Extrinsic Evolvable Hardware on the RISA Architecture

A. J. Greensted and A. M. Tyrrell

Intelligent Systems Research Group,
Department of Electronics,
University Of York,
Heslington, York, UK,
YO10 5DD
{ajg112, amt}@ohm.york.ac.uk
http://www.bioinspired.com

Abstract. The RISA Architecture is a novel reconfigurable hardware platform containing both hardware and software reconfigurable elements. This paper describes the architecture and the features that make it suitable for implementing biologically inspired systems such as the evolution of digital circuits. Some of the architecture's capabilities are demonstrated with the results of evolving a simple combinatorial circuit using one of the fabricated RISA devices.

1 Introduction

Evolvable hardware provides a unique challenge to hardware engineers. In a field that uses well defined components, design tools and procedures, evolving electronic circuitry requires the ability to instantiate and connect circuit elements in a random fashion. This is of course quite contrary to the intended role of most standard electronic devices. Those wishing to investigate evolving electronic circuits work outside these constraints in order to achieve their goals.

Field Programmable Gate Arrays (FPGAs) [1] provide a useful platform for evolving circuits. Their reconfigurability provides a method repeatedly evaluate the fitness of the candidate solution circuits encountered during an evolutionary process. However, commercial FPGAs still impose difficulties when implementing evolvable hardware systems.

At the heart of these difficulties is the proprietary configuration bitstream used to configure commercial FPGAs. Without knowledge of the bitstream's construction, it is difficult to use an evolutionary process to create and test low-level bitstream based candidate solutions without risking device damage. Without resorting to well documented, simplistic, or out-of-date technology like the Xilinx® XC6200 [2], or proprietary bitstream manipulation tools, such as the also out-of-date JBits API[3], users must utilise a reconfigurable super-platform on top of an existent FPGA [4, 5]. Although this strategy works well, the inefficiency of creating one reconfigurable architecture using another reduces the size of circuit that may be created.

An alternative strategy is to design and fabricate a new reconfigurable architecture specifically with biologically-inspired systems, like evolvable hardware, in mind. Projects such as POEtic[6, 7] have taken this route. The main advantage is a greater efficiency in circuit use and modes of operation appropriate to the required task. A third alternative is to completely remove the constraints of electronics entirely and evolve systems using alternative substances [8].

The work described in this paper takes the first of these alternative approaches, a novel digital electronic architecture, designed for implementing bio-inspired systems, such as the evolution of digital circuits. The Reconfigurable Integrated System Array (RISA) [9] addresses a number of the shortfalls of using commercial reconfigurable devices for evolution: the ability to safely apply randomly generated configuration bitstreams, fine grained partial reconfiguration and a fully documented bitstream structure.

The paper is organised as follows: Section 2 provides an introduction to the RISA architecture. Sections 2.1-2.2 and 2.3 describe the two main constituent parts of the RISA architecture, a custom FPGA Fabric and microcontroller core respectively. Fabrication details of the initial RISA Device are given in Section 2.4. An initial evolvable hardware experiment using the RISA platform is described in Section 3. Ideas for future development are outlined in Section 4. Conclusions are made in Section 5.

2 RISA

The Reconfigurable Integrated System Array (RISA) is a digital electronic architecture designed for exploring biologically inspired systems. RISA extends the standard FPGA paradigm by integrating distributed reconfigurable software elements with reconfigurable hardware resources. The architecture's name reflects that each element of the array comprises a whole integrated system, the combination of a microcontroller with an area of gate array fabric. Figure 1 illustrates the RISA architecture.

The structure of the RISA cell allows a number of different system configurations to be implemented. Each cell's FPGA fabric may be combined into a single area and used for traditional FPGA applications. Similarly, the separate microcontrollers can be used in combination for multi-processor array systems such as systolic arrays [10]. However, the intended operation is to use the cell parts in conjunction, allowing the microcontroller to control its adjoining FPGA configuration. This cell structure is inspired by that of biological cells. As illustrated in Figure 2, the microcontroller provides functionality similar to a cell nucleus. By storing and manipulating different FPGA configurations, which are analogous to DNA, the cell's overall functionality, performed by the FPGA fabric, can be controlled and altered.

2.1 The RISA FPGA Fabric

The RISA FPGA fabric uses an island-style architecture [11]. The fabric design does not attempt to compete with commercial FPGA designs in terms of

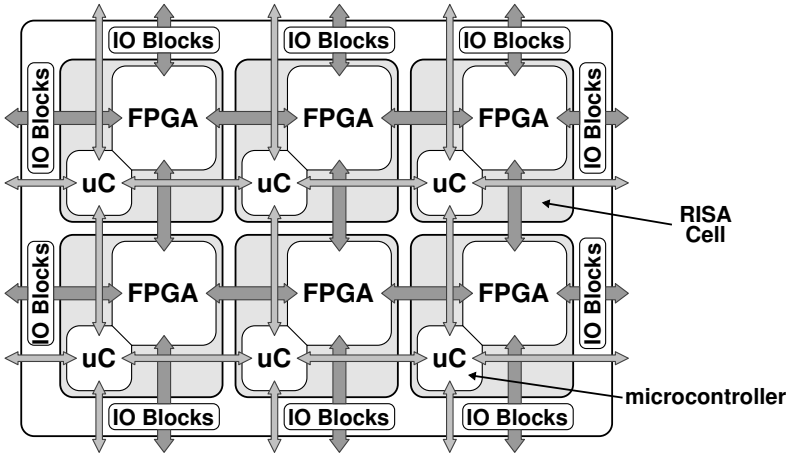


Fig. 1. The RISA Architecture comprises an array of RISA Cells. Each cell contains a microcontroller and a section of FPGA fabric. Input/Output (IO) Blocks provide interfaces between FPGA sections at device boundaries. Inter-cell communication is provided by dedicated links between microcontrollers and FPGA fabrics.

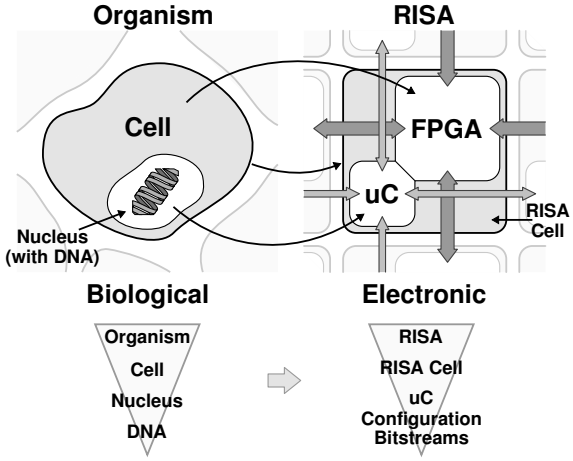


Fig. 2. The structure of the RISA cell is based upon biological cells. The microcontroller operates as a centre of cell operations, controlling the cell functionality implemented in the FPGA fabric. FPGA fabric configuration bitstreams may be stored and manipulated in the microcontroller.

density or flexibility of implementable circuits. However, the RISA fabric does provide fine grained partial reconfiguration allowing small areas of the fabric to be targeted for reconfiguration. Also the fabric is random configuration safe, so evolved bitstreams will not cause physical damage to the device. Equally important, the design enables full access to the format of the configuration bitstream. This provides the opportunity for custom bitstream generation and the reverse engineering and inspection of evolved bitstreams.

The lowest level of the RISA FPGA fabric is the *Function Unit*, shown in Figure 3. The main parts are a Function Generator, a 2:1 multiplexer and a D flip-flop. Combining these units provides the functionality listed below.

- 4 input, 1 output Look-Up Table (LUT)
- 16x1 bit RAM block
- 1 to 17 bit variable length Shift register (via external local routing)
- 4-1 multiplexer (when used with the 2-1 multiplexer)
- 1 bit full adder (when used with the carry chain)

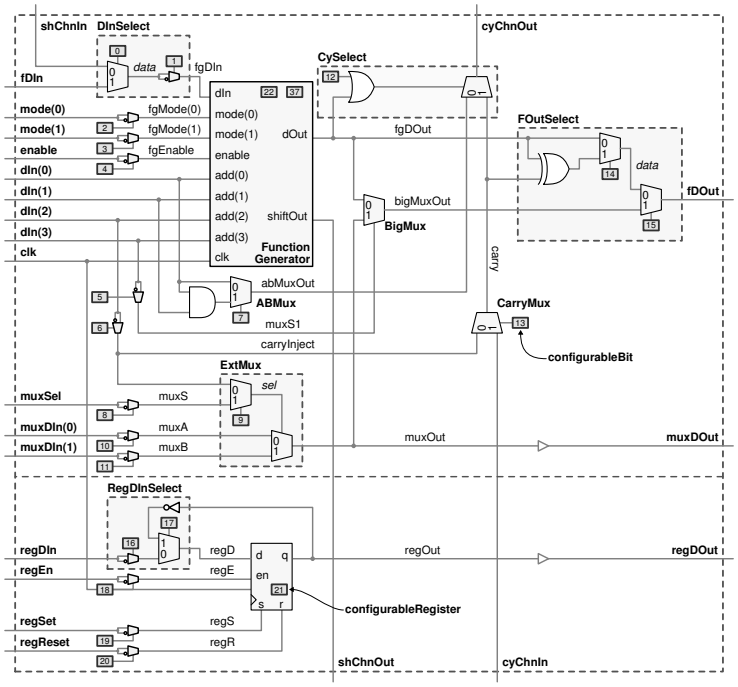


Fig. 3. The RISA Function Unit is the lowest level of the FPGA fabric structure.

The *Cluster* is the next level of FPGA structure. Each Cluster contains four Function Units, local routing, and connections to inter-cluster routing.

Clusters are arranged in a square grid to form the FPGA fabric. Figure 4a shows this arrangement. The figure also illustrates a key part of the fabric's routing scheme. Each of the Cluster's four Function Units is assigned to a different routing direction. The direction dictates how a Function Unit's circuitry may be connected.

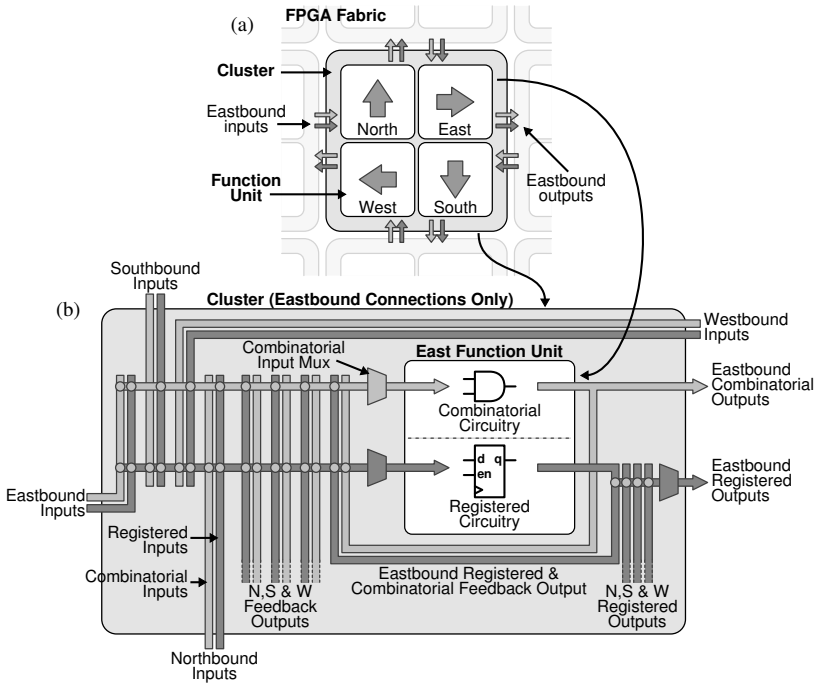


Fig. 4. The multiplexer based FPGA routing design can be randomly configured without risk of forming combinatorial feedback paths or signal contentions. Combinatorial paths may only be connected within their assigned directions. Registered paths can connect to all signal directions.

In order to achieve a random configuration safe architecture, two configuration scenarios must be avoided: the possibility of signal contention and the creation of combinatorial feedback paths. The former causes a high current to flow between alternatively driven wire endpoints, which can lead to device damage. The latter can create fast oscillations, which unnecessarily consume power, cause noise in neighbouring signals via crosstalk and can cause metastability issues on connected registers. The directed Function Unit scheme and multiplexer based routing prevents both these problems.

Figure 4b illustrates the input and output routing of a single Function Unit, in this case the Function Unit assigned to the east direction. Table 1 summarises how signals may be routed. The only restriction is that connections between

combinatorial signals must be of the same direction. This stops an unregistered loop being created.

		Inputs	
		Combinatorial	**Registered**
Outputs	**Combinatorial**	Of the same direction	Any
	Registered	Any	Any

Table 1. Function unit connectivity. The routing of combinatorial signals is limited to remove the problem of combinatorial feedback paths.

2.2 Fabric Configuration

The configurable resources of the FPGA fabric are divided into either routing or logic; routing being the FPGA connectivity, logic being the functional circuitry. The RISA configuration process supports the features listed below.

- Concurrent configuration of routing and logic resources.
- Reconfiguration without disrupting the currently operating configuration.
- Single clock cycle configuration switching.
- Configuration readback with device state inspection.

Two pairs of serial data chains, one pair for routing, the other for logic, are used for loading configuration information into the fabric. Each pair comprises a chain for selecting a target for configuration and a chain for shifting the actual configuration data. It is this approach that provides the partial reconfiguration feature of the RISA architecture. Figure 5 illustrates the configuration circuitry for a single chain pair. The same circuit design is used for both routing and logic configuration.

Figure 5a shows the *ConfigSelectUnits* that are used to select or bypass sections of the configuration chain and thus areas of configurable fabric. Figure 5b shows the two types of element that make up the configuration chain. The *ConfigurableRegister* is a D-type flip-flop that can have its initial value set and its state read back. The *ConfigurableBit* is used for setting single bit control lines, such as multiplexer selection bits and signal inverter states. Examples of these configurable elements can be seen in the Function Unit shown in Figure 3.

The lowest level unit selectable for configuration, either routing or logic, is the Cluster. Using the selection chains, any arrangement of Clusters can be targeted.

2.3 SNAP, the RISA microcontroller

A microcontroller core is incorporated into each RISA cell. Within the RISA architecture, these microcontrollers form a processor array. A custom microcontroller, the Simple Networked Application Processor (SNAP), was designed specifically for the RISA architecture.

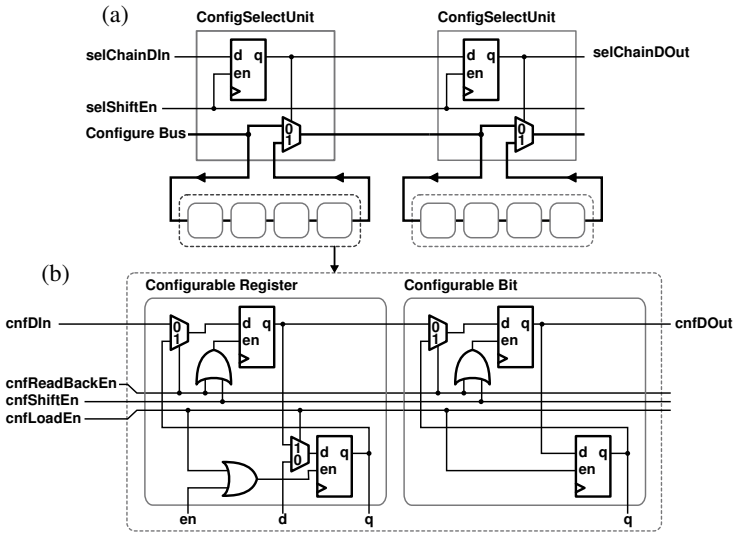


Fig. 5. Routing and Logic configuration can be configured separately, each using a pair of serial data chains. Each pair comprises a target selection chain and a configuration chain. (a) Shows the target selection circuitry. (b) Shows the two types of configurable element.

The SNAP core is, in summary, a 4 stage pipelined, RISC, Von Neumann memory design with a 16 bit data width and address space. It includes a set of interfacing links dedicated to inter-core communication. Furthermore, the core is tightly integration with the RISA FPGA fabric. The core includes a number of modules similar to those found in many commercial microcontrollers [12] such as timers, UARTs, a watch dog timer and general purpose IO ports. A hardware random number generator is also included.

One set of features that makes the SNAP design novel are the techniques used for integration with the FPGA core. First, configurable IO ports that connect directly to the FPGA fabric are built into the microcontroller's register file. Second, signals routed from the fabric can directly determine instruction execution; condition codes controlled by these signals can be encoded directly into instructions. Using this technique, blocks of code can be enabled depending on FPGA signal values without the need for more timely test code. Last, the SNAP microcontroller has direct access to the FPGA's configuration chains. Four access ports connect to each configuration chain, all being operable concurrently.

2.4 The RISA Device

So far the RISA project has produced one run of fabricated RISA devices. A cell-based 0.18μm process was used, the die layout is shown in Figure 6. Each device of this initial revision contains a single RISA cell. The FPGA fabric contains 36

Clusters, in a 6x6 grid. As can be seen, the FPGA fabric accounts for the larger area of the device. As with commercial FPGAs, routing takes up a great deal of the die area [13], especially in this case due to the routing being multiplexer based.

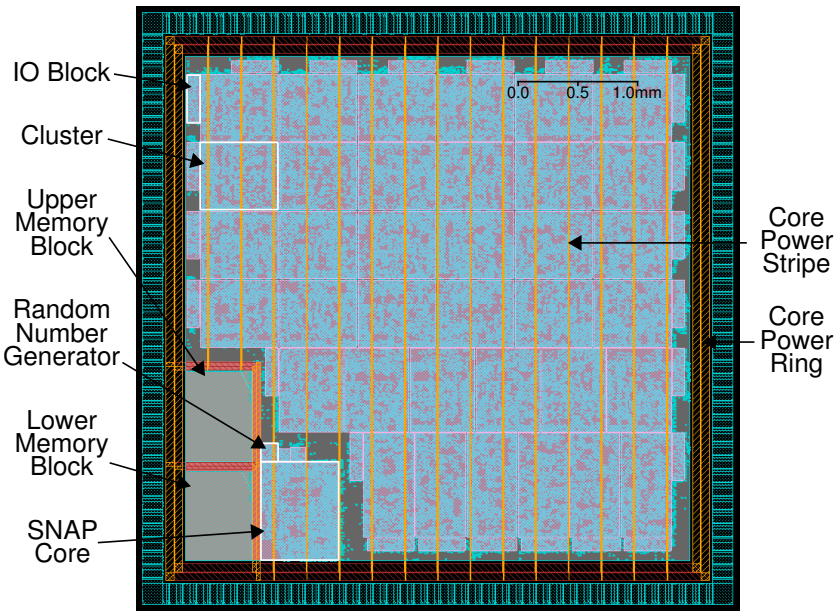
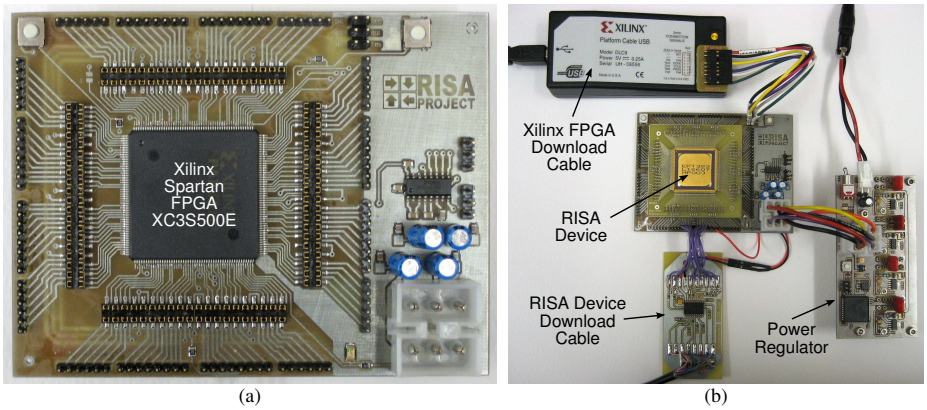


Fig. 6. The first fabricated version of the RISA architecture contains a single RISA Cell. The FPGA fabric accounts for the majority of the die area. The die is 5mmx5mm in size and fabricated using a 0.18μm process.

3 Evolvable Hardware using the RISA platform

A simple evolvable hardware experiment has been undertaken to demonstrate the RISA platform in operation. The initial experiment uses one RISA device, containing a single RISA cell. The cell's FPGA fabric was used to evolve a simple circuit. Figure 7 shows the experiment setup. The device motherboard, shown in Figure 7a, contains a Xilinx® Spartan FPGA which is used to apply test vectors to the RISA device straddled above, as shown in 7b.

For the experiment presented here, the evolutionary algorithm is performed extrinsically on the Xilinx® FPGA. RISA FPGA fabric configurations are downloaded from the Xilinx FPGA to the RISA device via the Configuration Access Port. Input vectors are applied to and circuit outputs are read from the RISA FPGA fabric via the *IOBlocks*. This setup is shown in Figure



(a) (b)

Fig. 7. (a) The RISA device motherboard without RISA device attached. (b) The experiment setup, with RISA device connected (Serial cable not shown).

8a. The Xilinx Embedded Development Kit (EDK) was used to implement the evolutionary algorithm running on a Microblaze soft core. With future development the aim is to move the evolutionary algorithm into the on-board SNAP microcontroller. Furthermore, multiple RISA cells will be used to speed up evolution time by performing multiple fitness evaluations in parallel.

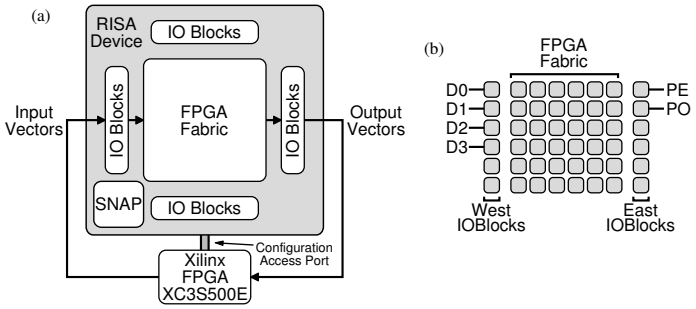
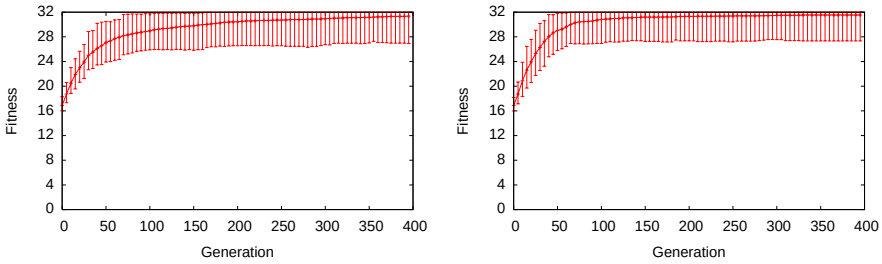


Fig. 8. (a) Circuits are evolved within the RISA FPGA fabric under the control of the motherboard's Xilinx® FPGA. (b) Input vectors are applied to the FPGA fabric via the west IO blocks. The output data, for this experiment parity data, is read from the east IO blocks.

The evolutionary algorithm is designed to evolve a 4 bit parity generator, calculating both odd and even parity. The input and output assignments are shown in Figure 8b. The aim at this early stage in system development is to prove the RISA architecture can be used for evolving circuits, rather than increase the complexity bounds of evolved hardware.

The evolutionary algorithm parameters are as follows; population size of 32; tournament selection size of 4; a generation limit of 400. Figures 9a and 9b show the mean fitness with positive and negative standard deviations for 100 runs. Figure 9a shows the results for a mutation rate of 4 out of 256, or 1.56%. 86 out of 100 runs found a successful solution. Figure 9b shows the result for a mutation rate of 2 out of 256, or 0.78%. The number of successful runs were 89 out of 100.



(a) (b)

Fig. 9. Fitness against generation averaged (mean) over 100 runs when evolving 4-bit parity generator. Also shown are positive and negative standard deviations. (a) has a mutation rate of 4 out of 256, (b) 2 out of 256.

The evolutionary algorithm makes use of only the east bound direction of cluster Function Units. Furthermore, only the combinatorial circuitry is utilised. A novel, and completely feasible, approach to accelerating evolution time would be to make use of all four Function Unit directions concurrently. This would allow four candidate configurations to be loaded and tested together.

4 Future Development for the RISA Architecture

The RISA architecture was designed for array based systems. Although, as Section 3 demonstrates, interesting experiments can be performed using a single RISA cell, the architecture's true power is in performing multicellular system operations. The next stage in the architecture's development is the implementation of different multicellular systems, such as fault tolerant Embryonic Arrays [14, 15]. Work is currently underway to develop a novel distributed rerouting algorithm for circumventing faults in array based circuits. The system can be realised by implementing system functionality in RISA cell FPGA fabric and the routing algorithm in the SNAP microcontrollers.

Other fault tolerant multicellular systems such as developmental systems [16, 17] and endocrinologic architectures [18] are also fully implementable using RISA devices. The design and manufacture of a multi-RISA device motherboard to support these systems is underway.

As the RISA architecture is a completely custom design, the suite of supporting software tools also need developing. The experiment outlined in Section 3 provided a practical way of developing and testing these tools. So far an assembler, configuration API and bitstream manipulation API have been created.

A major future development is the fabrication of a second revision RISA device. The main goal of this step will be to incorporate multiple RISA cells within a single device. As illustrated in Figure 6, the FPGA fabric uses a considerable area. By moving the ASIC design from cell-based to full-custom, the fabric density could be significantly reduced.

5 Conclusions

A custom architecture ASIC, called RISA, has been produced to simplifying the task of creating bio-inspired systems. A random configuration safe reconfiguration system is used, making the RISA platform very suitable for the evolution of digital circuits. An embedded microprocessor array provides a reconfigurable software element suitable for implementing rerouting algorithms used in cellular based fault tolerant systems.

The experiment outlined in Section 3, although simplistic, clearly demonstrates the operation of the architecture's hardware reconfigurable fabric. While the authors do not consider evolutionary algorithms as a particularly useful approach for creating digital circuits, their use in these experiments demonstrate the architecture's ability to perform extrinsically controlled reconfiguration. This functionality is believed to be critical in many applications that require adaptation and fault tolerance.

Further information on the RISA device, such a full HDL and schematics, can be found on the project website: `http://www.bioinspired.com/users/ajg112`.

6 Acknowledgements

The authors would like to thank the EPSRC (grant number GR/R74512/01) and the MoD for funding this project. Many thanks also to the members of Europractice's Microelectronics Support Centre for their expert assistance with ASIC development.

References

1. Wolf, W.: FPGA Based System Design. Prentice Hall (2004)
2. Xilinx: XC6200 Field Programmable Gate Arrays - datasheet. (1997)
3. Xilinx: JBits SDK web site. `http://www.xilinx.com/labs/projects/jbits/` (2007)
4. Sekanina, L.: Towards evolvable IP cores for FPGAs. In: Proceedings of the 3rd NASA/DoD Conference on Evolvable Hardware, EH-03, Washington, DC, USA, IEEE Computer Society (2003) 145–154

5. Sekanina, L.: Virtual reconfigurable circuits for real-world applications of evolvable hardware. In: Proceedings of 5th International Conference on Evolvable Hardware, ICES-03. Number 2606 in LNCS, Springer Verlag (2003) 186–197
6. POEtic: Project web site. http://www.poetictissue.org/ (2007)
7. Tyrrell, A.M., Sanchez, E., Floreano, D., Tempesti, G., Mange, D., Moreno, J.M., Rosenberg, J., Villa, A.E.: Poetic tissue: An integrated architecture for bio-inspired hardware. In: Proceedings of 5th International Conference on Evolvable Systems. (2003) 129–140
8. Harding, S., Miller, J.: Evolution in materio: A tone discriminator in liquid crystal. In: Proceedings of the Congress on Evolutionary Computation 2004. Volume 2. (2004) 1800–1807
9. Greensted, A., Tyrrell, A.: RISA: A hardware platform for evolutionary design. In: Proceedings of 2007 IEEE Workshop on Evolvable and Adaptive Hardware. (2007)
10. Hwang, K., Briggs, F.: Computer Architecture and Parallel Processing. McGraw Hill (1984)
11. Betz, V., Rose, J., Marquardt, A.: Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers, Norwell, MA, USA (1999)
12. ATMEL: AVR ATmega128 - datasheet. http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf (2006)
13. Ahmed, E., Rose, J.: The effect of LUT and cluster size on deep-submicron FPGA performance and density. In: FPGA '00: Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays, New York, NY, USA, ACM Press (2000) 3–12
14. Mange, D., Sipper, M., Stauffer, A., Tempesti, G.: Towards robust integrated circuits: The embryonics approach. Proceedings of the IEEE **88**(4) (2000) 516–541
15. Ortega-Sánchez, C., Tyrrell, A.: A hardware implementation of an embryonic architecture using Virtex® FPGAs. In: Proceedings of ICES 2000, 3rd International Conference on Evolvable Hardware. Number 1801 in LNCS, Springer Verlag (2000) 155–164
16. Miller, J.F.: Evolving a self-repairing, self-regulating, french flag organism. In: Proceedings of GECCO 2004, Genetic and Evolutionary Computation Conference, Springer Verlag (2004)
17. Liu, H., Miller, J., Tyrrell, A.: An intrinsic robust transient fault-tolerant developmental model for digital systems. In: GECCO 2004 workshop, Genetic and Evolutionary Computation Conference. (2004)
18. Greensted, A., Tyrrell, A.: Implementation results for a fault-tolerant mulitcellular architecture inspired by endocrine communication. In: Proceedings of EH 2005, 7th NASA/DoD Conference on Evolvable Hardware, IEEE Computer Society (2005)