Global Journal of Technology and Optimization

Popa, Glob J Tech Opt 2015, 6:2
http://dx.doi.org/10.4172/2229-8711.1000174

Research Article Open Access

Toward EHW 2.0 – Some Ideas in HDL-Based Evolution of Digital Circuits

Rustem Popa*

Department of Electronics and Telecommunications – "Dunarea de Jos" University in Galati, Romania

***Corresponding author:** Rustem Popa, Department of Electronics and Telecommunications – "Dunarea de Jos" University in Galati, Stiintei Str., nr. 2, 800210, Galati, Romania, Tel: (0236) 460182; E-mail: Rustem.Popa@ugal.ro

Received date: January 06, 2015; **Accepted date:** January 07, 2015; **Published date:** January 17, 2015

Abstract

Evolvable Hardware (EHW) is a field of Evolutionary Computation (EC) started in the early 1990's that includes a subfield of Evolvable Hardware Design and a subfield of Adaptive Hardware. Two methods of evolvable hardware design of a one-bit full adder are analyzed in this paper: first method is based on the well-known idea of gate-level design using a network of programmable gates, and the second method uses Verilog instructions coded in chromosomes represented as binary strings. Eventually, the two solutions were compared in terms of hardware resources and propagation times.

Keywords Evolvable hardware; Genetic algorithms; FPGA; Digital electronics; Verilog HDL

Introduction

The conventional design process in electronics, either analog or digital, is top-down and begins with a precise specification. EHW design may be used when no hardware specification is known before. Its structure is adaptively searched using a Genetic Algorithm (GA) in a bottom-up way. EHW design combines knowledge of both GA and electronic design to evolve electronic circuits. It can offer new design solutions, which differ from the patterns typically used by humans.

In this paper we deal only with extrinsic evolution, which uses a model of the hardware and evaluates this model by simulation in software. Although [1] deals only with intrinsic evolution, which uses a real hardware programmable device, it is a very good up to date critical review of the field. It compares the current state of the field to that described in [2], a famous paper published by Yao and Higuchi 15 years ago.

Taking into account the progress of this research field in the 1990's, the two authors discussed the promises and possible advantages of EHW, and pointed on the challenges meet in order to develop real large-scale EHW. Unfortunately, [1] considers that the field is in a critical stage today. In its early stages, the field was developed with small steps forward. But the hope of bigger steps in the near future has not been confirmed by the passage of time. One of the main challenges remains scalability, which is representation of evolved complex circuits in terms of chromosomes [1,3,4]. Other challenge is measurement, that is fitness evaluation, or the metric used to evaluate the viability of an evolved circuit, and, on the other hand, the ability to compare traditional and evolved designs [1,5].

One of the first ideas for the use of Hardware Description Languages (HDLs) in the field of EHW was introduced in [6]. In order to improve the scalability, and taking into account that only structural HDL is synthesizable, while behavioral HDL is not, it was proposed, as a promising research direction, evolutionary-based compilation of behavioral HDL to structural HDL. A method of EHW design using Grammatical Evolution (GE) has been proposed in [7]. GE allows evolution in any programmable language, including Verilog HDL. As we can see in [8], there are a lot of functional HDLs, but Verilog HDL is the Industry Standard HDL.

The remainder of the paper is structured as follows: next section describes the two implementations of a one-bit full adder, the following section discuss some experimental results by comparing the two solutions in terms of hardware resources and propagation times in a FPGA circuit, and the last section contains conclusions.

Methods

Genetic implementation using a network of logic gates

The main stages of evolutionary design in electronics, both analog and digital, are illustrated in Figure 1. First, a population of chromosomes is randomly generated. Then, these chromosomes are converted into circuit models and circuit responses are compared with design specifications. Those chromosomes that generate the best circuits receive the best evaluations or the best fitness. They are selected as parents for a new generation of chromosomes.

If evolutionary algorithm is a GA, then new chromosomes are generated using genetic operators such as crossover and mutation. Through crossover, chromosomes are chosen two at a time, as parents, and their off springs are generated by exchanging parts of their structure. So, each offspring inherits a combination of features from both parents. Mutation means a small change in a new generated chromosome, with a small probability. In this way the algorithm can explore new features that are not yet in the population, in order to avoid premature convergence, when the diversity in population is lost. It makes the entire search space reachable despite the finite population size. The whole process is repeated for several generations, until is generated the best chromosome which represents a valid solution for our design.

Our problem is to design a circuit that performs a desired binary function, specified by a truth table, using a certain specified set of available logic gates. Based on the ideas presented in [9] and [10], we have used a network of programmable logic gates for each binary function in the circuit.

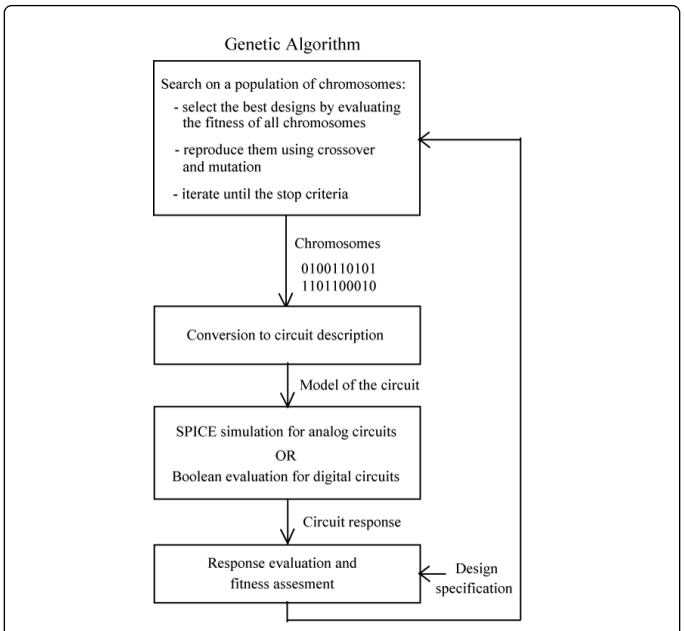

Figure 1: Evolutionary design of electronic circuits.

The proposed digital circuit is a one-bit full adder, which is in fact a system of two binary functions, each of them with three Boolean variables. Each function is implemented in a network of four gates, using a standard GA. Each gate has maximum two inputs and may be one of the following 8 types of logic gates: AND, NAND, OR, NOR, NOT taking into account the first input, NOT taking into account the second input, XOR, and the complement of XOR. An example with only 4 types of gates is given in Figure 2.

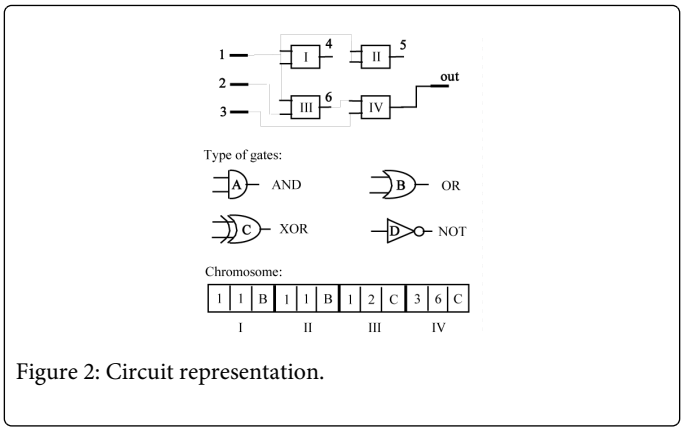

Figure 2: Circuit representation.

The first interesting aspect of this problem is the encoding of solutions as strings of bits, each string being a chromosome that the GA can evolve. Figure 2 explains the main idea used for this purpose. Each chromosome contains the whole information about the connections between inputs and outputs of the gates. Each gate contain three fields: the two first fields refer to each of the inputs used, and the third is the type of the gate [11].

If we want to implement the output si of the full adder of rank i, which has the boolean equation:

$$s_i = x_i \oplus y_i \oplus c_{i-1} \qquad (1)$$

then, as we can see in Figure 2, we need only two XOR gates, corresponding to the positions III and IV. The gates numbered with I and II are not used to implement this function, so, it doesn't matter how they are connected, and this part of the chromosome may be considered as beeing in fact "junk DNA". This evolved function is represented in Figure 3.

The other function of the adder, ci, was implemented on another similar network of four programmable gates, and Figure 3 presents the whole circuit.

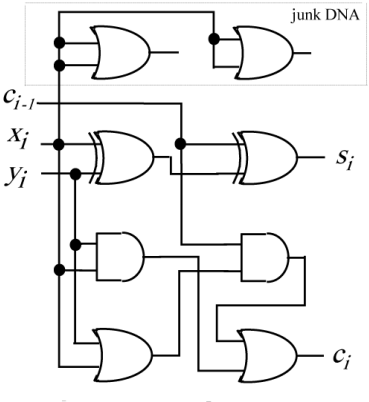

Figure 3: A possible evolved one-bit full adder.

The optimal equation of the carry output of the adder, ci, is:

$$c_i = x_i \cdot y_i + c_{i-1} \cdot (x_i \oplus y_i) \qquad (2)$$

because the XOR gate is shared by both functions of the adder. One of the potential evolved equations is represented in Figure 3:

$$c_i = x_i \cdot y_i + c_{i-1} \cdot (x_i + y_i) \qquad (3)$$

Equation (3) gives the same function as that represented in equation (2), and it's not optimal because the two function of the adder have evolved separately.

In order to represent the chromosomes that describe a binary function using circuit representation in Figure 2, we may choose the following codes to inputs and outputs of the gates: 1 – 000 or 110, 2 – 001 or 111, 3 – 010, 4 – 011, 5 – 100, and 6 – 101. Double codes could be used for constants 0 and 1. The gates may have the following codes: AND – 000, NAND – 001, OR – 010, NOR – 011, NOT1 – 100, NOT2 – 101, XOR – 110, and the complement of XOR – 111.

Thus, the length of a chromosome is 36 bits, and the best chromosomes that describe the functions si and ci are [11]:

$$crom\_s_i = 110000010110111110110110010010100110 \qquad (4)$$

$$crom\_c_i = 001000000000001010010100000011101010 \qquad (5)$$

As we can see, the first half of the chromosome given in (4) does not encode anything because two of the gates are not connected in circuit. So, half of the genetic information in first chromosome is rather "junk DNA".

Genetic implementation using Verilog HDL

Probably one of the first papers that proposes the use of HDL languages in EHW design is [6]. Taking into account that only

structural VHDL (Verilog) is synthesizable, while behavioral VHDL is not (because structural VHDL offers a problem decomposition), the following two research directions are proposed:

a) Evolutionary-based compilation of behavioral HDL to structural HDL. Force specifications to be in a standard behavioral language. b) Evolutionary-based synthesis of structural AHDL. AHDL is the acronym from Analog HDL. Structural AHDL may be the required first step to automatic analog synthesis.

Verilog HDL is an Industry Standard HDL and its syntax is simpler than that of VHDL. In this paper we deal with the synthesis of digital circuits, but the method may be also used for the synthesis of analog circuits. The building blocks may be sufficiently small to allow evolution toward optimal solution [6,12].

Verilog code for the one-bit full adder is given in Figure 4. It is a behavioral code, because Boolean functions are defined with instruction assign and there is no reference to the circuit structure, that is we don't care how the logic gates are interconnected. The only instructions that change in the evolutionary process are the two instructions assign, one for each of the two binary functions.

```
module one-bit_adder(X,Y,Carry_in,S,Carry_out);

    input X,Y,Carry_in;
    output S,Carry_out;

    assign S = X^Y^Carry_in;
    assign Carry_out = (X&Y)|(Carry_in&X)|(Carry_in&Y);

endmodule
```

Figure 4: Verilog HDL code for one-bit full adder.

First instruction assign describes function si, which is given in equation (1). If we represent this instruction in a binary coded chromosome, and we guess that our function follows a pattern like the one given in (6), where in1, in2, and in3 are any of the three inputs, and op1 and op2 are any of the valid logic operators, then the chromosome may be the one from (7).

$$crom\_s_i = in1\_op1\_in2\_op2\_in3 \quad (6)$$

$$crom\_s_i = 001100111010 \quad (7)$$

We have generated this chromosome using the following rules: in1 – 00, in2 – 01, in3 – 10, and logic operators are the same used for the circuit representation in Figure 2 (XOR – 110). This representation uses only 12 bits instead of 36 bits and the search space is much smaller in this case.

Function ci has another pattern. Instruction assign in Figure 4 (for function Carry_out) has three product terms written in a disjunctive form. If in is any of the three inputs, and op is any of the valid operators, then the pattern could be:

$$crom\_c_i = in\_op\_in\_op\_in\_op\_in\_op\_in\_op\_in \quad (8)$$

$$crom\_c_i = 000000101010000000101000001 \quad (9)$$

and the chromosome will have only 27 bits. If we use distributive for the last 2 product terms, then will remain only 4 operations, and the length of the chromosome will be only 24 bits. We can further reduce the length of the chromosome to only 22 bits, if we consider only AND, OR and NOT operators. This last pattern may be used also for si function, due to the idempotency theorem.

It is obvious that when we evaluate a chromosome as a solution of the problem, we must take into account the precedence of the operations. All these potential solutions are evaluated using a fitness function. In our case, for a single Boolean function, fitness is the ratio between the number of the correct values of the function and the number of all possible values (which is 2^n, if the Boolean function has n input variables). A well-designed circuit will be obtained only when the value of fitness is 100%. An approximately value of the fitness is unacceptable in our case.

Experimental Results

Another structure of an evolved one-bit full adder using a network of gates is given in Figure 5. This time, useful part of the first chromosome is the same, but the second chromosome is completely different, as we can see in (10) and (11):

$$crom\_s_i = 000001010110111110011010000010100110 \quad (10)$$

$$crom\_c_i = 000111001001000110010100001001101001 \quad (11)$$

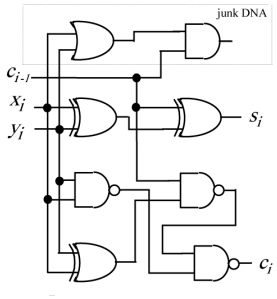

Figure 5: Another valid evolved one-bit full adder.

Both methods described above used a standard GA with the following parameters: population size was 128 chromosomes (a tournament type selection was used), 40 or 80 of them have been changed each generation, crossover probability 1 or 0.8, mutation probability 10%, and number of generations, 500 for the first method and only 100 for the second method.

GA was implemented in Matlab environment on an Intel Core 2 Duo CPU, 2,4 GHz, and, using the first method, the average time in 10 runs for si function was 5,487s, and for ci function was 14,269s.

Using the second method, based on Verilog HDL, the average time in 10 runs, for si function was only 0,506s, and for ci function was 0,694s.

We have tried also to implement this circuit in a real FPGA, in order to compare the resources used and the times of propagation. We have synthesized the circuit in a Spartan 3 FPGA from Xilinx, using the development integrated environment ISE 14.1. We found that there is no difference between the analyzed solutions. Both solutions implemented with gates, the conventional and the evolutionary, use 2 LUTs, each of them with 4 inputs, and 1 slice, time delay being between 5,479 and 5,882 seconds. Implementation with Verilog HDL, in structural and behavioral versions, use also 2 LUTs with 4 inputs and 1 slice, and global time propagation is between 5,481 and 5,853 seconds.

So, in conclusion, there is no advantage in FPGA implementation, because ISE environment optimizes the internal resources used to implement our functions, regardless of the form of representation. However, as we claimed from the beginning, EHW can provide surprising solutions in some cases, especially at a certain level of complexity, when a favorable pattern can provide a gain compared to conventional solutions. And the use of HDL languages seems to be a promising solution for the future.

Conclusions

The field of EHW has been developed for almost 20 years without having a uniform rate of growth. Current results are much more limited than previous expectations. We are confident that using of HDL languages can provide another significant development of this interesting area of research.

We fully agree with the authors of [1], who write that: "One thing is for certain, if we want to keep the field of evolvable hardware progressing for another 15 years we need to take a fresh look at the way we think about evolvable hardware: about which applications we should be applying it to; about where it will actually be of real use; about where it might finally be accepted as a valid way of creating systems and about what medium it should be based in.

We wish us all luck in this, the second generation of evolvable hardware."

References

1. Haddow P, Tyrrell A (2011) Challenges of evolvable hardware: past, present and the path to a promising future, Genetic Programe Evolvable Machines 12:183–215.
2. Yao X, Higuchi T (1999) Promises and Challenges of Evolvable Hardware, IEEE Trans. on Syst., Man, and Cyber. - Part C, 29: 87-97.
3. Vassilev VK, Miller JF (2000) Scalability Problems of Digital Circuit Evolution - Evolvability and Efficient Designs, NASA/DoD Workshop on Evolvable Hardware: 55-64.
4. Zebulum RS, Pacheco MA, Vellasco M (2000) Variable Length Representation in Evolutionary Electronics, Evolutionary Computation 8(1): 93-120. MIT Press.
5. Greenwood G, Tyrrell A (2007) Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems, Wiley–IEEE Press, New Jersey.
6. Stoica A, Zebulum RS, Keymeulen D, Ferguson MI, Guo X (2002) On Two New Trends in Evolvable Hardware: Employment of HDL-based Structuring, and Design of Multi-functional Circuits, NASA/DoD Conference on Evolvable Hardware: 56-59.
7. Cullen J (2008) Evolving Digital Circuits in an Industry Standard Hardware Description Language, 7-th Int Conf on Simulated Evolution and Learning SEAL 2008, Melbourne, Australia: 514-523.
8. Chen G (2012) A Short Historical Survey of Functional Hardware Languages, ISRN Electronics, 2012: 1:11.
9. Miller JF, Job D, Vassilev VK (2000) Principles in the Evolutionary Design of Digital Circuits—Part I, Genetic Programm Evolvable Machines, 1: 7–35.
10. Ali B, Almaini AEA, Kalganova T (2004) Evolutionary Algorithms and Their Use in the Design of Sequential Logic Circuits, Genetic Programm Evolvable Machines, 5: 11–29.
11. Popa R (2004) A Complete Laboratory on Evolutionary Electronics, Scientific Bulletin of the "Politehnica" University of Timisoara, Transactions on Electronics and Communications, 49(63), fascicle 2: 335-340.
12. Mita R, Palumbo G (2006) Modeling of Analog Blocks by Using Standard Hardware Description Language, Analog Integr Circ Sig Process 48:107–120.