# Evolving High-Speed, Energy-Efficient Integrated Circuits

Frank Sill and Ralf Salomon
Faculty of Computer Science and Electrical Engineering
University of Rostock, 18051 Rostock, Germany
Email: {frank.sill,ralf.salomon}@uni-rostock.de

***Abstract*— State-of-the-art technologies in very large scale integration (VLSI) aim at the realization of fast but low-power consuming circuits. Recent technological advents offer a design parameter with which both the processing speed and power consumption of every single gate, which is *the* basic building block of any circuit, can be fine tuned. With respect to this design parameter, a VLSI design constitutes a multi-dimensional multi-modal optimization problem, for which previous research has already developed some problem-specific optimization procedures. But since they yield results worse than engineers, this paper investigates how genetic algorithms perform in this application domain. It turns out that in comparison to the procedures mentioned above, genetic algorithms are able to reduce the power consumption by about 10-40 %. These achievments are practically relevant, since they extend the circuits' times-of-operation by the same amount. In addition, this paper also considers some problem-specific variations, which significantly speed up the optimization process.**

## I. Introduction

Off-the-shelf products offered virtually everywhere indicate that the processing speed of digital devices, such as desktop computers, laptops, personal digital assistents, cellular phones, and the like, is of high importance to many end-users. In other words, end-users expect their devices to operate at a processing speed as *high* as possible. With respect to *mobile devices*, the markets today also suggest another trend: mobile devices are expected to yield times-of-operation as long as possible, probably in order to maximize the end-user's independence on electrical wires.

The issue of a suitable power supply, e.g., by means of rechargeable batteries, becomes even more important in *small* mobile devices, such as cellular phones and personal digital assistants. For example, most end-users would probably not accept, if the battery was larger and/or heavier than the actual cellular phone. High processing speed and long time-of-operation are probably *the* driving forces for research on low-power technologies. Consequently, current research on low-power VLSI technologies [2], [7], [8], [14], [15] tries, among other aspects, to minimize a circuit's energy consumption without tampering its processing speed.

As Section II briefly reviews, an integrated circuit consists of very many interconnected gates. A particular design parameter, called the gate threshold-voltage $V_{TH}$, determines both a gate's energy consumption and its processing speed. Due to technological reasons, these two parameters are inversely correlated; they compete with each other by their very nature. Fortunately, not all gates have the same importance with respect to the circuit's overall processing speed. Thus, some gates must work at the highest processing speed possible, consuming high amounts of energy, wheras others can be slowed down, which conserves valuable energy resources. From an optimization point of view, the *goal* is to obtain the fastest processing speed by paying minimal energy consumption. Theoretically, this combinatorial optimization problem is *NP*-hard for the general case.

Section II also briefly reviews previous research [14], [15], [18], [19], which has developed some special-purpose algorithms for the problem at hand. Even though these algorithms yield quite encouraging results, a comparison with human-optimized designs indicates that these solutions are only sub-optimal. Apparently, the algorithms got stuck at sub-optimal solutions, also known as diverting local optima in the pertinent literature on optimization.

Since the optimization procedures mentioned above do not reliably yield optimal solutions, this paper applies evolutionary algorithms to the problem at hand. Evolutionary algorithms are heuristic population-based search procedures that incorporate random variation and selection. This paper focuses on the application of a particular instance, called genetic algorithms, since both numerous experiments and theoretical analyses [1], [5], [13] stress their superior global optimization performance when applied to rather combinatorial optimization tastks, such as the optimization problem at hand, especially in the presence of unwanted local optima. Therefore, Section III presents a short description of this class of algorithms. Standard genetic algorithms do not exploit any problem-specific properties. Thus, Section III also proposes a few problem-specific enhancements, in order to accelerate the optimization process.

In order to allow for an evaluation, this paper applies selected evolutionary algorithms to some rather standard design problems, which are drawn from the ISCAS benchmark suite [6]. Section IV provides a short description of these tasks, and also summarizes all the relevant parameter settings. The results presented in Section V suggest that the selected algorithms evolve designs better than those previously reported. The results furthermore indicate that the modifications proposed in Section III lead to a significant speed up. Finally, Section VI concludes with a brief discussion.

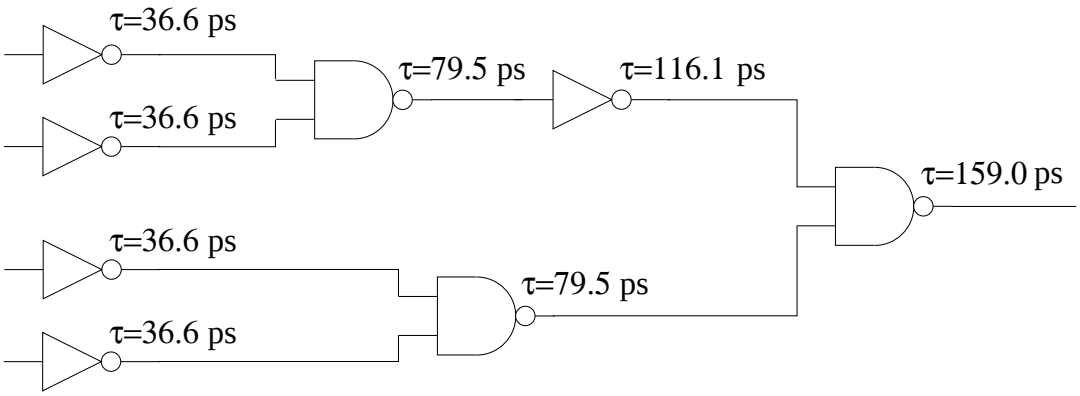


Fig. 1. A simple CMOS-circuit with different path delays, caused by different numbers of gates in each path. The lower path is non-critical, and thus may be subject to the implementation in slow high-voltage technology. Here, $\tau$ refers to the cumulated delays from the circuit's input to the gate's output, rather than the gate's individual delays.

## II. Background and Previous Research

A digital VLSI circuit generally consists of very many gates of different types, such as NAND, NOR, inverters, etc., and varying numbers of inputs, which together realize the circuit's functionality, e.g., a network adapter, a microprocessor, or something else. Each gate requires some time to process its input data. This time is generally called the gate's delay $\tau$. Figure 1 presents a simple example, which allows for the following two observations: First, the delay $\tau$ is not equivalent for all gates but depends on both the gate's functionality[1] and the number of inputs. Second, not all gates are equally important for the circuit's overall delay. In this example, the upper path has more gates in sequence and thus constitutes the circuit's critical path, since it determines its overall delay, whereas the lower path processes its signals faster anyhow. In other words, the gates residing in the non-critical path could be processing their signals at a lower speed without affecting the circuit's overall delay to some extent and thus could be saving valuable energy, if that was technologically possible.

It might be well known to many readers that every gate is realized by a certain number of transistors. A transistor's electrical characteristics depend, among other things, on a specific design parameter, called the gate threshold-voltage $V_{TH}$. Given a specific technology, this parameter determines both the transistor's processing delay[2] as well as its power consumption. Conversaly, if either the transistor's delay or its power consuption is given, the other parameter is also determined. Unfortunately, these two parameters are inversely correlated by their very nature. That is, a transistor has either a short delay and a high power consumption or vice versa.

It used to be that the very same threshold voltage $V_{TH}$ had to be used for *all* transistors throughout the entire VLSI circuit. State-of-the-art design technologies [2], [7], [8], [14], [15], [18], [19], however, allow for using varying threshold voltages for the transistors. The designer can thus fine tune both the delay and the power consumption of every gate. Table I provides three examples of three different realizations with their resulting delays and leakage currents, which are the main contribution to the gate's power consumption. For further technological details, the interested reader is referred to [14], [15].

TABLE I

This table shows three different realizations with their resulting delays and leakage currents for NOR-2, NAND-2, and an inverter, with the number denoting the number of gate inputs.

| NOR-2 | | NAND-2 | | INV | |
|---|---|---|---|---|---|
| 66.3 ps | 86.0 nA | 42.9 ps | 135.0 nA | 36.6 ps | 92.8 nA |
| 78.0 ps | 36.6 nA | 51.3 ps | 46.5 nA | 37.6 ps | 62.5 nA |
| 90.0 ps | 10.6 nA | 58.3 ps | 20.3 nA | 45.8 ps | 12.6 nA |

By having the option of choosing from different gates, the designer may select fast gates (consuming high energy) for the critical path and slower gates (consuming low energy) for the non-critical paths. By offering $p$ specific implementations per gate, i.e., a specific combination of delay and energy consumption, The design task consists of selecting a particular implementation for every gate such that the circuit's overall delay is as short as possible and that simultaneously the circuit's overall power consumption assumes a minimum. Once particular gate implementations have been selected, state-of-the-art design tools [] ??? frank sill fragen ??? automatically determine both the circuit's entire delay and its entire power consumption.

With a total of $g$ gates, the very same circuit can be realized in potentially $n{=}g^p$ alternatives. Previous research [9] has suggested that $p{=}3$ different implementations per gate are optimal[3]. Due to the possible interconnections between different paths, this optimization problem *can* be *NP*-hard in the general case.

[1]Different functionalities requiere a different number of internal elements, i.e., transistors, which influences a gate's overall delay $\tau$.

[2]On the transistor as well as gate level, the term *delay* rather than processing speed is more commonly used.

[3]Unfortunately, the literature [9] does not provide any substantial indication, why $p{=}3$ is supposed to be optimal.

For the task of finding optimal designs, previous research [8], [14], [15], [18] has employed various algorithms, from which two serve as a baseline for comparison purposes. The first algorithm [14], denoted as SFA-I (straight-forward algorithm, variant I) for short, starts off by using the slowest implementation for all gates. It then accelerates the critical path by substituting some of them with their fastest counterparts until either this path has turned into a non-critical one or no further gates can be accelerated. This step is repeated as long as it can change a critical path into a non-critical one. Finally, all fast gates are substituted by the medium ones as long as this does not affect the circuit's overall delay.

The second algorithm [15], denoted as SFA-II for short, is a modification of a previous development [10], [17]. It starts off by selecting the fastest alternatives for all gates. It then consecutively substitutes them with medium or slow alternatives. In so doing, it prefers gates with a high fan-out. The step is repeated until no further gate can be slowed down without affecting the circuit's overall delay. For further details, the interested reader is referred to the literature [14], [15].

## III. The Evolutionary Approach

The term *evolutionary algorithms* refers to a class of heuristic population-based search procedures that incorporate random variation and selection, and provides a framework that mainly consists of genetic algorithms [5], evolutionary programming [3], [4], and evolution strategies [11], [13].

Even though all evolutionary algorithm have their own peculiarities, they share many common features. All evolutionary algorithms maintain a population of $\mu$ individuals, also called parents. In each generation, an evolutionary algorithm generates $\lambda$ offspring by copying randomly selected parents and applying variation operators, such as mutation and recombination. It then assigns a fitness value (defined by a fitness or objective function) to each offspring. Depending on their fitness, each offspring is given a specific survival probability.

By selecting certain individuals as parents, an evolutionary algorithm advances from one generations to another. The two most-commonly used selection schemes are denoted as either ($\mu$,$\lambda$) or ($\mu$+$\lambda$). The first selection scheme, i.e., ($\mu$,$\lambda$) indicates that the algorithm chooses the parents for the next generation *only* from the offspring, whereas the latter selection scheme selects from the union of the previous parents *and* the current offspring; the latter form is also known as $\mu$-fold elitism.

As has been discussed above and exemplified in Table I, all gates can be configured with three different leakage currents[4]. Therefore, the optimization problem at hand is *discrete* by its very nature for which the traditional form of genetic algorithms is particularly suited. In the experimental comparisons, these algorithms are denoted as ($\mu$+$\lambda$)-GA or ($\mu$,$\lambda$)-GA for short. The other evolutionary algorithm variants, particularly evolutionary programming and evolution strategies, are *rather* tailored to continuous parameter optimization and are thus not further considered in this paper.

In order to be optimizable for a genetic algorithm, this paper adopts a direct coding in which every gate is represented by a particular gene, which codes for the particularly chosen gate threshold-voltage $V_{TH}$. Thus, a device that consists of $n$ gates is represented by a genome of $n$ positions with each being able to assume three different values (see above and also [9]).

The genetic algorithms described above are rather generic and do not account for any problem-specific property. In the problem at hand, for example, it does not make sence to mutate any gate of the critical path(s); they have to be as fast as possible. Therefore, the first variation, denoted as ($\mu^+_,\lambda$)-NC-GA, considers only gates from non-critical path(s). The chromosom length has to be reduced appropriately.

The second variation accounts for the following observation: gates with many inputs and outputs are likely to be in many different paths. Thus, several paths would benefit from a fast implementation of such gates. Therefore, the mutation operator might be *biased* such that it chooses faster gates more often for gates with higher-than-average connectivity, and slower ones for gates with lower-than-average connectivity. To this end, the algorithm calculates a correction value

$$\epsilon_i = \begin{cases} 0.15 & : \quad \frac{c_i}{\max_i c_i} > 0.8 \\ -0.15 & : \quad \frac{c_i}{\max_i c_i} < 0.2 \\ 0 & : \quad \text{else} \end{cases} . \qquad (1)$$

The probability of choosing a fast, normal (medium), or slow implementation for gate $i$ is then $p_f = 1/3 + \epsilon_i$, $p_n = 1/3$, and $p_s = 1/3 - \epsilon_i$, respectively. ??? frank, sind hier formeln richtig ... ??? Genetic algorithms that employ this form of biased mutations are denoted as ($\mu^+_,\lambda$)-BM-GA for short.

## IV. Methods

All experiments were done by means of a state-of-the-art design tool []. ??? frank sill fragen ??? To this end, the circuit's gate configuration is defined by a *net-list*, which specifies all connections between the inputs and outputs of all gates. In addition, the design tools reads a list, which specifies the delay of all gates. Then, the design tool calculates both the circuit's delay and its power consumption. In order to ease the implementation, this paper adopts a direct encoding in which each allel directly codes the gate's particular implementation (see also Section II). In order to speed up the optimization process, the fastest implementations are chosen for all gates. Since each gate can choose its leakage current only from three different values, the implementation of an appropriate mutation operator is straight forward: it chooses the next lower or higher value. In accordance with the literature [5], [12] a mutation probability $p_m = 1/n$ with $n$ denoting the number of gates was chosen in all experiments.

Because the chromosom is not given in any particular gate ordering, this paper employs uniform recombination, which exchanges corresponding genes of two randomly selected parents with a probability $p_r$=0.5. Even though the literature

[4]It should be noted that the actual values of the leakage currents are not equivalent for all gates, but depend on their number of inputs, functionality, and other parameters.

[11] suggests that $\mu$=1 parent and $\lambda$=6 offspring yield the highest sequential efficiency, this paper also considers larger population sizes for comparitive purposes.

As has been outlined above, the fitness function should incorporate both the network's delay and its energy consumption. Since the network's delay is of primary interest (by definition), the following fitness function has been used:

$$f = \texttt{delay} - \frac{1}{\texttt{leakage}} \ . \tag{2}$$

Since the circuit's `delay` and `leakage` are of type integer and strictly greater than zero, `delay` dominates the fitness $f$.

In order to perform a comparative study, this paper has selected the following five standard designs from the ISCAS benchmark suite (for further details, see [6]):

**C432** is a 27-channel interrupt controller [20] with a total of 36 inputs and 7 outputs. The controller has 27 interrupt request inputs, which are grouped into 3 buses with 9 lines each. It has further 9 control inputs, which activate/de-activate the associated interrupt lines. The implementation of such an interrupt controller, requires 160 gates.

**C1355** is a 32-Bit single-error correcting circuit [21]. By utilizing a (40,32) Hamming code matrix, it generates a 8-bit long syndrom by reading the 32 input lines. The 41 input lines are forwarded along with the 8-bit syndrom to a correction unit. The implementation of this device requires 546 gates.

**C3540** is a 8-bit arithmetic-logical-unit (ALU) [22] with 50 inputs and 22 outputs. It realizes various arithmetic, logical, BCD, shift, and other operations on 8 input lines, and its implementation requires 1669 gates.

**C5315** is an extension of the `C3540`-circuit [23], in that it realizes a 9-bit ALU with 178 inputs and 123 outputs, which requires 2406 gates for its implementation.

**C7552** is a device [24] that contains a 34-bit adder, a 34-bit comparator, which requires an additional 34-bit adder, and an 34-bit parity checker. The circuit requires 3512 gates to map the 207 inputs onto 108 outputs.

For the realization, this paper used a previously developed gate library [16], which is based on the 65 nm Berkeley predictive technology models (BPTM). ??? frank: eine reference ???

## V. Results

Direct performance comparisons are not straight forward for the following two reasons: first, two quality measures are simultaneously subject to the optimization process, and second, the optimization procedures considered in this paper operate on different time scales. Therefore, this section starts off with a detailed discussion of Figures 2-5, which show various performance figures obtained on the ALU-design problem C5315.

Figure 2 shows the evolution of both the delay and leakage when using both a (1+6)-GA and a (1,6)-GA. Since the genetic

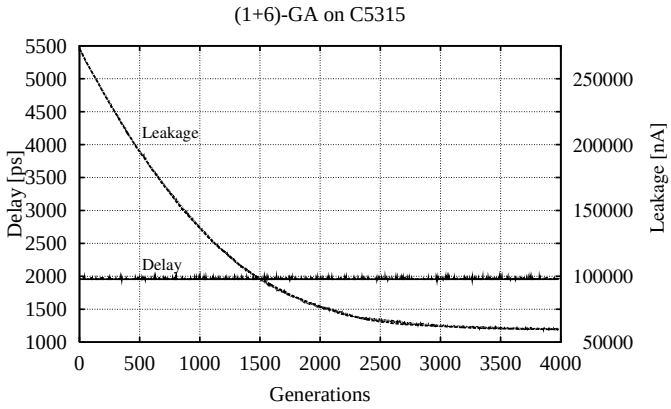


Fig. 2. The evolution of both the circuit's delay ($y$-axis on the left-hand-side) and leakage ($y$-axis on the right-hand-side) when using $(1\overset{+}{,}6)$-genetic algorithms for the C5315 problem

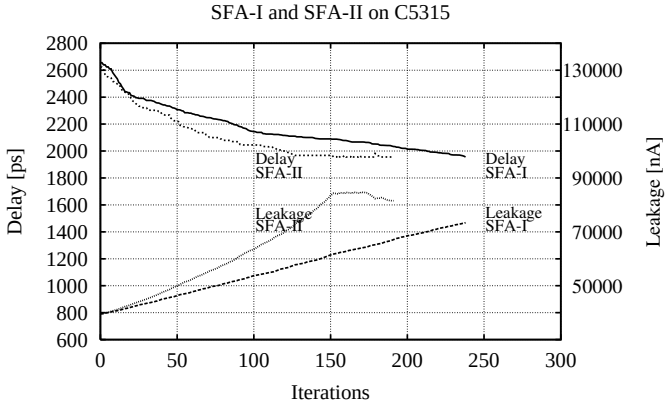


Fig. 3. The evolution of both the circuit's delay ($y$-axis on the left-hand-side) and leakage ($y$-axis on the right-hand-side) when applying a straight-forward optimization algorithm on the C5315 problem

algorithms initialize all gates with the fastest realizations, the delay starts at 1955 ps and a (total) leakage of 272,600 nA. During the corse of evolution, then, the leakage drops to almost a fifth of that value, i.e., about 58,597 nA, without increasing the circuit's delay as requested. The small jitter of the jitter is due to some imprecissions of some floating-point operations. Since the performance graphs of both procedures are virtually identical, the remainder of this section focuses on the "plus" selection scheme and does not consider the other one.

For comparison purposes, Figure 3 presents the development of both delay and leakage when using the procedures SFA-I and SFA-II previously developed [14], [15]. It can be seen that both procedures start off with a relatively large delay of about 2656 ps, but arrive at the same final value of 1955 ps after about 150 to 250 iterations. In order to attain this improved processing, both procedures have increased the leakage to about 73,443 nA and 81,580 nA, respectively.

For a better comparison of the two parameters under optimization, Figures 4 and 5 combine those graphs into two figures. To this end, the time scales, i.e., $x$-axes, have been

rescaled such that the time is normalized to a 100 time units. It can be clearly seen that the circuit's final delay arrive at the same values (Figure 4), whereas the genetic algorithms were able to improve the leakage by about 20-30% (Figure 5). This, however, came at the cost of a significant increase in the computational requirements. With respect to the end-users expectations on the time-of-operation, this additional optimization effort might by worth it, especially since this has to be done only once during the circuit's design process.

Figures 6 to 9 show how the optimization procedures under consideration evolve the leakage over time for the other four design problems C432, C1355, C3540, and C7552, respectively. As for the C5315 problem discussed first, all procedures exhibit a similar behavior. Furthermore, all procedures arrive at the same final delay (not shown in any figure) for each problem.

The performance graphs may be summarized as follows: In comparison to SFA-I, SFA-II constitutes a significant improvement in that it requires shorter optimization time and often yields lower leakage values. SFA-II increases the leakage by substituting slow high-voltage gates by their fastest counter parts, until the circuit has the shortest delay possible. It then reduces the resulting leakage by also considering medium-voltage gates.

The genetic algorithms by contrast, yielded the lowest overall leakage and thus energy consumption values, but required substantially more time. This observation goes inline with the pertinent literature [11]: evolutionary algorithms are a general framework, which might be slower than special-purpose procedures in many cases but have the ability to escape from local optima, and are thus often able to yield superior results. For comparison purposes, Table II presents the final values for delay and leakage for all algorithms over all problems considered in this paper.

In order to assess the utility of large population sizes, Figures 10 and 11 illustrate the application of a (10+40)-GA to the C432 and C5315 problems, respectively. A comparison

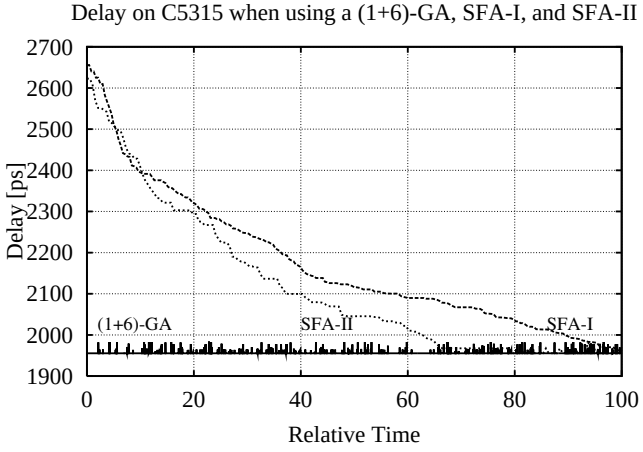


Fig. 4. The evolution of the delay for all algorithms

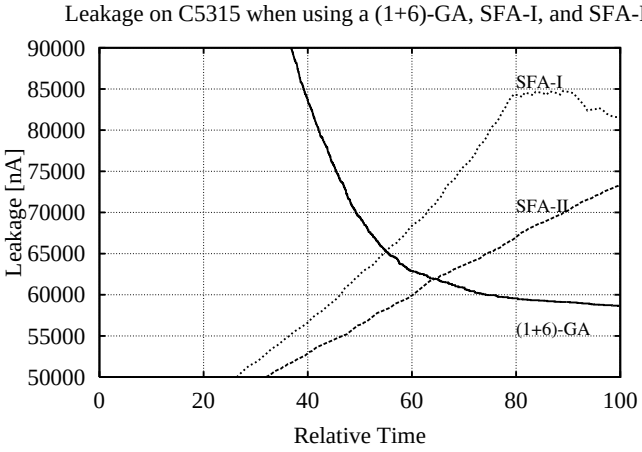


Fig. 5. The evolution of the leakage for all algorithms

with Figures 6 and 9, respectively, indicates that a (10+40)-GA might be faster in terms of the number of generations but significantly slower in terms of the number of functions evaluations, which is the product of the number of generations and the number of offspring $\lambda$.

## VI. Conclusions

This paper has argued that processing speed and energy consumptions are properties, which end-users consider important for mobile devices. It has been discussed that these two parameters depend on each other due to technological reasons. This paper has furthermore reviewed two optimization procedures, which have been investigated in previous research. Since previous research has led to optimized designs, which are inferior to devices designed by humans, this paper has applied genetic algorithms to this optimization problem. The experimental results indicate that genetic algorithms were able to reduce the leakage by about 10-40% as compared to previously optimized designs. The results also indicate,

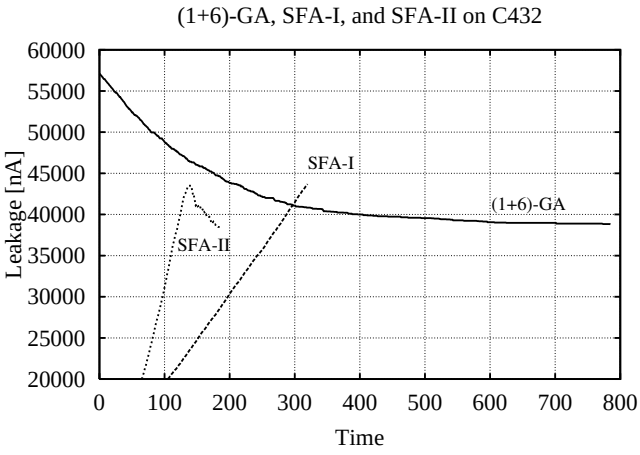


Fig. 6. The evolution of the leakage when applying a (1+6)-GA, SFA-I, and SFA-II to the C432 problem

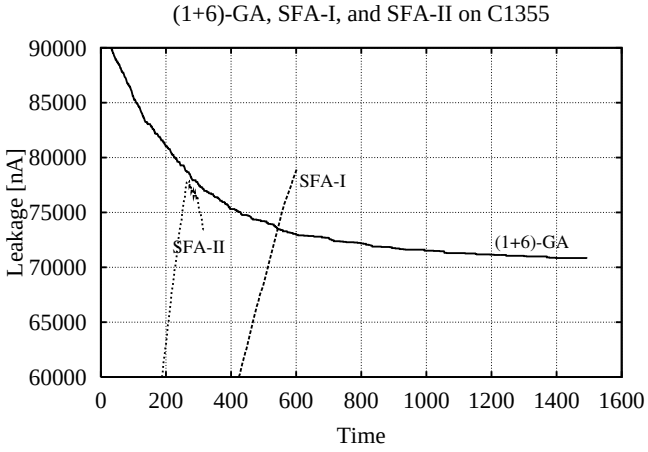


Fig. 7. The evolution of the leakage when applying a (1+6)-GA, SFA-I, and SFA-II to the C1355 problem

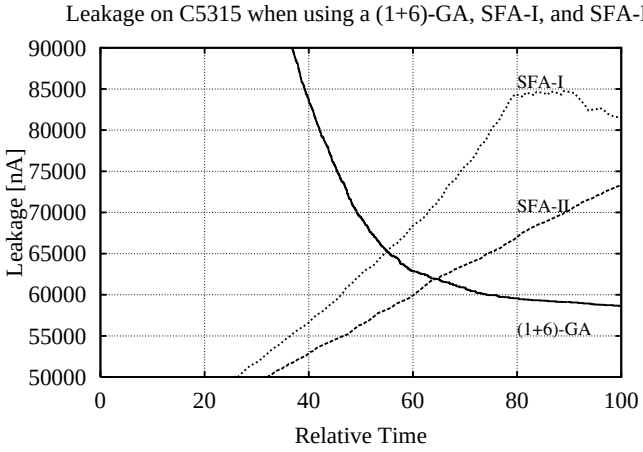


Fig. 9. The evolution of the leakage when applying a (1+6)-GA, SFA-I, and SFA-II to the C5315 problem

however, that genetic algorithms require substantially more computation time.

Future research will be dedicated to the investigate of further optimization approaches, such as simulated annealing and other evolutionary algorithm variants, Furthermore, future research will be investigating to what extent an increase of the number $p$ of different gate implementations can benificial for the overall energy consumption.

## ACKNOWLEDGEMENTS

The authors gratefully thank Ralf Joost for fruitful discussions and many valuable comments on draft versions of this paper. The authors also thank Prof. Timmermann for his continuous support. Part of this research was supported by the German Research Foundation (DFG), grant number 466.

## REFERENCES

[1] T. Bäck, U. Hammel, and H.-P. Schwefel, Evolutionary Computation: Comments on the History and Current State. *IEEE Transactions on Evolutionary Computation*, **1**(1):3-17, 1997.

[2] W. Chen, W. Hang, P. Kudva, G.D. Gristede, S. Kosonocky, and R.V. Joshi. Mixed Multi-Threshold Differential Cascode Voltage Switch (MT-DCVS) Circuit Styles and Strategies for Low Power VLSI Design, in E. Macii, V. De, and M.J. Irwin (Eds.) *Proceedings of the 2001 International Symposium on Low Power Electronics and Design (ISLPED'01)*, pp. 263-266, 2001.

[3] Fogel, L.J., 1962. Autonomous Automata. *Industrial Research*, **4**:14-19.

[4] D.B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Learning Intelligence.*. IEEE Press, NJ, 1995.

[5] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[6] M. Hansen, H. Yalcin, and J. P. Hayes. Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering, in *IEEE Design and Test*, **16**(16 ):72-80, 1999.

[7] J.K. Kao and A. Chandrakasan. Dual-Threshold Voltage Techniques for Low-Power Digital Circuits, *IEEE Journal of Solid State Circuits*, **35**(7):1009-1018, 2000.

[8] T. Karnik, Y. Ye, J. Tschanz, L. Wei, S. Burns, V. Govindarajulu, V. De, and S. Borkar, Total Power Optimization by Simultaneous Dual-Vt

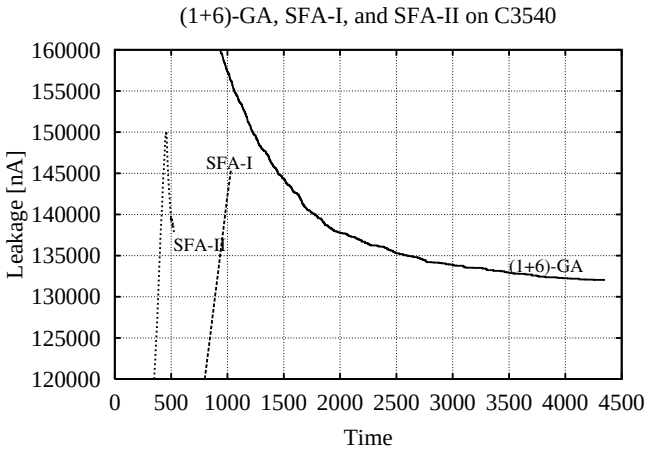


Fig. 8. The evolution of the leakage when applying a (1+6)-GA, SFA-I, and SFA-II to the C3540 problem

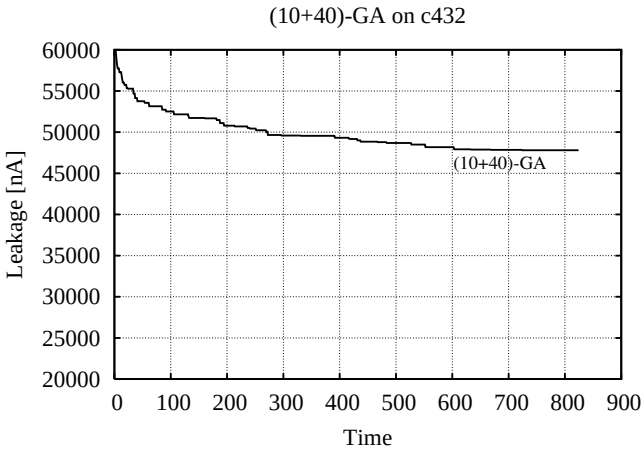


Fig. 10. The evolution of the leakage when applying a (10+40)-GA to the C432 problem

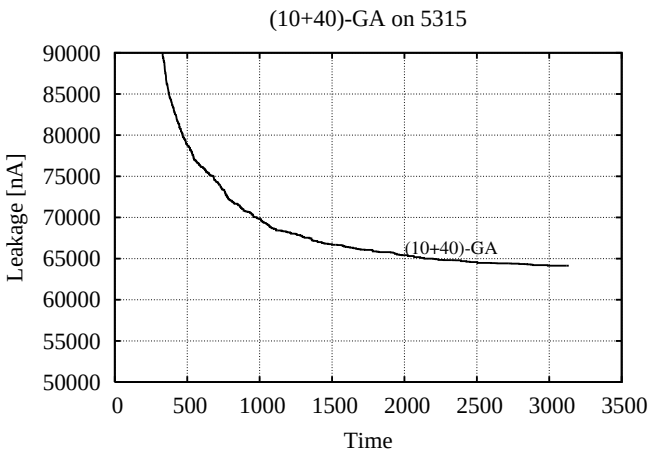


Fig. 11. The evolution of the leakage when applying a (10+40)-GA to the C5315 problem

TABLE II

COMPARISON OF THE FINAL DELAY AND LEAKAGE VALUES OVER ALL PROCEDURES AND ALL PROBLEMS

| **C432** | Delay | Leakage |
|---|---|---|
| SFA-I | 1045 | 43,699 |
| SFA-II | 1045 | 38,482 |
| (1+6)-GA | 1045 | 38,779 |

| **C1335** | Delay | Leakage |
|---|---|---|
| SFA-I | 925 | 79,037 |
| SFA-II | 925 | 73,443 |
| (1+6)-GA | 925 | 70,781 |

| **C3540** | Delay | Leakage |
|---|---|---|
| SFA-I | 1618 | 145,504 |
| SFA-II | 1618 | 137,857 |
| (1+6)-GA | 1618 | 132,012 |

| **C5315** | Delay | Leakage |
|---|---|---|
| SFA-I | 1955 | 73,443 |
| SFA-II | 1955 | 81,580 |
| (1+6)-GA | 1955 | 58,597 |

| **C7552** | Delay | Leakage |
|---|---|---|
| SFA-I | 1198 | 180,714 |
| SFA-II | 1198 | 195,898 |
| (1+6)-GA | 1198 | 169,948 |

Allocation and Device Sizing in High Performance Microprocessors, in *Proceedings of the 39th Conference on Design Automation*, pp. 486-491, 2002.

[9] T. Kuroda. Low-Power, High-Speed CMOS VLSI Design. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computer and Processors (ICCD 2002)*, pp. 310-315, 2002.

[10] M. Liu, W.S. Wang, and M. Orshansky. Leakage Power Reduction by Dual-Vth Designs under Probabilistic Analysis of $V_{th}$ Variation, in R.V. Joshi, K. Choi, V. Tiwari, and K. Roy (Eds.), *Proceedings of the 2004 International Symposium on Low Power Electronics and Design (ISLPED 2004)*, pp. 2-7, 2004.

[11] I. Rechenberg, *Evolutionsstrategie* (Frommann-Holzboog, Stuttgart, 1994).

[12] R. Salomon. Reevaluating Genetic Algorithm Performance under Coordinate Rotation of Benchmark Functions; A survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, **39**(3):263-278, 1996.

[13] H.-P. Schwefel. *Evolution and Optimum Seeking.* John Wiley and Sons, NY. 1995.

[14] F. Sill, F. Grassert, and D. Timmermann. Low Power Gate-level Design with Mixed-Vth (MVT) Techniques, in *Proceedings of the 17th Symposium on Integrated Circuits and Systems (SBCCI)*, 2004.

[15] F. Sill, F. Grassert, and D. Timmermann. Reducing Leakage with Mixed-Vth (MVT), in *Proceedings of 18th Conference on VLSI Design*, pp. 874-877, 2005.

[16] F. Sill, F. Grassert, and D. Timmermann. Total Leakage Power Optimization with Improved Mixed Gates, in *Proceedings of the 18th Symposium on Integrated Circuits and Systems Design (SBCCI 2005)*, 2005.

[17] A. Srivastava, D. Sylvester, and D. Blaauw. Statistical Optimization of Leakage Power Considering Process Variations using Dual-$V_{th}$ and Sizing, in S. Malik, L. Fix, and A.B. Kahng (Eds.), *Proceedings of the 41st Design Automation Conference (DAC 2004)*, pp. 773-778, 2004.

[18] V. Sundararajan and K. Parhi. Low Power Synthesis of Dual Threshold Voltage CMOS VLSI Circuits, in *Proceedings of the IEEE International Symposium on Low Power Electronics and Design*, pp. 139-144, 1999.

[19] L. Wei, K. Roy, and C. Koh. Power Minimization by Simultaneous Dual-Vth Assignment and Gate-sizing, in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 413-416, 2000.

[20] http://www.eecs.umich.edu/∼jhayes/iscas/c432.html

[21] http://www.eecs.umich.edu/∼jhayes/iscas/c499.html

[22] http://www.eecs.umich.edu/∼jhayes/iscas/c3540/c3540.html

[23] http://www.eecs.umich.edu/∼jhayes/iscas/c5315/c5315.html

[24] http://www.eecs.umich.edu/∼jhayes/iscas/c7552/c7552.html