Fitness Functions for the Unconstrained Evolution of Digital Circuits

Tüze Kuyucu, Martin Trefzer, Andrew Greensted, Julian Miller and Andy Tyrrell

Abstract— **This work is part of a project that aims to develop and operate integrated evolvable hardware systems using unconstrained evolution. Experiments are carried out on an evolvable hardware platform featuring both combinatorial and registered logic as well as sequential feedback loops. In order to be able to accurately assess the transient output of the system and at the same time speed up evolution, new fitness evaluation methods are introduced. These bitwise and hierarchical fitness evaluation methods are adapted and further developed specifically for hardware implementation. It is shown that the newly developed approaches are particularly powerful in coping with two important issues: computational ambiguities, which generally occur when evaluating binary strings, and transient effects resulting from measuring hardware output. On two combinatorial problems it is shown that the new fitness functions improve the performance of evolution and allow stable solutions to be found more reliably. The experiments are carried out with a recently developed hardware platform called reconfigurable integrated system array (RISA).**

I. Introduction

The range of tasks, to which evolutionary computation is successfully applied, is constantly becoming broader and evolutionary systems are becoming more complex and computationally intensive. It is inevitable that this rise in complexity will increase, in order to build more generic and smarter applications. However, these systems depend on great computational power, which can only be provided by large parallel computing systems. Approaches that are particularly computationally expensive are, for instance, genetic programming (GP) [14], [15], [17], [18], [20], where the growth of the variable length genotype is not limited, and those, where complex genotype-phenotype mapping or fitness evaluation is implemented. [7], [19] are examples where the latter two problems are addressed.

If an evolvable device is desired, which can be operated in the field as part of an autonomous system, the previously mentioned approaches will no longer be suitable. Some of them will even be impossible to include in a system that is bound to limited resources and has to cope with unknown environments. As a consequence of this, it becomes equally important to develop powerful and evolvable integrated systems, i.e. systems on a chip, which contain all the constituents of an evolutionary system—hardware and software—and therefore enable adaptivity to previously unknown environments, fault tolerance and autonomous behaviour. In order to achieve this, it is necessary to develop efficient and resource saving architectures and algorithms, which are suitable to be integrated in hardware and are at the same time sufficiently powerful in solving complex tasks. As yet, there are only a few examples where the entire evolvable system has been realized as system on a chip, e.g. [5], [6], [10], [16]. In most cases, the presence of a high-level controller, carrying out the evolutionary algorithm (EA) and managing the population is required.

This work is part of a project that aims to develop and operate this kind of evolvable and developmental system. In order to achieve this, an important task will be the development of efficient algorithms, which are suitable to be integrated in systems on a chip. Therefore, as a first step, generic fitness evaluation methods have been adapted and developed with consideration for implementing them in hardware. Two novel fitness functions have been implemented: the first one, which is particularly suitable to address transient effects resulting from measuring real hardware is an extension to the traditional bitwise fitness calculation (hamming-distance), and is referred to as the bitwise modified for hardware (BMH) method. The second custom fitness evaluation method that has been developed is inspired by the HIFF method, described in [22]. It is based on sampling the bit string by evaluating blocks of bits of variable size and is therefore referred to as hierarchical bit-string sampling (HBS).

The main focus of this paper is on finding efficient fitness evaluation methods, which allow for accurately assessing transient effects of real hardware systems that include feedback. As yet, to the authors' knowledge, feedback is most often not included in the evolution of digital circuits, although it should yield interesting results. In this case, the term feedback refers to circuits that feature the ability to oscillate rather than being merely finite sequential circuits: examples for this can be found in [12], [21]. Hence, feedback is enabled in the experiments presented in this paper, and, as a consequence of this, the additional challenge to the EA is to find a static solution in a transient dynamic system. If a suitable fitness evaluation method is found, which is able to accurately assess and control transient effects, it will be suitable for both the evolution of static and dynamic circuits. Due to the considerably high complexity of these kinds of systems, the XOR is chosen as a relatively basic, but representative task. The performance of all four fitness functions, namely bitwise, BMH, HIFF and HBS, is presented. Additionally, a 4 bit parity generator is evolved to

Tüze Kuyucu, Martin Trefzer, Andrew Greensted, Julian Miller and Andy Tyrrell are with the Department of Electronics, Intelligent Systems Group, University of York. {tk519, mt540, ajg112, jfm7, amt}@ohm.york.ac.uk, http://www.bioinspired.com.

The authors would like to thank James Walker for his contributions on the HIFF fitness evaluation method.

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

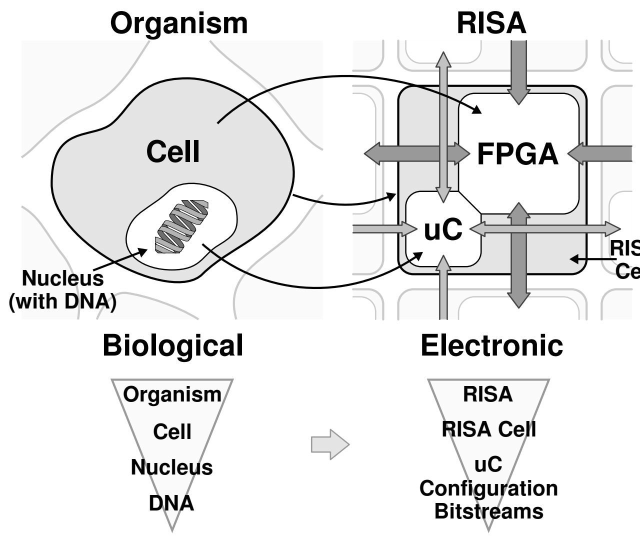

Fig. 1. The structure of the RISA cell is inspired by biological cells. The microcontroller operates as a centre for cell operations, controlling the cell functionality implemented in the FPGA fabric. FPGA fabric configuration bit streams may be stored and manipulated in the microcontroller [3], [4]

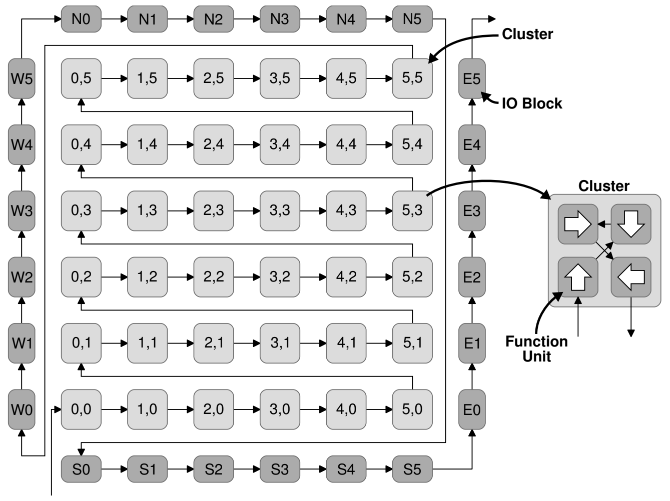

Fig. 2. The FPGA substrate of RISA consists of an array of 36 functional clusters surrounded by input/output (IO) blocks. The configuration chain for the clusters and the IO blocks are connected serially, but each cluster and IO block can be configured individually, providing partial (re-)configuration. Each cluster offers a rich variety of configuration options: 152 bits are required to configure the logic and 320 bits are required to configure the routing of one cluster, resulting in a total of 16992 bits for the whole 36 cluster configuration bit-string. Each cluster features four flexible functional units that can either be configured as 16 bit look-up table, shift register or RAM as well as according routing resources.

show that the presented approaches work equally well on another problem.

All experiments are carried out on the custom made RISA evolvable hardware platform [3], [4], which has been developed for biologically inspired experiments with digital hardware. As yet, the assistance of a Xilinx Spartan 3 FPGA is required for hosting the EA, however, the EA is planned to be moved to the customised microprocessor on RISA in the future.

II. RISA Hardware Evolution Platform

The reconfigurable integrated system array (RISA) is a reconfigurable digital device, which was designed as a platform for intrinsic hardware evolution and development at the Department of Electronics, University of York [3], [4].

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, as shown in figure 1. The custom designed microcontroller on RISA is called a simple networked application processor (SNAP). Inspired from the nucleus, SNAP stores and processes configuration data and is able to (re-)configure logic at runtime, i.e. without interfering with the currently running circuit configuration. 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 2, the FPGA consists of an array of 6 × 6 functional 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. Like a biological cell's body, the FPGA fabric carries out the tasks of the respective RISA module. Additionally, configurable logic of different RISA modules can be directly interconnected, in order to build larger circuits or, in terms of biology, larger organisms.

Each cluster provides four mutable functional units that can either be configured as 16 bit look-up table (LUT), shift register or random access memory (RAM). Thereby, the term 'mutable functional unit' refers to the fact that the 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 offers a rich variety of configuration options to the EA.

In addition, the FPGA offers features that make it particularly suitable for evolution experiments: first, it is designed in a way that it cannot be destroyed by random bit strings. As a consequence, as concluded in [11], unconstrained evolution can take place. The latter 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. Second, the configuration of clusters can be changed independently from each other, hence, the logic offers partial reconfiguration. This can considerably accelerate hardware evolution [10], 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.

Further information about RISA can be found in [3], [4].

III. Fitness Measures

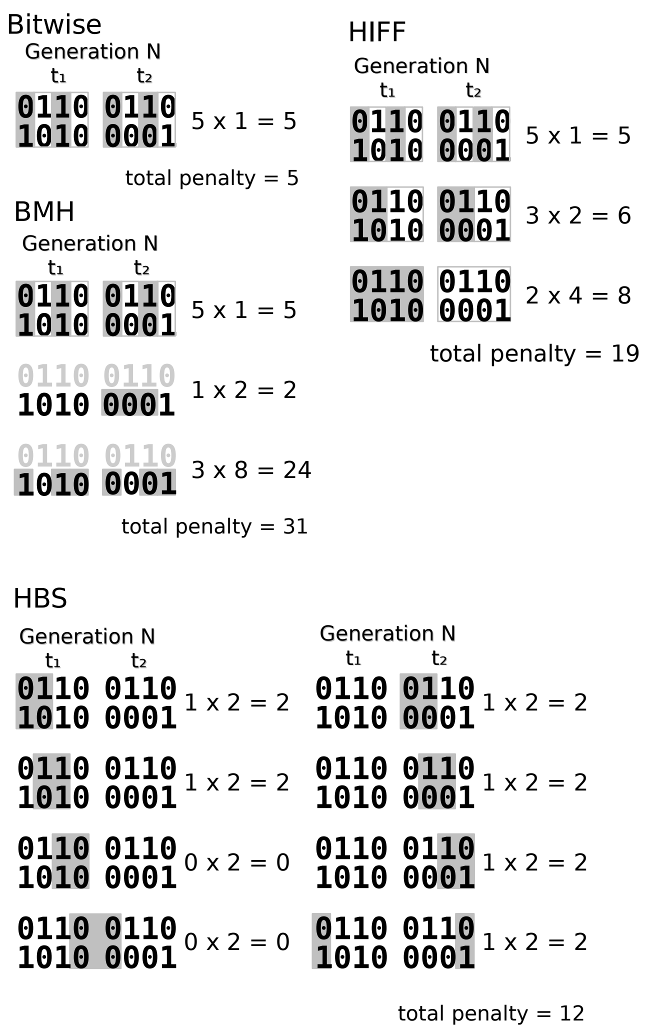

Fig. 3. Example fitness calculation for all four approaches are shown; first line of the two sets of outputs is always the desired output (XOR in this case), and the second line is the actual outputs from generation N. *Bitwise:* Only a single iteration is needed to calculate the fitness value, and it involves bit by bit comparison, for every wrong bit the fitness is increased by 1 (there are 5 in this case). *BMH:* For calculating the fitness using this method, 3 checks are done in a single iteration on the output, the first one is the same with the bitwise fitness calculation, the second check involves a bit variety check (the penalty is the difference between the number of '1's and number of '0's in single sets of outputs, 2 in this case), the third check looks for a change in two different sets of outputs for the same input (transient behaviour [3 in this case]), and penalises each change by the number of output bits tested (8 in this case). *HIFF:* For HIFF 3 iterations are needed to calculate the fitness value: the first iteration is same with bitwise, the second iteration compares bits in sets of two and penalises each non-matching set by 2 (in this case there are 3 non-matching sets), and in the third iteration the bits are compared in the sets of 4 (a complete set of outputs), and each non-matching set is penalised by 4. *HBS:* In the example above only a binary block size of 2 bits is used, so only 1 iteration is needed. The only difference from HIFF block size 2 is that the binary block is moved every bit rather than every 2 bits. For the last output bit the binary block wraps around to the first output bit in the generation.

The goal is to find efficient fitness evaluation methods, which allow for accurate assessment of transient effects of hardware systems that include feedback, and to speed up the evolutionary process at the same time. In order to achieve this, two new fitness functions are developed and are applied to the evolution of digital circuits on the RISA hardware. In addition, two existing methods have been implemented and tested on RISA. Before discussing their behaviour in section IV and comparing their performance in section V, the fitness methods are introduced. Examples of how the fitness is calculated in each case are given in figure 3.

A. Bitwise Fitness Calculation

The bitwise fitness calculation is a straight forward measure for assessing bit strings. It is simply calculated as the hamming distance between the measured output and the desired output. Thus, every false bit is penalised with one fitness point (or every correct bit is awarded one fitness point).

B. Bitwise Fitness Modified for Hardware (BMH)

BMH fitness evaluation method is specifically developed for combinatorial hardware evolution on unconstrained hardware where feedback loops are allowed. It is built around the simple bitwise comparison between measured and desired outputs, but it also undertakes a parity check and a transient faults check: thus, the complete logic input pattern has to be iterated through the solution at least twice for every evaluation.

To overcome the problem of being deceived by intermittent solutions, the BMH fitness evaluation method penalises transient behaviour by comparing the outputs of a given input combination at two different time steps for a change in the corresponding outputs: i.e. on the same circuit evolved, for test case X_N if the output of input A at time t_n is different to its output at time t_{n+1}, the fitness is penalised by an appropriate value. This is a simple yet effective way of directing evolution away from the solutions that provide intermittent results.

Other common problems that occur in digital hardware include unconnected terminals and stuck-at gates [13]. In the case of stuck-at gates, their output is always either '0' or '1', independent of their actual input and their logic function. Therefore, for certain test patterns, the output of the circuit is no longer correlated to it's input. During the course of evolution experiments it is likely for the output to be stuck at one value. In bitwise fitness calculation, a stuck-at output often gives a 50% success in the fitness value, thus trapping evolution in local optima. To overcome this problem BMH fitness evaluation method awards bit variance via parity check to push evolution away from the solution region where outputs are all '0's or all '1's and push it towards better solutions.

C. Hierarchical If-And-Only-If (HIFF)

The hierarchical if-and-only-if fitness calculation (HIFF) used for the experiments in this paper has been proposed for hierarchical if-and-only-if problems in [22]. It extends the bitwise fitness calculation (hamming distance) by introducing additional steps of fitness calculation, where blocks of bits of increasing size are compared, and the penalty is increased proportionally to the block size of the respective step. As a consequence of this, larger schema within the bit-string are recognised, and accordingly penalised (or rewarded) and are therefore preserved in case they match the desired output.

An example of how to calculate the HIFF fitness for a resulting bit string is shown in figure 3. As can be seen from table I and figure 4, the HIFF approach is able to resolve one major ambiguity of bitwise fitness calculation: it is possible to assign a finer grained fitness value range, due to the fact that bits are additionally evaluated with respect to their context and larger schema. Therefore, it becomes more unlikely to get stuck in local optima, especially those caused by trivial solutions like 0000.

D. *Hierarchical Bit-string Sampling (HBS)*

In order to benefit from the context sensitivity of the HIFF method and at the same time further increase the granularity of the fitness value range, a hierarchical bit-string sampling (HBS) method is proposed. Rather than dividing the bit string into disjunct blocks of increasing size, the bit string is evaluated by sampling it with overlapping windows (blocks) of increasing size and adding the resulting penalties. As depicted in figure 4, the HBS method provides a finer granularity. It is also suggested that the increased context sensitivity—predecessive and successive bits are now taken into account—provides further benefit to the EA.

Nevertheless, with both HIFF and HBS approaches, it is still not possible to explicitly penalise different results for the same input, even in the case of a randomised input pattern. However, randomising the input is mandatory when measuring real hardware, in order to prevent the EA from exploiting previous states of the substrate. The latter feature is only provided by the BMH approach, described in section III-B, which has been tailored especially for unconstrained hardware evolution experiments.

IV. Ambiguous Fitness Assignment

When evaluating the fitness of binary strings that result from measuring digital circuits, ambiguities in the fitness measure have to be considered. Depending on the task and the objective function, it will not be clear on which solution to promote, if the fitness values calculated from different output bit-strings are the same. This is particularly true when the fitness is given as the hamming distance between desired and measured output. As can be seen from table I, where bitwise fitness represents hamming distance, four different outputs (X_0, X_1, X_3 and X_4) result in the same fitness value, although, X_0 is considered to be a better solution than X_1, due to the fact that the presence of an unconnected output could easily lead to measuring X_1 and therefore to a lower probability for evolution to improve the candidate circuits. The consequence of this can be either getting stuck in a local optimum or, even worse, misleading the EA towards worse solutions.

Furthermore, in the case of measuring circuits with feedback on hardware, transient effects and possible oscillations

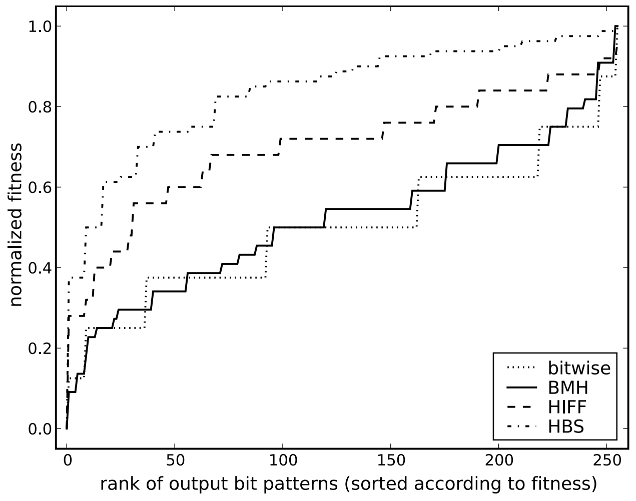

Fig. 4. Resulting fitness values for all approaches and all 256 possible logic input vectors for t_0 and t_1 (refer to table I)are calculated and plotted. In all cases, the values are sorted in ascending order of fitness providing a better overview over the fitness landscape. Ambiguities in the fitness measures become visible in regions where the fitness remains constant for a wide range of different input patterns. In these cases, the EA is not able to decide which solution to promote.

have to be considered as well. These effects cause further ambiguities in the fitness assignment, thus misleading the EA. For instance, in the case of oscillations and unknown on-set times, the same input can produce different outputs when measured at different points in time; as a consequence, good solutions cannot easily be recognised.

Therefore, suitable fitness measures are developed in this work, which inherently take the above mentioned ambiguities and transient effects into account. In order to save resources, which is important when dealing with hardware, it is also desirable that the actual fitness calculation is not too computationally expensive. In the following, the behaviour of the four different fitness measures, introduced in section III, will be compared.

A. *Implications of Transient Effects in Hardware*

Hardware evolution has various consequences that are not addressed in software evolution. In particular, intrinsic hardware evolution demonstrates various types of behaviour that are not recognised by using simple fitness evaluations. Transient effects are one of these examples for behaviour that is encountered in hardware [9], [21] and which is addressed in this paper.

To get the most out of intrinsic hardware evolution, and evolve complex and novel systems, an evolvable platform capable of implementing complex behaviours and an unconstrained evolutionary approach is required [21]. However, most of the time unconstrained intrinsic hardware evolution exhibits unusual circuits that can not be evaluated fully by using simple fitness functions that award/penalise a circuit by making bitwise comparisons on the evolved circuit's current and desired outputs. This is due to the transient effects in the hardware, exploited by the unconstrained evolution. When

TABLE I

AMBIGUITIES IN FITNESS ASSIGNMENT FOR AN XOR ARE SHOWN IN THE TABLE. THE GOAL IS TO MINIMISE THE FITNESS. THERE ARE TWO INPUTS, NAMELY A:**0011** AND B:**0101**. THE DESIRED OUTPUT IS THE RESULT OF AN XOR:**0110**. THREE DIFFERENT FITNESS MEASURES, WHICH ARE DESCRIBED IN SECTIONS III-B, III-C ARE COMPARED. THE VALUES IN THE PARENTHESES USED FOR FITNESS CALCULATION REFER TO; FOR BMH: BITWISE, PARITY, TRANSIENT FAULT PENALTY, FOR HIFF: BINARY BLOCK SIZES 1,2,4, AND FOR HBS: BINARY BLOCK SIZES 1,2,4.

case	measuring at		fitness			
	t_0	t_1	bitwise	BMH	HIFF	HBS
A XOR B	0110	0110	0	0 (0+0+0)	0 (0+0+0)	0 (0+0+0)
X_0	0101	0101	4	4 (4+0+0)	16 (4+4+8)	48 (4+12+32)
X_1	0000	0000	4	8 (4+4+0)	20 (4+8+8)	50 (4+14+32)
X_2	0101	0111	3	12 (3+1+8)	15 (3+4+8)	45 (3+10+32)
X_3	0101	1111	4	22 (4+2+16)	18 (4+6+8)	48 (4+12+32)
X_4	0101	1010	4	36 (4+0+32)	16 (4+4+8)	42 (4+10+28)

unconstrained, evolution may explore any solution that is available on the medium, and may sometimes find solutions that are only transient [9], which may trick evolution in to believing that it has found the correct solution. To be able to guide evolution towards more robust and meaningful solutions without constraining its evolvability, an effective yet computationally feasible fitness evaluation method is required. Also, for on-chip intrinsic hardware evolution, the fitness calculations need to be computationally inexpensive.

B. Computational Ambiguities in Fitness Assignment

Even when only considering the computation of fitness for a measured binary string, a great number of ambiguities can mislead the evolutionary optimisation process, as can be seen from the examples listed in table I. In the case of hardware, it is necessary to perform multiple subsequent measurements (indicated as t_0 and t_1), due to possible feedback loops and oscillations. Four different fitness measures are compared for a set of examples of possible solutions to evolve an XOR gate. It can be seen that the bitwise fitness assignment is not able to distinguish among the four different examples shown. Despite these solutions are not considered as equally good, the decision on which one to promote can only be based on random selection in the latter case. BMH, HIFF and the HBS approaches provide finer grained fitness measures, however, note that the distribution of the fitness values is different in each case.

Depending on the chosen measure, the fitness ranking is ambiguous and task dependent. Since the fitness landscape is not known before actually performing sufficient evolutionary runs, it is not always clear which solution is better. Consider for example solution X_0 or X_1 from table I, which cannot be distinguished by the bitwise fitness measure. However, X_0 should be considered as better than X_1, due to the fact that X_1 most likely corresponds to the trivial solution of an unconnected output. BMH, HIFF and the HBS methods overcome this ambiguity and assign better (lower) fitness values to X_0.

The respective rank of X_1 and X_2 is different for BMH than for HIFF and HBS, due to the fact that in the case of BMH, transient faults are additionally penalised, independent of the actual position of the bit that causes the fault. Therefore, despite the fact that X_2 is featuring more correct bits than X_1, it obtains a worse fitness due to the extra penalty for bit 3, which delivers different results for measurements at different times.

The full consequence of using the different fitness measures can be seen in figure 4, where the fitness values for all approaches and all possible results for t_0 and t_1 are plotted.

It can be seen from figure 4 that bitwise comparison has the worst fitness landscape with lots of wide horizontal steps, which is a challenging problem for evolution to tackle. HIFF provides a better landscape than bitwise with smaller steps, whereas BMH and HBS methods provide even better sampling of the landscape with very small horizontal steps.

V. APPLICATION TO HARDWARE

As the main aim of this work is to introduce and adapt new ways of calculating the fitness in order to improve hardware evolution, we have undertaken experiments of evolving an XOR gate on the RISA chip using the aforementioned fitness methods. The goal is to compare their performance when applied to unconstrained intrinsic hardware evolution. Runtime and the ability of finding stable solutions of the different approaches are therefore of particular interest, in order to get an idea about their features and shortcomings.

Furthermore, the evolution of 4 bit parity on a constrained RISA platform is tackled, in order to demonstrate that the novel fitness methods are suitable for different hardware evolution experiments and are not bound to only work for XOR.

A. Setup

The hardware setup includes a PC, a RISA chip, and a Spartan 3 FPGA with on-chip synthesised microblaze microprocessor. For all experiments, a $\mu + \lambda$ evolutionary strategy (ES) is used with two elitists and a population size

TABLE II

Truth table used as input test pattern for XOR experiments.

test case	XOR	A	B
I	0	0	0
II	1	1	0
III	1	0	1
IV	0	1	1

of 7 (2+5). Mutation rate is always 2% and the maximum number of generations is set to 5000. Crossover is not applied.

The EA operates directly on the configuration bit-string for RISA and evolution is completely free to use any resources available (routing and logic) and connect them freely. In the case of the XOR experiment, the 3×3 upper left RISA clusters (see Figure 2) are used, which are connected to predefined two inputs and one output of the chip. Due to each cluster requires 152 bits to configure the logic and 320 bits to configure the routing, an area of 9 clusters corresponds to a bit-string of length 4248 and a search space of size 2^{4248}, which is huge. The experiments are initiated and monitored through a serial port by a PC running a python script.

40 independent evolution runs are carried out using the four fitness evaluation methods described in section III: Bitwise, BMH, HIFF and HBS. For all experiments, an array of 64 input vectors is used for the evaluation of each candidate: for the first 32 entries, each test case I-IV (shown in Figure II) is repeated for 8 times. The second half of the input vector contains 8 times the full input pattern I-IV, half of them in randomised order. Repeatedly testing the same test case and randomising the full input pattern makes it possible to measure transient effects like oscillations and delayed changes of the output. Therefore, through this method it is possible to detect and avoid intermittent results.

For the HIFF and HBS method, the block sizes used for evaluating the output—which is 64 bit wide due to the number of input test vectors—were 1, 2, 4, 8, 16 and 32 bits per block respectively. In the case of HBS there was no wrap around when the sampling window reached the end of the bit-string, to conserve computational resources.

B. Unconstrained Evolution of an XOR

The experimental results of intrinsically evolving an XOR gate on RISA turned out to be supportive of our statements about the shortcomings of the Bitwise fitness calculation method. The data obtained relies on 40 independent evolution runs for each method and the results are listed in Table III. As can be seen from the latter table, the type of fitness calculation method has little effect on the average speed of evolution, which is in each case $\approx$ 3.3 seconds per generation. Thus the higher complexities of BMH, HBS and HIFF fitness calculation methods do not appear to add considerable overhead to the evolutionary process.

The results show that, bitwise fitness calculation method had the lowest evolutionary success rate where it found 16 solutions out of 40 runs and only 10 of these were successful (i.e the other 6 were intermittent, and did not provide stable XOR gates). Whereas BMH method found 34 solutions out of 40 runs with 8 unsuccessful solutions, and HIFF and HBS found solutions in 39 of the 40 runs with 30 and 31 of them being successful respectively. If we look at the average runtime or number generations as well as the standard deviation in the number of generations for each fitness calculation method, it can be seen that bitwise takes longer than any other method in average to finish a run, and HBS is the fastest and most dependable of all four. Thus in the results, HBS proves to be the most reliable, and fastest evaluation technique among the four tested methods.

TABLE III

Evolutionary run results on evolving an XOR gate. 40 experiments for each fitness calculation method has been run, and the results of these runs have been analysed and put together in this table to give an idea of the performance and dependability of each method. Displaying: number of solutions found in 40 runs (Sol Found), number of successful solutions found (Success), average number of generations for each experiment (Avg Gens), standard deviation of average number of generations (Std Dev), average run time for each experiment (Avg RT), run time per generation (RT/Gen).

fitness	results					
	Sol found	success	avg gens	std dev	avg RT (mins)	RT/gen (secs)
bitwise	16	10	3792	1684	206.03	3.26
BMH	34	26	2351	1720	127.65	3.26
HIFF	39	30	1600	1271	87	3.26
HBS	39	31	1364	1054	75.2	3.31

C. Evolution of 4 Bit Parity

Experiments were carried out for intrinsically evolving a 4 bit odd and even parity generator on RISA. Since the aim was to demonstrate that the proposed fitness evaluation methods work equally well on a different problem, rather than provide a benchmark for the proposed fitness functions, the evolutionary runs were done on constrained hardware to diminish the search space. The evolution was only allowed to work on the combinatorial logic resources and a limited amount of routing on all (36) of the clusters on the RISA chip.

The results are shown in table IV. The newly developed fitness functions perform slower on average for each run when compared to bitwise and HIFF, which both perform approximately the same. This is due to the smaller search space (2^{1368}) on a constrained combinatorial-only setup, and the extra transient checks of BMH and finer grained search of HBS does not provide the same benefits to evolution for this sort of configuration. However, the results indicate that the introduced hierarchical fitness evaluation methods perform as good on different problems, and they are not bound to XOR functions.

TABLE IV
EVOLUTIONARY RUN RESULTS ON EVOLVING 4 BIT PARITY GENERATORS. 20 RUNS FOR EACH FITNESS MEASURE ARE CARRIED OUT.

fitness	results			
	avg gens	std dev	best gens	avg RT (mins)
bitwise	305	253	75	64.0
BMH	411	220	80	86.1
HIFF	316	227	118	66.4
HBS	505	382	46	106.4

VI. CONCLUSIONS

In this paper we describe results from experiments with intrinsic unconstrained evolution on RISA, a unique evolutionary platform. This is the first time such experiments have been undertaken. Furthermore, We have investigated the performance of four different fitness functions and their suitability for unconstrained intrinsic hardware evolution. Two fitness measures, bitwise and HIFF [22], have been adapted to the RISA evolvable hardware platform and two additional measures, BMH and HBS, have been newly developed that target hardware implementation. All four approaches have been successfully applied to the unconstrained evolution of an XOR gate, as well as a 4-bit even and odd parity generator.

It is shown that the conventionally used bitwise fitness measure fails to resolve ambiguities in the hardware measurements, which are caused by transient effects, and therefore in many cases only finds intermittent results, rather than stable solutions. Due to its coarseness, the Bitwise approach requires on average the largest number of generations before finding a solution. In contrast to this, the convergence towards finding solutions is improved in the case of BMH, HIFF and HBS. It is observed that the hierarchical approaches (HIFF and HBS) feature the best performance in finding stable solutions. In accordance with convergence speed, the average runtime of a run using bitwise fitness calculation is significantly longer than the average runtime of runs using the other methods. HBS requires less than half of the time to find a solution compared with bitwise techniques.

Despite the increased computational complexity of BMH, HIFF and HBS, the time required to process one generation remains almost the same, which is due to the fact that evaluation time is dominated by the time required for configuring the chip. It is satisfying to observe that this is not a drawback when using the more powerful methods.

Concluding, our results show that the implemented new fitness evaluation methods, HBS in particular, perform more reliably and as quickly as simple bitwise evaluation in finding solutions using unconstrained intrinsic evolution on a dynamic hardware platform, where feedback loops are allowed. Thus, hierarchical fitness evaluation methods are proven to be particularly effective for hardware evolution. While both the example circuits evolved are rather simple, there is no suggestion that these evolved circuits are of use in themselves. Rather than focussing on more complex tasks, we have elucidated underlying principles related to fitness evaluation that are generic to intrinsic evolvable hardware.

VII. FUTURE WORK

The unconstrained evolution of more complex circuits using hierarchical fitness evaluation will be tackled in future experiments. This will vastly increase the search space and therefore make it considerably more difficult for the EA to find solutions, however, it will only be possible to entirely investigate an evolvable hardware substrate and possible pathways of evolution, if the algorithm is carried out in an unconstrained fashion.

It is suggested that the fitness calculation methods presented in this paper could be equally suitable for sequential circuits, if the starting point of the target pattern is not restricted, i.e. the desired output is a sequence based on the first sample of the measured output, rather than a fixed pattern. Sequential circuits will be targeted in future experiments.

Additionally, the evolvability of the current FPGA architecture and the genetic representation will be observed, as this is especially crucial for successful evolution of digital circuits [8].

Further experiments will address the scalability of RISA. A major question will be whether unconstrained evolution will still be feasible, since the search space will exponentially increase as the system grows. Partitioning the tasks and hierarchical problem solving will be tackled to overcome these problems [1]. Finally, developmental methods will be implemented as an approach to improve scalability [2].

REFERENCES

[1] E. De Jong, R. A. Watson, and D. Thierens, "On the complexity of hierarchical problem solving," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, 2005.

[2] T. Gordon and P. Bentley, "Towards development in evolvable hardware," 2002. [Online]. Available: citeseer.ist.psu.edu/gordon02towards.html

[3] A. Greensted and A. Tyrrell, "Extrinsic evolvable hardware on the RISA architecture," in *Proceedings of 2007 International Conference on Evolvable Systems*, September 2007.

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

[5] G. W. Greenwood and A. M. Tyrrell, *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems*, ser. IEEE Press Series on Computational Intelligence. Wiley-IEEE Press, 2007.

[6] P. Haddow and G. Tufte, "Evolving a robot controller in hardware," in *In Proc. of the Norwegian Computer Science Conference (NIK-99)*, 1999, pp. 141–150.

[7] P. C. Haddow and G. Tufte, "Bridging the genotype-phenotype mapping for digital fpgas," in *EH '01: Proceedings of the The 3rd NASA/DoD Workshop on Evolvable Hardware*. Washington, DC, USA: IEEE Computer Society, 2001, p. 109.

[8] ——, "Bridging the genotype-phenotype mapping for digital fpgas," in *EH '01: Proceedings of the The 3rd NASA/DoD Workshop on Evolvable Hardware*. Washington, DC, USA: IEEE Computer Society, 2001, p. 109.

[9] S. Harding, "Evolution in materio," Ph.D. dissertation, University of York, 2006.

[10] G. Hollingworth, S. Smith, and A. Tyrrell, "The intrinsic evolution of virtex devices through internet reconfigurable logic," in *3rd International Conference on Evolvable Systems: from Biology to Hardware*. Edinburgh,: Springer-Verlag, April 2000, pp. 72–79.

[11] ——, “The safe intrinsic evolution of virtex devices,” in *2nd NASA/DoD Workshop on Evolvable Hardware*, Silicon Valley, USA, July 2000.

[12] L. Huelsbergen, E. A. Rietman, and R. Slous, “Evolving oscillators in silico.” vol. 3, no. 3, 1999, pp. 197–204.

[13] N. Jha and S. Gupta, *Testing of Digital Systems*, 1st ed. Cambridge University Press, 2003.

[14] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[15] W. B. Langdon and R. Poli, “Genetic programming bloat with dynamic fitness,” in *EuroGP ’98: Proceedings of the First European Workshop on Genetic Programming*. London, UK: Springer-Verlag, 1998, pp. 97–112.

[16] H. Liu, J. F. Miller, and A. M. Tyrrell, “Intrinsic evolvable hardware implementation of a robust biological development model for digital systems,” in *EH ’05: Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 87–92.

[17] S. Luke and L. Panait, “Fighting bloat with nonparametric parsimony pressure,” in *PPSN VII: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*. London, UK: Springer-Verlag, 2002, pp. 411–421.

[18] ——, “A comparison of bloat control methods for genetic programming,” *Evol. Comput.*, vol. 14, no. 3, pp. 309–344, 2006.

[19] M. Salami and T. Hendtlass, “A fitness estimation strategy for genetic algorithms,” in *IEA/AIE ’02: Proceedings of the 15th international conference on Industrial and engineering applications of artificial intelligence and expert systems*. London, UK: Springer-Verlag, 2002, pp. 502–513.

[20] S. Silva and E. Costa, “Resource-limited genetic programming: the dynamic approach,” in *GECCO ’05: Proceedings of the 2005 conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2005, pp. 1673–1680.

[21] A. Thompson, I. Harvey, and P. Husbands, “Unconstrained evolution and hard consequences,” in *Towards Evolvable Hardware: The evolutionary engineering approach*, ser. LNCS, E. Sanchez and M. Tomassini, Eds. Springer-Verlag, 1996, vol. 1062, pp. 136–165.

[22] R. A. Watson, G. S. Hornby, and J. B. Pollack, “Modeling building-block interdependency,” *Lecture Notes in Computer Science*, vol. 1498, pp. 97–106, 1998.