On-chip and On-line Self-Reconfigurable Adaptable Platform: the Non-Uniform Cellular Automata Case

Andres Upegui, Eduardo Sanchez

Ecole Polytechnique Fédérale de Lausanne, Logic Systems Laboratory,
1015 Lausanne, Switzerland
(andres.upegui, eduardo.sanchez)@epfl.ch

Abstract

In spite of the high parallelism exhibited by cellular automata architectures, most implementations are usually run in software. For increasing execution parallelism, hardware implementations on FPGAs have been proposed, under the cost of being un-flexible, and inefficient in terms of resource utilization. In this paper we present a platform for evolving CA by exploiting the partial re-configurability of current commercial FPGAs. Our implementation includes an on-chip soft-processor that generates a partial bitstream, reconfigures the FPGA, and computes the fitness. After finding a good individual, the evolved CA can be used as a peripheral for performing useful computation. As case study we present CA co-evolution for a random number generator and for the firefly synchronization problem.

1 Introduction

Designing analog and digital electrical circuits is, by tradition, a hard engineering task, vulnerable to human errors, and no one can guarantee the optimality of a solution. Design automation has become a challenge for tool designers, and given the increasing complexity of circuits, higher description levels are needed. Evolvable Hardware (EHW) arises as a promising solution to this problem: from a given behavior specification of a circuit, an evolutionary algorithm (EA) can find a circuit able to satisfy the specification.

EAs take inspiration from the principles of biological evolution decoding a phenotype from a genotype. The genotype is a number string, where the genetic operations, reproduction and mutation, are applied. Reproduction is performed by genome crossing and mutation is performed in a probabilistic way. In the case of EHW, a phenotype is decoded from this genome for obtaining a circuit with a given set of components and connectivity. A fitness note is assigned to this individual given the performance exhibited. EHW have shown to perform well finding solutions from simple Boolean functions to complex analog circuits, sometimes performing better than engineered solutions.

The hardware substrate supporting the evolution is one of the most important initial decisions to make when evolving hardware. The hardware architecture is closely related with the type of solution being evolved. Hardware platforms have, in most cases, a cellular structure composed of uniform or non-uniform components. In some cases one can evolve the components functionality, in others the connectivity, or, in the most powerful ones, both. Field Programmable Gate Arrays (FPGAs) fit well for this third category: they are composed of configurable logic elements interconnected by configurable switch matrices.

In this paper we present a novel system approach for evolving hardware. The main novelty of the proposed system consists on the mapping from the genotype to the phenotype: the genome directly determines the hardware configuration implementing the rule function by partially reconfiguring the hardware substrate supporting the CA. All this performed by a system on chip, allowing an on-chip and on-line self-reconfigurable adaptable system.

2. Evolvable Hardware Platforms

When evolving hardware, there is a first main issue to address: the hardware substrate supporting the evolved circuit. Custom evolvable chips use to provide dynamic and partial reconfiguration, dispose of multi-context configuration memories and can be configured with random bitstreams. The commercial options main advantage is the absence of non-recurrent engineering, as any general purpose architecture, under the cost of reduced flexibility and performance.

1-4244-0054-6/06/$20.00

Among commercial options, the obsolete FPGA XC6200 from Xilinx constituted the perfect platform for intrinsic evolvable hardware; it was possible to download any arbitrary bitstream without risking contentions. Maybe the most known work using these devices is that of Adrian Thompson [1] who evolved analog circuits, by exploiting the dynamics inherent to the physical properties of the FPGA internal components.
More recent work on evolvable circuits on commercial FPGAs has focused on Virtex architectures from Xilinx. The interest on these devices is their partial dynamic reconfigurability, with the limitation that no arbitrary configuration bitstreams can be loaded. Anyway, in [2] there are presented 3 techniques for EHW on Virtex families with coarse and fine grained level solutions.

3. Dynamic Partial Reconfiguration on Xilinx FPGAs

FPGAs are programmable logic devices that permit the implementation of digital systems. They provide an array of logic cells that can be configured to perform a given function by means of a configuration bitstream. Some FPGAs allow performing partial reconfiguration, where a reduced bitstream reconfigures only a given subset of internal components. Dynamic partial reconfiguration (DPR) is done while the device is active.
Xilinx FPGAs configuration bitstream is composed by frames. A frame constitutes the minimum configuration information that can be modified on these devices. For Xilinx FPGAs there are two documented flows to perform DPR: Module Based and Difference Based [3].
Even if only these flows are supported by the FPGA vendor, other approaches have been proposed. Self-reconfigurable platforms generate a special interest on the field, given the autonomy they provide. Virtex II FPGAs include an Internal Access Configuration Port (ICAP), which allows reading and writing the configuration bitstream from inside the FPGA. This ICAP allows an on-chip processor to self-reconfigure the FPGA supporting it. Self-reconfigurable platforms modify the system by re-configuring the FPGAs with partial bitstreams. The main drawback of these partial bitstreams is that they must be pre- placed and routed on a workstation, restricting the number of reconfigurable systems to a predefined value.
An attempt for allowing a platform to self-reconfigure with a design description conceived on the fly has been proposed [4]: XPART (Xilinx Partial Reconfiguration Toolkit) is an application program interface (API), for Xilinx embedded processors, that provides methods to read and modify selected FPGA resources by using the ICAP. Anyway, XPART was never released.

4. Cellular Programming

Cellular Automata (CA) are discrete time dynamical systems, consisting on an array of identical computing cells [5]. A cell is defined by a set of discrete states, and a rule for determining the transitions between states. On the array, states are synchronously updated according to the rule, which is function of the current state from the cell itself and the states of the surrounding neighbours.
Non-uniform CA differ from their uniform counterpart in the state transition rule diversity exhibited by the non-uniform ones. Uniform CA constitute a sub-set of non-uniform CA, making the non-uniform ones a more general and powerful platform. On the same way, this power improvement is compensated with a higher difficulty when designing them. That's the reason why evolutionary techniques have been used for finding non-uniform CA state transfer rules[6, 7]. Several evolutionary algorithms have been used for non-uniform CA: mainly genetic algorithms [7] and cellular programming [6, 8, 9].
In cellular programming each cell's state transfer rule is coded as a bit-string, most known as a genome. This genome implements a rule for computing the next state. Each genome is, thus, composed of 8 bits for CA with neighborhood radius r = 1. Instead of using a population of CA as genetic algorithms, the cellular programming approach involves a single, non-uniform CA. This fact implies that the final solution would not be an individual selected from a population (like on GAs), but the population itself.
When running the algorithm, initial cell rules are initialized at random. Then, initial states are equally randomly initialized; we let the CA run for M iterations, and we repeat it for $C=300$ different initial states. There is not a global fitness, as in genetic algorithms, but a local fitness for each automaton. Each cell's fitness is accumulated for the C state initializations, according to a performance measure according to the behaviour desired.
After computing the fitness, the genetic operators (reproduction, crossover, and mutation) are applied to genomes. In this algorithm, evolutionary operators act on a local manner, by limiting the reproduction and crossover operators to use genomes from neighbour cells (see more details in [8, 9]).

5. The evolvable platform

In this section we present a platform able to self-reconfiguring non-uniform CA state transitions through the ICAP. Our platform consists on a Microblaze soft-processor running on a Virtex-II FPGA from Xilinx

5.1. General System Description

The complete system schematic is depicted in figure 1. A Microblaze soft-processor from Xilinx runs an evolutionary algorithm. The program is stored on an internal BRAM, and an external SRAM is used for data storing – i.e. genome storing in this case. The system interfaces through an UART peripheral with the external world. The one-dimensional (1-D) CA to be evolved can be accessed for reading or for writing the states through general purpose I/O interfaces, anyway rule modifications are exclusively performed by the HWICAP peripheral. The HWICAP module allows the Microblaze to read and write the FPGA configuration memory through the Internal Configuration Access Port (ICAP) at run time, enabling our evolutive algorithm to modify the circuit structure and functionality during the circuit's operation.

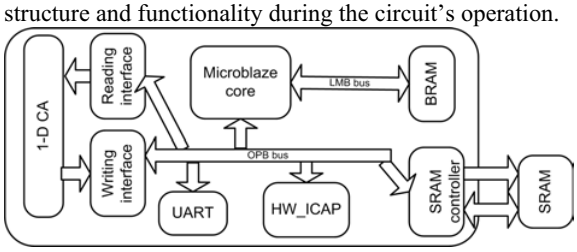

Figure 1. System schematic.

5.2. Cellular Automata Implementation

In this work we concentrate on 1-D grids, with a number of states per cell $k=2$, denoted 0 and 1. In such CA each cell is connected to r local neighbours (cells) on either side, as well as to itself. Where r is a parameter referred to as the radius (each cell has $2r + 1$ neighbours). In our case, the radius $r=1$, thus the neighborhood equals 3.

A 1-D CA composed of 50 automata is included; it can be configured for running on free-run mode – i.e. a state update at each clock cycle – or on controlled iterative mode. An initial state for the CA can be configured through the *writing interface*, while the full state can be read by the *reading interface*.

We have focused our interest on 1-D CA, with $k=2$ and $r=1$, given their analogy with FPGAs basic elements (LUTs and flip-flops). Such automaton implemented in hardware would require a flip-flop, for storing the current state, and a 3-input LUT. The most basic logic cell of Virtex-II FPGAs is a slice, which contains 2 flip-flops and 2 4-input LUT, fitting well for implementing two of the above described automata.

Hard macros allow specifying the exact placement of a desired component on a design. In [10] hard macros are used for instantiating fuzzy rules, which are then evolved from a PC. In our case, we design a hard macro consisting in a slice containing 2 automata; the 2-CA hard macro is depicted in figure 2. Then we instantiated a 1-D automata with size 50 – i.e. 25 hard macros .

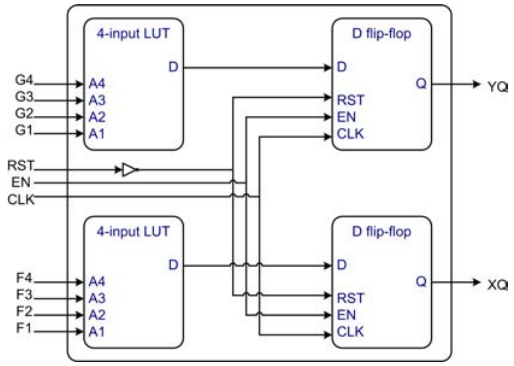

Figure 2. 2-CA hard macro.

As described in [2], one can access the LUT configuration of a whole column of slices in a single configuration frame. That's the reason why we dispose the set of 25 hard macros on a single column. Then, just by reading and writing a single frame one can evolve the configuration bitstream containing the LUTs' functions. By using this approach on an FPGA Virtex-II 1000 we can evolve a CA with up to 160 cells by just modifying a single frame.

6. Experimental Setup and Results

Two problems were chosen for validating our platform: firefly synchronization [8] and a random number generator [9]. In both cases we used the same cellular programming algorithm described in section 4.

6.1. Firefly synchronization

In CA, the firefly synchronization problem consists on synchronizing the firing of a set of 2-state automata. CA are initialized at a random configuration, and after a number of iterations each automaton must swap its state synchronizing with its neighbors.

For the cellular programming approach, we initialize the genome for every cell in a random way, and through the HWICAP peripheral we map the genome contents on the frame containing the LUT contents.

Once the frame is re-configured, one can test the CA through the reading and writing interfaces. A random initial state is loaded on the CA, and we let it run for 54 iterations. The fitness is computed by the Microblaze soft-processor, by reading the CA state. For computing the fitness we let it execute four more iterations: if the sequence is 0-1-0-1 the fitness is 1, otherwise it is 0. The total fitness is the accumulation of the fitness of 300 runs.

Then a new genome for each cell is generated as described in section 4. Our platform achieves to successfully finding genomes able to synchronize the switching of the states, as well as described in [8].

6.2. Pseudo-random number generator

Good random number generators are mainly consequence of natural physical processes. Pseudo-random number generators are commonly used by information systems: starting from a seed value, a non-linear transformation is applied for simulating real random number generators. Measuring the quality of a given transformation function is difficult; however, a simple and effective way of doing it is to use the entropy of the generated sequence [11].

For the pseudo-random number generator we use the CA described in the previous section, as well as the same cellular programming algorithm (excepting the fitness function). In [9], Sipper and Tomassini evolved a random number generator in a 1-D 50-cell CA using cellular programming. We implement the same algorithm, with the difference that we do not read a value at each CA update, but we let the CA running in free-run mode.

The fitness computation consists on:

- Partial configuring the FPGA with a given CA,
- Random initialization of states and sampling of 4096 consecutive values.
- Compute entropy of the system as the mean entropy for each bit subsequence, with the expression:

$$E_h = \frac{\sum_{i=1}^{n} E_h^i}{n} \quad \text{with:} \quad E_h^i = -\sum_{j=1}^{k^h} ph_j^i \log_2(ph_j^i)$$

  Being n the number of cells, h the subsequence length, and E_h^i the entropy for the cell i considering a subsequence length h, ph_j^i is the probability of obtaining a given subsequence j on the cell i when the subsequence length is h.
- Repeat, from the initialization, 300 experiences; the fitness is computed as the average value.

In our experiments we considered a subsequent length value $h=4$ allowing a maximal theoretical value of entropy $E_h=4$. In [9] it is reported a maximum fitness of 3.997; However they do not specify how many evolutions were performed before arriving to such solution. The maximum fitness obtained by our platform after running 20 evolutions is 3.963.

7. Conclusions

The methodology proposed in this paper deals with useful issues when evolving hardware in a general way. Performing on-chip evolution on reconfigurable platforms has always been an important challenge, and this paper describes how to do it, in an efficient way, on nowadays commercial devices.

We have presented a novel system approach for evolving hardware. Our platform has shown to be suitable for evolving non-uniform CA, and the same approach can be easily extended to other cellular structures – like artificial neurons or fuzzy rules – just by defining their respective hard macro. The system on chip supporting these reconfiguration capabilities provides the hardware platform to support the previously called *on-chip and on-line self-reconfigurable adaptable systems*, by providing the flexibility needed by a real phenotype modification on the evolved hard individual.

References

[1] A. Thompson, "An evolved circuit, intrinsic in silicon, entwined with physics", *Evolvable Systems: From Biology to Hardware, LNCS*, vol. 1259, pp. 390-405, 1997.

[2] A. Upegui and E. Sanchez, "Evolving hardware by dynamically reconfiguring xilinx FPGAs", *Evolvable Systems: From Biology to Hardware, LNCS*, vol. 3637, pp. 56-65, 2005.

[3] Xilinx_Corp., "XAPP 290: Two Flows for Partial Reconfiguration: Module Based or Difference Based": www.xilinx.com, Sept, 2004.

[4] B. Blodget, P. James-Roxby, E. Keller, S. McMillan, and P. Sundararajan, "A self-reconfiguring platform", *Proceedings of Field-Programmable Logic and Applications, LNCS*, vol. 2778, pp. 565-574, 2003.

[5] S. Wolfram, *A new kind of science*. Champaign, IL: Wolfram Media, 2002.

[6] M. Sipper, "Co-evolving non-uniform cellular automata to perform computations", *Physica D-Nonlinear Phenomena*, vol. 92, pp. 193-208, 1996.

[7] M. Mitchell, J. P. Crutchfield, and P. T. Hraber, "Evolving Cellular-Automata to Perform Computations - Mechanisms and Impediments", *Physica D*, vol. 75, pp. 361-391, 1994.

[8] M. Sipper, M. Goeke, D. Mange, A. Stauffer, E. Sanchez, and M. Tomassini, "The firefly machine: online evolware", *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 181-186, 1997.

[9] M. Sipper and M. Tomassini, "Generating parallel random number generators by cellular programming", *International Journal of Modern Physics C*, vol. 7, pp. 181-190, 1996.

[10] G. Mermoud, A. Upegui, C. A. Pena, and E. Sanchez, "A dynamically-reconfigurable FPGA platform for evolving fuzzy systems", *Computational Intelligence and Bioinspired Systems, LNCS*, vol. 3512, pp. 572-581, 2005.

[11] J. R. Koza, *Genetic programming : on the programming of computers by means of natural selection*. Cambridge, Mass.: MIT Press, 1992.