Task Decomposition and Evolvability in Intrinsic Evolvable Hardware

Tüze Kuyucu, Martin A. Trefzer, Julian F. Miller and Andy M. Tyrrell

Abstract— **Many researchers have encountered the problem that the evolution of electronic circuits becomes exponentially more difficult when problems with an increasing number of outputs are tackled. Although this is an issue in both intrinsic and extrinsic evolution experiments, overcoming this problem is particularly challenging in the case of evolvable hardware, where logic and routing resources are constrained according to the given architecture. Consequently, the success of experiments also depends on how the inputs and outputs are interfaced to the evolvable hardware. Various approaches have been made to solve the multiple output problem: partitioning the task with respect to the input or output space, incremental evolution of sub-tasks or resource allocation. However, in most cases, the proposed methods can only be applied in the case of extrinsic evolution. In this paper, we have accordingly, focused on scaling problem of increasing numbers of outputs when logic circuits are *intrinsically* evolved. We raise a number of questions: how big is the performance drop when increasing the number of outputs? Can the resources of evolvable hardware be structured in a suitable way to overcome the complexity imposed by multiple outputs, without including knowledge about the problem domain? Can available resources in hardware still be efficiently used when pre-structured? In order to answer these questions, different structural implementations are investigated. We have looked at these issues in solving three problems: 4-bit parity, 2-bit adder and 2-bit multiplier.**

I. Introduction

One of the aims of evolvable hardware is to facilitate and automate the design process for increasingly complex applications. In order to achieve this, it becomes necessary to divide the task into smaller ones, which are easier and faster to evolve, and offer the possibility of reuse in different contexts. However, it is a highly non-trivial task to provide algorithms that are able to automatically decompose a given complex problem into smaller sub-tasks. This is particularly the case when no previous knowledge of the problem domain is available. Furthermore, another important factor, which is often unknown in the case of evolvable hardware, is the knowledge about the architecture of the evolvable hardware itself. Even when this knowledge is available and can thus be included in the search algorithm, it is still a complex task—and not always possible—to effectively access and make use of the given resources. Thus, there is a great demand for algorithms that autonomously break down complex tasks by automatically determining building blocks and are able to reuse them. Furthermore, the operation principles of these algorithms should be based on sufficiently general principles in order to be able to apply them not only to different problems, but also implement them with different representations and on different platforms.

There are numerous approaches in evolvable hardware where automatic decomposition of complex tasks are tackled; there are examples where automatically defined functions (ADFs) [6] and module acquisition, for instance in embedded cartesian genetic programming (ECGP) [17], are automatically achieved in genetic programming. There are some examples of ECGP where multiple chromosomes are utilised, to divide the given task with respect to its outputs. Each chromosome thereby represents an independent cartesian genetic program (CGP), which is required to solve the task for only one of the outputs. Approaches where complex tasks are automatically partitioned by evolving modules, which satisfy subsets of the demanded functionality can be found in [3], [4], [10], [12]. In the case of [4] the obtained modules are merged and optimised in terms of redundancy in a second stage (incremental evolution) of the evolutionary algorithm (EA). It is reported that, apart from increasing the level of achievable complexity, ECGP and incremental evolution also speed up the evolution process. Other approaches employ multi-objective optimisation in order to be able to evolve complex circuits, particularly ones with high input/output (IO) count [13], [20], or use genomes with variable length to account for hierarchical designs [21].

Various published results state that evolving electronic circuits will become exponentially more difficult as the input and output count increases [11], [15], [19].

We have to look closely at the results obtained from extrinsic hardware evolution experiments in order to successfully derive methods that will perform well in smaller systems. Even though there are elaborate techniques in software from which we can learn a lot, they are often rendered useless in the case of an embedded systems. Although, one of the aims of intrinsic evolution is to speedup runtime, the lack of flexibility and computing power prevent the use of these highly advanced techniques that work well in the case of extrinsic evolution. Thus, with the new developments in extrinsic evolution, the successful relative performance of intrinsic evolution has been left behind. However, the major advantage of hardware evolution is still its intrinsic realism: solutions that are found by evolution are proven to work in a physical implementation and environment. As stated in [9], there will always be problems when taking a result from simulation into the real world. Therefore it is important that further methods for improving intrinsic evolutionary

Tüze Kuyucu, Martin A. Trefzer, Julian F. Miller and Andy M. Tyrrell are with the Department of Electronics, Intelligent Systems Group, University of York. {tk519, mt540, jfm7, amt}@ohm.york.ac.uk, http://www.elec.york.ac.uk/research/intSys/bioInsp/evo.html.

This work is part of a project that is funded by EPSRC - EP/E028381/1.

978-1-4244-2959-2/09/$25.00 © 2009 IEEE

techniques are investigated.

In this paper we examine the scalability issue of logic circuits with multiple outputs. A method to structure hardware substrates is proposed in this paper in order to achieve successful evolution of circuits with multiple outputs, and to gain insight into the evolvability of different hardware topologies. The presented methods are particularly suitable for—but not restricted to—the RISA evolvable hardware platform, which is used to perform the experiments shown. In fact, the techniques developed are aimed at improving the performance of any multiple-output hardware system. The experiments undertaken tackle 4-bit parity, 2-bit adder and 2-bit multiplier, as these functions represent non-trivial circuits with increasing numbers of outputs.

II. Complexity and Evolvability of Evolutionary Hardware

Since the early days of evolutionary hardware it has been a major aim to increase the evolvability of the representations used and find solutions to more complex problems. As a consequence of this, there is a great number of publications that address these topics. Within the area of evolutionary computation, hence also in evolvable hardware, there are numerous definitions and opinions of evolvability and complexity. In the case of complexity this paper refers to multiple-output electronic circuits, rather than complexity in the sense of large distributed systems. However, the definition of evolvability requires a more detailed discussion, which is provided in section II-C.

A. Accessibility of Available Resources in Hardware

As stated in [19] evolution does not search for circuits, but for the *behaviour* of circuits, due to the fact that fitness functions describe the desired functional outcome (behaviour) of the sought circuit, rather than providing information on how to use gates and switch boxes to achieve the solution. Therefore, it is not surprising that EAs are not capable of automatically partitioning a given task or accessing provided hardware resources efficiently in the case of designed architectures; unless additional methods are applied to take these issues into account [4], [17], [18]. Therefore, the pathways available to evolutionary search on a provided hardware is usually not predictable, and thus, form a challenge to design hardware that can provide the optimal search space for evolution in the design of circuits.

Further investigations considering the accessibility of functional hardware resources and IO are conducted in [11], [14]. It is found that, in the case of arranging functional blocks on a Cartesian grid, the success rate and performance of evolution is extensively dependent on the width and length of the array: rectangular layouts where the width to length ratio is large, i.e. extremely narrow or extremely shallow, are shown to be disadvantageous for the evolution of logic circuits.

Feasibility of both partitioning and arranging resources in evolvable hardware determines the accessibility of these resources through the EA. Hence, for the design of evolvable hardware, it has to be considered that functional resources can be accessed in more than one way in order to increase their accessibility. In section II-C we discuss how the functionality-to-connectivity ratio also affects evolvability (which is a result of the accessibility of the resources provided).

B. Increased Complexity due to Multiple Outputs

It is found by various researchers that the success rate of evolution drastically decreases when the IO count of the targeted application is increased [11], [17]. As it was stated in [19], evolution does not know about structure and possible partitioning of the evolution substrate, because the fitness function is defining a circuit behaviour, rather than design principles. Therefore, when multiple output circuits are aimed to be evolved, evolution would struggle more than it would with single-output circuits due to its inability to partition the hardware substrate.

In extrinsic evolution, ways around the increased complexity due to multiple outputs have been via circuit partitioning that either directly target outputs [17] or use mechanisms that automatically partition the circuits [4], [18].

C. Evolvability of Hardware Architectures

The definition of evolvability in the case of evolvable hardware is not straight forward: the most concise definition of evolvability is probably given in [5], where it is stated that *"Evolvability is an organism's capacity to generate heritable phenotypic variation"*, i.e. if the number of viable and sufficiently fit phenotypes that can be developed from the same genome is high, it will feature a high level of evolvability. In the case of hardware evolution/this paper, the term 'evolvability' is also used in order to refer to the capability of a given hardware architecture to produce an output that changes smoothly when configuration bits are flipped. This behaviour should be more suitable for optimising and evolutionary algorithms. As this is usually not the case for digital hardware, architectures that allow for more gradual improvements during evolution are considered to feature a higher 'evolvability'.

Previous work in literature that address evolvability of hardware suggest that neutrality in the genotype leads to a high level of evolvability. This is due to its property of providing stability against deleterious mutations and its ability to allow a sampling of the search space that helps avoid local optima (neutral search) [2], [16]. However, neutrality may also cause the search space to grow much faster on larger problems, which may considerably slow down the intrinsic evolution process. Other approaches state that evolvability can be increased, in the sense of improving the performance of the evolution process, by using adaptive representations [8] or evolving independent subsystems [18].

In the case of intrinsic hardware evolution on embedded systems, technical limitations of the phenotype most often do not allow for providing a great amount of neutrality. Indeed, the aim is rather to efficiently and effectively make use of the

available resources. Therefore, the idea of neutrality can not be used as effectively in hardware to increase evolvability.

This leads to the following question: what should a hardware architecture with a high degree of evolvability look like and what should be the optimal functionality-to-connectivity ratio? One could argue that this might depend entirely on the size of the search space and the tackled problem, however, in [7] it is stated that an increasingly large search space (i.e. when providing more routing) is not necessarily an issue as long as the solution space grows to a similar extent. This suggests that the navigability of the fitness landscape is ultimately more important than its overall size. It is also found that it will be extremely difficult to evolve correct designs, if there are insufficient routing options to allow the functional blocks to find many routing alternatives to other functional blocks.

Due to the fact that in intrinsic evolution there is always a trade-off to be made between providing a great amount of resources and manufacturability [9], the conclusion would be to generally keep the amount of functional blocks lower than the routing resources in order to achieve high evolvability. However, the functionality-to-connectivity ratio might often not be optimal in available platforms, including the one used in this paper.

III. Setup for Intrinsic Hardware Evolution

The results in this paper are obtained with an embedded digital hardware system setup for intrinsic evolution. It comprises the reconfigurable integrated system array (RISA) chip and a Xilinx Spartan3E FPGA. The RISA chip is a reconfigurable digital device, which was designed at the Department of Electronics, University of York, and it is the evolution platform. The Spartan3E FPGA provides the framework to run the evolutionary algorithm (EA) and to interface RISA.

A. RISA Hardware Platform

One RISA chip provides both a programmable microcontroller and a configurable logic substrate, which are inspired by the main constituents of biological cells, namely the nucleus and the cell body. The custom designed microcontroller on RISA is called a simple networked application processor (SNAP). Inspired from the nucleus, SNAP can store and process configuration data and is able to reconfigure logic at runtime, i.e. without interfering with the circuit configuration that is currently in operation. SNAP is a reduced instruction set computer (RISC) and its instruction set is tailored to meet the needs of evolutionary computation (EC). Furthermore, it provides communication interfaces to other RISA modules, as well as to the outside world.

The configurable logic is designed in a similar fashion to field programmable gate arrays (FPGAs). In this paper, FPGA will refer to RISA's FPGA fabric unless indicated otherwise. As can be seen from figure 1, the FPGA consists of an array of 6×6 clusters, surrounded by input/output (IO) cells. The IO cells provide a total of 12 IOs at each side of the RISA module (each cell providing 2 IOs), which can be independently configured as either an input or an output of the FPGA.

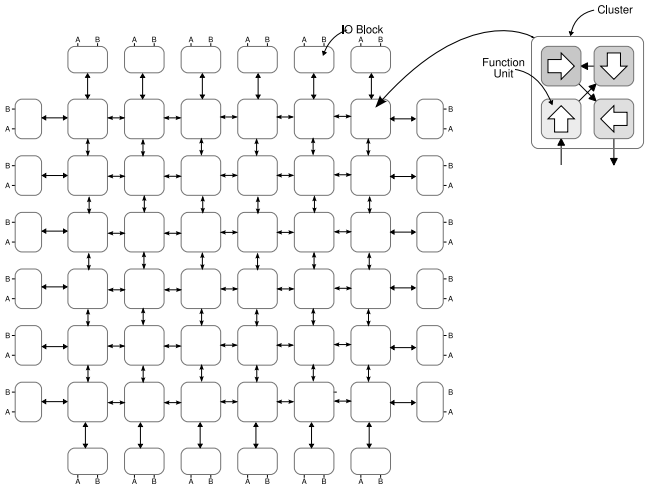

Fig. 1. The FPGA substrate of RISA consists of an array of 36 functional clusters surrounded by input/output (IO) blocks. Each cluster and IO block can be configured individually, providing partial reconfiguration. The clusters contain four function units, each offering a rich variety of routing and logic configuration options. Although all four function units can be connected with one another, they are given a primary orientation (north, south, east, west) in which a greater number of connections are possible.

Each cluster provides four functional units that can either be configured as 16 bit look-up table (LUT), shift register or random access memory (RAM). This operation mode can be changed by the evolutionary algorithm at runtime, without having to reset the registers that are used for realizing the LUT, shift register or RAM. These functional units, the available routing resources and the possibility of creating feedback loops offer a rich variety of configuration options to the EA.

RISA is designed in a way that it cannot be destroyed by random bit strings. This feature is not generally present in current commercial FPGAs: the synthesis tools of the manufacturers either constrain the access to the bit-string, in order to protect the device, or it is actually possible to destroy it. Furthermore, it is possible to partially reconfigure RISA on cluster level. This considerably accelerates hardware evolution, since only those parts of the bit-string, which have actually been changed by the EA, need to be reloaded into the device, instead of reconfiguring the entire device. A more detailed description of RISA can be found in [1].

B. Evolutionary Algorithm & Genotype

Evolutionary strategy (ES) is used as the evolutionary algorithm for the experiments that are undertaken. The mutation rate is adaptive and encoded in the genome; prior to the mutation operation, the current mutation rate is increased, decreased, or kept by a normally distributed random number between $-1 \ldots 1$, with $\sigma = 0.02$. The lower limit for the mutation rate is zero and the upper limit is 5%. Subsequently, mutation is carried out probabilistically with the new mutation rate. A variable, co-evolved mutation rate is chosen, as

TABLE I
THE EVOLUTIONARY STRATEGY PARAMETERS FOR THE EXPERIMENTS PRESENTED IN THIS PAPER.

Parameter	Value
parent size	2
population size	7
generation limit	10,000
adaptive mutation rate	$0 \ldots 5\%$, σ=0.02
number of runs	20
genome size	36×128=4608 bits

it provides a means to maintain exploration and exploitation capabilities throughout the course of evolution. This should be advantageous in the case of intrinsic evolution on the bit-string, where a rugged fitness landscape is present and neutral search is limited. Evolution is stopped when either the solution is found, or the maximum generation of 10,000 is reached. A total of 20 independent evolution runs are carried out for all experiments. Settings for the EA are summarised in table I.

The genotype used is constrained in a way that the logic of the clusters is separated into the four cardinal directions of RISA, as shown in figure 1. It consists of 128 configuration bits per cluster, resulting in a total of $36 \times 128 = 4608$ configuration bits to configure the RISA FPGA. Hence, according to the four-fold architecture of RISA, the logic and routing of each direction—north, south, west and east—require $4608 \div 4 = 1152$ configuration bits respectively.

IV. STRUCTURING EVOLVABLE HARDWARE PLATFORMS TO FACILITATE EVOLUTION

As discussed in section II, it is a challenging task (which might also be limited due to technical and financial reasons) to design a hardware platform which is particularly suited for evolution experiments. Desired features of a hardware platform designed for evolution experiments are: first, a large amount of functional resources, and an even larger amount of configurable routing to make the functional resources accessible to the EA (section II-A). However, at the same time the search space should not become too large, as the EA is carried out on an embedded system with limited memory. Second, the connectivity and arrangement of the functional blocks is important. Generally, the logic resources in hardware are not fully connected, but are constrained with respect to a given configurable routing scheme. Therefore, the success of experiments can depend on how the inputs and outputs are interfaced to the evolvable hardware (section II-C). Third, when tackling circuits with multiple outputs, the task of automatically partitioning the hardware resources and the problem structure in order to achieve multiple outputs imposes an additional level of complexity on the evolutionary search.

Since the aim of this paper is to improve intrinsic evolution of electronic circuits on the RISA platform, two methods that aim to increase the platform's evolvability are introduced: in the first approach available resources are re-shaped in order to make them more accessible to the EA, whereas in the second case, available resources are partitioned in a fashion that particularly suits—but is not restricted to—the RISA evolvable hardware platform. The performance of both approaches is tested on the evolution of multiple-output logic circuits.

A. Increasing the Accessibility of Resources

To investigate whether evolution would benefit from a larger number of logic blocks when constructing circuits on hardware, or whether it would suffer from the bigger search space, RISA's directional function units are constrained and connected in a serial fashion in order to form a long chain of logic blocks. The intention is thereby to find out whether a hardware substrate would be more evolvable if it would feature a large number of logic components with constrained connectivity, or not.

In order to obtain a longer chain of components, RISA is forced to work in a single directional fashion, i.e. the connectivity between the different function units is constrained to be always in their primary direction. When this is the case, RISA provides 6×6 function units when looking from each of the cardinal directions (N,S,W,E) respectively. By feeding the outputs of the RISA chip back into the chip, it is possible to link these resources together in order to form a chain of 24×6 function units. An illustration of how this is implemented is given in figure 2.

Experiments are done comparing the effect of *serialisation*, chaining the function units of RISA together to form a larger array of function units, with the non-serialised case—which is referred to as the "classic" case in this paper—where only a single directional set of function units are used (i.e. the size of the available function units are 6×6). The experiments are presented in section V.

B. Breaking down Task Complexity by Hardware Decomposition

In order to investigate the effects of output decomposition in hardware, but break down the given task in a way that does not require specific knowledge about it, a way to decompose the RISA hardware platform is proposed that suits the given architecture. The proposed method is not strictly restricted to RISA as the principles are generally valid. Due to the four-fold architecture of the RISA chip, the four directions are used as independent logic blocks. In principle the routing configuration options allow the interconnection of four function units within each cluster. However, in this case, the genome is designed in a way that the four function units are kept separate and can only connect to the function units of the same kind of the neighbouring clusters.

The aim is to use the four separate blocks of circuitry obtained in RISA to evolve a multiple output circuit by partitioning it with respect to its outputs. Inputs would be applied to all four circuits obtained, and each circuit would then be expected to evolve one of the outputs, rather than all of them, see figure 3.

Using this decomposition method, the performance and the success rate of evolution in constructing multiple output

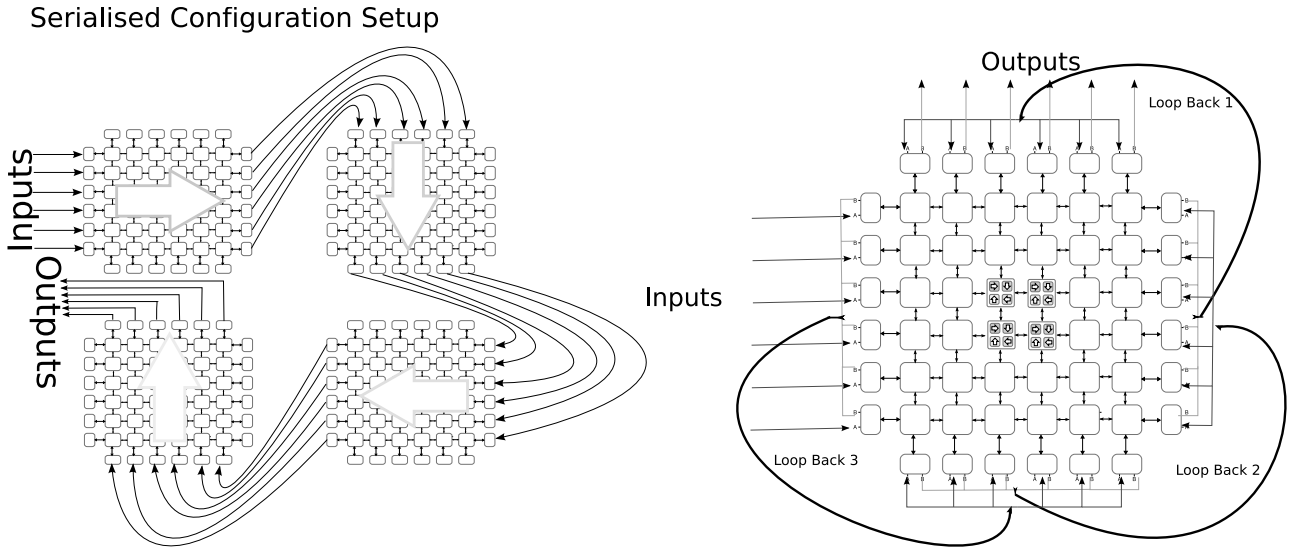

Fig. 2. The resources of RISA in all four directions are connected in serial in order to provide evolution with a long chain of logic blocks. To achieve this the inputs of the circuit is applied to the ports A of the western I/O blocks to go through the eastward function units of the available clusters, then the outputs from the ports B of the eastern I/O blocks are connected to the ports A of the northern I/O blocks, and etc. Looping the inputs and outputs around the chip, all of the available I/Os and FUs are utilized. The final circuit output is then retrieved from the ports B of the Northern I/O blocks. Another option of serialising the four directions would be, for instance, to feed the outputs (B) from the North and the South into the inputs (A,B) on the West. However, due to the limited number of I/Os on each side, the East direction could not be used in the latter case. Despite the second setup example is possibly beneficial for certain problems, the one that is depicted above is chosen, because it allows to use the entire available logic of RISA and all I/Os.

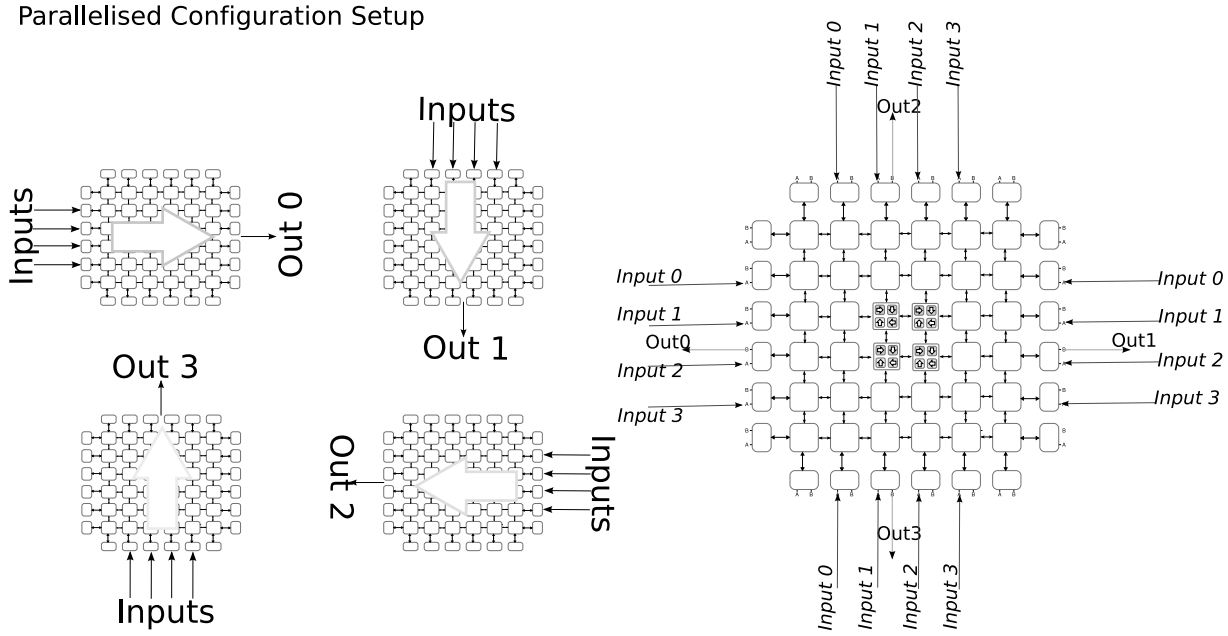

Fig. 3. To partition the problem outputs, the four directional function units of each cluster are constrained again to direct the flow of the circuit, and then all inputs are applied from all four directions using the ports A of the I/O blocks. The outputs are then divided to four and each output chunk is retreived from a different direction of the chip (using ports B of the I/O blocks).

circuits is shown to improve by the experiments presented in section V.

V. Results for Parity, Fulladder and Multiplier

Various experiments are carried out in order to determine the effects of; evolving circuits with multiple outputs in hardware, and the amount of resources evolution is forced to use. The problems examined in the experiments are, 4-bit even parity, 2-bit full adder, and 2-bit multiplier. Results are shown in table II.

In order to investigate the effects of multiple outputs on the performance of evolution, experiments are carried out where only a single output of a multiple output circuit is evolved. Subsequently, the number of outputs is increased

TABLE II

RESULTS OBTAINED FROM 20 INDEPENDENT RUNS FOR 4-BIT PARITY, 2-BIT FULL ADDER AND 2-BIT MULTIPLIER ARE SHOWN. ALL EXPERIMENTS ARE CARRIED OUT WITH THREE DIFFERENT SETUPS, NAMELY *classic*, *serialised* AND *parallelised*. DIFFERENT SERIES OF EXPERIMENTS ARE PERFORMED WITH AN INCREASING NUMBER OF OUTPUTS, IN ORDER TO ILLUSTRATE THE INCREASING LEVEL OF DIFFICULTY OF THE RESPECTIVE TASK. FOR INSTANCE, THE RUNS WHERE THE TASK IS TO EVOLVE A MULTIPLIER WITH ONLY ONE OUTPUT, THREE OF ITS OUTPUTS WERE NOT CONSIDERED IN THE FITNESS FUNCTION.

logic circuit	hardware setup	outputs	successes	avg. best fitness	avg. no. generations
4-bit parity	classic	1	20	0	126 ± 123
	serial (4x)	1	20	0	154 ± 142
2-bit full adder	classic	1	19	2.0	1214 ± 1709
		2	8	9.5 ± 5.0	5156 ± 2086
		3	3	3.6 ± 2.3	3536 ± 1534
	serial (4x)	1	2	23.0 ± 13.4	5276 ± 3116
		2	0	42.0 ± 25.1	N/A
		3	0	47.0 ± 24.6	N/A
	parallel	2	20	0	4903 ± 2816
		3	20	0	6491.5 ± 3115.3
2-bit multiplier	classic	1	20	0	140 ± 86
		2	19	9.0	2578 ± 1595
		3	16	6.3 ± 2.8	3892 ± 3133
		4	10	7.5 ± 6.3	5420 ± 2999
	serial (4x)	1	1	37.0 ± 20.2	5017
		2	2	54.1 ± 19.0	4546 ± 470
		3	1	71.9 ± 46.2	4076
		4	0	152.0	N/A
	parallel	2	20	0	1991 ± 1895
		3	20	0	2672 ± 1756
		4	19	29	3266 ± 2126

in a series of following experiments until all outputs of the desired circuit are included in the experiment. Not surprisingly, by doing so the evolutionary performance degrades for the classic case as the number of outputs evolved are increased. On the other hand, the experiments for the serial case, where a larger number of function units are chained together to form a 4 times larger chain of logic for the evolutionary experiments, perform poorly. This suggests that evolutionary performance on hardware does not benefit from merely linking an excessive number of circuit components together, without implementing a routing scheme that guarantees sufficient interconnectivity at the same time. Due to the restricted routing that is often present in hardware, the latter cannot always be achieved [15], [19]. Partitioning the RISA chip in order to provide 4 different circuits to evolve, keeps the evolutionary performance stable when the number of outputs is increased.

This suggests that, partitioning the outputs of the circuit evolved on hardware can make the evolution of circuits with multiple outputs more effective. On the other hand, providing excessively large amount of components for evolution on hardware can have a negative effect on the performance of evolution. Furthermore, with a significantly increasing number of components, the accessibility of the functional blocks must be maximised with a suitable routing scheme. However, the routing could only be changed in a restricted fashion for the experiments shown.

VI. CONCLUSIONS

Two methods of reshaping a hardware substrate in order to improve the evolution of circuits with multiple outputs have been introduced: in the first case, the structure of the available resources has been changed in a serial fashion with the intention to facilitate the accessibility of the functional blocks; and to provide a greater number of options to the EA (*serialised* configuration). In the second case, the architecture at hand is divided into four parallel subcircuits, in order to break down the increasing level of complexity that comes with multiple output problems (*parallelised* configuration). These approaches have been compared with the straightforward way of using the substrate (*classic* case) on the evolution of 4-bit parity, 2-bit fulladder and 2-bit multiplier.

The results from the series of experiments with the *classic* configuration, where the difficulty of the task was successively increased by demanding more and more outputs to be correctly evolved, show that it is the case for all problems that the success rate of evolution significantly decreases. It is observed that not only does the success rate drop, but the required average number of generations to find solutions increases at the same time. Please note that the

more important number is the success rate; the standard deviations define rather boundaries for best/worst cases, due to the skewed nature of the fitness distributions.

It was hoped that evolvability of the substrate would be increased by making resources more accessible and providing more configuration options to the EA via *serialisation*, however, the results show that this has not been achieved. In fact it seems that the task has become even harder, as the success rate is almost always equal to zero. This suggests that it is not always beneficial when designing an evolvable hardware platform to just line up a great amount of logic resources. It is shown in [7], that a large amount of logic components may help extrinsic evolution in designing electronic circuits. However, this is not generally the case for hardware, as the presented results suggest. Providing hardware to work with a larger number of circuit components seems to make it harder for evolution to find solutions in this case. This might be because of the relatively limited connectivity available in hardware.

It is observed that, unlike *serialisation*, the *parallelised* configuration provided a major improvement and in almost all cases yielded a success rate of 100% for all problems at the highest level of difficulty, i.e. when all outputs are evolved. This is particularly satisfactory, since the evolution of circuits with multiple outputs is also a major problem in extrinsic evolution. By output partitioning it was shown that the evolution of the desired circuit becomes much easier, even without complex approaches that are only feasible in simulation. To conclude, it has been shown that partitioning hardware resources is a useful way of achieving both: solutions to complex multiple-output circuits and speeding-up evolution. The results also suggest that the techniques presented should improve the performance of any multiple-output hardware system.

Although it would be interesting, we did not consider the increase in complexity in the case of increasing number of inputs, as presented in [11]. In our case of intrinsic evolution we observed in preliminary experiments that an increase in number of outputs caused a more severe decrease in performance of evolution than the input count. However, high input count is a problem in the case of intrinsic evolution as well, since the number of IOs is limited. Therefore, a possible direction for future work could be to employ a MUX array to compress a large number of inputs—in a similar fashion as it is done in [10]—before applying them to the evolvable sub-system. A further improvement in the case of the *parallelised* architecture could be to use multi-chromosome genomes, where each chromosome represents only parts of the hardware configuration that are responsible for the same output, similar to [17].

REFERENCES

[1] A. Greensted and A. Tyrrell, "RISA: A hardware platform for evolutionary design," in *Proceedings of 2007 IEEE Workshop on Evolvable and Adaptive Hardware*, April 2007.

[2] I. Harvey and A. Thompson, "Through the labyrinth evolution finds a way: A silicon ridge," in *Proc. 1st Int.Conf.on Evolvable Systems (ICES'96)*, ser. LNCS, vol. 1259. Springer-Verlag, 1997, pp. 406–422.

[3] J.-H. Hong and S.-B. Cho, "Meh: modular evolvable hardware for designing complex circuits," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2003, 8 - 12 December 2003, Canberra, Australia*. IEEE, 2003, pp. 92–99.

[4] T. Kalganova, "Bidirectional incremental evolution in extrinsic evolvable hardware," in *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society, 2000, pp. 65–74.

[5] M. Kirschner and J. Gerhart, "Evolvability." *Proc Natl Acad Sci U S A*, vol. 95, no. 15, pp. 8420–8427, July 1998.

[6] J. R. Koza, *Genetic programming II: automatic discovery of reusable programs*. Cambridge, MA, USA: MIT Press, 1994.

[7] J. F. Miller and P. Thomson, "Aspects of digital evolution: Evolvability and architecture," in *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, A. Eiben, T. Baeck, M. Schoenauer, and H.-P. Schwefel, Eds. London, UK: Springer-Verlag, Heidelberg, 1998, pp. 927–936.

[8] J. Reisinger and R. Miikkulainen, "Acquiring evolvability through adaptive representations," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM Press, 2007, pp. 1045–1052.

[9] A. Stoica, R. Zebulum, X. Guo, D. Keymeulen, I. Ferguson, and V. Duong, "Taking evolutionary circuit design from experimentation to implementation: Some useful techniques and a silicon demonstration," *IEEE Proc.-Comp. Digit. Tech.*, vol. 151, no. 4, pp. 295–300, 2004.

[10] E. Stomeo and T. Kalganova, "Improving ehw performance introducing a new decomposition strategy," in *Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems*, Singapore, December 2004.

[11] E. Stomeo, T. Kalganova, and C. Lambert, "Analysis of genotype size for an evolvable hardware system." in *IEC (Prague)*, C. Ardil, Ed. Enformatika, anakkale, Turkey, 2005, pp. 74–79. [Online]. Available: http://dblp.uni-trier.de/db/conf/wec/iec2005prague.html#StomeoKL05

[12] J. Torresen, "Evolving multiplier circuits by training set and training vector partitioning," in *ICES*, 2003, pp. 228–237.

[13] M. Trefzer, J. Langeheine, J. Schemmel, and K. Meier, "Operational Amplifiers: An Example for Multi-Objective Optimization on an Analog Evolvable Hardware Platform," in *Evolvable Systems: From Biology to Hardware, Sixth International Conference, ICES 2005*, ser. LNCS, J. M. Moreno, J. Madrenas, and J. Cosp, Eds., no. 3637. Sitges, Spain: Springer-Verlag, September 2005, pp. 86–97.

[14] M. A. Trefzer, T. Kuyucu, A. J. Greensted, J. F. Miller, and A. M. Tyrrell, "The input pattern order problem: Evolution of combinatorial and sequential circuits in hardware," in *Proceedings of the International Conference on Evolvable Systems (ICES 2008)*, Prague, September 2008.

[15] V. K. Vassilev and J. F. Miller, "Scalability problems of digital circuit evolution: Evolvability and efficient designs," in *EH '00: Proceedings of the 2nd NASA/DoD workshop on Evolvable Hardware*. Washington, DC, USA: IEEE Computer Society, 2000, p. 55.

[16] V. Vassilev and J. Miller, "The advantages of landscape neutrality in digital circuit evolution," in *Proceedings of the 3rd International Conference on Evolvable Systems: From Biology to Hardware*. Springer, 2000, pp. 252–26.

[17] J. A. Walker, J. F. Miller, and R. Cavill, "A multi-chromosome approach to standard and embedded cartesian genetic programming," in *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2006, pp. 903–910.

[18] R. A. Watson and J. B. Pollack, "Modular interdependency in complex dynamical systems," *Artificial Life 11*, no. 4, pp. 445–457, 2005.

[19] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware," in *IEEE Transactions on Systems, Man and Cybernetics, Part C*, vol. 29, February 1999, pp. 87–97.

[20] R. S. Zebulum, M. A. Pacheco, and M. Vellasco, "A multi-objective optimisation methodology applied to the synthesis of low-power operational amplifiers," in *Proceedings of the XIII International Conference in Microelectronics and Packaging*, I. J. Cheuri and C. A. dos Reis Filho, Eds., vol. 1, Curitiba, Brazil, 1998, pp. 264–271. [Online]. Available: citeseer.ist.psu.edu/zebulum98multiobjective.html

[21] R. S. Zebulum, M. A. P. Pacheco, and M. Vellasco, "Variable Length Representation in Evolutionary Electronics," *Evolutionary Computation Journal*, vol. 8, no. 1, pp. 93–120, 2000.