Evolving digital circuits using particle swarm

Ganesh K. Venayagamoorthy
Missouri University of Science and Technology, ganeshv@mst.edu

Venu Gopal Gudise

Follow this and additional works at: http://scholarsmine.mst.edu/faculty_work

Part of the Electrical and Computer Engineering Commons

Recommended Citation
http://scholarsmine.mst.edu/faculty_work/1664

This Article - Conference proceedings is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Faculty Research & Creative Works by an authorized administrator of Scholars' Mine. For more information, please contact weaverje@mst.edu.
Evolving Digital Circuits Using Particle Swarm

Venu G. Gudise and Ganesh K. Venayagamoorthy
Dept. of Electrical and Computer Engineering
University of Missouri - Rolla, USA.
vugudise@umr.edu and gkven@ieee.org

Abstract — Particle swarm optimization (PSO) motivated by the social behavior of organisms is proposed for evolution of combinational logic circuits. Results are presented to show that PSO based evolution of digital circuits is equivalent to or even with better solutions (with minimum number of logic gates) than that of a human designer and other genetic algorithm (GA) based techniques. This PSO based approach converges faster than other approaches reported in literature using genetic algorithms and as a result the computational intensity involved in hardware evolution is reduced. Examples taken from the literature are used to evaluate the performance of the proposed PSO approach.

I. INTRODUCTION

There are many methods to design combinational circuits. Generally used methods are Karnaugh maps [1], Quine-McCluskey method [2]-[3] and Sasoos method [4]. The problem with the human designs is that they become cumbersome and problematic when the number of inputs, number of outputs, and complexity of the function increase. The intricacy of the combinational circuit depends on the number of gates in the circuit, and that of the gate depends on the number of inputs to the gate. For real world applications, combinational circuits with a minimum number of gates are preferred for simpler hardware realization.

In recent decades, algorithms that employ the principles of Darwinian evolution have been applied to the design of electronic systems [5]. Such work has become known as Evolvable Hardware (EHW), and this field has matured considerably in the last 3-6 years [6]. One of the goals of EHW has been to evolve complex designs, not achievable with the traditional design methods. This goal is still beyond our reach. EHW is built on software-reconfigurable logic devices, such as the programmable logic device (PLD) and the field programmable gate array (FPGA), whose architecture can be re-configured using an evolutionary technique(s). This reconfiguration can continue on-line to improve performance adaptively [7]. To design conventional hardware, it is necessary to know all the specifications of the hardware functions in advance. In contrast to this, EHW can configure itself without such specifications known in advance.

To date, hardware evolution has been reported using genetic algorithms (GAs) [7] and recently, many papers have proposed designing combinational circuits using genetic algorithms [8-14]. Genetic algorithms involve a population of chromosomes that are mutated and the chromosomes with greatest fitness survive over a number of generations. The convergence of a genetic algorithm can take a large number of generations and real-time implementation capability is still a gray area.

A new technique called the particle swarm optimization (PSO) that emerges and allies itself to evolutionary algorithms based on simulation of the behavior of a flock of birds or school of fish, has proven to have great potential for multi-objective optimization [15]. Swarm algorithms differ from evolutionary algorithms most importantly in both metaphorical explanation and how they work. What is new with the swarm algorithm is that the individuals (particles) persist over time, influencing one another's search of the problem space, unlike in the genetic algorithms where the weakest chromosomes are immediately discarded. The particles in PSO (similar to chromosomes in GA) are known to have fast convergence to local/global optimum position(s) over a small number of iterations [16].

The design of combinational digital circuits using particle swarm is presented and preliminary investigations show that PSO can evolve equally well as genetic algorithms and their variants [10]. The paper is organized as follows: Section II, a brief overview of the particle swarm optimization is given. Section III, evolving combinational circuits with PSO is described. Section IV, examples of swarm evolved combinational circuits are presented and compared against other techniques.

II. PARTICLE SWARM OPTIMIZATION

Particle swarm optimization is an evolutionary computation technique (a search method based on a natural system) developed by Kennedy and Eberhart [17] – [22]. PSO, like a generic algorithm (GA), is a population based optimization tool. However, unlike GA, PSO has no evolution operators such as crossover and mutation, and moreover, PSO has less parameters. PSO is an evolutionary algorithm that does not implement survival of the fittest, and unlike other evolutionary algorithms where an evolutionary operator is manipulated, the velocity is dynamically adjusted.

The system initially has a population of random solutions. Each potential solution, called a particle, is given a random velocity and is flown through the problem space. The particles have memory and each particle keeps track of its previous best position (called the pbest) and its corresponding fitness. There exist a number of pbest for the respective particles in the swarm and the particle with greatest fitness is called the global best (gbest) of the swarm. The basic concept of the PSO technique lies in accelerating each particle towards its pbest and gbest locations, with a random weighted acceleration at each time step and this is illustrated in Fig. 1, where $\beta^i$ is the current
position of a particle, \( P'^{k+1} \) is its modified position, \( V^{k+1} \) is its initial velocity, \( V^\text{mod} \) is its modified velocity, \( V^\text{pbest} \) is the velocity considering its pbest location and \( V^\text{gbest} \) is the velocity considering its gbest location.

![Diagram of a swarm particle's position](image)

\[ V^{k+1} = V^k + \frac{(P^k - X^k)}{V^k} \]  
\[ X^{k+1} = X^k + \frac{(P^k - X^k)}{V^k} \]

Fig. 1 Concept of a swarm particle’s position.

The main steps in the particle swarm optimization process are described as follows:

(i). Initialize a population of particles with random positions and velocities in \( d \) dimensions of the problem space and fly them.

(ii). Evaluate the fitness of each particle in the swarm.

(iii). For every iteration, compare each particle’s fitness with its previous best fitness (pbest) obtained. If the current value is better than pbest, then set pbest equal to the current value and the pbest location equal to the current location in the \( d \)-dimensional space.

(iv). Compare pbest of particles with each other and update the swarm global best location with the greatest fitness (gbest).

(v). Change the velocity and position of the particle according to equations (1) and (2) respectively. \( V_{id} \) and \( X_{id} \) represent the velocity and position of the \( i \)-th particle with \( d \) dimensions, respectively, \( rand1 \) and \( rand2 \) are two uniform random functions, and \( W \) is the inertia weight, which is chosen beforehand [22].

\[ V_{id} = W \cdot V_{id} + c1 \cdot rand1 \cdot (P_{id} - X_{id}) + c2 \cdot rand2 \cdot (P_{id} - X_{id}) \]  
\[ X_{id} = X_{id} + V_{id} \]

(vi). Repeat steps (ii) to (v) until convergence is reached based on some desired single or multiple criteria.

PSO has many parameters and these are described as follows: \( W \) called the inertia weight controls the exploration and exploitation of the search space because it dynamically adjusts velocity. Local minima are avoided by small local neighborhoods, but faster convergence is obtained by a larger global neighborhood, and in general a global neighborhood is preferred. Synchronous updates are more costly than the asynchronous updates. \( V_{\text{max}} \) is the maximum allowable velocity for the particles (i.e. in the case where the velocity of the particle exceeds \( V_{\text{max}} \), then it is limited to \( V_{\text{max}} \). Thus, resolution and fitness of search depends on \( V_{\text{max}} \). If \( V_{\text{max}} \) is too high, then particles will move beyond a good solution, and if \( V_{\text{max}} \) is too low, particles will be trapped in local minima. The constants \( c1 \) and \( c2 \) in (1) and (2), termed as cognition and social components, respectively, are the acceleration constants which changes the velocity of a particle towards pbest and gbest (generally, somewhere between pbest and gbest). The velocities of the particles determine the tension in the swarm. A swarm of particles can be used locally or globally in a search space. In the local version of the PSO, gbest is replaced with lbest and the entire process is the same.

III. EVOLVING COMBINATIONAL CIRCUITS

Particle swarm theory described above is used to evolve combinational logic circuits. The basic process of hardware evolution is illustrated in Fig. 2. The “desired” circuit refers to the circuit required to map 100 % exactly the outputs for corresponding inputs typically given by a truth table for digital circuits. The hardware evolution is carried out until the “desired” circuit is evolved and then downloaded to a reconfigurable hardware platform.

![Diagram of particle swarm optimization process](image)

The matrix shown in Fig. 3 is used to represent a circuit with \( m \) rows and \( n \) columns. The elements of the circuit are the logic gates which are selected from a predefined library of 2-input 1-output gates. The inputs to the first column of the matrix come from the truth table of the function to be implemented. For all other columns, the inputs may come from any of the previous column outputs.
For circuit evolution with PSO two matrices are used, one to represent the gate types, called the *gate matrix*, and the other to represent the interconnectivity between the gates, called the *input matrix*. The size of the input matrix is \(2n \times m\). There are \(2n\) rows because two elements of the input matrix correspond to one gate. The order of traversing the elements in the matrix is column wise, starting from the first column, going down through all the rows and then to the next column. For example, if there are 3 rows and 5 columns, then the input matrix is of size 6 by 5. The first two elements in the first column are inputs to the first gate, the next 2 to the second gate, and the last 2 to the third gate, in the first column of the gate matrix.

The initialization of the \(p\) (number of swarm particles) random circuits’ gate and input matrices are carried out first. Then their fitness is evaluated against the desired function to be implemented, given by a truth table. If the output of the circuit is equal to the output of the truth table for the corresponding inputs, then the fitness is increased by one. This is carried out for all inputs listed in the truth table. For each particle, the *pbest* gate matrix is determined and then the *gbest* matrix for the swarm is updated. The PSO algorithm is then applied to modify the gate matrix and the corresponding input matrix of each particle. This process is repeated until the fitness value of the *gbest* particle is equal to the number of the truth table outputs.

### IV. EXAMPLES OF EVOLVED CIRCUITS

Two examples are presented to describe the capability of particle swarm for hardware evolution. The library of gates used in this study is comprised of AND, OR, NOT, XOR, and a wire. The wire means no gate. The first example has three inputs and one output, as shown in Table 1. The evolved circuit satisfying the “desired” circuit is expected to have a fitness of eight in this case. A gate matrix size of 3 by 3 was chosen for the evolution with five swarm particles. The *gbest* evolved circuit over 300 generations contained two XOR gates and a NOT gate. The evolved circuit based on its gate and input matrices is given in Fig. 4. The results obtained by a human designer (HDI) and by the PSO are shown in Table 2. Here, the PSO does only as good as the human designer.

**TABLE 1:**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**TABLE 2:**

<table>
<thead>
<tr>
<th>HDI</th>
<th>PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F = Y \oplus (X \oplus Z))</td>
<td>(F = X \oplus (Y \oplus Z))</td>
</tr>
<tr>
<td>(3) gates</td>
<td>(3) gates</td>
</tr>
<tr>
<td>(2) XORs, (1) NOT</td>
<td>(2) XORs, (1) NOT</td>
</tr>
</tbody>
</table>

![Fig. 3 Structure of the random circuit matrix (inputs to each gate are obtained from gates in the previous columns).](image)

![Fig. 4 Circuit evolved by the PSO for the first example.](image)

Circuits used by Coello [10] using genetic algorithms and its variants, are studied with the PSO technique for comparison purposes. The second example is taken from [10]. This is a 3-even parity problem and its truth table is shown in Table 3. Again, this example has three inputs and one output. The evolved circuit satisfying the “desired circuit” is expected to have a fitness of eight in this case. A gate matrix size of 3 by 3 was chosen for the evolution with five swarm particles. The *gbest* evolved circuit over 227 generations contained one XOR gate, two AND gates, and one OR gate.

**TABLE 3:**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
For comparison, the result obtained by Coello is repeated in Table 4. Coello has used the n-cardinality GA (NGA) and multiobjective GA to evolve circuits with four gates and shown improvement over the human designer (HD1- five gates based on Karnaugh maps). A population of 90 chromosomes was used in both the MGA and NGA by the authors [10] and mutated over 300 generations. But with the PSO, a population size of five particles was used to get similar results with less than 300 iterations. This basically shows that PSO has faster convergence and fewer computations for these examples. Therefore, this is a potential technique for real-time hardware self-reconfiguration and needs to be further evaluated on large digital circuits. Similar results were obtained for other combinational circuit designs.

A large swarm population with 25 particles was initially chosen and gradually decreased. It was found that a population of five particles was sufficient to determine the best circuit (minimum number of gates) for this example. It is expected that for larger digital circuit designs, a large swarm population is required.

TABLE 4:
COMPARISON OF THE BEST SOLUTIONS BY THE HUMAN DESIGNER (HD1), MULTIOBJECTIVE GENETIC ALGORITHM (MGA), N-CARDINALITY GA (NGA), AND PSO.

<table>
<thead>
<tr>
<th></th>
<th>HD1</th>
<th>NGA</th>
<th>MGA</th>
<th>PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>F = Z(X + Y) + Y(X ⊕ Z)</td>
<td>Z(X ⊕ Y)</td>
<td>Z(X + Y)</td>
<td>Z(X + Y)</td>
<td>Z(X + Y)</td>
</tr>
<tr>
<td>5 gates</td>
<td>4 gates</td>
<td>4 gates</td>
<td>4 gates</td>
<td>4 gates</td>
</tr>
<tr>
<td>2 AND, 1 OR, 2 XOR</td>
<td>2 AND, 1 OR, 1 XOR</td>
<td>2 AND, 1 OR, 1 XOR</td>
<td>2 AND, 1 OR, 1 XOR</td>
<td></td>
</tr>
</tbody>
</table>

V. CONCLUSIONS

In this paper, it is shown that particle swarm optimization can also be applied to evolve combinational logic circuits. The emphasis was only on the generation of circuit functionality. From the few examples carried out, it is clear that particle swarm has potential for hardware evolution and even real-time hardware self-reconfiguration, since the convergence is faster. Reinforcement learning is a potential tool to be employed to minimize the number of active gates which are used in circuit evolution. Future work consists of concentrating on minimizing the active gates with multi-objective criteria, like considering a two level fitness function, where the first level obtains 100% functional circuits, while the second level minimizes the number of active gates and increases the number of wires used in the circuit evolution.

REFERENCES


