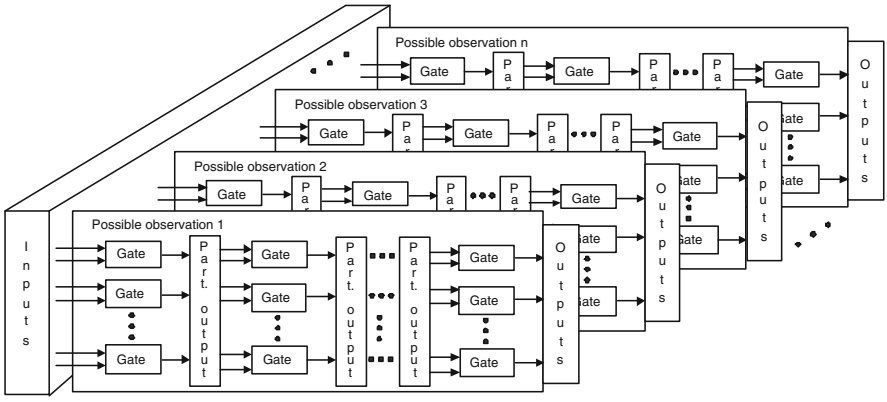


Fig. 9.9 Power of the quantum-inspired representation of an encoded circuit

The number of q-individual included in the population (population size) as well as the number of cells per q-individual are the parameters that should be adjusted considering the state machine complexity. The complexity depends on the number of inputs, outputs, states and number of states transitions of the machine.

9.7.2 Logic Fitness

This step of the quantum-inspired evolutionary algorithm evaluates the fitness of each binary solutions obtained from the observation of the states of the q-individuals. To evaluate the fitness of each solution, some constraints were considered: First of all, the evolved specification must obey the input/output behavior, which is given in a tabular form of the expected results given the inputs. This is the truth table of the expected circuit. Secondly, the circuit must have a reduced size. This constraint allows us to yield compact digital circuits. Finally, the circuit must also reduce the signal propagation delay. This allows us to reduce the response time and so discover efficient circuits.

We estimate the necessary area for a given circuit using the concept of gate-equivalent. This is the basic unit of measure for digital circuit complexity [10]. It is based upon the number of logic gates that should be interconnected to perform the same input/output behavior. This measure is more accurate that the simple number of gates [10].

When the input to an electronic gate changes, there is a finite time delay before the change in input is seen at the output terminal. This is called the propagation delay of the gate and it differs from one gate to another. We estimate the performance of a given circuit using the worst-case delay path from input to output. The number of gate-equivalent and an average propagation delay for each kind of gate were taken from [10].

Let C be a digital circuit that uses a subset or the complete set of allowed gates. The fitness function, which allows us to determine how much an evolved circuit adheres to the specified constraints, is given in Equation 9.18, wherein function $Soundness(C)$ returns the Hamming distance to evaluate the functionality of circuit C with respect to the input/output expected behavior, $Gates(C)$ returns the circuit gates equivalent and function $Delay(C)$ returns the propagation delay of the circuit C based. Parameters Ω_1 and Ω_2 are the weighting coefficients that allow us to consider both area and response time to evaluate the performance of an evolved circuit. For implementation issue, we minimize the fitness function of Equation 9.18, considering the normalized values of $Area(C)$ and $Delay(C)$ functions. The values of Ω_1 and Ω_2 are set to 0.6 and 0.4, respectively.

$$Fitness(C) = Soundness(C) + \Omega_1 \times Area(C) + \Omega_2 \times Delay(C), \tag{9.18}$$

where the objective of QIGA is the minimization of this function.

The Hamming distance is an non-negative integer that is proportional to the number of errors that result from the comparison between the output of the evolved circuit and those expected for each of the possible combination of the input

signals. Function $Soundness(C)$ is in Equation 9.19. Note that this definition sums up a penalty ψ for each error and so the total value is proportional to the number of output signal that are different from the expected ones.

$$Soundness(C) = \sum_{i=1}^{p} \left| y_j - x_j \right| \times \psi \tag{9.19}$$

wherein p is the number of possible combinations of the input signals, $\left| y_j - x_j \right|$ is the difference between the output signals of the evolved circuit and the expected ones, i.e. x_j e y_j respectively and ψ is a constant penalty for a single error. Note that if $Soundness(C) > 0$ then the circuit does not implement the desired behavior correctly and therefore, this is considered as a penalty for the individuals that encode circuit C.

Function $Area(C)$ returns the necessary hardware area to implement circuit C, which is evaluated using the number of gate-equivalent used. Let C be a circuit whose geometry is represented by a matrix $n \times m$. Recall that each cell $c_{i,j}$ of the circuit is formed by the gate type p together with the two inputs e_a e e_b. Function $Area(C)$ is defined in Equation 9.20. This definition is expressed using a recursive function $Area_{i,j}$, which allows us to compute the required area by the portion of circuit C that produces the output of the gate at cell $c_{i,j}$. This function is defined in Equation 9.21. Note that the area corresponding to the shared gates must only be counted once. For this purpose, a Boolean matrix $V : n \times m$ whose entry $V_{i,j}$ is updated when the gate of cell $c_{i,j}$ has been visited. In Equation 9.21, $GE_{c_{i,j}^{p}}$ represents the number of gate-equivalent for gate p at cell $c_{i,j}$ and $c_{i,j}^{ex}$ represents one of the inputs of that gate.

$$Area(C) = \sum_{i=1}^{s} Area_{i,m}, \tag{9.20}$$

wherein s is the number of output signals of C with $s \leq m$.

$$Area_{i,j} = \begin{cases} GE_{c_{i,j}^{p}} & \text{If } j = 1 \\ GE_{c_{i,j}^{p}} + \sum\limits_{x \in \{a,b\} e \neg V_{c_{i,j}^{ex}, j-1}} \left(Area_{c_{i,j}^{ex}, j-1} \right) & \text{If } j \in [2, m] \end{cases} \tag{9.21}$$

When the input of a given gate switches from 0 to 1 or 1 to 0, there exists a finite delay before the change is perceived at the output terminal of that gate. This delay is called *propagation delay* and it depends on the type of the gate, the technology used to implement it and the load factor that is put on the output terminal of this gate. The values of the gate propagation delays for CMOS technology are given in Table 9.6, where L represents the total load on the gate output. This delay does also depend on the signal transition, i.e. the propagation delay of a gate are different when a positive (t_{pLH}) or negative (t_{pHL}) transition occurs. The total load for a given gate is based on a basic load unit defined for each gate family. The total load is then a sum of all

Table 9.6 Gates, respective delays, load factor and area

Gate Type	Propagation Delay		Load factor (load unit)	Area (gate-equivalent)
	t_{pLH}(ns)	t_{pHL}(ns)		
NOT	$0.02+0.038L$	$0.05+0.017L$	1.0	1
AND	$0.15+0.037L$	$0.16+0.017L$	1.0	2
OR	$0.12+0.037L$	$0.20+0.019L$	1.0	2
XOR	$0.30+0.036L$	$0.30+0.021L$	1.1	3
NAND	$0.05+0.038L$	$0.08+0.027L$	1.0	1
NOR	$0.06+0.075L$	$0.07+0.016L$	1.0	1
XNOR	$0.30+0.036L$	$0.30+0.021L$	1.1	3

the load factor of every gate whose input signals is drawn from the output signal of the considered gate.

Let C be a circuit whose geometry is represented by a matrix $n \times m$. The delay introduced by cell $c_{i,j}$ is defined as in Equation 9.22, wherein $\alpha_{c_{i,j}^p}$ represents the average of the intrinsic delay of gate p at cell $c_{i,j}$. The average delay of the gate when the total load is 0 and $\beta_{c_{i,j}^p}$ the average delay due to the fanout of output signal of gate p of that cell. Table 9.7 shows the values of α and β for each of the used gates.

$$\tau gate_{i,j} = \alpha_{c_{i,j}^p} + \beta_{c_{i,j}^p} \times \left(\sum_{\substack{k \in [1,n], x \in \{a,b\} | \\ c_{k,j+1}^{e_x} = i}} factor(c_{k,j+1}^p) \right) \tag{9.22}$$

Table 9.7 Values of α and β for the gates used by QIGA

Gate Type	α	β
NOT	0.035	0.0465
AND	0.155	0.0270
OR	0.160	0.0280
XOR	0.300	0.0285
NAND	0.065	0.0325
NOR	0.065	0.0455
XNOR	0.300	0.0285

The propagation delay of a circuit is defined by the delay of its critical path. Considering all possible paths in a circuit, the critical path is the one that yields the largest delay. The propagation delay of a given path of a circuit is defined by the

sum of delay of each of the gates that is traversed by the signal from the input until the output of the circuit, as defined formally in Equation 9.23.

$$\tau path_{i,j} = \begin{cases} \tau gate_{i,j} & \text{If } j = 1 \\ \tau gate_{i,j} + \max\limits_{x \in \{a,b\}} \left(\tau path_{c_{i,j}^{ex},j-1} \right) & \text{If } j \in [2,m] \end{cases} \tag{9.23}$$

For a circuit of $s \leq n$ output signals, the propagation delay is determined by the largest delay among those imposed by all the paths of the circuit that reach the s gates located at the last column of the matrix representing the circuit. Function $Delay(C)$ is then defined as in Equation 9.24.

$$Delay(C) = \max_{i \in [1,s]} \tau path_{i,m} \tag{9.24}$$

9.8 Performance Results

This section is divided into two main parts: the result evolved by QIGA for the state assignment problem and those obtained for the synthesis of the control logic. The FSMs used are well-known benchmarks for testing finite state machines [28].

9.8.1 State Assignments Results and Discussion

In this section, we compare the assignment evolved by the quantum-inspired evolutionary algorithm presented in this chapter to those yield by the genetic algorithms [12, 6] and to those obtained using the non-evolutionary assignment system called NOVA. Table 9.8 shows the best state assignments generated by the compared systems.

The graphs presented in Figure 9.10 – Figure 9.14 show the progress of the evolutionary process of the best assignment fitness together with the average fitness with respect to all individuals of the population for some of the state machines used in the comparison.

The results introduced in Table 9.8 are depicted in the charts of Figure 9.15 for the comparison of the gate number, Figure 9.16 for the comparison of the hardware area and Figure 9.17 for the comparison of the propagation delays.

In order to determine whether the results obtained by QIGA are significantly better than those obtained by the genetic algorithm and the NOVA™ synthesis tool, we performed a statistical test of significance. The most commonly used method of comparing proportions uses the χ^2-test [29]. This test makes it possible to determine whether the difference existing between two groups of data is significant or just a chance occurrence.

Table 9.8 Best state assignments found by the compared methods

FSM	Method	State Assignments
bbara	AG_1	[0,6,2,14,4,5,13,7,3,1]
	AG_2	[0,6,2,14,4,5,13,7,3,1]
	NOVA™	[9,0,2,13,3,8,15,5,4,1]
	QIGA	[4,5,1,9,13,12,14,15,7,6]
bbsse	AG_2	[0,4,10,5,12,13,11,14,15,8,9,2,6,7,3,1]
	NOVA™	[12,0,6,1,7,3,5,4,11,10,2,13,9,8,15,14]
	QIGA	[5,3,11,7,9,6,14,10,8,12,4,1,0,2,13,15]
dk14	AG_1	[5,7,1,3,6,0,4]
	AG_2	[0,4,2,1,5,7,3]
	NOVA™	[1,4,0,2,7,5,3]
	QIGA	[5,7,4,0,6,3,1]
dk16	AG_1	[12,8,1,27,13,28,14,29,0,16,26,9,2,4,3,10,11,17,24,5,18,7,21,25,6,20,19]
	NOVA™	[12,7,1,3,4,10,23,24,5,27,15,16,11,6,0,20,31,2,13,25,21,14,18,19,30,17,22]
	QIGA	[14,30,22,6,4,5,13,25,18,20,31,9,10,26,23,28,29,7,15,3,16,8,21,17,1,11,24]
donfile	AG_1	[0,12,9,1,6,7,2,14,11,,17,20,23,8,15,10,16,21,19,4,5,22,18,13,3]
	NOVA™	[12,14,13,5,23,7,15,31,10,8,29,25,28,6,3,2,4,0,30,21,9,17,12,1]
	QIGA	[7,6,23,31,26,27,15,14,13,5,10,4,22,30,12,8,11,9,18,19,2,0,3,1]
lion9	AG_2	[0,4,12,13,15,1,3,7,5]
	NOVA™	[2,0,4,6,7,5,3,1,11]
	QIGA	[11,9,3,1,2,0,8,10,14]
mod12	AG_1	[0,8,1,2,3,9,10,4,11,12,5,6]
	NOVA™	[0,15,1,14,2,13,3,12,4,11,5,10]
	QIGA	[15,7,6,14,10,2,3,1,5,13,9,11]
shiftreg	AG_1	[0,2,5,7,4,6,1,3]
	AG_2	[0,2,5,7,4,6,1,3]
	NOVA™	[0,4,2,6,3,7,1,5]
	QIGA	[4,0,2,6,5,1,3,7]
train11	AG_2	[0,8,2,9,13,12,4,7,5,3,1]
	NOVA™	[0,8,2,9,1,10,4,6,5,3,7]
	QIGA	[9,11,13,3,1,2,0,12,8,5,4]

For the sake of completeness, we explain briefly how the test works. χ^2-test determines the differences between the observed and expected measures. The observed values are the actual experimental results, whereas the expected ones refer to the hypothetical distribution based on the overall proportions between the two compared algorithms if these are alike. Let $\lambda_o^{(a,m,q)}$ and $\lambda_e^{(a,m,q)}$ be respectively the observed and expected value of objective q obtained when using algorithm a with machine state m. Note that $\lambda_e^{(a,m,q)}$ is computed as described in Equation 9.25.

$$\lambda_e^{(a,m,q)} = \frac{\sum\limits_{(x,z)\in A\times Q} \lambda_o^{(x,m)} \times \sum\limits_{y\in M} \lambda_o^{(a,y)}}{\sum\limits_{(x,y,z)\in A\times M\times Q} \lambda_o^{(x,y,x)}}, \tag{9.25}$$

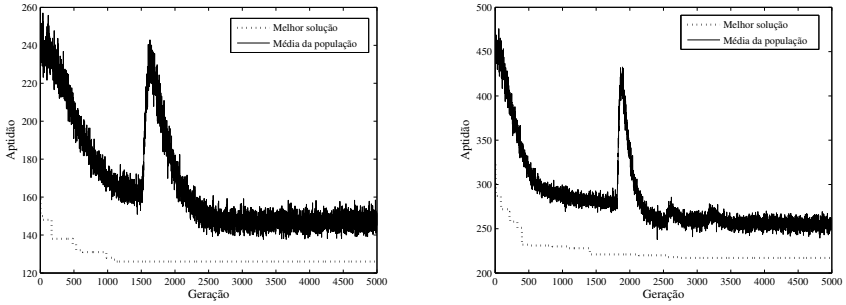


Fig. 9.10 Progress of the best solution fitness together with the average fitness for state machines *bbara* e *bbsse*

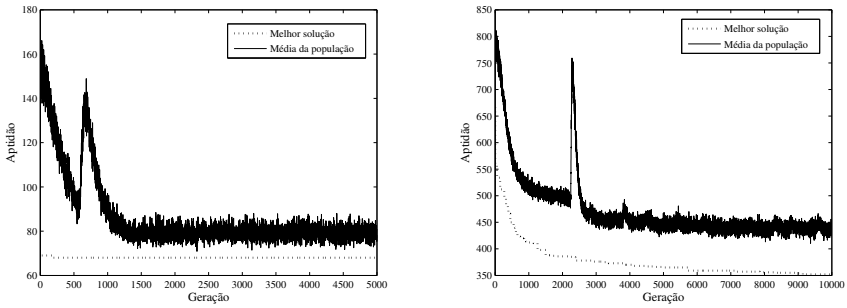


Fig. 9.11 Progress of the best solution fitness together with the average fitness for state machines *dk14* e *dk16*

wherein A={QIGA, NOVA$^{\text{TM}}$}, A={QIGA, AG$_2$} or A={QIGA, AG$_3$}, $M \subseteq \{bbara, bbsse, dk14, dk16, donfile, lion9, modulo12, shiftreg, train11\}$ and Q={*#gate*, *area*, *time*}. The χ^2-test is based on the value of χ^2 computed as in Equation 9.26, wherein set $a = \{\text{QIGA} \times \text{NOVA}^{\text{TM}}, \text{QIGA} \times \text{AG}_1 \text{ e QIGA} \times \text{AG}_2\}$ and $q = \{\#gate, area, time\}$.

$$\chi^2 = \sum_{(a,m,q) \in A \times M \times Q} \frac{\left(\lambda_o^{(a,m,q)} - \lambda_e^{(a,m,q)}\right)^2}{\lambda_e^{(a,m,q)}}. \tag{9.26}$$

The computed values for χ^2 for each of the comparisons are given in Table 9.9. The use of the χ^2-test is recommended when the proportions are small. Therefore, the time quantities were converted to 0.1 ns instead of 1 ns, thus avoiding the limitation imposed for the usage of the test.

The critical value of χ^2 is 0.05 (i.e. 95% of confidence) and considered the limit to assume the tested hypothesis. The degree of freedom depends on the amount

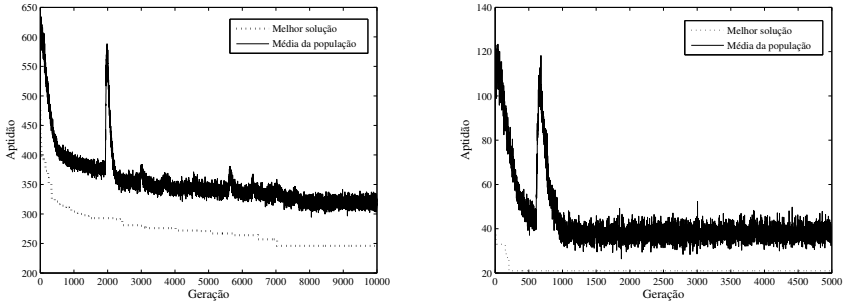


Fig. 9.12 Progress of the best solution fitness together with the average fitness for state machines *donfile* e *lion9*

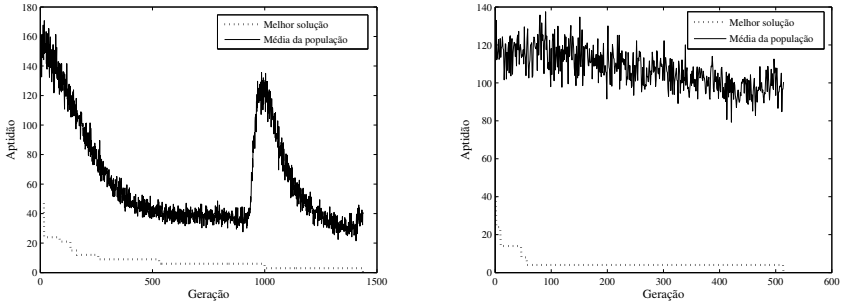


Fig. 9.13 Progress of the best solution fitness together with the average fitness for state machines *modulo12* e *shiftreg*

Table 9.9 Degree of freedom, computed χ^2, critical χ^2 for the confidence level of 99,5% e the degree of confidence obtained for the considered comparisons

Comparison	Degree of freedom	χ^2	Critical value	Confidence level
QIGA $\times$ NOVA^{TM}	40	73,302	66,766	>99,5%
QIGA $\times$ AG$_1$	25	68,281	46,928	>99,5%
QIGA $\times$ AG$_2$	30	64,740	53,672	>99,5%

of results used to compute χ^2. Assuming that the results are organized in a two-dimensional array of r rows and c columns, the degree of freedom is defined by $(r-1)\times(c-1)$. In this comparison, the number of rows coincides with that of state machines used as benchmarks and the number of columns is 6: one for each pair of objective/algorithm (we are considering 3 objectives and 2 algorithms in each comparison).

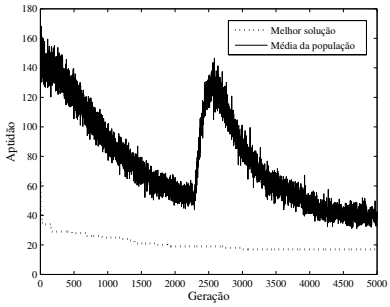


Fig. 9.14 Progress of the best solution fitness together with the average fitness for state machines *train11*

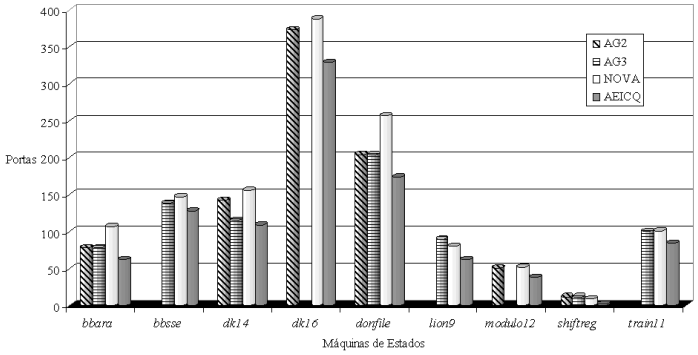


Fig. 9.15 Logic control comparison in terms of gates used

In the case of the comparison QIGA $\times$ NOVA^{TM}, all 9 state machines listed in Table 9.8 are considered, while in the case of the other two comparisons, i.e. QIGA $\times$ AG_1 and QIGA $\times$ AG_2, only some of the machines, 6 and 7 respectively, are used taking into account the results availability. At the light of the statistical analysis, we can conclude that QIGA performs significantly better than NOVA^{TM}, AG_1 e AG_2.

For all the simulations, we used a population of 50 q-individuals. However, we observed that for some state machines, such as *shiftreg* and *lion9*, the best solution was obtainable with a population of a single q-individual. Nevertheless, in this last case, the number of runs that reached the best result shrunk considerably. For instance, during the evolution of *shiftreg*, the global optimum was reached em all the runs when the population size was of 50 q-individuals while with a population of 1 q-individuals, this was the case for only in 50% of the runs. During the performed simulations, it was also possible to observe that QIGA was very robust with respect to the choice of the angle magnitudes θ_3 and θ_5 within the spectrum suggested in [23].

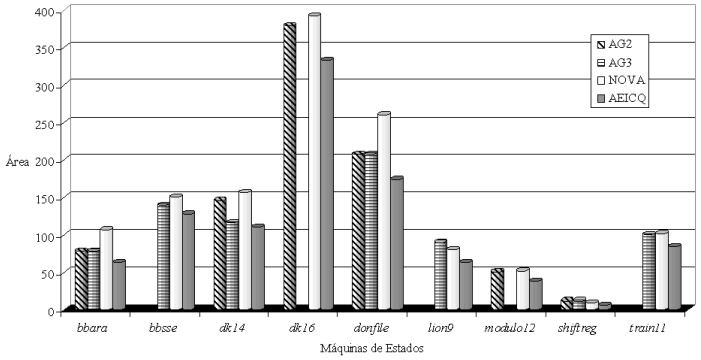


Fig. 9.16 Logic control comparison in terms of hardware area required

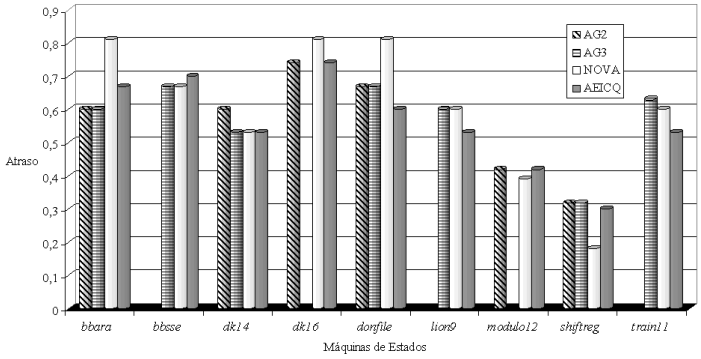
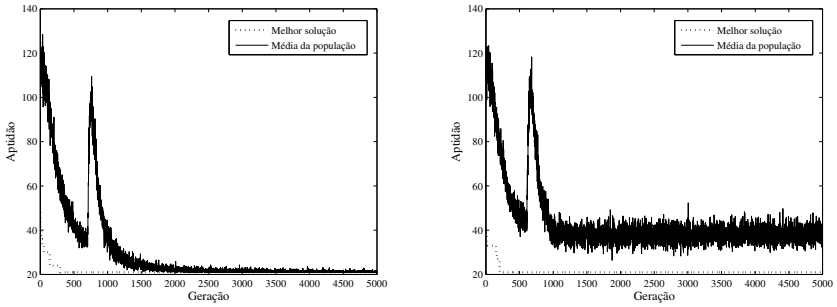


Fig. 9.17 Logic control comparison in terms of propagation delay imposed

The impact of the control phase of the probability amplitudes of the qubits, first contributed in QIGA, can be depicted in Figure 9.18. Figure 9.18–(a) shows that when the control is not imposed and the quantum-inspired algorithm does not evolve any new better solution, the average fitness of the population at hand gets very close to the fitness of the best q-individual, which has been yield so far. This happens due to the fact that the probabilities of the quantum states would practically be 100%, which would, in consequence lead to the measurement of the same solution in all generations. In contrast with this, Figure 9.18–(b) shows that the average of the probabilities is kept clear from the best solution. This is, actually, due to the control of the probability amplitudes of the qubits, which thus allows new solutions to be yield by the evolutionary process. This control step allows us to maintain a better diversity within the population individual and hopefully would accelerate the convergence of the optimization process.



(a) without control of the probability amplitudes (b) with control of the probability amplitudes

Fig. 9.18 Impact of the control phase of the probability amplitudes of the qubits

The impact of the global migration step can be viewed in Figure 9.19. In this step, the best solutions in $B(g)$, which are used in the update operation of the qubits, are all replaced by the best solution b. This substitution introduces a change in the population in the attempt to further improve its diversity. The picks, highlighted in the graphics of Figure 9.19, indicate three moments that allow for a clear observation of the effect caused by the global migration step on the average fitness of the population. The showed picks appear whenever 400 generations pass by without yielding a new better solution. Hence, as the control phase of the probability amplitudes of the qubits, this operation of global migration permits an remarkable improvement of the population diversity and thus leading to avoiding local minimums.

9.8.2 Logic Synthesis Results and Discussion

Table 9.10 shows the characteristics of the circuits that were synthesized using genetic programming (GP) [7, 8], genetic algorithms (GA) [6] and the ABC synthesis tool [9]. Table 9.11 shows the characteristics of the best circuit evolved by QIGA for each of the used machines.

The results listed in Table 9.11 and Table 9.10 are depicted as charts in Figure 9.20 for gate number comparison, Figure 9.21 for area comparison and Figure 9.22 for delay comparison.

The graphs presented in Figure 9.23 – Figure 9.27 show the progress of the evolutionary process of the best circuit fitness together with the average fitness with respect to all individuals in the population for some of the state machines used in the comparison.

As before, and in order to determine whether the results obtained by QIGA are significantly better than those obtained by GP [7, 8] and ABC [9]. The computed χ^2 for these comparisons are presented in Table 9.12.

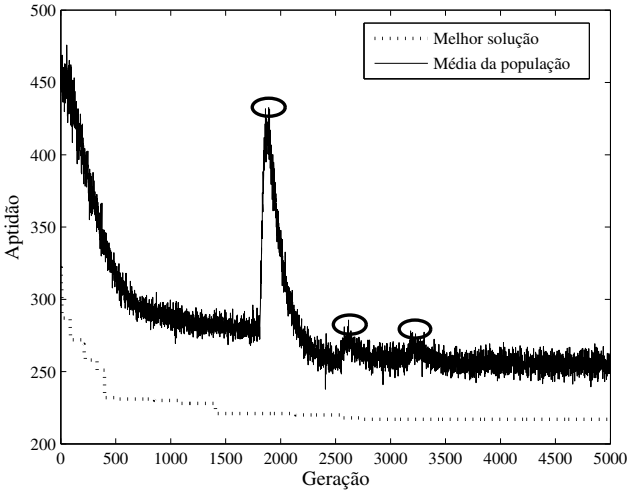


Fig. 9.19 Impact of the global migration

Table 9.10 Characteristics of evolved circuits by GP, GA, ABC

State	GP			GA			ABC		
machine	#Gates	Area	Delay	#Gates	Area	Delay	#Gates	Area	Delay
bbara	–	–	–	60	–	–	62	63	0,67
bbsse	–	–	–	–	–	–	128	128	0,70
bbtas	–	–	–	19	–	–	24	24	0,32
dk14	–	–	–	–	–	–	109	110	0,53
dk15	–	–	–	53	–	–	92	92	0,46
dk16									
dk27	–	–	–	16	–	–	25	25	0,32
dk512	–	–	–	47	–	–	63	63	0,46
donfile	–	–	–	–	–	–	174	174	0,60
lion9	21	39	0,70	50	–	–	62	63	0,53
modulo12	–	–	–	–	–	–	38	38	0,42
shiftreg	5	14	0,60	8	–	–	2	6	0,30
tav	–	–	–	26	–	–	31	31	0,46
train11	22	43	0,56	–	–	–	85	85	0,53

The property of *scalability* is of paramount importance for any kind of project and in electronic circuits projects, in particular [30, 31]. According to [32], scalability in evolutionary electronics can be approached in two different ways that are somehow related. The first focuses on the scalability of the individuals that represent electronic circuits. It was established that if no restriction on how basic components are connected is imposed then the size of the individuals will grow in the order of

Table 9.11 QIGA experimental results

State machine	#Gates	Area	Delay
bbara	**54**	78	0.88
bbtas	21	27	0.73
dk15	65	109	0.92
dk27	**15**	26	0.43
dk512	**47**	78	0.84
lion9	**20**	**29**	**0.52**
modulo12	**19**	**34**	0.56
shiftreg	**2**	**2**	**0.04**
tav	**26**	**24**	**0.32**
train11	25	**37**	**0.52**

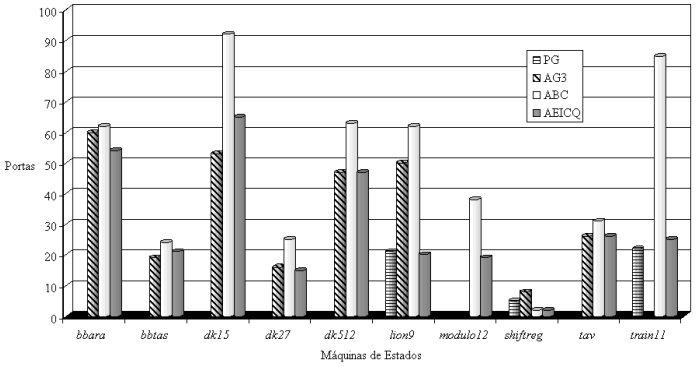


Fig. 9.20 Comparison of control logic for number of gates

Table 9.12 Degree of freedom, computed χ^2, critical χ^2 for the confidence level of 99,5% e the degree of confidence obtained for the considered comparisons

Comparison	Degree of freedom	χ^2	Critical value	Confidence level
QIGA $\times$ PG	10	18,898	18,31	>95,0%
QIGA $\times$ ABC	45	97,823	69,96	>99,5%

$O(n^2)$, wherein n is the number of functional components. However, if the connectivity is restricted to a local neighborhood in the proximity of the component, the order of $O(n)$ can be achieved. Note that the latter restricts the circuit that can be evolved. The second way to handle scalability in evolutionary circuits is to reduce, to a minimum, the complexity of the evolutionary process. Nowadays, scalability is the main problem that faces the extensive use of evolutionary electronics.

The problem of scalability was noted in many other works that used the evolutionary process to yield circuits [6, 33] and this was also the case for this work. For a sate machine of reduced complexity such as *shiftreg*, it is possible to encode the circuit with a 4×3 geometry composed of 108 qubits. In this case, the population

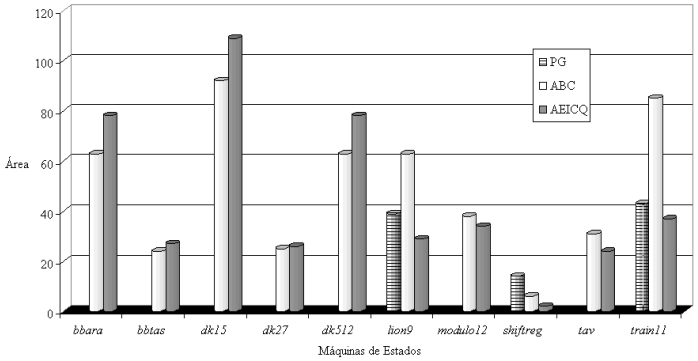


Fig. 9.21 Comparison of control logic for required area

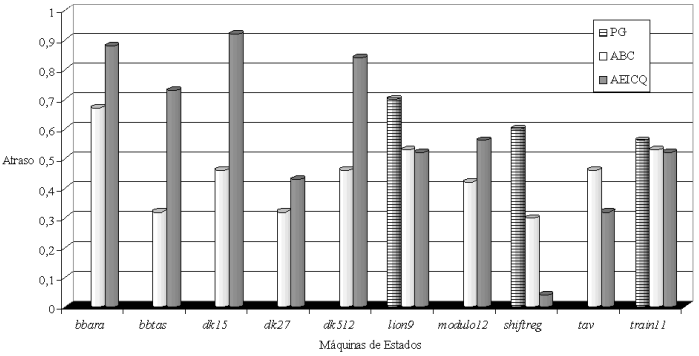


Fig. 9.22 Comparison of control logic for imposed delay

of 20 q-individuals is enough to yield optimal circuits. However, for more complex state machines, such as *bbara*, it is necessary to use a geometry of 32×5, composed of 2080 qubits. In this case, the search space becomes extremely large, dictating imperatively an increase of the population size, which in turn leads to a considerable increase of the average execution time. This time is about 3 minutes in the case of the *shiftreg* state machine and around 5 hours in the case of *bbara*.

The increase of required time of the evolution of circuits brings together an extra difficulty, which is the adjustment of the parameters needed in QIGA. This makes it inviable to refine the parameters considering the characteristics of the state machine at hand. To overcome this obstacle, we adjusted the parameter setup based on the state machines *lion9* and *train11* and these parameters were used during the evolution of the control logic of the remaining state machines. Even so, the results obtained for these machines are satisfactory, proving once again the robustness of QIGA.

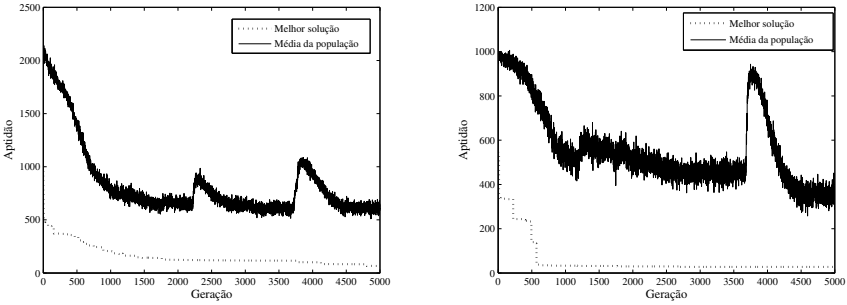


Fig. 9.23 Progress of the best solution fitness together with the average fitness for logic synthesis of state machines *bbara* e *bbtas*

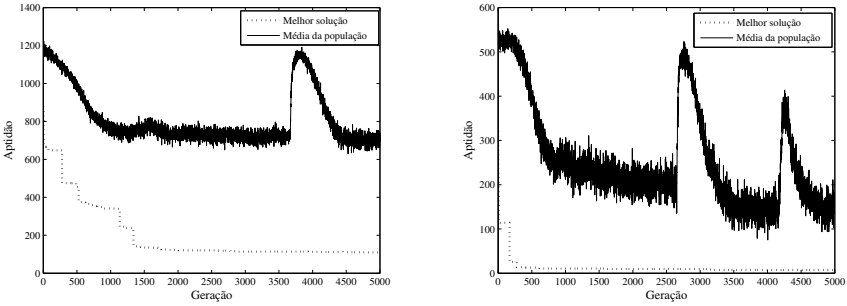


Fig. 9.24 Progress of the best solution fitness together with the average fitness for logic synthesis of state machines *dk15* e *dk27*

The results shown in this section suggest that QIGA is a tool of great potential to be used in automatic synthesis of electronic circuits. The evolved circuits show similar and some time better characteristics than those obtained by ABC, which is a well-known as a powerful tool for logic synthesis.

9.9 Summary

In this chapter we studied the application of quantum-inspired evolutionary methodology to solve two hard problems: the state assignment and the automatic synthesis of the control logic in the design process of synchronous finite state machines. We compared both the state assignment and the circuits evolved by the proposed algorithm QIGA for machines of different sizes and complexity with the results obtained by other method. QIGA almost always obtains better results. This proves that quantum-inspired evolutionary computation is very robust and leads to good results

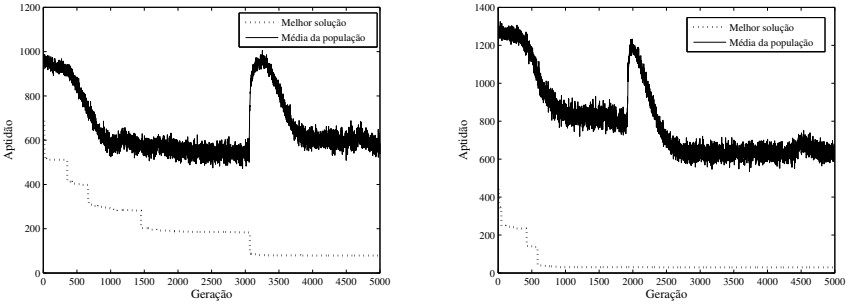


Fig. 9.25 Progress of the best solution fitness together with the average fitness for logic synthesis of state machines *dk512* e *lion9*

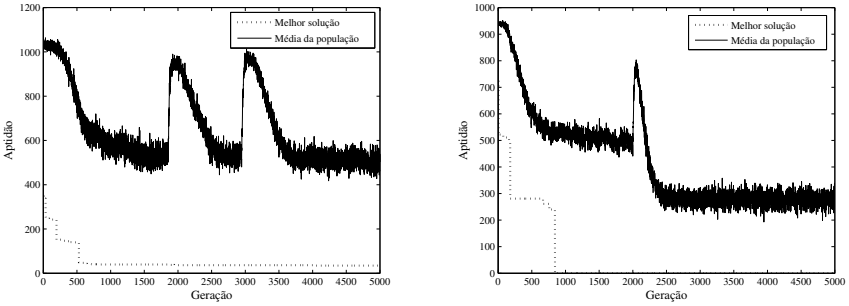


Fig. 9.26 Progress of the best solution fitness together with the average fitness for logic synthesis of state machines *modulo12* e *shiftreg*

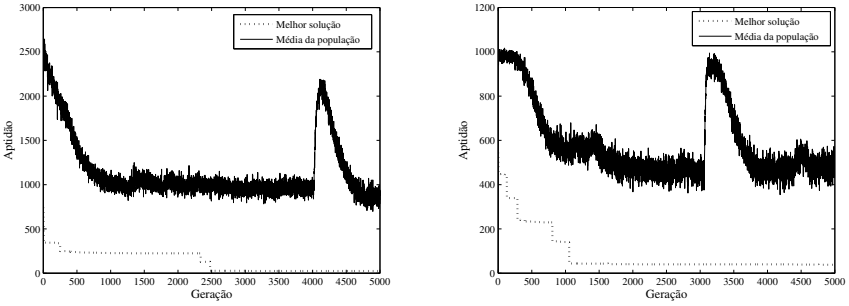


Fig. 9.27 Progress of the best solution fitness together with the average fitness for logic synthesis of state machines *tav* e *train11*

and therefore can be very profitable when embedded in automatic synthesis tools used in the design of digital systems.

Two main directions for future work emerges from this study. Regarding the state assignment problem, one can investigate the use of other heuristics other than or combined with Armstrong and Humphrey's [2, 3]. Regarding the logic synthesis, one can study the adaptation of QIGA so that it can evolve circuits at function level, instead of gate level as it is the case in this chapter. Another interesting investigation is the use of co-evolution technique in QIGA to accelerate further the evolutionary process. This allows one to catch up with the scalability problem.

References

1. Rhyne, V.T.: Fundamentals of digital systems design. In: Computer Applications in Electrical Engineering Series. Prentice-Hall (1973)
2. Armstrong, D.B.: A programmed algorithm for assigning internal codes to sequential machines. IRE Transactions on Electronic Computers EC-11(4), 466–472 (1962)
3. Humphrey, W.S.: Switching circuits with computer applications. McGraw-Hill, New York (1958)
4. Booth, T.L.: Sequential machines and automata theory. John Wiley & Sons, New York (1967)
5. Ali, B., Kalganova, T., Almaini, A.E.: Extrinsic evolution of finite state machine. In: Proc. of International Conference on Adaptive Computing in Design and Manufacture, pp. 157–168. Springer (2002)
6. Ali, B.: Evolutionary algorithms for synthesis and optimization of sequential logic circuits. Ph.D. Thesis, School of Engineering of Napier University, Edinburgh, UK (2003)
7. Nedjah, N., Mourelle, L.M.: Evolvable machines: theory and practice. STUD FUZZ, vol. 161. Springer, Heidelberg (2005)
8. Nedjah, N., Mourelle, L.M.: Mealy finite state machines: an evolutionary approach. International Journal of of Innovative Computing, Information and Control 2(4), 789–806 (2006)
9. ABC, A system for sequential synthesis and verification, Release 70930. In: Logic Synthesis and Verification Group, Berkeley (2005)
10. Ercegovac, M., Lang, T., Moreno, J.H.: Introduction to Digital Systems. John Wiley, USA (1998)
11. Hartmanis, J.: On the state assignment problem for sequential machines. IRE Transactions on Electronic Computers EC-10(2), 157–165 (1961)
12. Amaral, J.N., Tumer, K., Glosh, J.: Designing genetic algorithms for the state assignment problem. IEEE Transactions on Systems, Man, and Cybernetics 25(4), 686–694 (1995)
13. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proc. the Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE Computer Society Press (1994)
14. Lin, F.T.: An enhancement of quantum key distribution protocol with noise problem. International Journal of of Innovative Computing, Information and Control 4(5), 1043–1054 (2008)
15. Hey, T.: Quantum computing. Computing Control Engineering Journal 10(3), 105–112 (1999)
16. Dirac, P.A.M.: The principles of quantum mechanics, 4th edn. Oxford University Press (1958)

17. Zhang, G.: Novel quantum genetic algorithm and its applications. Frontiers of Electrical and Electronic Engineering in China 1(1), 31–36 (2006)
18. Narayanan, A.: Quantum computing for beginners. In: Proc. of the Congress on Evolutionary Computation, vol. 3, pp. 2231–2238. IEEE Press, Piscataway (1999)
19. Nedjah, N., Mourelle, L.M. (eds.): Swarm Intelligent Systems. SCI, vol. 26. Springer, Heidelberg (2006)
20. Uno, T., Katagiri, H., Kato, K.: An evolutionary multi-agent based search method for stackelberg solutions of bi-level facility location problems. International Journal of of Innovative Computing, Information and Control 4(5), 1033–1042 (2008)
21. Liu, C., Wang, Y.: A new evolutionary algorithm for multi-objective optimization problems. ICIC Express Letters 1(1), 93–98 (2007)
22. Zhang, X., Lu, Q., Wen, S., Wu, M., Wang, X.: A modified differential evolution for constrained optimization. ICIC Express Letters 2(2), 181–186 (2008)
23. Han, K.H., Kim, J.H.: Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE Transactions on Evolutionary Computation 6(6), 580–593 (2002)
24. Hinterding, R.: Representation, constraint satisfaction and the knapsack problem. In: Proc. of the Congress on Evolutionary Computation, vol. 2, pp. 1286–1292. IEEE Press, Piscataway (1999)
25. Akbarzadeh, M.R., Khorsand, A.R.: Quantum gate optimization in a meta-level genetic quantum algorithm. In: Proc. of IEEE International Conference on Systems, Man and Cybernetics, vol. 4, pp. 3055–3062. IEEE Press, Piscataway (2005)
26. Araujo, M.P.M., Nedjah, N., de Macedo Mourelle, L.: Optimised state assignment for fSMs using quantum inspired evolutionary algorithm. In: Hornby, G.S., Sekanina, L., Haddow, P.C. (eds.) ICES 2008. LNCS, vol. 5216, pp. 332–341. Springer, Heidelberg (2008)
27. Araujo, M.P.M., Nedjah, N., de Macedo Mourelle, L.: Logic synthesis for fSMs using quantum inspired evolution. In: Fyfe, C., Kim, D., Lee, S.-Y., Yin, H. (eds.) IDEAL 2008. LNCS, vol. 5326, pp. 32–39. Springer, Heidelberg (2008)
28. ACM/SIGDA, Collaborative Benchmarking and Experimental Algorithmic, North Carolina State University (2009), `http://www.cbl.ncsu.edu`
29. Diaconis, P., Efron, B.: Testing for independence in a two-way table: new interpretations of the chi-square statistic (with discussion). The Annals of Statistics 13, 845–913 (1985)
30. Higuchi, T.: Evolving hardware with genetic learning. In: Proc. of International Conference on Simulation Adaptive Behavior: A First Step Toward Building a Darwin Machine, pp. 417–424. MIT Press (1992)
31. Hemmi, H., Mizoguchi, J., Shimohara, K.: Development and evolution of hardware behaviors. In: Sanchez, E., Tomassini, M. (eds.) Towards Evolvable Hardware 1995. LNCS, vol. 1062, pp. 250–265. Springer, Heidelberg (1996)
32. Yao, X., Higuchi, T.: Promises and challenges of cvolvable hardware. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 29(1), 87–97 (1999)
33. Zebulum, R.S., Pacheco, M.A., Vellasco, M.M.: Evolutionary electronics: automatic design of electronic circuits and systems by genetic algorithms. CRC Press (2001)