Reconfigurable Hyper-Structures for Intrinsic Digital Circuit Evolution

S. Kazarlis, J. Kalomiros, V. Kalaitzis, D. Bogas,
P. Mastorokostas, A. Balouktsis
Dept. of Informatics Engineering
Technological Educational Institute of Central Macedonia,
62124 Serres, Greece
email: kazarlis@teicm.gr, ikalom@teicm.gr

V. Petridis
Dept. of Electrical and Computer Engineering
Aristotle University of Thessaloniki,
54124 Thessaloniki, Greece
email: petridis@eng.auth.gr

***Abstract*— A workbench for intrinsic evolution of digital circuits is presented, based on a Cartesian Genetic Programming algorithm running on a personal computer and a reconfigurable platform suitable for run-time reconfiguration. Two types of Cartesian cell structures are proposed, based on a cylindrical interconnection grid. In addition to a feed-forward network, the cylindrical grid can allow feedback loops as well. The proposed structures are combined with dedicated communication and control logic, producing automatically a fitness result for each circuit configuration. The proposed system is tested with known digital circuits and evaluated in terms of resource usage and configuration speed.**

Keywords - Evolvable Hardware; intrinsic evolution; reconfigurable hardware; Cartesian structures;

I. INTRODUCTION

A lot of research has been directed in recent years towards the study of evolvable hardware (EHW), which is a field of evolutionary computation that employs evolutionary algorithms for the building of electronic circuits [1]-[3]. Evolvable hardware is an offspring of Genetic Programming, an evolutionary technique originally proposed for the evolution of software. In EHW, the circuits are encoded into genotypes, traditionally using tree structures, and more recently using Cartesian lattices or other forms, like binary strings. From the genotype the actual circuit or phenotype is constructed and tested, either in a simulator, as in the case of extrinsic evolution [4] [5] or in a reconfigurable device, as in intrinsic evolution [6]-[8]. Evolvable hardware can have a number of important applications, most notably in the automatic design of adaptive and fault-tolerant systems [3] and in the design of digital circuits, where new unconventional forms of known circuits can be found and new design principles can be derived [9] [10].

A variation of Genetic Programming, called Cartesian Genetic Programming (CGP), encodes a digital circuit as a directed graph, where functional units are represented by a rectangular array of nodes connected together to perform a computational task on binary input data [9] [11]. The genotype is a binary string that represents connections and gate functions. Based on this concept evolvable hardware platforms have been proposed, both for extrinsic and for intrinsic evolution of digital circuits [5] [8] [9]. Also, following the notion of a Cartesian node array, a new type of reconfigurable platform has been introduced, the Virtual Reconfigurable Circuit, or VRC [12] [13]. A VRC is a new reconfigurable device realized on top of an ordinary Field Programmable Gate Array (FPGA), consisting of an array of Programmable Elements, interconnection network and configuration memory, all implemented on the available resources of a common FPGA device. The VRC concept has been utilized for the evolution of combinational circuits [14], and the evolution of components for image and signal processing [8].

The simple merit of such circuits is that while they adhere to the basic LUT cell structure of an FPGA chip, they are still open to full run-time reconfiguration by the user, through well determined configuration rules set by the matrix designer. In this way, the VRC reconfiguration circumvents the need for low-level configuration. The latter requires complicated low-level knowledge of the particular FPGA chip and the development of custom compilation tools. Both tasks are daunting and are usually hindered by undisclosed information or by the advent of new devices that revolutionize the field.

In this paper, a workbench for intrinsic digital evolution experiments is designed and implemented in a Field Programmable Gate Array. The system includes a host computer running a genetic programming application and a communication channel that allows the run-time reconfiguration of the evolvable platform. The configuration string is composed of the genotype encoded according to the CGP principles, while the phenotype is implemented and evaluated in the reconfigurable device.

The concept implemented in the proposed workbench is based on reconfigurable hyper-structures following the general idea of the VRCs. They form two-dimensional arrays of cells, which are interconnected with a predefined fixed or programmable switching array. The proposed structures adhere to specific interconnection properties derived from a cylindrical interconnection grid. In addition to the feed-forward network, the cylindrical grid can allow feedback loops as well. The proposed Configurable Cylindrical Structures or CCS are combined with custom

 ISBN: 978-1-61208-430-5

communication and control logic, implemented as finite state machines. The peripheral logic allows communication with a PC host application over a serial port. An embedded register file is used in order to store the configuration values. Additional logic automatically produces a fitness result for each circuit configuration. The controller returns this fitness result to the host computer and the host CGP application proceeds to reconfigure the CCS.

In this preliminary phase, the proposed workbench is tested using configuration strings corresponding to typical test-benches for evolutionary design. The overall time for CCS configuration and fitness response is measured as a function of CCS dimensions. The required FPGA resources for the implementation of the CCS are also measured as a function of circuit complexity. In this way, the suitability of the proposed workbench for intrinsic evolution experiments is evaluated.

The remaining of the paper is organized as follows. In Section II, two alternative CCS circuits are reported and their differences are discussed. In Section III, the overall architecture, including the dedicated controllers and fitness logic, is presented. In Section IV, test configurations are conducted and evaluation results are reported, while in Section V, the paper is concluded.

II. THE CONFIGURABLE CARTESIAN STRUCTURES

A. *CCS-1: A feed-forward Cartesian structure*

The proposed configurable structures are developed as parameterizable blocks using the hardware description language VHDL, where external parameters are the required number of rows and columns in the Cartesian structure and the number of inputs and outputs in the CCS device. In Fig. 1, the first hyper-structure (CCS-1) implemented in the proposed workbench is presented. It is a two-dimensional lattice of two-input one-output cells connected with a fixed feed-forward interconnection grid. Each output can feed two separate forward inputs. In addition, the interconnection grid has a cylindrical structure, meaning that the lower-row cells are seamlessly interconnected with the upper-row cells. As a result, all cells of the hyper-structure receive inputs adhering to the same interconnection rules and the structure can automatically expand using a FOR GENERATE statement in VHDL.

The first column in the design of Fig. 1 is a set of multiplexers the role of which is to distribute the input signals to the front-end cells. There are two *p*-input multiplexers per cell, where *p* is the number of inputs of the target circuit. Depending on the required number of outputs *q*, *q* N:1 multiplexers in the output stage select one among the N possible outputs.

Each cell is composed by a 2-input LUT implemented by a four to one multiplexer, as shown in Fig. 2. The LUT is able to implement in total sixteen different two-input functions, including the basic digital gates.

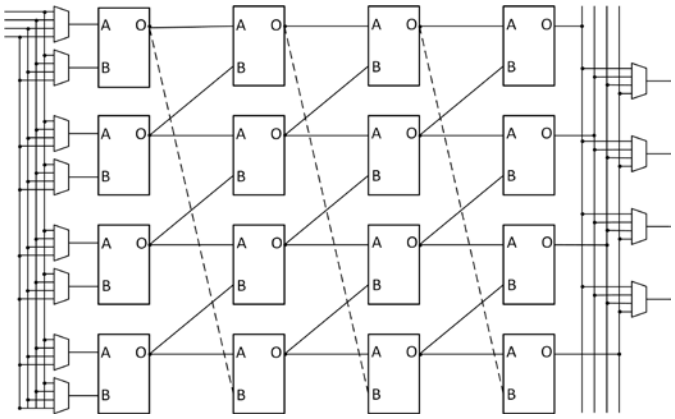

Figure 1. A simple 4x4 Cartesian structure (CCS-1) with a fixed grid of interconnections.

An embedded register file is used in order to store the configuration scheme. The cell can easily be enhanced, in future expansions, to include a flip-flop in each cell, for sequential circuit design. A 4-bit register, where the configuration bits are stored, corresponds to each cell in the hyper-structure. Additionally, configuration registers are attributed to the selection bits of the input multiplexers. The register file is rewritten during reconfiguration at run-time, at all instants when the genetic algorithm updates the evolving circuit. Table I presents all possible gates and logic functions that a cell can implement, along with their corresponding binary configuration patterns. *A* and *B* are the cell inputs. In order to configure the four-input, four-output 4x4 lattice of Fig. 1, a total of eighty eight configuration bits is required nominally. These bits are distributed between the selection bits of the eight 4:1 input multiplexers (2x8=16 bits), the sixteen lattice cells (4x16=64 bits) and the four output multiplexers (2x4=8 bits).

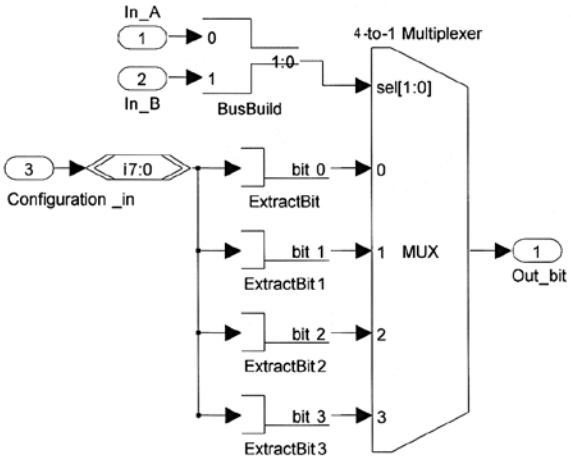

Figure 2. Four–to-one multiplexer implementing the 2-input LUT for each cell of the proposed hyper-structure.

TABLE I. THE SIXTEEN LOGIC FUNCTIONS CORRESPONDING TO 4-BIT CONFIGURATION PATTERNS

X3	X2	X1	X0	Implemented logic	Boolean function
0	0	0	0	*Always outputs zero*	$F = 0$
0	0	0	1	*F=A* NOR *B*	$F = \overline{A + B}$
0	0	1	0	*F=A* AND NOT*(B)*	$F = A \cdot \overline{B}$
0	0	1	1	*F*=NOT*(B)*	$F = \overline{B}$
0	1	0	0	*F*=NOT*(A)* AND *B*	$F = \overline{A} \cdot B$
0	1	0	1	*F*=NOT(*A*)	$F = \overline{A}$
0	1	1	0	F=A XOR B	$F = A \oplus B$
0	1	1	1	F=A NAND B	$F = \overline{A \cdot B}$
1	0	0	0	F=A AND B	$F = A \cdot B$
1	0	0	1	F=A XNOR B	$F = \overline{A \oplus B}$
1	0	1	0	Transfers A	$F = A$
1	0	1	1	If B then F=A	$F = A + \overline{B}$
1	1	0	0	Transfers B	$F = B$
1	1	0	1	If A then F=B	$F = \overline{A} + B$
1	1	1	0	F=A OR B	$F = A + B$
1	1	1	1	*Always outputs 1*	$F = 1$

The configuration file increases according to the dimensions of the Cartesian structure and the number of inputs and outputs. In the present implementation, the register file consists of 8-bit registers, since they are compatible with 8-bit communication over the serial port. The proposed register file architecture is shown in Fig. 3. Following this scheme, the configuration of the hyper-structure of Fig. 1 requires four bytes for input routing and sixteen bytes for cell configuration. If the circuit produces two outputs, then two additional bytes are needed. In this, way, the configuration file includes many redundant bits which however can be used in future expansions. For example, attributing one byte to each pair of input multiplexers, allows for up to four useful selection bits or up to sixteen input channels. This is more than the number of inputs required in most of our present evolution tests. Also, according to Fig. 3, one 8-bit register is attributed per lattice cell. Although only the four lower bits are useful in the present design, the higher bits can be used in later upgrades in order to support function generators with 3-input LUTs. The role of input, output and configuration bits in the basic 2-input LUT cell is shown in Fig. 2.

B. CCS-2: A more General Cartesian Structure

An alternative Cartesian Structure (CCS-2) is presented in Fig. 4. The configurable cells belong again to an NxM lattice; however the interconnection grid is more flexible than that of CCS-1, since it is implemented by multiplexers allowing sets of predefined connections. The output of each cell can be selected to provide input to four different neighboring cells, namely to three forward cells in the next column and to the adjacent cell on the row below.

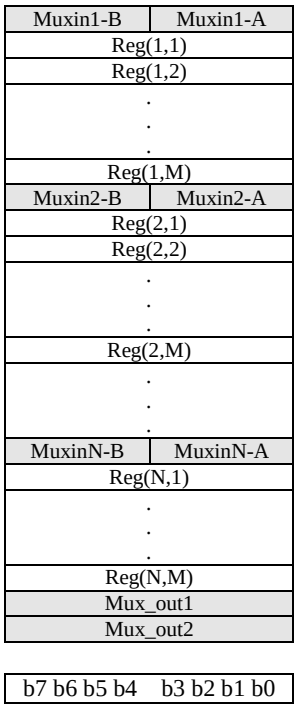

Figure 3. Architecture of the 8-bit register file used for the configuration of the CCS structure of Fig. 1. Indices correspond to the cells of the 2D lattice.

Each cell input can be connected to one of two possible outputs. The selection process is achieved by two-to-one multiplexers. The interconnection grid has again a cylindrical structure as indicated by the arrows in Fig. 4. In this case, the cylindrical interconnections allow the creation of feedback loops, since an output can be transferred through a column and return as input to the same cell. For example, the output of cell 3 can go through cells 7, 11, 15 and return as input to cell 3.

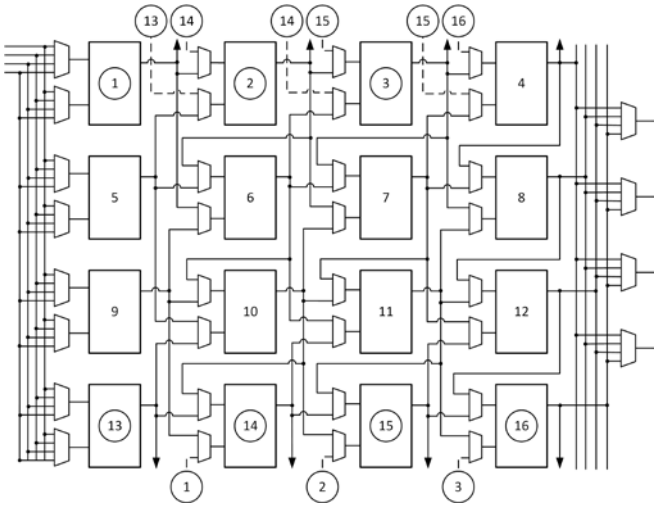

Figure 4. The hyperstructure CCS-2. Two-input multiplexers are used for the routing of interconnections between cells.

Muxin1-B		Muxin1-A
muxb-1,1	muxa-1,1	Config(1,1)
muxb-1,2	muxa-1,2	Config(1,2)
.		
.		
.		
muxb-1,M	muxa-1,M	Config(1,M)
Muxin2-B		Muxin2-A
muxb-2,1	muxa-2,1	Config(2,1)
muxb-2,2	muxa-2,2	Config(2,2)
.		
.		
.		
muxb-2,M	muxa-2,M	Config(2,M)
.		
.		
.		
MuxinN-B		MuxinN-A
muxb-N,1	muxa-N,1	Config(N,1)
.		
.		
.		
muxb-N,M	muxa-N,M	Config(N,M)
Mux_out1		
Mux_out2		

b7 b6	b5 b4	b3 b2 b1 b0

Figure 5. Architecture of the 8-bit register file used for the configuration of CCS-2. Indices correspond to the cells of the 2D lattice.

The register file employed for the configuration of CCS-2 is shown in Fig. 5. In this file, each 8-bit cell register is divided in a four-bit nibble for cell configuration (b3 down to b0) and a nibble for multiplexer configuration (only bits b4 and b6 are used). Input multiplexers have dedicated registers at the beginning of each row, while first column cells use only the lower nibble of a configuration register.

Variations of the above Cartesian structures can lead to a trade-off between interconnection flexibility and reduced complexity. More interconnection options increase the possibility to reach a solution. At the same time, the search space is expanded and complexity is increased. A fixed interconnection grid can reduce complexity for some problems but it may also require a larger grid in order to implement a solution.

III. CONTROLLER ARCHITECTURE

The main setup of the proposed workbench consists of a PC running the evolutionary algorithm and an FPGA board. The FPGA is configured to implement the CCS design and supportive hardware logic for configuration, testing and control. A dedicated custom controller and datapath was designed for the configuration of the Cartesian structure in the FPGA device. The datapath includes logic which produces the test patterns for the evaluation of each circuit configuration and returns a fitness result to the evolutionary algorithm. The overall architecture is based on 8-bit registers and is presented in Fig. 6. It includes a UART peripheral controller supporting communication with the host PC application over the serial port and a streaming controller implementing the algorithmic steps of the configuration and testing procedure, in the form of a finite state machine. Other system blocks are the register file for the storage of configuration data, the CCS structure which is configured by the evolutionary algorithm and a "ground truth" block, where the target logic is implemented. Finally, a computational block extracts the Hamming distance between the truth tables of the target logic and the CCS logic under test.

The heart of the system is the streaming controller. It produces clock and control signals to all other blocks and makes data available to other blocks through the system bus. It can initiate a UART "receive" or "transmit" operation, it clocks successive test inputs to the CCS and returns the Hamming distance to the host computer, as a fitness result

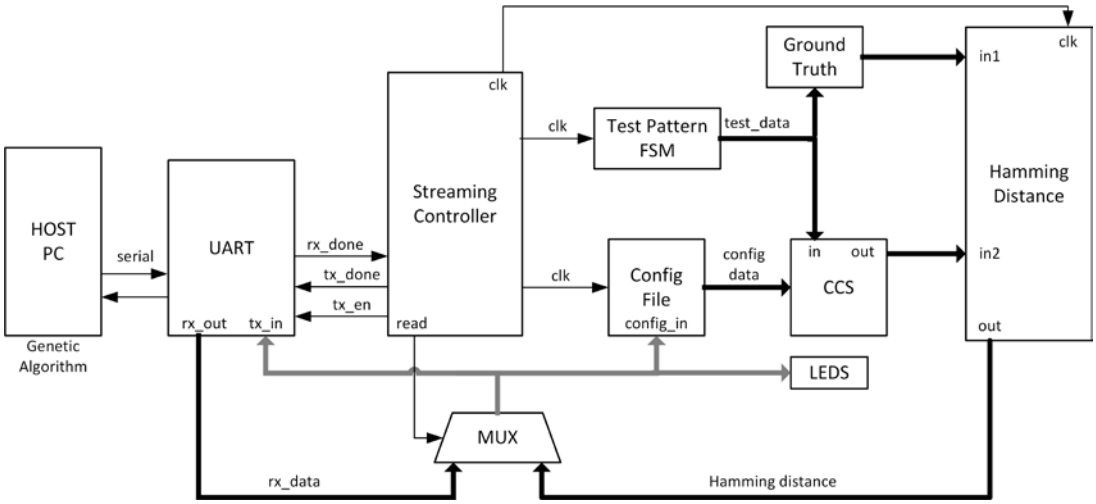

Figure 6. Block diagram of the implemented system architecture.

for circuit evaluation. Then, the genetic algorithm evaluates the result and produces a new genotype in the form of a new configuration array. This procedure is repeated until the genetic algorithm reaches a predefined number of generations. Fig. 7 presents the basic state diagram of the streaming controller, between successive configurations. At the beginning, the controller is at the "idle" state waiting for a protocol character, signaling the beginning of a configuration stream. The controller enters the "receive" state and counts the number of received data. It repeats the reception until all expected data in the configuration array have been received. Then, a "test" process begins, where the controller employs a finite state machine in order to create successive test patterns as input to the CCS and the ground truth blocks. At each repetition, a clock pulse is sent to the Hamming distance block, where the Hamming distance is accumulated. When all test patterns have been tested, the total Hamming distance is transmitted back to the computer via the serial port. The controller returns to the "idle" state waiting for a new configuration array.

IV. TESTS AND EVALUATION

At the present stage, the proposed workbench is used to configure a number of test circuits in the CCS. The system is evaluated in terms of the required hardware resources and total response time. The response time is significant in evolution experiments, since the configuration cycle is repeated for hundreds of thousands times.

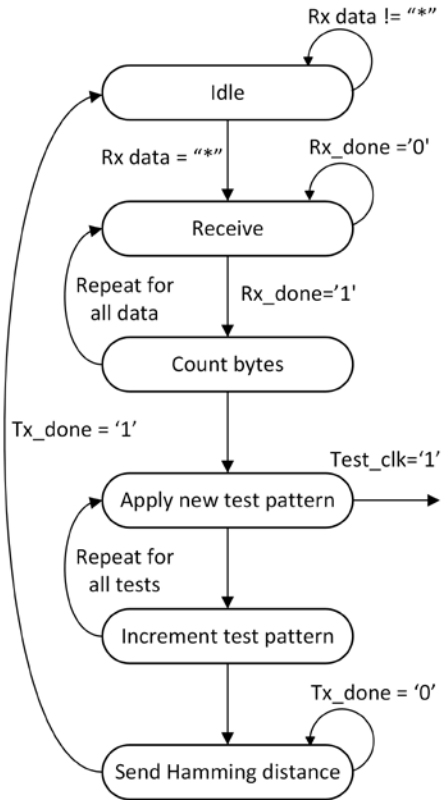

Figure 7. State diagram of the implemented controller.

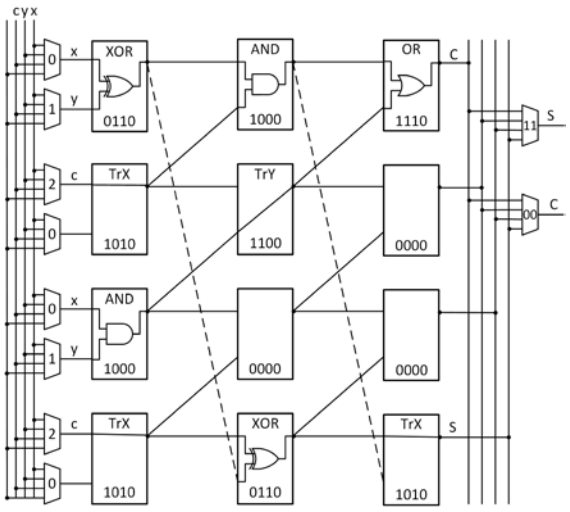

Figure 8. Example configuration of the full-adder, implemented with the Cartesian structure of Fig. 1 (CCS-1).

The structures were verified with a number of test configurations. The following widely used test circuits were implemented: *a.* the half adder, *b.* the full adder, *c.* the 2:4 binary decoder, *d.* the 2:1 and 4:1 multiplexer, *e.* the 2-bit multiplier. These circuits can be effectively implemented by both hyper-structures employing grids of variable sizes. The possibility for feedback loops in CCS-2 can be used to implement latches. The list of our test-circuits is therefore concluded with *f.* the S-R latch *g.* the D-latch.

An interesting implementation is that of the full adder. CCS-1 can implement the full adder using a 4x3 cell grid configured as shown in Fig. 8. Several cells are configured as "transfer" gates. Eighteen configuration bytes are required in this example. Two bytes correspond to the output multiplexers. CCS-2 can implement the same circuit in a 3x3 grid. An implementation of the S-R latch is shown in Fig. 9.

The resource requirements of the overall system shown in Fig. 6 are quite low. As shown in Table II, the supportive control-and-test logic requires 220 logic elements (LE) and 150 registers, while the CCS structures require an increasing amount of LE out of a Cyclone II 2C35 FPGA device.

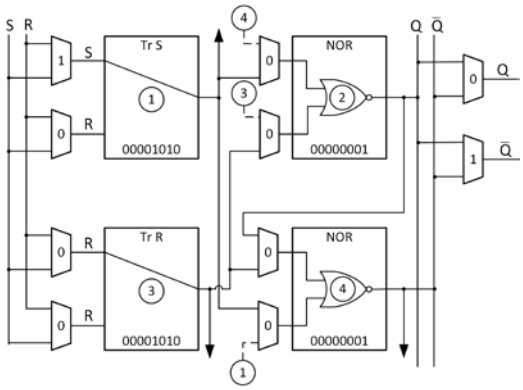

Figure 9. S-R latch implemented using the CCS-2 structure.

TABLE II. RESOURCE USAGE (CYII 2C35F672)

Hardware block	Logic Elements	Total registers
Control and test logic	220	150
CCS-1 (2x2)	19	18
CCS-1 (4x4)	126	79
CCS-1 (8x8)	438	293
CCS-1 (16x16)	1521	1099
CCS-2 (2x2)	79	40
CCS-2 (4x4)	208	127
CCS-2 (8x8)	591	429
CCS-2 (16x16)	2003	1603

CCS-1 and CCS-2 refer to the structures of Figures 1 and 4, respectively. The number of required LEs follows an almost linear dependence on the number of cells in the structure. The FPGA device used in our experiments provides a total of 33216 LE; therefore, very large structures can be implemented. The system was clocked at 100 MHz.

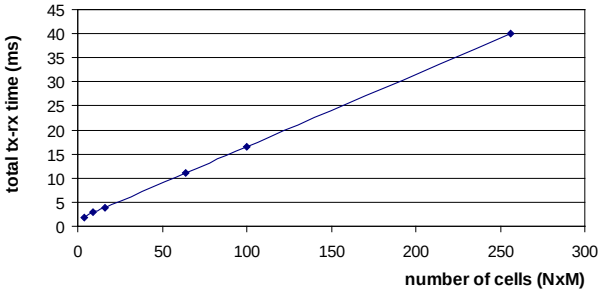

Figure 10. Total time for configuration and fitness response, as a function of grid size.

Another test concerns the response time for the full CCS configuration and response loop. Fig. 10 shows the total response time measured from the beginning of the transmission of the configuration string until the reception of the Hamming distance, for various sizes of the cell array. The implemented baud rate is 115Kbps. Since the total response time is within several milliseconds, the system can implement and test a large number of phenotypes within a reasonable time interval.

V. CONCLUSIONS

A workbench for intrinsic evolution of digital circuits is proposed. Genotypes are encoded following the principles of Cartesian Genetic Programming, while phenotypes are implemented in a reconfigurable device, making use of expandable 2D arrays of cells. As opposed to previous implementations, the proposed hyper-structures are based on a cylindrical interconnection grid, which reduces complexity and increases interconnection flexibility. Also, the proposed grids allow for feed-forward as well as for feed-back connections between the matrix cells.

A custom embedded controller configures the hyper-structures at run time while additional supportive task logic produces the required test patterns for fitness evaluation. The system is verified by implementing a series of test circuits and is evaluated in terms of the required resources and response time, for various matrix dimensions.

ACKNOWLEDGMENT

This work has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: ARCHIMEDES III, Investing in knowledge society through the European Social Fund.

REFERENCES

[1] Evolvable Hardware, T. Higuchi, Y. Liu, and X. Yao, Eds. Springer Science & Business Media, vol. 11, 2006.

[2] P.C. Haddow and A. M. Tyrrell, “Challenges of evolvable hardware: past, present and the path to a promising future”, Genetic Programming and Evolvable Machines, vol. 12, no. 3, 2011, pp. 183-215.

[3] A. Thompson, P. Layzell, and R.S. Zebulum, “Exploration in design space: unconventional electronics design through artificial evolution”, IEEE Transactions on Evolutionary Computation, vol. 3, no. 3, 1999, pp. 167-196.

[4] J. Miller, P. Thomson, and T. Fogarty, “Designing electronic curcuits using evolutionary algorithms. Arithmetic circuits: a case study”, Genetic Algorithm and Evolution Strategies in Engineering and Computer Science, D. Quagliarella, J. Periaux, C. Poloni, and G. Winter, Eds. Chechester, UK: Wiley, 1997, pp. 105-131.

[5] S. Kazarlis, J. Kalomiros, A. Balouktsis, and V. Kalaitzis, “Evolving optimal digital circuits using Cartesian genetic programming with solution repair methods”, in Proc. of the 2015 International Conference on Systems, Control, Signal Processing and Informatics (SCSI 2015), Barcelona, Spain, April 7-9, 2015, pp. 39-44.

[6] Z. Vasicek and L. Sekanina, “An evolvable hardware system in Xilinx Virtex II Pro FPGA”, International Journal of Innovative Computing and Applications, vol. 1, no. 1, 2007, pp. 63-73.

[7] A. Thompson, Hardware evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution, Springer Science & Business Media, 2012.

[8] L. Sekanina, “Evolvable computing by means of evolvable components”, Natural Computing, vol. 3, 2004, pp. 323-355.

[9] J. Miller, D. Job, and V. Vassilev, “Principles in evolutionary design of digital circuits - part I”, Genetic Programming and Evolvable Machines, vol. 1, 2000, pp. 7-35.

[10] J. R. Koza, M. A. Keane, and M. J. Streeter, “What’s AI done for me lately? Genetic programming’s human-competitive results”, IEEE Intelligent Systems, vol. 18, no.3, 2003, pp. 25-31.

[11] J. F. Miller and P. Thompson, “Cartesian genetic programming”, in LNCS, Euro GP 2000, vol. 1802, R. Poli, W. Bazhaf, W.B. Langdon, J. Miller, P. Nordin, and T.C. Fogarty, Eds. Heidelberg: Springer, 2000, pp. 121-132.

[12] L. Sekanina, and R. Ruzicka, “Design of the special fast reconfigurable chip using common FPGA”, in Proc. of the IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop, Bratislava, Smolenice, 2000, pp. 161-168.

[13] L. Sekanina, Evolvable components: from theory to hardware implementations, Springer Science & Business Media, 2012.

[14] L. Sekanina and S. Friedl, “An evolvable combinational unit for FPGAs”, Computing & Informatics, vol. 23, no. 5, 2004, pp. 461-486.

 ISBN: 978-1-61208-430-5