Optimized Datapath Design by Evolutionary Computation

S. G. Araújo, A. C. Mesquita, A. C. P. Pedroza
Electrical Engineering Dept. – UFRJ
{granato,aloysio}@gta.ufrj.br mesquita@coe.ufrj.br

Abstract

High-level design entry tools offer a nice framework to deal with today's complex systems while shortening the design cycle. Nevertheless, such tools provide poor quality results both in area usage and timing performance issues. This paper presents a methodology to design optimized datapaths based on evolutionary techniques and HLS tools. VHDL descriptions of the system are automatically generated by Genetic Programming. To improve the design structural quality of such descriptions a two-stage, multiobjective optimization algorithm is used to insure both desired functionality and area constraints.

1. Introduction

Today's electronic digital systems are marked by increasing complexity and time-to-market pressures. Such systems are very difficult to design using methodologies starting from structural or physical descriptions. Automation of the entire design process via high-level synthesis (HLS) tools has become essential [1].

At the beginning of a HLS approach, the system functional specification must be supplied. To this end a variety of conceptual models can be used [2]. Hardware description languages (HDLs), such as VHDL, are conceptual models of particular interest considering that they describe data, activity and control simultaneously. The HLS outputs a register-transfer level (RTL) structural model; logic and layout synthesis steps follow in the sequence.

Even with high EDA tools availability, there is still a market resistance to the automatic behavioral synthesis approach: poor-quality results and lack of interactivity during the synthesis process are the main drawbacks [3]. High-level behavioral specifications are usually considered not quite "synthesizable" since at this abstraction level many low-level implementation details are masked from the designer [4].

In this work Genetic Programming (GP) is combined to HLS tools to automatically explore the design space and improve the design quality. GP, unlike other automatic learning paradigms (e.g., neural networks and genetic algorithms), emphasizes desired features such as the use of procedural representations, being adequate to evolve HDL descriptions. The datapath synthesis of a square-root operation, a difficult-to-implement function widely used in computer graphics and scientific calculation application, illustrates the methodology.

2. Previous works

Recent works [5, 6, 7] present evolved structures entirely in software using computer simulations. Kalganova & Miller [5] addressed a two-stage *multiobjective* fitness function. In the first stage a 100% functional solution, which occurs when the truth table is matched, is sought. In the second stage the number of gates actually used in each candidate solution is taken into account in the fitness function. This allows circuits to evolve with the desired functionality minimizing their number of active gates.

Hounsell & Arslan [6] addressed an EHW environment called *Virtual Chip* in which functional macros, such as half-adders and compound gates, are available to the evolutionary process. The main contribution of this work is the idea of using a *single* VHDL program describing all candidate circuits. In addition, formal grammars have been used to represent the evolution of hardware designs [7]. The authors use GP to evolve trees from productions (rephrased rules in the HDL grammar) that create SFL (Structured Function description Language) programs.

In the present approach, formal grammar of an HDL language was used to build candidate solutions, as in [7]. However, the place-and-route task was placed *inside* the evolutionary process in order to supply the fitness function with chip area information. The multiobjective fitness function adopted closely parallels the two-stage

approach suggested in [5], though applied in a different framework. Also, an experience was made using a single VHDL program, as suggested in [6].

3. Concepts

Evolutionary Computation (EC) consists of a class of probabilistic algorithms inspired by the principles of Darwinian natural selection and variations. Examples of EC are Genetic Algorithm (GA) and Genetic Programming (GP).

GA emulates the way nature improves the traits of living beings. The *chromosome* is the basic component of the GA. It represents a point, called in this context an individual, of the search (or solution) space of the problem. The fitness value measures how close the individual is to the solution. By iteratively applying genetic operations (fitness evaluation, fitness-based selection, reproduction, crossover and mutation) to a randomly started population of individuals, such a population evolves to breed one offspring with the desired behavior. GA usually works with strings (chromosomes) of *fixed length* in which one or more parameters are coded. GP is a branch of GA, the main difference being the solution representation. Unlike GA, GP can easily code chromosomes of *variable length*, which increases the capacity in structure creation.

Koza introduced canonical GP [8] with tree-based genome using LISP language. Other GP systems were developed to generate programs in arbitrary languages (other than LISP) by combining a context-free grammar (CFG) with GP, known as G^3P (Grammar-Guided GP). G^3P provides a framework for automatic creation of error free programs in arbitrary language once a BNF is provided as input. Genetic Programming Kernel (GPK) [9] is a complex G^3P system and was used as the core engine for the GP system used in this paper.

4. Methodology

The methodology evolves independent VHDL programs in search of a hardware description that implement the desired functionality *and* satisfies design constraints for area. The execution flow of the proposed methodology is drawn in Fig. 1. This figure shows the two main components of the methodology: the GP core and the Valuation function. Also, it can be seen that, differently to other approaches [6, 7], synthesis and place-and-route are performed for each candidate solution, which gives an exact value for the area.

Initially, a population of M individuals is created at random with the restriction of a pre-defined BNF containing a subset of VHDL grammar obeying the *sufficiency* property [8]. Then, an objective function is used to evaluate the fitness of each individual (genome) and a fitness value is assigned to each one. If any individual does not meet the target value of the objective function a new population is generated using the genetic operators.

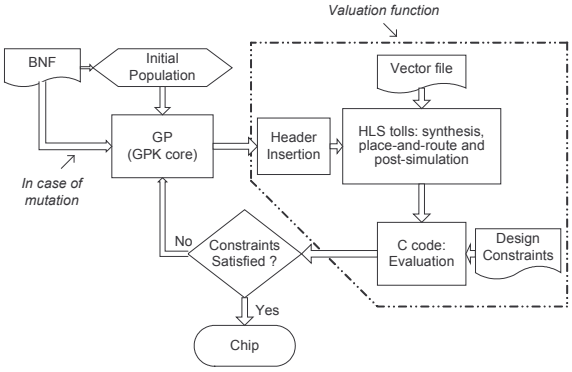


Fig. 1. Execution flow of the methodology

The steps involved in the evaluation of the objective function are depicted in Fig. 2. After a new population is created the VHDL source code of each individual is completed by inserting the entity declaration and the architecture declarative part. Then, the individuals are synthesized using Altera MAX+PLUS II compilation tool.

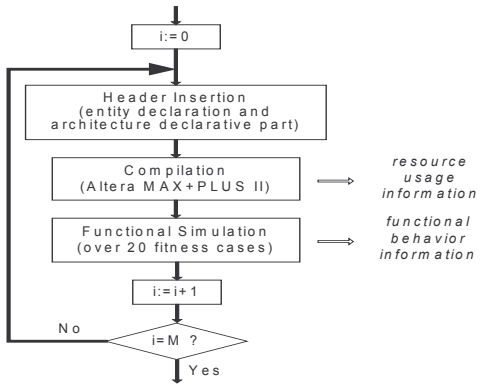


Fig. 2. Valuation block of the Fig. 1

The objective function to be minimized is a multi-parameter error function combining *functionality* with the *area* parameter in a two-stage approach similar to that proposed in [5]. In this process the functional specification (F_1 *fitness*) must be first matched to trigger the search for an optimal area implementation (F_2 *fitness*). Let the average functionality error, fe_{av}, be defined as:

$$fe_{av} = \frac{1}{N}\sum_{i=1}^{N} w_i \left| D_i - S_i \right| \quad (1)$$

Where, D_i is the desired value for fitness case i, S_i is the simulated value obtained for fitness case i, w_i is a weighting factor and N is the number of fitness cases. The fitness function F is defined as follows:

$$F = \begin{cases} F_1 = fe_{av} & if\ \exists i \in \{1,\mathrm{L}\ ,N\} \mid \left| \frac{D_i - S_i}{D_i} \right| > te \\ F_2 = fe_{av} * k(lu) & \text{otherwise} \end{cases} \quad (2)$$

Where, te is the target functional specification error for the fitness cases, lu is the number of used logic elements and k is an exponential function of lu that privileges the individuals containing the lowest quantities of logic elements. The *k(lu)* function takes the form:

$$k(lu) = ae^{b*lu} \quad (3)$$

Where, a and b are constants that must be defined by solving the following system:

$$\begin{aligned} k_{\min} &= ae^{b*lu_{\min}} \\ k_{\max} &= ae^{b*lu_{\max}} \end{aligned} \quad (4)$$

Where, lu_{min} *and* lu_{max} are minimum and maximum quantities of logic elements and k_{min} and k_{max} are the minimum and maximum values of *k(lu)*, respectively.

5. Experiment

To test the proposed methodology, it was selected a datapath design taken from [3]. The aim is to compute the square-root approximation (SRA) of two integers, a and b. In [3] the following approximation formula is given:

$$\sqrt{a^2 + b^2} \cong \max((0.875x+0.5y),x) \quad (5)$$

Where $x = max$ ($|a|$, $|b|$), and $y = min$ ($|a|$, $|b|$). In this problem of *symbolic multiple regression* [8] the goal is to find a function in symbolic form that is a good, or an exact, fit to a group of 20 sets of numerical data points. The *black box* of the SRA is given in Fig. 3. In this figure, the start input and done output signals are present for interface control issues.

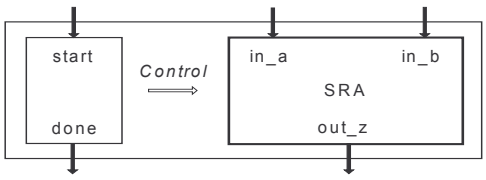


Fig. 3. *Black box* of the SRA

5.1. BNF definition

A BNF grammar describes admissible structures of a language, i.e., its syntax. The following BNF was defined to the SRA problem:

```
S      ::= <fe>;
<fe>   ::= "PROCESS BEGIN WAIT UNTIL (clk'event
       and clk='1'); IF in_a>=in_b THEN x<=in_a;
       y<=in_b; ELSE y<=in_a; x<=in_b; END IF;
       t6<=" <expr> "; IF t6>=x THEN t7<=t6;
       ELSE t7<=x; END IF; out_z<=t7; done<='1';
       END PROCESS;";
<expr>::= "(" <expr> ")" <op> "(" <expr> ")" |
       <var> |<var> "(7 downto " <shf_size> ")";
<op>   ::= " + " | " - ";
<var>  ::= " x " | " y " | " w ";
<shf_size> ::= "1"|"2"|"3"|"4"|"5"|"6"|"7";
```

In the above BNF description the signals “x”, “y”, “w”, “t6” and “t7” are internal signals of the VHDL architecture body. The signal “w”, assigned zero, is present to make ease the insertion or removal of a terminal.

5.2. Results

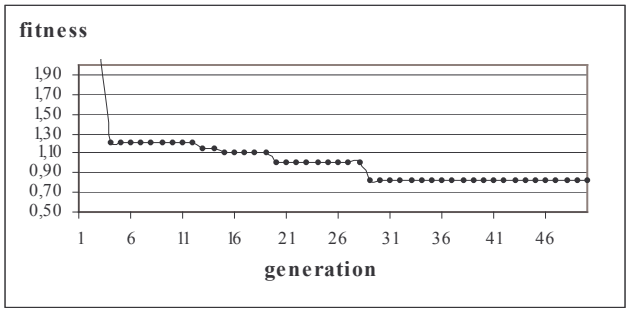
The EPF8282A FPGA from Altera FLEX family, with up to 282 logic elements (LEs), was used. In this experiment, area will be measured in number of LEs. The target functional specification error (*te*) was set to 2.5%. It establishes for *each* fitness case the maximum relative error between the simulated value of the evolved dependent variable, produced by the evolved VHDL programs, and the desired value.

A population size of 100 individuals was used, evolving up to a maximum of 51 generations. Crossover and mutation probability were set to 73% and 12%, respectively. Experiments have been carried out with N=20, w_i=1 and $k(lu)=0.25e^{(0.0139*lu)}$, which guarantees $k(lu) \in [0.5,1.0]$ for $lu \in [50,100]$ (Cf. eq.(3) and eq.(4)). A batch file containing VHDL compilation and simulation calls for each individual was built, implementing the valuation block (Fig. 2). The first functional individual was created at generation 18. The system started, then, the search for an implementation with optimal area. At the second stage of the evolution a number of individuals agreeing with the specified *te* value emerged, although with a non-optimal area use. Functional individuals with 95, 92 and 90 LEs have been reported. After 51 generations the best individual emerged. It was selected at generation 29, with main fitness F=0.827 and average functionality error fe_{av}=1.103. This individual was synthesized with 78 LEs. Its VHDL program statement

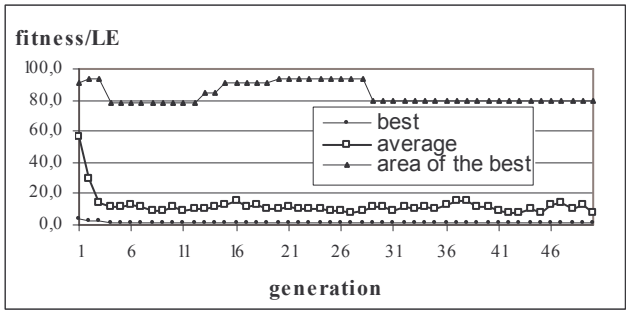
part was:

```
BEGIN
w <="00000000";
PROCESS BEGIN WAIT UNTIL (clk'event and
clk='1');
  IF in_a >= in_b THEN x <= in_a; y <= in_b;
  ELSE y <= in_a; x <= in_b; END IF;
  t6 <=((y(7 downto 1))+((x)-(x(7 downto3))))+
                          (y(7 downto 6));
  IF t6 >= x THEN t7 <= t6;
  ELSE t7 <= x; END IF;
  out_z <= t7;  done <= '1';
END PROCESS;
```

Fig. 4 shows (a) the evolution of the fitness of the best individual and (b) its area versus generation curve. In Fig. 4 (b) it can be noted that at generation 13 the best individual increases its area in order to achieve better accuracy. At generation 18 it reaches the desired accuracy and begins to look for optimal area, which is found at generation 29. Table 2 reports optimal area implementation (LE column) obtained for other values of *te* with the respective hardware descriptions.



a)



b)

Fig. 4. (a) Histogram of the fitness and (b) of the area of the best individual

Table 2. LE quantities for other *te* values with respective HW descriptions

te	LE	Hardware description
25%	51	`x + x(7 downto 3)`
15%	58	`x + y(7 downto 2)`
5%	70	`x - x(7 downto 3) + y(7 downto 1)`
2.1%	93	`((y(7 downto 7)) + ((y(7 downto 7)) + ((y(7 downto 6)) + ((x - (x(7 downto 3)))))) + y(7 downto 1))`

The downside of the methodology is its excessive time consumption. This is mainly due to the place-and-route task. Then, a pre-stage, to be run before the start of the main optimization process shown in Fig. 1, was introduced. In the pre-stage the system evolves as a single VHDL program as in [6], each individual is described in a single VHDL PROCESS. No place-and-route is accomplished at this time. The pre-stage finishes when the first functional individual comes out.

6. Conclusion

Evolutionary techniques have been presented as main alternatives to the rationalist bias of conventional automatic learning symbolic methods. Genetic Programming is an instance of evolutionary computation that generates solutions as procedural representations, being adequate to explore the solution space of HDL programs.

This work presents a digital IC design methodology based on Genetic Programming and high-level synthesis. In this methodology, the system description is in the form of a multiobjective function, i.e., a mathematical equation relating outputs to inputs and quality (area) parameters. A BNF, with basic VHDL operators, completes the input specification. The methodology may be extended to comprise performance optimization by inserting timing parameters in the objective function.

References

1. D. Gajski, N. Dutt, A. Wu and S. Lin, *High Level Synthesis: Introduction to Chip and System Design*, Kluver Academic Publishers, 1992.
2. D. Gajski, J. Zhu and R. Dörner, "Special Issues in Codesign", Technical Report ICS-97-26, 1997.
3. D. Gajski, I. Tadatoshi, V. Chaiyakul, H. Juan and T. Hadley, "A Design Methodology and Environment for Interactive Behavioral Synthesis", Technical Report 96-29, 1996.
4. K. Kuusilinna, T. Hämäläinen and J. Saarinen, "Practical VHDL Optimization for Timing Critical FPGA Applications", Microprocessors and Microsystems 23, pp. 459-469, 1999.
5. T. Kalganova and J.Miller, "Evolving More Efficient Digital

Circuits by Allowing Circuit Layout Evolution and Multi-Objective Fitness", Proc. of the 1st NASA/DoD Workshop on EHW, pp. 54-63, CA, IEEE Press, 1999.
6. B. Hounsell and T. Arslan, "A Novel Evolvable Hardware Framework for the Evolution of High Performance Digital Circuits", GECCO-2000, pp. 525-532, 2000.
7. H. Hemmi, J. Mizoguchi and K. Shimohara, M. Tomassini, "Development and Evolution of Hardware Behaviors", Int. Workshop, Lausanne, LNCS 1062, pp. 250-265, 1996.
8. J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.
9. H. Hörner, "A C++ Class Library for Genetic Programming, Release 1.0 Operating Instructions", Viena University of Economy, 1996.