CSRP 572 – Low Overhead Self-Checking Combinational and Sequential Circuits Designed by Evolution

Miguel Garvie and Adrian Thompson
Centre for Computational Neuroscience and Robotics, Department of Informatics,
University of Sussex, Brighton BN1 9QH, UK.
m.m.garvie, adrianth @sussex.ac.uk , +44 (0)1273 872945, fax +44 (0)1273 877873

Abstract

Evolutionary techniques are applied to the design of self-checking digital circuits in simulation. For the combinational and sequential benchmarks attempted, evolved designs are totally self-checking with respect to single stuck-at faults in mission logic, have no latency and use significantly less resources than hand-designed equivalents. The approach can be extended to evolve fail-safe circuits, analog self-test, and self-checking checkers under multiple faults.

1 Introduction

As hardware is increasingly used in mission critical applications such as medicine, transport, space and industry, avoiding faulty circuit behaviour can save lives and money. Permanent and transient hardware faults occurring in the field become of increasing concern as component density increases, costs are minimised in mass produced products, and hardware is deployed in more hostile environments. This paper demonstrates that artificial evolution can be used to design self-checking circuits superior to the best equivalents found in the literature for small benchmarks.

Off-line built-in self-test (BIST) techniques [31, 37, 10] are used to scan large circuits for faults post-manufacturing. Some circuits with on-line BIST using scan methods require the mission logic to go off-line temporarily or exhibit fault latency [28, 3]. *Self-checking* (SC) circuits perform concurrent error detection during normal operation, are necessary for detecting transient faults and are desirable in mission critical applications where mission hardware must neither stop nor fail without warning. *Self-testing* [30] circuits containing a fault will have a non-codeword at the outputs for some input codeword. *Fault-secure* circuits containing a fault will under all input codewords either be unaffected or output a non-codeword. *Totally Self-Checking* (TSC) circuits are self-testing and fault-secure. Sequential TSC circuits must also produce non-codeword outputs for false state transitions. The most stringent SC designs are TSC checkers which under multiple faults to mission and testing logic satisfy the following: "*the first erroneous output resulting from an internal fault of the digital circuit is detectable.*"

Traditional SC architecture involves duplicating mission logic – sometimes with diversity [2] or inversion with two-rail logic – and comparing the duplicated outputs, thus achieving full fault-coverage of mission logic. Error detecting codes (EDC) such as Hamming, parity [26], 1-of-n [11] and Berger [24] can achieve lower overhead often at the expense of fault coverage. Berger codes are used in self-checking checkers detecting multiple unidirectional faults under certain assumptions. SC Synchronous Sequential Circuit (SSC) design methods have several limitations [27] yet sometimes achieve error recovery under transient faults [19] and there have been cases of less than duplication overhead [5] with near full fault coverage. It is generally accepted [11] to be unrealistic to expect low overhead for TSC random logic. However as other forms of testing become increasingly expensive [25] and as mission critical hardware becomes widespread, such low overhead TSC circuits will be highly desirable.

Evolutionary methods [13, 9, 16] such as genetic algorithms (GAs) applied to hardware [12, 22, 29, 17, 23, 32, 20] have produced circuits comparable to those designed by experts [23, 36, 21, 17] and also unconventional circuits [34, 18] in which hardware resources are used extremely efficiently. Moreover, many evolved systems in nature exhibit self-diagnostics such as the lymphatic system [1]. This leads to the possibility that evolutionary methods could explore areas of design space which reuse hardware components so that they contribute both to the circuit's main functionality and its BIST, leading to low overhead SC solutions.

This hypothesis was proved in [8, 7] where TSC full adders and two bit multipliers were evolved using around half the overhead of a duplicate and compare (D&C) strat-

egy. A TSC edge-triggered D-latch (ETDL) was also evolved equivalent to D&C. This previous work has established that the creativity of evolutionary search can be used to reach solutions to the TSC problem, finding solutions competitive with conventionally designed ones. Also, several principles of operation were extracted from evolved designs which could be of use to add to a designer's toolset. The questions addressed by this paper are:

1. Can evolution arrive at low overhead TSC sequential circuits?
2. What more principles of operation can we extract from evolved circuits? Are they all ad-hoc? Is there one operating principle common to all of them?
3. Does this method scale up for larger circuits?

Section 2 will describe the GA, the simulator, the fault model, the tasks to be evolved and the fitness evaluation mechanism for SC. Section 3 presents and discusses the results achieved while section 4 reports what was learned and future avenues.

2 Method

2.1 The Genetic Algorithm

A generational Genetic Algorithm (GA) is used with a population size of 32 with 2 elites where 80% of the next generation is created through mutation and the rest by single-point crossover.

The circuits to be evolved consisted of configurable logic blocks (CLBs, inspired by those of FPGAs) each composed of a look-up table (LUT) and, in the case of sequential circuits, an ETDL. The LUT contents and the routing information for inputs were encoded as a binary string (the genotype). Half of mutations affect single genotype bits, a quarter randomize a LUT's contents while the remaining quarter randomize a routing connection. Following from earlier work [34] we adopted the model of a small genetically semi-converged population evolving for many generations. Fitness ranges from 0 (worst) to 1 (best). Linear rank selection is used, such that the elite has twice the selective advantage of the median of the population.

The genotype-phenotype mapping used is similar to [23] excepting that: there are no limitations on connectivity allowing recurrent circuits, the genotype is encoded in binary, and in the case of combinational tasks the outputs are taken from predefined CLBs. For sequential tasks, the routing specification for a CLB was extended to describe the input source for the latch. All routing specifications were then augmented by an extra bit indicating whether a connection is driven by the LUT or the ETDL in the source CLB. The genotype length depends on the task being evolved. To allow implementation across a network of approximately 200 2GHz processors [6], an island based model [33] with 2D grid topology was used with a low migration rate.

2.2 The Simulator

The simulator used is a simple version of an event driven digital logic simulator in which each logic unit is in charge of its own behaviour when given discrete time-slices and the state of its inputs. For combinational circuits logic units are LUTs of two inputs capable of representing any two input logic gate. For sequential circuits half of the logic units are LUTs of four inputs and the rest ETDLs. Any unit can be connected to any other allowing recurrence, so care must be taken to update all units simultaneously. Time-slices are sent to the logic units in two waves: the first to read their inputs and the second to update their outputs. During the presentation of each input vector, circuit inputs were kept stable for 30 time-slices and the outputs were read in the second half of this cycle allowing them time to settle. For sequential tasks the clock fed into the latches was high during the first 15 of these steps and low during the second half.

Gate delays are simulated in time-slice units and are randomised with a Gaussian distribution ($\lfloor \mu = 1.5, \sigma^2 = 0.5 \rfloor$). This amounts to a noisy fitness evaluation with the intention to facilitate transfer of sequential circuits to real hardware as in a 'Minimal Simulation' [14]. At the start of each generation, e different sets of logic delays are generated, simulating e different variations to the circuit delays from manufacturing or environmental variation. The performance of each circuit in the population is then evaluated in each of the e conditions, and its fitness is the mean value. The number e is occasionally adjusted by hand during the run, such that more evaluations are used as the population leaves the "exploring" stage and enters the "exploiting" stage. It was set such that twice the standard error of the series of fitness trials is significantly smaller than the difference in fitness between adjacent individuals in the rank table with non-equal fitnesses.

The Single-Stuck-At (SSA) Fault model is adopted as it covers most transient single event upsets [31, 15] and is an industry standard. It has been shown that tests generated for SSA faults are also good at detecting other types of faults. SSA faults can be introduced at any of the logic units of the simulator simply by setting its output always to 0 or 1.

2.3 The Tasks

Combinational and sequential benchmark circuits for which to evolve SC were taken from the MCNC '91 test suite. All circuits were first optimised and synthesised using

SIS [4] using the script recommended in the manual for Xilinx LUT architectures. In order to save time and given that evolution is able to modify hand-designed circuits [8, 36] every run was seeded with the synthesised design of the mission logic. In the runs referred to as *Locked* the mission logic is immutable and evolution must find ways of performing SC without modifying it.

Even when evolving combinational circuits the evolving networks may be recurrent and could show an unwanted dependence on the order in which inputs are presented, and on the networks' internal state. To demand insensitivity to input ordering, the same approach was taken as for the randomization of logic delays (above): at the start of each generation, e (the same number e defining the number of evaluations with random gate delays above) different orderings of the full set of possible inputs for that task were generated, and the individuals of that generation evaluated on all of them. On each of the e evaluations the circuit state was reset, then the ordering of the full set of inputs was presented twice in sequence, to prevent dependence on initial conditions.

The same procedure was carried out for the sequential task excepting the random test pattern generation: a directed graph built from the Moore FSM of the benchmark circuit is built such that some nodes are assigned as reset states and enough edges are marked as permanent so that all states can be reached by moving along them. We first choose a reset state to start from and add the input pattern necessary to bring the FSM to this state from any other onto the generated test pattern. A random walk is now begun such that walking along an edge appends its input vector to the generated pattern and removes it from the graph unless it is marked permanent. The walk ends when all edges have been walked along thus ensuring that the random test pattern generated will test all FSM state transitions.

The task evaluation score was measured as follows. Let Q_r be the series of values at the r^{th} output bit for the final 15 time-slices of the presentation of each input, concatenated over all input vectors presented during an evaluation, and Q'_r the desired response. We take the modulus of the correlation of Q_r and Q'_r, averaged over all N outputs:

$$f_{\text{t}} = \frac{\sum_{r=0}^{N-1} |corr(Q_r, Q'_r)|}{N} \qquad (1)$$

2.4 Evolving Self-Checking

An extra output E was recorded from circuits with the aim that it would go high whenever a fault affected any other output. The performance of a circuit at its main task f_{t} and at SC behaviour f_{c} were evaluated separately. SC behaviour was evaluated with three fitness measures:

1. Per fault f_{c_F}: Let u_{f} be the number of faults affecting task performance for which none of the possible input vectors raises E. Then f_{c_F} encourages faults to be 'detectable': $f_{\text{c}_\text{F}} = 1/\left(1 + u_{\text{f}} \times k_{\text{f}}\right)$ where k_{f} was chosen to be 25, to give f_{c_F} good sensitivity when u_{f} is small.

2. Per instance f_{c_I}: Let SFI denote a combination of circuit state, SSA fault, and input vector. Define u_{i} to be the number of instances out of all possible SFI combinations for which the task output is incorrect but E is low. Then f_{c_I} encourages immediate detection of faults: $f_{\text{c}_\text{I}} = 1/\left(1 + u_{\text{i}} \times k_{\text{i}}\right)$ where k_{i} was chosen to be 200.

3. Per transition f_{c_T}: Let u_{t} be the number of SFI instances for which mission outputs are unaffected, E is low, but the transition to the next state is incorrect. f_{c_T} discourages FSM failure: $f_{\text{c}_\text{T}} = 1/\left(1 + u_{\text{t}} \times k_{\text{t}}\right)$ where k_{t} was chosen to be 50.

f_{c_F} and f_{c_I} were used when evolving SC combinational benchmarks. If we define output codewords to have E low then f_{c_F} measures self-testing and f_{c_I} also measures fault-security, so a circuit with $f_{\text{c}_\text{F}} = f_{\text{c}_\text{I}} = 1$ is TSC. f_{c_I} and f_{c_T} were used for sequential benchmarks and $f_{\text{c}_\text{I}} = f_{\text{c}_\text{T}} = 1$ is required for TSC.

u_{f} is measured by evaluating task fitness f_{t} separately under all SSA faults to every unit able to affect the task outputs. The same set of e evaluation conditions chosen for the current generation is used. If f_{t} falls by at least 0.01 due to a fault, then it is considered to affect task performance. u_{i} is measured by comparing the output of the circuit at each SFI combination with its output for the same state and input under no faults. Outputs are deemed unaffected if they are not different for more than 5 time steps of the 15 over which they are monitored. To test sequential circuits at particular states they are run fault-free with the randomly generated test pattern and a circuit state snapshot is saved as they enter each state for the first time. At each state, for each fault, all input vectors are applied and undetected failure instances are counted. When testing a new fault at a given state, the circuit state snapshot is restored before the fault is inserted. When all faults have been tested the snapshot is restored and the test pattern execution is resumed until the next untested state. For sequential benchmarks, u_{t} is measured as follows. Every time a SFI combination neither affects an output nor raises E, the circuit is clocked into the next state in order to evaluate whether the state transition is performed correctly. If not, and E is low, then u_{t} is incremented.

If during the simulated time E goes high for more than $eSize$ consecutive time slices then it is considered to have gone high. $eSize$ is set at 14: greater than average race conditions yet not excluding a wide range of behaviours. A

circuit exhibiting a high E when no faults were in place was deemed to have $f_{c_F} = f_{c_I} = f_{c_T} = 0$.

It seems foolish to concentrate on the detection of faults at every instance when dealing with a circuit in which some faults are not detected at any instance. Fitness objectives were given hierarchical priorities. For combinational benchmarks the priorities in descending order where f_t, f_{c_F}, f_{c_I} while for sequential benchmarks they were f_t, f_{c_I}, f_{c_T}. When sorting the individuals for rank selection the comparison operator only considered an objective if higher priority objectives were equal. An extra objective encouraging parsimony was also used, having the lowest priority of all.

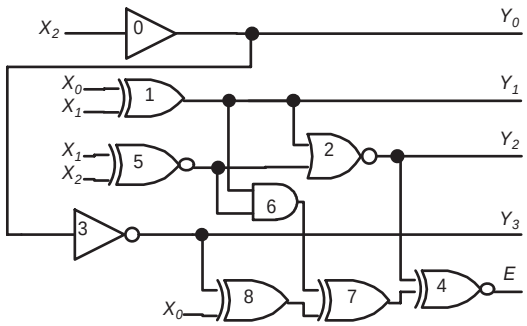

Figure 1. HENMANIAC: a TSC b1 benchmark circuit with 23% D&C overhead.

3 Results

The maximum number of logic units available to evolution was constrained between 30 and 110 (the genotype length consequently ranged from 300 to 2200 bits) and the time evolution was allowed to run varied from several hours to several weeks. These factors depended monotonically on the size of the benchmark. These runs were performed by the concerted effort of approximately 200 2GHz computers contributed by volunteers around the world [6].

SC circuits with full fault coverage and no error latency over SSA faults in mission logic were evolved for all benchmarks attempted. They all satisfy the TSC condition since during their operation E goes high before or immediately after outputs are incorrect due to a fault. All evolved results can be viewed and tested on-line in a visual digital logic simulator [6].

Table 1 compares the overhead of evolved self-checkers with equivalents produced by other means reported in the literature. On average, evolved TSC circuits require only 66.9% overhead over mission logic. This is an average 42.0% of the duplicate & compare (D&C) overhead. This is significantly lower than the average 73.2% of D&C overhead required by the best previous results out of [24, 11, 5] for this test suite.

We will now look at three evolved TSC circuits: HENMANIAC performs the b1 combinational benchmark, SMILODON and Oryx perform the mc sequential benchmark. By error at U_n we mean that unit n is behaving differently from how it would if there were no faults in the circuit. For example there is no error at $U_x = a + b$ if it is SSA 1 and $a = 1$, but there would be if $a = b = 0$.

3.1 HENMANIAC

This TSC b1 benchmark circuit is composed of 9 gates and detects all SSA faults affecting mission behaviour with a maximum 3 gate delay latency. Even though the Xilinx LUT synthesis procedures recommended in SIS produced a circuit with 6 gates for the b1 benchmark, the evolved circuit in Fig. 1 requires only 5 gates for mission outputs. This already makes the mission logic smaller and reduces the amount of potential faults. Even taking the mission logic size to be 5 gates, the 4 gate overhead is still 33.3% of the 12 required for a D&C approach.

The circuit exploits non-canalyzing gates: these are gates for which a bit-flip at any input will always change the output. Non-canalizing gates at outputs are cascaded such that Y_0 is fed to Y_3 and Y_1 to Y_2 propagating errors and reducing the outputs to be checked without incrementing the mission input–output delay of 2 gates. Units 6,7 and 8 recalculate the inverse of Y_2 using Y_3 and reusing units 1 and 5. The error output E simply compares Y_2 against unit 7, going high when they are equal thus detecting an error at U_2. There is a non-canalizing path U_0=Y_0, U_3=Y_3, U_8, U_7, U_4=E propagating all errors in these units up to the error signal. Errors at units 1 and 5 always affect either Y_2 or U_7 – never both – and are thus signalled at E. Given that errors at U_6 will also be propagated down to E then this circuit is TSC with respect to faults in all units except U_4 (E) stuck-at 0.

3.2 SMILODON

This TSC circuit performs the mc benchmark detecting all SSA faults in mission logic which affect output or transitions. Fault latency is less than a clock cycle. From its specification in table 2 we see it has a complex interconnectivity of four-input LUTs and latches. Mission output units are influenced by all other logic units so there is no straightforward separation of mission and testing logic as there was in HENMANIAC. There are however some duplicated LUTs ($U_7 = U_{10}$) and latches ($U_5 = U_{12}$). Output $Y_1 = U_1$ is fed by U_{11} which in turn depends on outputs $Y_2 = U_2$ and $Y_4 = U_3$ suggesting an output cascading approach as above. This output $Y_1 = U_1$ is then fed into the error output $E = U_4$ together with latches U_8 and U_{13}

Table 1. Overhead comparison between evolved (E) self-checkers, D&C (DC) and the previous best (PB) of [24, 11, 5]. All evolved self-checkers provide full fault coverage of SSA faults to mission logic with no latency. Number of circuit inputs, outputs and states where applicable are found in the $I/O[/S]$ column. Gate count is in terms of two-input LUTs for combinational and four-input LUTs and ETDLs for sequential benchmarks. M is the gate count for the synthesised and optimised mission logic. DC, E_L, E are gate overhead for the D&C, locked evolved and fully evolved approaches. The next column provides overhead as percentage of original mission logic. The last three columns provide overhead as percentage of D&C overhead for the locked evolved, fully evolved approach and for the best reported conventional design result. Previous best results marked with † provide less than 95% fault coverage. The result marked * was calculated here by adopting a conservative estimate D&C overhead of 61 NANDs.

Benchmark	$I/O[/S]$	M	DC	E_L	E	$\frac{E}{M}\%$	$\frac{E_L}{DC}\%$	$\frac{E}{DC}\%$	$\frac{PB}{DC}\%$
C17	5/2	6	9	9	8	133	100	88.9	
b1	3/4	6	13	4	3	50	30.7	23.0	74.3†
cm42a	4/10	18	37	13	12	66.7	35.1	32.4	67.4
decod	5/16	26	57	21	17	65.3	36.8	29.8	60
cm138a	6/8	16	31	11	10	62.5	35.5	32.2	79.1
dk27	1/2/7	8	9	7	5	62.5	77.8	55.6	
mc	3/5/4	10	13	8	5	50	61.5	38.5	85.2†*
beecount	3/4/7	11	14	10	5	45.4	71.4	35.7	
Average						66.9	56.1	42.0	73.2

Table 2. SMILODON: a TSC mc sequential benchmark using four-input LUTs and ETDLs. C is the clock input. Circuit outputs $Y_0 \ldots Y_4$ are read from $U_0, U_1, U_2, \overline{U_1}$ and U_3 respectively.

Unit	LUT function / Latch	Unit inputs
U_0	$\bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c}d + \bar{a}b\bar{c}\bar{d} + a\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}d + ab\bar{c}\bar{d} + ab\bar{c}d + abc\bar{d} + abcd$	X_2, U_5, U_{10}, U_{10}
U_1	$\bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}b\bar{c}\bar{d} + a\bar{b}\bar{c}\bar{d} + a\bar{b}c\bar{d} + a\bar{b}cd + abcd$	$U_8, U_{10}, U_{11}, U_{13}$
U_2	$\bar{a}\bar{b}cd + \bar{a}b\bar{c}\bar{d} + a\bar{b}\bar{c}d + ab\bar{c}\bar{d} + ab\bar{c}d + abc\bar{d} + abcd$	U_8, U_5, U_9, U_{10}
U_3	$\bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}c\bar{d} + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + a\bar{b}\bar{c}d + a\bar{b}c\bar{d} + ab\bar{c}\bar{d} + abcd$	U_8, U_5, U_0, U_6
U_4	$\bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}cd + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd + a\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}d + a\bar{b}cd$	U_{13}, U_1, U_8, U_8
U_5	Edge Triggered D-Latch	C, U_6
U_6	$\bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd + a\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}d + ab\bar{c}\bar{d}$	X_2, U_5, U_7, X_1
U_7	$\bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}cd + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd + a\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}d + a\bar{b}c\bar{d} + a\bar{b}cd + ab\bar{c}d + abcd$	X_0, X_1, U_8, U_5
U_8	Edge Triggered D-Latch	C, U_9
U_9	$\bar{a}\bar{b}cd + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + a\bar{b}c\bar{d} + a\bar{b}cd + ab\bar{c}\bar{d} + ab\bar{c}d$	X_2, U_8, U_5, U_0
U_{10}	$\bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}cd + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd + a\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}d + a\bar{b}c\bar{d} + a\bar{b}cd + ab\bar{c}d + abcd$	X_0, X_1, U_8, U_5
U_{11}	$\bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}cd + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + a\bar{b}\bar{c}d + a\bar{b}cd + ab\bar{c}\bar{d} + ab\bar{c}d + abc\bar{d}$	U_3, U_2, U_3, U_{12}
U_{12}	Edge Triggered D-Latch	C, U_6
U_{13}	Edge Triggered D-Latch	C, U_{14}
U_{14}	$\bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}cd + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd + a\bar{b}c\bar{d} + a\bar{b}cd + ab\bar{c}\bar{d} + ab\bar{c}d$	X_2, U_8, U_5, U_4

whose sources U_9 and U_{14} are similar. Even though at first sight its operation seems to be using output cascading together with some recalculation as above (but with a non-modular structure similar to the multiplier in [7]), a more thorough investigation by observing dynamic circuit operation and analyzing LUT functions would provide more insight possibly uncovering novel useful principles of operation. However it is not the first time that efficient evolved circuits elude our understanding [35] due to evolution's capacity to exploit available resources fully without caring for modularity or other design principles. This circuit uses 15 units compared to 23 required for a D&C approach.

3.3 Oryx

This TSC circuit was not allowed to modify the 8 four-input LUTs and two latches generated by SIS for the mc benchmark. Two latches and 6 LUTs were added detecting all SSA faults in mission units. Some of these units are nearly duplicates of mission ones, others are fed the mission outputs together with circuit inputs. All 5 outputs are checked without modifying mission logic and with less than duplication (let alone comparison) overhead. It is possible that by such unconstrained evolution, an optimal EDC scheme has been found for this circuit.

3.4 Discussion

Even if LUT gate count is not an accurate measure of circuit area for every technology, this measure of overhead favours D&C as much as evolved circuits. For example the D&C overhead for the sequential benchmark dk27 is a copy of the optimised 8 units plus a single four-input LUT to compare both outputs. It would be possible to perform mapping onto a desired technology prior to parsimony fitness evaluation. It is believed that the power of evolution to exploit particular technologies would result in even greater overhead gains.

Some evolved SC circuits may have worse timing characteristics due to output cascading. However in many cases the amount of gate delays from inputs to outputs was not increased with respect to the original synthesised version. Moreover, circuit modifications are also required in order to make circuits unidirectional for use with Berger codes. Even then, the evolved TSC circuits with unmodified (locked) mission logic required an average 56.1% of D&C overhead still significantly lower than the previous best equivalent. The mission logic in these circuits has exactly the same specifications as before SC was added. This method could then be applied to specially designed circuits such as high speed or unmodifiable IP cores without disrupting their characteristics. If SC circuits with a certain characteristic were required, all that would be necessary would be to add extra pressure in the fitness function. For example a fourth fitness measure selecting for circuits with low input–output delay could be added. In this way the nature of evolved circuits is highly customizable.

HENMANIAC can be trivially made into a self-checking checker under multiple faults under the assumption that *all input vectors are applied between faults arriving*. Replace U_4 so that E is taken from the output of one of two XOR gates connected back to back such that each of their outputs is an input of the other and the remaining inputs are taken from Y_2 and U_7. Now define an error signal or non-codeword to be when E is non-oscillating. Now E will oscillate whenever $Y_2 \neq U_7$ and will stop otherwise. Since all SSA faults in all units except U_4 were detectable as soon as they caused errors at outputs, and SSA faults in the oscillating XORs will be detected immediately, then all SSA faults in all units are now detectable for some input vector and they never produce errors at any output without oscillation stopping. Thus this modified HENMANIAC is a totally self-checking checker under any amount of faults under the assumption above. After observation, various evolved TSC circuits operate in similar ways as HENMANIAC and could be modified to be self-checking checkers in a similar fashion.

Further analysis of SC mechanisms used by evolved circuits may arrive at mathematical models of EDC used. It is possible that such codes are tailored for each circuit by evolution finding the optimal one based on its characteristics.

Better results may have been found by evolution if given more time to run. It often took days of evolution to reduce the size of a circuit already fulfilling the TSC fitness criteria. These are not known to be optimal solutions since evolving for a few more days may have reduced the size even further. Our experience is that while there are better solutions to be found evolution will make progress. If there are any better solutions then provided with enough processing power and time evolution is likely to find them.

The possibility that evolved SSC had hidden states or false transitions to states with equivalent I/O was ruled out by confirming their appropriate behaviour under 10000 random test patterns of length 1000 in which faults were inserted at random positions.

4 Conclusion

The questions addressed in §1 can now be answered. Evolution has been used to automatically generate TSC sequential circuits using an average 43.3% of D&C overhead. This method scales up for larger combinational benchmarks of 26 two-input gates generating TSC circuits requiring on average 41.3% of D&C overhead. The overhead of these evolved TSC circuits with full SSA fault coverage and zero latency is significantly lower than that of the best equiva-

lents found in the literature. In future as increasing amounts of processing power are available and the methodology matures, larger benchmarks should be within reach.

The principles of operation of evolved circuits are similar to those in [8, 7] such as cascading outputs, using non-canalizing functions and reusing units in a duplicated section such that errors at the units only affect the original or the duplicate. Even though the circuits analyzed share some of these, there may be uniquely tailored EDCs for each of them. The use of this method is two-fold. On the one hand TSC evolved circuits can themselves be used in mission critical applications. On the other, their analysis may provide understanding of their operation and this learning could be added to design techniques so that such low overhead TSC circuits could be synthesised without resorting to evolutionary search. This would not be the first time engineering has learnt from evolved designs such as those found in nature.

There are opportunities for extension of this work such as attempting larger benchmarks, other technology mappings, and including other circuit requirements such as timing. At the time of writing, self-testing checkers are being evolved detecting multiple faults assuming or not that all input vectors are applied between fault arrivals. TSC checkers without this assumption with full fault coverage and no latency are the ultimate goal of SC design and one such circuit has already been evolved. Future avenues also include evolution of SC analog hardware.

4.1 Acknowledgments

Thanks to Andy Balaam for campaigning advice, to the patient people at COGS helpdesk, the COGS bursary that supports Miguel Garvie's research and to all contributors of the Distributed Hardware Evolution Project of which there is only space to mention the top 20 processing power contributing islands and teams:

stephen_boulet, Proto_Clown, HisMastersVoice, TheOneIsland, Sewer_Urchin, Bigparsnip, Professor-Booty, Feral_Boy, Pineapple_Pokopo, sozo, demonie.com, safemode, Omnipotus, KentMein, sonicbadger, Der_Fledermaus, safemode2, Brainchild, macka3, American_Maid, and teams Free-DC, US-Distributed, AMD_Users, Changelings_Crew, teamsafe, Akihabara, Poland, Rechenkraft.net_Germany, faf, SWilcoxon, DPC, Cambridge, Team_351, Finland, Toker-Ring, prescott, Smartpals, The_Euler_Matrix, DontPanic, Landgre

References

[1] A. Avizienis. Design diversity and the immune system paradigm: Cornerstones for information system survivability, 2000.

[2] A. Avizienis and John P. J. Kelly. Fault-tolerance by design diversity: Concepts and experiments. *Computer*, 17(8):67–80, August 1984.

[3] D.W. Bradley and A.M. Tyrrell. Immunotronics: Novel finite state machine architectures with built in self test using self-nonself differentiation. *IEEE Transactions on Evolutionary Computation*, 6(3):227–238, 2001.

[4] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, 1992.

[5] F. Gao and J. Hayes. On-line monitor design of finite-state machines. In *Proc. of* 8^{th} *IEEE International On-Line Testing Workshop*, pages 74–78. IEEE Computer Society, July 2002.

[6] M. Garvie. Distributed Hardware Evolution Project `http://www.cogs.susx.ac.uk/users/mmg20/dhe`.

[7] M. Garvie and A. Thompson. Evolution of combinational and sequential on-line self-diagnosing hardware. In J. Lohn, R. Zebulum, J. Steincamp, D. Keymeulen, A. Stoica, and M. Ferguson, editors, *Proc. 2003 NASA/DoD Conf. on Evolvable Hardware*, pages 167–173. IEEE Computer Society, 2003.

[8] M. Garvie and A. Thompson. Evolution of self-diagnosing hardware. In A. Tyrrell, P. Haddow, and J. Torresen, editors, *Proc. 5th Int. Conf. on Evolvable Systems (ICES2003): From biology to hardware*, volume 2606 of *LNCS*, pages 238–248. Springer-Verlag, 2003.

[9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison Wesley, 1989.

[10] H. Golnabi and J. Provence. RMBITP: A reconfigurable matrix based built-in self-test processor. *Microelectronics Journal*, 28:115–127, 1997.

[11] M. Gossel, Vl. Saposhnikov, , A. Dmitiriev, and Vl. Saposhnikov. A new method for concurrent checking by use of a 1-out-of-4 code. *Proceedings of 6th IEEE International On-Line Testing Workshop*, pages 147–152, July 2000.

[12] T. Higuchi, M. Iwata, and L. Weixin, editors. *Proc. 1st Int. Conf. on Evolvable Systems: From Biology to Hardware*, volume 1259 of *LNCS*. Springer-Verlag, 1997.

[13] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975.

[14] N. Jakobi. Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics. In Phil Husbands and Inman Harvey, editors, *Proc. 4th Eur. Conf. on Artificial Life (ECAL'97)*, pages 348–357. MIT Press, 1997.

[15] R. Katz. The NASA/GSFC Radiation Effects and Analysis Home Page. `http://radhome.gsfc.nasa.gov/`.

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

[17] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming. In T. Higuchi, M. Iwata, and L. Weixin, editors, *Proc. 1st Int. Conf. on Evolvable Systems: From biology to hardware (ICES-96)*, number 1259 in LNCS, pages 312–326. Springer-Verlag, 1996.

[18] P. Layzell. A new research tool for intrinsic hardware evolution. In M. Sipper, D. Mange, and A. Pérez-Uribe, editors, *Proc. 2nd Int. Conf. on Evolvable Systems (ICES'98)*, volume 1478 of *LNCS*, pages 47–56. Springer-Verlag, 1998.

[19] I. Levin, V. Sinelnikov, M. Karpovsky, and S. Ostanin. Sequential circuits applicable for detecting different types of faults. In *Proc. of* 8^{th} *IEEE International On-Line Testing Workshop*, pages 44–48. IEEE Computer Society, July 2002.

[20] J. Lohn, A. Stoica, D. Keymeulen, and S. Colombano, editors. *Proc. 2nd NASA/DoD workshop on Evolvable Hardware*. IEEE Computer Society, 2000.

[21] J. Miller. On the filtering properties of evolved gate arrays. In A. Stoica, J. Lohn, and D. Keymeulen, editors, *The First NASA/DoD Workshop on Evolvable Hardware*, pages 2–11, Pasadena, California, 1999. Jet Propulsion Laboratory, California Institute of Technology, IEEE Computer Society.

[22] J. Miller, A. Thompson, P. Thomson, and T. Fogarty, editors. *Proc. 3rd Int. Conf. on Evolvable Systems (ICES2000): From Biology to Hardware*, volume 1801 of *LNCS*. Springer-Verlag, 2000.

[23] J. F. Miller, D. Job, and Vesselin K. Vassilev. Principles in the evolutionary design of digital circuits - part I. *Genetic Programming and Evolvable Machines*, 1(3), 2000.

[24] A. Morozov, V. Saposhnikov, Vl. Saposhnikov, and M. Gossel. New self-checking circuits by use of berger-codes. *Proceedings of 6th IEEE International On-Line Testing Workshop*, pages 141–146, July 2000.

[25] Phil Nigh. SIA roadmap: Test must not limit future technologies. In *Proc. of the International Test Conference 1998*. IEEE Computer Society Press, 1998.

[26] M. Pflanz, K. Walther, C. Galke, and H. Vierhaus. On-line error detection and correction in storage elements with cross-parity check. In *Proc. of* 8^{th} *IEEE International On-Line Testing Workshop*, pages 69–73. IEEE Computer Society, July 2002.

[27] S. Piestrak. Limitations of design methods of self-checking synchronous sequential machines. *Proceedings of FTCS-29, The 29th International Symposium on Fault-Tolerant Computing*, June 1999.

[28] N. Shnidman, W. Mangione-Smith, and M. Potkonjak. On-line fault detection for bus-based field programmable gate arrays. *IEEE Transactions on VLSI systems*, 6(4):656–666, 1998.

[29] M. Sipper, D. Mange, and A. Pérez-Uribe, editors. *Proc. 2nd Int. Conf. on Evolvable Systems (ICES98)*, volume 1478 of *LNCS*. Springer-Verlag, 1998.

[30] J. Smith and G. Metze. Strongly fault secure logic networks. *IEEE Trans. Comp.*, C-27(6):491–499, June 1978.

[31] A. Steininger. Testing and built-in self-test - a survey. *Journal of Systems Architecture*, 46:721–747, 200.

[32] A. Stoica, D. Keymeulen, and J. Lohn, editors. *Proc. 1st NASA/DoD workshop on Evolvable Hardware*. IEEE Computer Society, 1999.

[33] R. Tanese. Distributed genetic algorithms. In J.D. Schaffer, editor, *Proc. of the Third International Conference of Genetic Algorithms*, pages 434–439. Morgan Kauffmann, 1989.

[34] A. Thompson, I. Harvey, and P. Husbands. Unconstrained evolution and hard consequences. In E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware: The evolutionary engineering approach*, volume 1062 of *LNCS*, pages 136–165. Springer-Verlag, 1996.

[35] A. Thompson and P. Layzell. Analysis of unconventional evolved electronics. *Communications of the ACM*, 42(4):71–79, April 1999.

[36] V. Vassilev, D. Job, and J. Miller. Towards the automatic design of more efficient digital circuits. In J. Lohn, A. Stoica, and D. Keymeulen, editors, *The Second NASA/DoD workshop on Evolvable Hardware*, pages 151–160, Palo Alto, California, 13-15 2000. IEEE Computer Society.

[37] H. Wunderlich. BIST for systems-on-a-chip. *INTEGRATION, the VLSI journal*, 26:55–78, 1998.