On-line Evolution of FPGA-based Circuits: A Case Study on Hash Functions

Ernesto Damiani, Andrea G. B. Tettamanzi
Università degli Studi di Milano
Polo Didattico e di Ricerca di Crema
Via Bramante 65, 26013 Crema, Italy
edamiani@crema.unimi.it, tettaman@dsi.unimi.it

Valentino Liberali
Università degli Studi di Pavia
Dipartimento di Elettronica
Via Ferrata 1, 27100 Pavia, Italy
valent@ele.unipv.it

Abstract

An evolutionary algorithm is used to evolve a digital circuit which computes a simple hash function mapping a 16-bit address space into an 8-bit one. The target technology is FPGA, where the search space of the algorithm is made of the combinational functions computed by cells and of the interconnections among cells. An experimental study is carried out to determine the best set of parameters for on-line execution. It is observed that small population size leads to more effective results when short execution time is required. An application of the evolutionary approach presented in the paper for on-line tuning of the function during cache memory operation is also discussed.

1 Introduction

Evolutionary algorithms are a broad class of optimization methods inspired by Biology, that build on the key concept of Darwinian evolution [1, 2]. After some successful applications of evolutionary algorithms to physical design (partitioning, placement and routing), now the same techniques are being considered also for structural design. This has given rise to an innovative field of research, called *evolvable hardware* [3].

Bio-inspired electronic design methods can be used in a variety of circumstances [4]. Evolutionary design automation has been applied to digital electronic design [5, 6] and has been proposed for analog circuits as well, with encouraging results [7]. The evolutionary approach to circuit synthesis is a powerful technique, which could provide innovative solutions to hard design problems, also when classical decomposition methods may fail [8].

Evolvable hardware coming of age is demonstrated by the first industrial applications having started to appear [9, 10].

The main concepts and issues relevant to evolutionary algorithms can be found, among others, in [11, 12, 13, 14, 15, 16]; [17, 18, 19] are more of a historical interest.

Starting from a previous experience on the synthesis in hardware of a digital non-linear function [20], the work presented in this paper aims at understanding how a similar method could be adapted to *on-line* reconfiguration of such circuits. By on-line, we mean that evolution takes place while the current best circuit operates. Therefore, memory and time constraints are placed on the optimization process.

In this work, circuits are evolved to compute a simple hash function mapping a 16-bit address space into an 8-bit index space as uniformly as possible, and experiments are carried out to identify a set of parameters suitable for on-line execution.

Based on the hardware architecture described in Section 3, an evolutionary algorithm (described in Section 4) is devised to evolve the hash function (Section 2). Section 5 presents experimental results, while Section 6 envisages an application of the circuit to the design of *set-associative* cache memories.

2 Problem Statement

Integrated digital electronics is traditionally considered the ideal testbed for automated design techniques. Indeed, though to this date there is no known general technique for automatically designing VLSI digital circuits satisfying given specifications, considerable progress has been made to automate the design of some classes of digital systems.

Over the last decades, the number of devices integrated into a single chip increased exponentially, according to Moore's law. Such a scenario leads to a demand for new design solutions, capable of managing increasing circuit complexity. A great deal of effort has been devoted by the research community to the development of suitable design automation tools.

The example presented in this paper deals with circuits evolved to compute a simple hash function, i.e. a many-to-one mapping of a key set into an address set, having the design property of uniform filling of the co-domain. A hash function can be used, for example, in set-associative cache memories, in order to map blocks of RAM into cache blocks, and in hardware cryptographic systems, in order to extract digital blueprints of documents.

Since it would be interesting to have the evolutionary algorithm to work in parallel with normal circuit operation (i.e. on-line), we focused our research on tuning our approach to ensure well-behaved operation in an environment where memory and time constraints are in place.

3 Hardware Environment and Behavioral Description

The hash function is implemented using an FPGA. Regularity and modularity of FPGAs, as well as their easy-to-use development tools, make them very attractive to implement evolvable digital circuits [5].

The Xilinx SRAM-based reconfigurable FPGAs have been chosen as the reference architecture for this application [21].

This choice requires that the evolved circuits comply with the constraints imposed by the target FPGA.

At this stage of our investigation, evolution is completely carried out in software on a host computer. The individuals obtained as results of the evolutionary algorithm contain all the information needed to configure the reference FPGA through the use of the XACT development system provided by Xilinx. Configuration is obtained by specifying user-designated partitioning and placement as part of the design entry process [21].

3.1 Architecture of FPGA Implementation

We consider the FPGA device XC3020, made of an array of 8×8 configurable logic blocks (CLBs) [22]. Each CLB has one output bit and four input bits. More recent FPGA families (e.g. XC4000) offer more design flexibility, both in the number and in the complexity of blocks. However, the simplicity of the XC3020 makes it the ideal candidate for an evolutionary study. Moreover, later FPGA families are fully compatible with the XC3020 device.

Since the hash function does not require any state information and therefore can be computed through a combinational circuit, the registers of the FPGA are not used. Our design employs only the combinational logic of each CLB, as illustrated in Figure 1.

The complete combinational circuit is illustrated in Figure 2. The output bits of the 8 rightmost blocks are the output of the circuit, i.e. the index computed by the hash function.

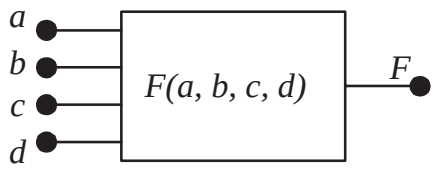

Figure 1. The elementary cell of the circuit.

In [23] we describe a comparative study of several different interconnection strategies for the circuit cells. In this paper, we consider one of them, called *gamma*, which proved to be robust and able to generate the best circuits according to the criteria adopted.

In the *gamma* circuit topology, illustrated in Figure 2, each cell can receive the outputs of the three adjacent cells in the previous column and of the cell above it in the same column, as well as two bits of the input key. The 16 external lines are driven horizontally or vertically, one for each row and one for each column. Such input lines are available to all cells of the corresponding row or column. This interconnection of external inputs exploits the routing facilities available in the FPGA.

It can be easily verified that the above connection strategy is guaranteed to generate routable circuits.

In all possible circuits, the signal flow is from left to right and from top to bottom and there is no feedback loop.

3.2 Behavioral Description

The modularity of the circuit allows us to specify the behavior simply by describing each CLB. For each block, we must specify its four inputs (which can come from input lines, from other cells, or can be grounded), and how they are combined to give the output. No global signal or supervision mechanism is used.

4 The Evolutionary Algorithm

In order to make hash circuits evolve, a very simple generational evolutionary algorithm using fitness-proportionate selection with elitism and linear scaling of the fitness has been implemented.

The overall flow of the evolutionary algorithm, which operates on an array *individual*[*popSize*] of individuals (i.e. the population) is illustrated in Figure 3; even though this is not explicitly demonstrated in the pseudo-code for sake of readability, crossover and mutation are never applied to the best individual in the population.

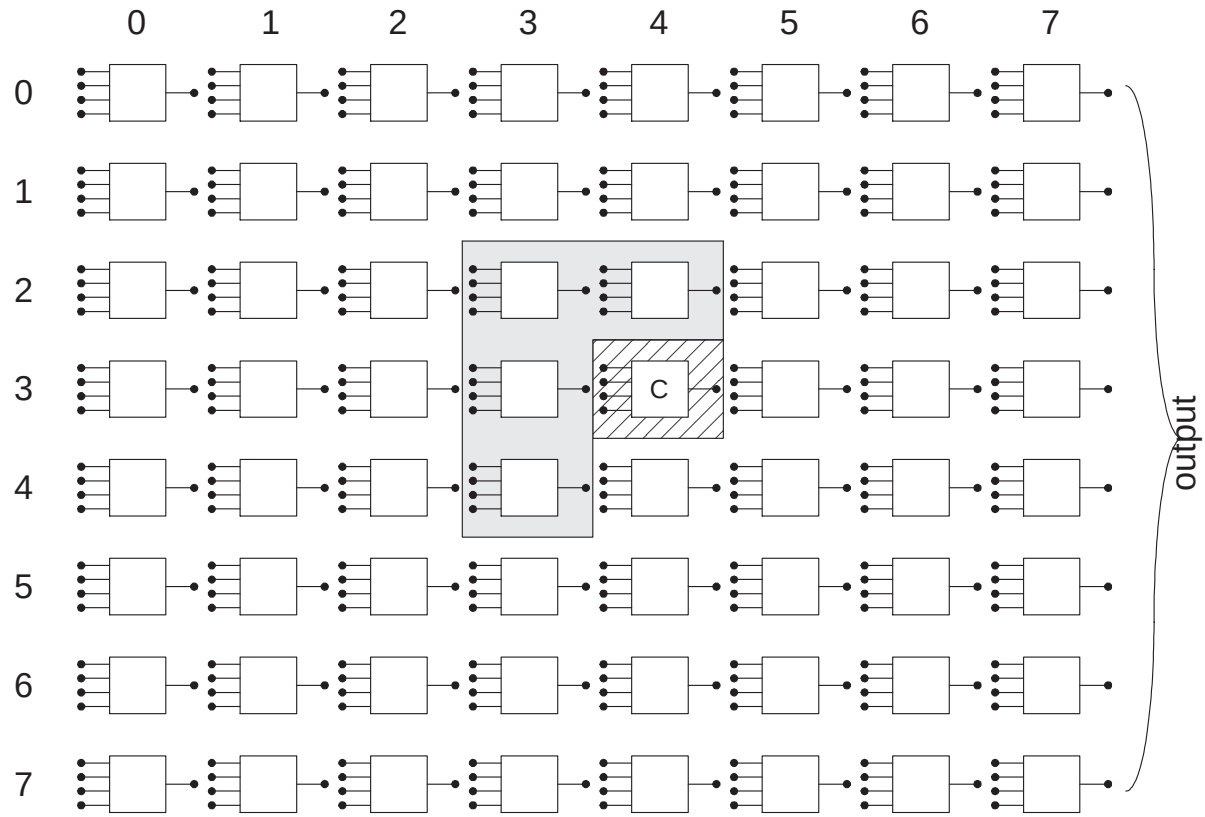

Figure 2. The reference architecture adopted for the hash circuits and an example of the "gamma" neighborhood.

The various elements of the algorithm are illustrated in the following subsections.

4.1 Encoding

An encoding was adopted that satisfies the constraints described in Section 3 by construction. Each cell is encoded through five 16-bit unsigned integers.

The first four integers define the four inputs to the cell, which can be either input lines or outputs of other cells belonging to the neighborhood. For example, cell C in Figure 2, located at row 3 and column 4, can receive inputs from cells (2, 3), (2, 4), (3, 3) and (4, 3), or from external lines 11 and 4. In addition, every input can also be grounded.

Each of the integers encoding for the cell inputs is decoded by taking the rest of the division of its value by the number of legal inputs to the cell. For cell C above, there are seven legal inputs (including the ground line).

The fifth integer encodes the truth table of the combinational network the cell implements.

Overall, a circuit is represented by a string of 5,120 bits, interpreted as a vector of $64 \times 5 = 320$ 16-bit integers. This is the genotype corresponding to a candidate solution.

4.2 Mutation

Whenever an individual is replicated to be inserted in the new generation, each bit of its genotype is flipped with the same probability p_{mut} and independently of the others.

There are some considerations to be done with respect to the encoding of cells:

1. for the truth table part, this mutation induces a small perturbation that does not disrupt the semantics of the genotype;

2. for the input selection part, flipping one or more bits amounts to replacing the old input with a new one at random, at least to a first approximation.

The mutation rate p_{mut} is dynamically adjusted in order to maintain the ratio $r = f^*/\bar{f}$, where f^* is the fitness of the

```
SeedPopulation(popSize)
generation := 0
while true do
    for i := 1 to popSize do
        EvaluateFitness(i)
    end for
    Selection
    for i := 1 to popSize step 2 do
        Crossover(i, i + 1, p_cross)
    end for
    for i := 1 to popSize do
        Mutation(i, p_mut)
    end for
    generation := generation + 1
end while
```

Figure 3. The pseudo-code of the evolutionary algorithm.

best individual and $\bar{f}$ is the average fitness of the population, as close as possible to a user-provided set-point R, which is a parameter of the algorithm. The updating rule is:

$$p_{\mathrm{mut}} \leftarrow \frac{R}{r} p_{\mathrm{mut}}. \tag{1}$$

If the distance between the best and the average circuit increases, p_{mut} is accordingly lowered, while if the average circuit is catching up with the best, p_{mut} is increased to allow for more diversity in the population.

4.3 Recombination

For recombination, each 16-bit unsigned integer is treated as an atomic unit of the genotype.

Each such unit is inherited from either parent with $\frac{1}{2}$ probability (uniform crossover). This ensures that recombination will preserve the CLBs, which are the basic building blocks of a circuit.

4.4 Objective Function and Fitness

A "good" hash function should map a set of M input keys into a set of indices $i = 1, \ldots, N$ in the most uniform way as possible. In order to assess the quality of the generated circuits, a set of keys is randomly generated over the set $\{1, \ldots, 2^{16}\}$ of possible key values. These keys are offered as inputs to each circuit and the number of hits h_i for each index i is calculated. The fitness of a circuit is then given by the following formula:

$$f = \frac{1}{1 + \frac{1}{N} \sum_{i=1}^{N} (h_i - \bar{h})^2}, \tag{2}$$

where $\bar{h} = M/N$ is the expected number of hits per index in the ideal case of uniform key distribution.

In order to evaluate the results, it is interesting to study the probabilistic behavior of a "random" hash function. By "random" we mean that the probability that a random key is mapped to any index is independent of the index and uniform over the N indices. The number of hits for all indices $i = 1, \ldots, N$, would have binomial distribution with probability

$$\Pr[h_i = h] = \binom{M}{h} \frac{(N-1)^{M-h}}{N^M}. \tag{3}$$

Therefore, the mean is

$$E[h_i] = \bar{h} = \frac{M}{N} \tag{4}$$

and the variance is

$$\mathrm{var}[h_i] = \frac{M(N-1)}{N^2}. \tag{5}$$

5 Experiments and Results

According to (2), the standard deviation σ of hits per index for a hash circuit having fitness f is

$$\sigma = \sqrt{\frac{1}{f} - 1}. \tag{6}$$

For the random hash function, the distribution of hits per index given in (3) would have mean $\bar{h}$ = 16 and variance 15.9375, or standard deviation σ_{rnd} = 3.9922. This can serve as a term for comparison to assess the effectiveness of the evolved solutions.

The scheme of a typical circuit synthesized by the algorithm is shown in Figure 4. It was obtained after 13,911 generations, and it produces a key distribution with a standard deviation of σ = 2.4812 hits per index, which is considerably better than the σ_{rnd} of the random hash function. This means that the evolved circuit is fitted to the actual set of keys used for evaluating fitness.

It has been observed that the evolution proceeds in three successive phases:

1. in the first few generations, the best combinational functions are selected for blocks;

2. in the second phase, the topology of circuits evolves in such a way that each input bit has a path through the output;

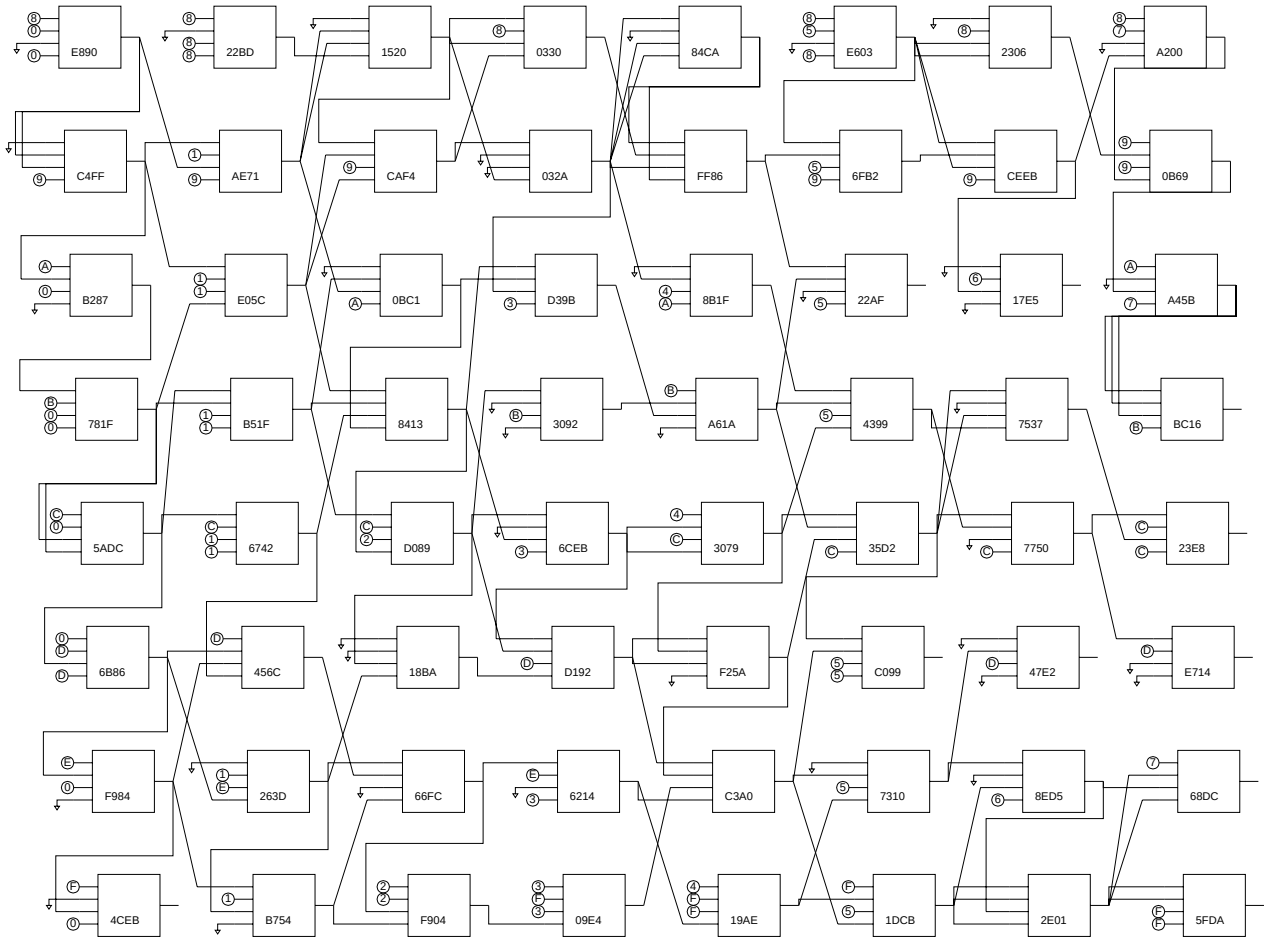

Figure 4. The best circuit produced for the "gamma" topology at generation 13,911, having fitness 0.139738.

3. in the last phase, the evolution process is slower, and tends to increase the connectivity within circuit blocks.

As shown in Figure 5, the first phase is characterised by a steep increase of fitness and by a high rate of change in the population. In subsequent phases, one can observe the typical punctuated equilibria where abrupt changes are interleaved with long periods of stasis.

Experiments have been carried out to identify the most promising parameter setting for the evolutionary algorithm in view of on-line execution.

All runs were carried out with N = 256 and M = 4,096 out of the 65,536 possible keys. This gives an average number of hits per index $\bar{h}$ = 16, regardless of the hash function employed.

First of all, the best value for set-point R was to be determined. In a previous stage of experiments, we had found that a value $R = \frac{3}{2}$ seemed to ensure best results in the off-line setting, where multiple runs can be made, from which the best circuit is extracted. Here, we desire an R that consistently provides acceptable results in a pre-determined number of evaluations in *one single* run.

Table 1 summarizes the statistics of a number of runs using four values of R, namely $\frac{5}{4}$, $\frac{3}{2}$, 2 and 3, chosen on a logarithmic scale. A population size of 100 was used, with crossover rate $p_{\mathrm{cross}} = \frac{1}{2}$.

From these experiments, it is evident that R = 2 works best for a small number of evaluations, which is what we are interested in. As the number of evaluations increases, a smaller value of R gives better results. To the upper end, the smallest pressure toward diversity seems to work best, but 100,000 evaluations amounts to too much computing time to be realistic for an on-line scenario. Therefore, in subsequent experiments, we fixed R = 2 and we considered only the first 10,000 evaluations of every run.

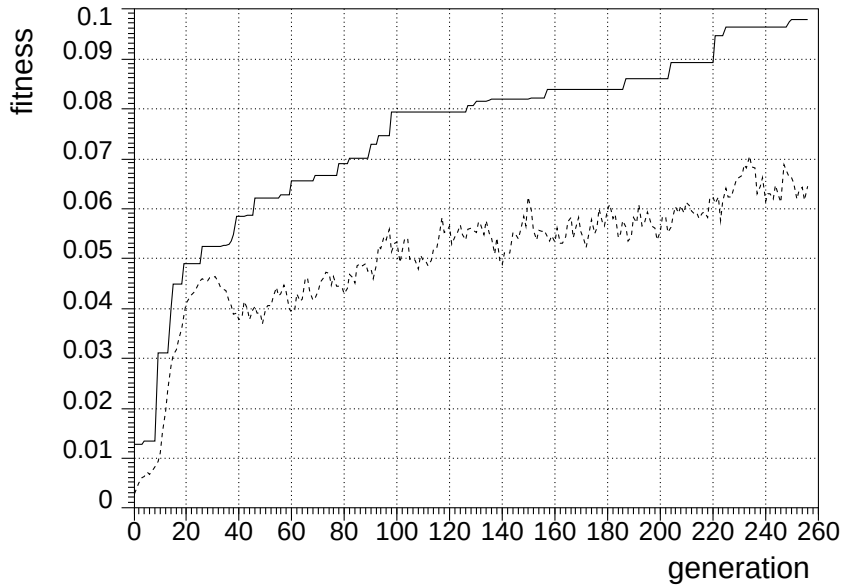

Figure 5. Average and best fitness during a sample run of the evolutionary algorithm.

An important issue, when dealing with time constraints, is to determine the optimal trade-off between population size and number of generations allowed to evolution. Of course, when the total number of fitness evaluations is bounded, the larger the population, the better is the sampling of the solution space, but the fewer the generations spanned by evolution.

Table 2 summarizes the statistics of a number of runs using three population sizes, respectively 16, 32 and 64 individuals. Once again, a crossover rate $p_{\mathrm{cross}} = \frac{1}{2}$ was used.

From these experimental data, a general result can be drawn. For a shorter computation time, and therefore a lower number of fitness evaluations, the best result is achieved with a smaller population size. This is a fortunate result, since it supports the feasibility of the proposed approach in an on-line execution setting using dedicated hardware.

6 Applications

This Section outlines a possible application for the evolutionary technique described in this paper: the adaptive design of cache memories. Commonly available cache memories are either *fully* or *set-associative* [24]. A fully associative cache memory comprises several *block frames* holding copies of main memory blocks. The cache also contains a table, holding the address of the main memory block associated to each block frame.

If a linear 32-bit address space is used for main memory and a block size of 64 kbyte is adopted, the table holds a set of 16-bit block addresses or *tags*, each coupled with the corresponding block frame identifier. Fully associative cache memories operate as follows: before the CPU performs an access to main memory, a fast combinational network simultaneously compares the 16 most significant bits of the desired address with all the block tags stored in the table. If a matching is found, the cache block frame is accessed instead of the main memory block. Otherwise, main memory is accessed and the missing block is copied in cache. The capacity of fully associative cache memories is severely limited by the topological complexity of the combinational network used to perform simultaneous comparisons. For this reason, large cache memories used in current microcomputer architectures are usually set-associative. A typical set-associative cache maps 16-bit block identifiers to a smaller set of 256 tags. This is usually done by performing associative comparison on the most significant 8 bits of the block frame identifier to select a set of blocks. The desired block is then located via linear search, usually without excessive performance degradation [25].

The circuit described in previous sections can be readily used to achieve uniform mapping of block identifiers to tags, as an alternative to associative comparison. However, it is interesting to remark that if actual memory accesses are not uniformly distributed across blocks, as it is often the case, uniform mapping is not the best choice for set-associative cache memories. Indeed, the cost of linear

Table 1. Statistics of best fitness for different values of R and different time windows (in evaluations).

Evaluations		R			
		$\frac{5}{4}$	$\frac{3}{2}$	2	3
1,000	avg	0.01351	0.03119	0.03791	0.02185
	stdev	0.01065	0.02899	0.03009	0.00963
10,000	avg	0.06974	0.07753	0.08153	0.07651
	stdev	0.01284	0.00882	0.00678	0.00508
100,000	avg	0.10448	0.10670	0.09434	0.09505
	stdev	0.00517	0.00989	0.00812	0.00766

Table 2. Statistics of best fitness for different population sizes and different time windows (in evaluations).

Evaluations		Population size		
		16	32	64
1,000	avg	0.05745	0.04727	0.03502
	stdev	0.02291	0.01447	0.02276
5,000	avg	0.08351	0.08124	0.07396
	stdev	0.00849	0.00676	0.00536
10,000	avg	0.08641	0.09034	0.08305
	stdev	0.00750	0.00606	0.00174

search would be decreased by letting several seldom consulted blocks to share a tag and countersigning frequently accessed blocks with a unique tag. The "best" mapping between tags and block sets should be dynamically searched while the circuit operates, through a second evolutionary phase of *on-line* optimisation.

7 Conclusion

This paper has described the evolutionary design of a digital circuit to compute a non-linear mapping suitable for hash and associative memories. The evolved circuit is significantly different from the ones developed using traditional design techniques, as it was already observed in the literature [8].

The feasibility of the approach in an on-line execution setting has been demonstrated and empirical relationships between the evolutionary algorithm parameters have been observed.

The experimental results presented in Section 5 allow us to conclude that no practical concern about area and cost prevents the full implementation of the proposed approach on silicon. From the architectural point of view, this option has been recently explored in [26], although further research is in order.

Acknowledgments

This work was partially supported by M.U.R.S.T. 60% funds.

The authors would like to thank Prof. Gianni Degli Antoni for his support and valuable criticism and Prof. Nello Scarabottolo for his advice on hardware applications.

References

[1] C. Darwin. *On the Origin of Species by Means of Natural Selection.* John Murray, London, UK, 1859.

[2] R. Dawkins. *The Blind Watchmaker*. Norton, New York, NY, USA, 1987.

[3] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe, and A. Stauffer, "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems", *IEEE Trans. on Evolutionary Computation*, vol. 1, no. 1, pp. 1–14, Feb. 1997.

[4] M. Sipper, D. Mange, and E. Sanchez, "Quo vadis evolvable hardware?", *Communications of the ACM*, vol. 42, no. 4, pp. 50–56, Apr. 1999.

[5] J. F. Miller, P. Thomson, and T. Fogarty, "Designing electronic circuits using evolutionary algorithms. arithmetic circuits: a case study", in D. Quagliarella, J. Périaux, C. Poloni, and G. Winter (editors), *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, John Wiley & Sons, New York, NY, USA, 1998.

[6] R. Drechsler. *Evolutionary Algorithms for VLSI CAD*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[7] J. R. Koza, F. Bennett, D. Andre, and M. Keane, "Evolution using genetic programming of a low distortion 96 decibel operational amplifier", in *Proc. ACM Symp. on Applied Computing (SAC '97)*, San José, CA, 1997.

[8] A. Thompson and P. Layzell, "Analysis of unconventional evolved electronics", *Communications of the ACM*, vol. 42, no. 4, pp. 71–79, Apr. 1999.

[9] M. Tanaka et al., "Data compression for digital color electrophotographic printer with evolvable hardware", in M. Sipper, D. Mange, and A. Pérez-Uribe (editors), *Proc. Second International Conference on Evolvable Systems (ICES '98)*, Lausanne, Switzerland, Sept. 1998, pp. 106–114.

[10] T. Higuchi and N. Kajihara, "Evolvable hardware chips for industrial applications", *Communications of the ACM*, vol. 42, no. 4, pp. 60–66, Apr. 1999.

[11] H.-P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester; New York, 1981.

[12] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA, USA, 1989.

[13] L. Davis. *Handbook of Genetic Algorithms*. VNR Computer Library. Van Nostrand Reinhold, New York, NY, USA, 1991.

[14] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, Germany, 1992.

[15] J. R. Koza. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, MA, USA, 1993.

[16] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, Oxford, UK, 1996.

[17] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, USA, 1975.

[18] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, NY, USA, 1966.

[19] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, Germany, 1973.

[20] E. Damiani, V. Liberali, and A. G. B. Tettmanzi, "Evolutionary design of hashing function circuits using an FPGA", in M. Sipper, D. Mange, and A. Pérez-Uribe (editors), *Proc. Second International Conference on Evolvable Systems (ICES '98)*, Lausanne, Switzerland, Sept. 1998, pp. 36–46.

[21] Xilinx, Inc., San José, CA. *The Programmable Logic Data Book*, 1996.

[22] J. H. Jenkins. *Designing with FPGAs and CPLDs*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1994.

[23] E. Damiani, V. Liberali, and A. Tettamanzi, "FPGA-based hash circuit synthesis with evolutionary algorithms", to appear in *IEICE Transactions on Fundamentals* (Sept. 1999).

[24] A. Smith, "Cache memory design: an art evolves", *IEEE Spectrum*, vol. 24, 1987.

[25] W. Burkardt, "Locality aspects and cache memory utility in microcomputers", *Euromicro Journal*, vol. 26, 1989.

[26] I. Kajitani et al., "A gate-level EHW chip: Implementing GA operations and reconfigurable hardware on a single LSI", in M. Sipper, D. Mange, and A. Pérez-Uribe (editors), *Proc. Second International Conference on Evolvable Systems (ICES '98)*, Lausanne, Switzerland, Sept. 1998, pp. 1–12.