Evolutionary Design and FPGA Implementation of Digital Filters

Antonia Azzini[a], Matteo Bettoni[a], Valentino Liberali[a],
Roberto Rossi[b], and Andrea Tettamanzi[a]

[a]Dept. of Information Technologies, University of Milano, 26013 Crema, Italy
[b]Dept. of Electronics, University of Pavia, 27100 Pavia, Italy

ABSTRACT

This paper discusses the use of evolutionary algorithms to design digital circuits. It is shown that evolutionary design can be fully compliant with the existing design methodologies. Moreover, the evolutionary design is capable to perform a better exploration of the design space, and therefore it can find solutions having different features with respect to conventional design. In some cases, evolved circuits can have better performances, or they can be optimized with respect to different parameters. An example on design of a multi-rate digital filter with reduced power consumption is presented and discussed. FPGA implementation demonstrates that evolutionary design can lead to both area and power saving with respect to conventional design.

Keywords: digital filters, low-power design, evolutionary design, VLSI CAD tools, FPGA

1. INTRODUCTION

Electronics design automation must cope with technological trend in silicon integration. Nowadays, the possibility of integrating millions of transistors onto a single silicon chip is demanding for new CAD tools, to bridge the gap between technology and design. As integration technology enables the development of deep submicron CMOS circuits for digital signal processing (DSP) in an ever and ever increasing variety of applications, designers have to face new problems which require new design methodologies and tools.

The increasing gap between technology capability and designer productivity is demanding for new design methodologies and innovative CAD tools, as in previous stages of the "design crisis".[1] Following the introduction of electronics design automation tools for physical design (partitioning, placement and routing) and, more recently, for digital circuit synthesis from behavioral description (hardware description languages), now a remarkable effort is being spent, aiming at the automatic synthesis starting from functional specifications rather than from behavioral representation.

Therefore, new branches in computer aided design are expected to emerge in the next future: among them, evolutionary algorithms seem to be very promising, due to their capability to provide solutions to hard design problems. Evolutionary algorithms are a broad class of optimization methods, built on the key concept of Darwinian evolution in biology.[2] Now bio-inspired electronic design methods are being considered in a variety of circumstances.[3,4] Evolutionary algorithms for circuit synthesis are a powerful technique, which could provide innovative solutions to several design problems, also when classical decomposition methods may fail.[5]

This paper presents an evolutionary approach to the design of digital filters. Starting from frequency specifications, the VHDL code of the digital filter is generated automatically. The proposed method has been successfully applied to the design of a decimation filter for a $\Sigma\Delta$ analog-to-digital converter. The evolutionary design has been implemented on a Xilinx FPGA, demonstrating that circuit designed through evolutionary algorithms can be easily optimized by FPGA tools for synthesis, placement and routing, thus resulting in smaller area and better timing and power performance.

The paper is organized as follows. Sect. 2 describes the evolutionary design methodology. Sect. 3 illustrates the proposed representation of digital filters, suitable for the evolutionary algorithm described in Sect. 4. Finally, Sect. 5 presents a design implementation on FPGA and discusses the advantages of the proposed methodology.

Send correspondence to V.Liberali:
E-mail: liberali@dti.unimi.it, Telephone: +39 0250330071, Address: Department of Information Technologies, University of Milano, Via Bramante 65, 26013 Crema, Italy

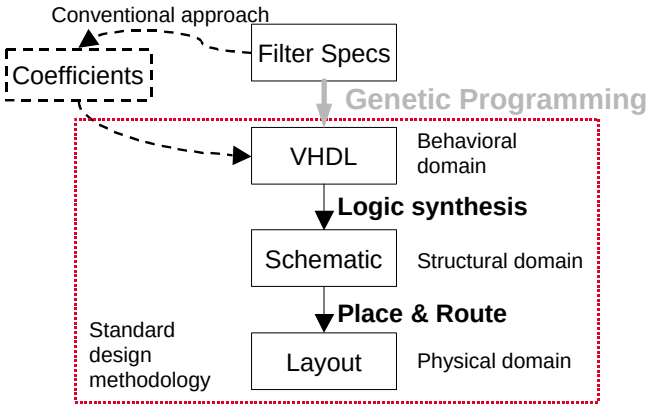

Figure 1. Digital design methodology.

2. EVOLUTIONARY DESIGN

Evolutionary algorithms have been proven to be able to produce designs which are more efficient than structured conventional approaches in the design of combinational circuits.[6]

Genetic approaches have been also applied to the design of sequential circuits, especially for digital filters. However, in most approaches described in literature, the genetic algorithm is just applied to optimize the coefficients of a digital filter obtained in a conventional manner.[7,8] A completely different design style consists in having the whole design performed by an evolutionary algorithm.[9] Indeed, a digital filter can be represented as a sequence of elementary operations, which can easily be encoded to be handled by a genetic algorithm. Such a representation is closer to the low-level filter structure, and gives us the opportunity to explore the design space in a more complete way. In particular, it is possible to evaluate directly the power consumption through the switching activity of digital nodes, therefore aiming at low-power design.

Moreover, the proposed design approach is fully compliant with the well-consolidated digital design methodology: the genetic algorithm is used just to produce the synthesizable VHDL code, which is translated into structural and physical domains by means of other tools commonly used in electronic design automation. Fig. 1 represents a diagram of the digital design tools used for the design of a digital filter. Conventional approach, illustrated by the dashed line, would consist in the design of an "ideal" filter (with infinite precision coefficients), and then in the approximation with finite word arithmetic. Automatic generation of VHDL code through a genetic algorithm is more straightforward, and allows the designer to move at a higher design entry level.

3. GENETIC REPRESENTATION OF DIGITAL FILTERS

A description of digital filters can derived from their frequency response in the z-domain.[10] The frequency response of a finite impulse response (FIR) digital filter is:

$$H(z) = \sum_{k=0}^{N-1} h(k) z^{-k} \tag{1}$$

and the canonical direct form of the filter shown in Fig. 2.

To save area and to reduce power consumption, generic multiplier blocks are often replaced with shifters and adders.[11] As an example, multiplication by 13 can be implemented with two shifts and two additions, as illustrated in Fig. 3, where the block "$<< n$" means "left shift by n positions".

The canonical signed digit (CSD) representation assigns a separate sign to each digit: 0, 1 and $\bar{1}$ (= –1). Its goal is to minimize the number of non-zero digits: by encoding the filter coefficients with CSD, the filter output can be computed using a reduced amount of hardware, since multiplications by zero are simply not

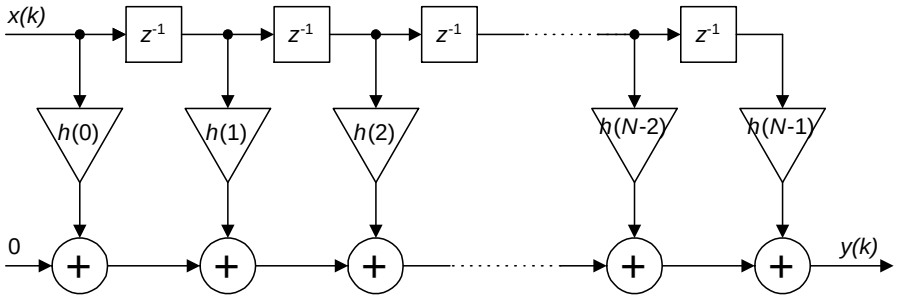

Figure 2. Canonical direct form of a FIR filter.

implemented.[12] As an example, consider the multiplication by 15: since $15 = 2^3 + 2^2 + 2^1 + 2^0 = (001111)_2$, this operation in binary arithmetic would require three shifts and three additions; while using CSD we can write $15 = 2^4 - 2^0 = (01000\bar{1})_2$, and we implement the same operation using only one shifter and one subtractor (Fig. 4).

Starting from these considerations, a digital filter can be described using a very small number of elementary operations. The primitives selected for digital filters are listed in Table 1. Each elementary operation is encoded by its own code (one character) and by two integer numbers, which represent the relative offset (calculated from the current position) of the two operands. When all the offsets are positive (i.e. each block can receive data from previous blocks only), no feedback loop occurs and the resulting structure is a FIR filter. All primitives include a delay z^{-1}, to avoid possible problems due to timing violations during the synthesis process.

Since the primitive operators requiring an adder block are more expensive in terms of power dissipation, a relative weight factor has been assigned to sum, difference and complement.

As an example, the following sequence is made of 6 primitives (6 genes):

```
(I 0 2) (D 1 3) (L 2 2) (A 2 1) (D 1 0) (S 1 5)
```

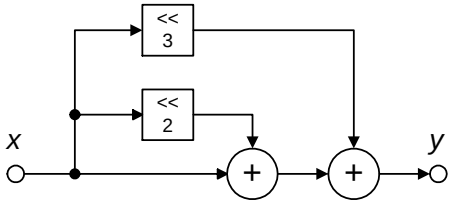

Figure 3. Multiplication by 13 implemented with shifters and adders.

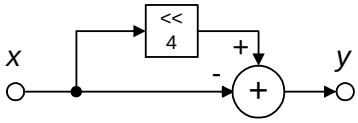

Figure 4. Multiplication by 15 implemented with one shifter and one subtractor.

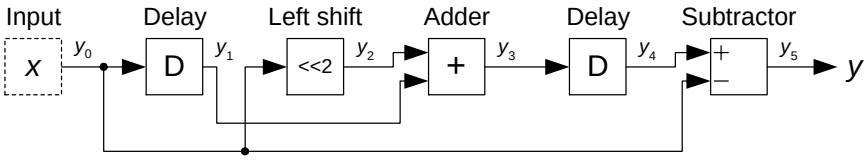

Figure 5. Schematic diagram corresponding to a sequence of 6 primitives.

It corresponds to the schematic diagram shown in Fig. 5, and it is interpreted as follows:

$$\begin{aligned} y_0 &= x \\ y_1 &= y_0 z^{-1} \\ y_2 &= 2^2 y_0 z^{-1} \\ y_3 &= (y_1 + y_2) z^{-1} \\ y_4 &= y_3 z^{-1} \\ y = y_5 &= (y_4 - y_0) z^{-1} \end{aligned} \tag{2}$$

The last value is the output of the filter. By merging the equations (2), we obtain the transfer function:

$$H(z) = \frac{y}{x} = 5z^{-4} - z^{-1} \tag{3}$$

Such a representation has the same essence as a program in a simple imperative programming language. Therefore, digital filter design can be automated through genetic programming.[13,14] Fine granularity of primitives leads to a simple genetic encoding, and allows the evolutionary algorithm to perform a better search within the design space.

Efficiency of the designed structure (in term of power consumption) is an important target, which should be optimized under the specification constraints. The dynamic power P due to logic transitions of circuit nodes is:

$$P = \frac{1}{2} C V^2 f \alpha \tag{4}$$

where C is the total capacitance, V is the power supply voltage, f is the clock frequency, and α is the transition activity, i.e. the average number of logic transitions in a clock period.[15,16]

4. THE EVOLUTIONARY ALGORITHM

In order to make filters evolve, a variable population size island distributed evolutionary algorithm has been implemented. Details on the genetic operators have been presented in the literature.[9]

Table 1. Primitives of the genetic algorithm.

Name	Code	Op 1	Op 2	Description
Input	`I`	not used	not used	Copy input: $y_i = x$
Delay	`D`	n_1	not used	Store value: $y_i = y_{i-n_1} z^{-1}$
Left shift	`L`	n_1	p	Multiply by 2^p: $y_i = 2^p y_{i-n_1} z^{-1}$
Right shift	`R`	n_1	p	Divide by 2^p: $y_i = 2^{-p} y_{i-n_1} z^{-1}$
Adder	`A`	n_1	n_2	Sum: $y_i = (y_{i-n_1} + y_{i-n_2}) z^{-1}$
Subtractor	`S`	n_1	n_2	Difference: $y_i = (y_{i-n_1} - y_{i-n_2}) z^{-1}$
Complement	`C`	n_1	not used	Multiply by -1: $y_i = -y_{i-n_1} z^{-1}$

4.1. Fitness Function

The evolutionary algorithm must find a design that satisfies user specifications and minimizes transition activity of digital gates. These two objectives, however, are hierarchical: specification fulfillment is mandatory, while reduction of digital transitions is just an additional quality, which is meaningful only when the circuit is fully compliant with requests.

Therefore, a two-step fitness function has been devised. First of all, we consider filter specifications (given as a frequency mask): the frequency range is sampled at N equally spaced frequencies ω_i, and the filter response $H(\omega_i)$ is calculated for every ω_i in the pass-band and in the stop-band. The partial error ϵ_i is set to zero, if $|H(\omega_i)|$ lies within the specifications; otherwise it is proportional to the overshoot or undershoot with respect to the given tolerance. The total error E_{tot} is simply the sum of all partial errors: it is zero when the frequency response of the filter lies within the mask; otherwise, it has a non-zero value. Mask fitness is defined as:

$$f_M = \frac{1}{1 + E_{\text{tot}}} \tag{5}$$

It measures the extent to which the filter complies with frequency specifications and $f_M = 1$ when specifications are completely met.

The second step is the evaluation of the activity fitness, defined as:

$$f_A = 1 + \frac{a}{N_T} \tag{6}$$

where N_T is the number of weighted digital transitions per input sample and a is a constant; its contribution is higher for filters with low transition activity, which is responsible for power consumption.

Finally, the overall fitness is defined as:

$$f = \begin{cases} f_M & \text{if } E_{\text{tot}} > 0 \\ f_M + f_A & \text{otherwise} \end{cases} \tag{7}$$

Such a definition guarantees that circuits complying with specifications have a larger fitness than circuits with a response outside the frequency mask.

4.2. Selection

Different selection strategies have been implemented:

1. an "aging" rule, which assigns a life expectation to every individual at its birth, according to its ranking: life expectation linearly decreases from 100 to 1 generation as the ranking goes from best to worst;
2. fitness-proportionate selection, whereby each individual has a survival probability equal to its normalized fitness;
3. linear ranking selection with elitism, whereby the best two individuals survive with probability $P_s = 1$, and the individual of rank i has a probability $P_s = 1 - \frac{i}{2\bar{n}}$ of surviving, where $\bar{n}$ is a parameter of the algorithm affecting the average population size;
4. selection based on normalized fitness and ranking, whereby each individual has a survival probability $P_s = \sqrt{f_i' r_i'}$ where f_i' and r_i' are the normalized fitness and rank;
5. histogram-based selection, whereby the selection aims at a uniform distribution of the fitness.

Experimental evidence shows that each selection strategy performs better in a particular stage of the evolution. In the examples presented below, all strategies have been used in parallel.

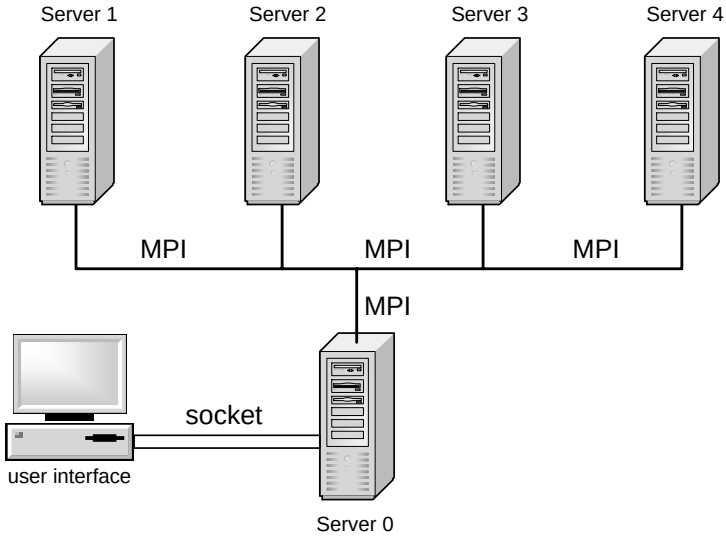

Figure 6. Distributed genetic algorithm implemented with MPI.

4.3. MPI Implementation

Message Passing is a paradigm widely used for loose parallel algorithms. Although there are many variations, the basic concept of processes communicating through messages is well understood, furthermore several systems have demonstrated that a message passing system can be efficiently and portably implemented. The main reasons for using the message passing can be identified in the following points: (1) standardization: hardware acquaintances are minimal, as function libraries are available for several processor and system architectures; (2) portability: it is platform independent; (3) functionality: many routines are defined in the core library; and (4) easiness: it supports dynamic control.

Message Passing Interface (MPI) is one of the more useful standards that support this type of communication. It is defined such as a portable message passing standard that facilitates the development of parallel applications and libraries. This standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in the C language.

The main advantages of establishing a message passing standard communication are portability and ease of use. In this case the interface should establish a practical, portable, efficient, and flexible standard for message passing. In a distributed memory communication environment, in which the higher level routines and/or abstractions are build upon lower level message passing routines, the benefits of standardization are particularly apparent.[17, 18]

It is worth highlighting that MPI is based on objects, such as communicators and groups, in order to establish which set of processes can communicate. A group is an ordered set of processes. Each process in a group is associated with a unique integer rank. A group is always associated with a communicator object, which specifies the group of processes involved and the communication context. The default communicator includes the entire set of processes.

Fig. 6 shows the structure of the genetic version that support MPI. A client can control many server using a support server identified as “Server 0”, that use MPI for the genetic communications. The other servers execute the genetic algorithm and can exchange information during the simulated evolution, using the MPI standard. In our implementation, each server process simulates the evolution of a separate population of digital circuits. At defined time intervals, an individual is randomly extracted from a population (a server process) and sent to another process. Such a “migration” helps in improving the convergence of the genetic algorithm towards a valid solution.

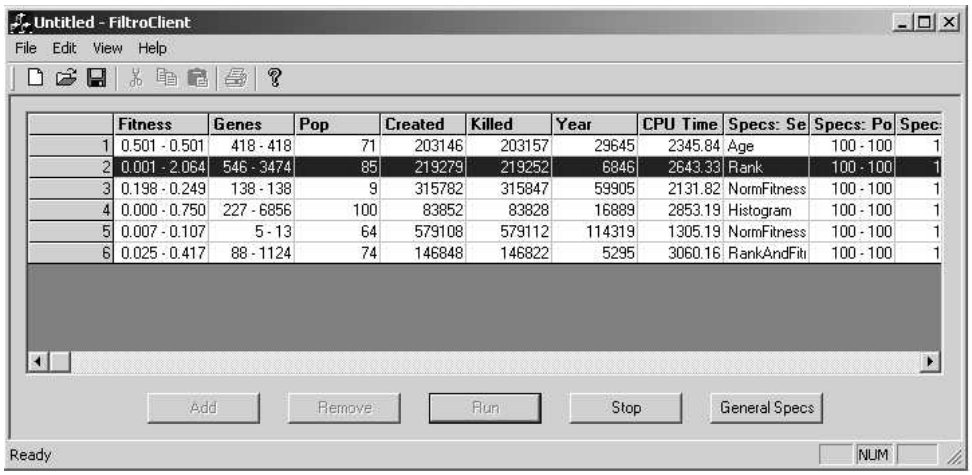

Figure 7. Client window.

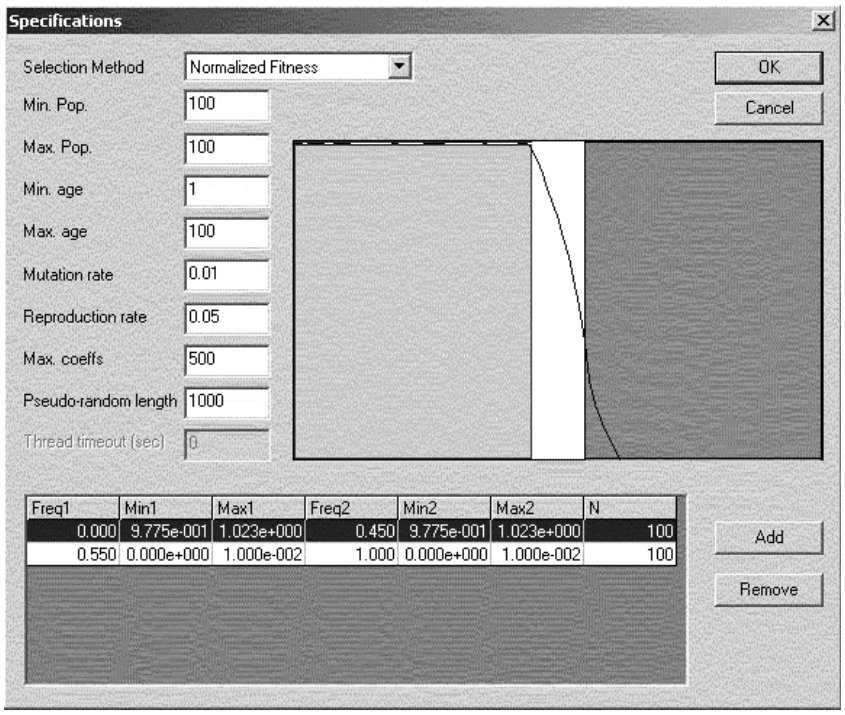

Figure 8. Graphical interface with frequency mask.

The client window, depicted in Fig. 7, shows a number of processes running in parallel on different CPUs. Each process implements the genetic algorithm, using different selection rules. The best solution of each process can be displayed on the user console, as illustrated in Fig. 8.

5. A DESIGN EXAMPLE

The proposed design method has been applied to the design of a multi-stage digital filter to be used for decimation in a $\Sigma\Delta$ analog-to-digital converter. Table 2 summarizes the specifications of the filter. The decimation factor is 16 and the maximum data rate (at the input) is 80 MHz.

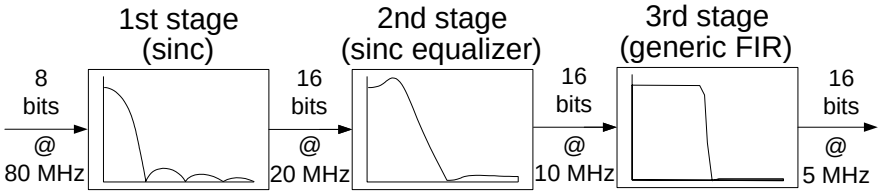

Figure 9. Three-stage filter architecture with a decimation ratio of 16.

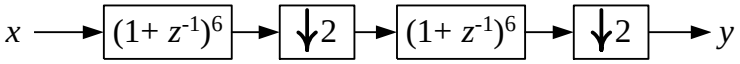

Figure 10. Block diagram of the first stage used in the filter.

5.1. Multi-Stage Architecture

Digital filtering and sampling rate reduction is carried out in several stages,[19] since a single stage implementation would require a huge number of coefficients, thus resulting in a prohibitively large silicon area occupation and power consumption. In our case, it was deemed convenient to implement this decimator as a three-stage configuration, as this allows a good trade-off between overall power consumption and design complexity.[20] The oversampling ratio of 16 was factorized as $4 \times 2 \times 2$, and the resulting three-stage architecture is illustrated in Fig. 9. FIR filters are used for decimation, because they have linear phase and inherent stability.[21]

The first decimation stage can be implemented with a simple sinc^k filter (where k is larger than the modulator order) operated at a very high sampling rate.[22,23] The architecture of the first stage is shown in Fig. 10. The decimation factor of 4 has been factorized as 2×2 to reduce area and power. By applying the Chu-Burrus commutative rule,[24] the combination of the two filtering blocks then gives:

$$\left(1 + z^{-1}\right)^6 \cdot \left(1 + z^{-2}\right)^6 = \left(1 + z^{-1} + z^{-2} + z^{-3}\right)^6 \tag{8}$$

which is easily recognized as a sixth-order, four-tap sinc filter.

Subsequent decimation stages work at a lower sampling rate, but have to process long words to achieve the required accuracy. Moreover, since the output digital signal is at Nyquist rate, a steep transition band and a high stop-band attenuation are required so as to eliminate all the out-of-band noise while not affecting the in-band signal. In our design, the second and the third stages have been implemented with FIR filters.

A conventional design has been made using the `remez` function available in the Matlab Signal Processing Toolbox,[25] and by approximating ideal coefficients with finite precision numbers. Moreover, the second stage has been designed to compensate for the attenuation introduced by the sinc stage in the passband. An ASIC prototype in 0.25 μm CMOS technology was successfully characterized.[26]

Table 2. Filter specifications.

Digital input resolution	8 bit
Digital input rate	80 MHz
Digital output resolution	13 bit
Output S/N ratio	78 dB
Output in-band ripple	0.2 dB
Digital output rate	5 MHz
Decimation factor	16

5.2. Evolutionary Design

The evolutionary algorithm described in Sect. 4 has been used to redesign both the second and the third filtering stages. For both stages, the evolutionary designs have better characteristics in terms of number of equivalent gates and power consumption.

The genomes of the two evolved filter stages have 36 primitives and 211 primitives for the second and for the third stages, respectively. Fig. 11 shows an example of the generated VHDL code.

5.3. FPGA Implementation

The VHDL code generated by the algorithm contained the description of the RTL (register-transfer logic) architecture, which is imported into the Xilinx FPGA design environment and is used for the synthesis. The design has been synthesized for three different devices of the Virtex family: XCV600, XCV800, and XCV1000, having 600 Kgates, 800 Kgates, and 1000 Kgates, respectively.[27]

For comparison purposes, the conventional design has been synthesized for the same devices. Table 3 compares the characteristics of the designs after placement and routing, and Table 4 compares the interconnection delays of the two designs.

The filter has been validated on a board employing a Xilinx XCV1000 FPGA. The input pattern shown in Fig. 12 has been provided using a logic analyzer: the first part of the pattern is used to align the actual output sequence with the expected one; the rest of the pattern is a pseudo-random sequence. The evolutionary filter has been proven to be equivalent to the conventional design.

By comparing figures in Table 3, we can conclude that the effort in reducing the transition activity has led to an implementation with reduced hardware: when using the XCV1000 device, the evolutionary filter design requires only 39% of the digital blocks used for the reference design.

It is worth remarking also that the low-level RTL description is easily optimized by the Xilinx CAD tools, thus leading to the same block occupancy for all the three considered devices. On the other hand, we can point out that the reference design, described in a higher-level VHDL code, is not optimized in the same way for the different FPGA devices.

Finally, from Table 4 we can observe that smaller circuits lead to better timing performance, as expected.

6. CONCLUSION

This paper has described an evolutionary approach to digital design, illustrating how digital filters can be obtained from scratch by using a genetic algorithm. Genetic encoding of filter primitives has a fine granularity which is exploited by the evolutionary algorithm during its random search. The encoding is also a straightforward representation of the time-domain response of the filter, thus allowing a direct evaluation of transition activity of

Table 3. Number of configurable logic blocks required (after placement and routing).

used logic blocks	XCV600	XCV800	XCV1000
conventional design	6910 (99%)	9406 (99%)	10078 (82%)
evolutionary design	3919 (56%)	3919 (41%)	3919 (31%)

Table 4. Interconnection delay (average value for 10 worst interconnections, after placement and routing).

worst delay	XCV600	XCV800	XCV1000
conventional design	12.9 ns	15.6 ns	16.2 ns
evolutionary design	10.7 ns	11.4 ns	12.4 ns

```
library ieee, work;
use ieee.std_logic_1164.all;
use work.Terne.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Filter is
   port(
      y:out std_logic_vector(30 downto 0);
      clk16: in std_logic;
      clk:in std_logic;
      rst:in std_logic;
      x:in std_logic_vector(15 downto 0)
   );
end entity;

architecture RTL of Filter is
   signal yclock: std_logic_vector(15 downto 0);
   signal y0: std_logic_vector(15 downto 0);
   signal y1: std_logic_vector(16 downto 0);
   signal y2: std_logic_vector(16 downto 0);
   signal y3: std_logic_vector(17 downto 0);
   signal y4: std_logic_vector(17 downto 0);
   ...
   signal y35: std_logic_vector(30 downto 0);

begin
   y <= y35;

   gene00: ClockDomainInterface_gen generic map(Bits => x'length, FF1 => 0, FFInv => 1, FF2 => 0)
           port map(x => x, y => yclock, clk1 => clk16, clk2 => clk, rst => rst);
   gene0: init_gen generic map(Bits => x'length)
          port map(x => yclock, y => y0, clk => clk, rst => rst);
   gene1: adder_gen generic map(Bits_x1 => y0'length, Bits_x2 => x'length, Bits_y => y1'length)
          port map(x1 => y0, x2 => x, y => y1, clk => clk, rst => rst);
   gene2: change_s_gen generic map(Bits => y2'length)
          port map(x => y1, y => y2, clk => clk, rst => rst);
   gene3: subtr_gen generic map(Bits_x1 => y2'length, Bits_x2 => y1'length, Bits_y => y3'length)
          port map(x1 => y2, x2 => y1, y => y3, clk => clk, rst => rst);
   gene4: delay_gen generic map(Bits => y4'length)
          port map(x => y3, y => y4, clk => clk, rst => rst);
   ...
   gene35: subtr_gen generic map(Bits_x1 => y34'length, Bits_x2 => y33'length, Bits_y => y35'length)
           port map(x1 => y34, x2 => y33, y => y35, clk => clk, rst => rst);
end architecture;
```

Figure 11. Synthesizable VHDL code generated by the algorithm

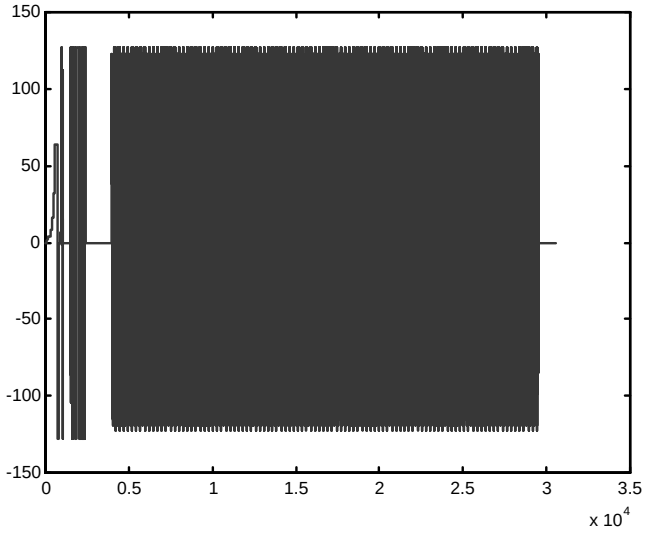

Figure 12. Input test pattern.

digital blocks, and, hence, of power consumption. The "best" result produced by the algorithm is automatically translated into VHDL code, which is directly synthesized into a circuit according to the standard digital design methodology.

The results obtained with the simulated evolution show that minimization of transition activity leads to a dramatic reduction of the hardware with respect to the conventional design methodology, while maintaining the same performance.

Implementation of a decimation filter on a Xilinx FPGA shows a reduction from 43% to 61% in the number of configurable logic blocks used (the actual figure depending on the specific FPGA device). Area reduction leads to improvements in both power consumption and circuit speed.

REFERENCES

1. K. Wakabayashi and T. Okamoto, "C-based SoC design flow and EDA tools: An ASIC and system vendor perspective," *IEEE Trans. Computer-Aided Design of Integr. Circ. and Syst.* **19**, pp. 1507–1522, Dec. 2000.
2. T. Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, Oxford, UK, 1996.
3. R. Drechsler, *Evolutionary Algorithms for VLSI CAD*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.
4. R. S. Zebulum, M. A. C. Pacheco, and M. M. B. R. Vellasco, *Evolutionary Electronics*, CRC Press, Boca Raton, FL, USA, 2002.
5. A. Thompson, P. Layzell, and R. S. Zebulum, "Explorations in design space: Unconventional electronics design through artificial evolution," *IEEE Trans. Evolutionary Computation* **3**, pp. 167–196, Sept. 1999.
6. J. F. Miller, D. Job, and V. K. Vassilev, "Principles in the evolutionary design of digital circuits – Part I," *Genetic Programming and Evolvable Machines* **1**, pp. 7–35, Apr. 2000.
7. S. P. Harris and E. C. Ifeachor, "Automatic design of frequency sampling filters by hybrid genetic algorithm techniques," *IEEE Trans. Signal Processing* **46**, pp. 3304–3314, Dec. 1998.
8. A. Lee, M. Ahmadi, G. A. Jullien, R. S. Lashkari, and W. C. Miller, "Design of 1-D FIR filters with genetic algorithm," in *Proc. IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, **III**, pp. 295–298, (Orlando, FL, USA), 1999.
9. M. Erba, R. Rossi, V. Liberali, and A. G. B. Tettamanzi, "An evolutionary approach to automatic generation of VHDL code for low-power digital filters," in *Genetic Programming – Proc. 4th European Conference (EuroGP2001)*, pp. 36–50, (Como, Italy), Apr. 2001.

10. L. B. Jackson, *Signals, Systems, and Transforms*, Addison-Wesley, Reading, MA, USA, 1991.
11. Q. Zhao and Y. Tadokoro, "A simple design of FIR filters with power-of-two coefficients," *IEEE Trans. Circ. and Syst.* **35**, pp. 556–570, May 1988.
12. P. Pirsch, *Architectures for Digital Signal Processing*, John Wiley & Sons, Chichester, UK, 1998.
13. J. R. Koza, *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, The MIT Press, Cambridge, MA, USA, 1993.
14. J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming – Proc. 3rd European Conference (EuroGP 2000)*, R. Poli *et al.*, eds., *Lecture Notes in Computer Science* **1802**, pp. 121–132, Springer-Verlag, (Berlin, Germany), Apr. 2000.
15. C.-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. M. Despain, and B. Lin, "Power estimation methods for sequential logic circuits," *IEEE Trans. VLSI Systems* **3**, pp. 404–416, Sept. 1995.
16. M. Pedram, "Power minimization in IC design: Principles and applications," *ACM Trans. on Design Automation of Electronic Systems* **1**, pp. 3–56, Jan. 1996.
17. P. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann Publishers, San Mateo, CA, USA, 1997.
18. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI – The Complete Reference: Vol. 1, The MPI Core*, MIT Press, Cambridge, MA, USA, 1998.
19. R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1983.
20. M. Brambilla, D. Guidi, and V. Liberali, "High speed FIR filters for digital decimation," in *Proc. XII Brazilian Symposium on Integrated Circuit Design*, pp. 124–127, (Natal, Brazil), 1999.
21. A. V. Oppenheim and A. S. Willsky, *Signals & Systems (2nd ed.)*, Prentice-Hall, Upper Saddle River, NJ, USA, 1997.
22. E. B. Hogenauer, "An economical class of digital filters for decimation and interpolation," *IEEE Trans. Acoust., Speech and Sign. Proc.* **29**, pp. 155–162, Apr. 1981.
23. J. C. Candy, "Decimation for sigma delta modulation," *IEEE Trans. Communications* **34**, pp. 72–76, Jan. 1986.
24. S. Chu and C. S. Burrus, "Multirate filters designs using comb filters," *IEEE Trans. Circ. and Syst.* **31**, pp. 913–924, Nov. 1984.
25. The Mathworks, Inc., *Signal Processing Toolbox*. Natick, MA, USA, 1983.
26. V. Liberali, R. Rossi, and G. Torelli, "A multiplierless decimation filter for $\Sigma\Delta$ A/D conversion," in *Proc. Int. Conf. on Electronics, Circuits and Systems (ICECS)*, pp. 303–306, (Dubrovnik, Croatia), Sept. 2002.
27. Xilinx, Inc., San José, CA, USA, *FPGA Device Families Data Sheets*. `http://www.xilinx.com.`